IMT School for Advanced Studies, Lucca
Lucca, Italy

Efficient and Accurate Analysis of Two Classes of Transparent Generative Models

PhD Program in Systems Science

Track in Computer Science and Systems Engineering

XXXV Cycle

By

Francesca Randone

2024

The dissertation of Francesca Randone is approved.

PhD Program Coordinator: Alberto Bemporad, IMT School for Advanced Studies Lucca

Advisor: Prof. Mirco Tribastone, IMT School for Advanced Studies Lucca

Co-Advisor: Prof. Luca Bortolussi, Università degli Studi di Trieste

The dissertation of Francesca Randone has been reviewed by:

Prof. Ezio Bartocci, Vienna University of Technology

Prof. Laura Carnevali, University of Florence

IMT School for Advanced Studies Lucca
2024

Contents

List of Figures

List of Tables

Acknowledgements

Any person asked about his or her PhD will probably give an answer involving the word “challenging”. For me, it is the same, but overcoming this challenge was possible also thanks to the people who accompanied me through this journey.

First, I must thank my advisor, *Mirco Tribastone*, a fantastic mentor and irreplaceable support through doubts and (the many) rejections. Thanks especially for his enthusiasm and for teaching me how to have fun doing this job, which can be quite demanding but incredibly satisfying.

My co-advisor, *Luca Bortolussi* also deserves my deepest gratitude for first showing me “the dark side of mathematics” and eventually convincing me that computer science could be at least as enjoyable. If today I am at least partially a computer scientist, it is thanks to him.

I must also thank other professors and researchers I met along my journey. *Rocco De Nicola*, for first introducing me to formal methods and logic in his courses, *Joost-Pieter Katoen* for letting me visit his amazing research group at RWTH Aachen in what has been an incredibly formative experience and *Emilio Incerto* for his patience in working with the code written by a mathematician.

Thanks to *Giuseppe* and *Serenella*, with whom I shared the ups and downs of the PhD, both inside and outside IMT’s walls.

Finally, I need to thank the two people who made me the one I am, without which all this could not have been possible. Thanks to my parents for understanding and being present for me no matter what; each of my achievements is theirs as well.

To conclude, I would like to quote a young and promising Italian philosopher and computer scientist with a sentence that expresses how I genuinely feel about computer science and life in general: “L’uomo non è fatto per programmare” (“Man was not made for programming”).

Vita

November 9, 1995	Born, Siracusa, Italy
2014 - 2017	Bachelor Degree in Mathematics Final mark: 110/110 cum laude Università degli Studi di Catania, Italy
2017 - 2019	Master Degree in Mathematics Final mark: 110/110 cum laude Università degli Studi di Trieste, Italy
2019-2023	PhD Scholarship IMT School for Advanced Studies, Lucca
Apr 2023 - June 2023	Visiting PhD Student RWTH Aachen, Germany

Publications

1. F. Randone, L. Bortolussi, M. Tribastone, "Refining mean-field approximations by dynamic state truncation." *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(2), 1-30, 2021.
2. F. Randone, L. Bortolussi, M. Tribastone, "Jump Longer to Jump Less: Improving Dynamic Boundary Projection with h-Scaling." *Quantitative Evaluation of Systems: 19th International Conference, QEST*, Warsaw, Poland, September 12–16, 2022, pp. 150-170, Cham: Springer International Publishing, 2022.
3. F. Randone, L. Bortolussi, E. Incerto, M. Tribastone, "Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures." *Proceedings of the ACM on Programming Languages*, 8(POPL), 1882-1912.

Presentations

1. F. Randone, “Refining mean-field approximations by dynamic state truncation.” SIGMETRICS conference, online, 2021.
2. F. Randone, “Jump Longer to Jump Less: Improving Dynamic Boundary Projection with h-Scaling.”, QEST conference, Warsaw, Poland, 2022.
3. F. Randone, “Refining Deterministic Approximations Of Stochastic Reaction Networks Through Dynamic Boundary Projection”, Chemical Reaction Networks Workshop, Politecnico di Torino, 2022.
4. F. Randone, “Dynamic Boundary Projection: Refining Deterministic Approximations Of Stochastic Reaction Networks”, Mathematics of Reaction Networks, online, 2023.
5. F. Randone, “Refining Deterministic Approximations Of Stochastic Reaction Networks Through Dynamic Boundary Projection”, SIAM Conference on Dynamical Systems, Portland, 2023.
6. F. Randone, “Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures”, RWTH, Aachen, 2023.
7. F. Randone, “Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures”, SySMA Workshop, Lucca, 2023.
8. F. Randone, “Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures”, POPL conference, London, 2024.

Abstract

While widely used, generative models pose the challenging task of deriving and analyzing their underlying distribution. In this thesis, we focus on two classes of transparent generative models and present new methods to tackle this task.

Markov Population Processes use Continuous Time Markov Chains to describe the evolution of populations over time. Their analysis is often hindered by state-space explosion, tackled with deterministic approximation or truncation techniques. We propose a method, Dynamic Boundary Projection, that couples an exact stochastic description of a subset of states and a deterministic approximation that dynamically shifts the subset across the state space. The resulting finite set of ODEs is asymptotically exact. We show that our method performs well in terms of accuracy and runtimes on challenging systems. We also propose an extension that further reduces the number of equations while maintaining good accuracy.

Probabilistic Programs leverage the power of programming languages to define probabilistic models; however, no one-fit-for-all solution exists to derive the posterior distribution. We define a family of approximating semantics, Gaussian Semantics, that leverages moment-matching and the approximation power of Gaussian Mixtures to approximate the joint probability distribution over program variables. As the number of the moments matched increases, Gaussian Semantics tends to the exact semantics. We implement an instance of Gaussian Semantics that matches the first two order moments and show that our implementation performs competitively with respect to other state-of-the-art inference methods and excellently on two classes of models taken from the literature.

Introduction

Generative models are a class of statistical models that allow sampling from an underlying joint probability distribution (Ng and M. Jordan, 2001). They have attracted substantial attention within the scientific community due to their multifaceted applications and theoretical significance, which has led to applications in computer vision (Goodfellow et al., 2014), natural language processing (Kingma et al., 2014), style transfer (Gatys, Ecker, and Bethge, 2016) and data augmentation (Ohno, 2020).

However, analyzing generative models presents unique challenges, particularly in deriving the underlying distribution and extracting meaningful statistical properties. For example, to properly model a distribution, a generative model might be required to encode high-dimensional distributions, conditional dependencies between the variables, and time-dependence of the parameters. Over the years, specific techniques have been developed for each modeling paradigm to tackle these challenges, but how to best deal with each of them remains an open question (Goodfellow et al., 2014).

A major distinction in generative models is between black box and transparent models. Black box models, such as Generative Adversarial Networks (GANs) (Goodfellow et al., 2014), excel at capturing complex patterns in data and generating high-quality samples. However, they are generally not interpretable and make it difficult to directly encode in the model previous domain knowledge (Song et al., 2021; Dijk, Teräsvirta, and Franses, 2002).

On the other hand, transparent models, like Markov population processes and probabilistic programs, offer interpretability and provide clear insights into how the model arrives at its predictions. They allow a generative description of the model, exploiting previous knowledge of the phenomenon under analysis. This comes at the cost of limiting their ability to capture intricate patterns in highly complex datasets (Bishop and Nasrabadi, 2006).

Since the landscape of generative models is extremely vast and intricate, in this thesis, we restrict our attention to transparent models, and particularly to the two following classes.

- Markov population processes. In this case, the generative model is given as a Continuous Time Markov Chain, defining an underlying time-variable probability distribution over the state space (Kurtz, 1970).

- Probabilistic programs. In this case, an ad hoc programming language semantics is used to specify a joint probability distribution over program variables (Kozen, 1983).

In both cases, it is desirable not only to sample from the underlying distribution but also to extract some of its notable statistical quantities, such as the average dynamics for Markov population processes or the posterior mean for probabilistic programs. While an exact derivation of these quantities is generally unfeasible, accurate and efficient approximations are possible.

We briefly present the two cases in more detail.

Markov Population Processes

Markov population processes represent systems of N interacting agents (Benaim and Le Boudec, 2008a). They have been used to represent network protocols (Cecchi, Borst, and Leeuwaardena, 2015), information-spreading algorithms (Chaintreau, Le Boudec, and Ristanovic, 2009), peer-to-peer networks (Massoulié and Vojnovic, 2005), caching algorithms (Gast and Van Houdt, 2015), garbage collection (Van Houdt, 2013), and load balancing strategies (Mitzenmacher, 2001; Gast and Bruno, 2010; Tsitsiklis and Xu, 2011; Minnebo and Van Houdt, 2013; Xie et al., 2015).

In population processes, it is assumed that the interacting agents are divided into classes across which they can randomly transition. The system state is described by a state vector, whose components indicate how many agents of each class are present in the system at time t and changes according to discrete transitions fired at state-dependent exponential rates. The underlying Markov Chain has a state for each possible value of the state vector, therefore, its state space is usually very large ($\approx O(N^m)$ where m is the dimension of the state vector) or even infinite. Analyzing the exact dynamics of such systems would require the solution of a large system of ODEs, called the Master (or Kolmogorov) Equation, yielding one equation for each state, which in many cases is unfeasible (Van Kampen, 1992). Due to the state space's exponential explosion, simulation-based approaches quickly become computationally intensive (Gillespie, 2007).

Over the years, a number of approximation techniques have been proposed. In particular, a wide class of methods relies on truncation techniques (Munsky and Khammash, 2007; Kuntz et al., 2019; Gupta, Mikelson, and Khammash, 2017), in which the Master Equation is truncated to a finite number of states for which computations can be performed efficiently. Another class of methods is instead based on the seminal theorem by Kurtz (Kurtz, 1970), which guarantees that under suitable hypothesis, as N tends to infinity, the average dynamics of the normalized system tends to the solution of a system of m ODEs. This deterministic approximation does not give guarantees when N is finite and can generally deviate significantly from the true dynamics of the systems in the presence of noise, such as high variance and oscillations. To overcome these limitations, various refinements of Kurtz's approximation have been proposed (Gast, Bortolussi, and Tribastone, 2019; Gast and Van Houdt, 2015).

The first contribution of this thesis is to present a method called Dynamic Boundary Projection (DBP) (Randone, Bortolussi, and Tribastone, 2021) that couples the truncation approach and deterministic approximation à la Kurtz. In DBP, a truncated version of the Master Equation, describing the stochastic evolution of a hyper-rectangular subset of states, is shifted through the state space to follow the significant portion of the probability mass. This dynamical shifting is provided by coupling the truncated Master Equation with a deterministic function, which can be seen as the deterministic approximation of an auxiliary process that keeps track of the hyper-rectangle in which the original process evolves. For systems exhibiting oscillatory behaviors or high variance, DBP is more accurate than state-of-the-art deterministic approximations while keeping computational times lower than those needed to solve the full Master Equation or perform simulations.

Despite this gain in computational performances, the system of equations yielded by DBP is still subject to exponential explosion. To tackle this problem and improve the approach's scalability, the states' subset can be rescaled. Intuitively, this amounts to covering the same portion of the state space with fewer rescaled states. This procedure yields a scaled DBP method, further reducing computational times (Randone, Bortolussi, and Tribastone, 2022).

Derivation of DBP and its scaled version, together with their applications to some examples are the subject of Chapters 1 (background and notation), 2 (DBP) and 3 (scaled DBP) and refer to the following publications:

- F. Randone, L. Bortolussi, M. Tribastone, "Refining mean-field approximations by dynamic state truncation." *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(2), 1-30, 2021,

- F. Randone, L. Bortolussi, M. Tribastone, "Jump Longer to Jump Less: Improving Dynamic Boundary Projection with h-Scaling." *Quantitative Evaluation of Systems: 19th International Conference, QEST*, Warsaw, Poland, September 12–16, 2022, pp. 150-170, Cham: Springer International Publishing, 2022.

Probabilistic Programs

Probabilistic programming languages extend standard programming languages with probabilistic primitives such as random assignments and observe statements. They allow representing complex probability distribution and enable modeling uncertainty in a wide array of applications, ranging from machine learning and artificial intelligence to statistical modeling and data analysis (Gordon et al., 2014).

Defining the semantics of a probabilistic program has been a well-studied problem since the seminal work of Kozen (Kozen, 1983). One of the most-used definitions of the semantics of a probabilistic program sees the program as a transformer over the space of probability distributions. Taking this approach, starting from an initial distribution, also called *prior distribution*, each program instruction transforms the joint distribution over the program variables. The problem of determining the distribution carried by a probabilistic program, called the *posterior distribution*, is commonly referred to as ”probabilistic inference.”

Various techniques have been proposed to tackle this problem, relying on Monte Carlo Markov Chain sampling (Nori et al., 2014; Goodman et al., 2008; V. Mansinghka, Selsam, and Perov, 2014; Pfeffer, 2009; A. Chaganty, Nori, and Rajamani, 2013), variational inference (Bingham et al., 2019; M. I. Jordan et al., 1999; Kucukelbir et al., 2015), symbolic execution (Gehr, Misailovic, and Vechev, 2016; Narayanan et al., 2016; Saad, Rinard, and V. K. Mansinghka, 2021), volume computation (Holtzen, Van den Broeck, and Millstein, 2020; Huang, Dutta, and Misailovic, 2021), and moment-based invariants (Katoen et al., 2010; Chakarov and Sankaranarayanan, 2014; Barthe et al., 2016; Bartocci, Kovács, and Stankovič, 2020; Moosbrugger et al., 2022). Some of the main challenges lay at the interplay between discrete and continuous distributions (Wu et al., 2018), the possible conflicting behaviors of loops and observe statements (Olmedo et al., 2018), and non-termination (Barthe et al., 2016).

The second main contribution of this thesis is to propose a family of semantics, defined using the transformer approach, called Gaussian Semantics, which can be used to approximate arbitrarily well the exact semantics of a probabilistic program. Gaussian Semantics exploit the approximation power of Gaussian Mixtures (Lo, 1972) to approximate the distribution carried by the program. In particular, programs are seen as control flow graphs in which each node acts as a transformer on a probability distribution. Each semantics is parameterized by a map associating each node with the order of the moments to be matched. In a program interpreted according to Gaussian Semantics, each node receives as input a Gaussian Mixture, transforms it according to the exact semantics, and outputs a new Gaussian Mixture matching the true distribution up to a certain order of moments. Notably, in this process, it is not needed to compute the exact transformed distribution, but only its first moments. This, and the Gaussian assumption, significantly simplify the inference problem. Moreover, as the number of moments matched increases, the output distribution yielded by Gaussian Semantics tends to the exact output distribution under mild conditions on the program.

Despite convergence being guaranteed in theory, matching moments above the second order is practically very difficult due to the necessity of solving a polynomial system of equations (Lasserre, 2009). Therefore, an efficient implementation of Gaussian Semantics exists only for the first two-order moments and goes under the name of Second Order Gaussian Approximation (SOGA). Experimental results show that this method can perform accurate inference on various benchmarks from the literature, even when competing approaches do not support inference on the given problem or scale poorly. Moreover, SOGA performs significantly better than its competitors on two classes of models taken from the machine learning literature, namely, models involving mixtures of continuous and discrete distributions (Wu et al., 2018) and collaborative filtering models (Koren, Rendle, and Bell, 2021). The derivation of Gaussian Semantics and the SOGA algorithm are the subject of Chapters 4 and 5, respectively, and refer to the following publication:

- F. Randone, L. Bortolussi, E. Incerto, M. Tribastone, "Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures." *Proceedings of the ACM on Programming Languages*, 8(POPL), 1882-1912.

Candidate's Contribution

Since the work presented in the thesis was published with other co-authors, we specify for each publication the candidate's contribution.

- "Refining mean-field approximations by dynamic state truncation." Francesca Randone developed the theory and performed the numerical experiments, under the supervision of Luca Bortolussi and Mirco Tribastone.

- "Jump Longer to Jump Less: Improving Dynamic Boundary Projection with h-Scaling." Francesca Randone developed the theory and performed the numerical experiments, under the supervision of Luca Bortolussi and Mirco Tribastone.

- "Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures." Francesca Randone developed the theory. The proof of the convergence theorem was developed by Francesca Randone and Luca Bortolussi. The implementation of SOGA and the numerical experiments were carried out by Francesca Randone and Emilio Incerto. Luca Bortolussi and Mirco Tribastone supervised the project.

Chapter 1

Background

In this first chapter, we introduce some of the background and notation needed to understand Chapter 2 and 3. In particular, in the first section, we introduce Markov population processes and their common approximations, while Section 1.2 is devoted to presenting the examples that will appear in the later chapters.

1.1 Approximation of Population Processes

1.1.1 Markov Population Processes

We consider a Markov population process $(X(t) \in \mathcal{S})_{t\geq 0}$ as a process evolving on a set $\mathcal{S} \subseteq \mathbb{N}^m$. We denote by $H(\mathcal{S}) \subseteq \mathbb{R}^m$ the convex hull of $\mathcal{S}$. The initial state of $X(t)$ is denoted by $\bar{x}_0 \in \mathcal{S}$. Given a finite set of jump vectors $\mathcal{L} \subseteq \mathbb{Z}^m$ and a set of transition functions $f_l : H(\mathcal{S}) \to \mathbb{R}_{\geq 0}$, for $l \in \mathcal{L}$, $X(t)$ makes transitions at rate $f_l(x)$ from state x to state $x + l$ for each $l \in \mathcal{L}$.

The exact dynamics of $X(t)$ can be described by its Master Equation (ME), describing the evolution of the probability $P(x;t)$ of being in state x at time t:

$$\frac{dP(x)}{dt} = \sum_{l\in\mathcal{L}} f_l(x-l)P(x-l;t) - \sum_{l\in\mathcal{L}} f_l(x)P(x;t) \quad \forall x \in \mathcal{S}. \tag{1.1}$$

The mean dynamics of $X(t)$ is then given by $\mathbb{E}[X(t)] = \sum_{x\in\mathcal{S}} xP(x;t)$.

1.1.2 Mean-field Approximation

Due to the quick growth of the state space, solving (1.1) is rarely feasible, and it is common to resort to approximations. One of the most common approaches, called *mean-field approximation*, uses a classic limit result, first stated by Kurtz (Kurtz, 1970). In the original formulation, such approximation relies on the *density-dependent* assumption, which is not required to apply our method. Therefore, we give a general definition of mean-field approximation and then specialize it in the case of a density-dependent process.

Defining the *drift* f as

$$f(x) = \sum_{l\in\mathcal{L}} l f_l(x), \tag{1.2}$$

the mean-field approximation of $X(t)$ is the solution to the Cauchy problem:

$$\begin{cases} \frac{dx}{dt} = f(x(t)) \\ x(0) = \bar{x}_0. \end{cases} \tag{1.3}$$

Density-Dependent Processes

Suppose that the process X has size $\gamma_{\bar{N}}$ and there exists a sequence of processes $\left(X^N\right)_{N\geq N_0}$ such that $X^{\bar{N}} = X$ and each X^N has size γ_N with $\lim_{N\to\infty} \gamma_N = \infty$. In the most general case, every X^N is defined on a state space $\mathcal{S}^N$, has initial condition x_0^N and performs transitions $x \to x + l^N$ firing with rate $f_l^N(x)$ for each l in a given set $\mathcal{L}$.

Given a sequence $\left(X^N\right)_{N\geq N_0}$, we consider the sequence of normalized processes $\left(\hat{X}^N\right)_{N\geq N_0}$, where each process has state vector $\hat{X}^N = \frac{1}{\gamma_N} X^N$, starts in $\hat{x}_0^N = \frac{x_0^N}{\gamma_N}$ and performs transitions $x \to x + \frac{l^N}{\gamma_N}$ with rate $\hat{f}_l^N(x) = f_l^N(\gamma_N x)$. We denote the normalized state space by $\hat{\mathcal{S}}^N$.

A particular case is given when the original sequence of processes is *density-dependent*, whereby γ_N grows linearly with N, for each N and

$l \in \mathcal{L}$ there exists a vector v_l such that $\hat{l}^N = \frac{v_l}{\gamma_N}$ and for each N and $l \in \mathcal{L}$ there exists a (locally) Lipschitz continuous and (locally) bounded function $g_l : E \to \mathbb{R}_{\geq 0}$ such that $\hat{f}_l^N(x) = \gamma_N g_l(x)$.

Finally, we define the *drift* $F^N : \hat{\mathcal{S}}^N \to \mathbb{R}^m$ as

$$F^N(x) = \sum_{l \in \mathcal{L}} \hat{l}^N \hat{f}_l^N(x).$$

Under these assumptions the following theorem holds.

Theorem 1 (Convergence to deterministic limit for MPPs (Bortolussi, Hillston, et al., 2013a)). *Let $E \subseteq \mathbb{R}^m$ be a closed set such that $\cup_N \hat{\mathcal{S}}^N \subseteq E$. Suppose that there exists $x_0 \in E$ such that $\lim_N \hat{x}_0^N = x_0$ and a Lipschitz vector field $F : E \to \mathbb{R}^m$ such that*

$$\lim_N \sup_{x \in \hat{\mathcal{S}}^N} \|F^N(x) - F(x)\| = 0.$$

Assuming the rates of convergence in Theorem 4.2 of Bortolussi, Hillston, et al., 2013a are verified, for any fixed time instant $T > 0$ and $\forall\, \epsilon \geq 0$

$$\lim_{N \to \infty} P\left(\sup_{0 \leq t \leq T} \|\hat{X}^N(t) - \hat{x}(t)\| > \epsilon \right) = 0$$

where $\hat{x}(t)$ is the solution to the initial value problem:

$$\begin{cases} \frac{d\hat{x}}{dt} = F(\hat{x}(t)) \\ \hat{x}(0) = x_0 \end{cases} \tag{1.4}$$

and $\hat{x}(t) \in E\, \forall\, t \geq 0$.

Observe that in the case of density-dependent processes, the drift is independent of N and the hypotheses of the Theorem hold trivially.

Moreover, if X is a density-dependent process admitting a deterministic limit defined by Equation (1.4) we have $x(t) = \gamma_N \hat{x}(t)$. In particular, while $\hat{x}(t)$ approximates the normalized process, and therefore the average proportion of agents in a certain class, $N\hat{x}(t)$ approximates the mean number of agents in each class.

1.1.3 Linear Noise Approximation

Another classic limit result has been obtained by Van Kampen (Van Kampen, 1992) applying a "size expansion" to the ME. It takes into account the stochastic fluctuations of the process around its deterministic limit. It can be proved that, in a first-order approximation, such fluctuations behave as a Gaussian process with zero mean. The result is stated in the following theorem.

Theorem 2 (Convergence to Linear Noise Approximation for MPPs (Van Kampen, 1992))**.** *Consider a sequence of density-dependent MPPs $\left(X^N\right)_{N \geq N_0}$, each starting in $X^N(0) = N\hat{x_0}$ and suppose that the drift $F(x)$ is continuously differentiable in E. Then, letting $\left(\hat{X}^N\right)_{N \geq N_0}$ denote the sequence of normalized processes and $\hat{x}(t)$ the solution of the initial value problem in* (1.4) *we have that*

$$\lim_{N \to \infty} \sqrt{N} \|\hat{X}^N - \hat{x}(t)\| = \xi(t) \tag{1.5}$$

where $\xi(t)$ is a Gaussian process identified by equations of the first two moments:

$$\frac{d\mu_i}{dt} = \sum_j \left(\sum_{l \in \mathcal{L}} \frac{\partial f_l}{\partial x_j}(\hat{x}(t)) \right) \mu_j(t) \tag{1.6}$$

$$\frac{d\Sigma_{ij}}{dt} = \sum_k \left(\sum_{l \in \mathcal{L}} l_i \frac{\partial f_l}{\partial x_k}(\hat{x}(t)) \right) \Sigma_{kj}(t) + \sum_k \left(\sum_{l \in \mathcal{L}} l_j \frac{\partial f_l}{\partial x_k}(\hat{x}(t)) \right) \Sigma_{ik}(t) + \\ + \sum_{l \in \mathcal{L}} l_i l_j f_l(\hat{x}(t)) \tag{1.7}$$

1.2 Examples

Example 1 ($M/M/k$ queue)**.** *In the following chapters, we will use the $M/M/k$ queue as a running example. It is defined by the following transition classes, denoting exogenous arrivals with Poisson rate λ and service with rate μ, respectively:*

$$\begin{aligned} l_1 &= +1, && \textit{at rate } \lambda, \\ l_2 &= -1 && \textit{at rate } \mu \min(x, k). \end{aligned}$$

Its Master Equation can be written as:

$$\frac{dP(x)}{dt} = \begin{cases} -\lambda P(0;t) + \mu \min(1,k) P(1;t) & \textit{if } x = 0 \\ -(\lambda + \mu \min(x,k)) P(x;t) + \lambda P(x-1;t) + & \\ \qquad\qquad +\mu \min(x+1,k) P(x+1;t) & \textit{else} \end{cases}$$

Assuming at time 0 *the queue is in a state with 0 customers, the solution of the following Cauchy problem gives its mean-field approximation:*

$$\begin{cases} \frac{dx}{dt} = \lambda + \mu \min(x(t),k) \\ x(0) = 0 \end{cases}$$

A variation of the M/M/k queue is the M/M/k/N queue, in which no more than N jobs are accepted in the queue. The transition classes are:

$$\begin{aligned} l_1 &= +1 \quad \textit{at rate } \lambda \mathbb{I}_{\{x<N\}}, \\ l_2 &= -1 \quad \textit{at rate } \mu \min(x,k) \end{aligned}$$

while the Master Equation becomes:

$$\frac{dP(x)}{dt} = \begin{cases} -\lambda P(0;t) + \mu \min(1,k) P(1;t) & \textit{if } x = 0 \\ -\mu \min(N,k) P(N;t) + \lambda P(N-1,t) & \textit{if } x = N \\ -(\lambda + \mu \min(x,k)) P(x;t) + & \\ \qquad +\mu \min(x+1,k) P(x+1;t) + & \\ \qquad\qquad +\lambda P(x-1;t) & \textit{if } x = 1,\dots,N-1 \end{cases}$$

Observe that, differently from the M/M/k queue, in the M/M/k/N queue, the Master Equation is actually a finite system of ODEs.

Again, assuming at time 0 *the queue is in a state with 0 customers, the mean-field approximation of the M/M/k/N queue is given by the solution of the following Cauchy problem:*

$$\begin{cases} \frac{dx}{dt} = \lambda \mathbb{I}_{\{x<N\}} + \mu \min(x(t),k) \\ x(0) = 0. \end{cases}$$

Example 2 (Coxian Queueing Systems). *We consider an M/Cox/N queuing system with Poisson arrivals with rate* $N\lambda$ *and service rate with a two-phase Coxian distribution, where we assume that the system is density-dependent with size* N*. The three degrees of freedom of the Coxian distribution are identified* *by parameters p (probability of transitioning from first to second phase of service), and μ_1, μ_2 (exponential rates at each stage). Following Bolch et al., 2005, a service-time distribution with mean E and variance V can be obtained by setting $p = E^2/(2V)$, $\mu_1 = 2/E$ and $\mu_2 = E/V$. Using a standard state space description (Bolch et al., 2005), the queuing system can be represented as a Markov population process with state $x = (x_{Q_1}, x_{Q_2}, x_S)$ where: x_{Q_1} is the number of jobs requiring the first phase of service; x_{Q_2} is the number of jobs in second phase; and x_S is the number of servers available for the first phase of service. Thus, the queue length in state x is $x_{Q_1} + x_{Q_2}$. For a system with N independent servers, the jump vectors and the associated transition functions are given by:*

$$\begin{aligned}
l_1 &= +e_{Q_1}, && \text{at rate } N\lambda,\\
l_2 &= -e_{Q_1} - e_S + e_{Q_2} && \text{at rate } p\mu_1 \min(x_{Q_1}, x_S),\\
l_3 &= -e_{Q_1} && \text{at rate } (1-p)\mu_1 \min(x_{Q_1}, x_S),\\
l_4 &= -e_{Q_2} + e_S && \text{at rate } \mu_2(N - x_S),
\end{aligned}$$

where e_i denotes the canonical vector with the i-th coordinate equal to one.

Observe that, due to the presence of the minimum in the transition functions, the mean-field approximation has a Lipschitz continuous but non-differentiable drift.

Larger Number of Phases *We can also consider $M/Cox/N$ queues whose service time is distributed according to a Coxian distribution with more phases. A K-phase Coxian distribution is defined by the parameter vectors $(p_1, \dots, p_{K-1})$ and $(\mu_1, \dots, \mu_K)$. Thus, the queuing system is represented by a Markov population process with state descriptor $x = (x_{Q_1}, \dots, x_{Q_K}, x_{S_1}, \dots x_{S_{K-1}})$. For N independent servers, we define the following transition functions:*

$$\begin{aligned}
a_1 &= +e_{Q_1}, & f_{a_1} &= N\lambda,\\
d_1 &= -e_{Q_1}, & f_{d_1} &= (1-p_1)\mu_1 \min(x_{Q_1}, x_{S_1}),\\
&\dots\\
a_i &= -e_{Q_{i-1}} - e_{S_{i-1}} + e_{Q_i} + e_{S_i} & f_{a_i} &= p_{i-1}\mu_{i-1} \min(x_{Q_{i-1}}, x_{S_{i-1}}),\\
d_i &= -e_{Q_i} - e_{S_i} + e_{S_1} & f_{d_i} &= p_i\mu_i \min(x_{Q_i}, x_{S_i}),\\
&\dots\\
a_K &= -e_{Q_{K-1}} - e_{S_{K-1}} + e_{Q_K} & f_{a_K} &= p_{K-1}\mu_{K-1} \min(x_{Q_{K-1}}, x_{S_{K-1}})\\
d_K &= -e_{Q_K} + e_{S_1} & f_{d_K} &= \mu_K \min\left(x_{Q_K}, N - \sum_{i=1}^{K-1} x_{S_i}\right).
\end{aligned}$$

Example 3 (Malware Propagation Model). *We consider the malware propagation model from Gast, Bortolussi, and Tribastone, 2019 (also proposed in Benaim and Le Boudec, 2008a and Khouzani, Sarkar, and Altman, 2012). It is composed of N nodes, where each node can be dormant (D), active (A), or susceptible (S). Since the total number of nodes is constant, the model can be described by the state vector $x = (x_D, x_A)$, where the number of susceptible nodes at each state is given by $N - x_D - x_A$. The process evolves according to the following transitions and jump vectors:*

$$\begin{aligned}
l_1 &= -e_D + e_A && \text{at rate } \left(1 + \frac{10x_A}{x_D + N\delta} x_D\right), \\
l_2 &= -e_A && \text{at rate } 5x_A, \\
l_3 &= +e_D && \text{at rate } \left(\beta + \frac{10}{N} x_D\right)(N - x_D - x_A).
\end{aligned}$$

In Gast, Bortolussi, and Tribastone, 2019, it is discussed that there exists a parameter value $\delta^* \approx 0.18$ such that for $\delta > \delta^*$ the mean-field approximation has a unique attractor; instead, for $\delta < \delta^*$ the ODE has an orbit cycle, which may cause significant approximation errors.

Example 4 (Egalitarian Processor Sharing). *We consider a simplified version of a queuing system with generalized processor sharing proposed in Parekh and Gallager, 1993 and Parekh and Gallager, 1994. We apply an* egalitarian *policy whereby all customers are assigned the same weight; in particular, we use the rate functions adopted in L. Zhu, Casale, and Perez, 2020. A system with K classes of customers can be represented as a Markov population process with state $x = (x_{Q_1}, \dots, x_{Q_K})$, where each component identifies a class of customers in the queue. The jump vectors and the transition functions associated with the process are:*

$$\begin{aligned}
a_i &= +e_{Q_i}, && \text{at rate } N\lambda_i, \\
d_i &= -e_{Q_i}, && \text{at rate } \mu \frac{x_{Q_i}}{\sum_{j=1}^{K} x_{Q_j} + N},
\end{aligned}$$

for $i = 1, \dots, K$.

Chapter 2

Dynamic Boundary Projection

The following two chapters of this thesis are devoted to efficient approximation and analysis of Markov population processes. Section 2.1 covers broadly the current state-of-the-art on the mean-field approximation of Markov population processes and presents an overview of our approach. In Section 2.2, the equations for Dynamic Boundary Projection are formally derived, first stating the assumptions under which our approach is applicable (Section 2.2.1), then introducing the joint process (Section 2.2.2) and augmented truncation approximations (Section 2.2.3), and finally introducing the whole dynamical framework (Section 2.2.4). Our main convergence theorem is proved in Section 2.3. Our proof is divided into three steps: first the dynamical system is decomposed into a linear and non-linear part (Section 2.3.1), then convergence for each part is proved separately in Section 2.3.2 and 2.3.3. Finally, numerical examples are presented in Section 2.4.

2.1 State-of-the-Art

Mean-field models are a well-known technique for the analysis of stochastic systems describing a population of N interacting objects (Benaim and

Le Boudec, 2008a; Kurtz, 1978). When N is large, the exact analysis of such models using Equation 1.1 is generally prohibitive: since closed-form solutions are available only in special cases, one must resort to computationally demanding numerical simulations to cope with state spaces that grow exponentially with m^N in the worst case, where $m \ll N$ is the cardinality of the state space of the individual object. Mean-field theory, instead, provides a simple system of differential (or difference) equations of size m, in the form of Equation 1.3, with guarantees of convergence under mild hypotheses on the stochastic system as N goes to infinity both in the transient and in the steady-state (Theorem 1).

The asymptotic results of convergence of mean-field theory provide no guarantees as to the quality of the approximation for finite N, which is model- and parameter-dependent in general. This has stimulated research into providing error bounds (Ying, 2016; Ying, 2017; X. Liu, Ying, et al., 2020; Bortolussi and Hayden, 2013) and rates of convergence (R.W.R. Darling and J. Norris, 2008; Gast, 2017; Vasantam and Mazumdar, 2019).

Recently, the problem of refining the mean-field approximation has attracted the attention of the performance evaluation community. Gast demonstrated that the expected value of a performance functional converges to the mean-field limit at rate $O(1/N)$ (Gast, 2017), both in the transient and in the stationary regime (under the assumption of a unique attractor that is exponentially locally stable). Later, Gast and Van Houdt used a Lyapunov argument to compute the constant associated with the $1/N$ term for the steady-state expectation of such functionals, effectively providing a refinement of the approximation for finite N (Gast and Van Houdt, 2017). The method is further extended with an expansion of the term $1/N^2$ for both the transient and the steady-state regimes (Gast, Bortolussi, and Tribastone, 2019; Grima, 2010).

These mean-field refinements fundamentally assume differentiability of the drift, i.e., the vector field of the limit system. This rules out their applicability to a class of queuing models for which corrections to the mean-field estimates of average queue lengths may be desirable. As a motivating example, let us consider the $M/Cox/N$ queue introduced in Example 2. Table 1 shows the errors between the average queue length (normal-

Table 1: Transient average queue length (normalized by N) at time $t = 100$ (computed by simulation) and mean-field estimate for an $M/Cox/N$ queue with arrivals at rate $N\lambda$, with $\lambda = 0.75$, and a two-phase Coxian distribution with unitary service time and varying variance (V).

Scaling	$V = 5$	$V = 10$	$V = 20$
$N = 1$	6.09	8.23	10.27
$N = 5$	1.47	1.89	2.30
$N = 10$	0.97	1.10	1.25
Mean field approximation	0.75	0.75	0.75

ized by N) and its mean-field approximation for service times distributed with a two-phase Coxian distribution with unitary mean and increasing variance, using the already mentioned fitting formulae in Bolch et al., 2005. The simulations, computed at an arbitrary time $t = 100$ for which the mean-field approximation approaches a stationary regime, show errors that decrease with N, but can be large for small N; in particular, the errors increase with the variance of the Coxian distribution for any fixed N. Furthermore, the mean-field approximation numerically shows insensitivity to the service process variance, always equal to λ—a fact that has been discussed in a more general setting in Bramson, Lu, and Prabhakar, 2010.

We conclude that the mean-field approximation may not adequately represent the stochastic queue length dynamics for finite, small, N. The aforementioned mean-field refinement results cannot be applied in this particular case since they require differentiability of the drift. However, for the model taken into consideration, the presence of the minimum function in the rate transitions of the population process makes the drift piece-wise linear and non-differentiable [1].

[1] For instance, in the special case of exponentially distributed service times, i.e., for an $M/M/N$ queue, service occurs with rate $\mu \min(X, N)$ where X is the queue length and μ is the service rate.

2.1.1 Proposed Approach

Due to the substantial lack of a satisfying approximation method for population processes with non-differentiable drift, we present a new method to refine average estimates. Although the main application lies with mean-field models, the method does not make use of the deterministic ODE equation as the limit behavior of the re-scaled Markov process when N goes to infinity; in particular, it does not assume a density-dependent process. Instead, it starts from the equation for the "true" average evolution of a population process $X(t)$,

$$\frac{d\mathbb{E}\left[X(t)\right]}{dt} = \mathbb{E}\left[f(X(t))\right],$$

where f is the drift defined in (1.2) (Singh and Hespanha, 2006; Van Kampen, 2007; Bortolussi, Hillston, et al., 2013b). When f is not linear, dependence on higher order moments is introduced in the left-hand side, preventing a direct solution of the previous equation. Hence the approximation of the average essentially consists in assuming $\mathbb{E}\left[f(X(t))\right] \approx f(\mathbb{E}\left[X(t)\right])$ in the above equation, yielding

$$\frac{d\mathbb{E}\left[X(t)\right]}{dt} = f(\mathbb{E}\left[X(t)\right]). \tag{2.1}$$

In the case of density-dependent models, the rescaled process $X^N(t)/N$ with drift f converges to the solution of the ODE $dx(t)/dt = f(x(t))$ as N tends to infinity, which has the same functional form as (2.1) but $x(t)$ is interpreted as a density/proportion of objects rather than a population. For this reason, we shall still refer to (2.1) as the equations for the mean-field approximation.

Our proposal for refining average estimates is based on the definition of an auxiliary process $Y(t)$, depending on the original process $X(t)$, such that each state y of $Y(t)$ identifies a hyper-rectangular subset of states in which $X(t)$ is at time t. Using a suitable state space truncation, the transition rate matrix governing the evolution of the probability distribution of $X(t)$ on y can be expressed as a function of y alone. Specifically, we consider a truncation-and-augmentation scheme where the transition rates toward states outside a given truncation are redirected to the "closest" state on its boundary. Augmented truncations preserve the stochasticity of the transition rate matrix of the truncated process, as opposed to the classical approaches for Markov chains in discrete (Seneta, 1967; Seneta, 1968) and continuous (Tweedie, 1971; Tweedie, 1973) time; they have received attention due to theoretical guarantees on the convergence of their stationary distributions to that of the original process (Hart and Tweedie, 2012) and the possibility to provide useful lower and upper bounds on the steady-state probability of single states (Y. Liu, Li, and Masuyama, 2018; Y. Liu and Li, 2018; Masuyama, 2017). Importantly for our refinement method, with the proposed augmentation it is possible to approximate $Y(t)$—which is non-Markovian—as a time-inhomogeneous Markov population process for which we can write equations for the mean-field approximation.

Overall, this results in a family of refined approximations depending on a parameter $n \in \mathbb{N}^m$ that gives the size of the truncation, i.e., the volume of the hyper-rectangle that defines the truncated state space. For any finite n, the approximation consists in a system of ODEs where the Master Equation governing the transient probability distribution of the truncated process is modulated by a continuous variable representing the mean-field equations of $Y(t)$. Effectively, this leads to a *dynamic* shift of the truncation across the state space of the original process. Due to the specific choice of truncation scheme, we call our method *dynamic boundary projection* (DBP).

Theoretically, the main formal result concerns a statement of asymptotic correctness. It is proved that, as $n \to \infty$ and under the assumption of boundedness of the drift, the family of DBP approximations converge to the original population process $X(t)$. Usually, truncation approximation methods yield linear systems of ODEs (Dinh and Sidje, 2016; Kuntz et al., 2019) for which component-wise or total variation convergence of the solution to the Master Equation can be proved using standard techniques (see Munsky and Khammash, 2006a and Hart and Tweedie, 2012 for convergence over finite time intervals and stationary distributions, respectively). DBP yields a nonlinear ODE system due to the coupling of the (linear) Master Equation with the (usually nonlinear) drift of $X(t)$. In this case, we need a stronger convergence result, namely uniform convergence in 1-norm in a suitable Banach space, and a different proof strategy, which we base on a perturbation argument for nonlinear ODEs.

From a numerical viewpoint, we apply DBP to three examples. First, we consider the aforementioned Coxian queuing system to show how the refinements can improve the mean-field approximation such that it is not insensitive to the variance of the service-time distribution. As discussed this is done on a population process that has a non-differentiable drift, hence available mean-field refinement methods are not applicable. Second, we consider the model of malware propagation (Gast, Bortolussi, and Tribastone, 2019) presented in Example 3 to show the instability in the computation of the refined $1/N$ and $1/N^2$ terms in the presence of orbit cycles in the mean-field approximation. Under these conditions, we numerically show that DBP does not exhibit instability and can improve the mean-field approximations, while it performs similarly to the refined scheme of Gast, Bortolussi, and Tribastone, 2019 if the mean-field model has a unique attractor. Finally, we consider the multi-class queuing system (Parekh and Gallager, 1994) with processor sharing discipline presented in Example 4 as a case study to show how the choice of the parameter n may impact the quality of the approximation.

2.2 Derivation of DBP

2.2.1 Assumptions

From now on we will assume that the following two assumptions on the drift are verified:

(H1) there exists a constant $L > 0$ such that

$$\|f(x+h) - f(x)\|_1 \leq L\|h\|_1 \, \forall x, x+h \in H(\mathcal{S});$$

(H2) there exists a constant $C > 0$ such that

$$\|f(x)\|_1 \leq C \, \forall \, x \in H(\mathcal{S}).$$

The first condition (local Lipschitz continuity) ensures the existence and uniqueness of the solution of (1.3). The second condition (boundedness of the drift) is required to prove uniform convergence in 1-norm of the proposed approximation to the original process (see Section 2.3). Observe that these conditions have to hold on the convex hull $H(\mathcal{S})$ and not only on the set of states since in our approximation $f(x)$ will also be evaluated on continuous values of x.

Remark 1. *We stress that this formulation does not assume density dependence. Hence, from now on each state of the Markov population process will be an unscaled population vector (rather than a vector describing the proportions of objects in every local state). The mean-field equation* (1.3) *is thus interpreted as the approximate dynamics of the average populations.*

Remark 2. *The hypothesis $\mathcal{S} \subseteq \mathbb{N}^m$ can be easily extended to the case $\mathcal{S} \subseteq \mathbb{Z}^m$. However, presenting the theory using the former hypothesis is more convenient since all our examples will evolve on $\mathbb{N}^m$.*

2.2.2 Joint Process

We now define a second process $Y(t)$ depending on $X(t)$, whose value identifies the truncation in which X is at time t. We fix a bound $n \in \mathbb{N}^m$ and define for each $y \in \mathcal{S}$ the set of states:

$$\mathcal{T}_y^{(n)} = \{x \in \mathcal{S} : y_i \leq x_i \leq y_i + n_i \quad \forall\, i = 1, \ldots, m\}. \tag{2.2}$$

We say that $\mathcal{T}_y^{(n)}$ is a (hyper-rectangular) truncation of the state space indexed by y.

For every $x, y \in \mathcal{S}$ we define $\Pi^{(n)}(x, y) \in \mathbb{Z}^m$ as:

$$\Pi_i^{(n)}(x,y) = \begin{cases} x_i & \text{if } x_i < y_i \\ x_i - n_i & \text{if } x_i > y_i + n_i \\ y_i & \text{if } y_i \leq x_i \leq y_i + n_i \end{cases} \qquad \forall\, i = 1, \ldots, m.$$

Intuitively, $\Pi^{(n)}(x, y)$ returns a vector y' such that the truncation $\mathcal{T}_{y'}^{(n)}$ is the closest truncation to $\mathcal{T}_y^{(n)}$ that contains x. We use $\Pi^{(n)}(x, y)$ to build a coupled process $Y(t)$ that tracks the truncation in which $X(t)$ is at time t. We define

$$Y(t) = y_k \qquad \forall t \in [t_k, t_{k+1}),$$

where

$$\begin{aligned} y_0 &= \Pi^{(n)}(\bar{x}_0, 0), & y_k &= \Pi^{(n)}(X(t_k), y_{k-1}), \\ t_0 &= 0, & t_k &= \inf\left\{t > t_{k-1} : X(t) \notin \mathcal{T}^{(n)}_{y_{k-1}}\right\}. \end{aligned}$$

According to this definition, $Y(t) = y_k$ means that at time t the original process $X(t)$ is in a state belonging to the truncation $\mathcal{T}^{(n)}_{y_k}$; $Y(t)$ changes value whenever $X(t)$ makes a jump into a state x outside the current truncation. Observe that $Y(t)$ is uniquely defined although x may belong to different truncations: the new value of $Y(t)$ is the truncation containing the current state closest to the previous truncation.

Denoting the characteristic function by $\mathbb{I}$, the joint process $(X(t), Y(t))$ is a time-homogeneous continuous-time Markov chain defined by the following transitions:

- $(X(t), Y(t))$ jumps from (x, y) to $(x+l, y)$ with rate $\mathbb{I}_{\{x+l\in\mathcal{T}^{(n)}_y\}} f_l(x)$ for every $l \in \mathcal{L}$;

- $(X(t), Y(t))$ jumps from (x, y) to $(x + l, \Pi^{(n)}(x + l, y))$ with rate $\mathbb{I}_{\{x+l\notin\mathcal{T}^{(n)}_y\}} f_l(x)$ for every $l \in \mathcal{L}$.

By definition of $\Pi^{(n)}$, the joint process evolves only on states (x, y) such that $x \in \mathcal{T}^{(n)}_y$.

The Master Equation for the transient joint probability distribution $P(x, y; t)$ of $(X(t), Y(t))$ is:

$$\frac{dP(x,y)}{dt} = -\left[\sum_{l\in\mathcal{L}} f_l(x)\right] P(x, y; t) + \tag{2.3}$$

$$+\sum_{l\in\mathcal{L}} \left[f_l(x-l)P(x-l, y; t) + \right.$$
$$\left. + \sum_{y':\Pi^{(n)}(x,y')=y} \mathbb{I}_{\{x\notin\mathcal{T}^{(n)}_{y'}\}} f_l(x-l)P(x-l, y'; t)\right]. \tag{2.4}$$

The sum in (2.3) accounts for the outgoing probability flux from (x, y) due to all possible jumps, whether exiting the current truncation or not.

The first summand in (2.4) accounts for the incoming probability flux into state (x, y) due to the transitions taking place in states (x', y) belonging to the same truncation $\mathcal{T}_y^{(n)}$ (observe that if $x - l \notin \mathcal{T}_y^{(n)}$, then $P(x - l, y; t) = 0$ for all t). The second summand in (2.4) accounts for the incoming probability flux due to those transitions starting in states (x', y') belonging to different truncations $\mathcal{T}_{y'}^{(n)}$ but having (x, y) as target state.

Example. *Let us consider the $M/M/k$ queue of Example 1 and let us write the Master Equation of the joint process $(X(t), Y(t))$. The summand in* (2.3) *and the first summand in* (2.4) *can be rewritten directly, so we focus on the second summand in* (2.4)*. For this term, the only non-zero contributions are given by those x and y' satisfying:*

$$\begin{aligned} x \notin \mathcal{T}_{y'}^{(n)} &\Rightarrow (x < y') \vee (x > y' + n), \\ x - l \in \mathcal{T}_{y'}^{(n)} &\Rightarrow y' \leq x - l \leq y' + n, \\ \Pi^{(n)}(x, y') &= y. \end{aligned}$$

Applying the first two conditions with $l_1 = +1$ gives $x = y' + n + 1$, while from the last condition, we get $y' = y - 1$; thus, this term appears in the Master Equation only when $x = y + n$. This agrees with the intuition that when the process is in $(y - 1 + n, y - 1)$ (last state of the truncation indexed by $y - 1$) and a new customer arrives, it jumps to the next truncation, indexed by y.

Applying the same argument to the service transition, we get $x = y' - 1$ and $y' = y + 1$, so that this term appears in the equation only when $x = y$. That is, from state $(y + 1, y + 1)$ (the first state of the truncation indexed by $y + 1$), upon service the process jumps to the previous truncation, indexed by y.

The resulting Master Equation is:

$$\begin{aligned} \frac{d}{dt} P_{x,y}(t) = &- (\lambda + \mu \min(x, k)) P_{x,y}(t) + \\ &+ \lambda P_{x-1,y}(t) + \mu \min(x + 1, k) P_{x+1,y}(t) + \\ &+ \mathbb{I}_{\{x=y+n\}} \lambda P_{y+n-1,y-1}(t) + \\ &+ \mathbb{I}_{\{x=y\}} \mu \min(y + 1, k) P_{y+1,y+1}(t). \end{aligned}$$

□

From (2.3)-(2.4) we can derive the expression for the exact mean of

$X(t)$ as it follows:

$$\begin{aligned}\mathbb{E}[X(t)] &= \sum_{x\in\mathcal{S}} xP(x;t) = \sum_{x\in\mathcal{S}} \sum_{y:x\in\mathcal{T}_y^{(n)}} xP(x,y;t)\\ &= \sum_{y\in\mathcal{S}} \sum_{x\in\mathcal{T}_y^{(n)}} xP(x,y;t)\\ &= \sum_{y\in\mathcal{S}} P(y;t) \sum_{x\in\mathcal{T}_y^{(n)}} xP(x|\, y;t).\end{aligned}$$

We observe that we can express all the states in a truncation $\mathcal{T}_y^{(n)}$ shifting by y the states in $\mathcal{T}_0^{(n)}$ so that the previous expression can be rewritten as

$$\mathbb{E}[X(t)] = \sum_{y\in\mathcal{S}} P(y;t) \sum_{x\in\mathcal{T}_0^{(n)}} (x+y)P(x+y|\, y;t). \tag{2.5}$$

The above expression is exact. In Section 2.2.3 we consider an approximation of the inner sum by means of a truncation of the state space. Based on this, in Section 2.2.4 we study a mean-field approximation of the outer sum. We prove the asymptotic correctness of these approximations in Section 2.3.

2.2.3 Augmented Truncation Approximations

Let us fix a value of n. Since the inner sum of (2.5) considers only states in a truncation $\mathcal{T}_y^{(n)}$, we aim to provide an approximate distribution of $X(t)$ on such subset of the state space. In order to do so, we consider an augmented truncation (see, e.g., Y. Liu, Li, and Masuyama, 2018 and references therein). Applying an augmented truncation corresponds to restricting the transition rate matrix of $X(t)$ to the states in $\mathcal{T}_y^{(n)}$ and redirecting the transitions from states $x \in \mathcal{T}_y^{(n)}$ to states $x' \notin \mathcal{T}_y^{(n)}$ to new target states $x^* \in \mathcal{T}_y^{(n)}$, with the same rates.

By doing this, it is possible to define a new process $X_y^{(n)}(t)$ with transition rate matrix $Q^{(n)}(y)$, such that if $X(t)$ and $X_y^{(n)}(t)$ start from the same initial condition in $\mathcal{T}_y^{(n)}$, they will evolve identically until $t^* =$

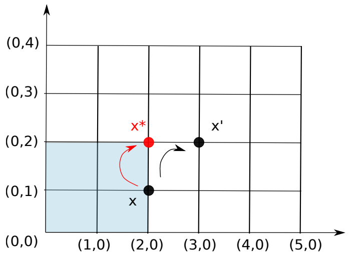


Figure 1: Example of the application of BP on $\mathcal{T}_{(0,0)}^{(2,2)}$. A reaction from $(2,1)$ to $(3,2)$ is redirected to $(2,2)$, the state in the current truncation closest to the real target state.

$\inf\{t > 0 : X(t) \notin \mathcal{T}_y^{(n)}\}$. At time t^*, $X(t)$ jumps outside $\mathcal{T}_y^{(n)}$, while $X_y^{(n)}$ will perform a transition with an identical rate to a state still in $\mathcal{T}_y^{(n)}$.

Observe that the transitions that need to be redirected are those starting in states "on the border" of a truncation. This leads us to define the following sets

$$\partial\mathcal{T}_y^{(l,n)} = \left\{x \in \mathcal{T}_y^{(n)} : x + l \notin \mathcal{T}_y^{(n)}\right\}, \text{ for } l \in \mathcal{L},$$
$$\partial\mathcal{T}_y^{(n)} = \bigcup_{l \in \mathcal{L}} \mathcal{T}_y^{(l,n)} = \left\{x \in \mathcal{T}_y^{(n)} : \exists\, l \in \mathcal{L} \text{ s.t. } x + l \notin \mathcal{T}_l^{(n)}\right\}$$

for which it is immediate to show that:

$$\partial\mathcal{T}_{y+w}^{(l,n)} = \partial\mathcal{T}_y^{(l,n)} + w = \left\{x + w : x \in \partial\mathcal{T}_y^{(l,n)}\right\}.$$

We introduce a specific augmented truncation that we call *boundary projection* (BP), in which every jump from $x \in \partial\mathcal{T}_y^{(l,n)}$ is redirected with same rate to a state x^* defined as:

$$x_i^* = \begin{cases} \min(y_i + n_i, x_i') & \text{if } x_i' > x_i \\ \max(y_i, x_i') & \text{if } x_i' < x_i \\ x_i & \text{if } x_i' = x_i. \end{cases} \tag{2.6}$$

Essentially, x^* is the projection of the target state on the boundary of the current truncation; in other words, it is the state in the current truncation closest to the real target state (see Figure 1).

After performing the augmentation, for each state $x \in \mathcal{T}_y^{(n)}$ we have a set of jump vectors $\tilde{\mathcal{L}}^{(n)}(x)$ such that for every $l \in \mathcal{L}$ we can now define a new vector $\tilde{l}^{(n)}(x)$ given by:

$$\tilde{l}^{(n)}(x) = \begin{cases} l & \text{if } x \notin \partial\mathcal{T}_y^{(l,n)}, \\ x^* - x & \text{if } x \in \partial\mathcal{T}_y^{(l,n)}, \end{cases}$$

where x^* is the target state in which the transition $x \to x + l$ has been redirected and the associated transition rates are $f_{\tilde{l}^{(n)}(x)}(x) = f_l(x)$.

With this choice, the transition rate matrices $Q^{(n)}(y)$ have the same functional form for every y. This will be derived by the following invariance property.

Proposition 1. *Assume that $x \in \partial\mathcal{T}_y^{(l,n)}$ and for $X_y^{(n)}$ transition l is redirected to state $x^* \in \mathcal{T}_y^{(n)}$, defined by BP as in* (2.6)*. Then, for any $w \in \mathbb{Z}^m$, for the process $X_{y+w}^{(n)}$, the transition from $x + w \in \partial\mathcal{T}_{y+w}^{(l,n)}$ is redirected to $x^* + w \in \mathcal{T}_{y+w}^{(n)}$.*

Proof. Suppose that for a truncation $\mathcal{T}_y^{(n)}$ and a state $x \in \partial\mathcal{T}_y^{(l,n)}$, applying Boundary Projection the transition l is redirected to x^* given by (2.6). Let $w \in \mathbb{Z}^m$. Then, $x + w \in \partial\mathcal{T}_{y+w}^{(l,n)}$ and the transition is redirected to $(x + w)^*$ such that:

- if $(x + w + l)_i > (x + w)_i$, then

  $$\min(y_i + w_i + n_i, x_i + w_i + l_i) = w_i + \min(y_i + n_i, x_i + l_i) = w_i + x_i^*;$$

- if $(x + w + l)_i < (x + w)_i$, then

  $$\max(y_i + w_i, x_i + w_i + l_i) = w_i + \max(y_i, x_i + l_i) = w_i + x_i^*.$$

□

We can now derive the Master Equation for $X_y^{(n)}$. Observe that for each truncation the total number of states is $\mathcal{N}(n) = \prod_{i=1}^m (n_i + 1)$. For a fixed y, using the definition in (2.6), we obtain the rate transition matrix $Q^{(n)}(y)$ where each component $[Q_y^{(n)}]_{x,x'}$ for $x, x' \in \mathcal{T}_y^{(n)}$ is given by:

$$[Q^{(n)}(y)]_{x,x'} = \begin{cases} \sum_{l \in \mathcal{L}} \mathbb{I}_{\{x' + \tilde{l}^{(n)}(x') = x\}} f_l(x') & \text{if } x \neq x', \\ -\sum_{l \in \mathcal{L}} \mathbb{I}_{\{\tilde{l}^{(n)}(x) \neq 0\}} f_l(x) & \text{if } x = x', \end{cases}$$

Observe that if a transition starting in x is redirected to x itself, we remove this transition from the approximated process $X_y^{(n)}$. Since Proposition 1 assures the invariance by translation of the target states, and all the states in a given truncation $\mathcal{T}_y^{(n)}$ can be written shifting by y the states in $\mathcal{T}_0^{(n)}$, for $x, x' \in \mathcal{T}_0^{(n)}$ we can rewrite the previous matrix as follows:

$$[Q^{(n)}(y)]_{x,x'} = \begin{cases} \sum_{l \in \mathcal{L}} \mathbb{I}_{\{x' + \tilde{l}^{(n)}(x') = x\}} f_l(x' + y) & \text{if } x \neq x' \\ -\sum_{l \in \mathcal{L}} \mathbb{I}_{\{\tilde{l}^{(n)}(x) \neq 0\}} f_l(x + y) & \text{if } x = x'. \end{cases} \tag{2.7}$$

We can see from this expression that the functional form of the transition rate matrix remains the same as y varies. Then, the resulting Master Equation for BP can be written as:

$$\frac{dP_y^{(n)}}{dt} = Q^{(n)}(y) P_y^{(n)}(\,\cdot\,; t) \tag{2.8}$$

where $P_y^{(n)}(\,\cdot\,; t)$ is an $\mathcal{N}(n)$-dimensional vector indexed by the states in $\mathcal{T}_0^{(n)}$ and each component $P_y^{(n)}(x; t)$ gives the probability of $X_y^{(n)}$ being in the state $x + y$.

We can now approximate the inner sum in (2.5) using $P_y^{(n)}(\,\cdot\,; t)$ as an approximation for $P(\,\cdot\, + y|\, y; t)$. This is justified by the fact that if the original process $X(t)$ never exits the truncation $\mathcal{T}_y^{(n)}$ for all $0 \leq t \leq T$, where T is a given finite time horizon, then $P_y^{(n)}(\,\cdot\,; t) = P(\,\cdot\, + y|\, y; t)$ for all $0 \leq t \leq T$. The resulting approximation is:

$$\mathbb{E}[X(t)] \approx \sum_{y \in \mathcal{S}} P(y; t) \sum_{x \in \mathcal{T}_0^{(n)}} (x + y) P_y^{(n)}(x; t). \tag{2.9}$$

Example. *Let us now apply BP to the $M/M/k$ queue in $\mathcal{T}_y^{(n)}$. We have $\partial\mathcal{T}_y^{(-1,n)} = \{y\}$ and $\partial\mathcal{T}_y^{(+1,n)} = \{y + n\}$, thus we need to redirect the transitions $y \to y - 1$ and $y + n \to y + n + 1$. According to* (2.6)*, they will have new target states y and $y + n$, respectively. Observe that since the transitions are redirected to the starting states in both cases, we will remove these transitions from the approximated process.*

The Master Equation of the approximated process on $\mathcal{T}_y^{(n)}$ is then given by:

$$\frac{dP_y^{(n)}(x)}{dt} = \begin{cases} -\lambda P_y^{(n)}(0;t) + +\mu \min(1+y,k) P_y^{(n)}(1;t) & \textit{if } x = 0 \\ -\mu \min(n+y,k) P_y^{(n)}(n;t) + \lambda P_y^{(n)}(n-1;t) & \textit{if } x = n \\ -(\lambda + \mu \min(x+y,k)) P_y^{(n)}(x;t) + & \\ \quad +\lambda P_y^{(n)}(x-1;t) + & \\ \qquad +\mu \min(x+1+y,k) P_y^{(n)}(x+1;t) & \textit{else} \end{cases}$$

where $P_y^{(n)}(x;t)$ is interpreted as the probability of $X_y^{(n)}$ being in the state $x+y$. Observe that this is equivalent to an $M/M/k/n$ queue in which there are always at least y jobs in the queue. Moreover, choosing a different value of y does not change the functional form of the vector field $Q^{(n)}(y)$. □

2.2.4 Dynamic Boundary Projection

To approximate the outer sum in (2.9), from (2.3)-(2.4) we can derive the equations for the evolution of the marginal probability distribution $P(y;t) = \sum_{x \in \mathcal{S}} P(x,y;t)$ of the joint process.

After some simple manipulations, we get:

$$\begin{aligned} \frac{dP(y)}{dt} = &- \sum_{l \in \mathcal{L}} \sum_{x \in \mathcal{S}} \mathbb{I}_{\{x+l \notin \mathcal{T}_y^{(n)}\}} f_l(x) P(x,y;t) + \\ &+ \sum_{l \in \mathcal{L}} \sum_{x} \sum_{y' : \Pi^{(n)}(x+l,y')=y} \mathbb{I}_{\{x+l \notin \mathcal{T}_{y'}^{(n)}\}} f_l(x) P(x,y';t), \end{aligned} \tag{2.10}$$

where the first term on the right-hand side describes the outgoing probability flux from $\mathcal{T}_y^{(n)}$, due to transitions taking place in states $x \in \partial\mathcal{T}_y^{(n)}$; the second term instead describes the incoming probability flux into $\mathcal{T}_y^{(n)}$ due to transitions from states $x \in \partial\mathcal{T}_{y'}^{(n)}$ such that $x + l \in \mathcal{T}_y^{(n)}$ for some l. Observe that since in general $Y(t)$ is not Markovian, it is not possible to express the equations for $Y(t)$ in a closed form independent from $X(t)$.

Similarly to what done in (2.5), we now express all the states in a truncation $\mathcal{T}_y^{(n)}$ shifting by y the states in $\mathcal{T}_0^{(n)}$. For a fixed y, the possible values of y' such that $\Pi^{(n)}(x+l,y') = y$ can be obtained shifting by y the values of y' such that $\Pi^{(n)}(x+l,y') = 0$. This follows from the following property.

Proposition 2. *$\Pi^{(n)}(x,y) = \Pi^{(n)}(x-y,0) + y$ for all $x, y \in \mathcal{S}$.*

Proof. Proceeding by cases:

$$\Pi_i^{(n)}(x,y) = x_i - n_i \Leftrightarrow x_i > y_i + n_i \Leftrightarrow$$
$$\Leftrightarrow x_i - y_i > n_i \Leftrightarrow \Pi_i^{(n)}(x-y,0) = x_i - y_i - n_i;$$
$$\Pi_i^{(n)}(x,y) = x_i \Leftrightarrow x_i < y_i \Leftrightarrow x_i - y_i < 0 \Leftrightarrow \Pi_i^{(n)}(x-y,0) = x_i - y_i;$$
$$\Pi_i^{(n)}(x,y) = y_i \Leftrightarrow y_i \le x_i \le y_i + n_i \Leftrightarrow$$
$$\Leftrightarrow 0 < x_i - y_i < n_i \Leftrightarrow \Pi_i^{(n)}(x-y,0) = 0.$$

□

Thus we have $\Pi^{(n)}(x+y'+l,y') = y$ if and only if $\Pi^{(n)}(x+l,0) = y-y'$; therefore, for a fixed y, the values taken by y' are given exactly by $y - \Pi^{(n)}(x+l,0)$ for $l \in \mathcal{L}$ and all $x \in \partial\mathcal{T}_0^{(l,n)}$. This leads us to define the function $\mathcal{Y}^{(l,n)}(x) = \Pi^{(n)}(x+l,0)$ for all $l \in \mathcal{L}, x \in \partial\mathcal{T}_0^{(l,n)}$.

Example. *For an $M/M/k$ queue and a fixed y, we want to find the values of y' for which $\Pi^{(n)}(x+l,y') = y$, i.e., we want to find the truncations $\mathcal{T}_{y'}^{(n)}$ from which we can transition to $\mathcal{T}_y^{(n)}$. As discussed, this happens when an arrival takes place in $(n+y-1,y-1)$ and when a service takes place in $(y+1,y+1)$; thus, the possible values of y' are exactly $y-1$ and $y+1$. We can obtain these values also in the following way:*

- *if $l=-1$, $\partial\mathcal{T}_0^{(-1,n)} = \{0\}$, so $y' = y - \mathcal{Y}^{(-1,n)}(0) = \Pi^{(n)}(-1,0) = -1$;*
- *if $l=+1$, $\partial\mathcal{T}_0^{(+1,n)} = \{n\}$, so $y' = y - \mathcal{Y}^{(+1,n)}(n) = \Pi^{(n)}(n+1,0) = 1$.*

□

We now replace the joint probabilities with the conditionals and rewrite (2.10) as (time dependence is suppressed to improve readability):

$$\frac{dP(y)}{dt} = -\sum_{l\in\mathcal{L}} \sum_{x\in\partial\mathcal{T}_0^{(l,n)}} f_l(x+y)P(x+y|\,y)P(y)+$$
$$+\sum_{l\in\mathcal{L}} \sum_{x\in\partial\mathcal{T}_0^{(l,n)}} f_l(x+y-\mathcal{Y}^{(l,n)}(x))P(x+y-\mathcal{Y}^{(l,n)}(x)|\,y-\mathcal{Y}^{(l,n)}(x))P(y-\mathcal{Y}^{(l,n)}(x)).$$

Again we can approximate the conditional probabilities $P(\,\cdot\, + y|\, y; t)$ with $P^{(n)}_y(\,\cdot\,; t)$. The resulting approximated equation for $P(y, t)$ is:

$$\frac{dP(y)}{dt} \approx \sum_{l\in\mathcal{L}} \sum_{x\in\partial\mathcal{T}_0^{(l,n)}} f_l(x+y)P^{(n)}_y(x;t)P(y;t)+$$
$$+\sum_{l\in\mathcal{L}} \sum_{x\in\partial\mathcal{T}_0^{(l,n)}} f_l(x+y-\mathcal{Y}^{(l,n)}(x))P^{(n)}_{y-\mathcal{Y}^{(l,n)}(x)}(x;t)P(y-\mathcal{Y}^{(l,n)}(x);t). \tag{2.11}$$

By doing this, we are approximating the non-Markovian process $Y(t)$ as a time-inhomogeneous Markov process $Y^{(n)}$. Indeed, while the (exact) conditional probabilities $P(\,\cdot\, + y|\, y; t)$ depend on $X(t)$ and cannot be computed knowing only the state of $Y(t)$ at a previous time instant, the approximated probabilities $P^{(n)}_y(\,\cdot\,; t)$ depend on the value of $Y^{(n)}(t)$ alone. In light of this, (2.11) can be interpreted as the Master Equation of $Y^{(n)}$, in which the first summand describes the outgoing probability flow from a state y and the second summand represents the incoming probability flow to state y from states $y - \mathcal{Y}^{(l,n)}(x)$. In particular, $Y^{(n)}(t)$ is a Markov population process with jump vectors $\mathcal{Y}^{(l,n)}(x)$ and transition functions $f_l(x+y)P^{(n)}_y(x;t)$ for all $l \in \mathcal{L}$ and $x \in \partial\mathcal{T}_0^{(l,n)}$, where $P^{(n)}_y(x;t)$ is a function of y, since it is the solution of system (2.8), with transition matrix $Q^{(n)}(y)$. Observe that the transition functions are time-dependent since $P^{(n)}_y(\,\cdot\,; t)$ is. Thus we can write the mean-field approximation of $Y^{(n)}$, which gives:

$$\frac{dY^{(n)}}{dt} = \sum_{l\in\mathcal{L}} \sum_{x\in\partial\mathcal{T}_0^{(l,n)}} \mathcal{Y}^{(l,n)}(x)f_l(x+Y^{(n)}(t))P^{(n)}_{Y^{(n)}}(x;t) \tag{2.12}$$

We can now approximate the distribution $P(y; t)$ appearing in (2.9) with a delta distribution peaked on the state $Y^{(n)}$ solution of (2.12). The result is the following approximation of the mean:

$$\mathbb{E}[X(t)] \approx \sum_{x\in\mathcal{T}_0^{(n)}} (x+Y^{(n)}(t))P^{(n)}_{Y^{(n)}}(x;t). \tag{2.13}$$

The quantities appearing in this approximation are governed by the following set of equations, which we call the *dynamic boundary projection*

(DBP) of $X(t)$:

$$\frac{dY^{(n)}}{dt} = \sum_{l\in\mathcal{L}} \sum_{x\in\partial\mathcal{T}_0^{(l,n)}} \mathcal{Y}^{(l,n)}(x) f_l(x+Y^{(n)}(t))P^{(n)}(x;t) \tag{2.14}$$

$$\frac{dP^{(n)}}{dt} = Q^{(n)}(Y^{(n)})P^{(n)}(\,\cdot\,,t). \tag{2.15}$$

Observe that we removed the subscript $Y^{(n)}$ from $P^{(n)}$ since from now on we will always consider a single truncated Master Equation with time-varying transition rate matrix $Q^{(n)}(Y^{(n)})$.

The initial conditions for the DBP equations are given by those of the joint process $(X(t), Y(t))$, which have been defined as $(X(0), Y(0)) = (\bar{x}_0, \Pi^{(n)}(0, \bar{x}_0)) = (\bar{x}_0, \bar{y}_0)$. Since $\bar{x}_0 \in \mathcal{T}_{\bar{y}_0}^{(n)}$ we can rewrite $\bar{x}_0$ as $\bar{x}_0^* + \bar{y}_0$ with $\bar{x}_0^* \in \mathcal{T}_0^{(n)}$. Then we set the initial conditions for (2.14)-(2.15) as:

$$Y^{(n)}(0) = \bar{y}_0 \qquad\qquad P^{(n)}(x;0) = \begin{cases} 1 & \text{if } x = \bar{x}_0^*, \\ 0 & \text{else.} \end{cases} \tag{2.16}$$

The DBP equations can be interpreted in the following way. We are considering all possible augmented truncation approximations of X on the sets $(\mathcal{T}_y^{(n)} : y \in \mathcal{S})$. For each of these approximations, (2.15) represents the associated Master Equation, which depends on the considered subset through the parameter $Y^{(n)}$. The value of such parameter evolves according to (2.14) and can be seen as a continuous approximation of the "most probable" truncation on which X is evolving at time t. Observe that due to the mean-field approximation, we are now considering continuous values of $Y^{(n)}$. Moreover we can now interpret the r.h.s. of (2.13) as if $P^{(n)}(x;t)$ gives the probability at time t of a state $x + Y^{(n)}(t)$.

Example. *For the M/M/k queue, applying DBP gives the following:*

$$\frac{dP^{(n)}(x)}{dt} = \begin{cases} -\lambda P^{(n)}(0;t) + \mu\min(1+Y^{(n)}(t),k)P^{(n)}(1;t) & \textit{if } x = 0 \\ -\mu\min(n+Y^{(n)}(t),k)P^{(n)}(n;t) + \\ \quad +\lambda P^{(n)}(n-1;t) & \textit{if } x = n \\ -(\lambda + \mu\min(x+Y^{(n)}(t),k))P^{(n)}(x;t) + \\ \quad +\lambda P^{(n)}(x-1;t) + \\ \qquad +\mu\min(x+1+Y^{(n)}(t),k)P^{(n)}(x+1;t) & \textit{else} \end{cases}$$

$$\frac{dY^{(n)}}{dt} = -\mu\min(Y^{(n)}(t),k)P^{(n)}(0;t) + \lambda P^{(n)}(n;t).$$

The approximate mean evolution of X is thus given by:

$$\mathbb{E}[X(t)] \approx Y^{(n)}(t) + \sum_{i=0}^{n} iP^{(n)}(i;t).$$

□

To highlight the relation between DBP, the mean-field approximation of $X(t)$, and its full Master Equation for the transient probability distribution, we consider the two degenerate cases associated with $n = 0$ and $n = \infty$. According to the definition in (2.2), these correspond to a single-state truncation and an infinite truncation covering the whole state space, respectively.

Proposition 3. *The following two propositions hold:*

- *For $n = 0$, the DBP equations* (2.14)-(2.15) *yield the mean-field approximation of $X(t)$.*
- *For $n = \infty$, the DBP equations* (2.14)-(2.15) *yield the Master Equation of $X(t)$.*

Proof. Let us start from the case $n = 0$. For $n = 0$ we have $\mathcal{T}_0^{(n)} = \{0\}$ so every truncation has a single state. By definition, the generator of a process with a single state is 0, so in (2.15) we have:

$$\frac{dP^0}{dt} = 0 \Rightarrow P^0(t) = P^0(0) = 1 \quad \forall t.$$

Substituting this in (2.14):

$$\frac{dY^0}{dt} = \sum_{l\in\mathcal{L}} \mathcal{Y}^{(l,0)}(0) f_l(Y^0(t)) = \sum_{l\in\mathcal{L}} l f_l(Y^0(t)).$$

Let us now consider $n = \infty$. For $n = \infty$ we have $\mathcal{T}_0^{\infty} = \mathcal{S}$ so the truncation covers the whole state space. Since there is no state outside the considered truncation, we do not need any augmentation and (2.15) for $n = \infty$ is exactly the Master Equation. Moreover, $\partial\mathcal{T}_0^{(l,\infty)} = \emptyset$ for any l, since no transitions exit the observed state so we have:

$$\frac{dY^{\infty}}{dt} = 0 \Rightarrow Y^{\infty}(t) = Y^{\infty}(0) = 0 \quad \forall t.$$

□

Observe that, in particular, the second result suggests the convergence for the solutions of the DBP equations proved in the next section.

2.3 Convergence

Consider the DBP equations in (2.14)-(2.15) with initial condition as in (2.16). We denote the solution of the associated Cauchy problem with the $(m+\mathcal{N}(n))$-dimensional vector $Z^{(n)} = [Y^{(n)}, P^{(n)}]^T$. Recall that by Proposition 3 we know that $Z^{\infty} = [0, P^{\infty}]^T$ is the solution to the Master Equation of the original process $X(t)$.

For a fixed $T > 0$ and for every $n \in \mathbb{N}^m$, $Z^{(n)}$ and Z^{∞} can be immersed in the Banach space $\mathcal{C} = (C^0([0,T], \mathbb{R}^m \times [0,1]^{|\mathcal{S}|}), \|\cdot\|_{\infty})$, where we assume that $\mathbb{R}^m \times [0,1]^{|\mathcal{S}|}$ is equipped with the 1-norm. For each solution, $Y^{(n)}$ (resp., Y^{∞}) denotes the continuous component, evolving in $\mathbb{R}^m$, while $P^{(n)}$ (resp., P^{∞}) denotes a probability distribution over $\mathcal{S}$. Observe that for a fixed n if $x \notin \mathcal{T}_0^{(n)}$ then $P^{(n)}(x;t) = 0\,\forall\, t \in [0,T]$.

2.3.1 Decomposition as a Perturbed Dynamical System

Before proving our result of convergence, we rewrite system (2.14)-(2.15) suitably. We extend the matrix $Q^{(n)}(y)$ to a new matrix $A^{(n)}(y)$ so that (2.14)-(2.15) can be written in matrix form. To do so we define the $(m + \mathcal{N}(n)) \times (m + \mathcal{N}(n))$ block-matrix:

$$A^{(n)}(y) = \left[\begin{array}{c|c} \underline{0} & R^{(n)}(y) \\ \hline \underline{0} & Q^{(n)}(y) \end{array}\right] \tag{2.17}$$

where the upper-left block has dimensions $m \times m$ and $R^{(n)}(y)$ is defined component-wise for $i = 1, \dots, m$ and $x \in \mathcal{T}_0^{(n)}$ as:

$$[R^{(n)}(y)]_{i,x} = \sum_{l \in \mathcal{L}(x)} \mathbb{I}_{\{x \in \partial \mathcal{T}_0^{(l,n)}\}} \mathcal{Y}_i^{(l,n)}(x) f_l(x_0 + y).$$

Using this matrix, we can rewrite the DBP equations (2.14)-(2.15) as:

$$\frac{dZ^{(n)}}{dt} = A^{(n)}(Y^{(n)}(t))Z^{(n)}(t). \tag{2.18}$$

Our proof relies on the decomposition of this system into two parts: a linear one and a non-linear one acting as a perturbation. To do so, in the previous equation we rewrite $A^{(n)}(Y^{(n)})$ as $A^{(n)}(0) + \Delta A^{(n)}(Y^{(n)})$ where we have set $\Delta A^{(n)}(Y^{(n)}) = A^{(n)}(Y^{(n)}) - A^{(n)}(0)$. Then, (2.18) takes the form:

$$\frac{dZ^{(n)}}{dt} = (A^{(n)}(0) + \Delta A^{(n)}(Y^{(n)}(t))Z^{(n)}(t) \tag{2.19}$$

We will refer to this system as the *non-linear perturbed system*. Instead, we define the *linear non-perturbed system* given by:

$$\frac{dZ_\Lambda^{(n)}}{dt} = A^{(n)}(0)Z_\Lambda^{(n)}(t) \tag{2.20}$$

whose solution we denote by $Z_\Lambda^{(n)} = [Y_\Lambda^{(n)}, P_\Lambda^{(n)}]^T \in \mathcal{C}$.

Proof Strategy. With this decomposition, we first prove convergence of the linear part in Section 2.3.2. Then, the convergence result for the full system will be proved in Section 2.3.3. The proofs will be based on the observation that the family of distributions $(P^\infty(\,\cdot\,;t) : t \in [0,T])$ is tight in $\mathcal{P}(\mathcal{S})$ (Ethier and Kurtz, 2009a), due to the fact that the function $P^\infty(\,\cdot\,;t)$ is continuous in t and we are considering a compact interval $[0,T]$, so we have that

$$\forall\,\epsilon > 0 \quad \exists\, K \subseteq \mathcal{S} \text{ compact s.t. } \sup_{t\in[0,T]} \sum_{x\notin K} P^\infty(x;t) \leq \epsilon.$$

This property can be rephrased by considering the states outside a truncation $\mathcal{T}_0^{(n)}$ and saying that there exists a non-negative sequence $C^{(n)}$ such that:

$$\sup_{t\in[0,T]} \sum_{x\notin\mathcal{T}_0^{(n)}} P^\infty(x;t) \leq C^{(n)} \xrightarrow{n\to\infty} 0$$

In particular, we can restrict the previous sum to states on the border of a truncation (that can be viewed as states outside a suitable smaller truncation). As a result, we can express the tightness property in the following form:

$$\sup_{t\in[0,T]} \sum_{x\in\partial\mathcal{T}_0^{(n)}} P^\infty(x;t) \leq C^{(n)} \xrightarrow{n\to\infty} 0. \tag{2.21}$$

In the rest of this section the proofs are presented for $\mathcal{S} = \mathbb{N}^m$ since the general case $\mathcal{S} \subset \mathbb{N}^m$ can be easily derived from it. In the latter case the limit behavior as $n \to \infty$ is intended for $n \in \mathbb{N}^m$ and $n_i \to \sup\{x_i : x \in \mathcal{S}\}$ for all i.

2.3.2 Convergence of the Linear Non-Perturbed System

Theorem 3. *Suppose that hypothesis (H2) holds. Then $Z_\Lambda^{(n)}$ solution of the linear non-perturbed system* (2.20) *is such that:*

$$\lim_{n\to\infty} \|Z_\Lambda^{(n)} - Z^\infty\|_\infty = 0$$

and, in particular, for some non-negative real sequence $\{\lambda^{(n)}\}_{n\in\mathbb{N}^m}$:

$$\lim_{n\to\infty} \sup_{t\in[0,T]} \|P_\Lambda^{(n)}(\,\cdot\,;t) - P^\infty(\,\cdot\,;t)\|_1 = 0, \tag{2.22}$$

$$\lim_{n\to\infty} \sup_{t\in[0,T]} \|Y_\Lambda^{(n)}(t)\|_1 \leq \lim_{n\to\infty} \lambda^{(n)} = 0. \tag{2.23}$$

We start by proving (2.22). Using (2.7) with $y = 0$, we write explicitly the equations for $P_\Lambda^{(n)}(x,t)$:

$$\begin{aligned}\frac{dP_\Lambda^{(n)}(x)}{dt} &= -\sum_{l\in\mathcal{L}} \mathbb{I}_{\{\tilde{l}^{(n)}(x)\neq 0\}} f_l(x) P_\Lambda^{(n)}(x;t) + \\ &\quad + \sum_{\substack{x'\in\mathcal{T}_0^{(n)} \\ x'\neq x}} \sum_{l\in\mathcal{L}} \mathbb{I}_{\{x'+\tilde{l}^{(n)}(x')=x\}} f_l(x') P_\Lambda^{(n)}(x';t).\end{aligned} \tag{2.24}$$

Observe that the dynamics of $P_\Lambda^{(n)}$ do not depend on $Y_\Lambda^{(n)}$ so we can solve these components independently and then plug the solution in the equations for $Y_\Lambda^{(n)}$.

We introduce a new function $\tilde{P}^{(n)}(t) = (\tilde{P}^{(n)}(x_\infty;t), \tilde{P}^{(n)}(x;t) : x \in \mathcal{T}_0^{(n)})$ whose evolution is given by the system of ODEs:

$$\frac{d\tilde{P}^{(n)}(x)}{dt} = -\sum_{l\in\mathcal{L}} f_l(x)\tilde{P}^{(n)}(x;t) + \sum_{x'\in\mathcal{T}_0^{(n)}} \sum_{l\in\mathcal{L}} \mathbb{I}_{\{x'+l=x\}} f_l(x')\tilde{P}^{(n)}(x';t) \tag{2.25}$$

$$\frac{d\tilde{P}^{(n)}(x_\infty)}{dt} = \sum_{l\in\mathcal{L}} \sum_{x\in\partial\mathcal{T}_0^{(l,n)}} f_l(x)\tilde{P}^{(n)}(x;t) \tag{2.26}$$

with initial conditions $\tilde{P}^{(n)}(x;0) = P^{(n)}(x;0)$ and $\tilde{P}^{(n)}(x_\infty;0) = 0$.

Comparing (2.24) with (2.25)-(2.26) one can conclude that:

$$\tilde{P}^{(n)}(x;t) \le P_\Lambda^{(n)}(x;t) \le \tilde{P}^{(n)}(x;t) + \tilde{P}^{(n)}(x_\infty;t) \quad \forall\, t \in [0,T],\, \forall\, x \in \mathcal{T}_0^{(n)}.$$

Therefore, (2.22) will be proved if we prove that the following two limits hold:

$$\lim_{n\to\infty} \sup_{t\in[0,T]} \sum_{x\in\mathbb{N}^m} |\tilde{P}^{(n)}(x;t) - P^\infty(x;t)| = 0, \tag{2.27}$$

$$\lim_{n\to\infty} \sup_{t\in[0,T]} \tilde{P}^{(n)}(x_\infty;t) = 0. \tag{2.28}$$

Let us start by proving the first limit. We have:

$$\frac{d(\tilde{P}^{(n)}(x) - P^\infty(x))}{dt} = -\sum_{l\in\mathcal{L}} (\tilde{P}^{(n)}(x;t) - P^\infty(x;t)) f_l(x) +$$
$$+ \sum_{l\in\mathcal{L}} \mathbb{I}_{\{x-l\in\mathcal{T}_0^{(n)}\}} (\tilde{P}^{(n)}(x-l;t) - P^\infty(x-l;t)) f_l(x-l) +$$
$$+ \sum_{l\in\mathcal{L}} \mathbb{I}_{\{x-l\notin\mathcal{T}_0^{(n)}\}} P^\infty(x-l;t) f_l(x-l).$$

Integrating between 0 and t gives:

$$\tilde{P}^{(n)}(x;t) - P^\infty(x;t) = -\int_0^t \sum_{l\in\mathcal{L}} (\tilde{P}^{(n)}(x;s) - P^\infty(x;s)) f_l(x)\, ds +$$
$$+ \int_0^t \sum_{l\in\mathcal{L}} \mathbb{I}_{\{x-l\in\mathcal{T}_0^{(n)}\}} (\tilde{P}^{(n)}(x-l;s) - P^\infty(x-l;s)) f_l(x-l)\, ds +$$
$$+ \int_0^t \sum_{l\in\mathcal{L}} \mathbb{I}_{\{x-l\notin\mathcal{T}_0^{(n)}\}} P^\infty(x-l;s) f_l(x-l)\, ds.$$

We consider the absolute value and sum over $x \in \mathcal{T}_0^{(n)}$, to obtain:

$$\sum_{x\in\mathcal{T}_0^{(n)}} |\tilde{P}^{(n)}(x;t) - P^\infty(x;t)| \le$$
$$\le 2\int_0^t \sum_{l\in\mathcal{L}} \sum_{x\in\mathcal{T}_0^{(n)}} |\tilde{P}^{(n)}(x;s) - P^\infty(x;s)| f_l(x)\, ds +$$
$$+ \int_0^t \sum_{l\in\mathcal{L}} \sum_{x\in\partial\mathcal{T}_0^{(l,n)}} P^\infty(x-l;s) f_l(x-l)\, ds.$$

Now, applying (H2) and tightness of P^∞ (2.21):

$$\sum_{x\in\mathcal{T}_0^{(n)}} |\tilde{P}^{(n)}(x;t) - P^\infty(x;t)| \leq$$
$$\leq 2|\mathcal{L}|C\int_0^t |\tilde{P}^{(n)}(x;s) - P^\infty(x;s)|\,ds + |\mathcal{L}|TCC^{(n)}.$$

Applying Gronwall's inequality (Ethier and Kurtz, 2009a) to $\sum_{x\in\mathcal{T}_0^{(n)}} |\tilde{P}^{(n)}(x;t) - P^\infty(x;t)|$:

$$\sup_{t\in[0,T]} \sum_{x\in\mathcal{T}_0^{(n)}} |\tilde{P}^{(n)}(x;t) - P^\infty(x;t)| \leq e^{2T|\mathcal{L}|C}|\mathcal{L}|TCC^{(n)}.$$

Finally, we can apply the tightness property of P^∞ again to conclude that:

$$\sup_{t\in[0,T]}\sup_{x\in\mathbb{N}^m} |\tilde{P}^{(n)}(x;t) - P^\infty(x;t)| \leq$$
$$\leq \sup_{t\in[0,T]}\sup_{x\in\mathcal{T}_0^{(n)}} |\tilde{P}^{(n)}(x;t) - P^\infty(x;t)| + \sup_{t\in[0,T]}\sum_{x\notin\mathcal{T}_0^{(n)}} P^\infty(x;t) \leq$$
$$\leq e^{2T|\mathcal{L}|C}|\mathcal{L}|TCC^{(n)} + C^{(n)} \to 0.$$

We have thus proved that (2.27) holds.

Limit (2.28) follows easily by observing that the previous proof implies that tightness property holds also for the distribution $\tilde{P}^{(n)}$ (possibly for a different sequence $C^{(n)}$). So we have:

$$\frac{d\tilde{P}^{(n)}(x_\infty)}{dt} = \sum_{l\in\mathcal{L}}\sum_{x\in\partial\mathcal{T}_0^{(l,n)}} \tilde{P}^{(n)}(x;t)f_l(x) \leq |\mathcal{L}|CC^{(n)} \to 0.$$

This concludes the proof of (2.22).

We now prove (2.23). In virtue of the uniform convergence of $P_\Lambda^{(n)}$ to P^∞ we can apply the tightness property also to this distribution (possibly considering a different sequence $C^{(n)}$). We have:

$$\left\|\frac{dY_\Lambda^{(n)}}{dt}\right\|_1 = \left\|\sum_{l\in\mathcal{L}}\sum_{x\in\partial\mathcal{T}_0^{(l,n)}} \mathcal{Y}^{(l,n)}(x)f_l(x)P_\Lambda^{(n)}(x;t)\right\|_1 \leq m\bar{l}|\mathcal{L}|CC^{(n)} \to 0.$$

wher $\bar{l} = \max_{l \in \mathcal{L}} \max_{i=1,\dots,m} |l_i|$.

This concludes the proof of Theorem 3.

2.3.3 Convergence Result

We now proceed to prove the convergence of the non-linear perturbed system by exploiting the results already proved for the linear counterpart. Our main result is stated in the following theorem.

Theorem 4. *Suppose that hypotheses (H1) and (H2) hold. Then $Z^{(n)}$ solution of the non-linear perturbed system* (2.19) *satisfies:*

$$\lim_{n \to \infty} \|Z^{(n)} - Z^{\infty}\|_{\infty} = 0$$

and, in particular

$$\lim_{n \to \infty} \sup_{t \in [0,T]} \|P^{(n)}(\,\cdot\,;t) - P^{\infty}(\,\cdot\,;t)\|_1 = 0, \tag{2.29}$$

$$\lim_{n \to \infty} \sup_{t \in [0,T]} \|Y^{(n)}(t)\|_1 = 0. \tag{2.30}$$

We start by proving the following proposition.

Proposition 4. *Hypothesis (H1) implies that there exists a constant $K > 0$ such that:*

$$\|\Delta A^{(n)}(y)\|_1 \leq K \|y\|_1 \quad \forall\, y \in \mathbb{R}^m_{\geq 0}, \forall\, n \in \mathbb{N}^m$$

Proof. Using (2.17) we can estimate the 1-norm of $\Delta A^{(n)}(y)$ as:

$$\begin{aligned} \|\Delta A^{(y)}(y)\|_1 = \max_{j \in \{1,\dots,m\} \cup \mathcal{T}_0^{(n)}} \sum_{i \in \{1,\dots,m\} \cup \mathcal{T}_0^{(n)}} |[\Delta A^{(n)}((y))]_{i,j}| \leq \\ \leq (m\bar{v} + 2) \max_{x \in \mathcal{T}_0^{(n)}} \sum_{l \in \mathcal{L}} |f_l(y + x) - f_l(x)| \end{aligned}$$

where the last inequality follows from the fact that, for a fixed column indexed by $x \in \mathcal{T}_0^{(n)}$, we have that the term $m\bar{v} \sum_{l \in \mathcal{L}} |f_l(y + x) - f_l(x)|$ takes into account the components of the submatrix $R^{(n)}(y)$ and the term $2 \sum_{l \in \mathcal{L}} |f_l(y + x) - f_l(x)|$ takes into accounts the components in the submatrix $Q^{(n)}(y)$ (observe that at most $|\mathcal{L}|$ of the non-diagonal entries of a given column are non-zero since they correspond to the reactions starting in the observed state indexing that column).

Applying (H1) to the r.h.s. of the previous inequality our assertion follows immediately. $\square$

It is possible to verify by direct derivation that $Z^{(n)}(t)$ can be expressed as:

$$Z^{(n)}(t) = Z^{(n)}_{\Lambda}(t) + \int_0^t e^{(t-s)A^{(n)}(0)} \Delta A^{(n)}(Y^{(n)}(s)) Z^{(n)}(s)\, ds. \tag{2.31}$$

The operator $e^{tA^{(n)}(0)}$ is uniformly bounded under our current hypotheses, i.e., for a constant $M > 0$ we have

$$\|e^{tA^{(n)}(0)}\| \leq M \quad \forall\, t \in [0,T], \forall n \in \mathbb{N}^m.$$

This can be proved using an argument similar to that used in the proof of Proposition 4 so that we have:

$$\begin{aligned}\|A^{(n)}(0)\|_1 = \max_{j\in\{1,\ldots,m\}\cup\mathcal{T}_0^{(n)}} \sum_{i\in\{1,\ldots,m\}\cup\mathcal{T}_0^{(n)}} |[A^{(n)}(0)]_{i,j}| \leq \\ \leq m\bar{l}|\mathcal{L}|C + |\mathcal{L}|C + |\mathcal{L}|C \leq (m\bar{l}+2)|\mathcal{L}|C.\end{aligned}$$

Let us now prove (2.30). Let us fix a constant $H \gg 2$ and define:

$$\Delta_1^{(n)} = \sup\left\{\delta \geq 0 : \sup_{s\in[0,T]} \|Z^{(n)}(s)\|_1 \leq H\right\}.$$

For n sufficiently large, we have that $\|Z^{(n)}(0)\|_1 = 1$ so $\Delta_1^{(n)} > 0$. Suppose $\Delta_1^{(n)} < +\infty$, then, by continuity, $\|Z^{(n)}(\Delta_1^{(n)})\|_1 = H$. Let n sufficiently large be fixed and consider $0 \leq t \leq \Delta_1^{(n)}$. From (2.31), we have:

$$\|Y^{(n)}(t)\|_1 \leq \|Y^{(n)}_{\Lambda}(t)\|_1 + tMKH \sup_{s\in[0,t]} \|Y^{(n)}(s)\|_1.$$

Now, suppose that $\liminf_n \Delta_1^{(n)} \in \left[0, \frac{1}{4MKH}\right]$, then for a suitable subsequence n_k we have $\Delta_1^{(n_k)} < \frac{1}{2MKH} \quad \forall\, k$. Then

$$\sup_{t\in[0,\Delta_1^{(n_k)}]} \|Y^{(n_k)}(t)\|_1 \leq \lambda^{(n_k)} + \frac{1}{2} \sup_{s\in[0,\Delta_1^{(n_k)}]} \|Y^{(n)}(s)\|_1$$

and

$$\sup_{t\in[0,\Delta_1^{(n_k)}]} \|Y^{(n_k)}(t)\|_1 \leq 2\lambda^{(n_k)} \xrightarrow{n\to\infty} 0$$

which contradicts the continuity hypothesis $\|Z^{(n)}(\Delta_1^{(n)})\|_1 = H \gg 2$, since this implies $\|Y^{(n)}(\Delta_1^{(n)})\|_1 > H - 1 \gg 1$. We have thus proved that $\lim_n \inf \Delta_1^{(n)} > \frac{1}{4MKH}$.

We can then set $t_1^* = \frac{1}{4MKH}$, for which we have, provided n is sufficiently large:

$$\sup_{t\in[0,t_1^*]} \|Y^{(n)}(t)\|_1 \leq \lambda^{(n)} + \frac{1}{4} \sup_{t\in[0,t_1^*]} \|Y^{(n)}(s)\|_1$$

and

$$\sup_{t\in[0,t_1^*]} \|Y^{(n)}(t)\|_1 \leq 2\lambda^{(n)} \xrightarrow{n\to\infty} 0.$$

Now, if $t_1^* \geq T$ we have completed our proof; if not, we need to iterate the argument to extend the interval. To do this, we define:

$$\Delta_2^{(n)} = \sup\left\{\delta \geq 0 : \sup_{s\in[t_1^*, t_1^*+\delta]} \|Z^{(n)}(s)\|_1 \leq H\right\}.$$

Observe that because of the previous step, for n sufficiently large we have $\|Z^{(n)}(t_1^*)\|_1 \leq 2\lambda^{(n)} \ll H$ so, for such values of n, $\Delta_2^{(n)} > 0$. Moreover, by continuity, if $\Delta_2^{(n)} < \infty$, $\|Z^{(n)}(t_1^* + \Delta_2^{(n)})\|_1 = H$. Now for $t_1^* \leq t \leq t_1^* + \Delta_2^{(n)}$ and for n sufficiently large:

$$\begin{aligned}\|Y^{(n)}(t)\|_1 \leq \|Y_\Lambda^{(n)}(t)\|_1 + t_1^* MKH \sup_{s\in[0,t_1^*]} \|Y^{(n)}(s)\|_1 + \\ + (t - t_1^*)MKH \sup_{s\in[t_1^*,t]} \|Y^{(n)}(s)\|_1 \leq \\ \leq \|Y_\Lambda^{(n)}(t)\|_1 + \frac{1}{2}2\lambda^{(n)} + (t-t_1^*)MKH \sup_{s\in[t_1^*,t]} \|Y^{(n)}(s)\|_1.\end{aligned}$$

Using an argument similar to the one used before we can prove that $\liminf_n \Delta_2^{(n)} > \frac{1}{4MKH}$, and set $t_2^* = t_1^* + \frac{1}{4MKH} = 2t_1^*$ so that

$$\sup_{t\in[0,t_2^*]} \|Y^{(n)}(t)\|_1 \leq 4\lambda^{(n)}.$$

If $t_2^* \geq T$ we can conclude; if not, we can reiterate the procedure as many times as needed. Since each time the interval is extended by a constant quantity, the whole $[0, T]$ interval is covered by a finite number h of iterations, and we get:

$$\sup_{t \in [0,T]} \|Y^{(n)}(t)\|_1 \leq 2^h \lambda^{(n)} \to 0.$$

Let us now prove (2.29). This follows easily by observing that from (2.31), and from and the convergence of $Y^{(n)}$, we have

$$\begin{aligned}
&\sup_{t \in [0,T]} \|P^{(n)}(\,\cdot\,; t) - P_\Lambda^{(n)}(\,\cdot\,; t)\|_1 \leq \\
&\qquad \leq TKM \sup_{t \in [0,T]} \|Y^{(n)}(t)\|_1 \left(1 + \sup_{t \in [0,T]} \|Y^{(n)}(t)\|_1\right) \\
&\qquad \leq TKM (2^h \lambda^{(n)})(2^h \lambda^{(n)} + 1) \xrightarrow{n \to \infty} 0.
\end{aligned}$$

Since by the previous theorem we have already proved the uniform convergence of $P_\Lambda^{(n)}$ to $P^{(n)}$, this concludes the proof of Theorem 4.

2.4 Examples

This section applies DBP to the three examples of density-dependent Markov population processes with scaling parameter N introduced in Chapter 1. As noted in Remark 1, DBP does not generally require the hypothesis of density-dependence for its applicability, and examples of non density-dependent models could also be presented. The choice of using density-dependent examples is justified by the wide diffusion of this class of models and to ease the comparison with existing refining approaches. For the presented cases, as discussed, the mean-field solution $x(t)$ represents the limit behavior of the re-scaled population process $X^N(t)/N$. Thus, for any given N, we will compare the DBP refinement of the unscaled process $\mathbb{E}[X^N(t)]$ against the mean-field approximation of expected populations given by $Nx(t)$. As ground truth, we take the average trajectory using Gillespie's direct stochastic simulation algorithm (Gillespie, 2007). The stopping criterion was chosen as follows: along the simulated time horizon we fix 100 equidistant time steps, and at each time step we compute the 95% confidence interval. If the semi-amplitude of the interval is greater than 1% of the mean at that step the number of simulations is increased by 1k runs or 10k runs (if the number of simulations has already exceeded 50k). This procedure is applied to all observed outputs.

Stochastic simulation as well as the numerical solutions of the mean-field equations and the DBP equations were performed in Matlab. The DBP equations were generated from a prototype implementation within the (Java-based) software tool ERODE (Cardelli et al., 2017). In the run-time comparisons, we do not report the time taken to generate the DBP Matlab file because we found it to be negligible with respect to the solution time. The expansions from Gast, Bortolussi, and Tribastone, 2019 were computed using the Python-based implementation therein reported. Since the considered implemented computes both expansions simultaneously. All experiments were conducted on a laptop equipped with a 2.8 GHz Intel i7 quad-core processor and 16 GB RAM.

2.4.1 Coxian Queuing Systems

We consider the $M/Cox/N$ queuing system with Poisson arrivals with rate $N\lambda$ and service rate with a Coxian distribution, which was introduced in Example 2. We separate the cases in which the Coxian distribution has two phases or a larger number of phases due to the significant difference in the sizes of the systems.

Two-phase Coxian distribution We first consider as service rate a two-phase Coxian distribution. We set the initial condition $\bar{x}_0 = (0, 0, N)$. As observed, the mean-field approximation has a Lipschitz continuous but non-differentiable drift, for which available results of mean-field refinement in Gast, Bortolussi, and Tribastone, 2019 are not applicable.

Figure 2 compares the transient evolution of average queue lengths for $\lambda = 0.75$ and different variances (using a unitary mean service time as in Table 1) and for various truncations in the form $n = (n_1, N, N)$, i.e., by truncating the number of jobs requiring first-phase service to n_1. In general, the mean-field approximation does not behave satisfactorily and, as mentioned in Section 1, provides estimates for long enough time horizons that are insensitive to the service time distribution. DBP can refine the mean in all cases, although larger values of n_1 are needed to improve the accuracy with larger variances of the service-time distribution.

Table 2 reports the runtimes as well as the size of the resulting system of equations when applying DBP. As expected, the computational cost of the DBP analysis grows with n_1 and is larger than that of the mean-field

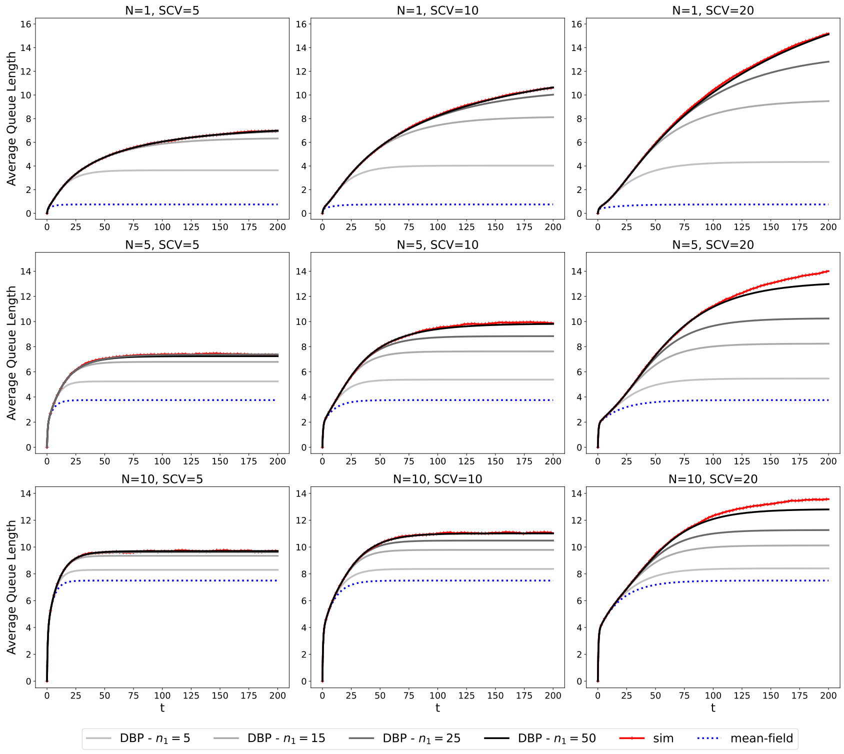


Figure 2: DBP applied to an $M/Cox/N$ queuing system where the truncation is applied to n_1 jobs waiting for service in the first phase of a two-phase Coxian distribution.

Table 2: Runtimes (in s) for simulations (SIM) and DBP with bound (n_1, N, N) for the two-phase Coxian distribution; the number of equations refers to the size of the resulting DBP models. The solution of the mean-field system takes 0.06s on average.

		SIM		*DBP*			
N	V			$n_1 = 10$	$n_1 = 30$	$n_1 = 50$	$n_1 = 100$
		Time	*# runs*	*Time*	*Time*	*Time*	*Time*
				(22 eqs.)	(62 eqs.)	(102 eqs.)	(202 eqs.)
1	5	204	130k	0.07	0.08	0.14	0.19
1	10	224	140k	0.08	0.10	0.14	0.21
1	20	213	140k	0.12	0.22	0.27	0.40
				(66 eqs.)	(186 eqs.)	(306 eqs.)	(606 eqs.)
5	5	574	70k	0.37	0.85	1.32	2.58
5	10	944	100k	0.45	0.90	1.43	2.71
5	20	1331	130k	0.53	0.97	1.56	2.82
				(121 eqs.)	(341 eqs.)	(561 eqs.)	(1111 eqs.)
10	5	487	29k	0.93	2.72	4.37	8.85
10	10	889	60k	0.93	2.90	4.54	9.22
10	20	1622	100k	1.02	3.01	4.53	9.24

solution. However, it can refine the mean estimate at a small fraction (no more than about 2% across all cases) of the time taken to perform stochastic simulation.

Larger Number of Phases We use a Coxian distribution with more phases to show how our method behaves with larger models. Figure 3 shows the results for $K = 5$ and $K = 10$ with $N = 5$. The Coxian parameters are manually tuned so as to have a unitary mean and variance $V = 5$ and are reported in Table 3. For $K = 5$ we used bounds in the form $n = (n_1, 1, 1, 1, 1, 0, 0, 0, 0)$, with $n_1 = 2, 5, 10$; for $K = 10$ we used the bounds $n_1 = 5, 10, 20$, $n_2 = \ldots = n_4 = 1, n_5 = n_{19} = 0$. Runtimes are reported in Table 4. Since we are using modest truncations, even if the computational time is larger than that of the classic mean-field approximation, for each of the chosen bounds, it does not exceed 2.2s. Nonetheless, in Figure 3 it is possible to see that the approximation is consistently improved.

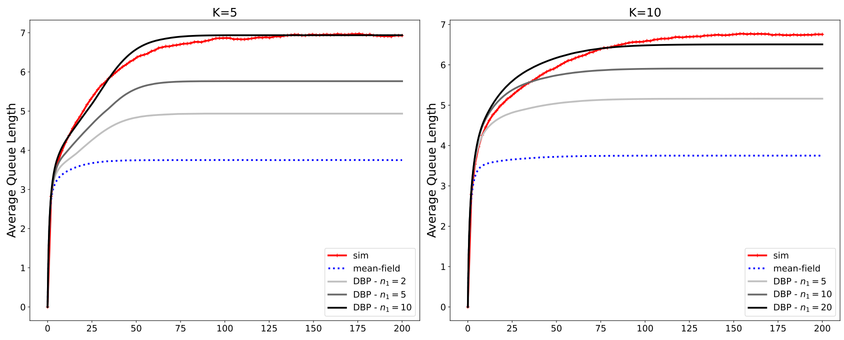


Figure 3: DBP applied to the $M/Cox/N$ queuing systems with $K = 5$ and $K = 10$ phases for the Coxian service-time distribution.

Table 3: Parameters for Coxian service time distribution with a larger number of phases.

K	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8	μ_9	μ_{10}
5	1.5136	0.5045	0.5045	0.1261	0.1261	-	-	-	-	-
10	1.8809	0.6270	0.6270	0.3135	0.3135	0.2821	0.1881	0.1567	0.0784	0.0784
K	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	
5	0.10	0.10	0.85	0.80	-	-	-	-	-	
10	0.20	0.20	0.20	0.50	0.50	0.80	0.80	0.80	0.80	

Table 4: Runtimes (in s) for simulations (SIM), mean-field (MF) and DBP for the $M/Cox/N$ queuing system with $N = 5$ and Coxian distributed service times with $K = 5$ and $K = 10$ phases, unitary mean and variance $V = 5$.

K	*Method*	*Parameters*	*Time*
5	*SIM*	80k runs	604
	MF	-	0.22
	DBP	$n_1 = 2$, 57 eqs.	0.41
	DBP	$n_1 = 5$, 105 eqs.	1.04
	DBP	$n_1 = 10$, 185 eqs.	1.95
10	*SIM*	100k runs	953
	MF	-	0.36
	DBP	$n_1 = 5$, 67 eqs.	0.43
	DBP	$n_1 = 10$, 107 eqs.	0.79
	DBP	$n_1 = 20$, 187 eqs.	2.20

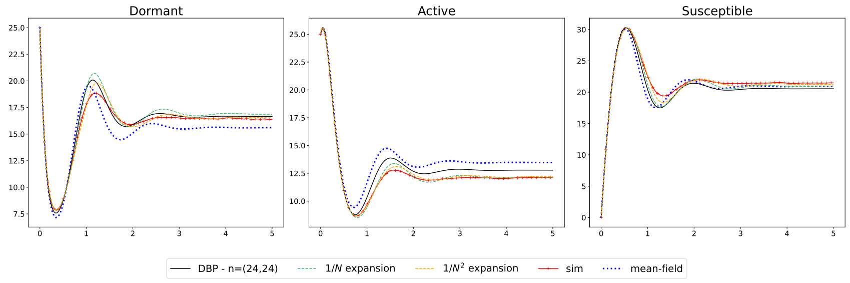


Figure 4: Numerical results for the malware propagation model with unique attractor orbit ($\delta = 0.50$) for mean-field approximation, $1/N$ and $1/N^2$ expansions in Gast, Bortolussi, and Tribastone, 2019, and DBP.

Table 5: Runtimes (in s) for simulations (SIM), mean-field (MP), DBP, and expansions in $1/N$ and $1/N^2$ (EXP) for the Malware Propagation Model with $N = 50$ agents.

Model	δ	*SIM*		*MF*	*DBP* $n = (24, 24)$		*EXP* $1/N$	$1/N^2$
		Time	*#runs*	*Time*	*Time*	*#eqs.*	*Time*	*Time*
Stable	0.5	100	21k	0.06	0.83	628	0.07	0.42
Unstable	0.1	2819	380k	0.08	2.17	628	0.06	0.39

2.4.2 Malware Propagation Model

Here we consider the malware propagation model presented in Example 3, in the two cases $\delta > \delta^*$ (stable model) and $\delta < \delta^*$ (unstable model).

Stable Model To show the error behavior in the case of a stable mean-field approximation, we set $N = 50, \delta = 0.5, \beta = 0.1$, and $\bar{x}_0 = (\bar{x}_D, \bar{x}_A) = (25, 25)$, as done in Gast, Bortolussi, and Tribastone, 2019. Figure 4 compares DBP applied with bound $n = (24, 24)$, against the $1/N$ and $1/N^2$ expansion approximations proposed in Gast, Bortolussi, and Tribastone, 2019. We found that these methods all perform comparably with each other in terms of accuracy, improving the mean-field estimation (especially for the average population of dormant nodes), while the $1/N^2$ expansion is faster than DBP (see Table 5).

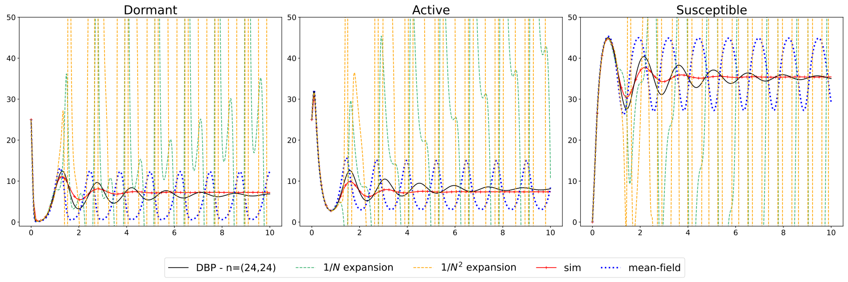


Figure 5: Numerical results for the malware propagation model with orbit cycle ($\delta = 0.10$) for mean-field approximation, $1/N$ and $1/N^2$ expansions in Gast, Bortolussi, and Tribastone, 2019, and DBP.

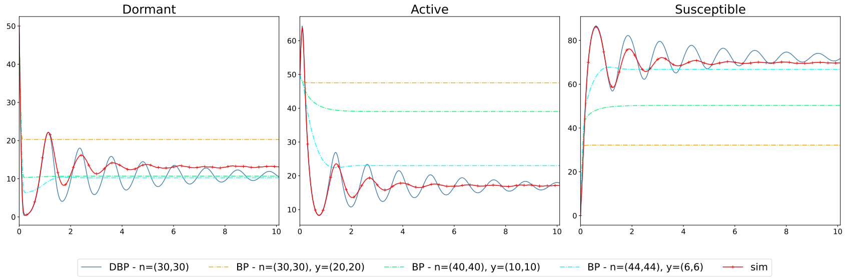


Figure 6: Numerical results for the unstable malware propagation model ($\delta = 0.1$) comparing DBP with bound $n = (15, 15)$ and BP on different truncations $\mathcal{T}_y^{(n)}$.

Unstable Model Using $\delta = 0.1$, Figure 5 shows the presence of an orbit cycle in the mean-field approximation. We recall that an orbit cycle is any solution curve of (1.3) that is not an equilibrium point (Perko, 2013). The presence of such orbit causes instability to both the expansions $1/N$ and $1/N^2$. Instead, using the same n as in the previous case, DBP exhibits damped oscillatory behavior that refines the mean-field approximation, especially over longer time horizons when the oscillations tend to fade away. Runtimes can be found in Table 5.

Impact of Dynamic Shift of Truncation The structure of this model allows us to show the impact of the dynamic shift. To do so, we consider the unstable model with $N = 100, \delta = 0.1, \beta = 0.2$ and compare DBP against different BPs that are not modulated by the mean-field approximation, i.e., solutions to systems in the form of (2.8). Figure 6 shows the results of DBP with $n = (30, 30)$ and BP applied on a truncated state space $\mathcal{T}_y^n$ with different values of n and y (defined as in (2.2)). We use initial condition $\bar{x}_0 = (\bar{x}_D, \bar{x}_A) = (50, 50)$. Observe that with this choice, when applying BP, we are forced to choose a truncation containing the initial state. The results indicate that DBP significantly outperforms BP for the same size of the truncated state space $n = (30, 30)$. Since the Master Equations for the truncated state space have the same functional form in DBP and BP, the difference in behavior is due to the dynamic shift of the truncation by the coupling with the two mean-field equations that approximate the average population of dormant and active nodes. To achieve comparable accuracy with DBP, BP requires much larger state spaces—i.e., using $n = (44, 44)$, corresponding to over a twofold increase of the truncated state space size (2025 states for BP, 961 states for DBP).

2.4.3 Egalitarian Processor Sharing

Finally, we discuss the queuing system with egalitarian processor sharing of Example 4.

We consider models with $K = 2, 3, 4$ classes, always setting the initial condition to $\bar{x}_0 = (0, \ldots, 0)$ and $N = 1$. For $K = 2$ classes, we first show how the choice of the bound n can impact the accuracy of the approximation. In fact, when the arrival rates λ_i are different, we expect that the probability mass in each truncation will be concentrated on states with $x_{Q_i} > x_{Q_j}$ for $\lambda_i > \lambda_j$; specifically, we expect that the ratio between the mean number of class-i customers and the mean number of

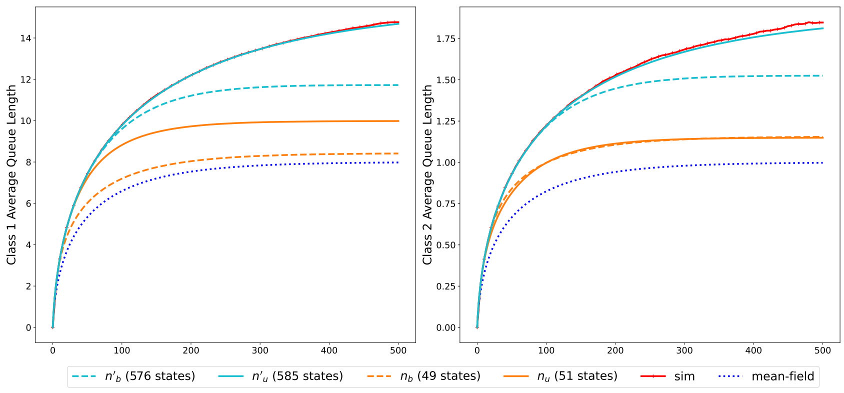


Figure 7: Numerical results for the egalitarian processor sharing queuing system with $K = 2$ classes using different DBP truncation parameters.

class-j customers is approximately equal to $\frac{\lambda_i}{\lambda_j}$. This suggests a heuristic of *unbalanced* bounds where each dimension is set proportionally to the arrival rate of each class. We will show that applying this heuristic, instead of considering balanced bounds of the form $n_i = n_j$ for any i and j, improves the approximation. In particular, we will show a direct comparison for $K = 2$ and then apply this heuristic also for $K = 3$ and $K = 4$. We recall that in this case the number of states used by DBP is $\mathcal{N}(n) = \prod_{k=1}^{K}(n_k + 1)$ where n_k is the bound chosen for class k, while the number of equation is given by $\mathcal{N}(n) + K$.

Case $K = 2$ To show the impact of the choice of n, we take two pairs of vectors. The pair $n_b = (6, 6)$ and $n'_b = (23, 23)$ considers two truncations of increasing size which are balanced, i.e., $n_{Q_1} = n_{Q_2}$; the pair $n_u = (16, 2)$ and $n'_u = (64, 8)$ considers two truncations where $n_{Q_1} = 8n_{Q_2}$. These values have been chosen such that the number of states in the truncated state spaces for n_b and n_u, as well as those for n'_b and n'_u, are similar. This way, the difference in the accuracy may be more directly related to the choice of the truncation. Figure 7 shows the results of the approximations for the average queue lengths for both classes of customers. They indicate that using a balanced truncation leads to less accurate approximations than an unbalanced truncation with a compara-

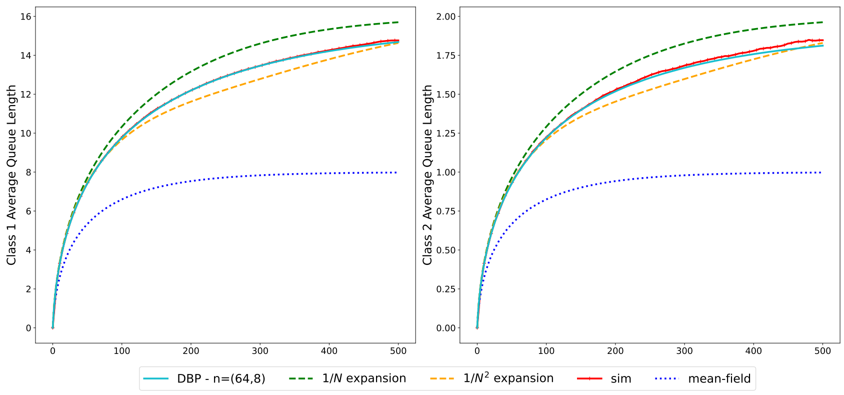


Figure 8: Comparison between DBP with $n = (32, 4)$ and the $1/N$, $1/N^2$ expansions from Gast, Bortolussi, and Tribastone, 2019 for the queuing system with egalitarian process sharing presented in Section 2.4.3.

ble number of states. Nevertheless, every DBP approximation improves the mean-field estimates in all cases considered here.

Using N as scaling parameter, this model admits a mean-field limit with a differentiable drift; therefore, applying the $1/N$ and $1/N^2$ refinements proposed in Gast, Bortolussi, and Tribastone, 2019 is possible. Figure 8 compares the refinements against the most accurate DBP approximation shown in Fig. 7, i.e., $n'_u = (64, 8)$. With this choice of bound, DBP is superior to both refinements across the transient regime. As the trajectories approach stationarity, the $1/N^2$ refinement and DBP accuracy become comparable. The runtime comparison, reported in Table 6, shows that the $1/N^2$ expansion requires less computational time (for $K = 2$ as well as in the forthcoming cases).

Case $K = 3, 4$ For $K = 3$ we chose $\lambda_1 = 0.8, \lambda_2 = 0.4, \lambda_3 = 0.1, \mu = 1.5$. Following the same argument applied for $K = 2$, we chose the bounds for Q_1, Q_2, Q_3 proportionally, so we set $n = (16, 8, 2), (24, 12, 4), (32, 16, 4)$ and $(36, 18, 5)$. For $K = 4$ and rates constants $\lambda_1 = 0.8, \lambda_2 = 0.4, \lambda_3 = 0.2, \lambda_4 = 0.1, \mu = 2$ we chose $n = (8, 4, 2, 1)$, $(12, 6, 3, 2)$, $(16, 8, 4, 2)$ and $(20, 10, 5, 3)$. Results for the average queue length are shown in Figure 9 while the results for per-class queue lengths are reported in Figure 10 and 11. In these cases, DBP performs comparably to the $1/N^2$ expansion,

Table 6: Runtimes (in s) for simulations (SIM), mean-field (MF), DBP and $1/N^2$ expansion (EXP) for the egalitarian processor sharing queuing system with K = 2,3, and 4 classes of customers.

K	*Method*	*Parameters*	*Time*
2	*SIM*	150k runs	924
	MF	-	0.07
	DBP	$n = (6, 6)$, 51 eqs.	0.10
	DBP	$n = (16, 2)$, 53 eqs.	0.08
	DBP	$n = (23, 23)$, 578 eqs.	0.69
	DBP	$n = (64, 8)$, 587 eqs.	0.74
	EXP	$1/N$	0.05
	EXP	$1/N^2$	0.32
3	*SIM*	180k runs	1209
	MF	-	0.07
	DBP	$n = (16, 8, 2)$, 462 eqs.	0.80
	DBP	$n = (24, 12, 3)$, 1303 eqs.	13.65
	DBP	$n = (32, 16, 4)$, 2808 eqs.	27.25
	DBP	$n = (36, 18, 5)$, 4221 eqs.	31.15
	EXP	$1/N$	0.25
	EXP	$1/N^2$	3.88
4	*SIM*	360k runs	1192
	MF	-	0.04
	DBP	$n = (8, 4, 2, 1)$, 274 eqs.	0.40
	DBP	$n = (12, 6, 3, 2)$, 1096 eqs.	7.34
	DBP	$n = (16, 8, 4, 2)$, 2299 eqs.	14.19
	DBP	$n = (20, 10, 5, 3)$, 5558 eqs.	31.37
	EXP	$1/N$	0.17
	EXP	$1/N^2$	4.89

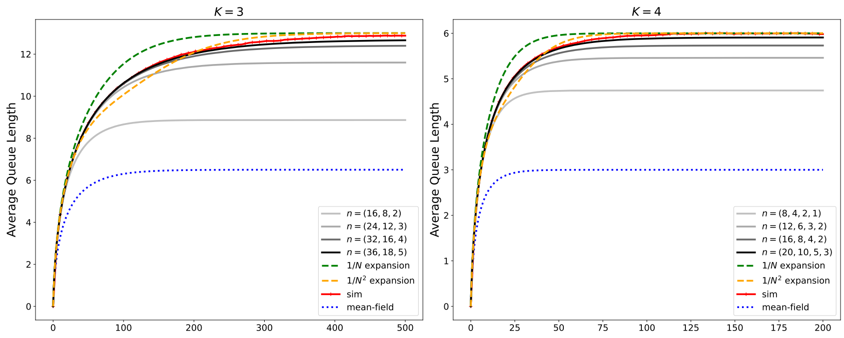


Figure 9: Numerical results for the estimation of the average queue length of the egalitarian processor sharing queuing system with $K = 3$ and $K = 4$ classes of customers.

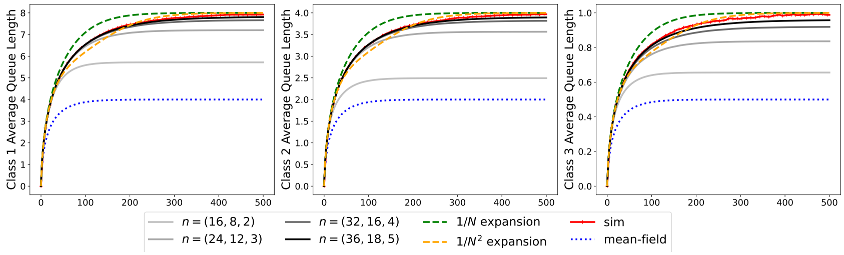


Figure 10: Numerical results for the egalitarian processor sharing queuing system with 3 classes of customers.

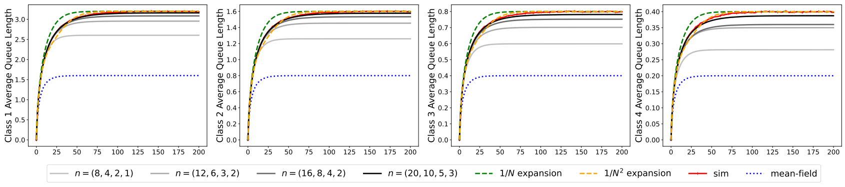


Figure 11: Numerical results for the egalitarian processor sharing queuing system with 4 classes of customers.

although it tends to better approximate the transient evolution. Regarding runtimes, the best DBP approximations take significantly longer than the $1/N^2$ expansions; however, DBP is still very competitive with respect to stochastic simulation, always requiring less than 3% runtime.

Chapter 3

h-scaling

Although the results in the previous chapter have shown that for certain MPPs DBP can significantly improve the estimation of the average dynamics of the system, the number of equations to be solved still scales exponentially in the dimension of the state vector. While the choice of n can help tackle this problem, as shown in Table 8, here we propose a second technique, called h-scaling, that can be used on its own or coupled with DBP to reduce the number of equations in any truncated version of the Master Equation. In Section 3.1 we first introduce h-scaling, how it can be used as a direct approximation of the Master Equation (static scaling), and how it can be coupled with DBP (scaled DBP). Preservation of limit results for the scaled process is proved in Section 3.2. Section 3.3 introduces a multi-scale version of h-scaling, while numerical examples are shown in Section 3.4.

3.1 Scaled Processes

3.1.1 *h*-scaling

Consider an MPP as defined in Section 1.1.1. Let $R(\mathcal{S})$ be a minimal (with respect to inclusion) hyper-rectangle in $\mathbb{N}^m$ containing $\mathcal{S}$ and let $v_R \in R(\mathcal{S})$ be such that $(v_R)_i = \min\{x_i | x \in R(\mathcal{S})\}$. We can imagine $\mathcal{S}$ as a subset of vertices of the m-dimensional grid covering $R(\mathcal{S})$ having edges of length 1, where v_R is the vertex of $R(\mathcal{S})$ with minimal components. We want to cover $R(\mathcal{S})$ with a coarser grid to contain fewer vertices in the original hyper-volume.

To do this, we fix a scalar parameter $h > 1$; in some cases, h can be chosen to be a vector, extending all the present results, as will be discussed in Section 3.3. Now, recall that by $H(\mathcal{D})$ denotes the convex hull in $\mathbb{R}^m$ of any discrete set $\mathcal{D}$. We define the state space of the scaled process as:

$$\mathcal{S}^h = \{x = v_R + h(k_1 e_1 + \ldots + k_m e_m) | k_i \in \mathbb{N}\, \forall\, i = 1, ..., m\} \cap H(R(\mathcal{S}))$$

where e_i is the m-dimensional with 1 as i-th component and 0 else.

We now define a process X^h evolving on $\mathcal{S}^h$. We set

$$X^h(0) = v_R + h\lfloor \frac{x_0 - v_R}{h} \rfloor \in \mathcal{S}^h \tag{3.1}$$

and define the set transitions of X^h as $\mathcal{L}^h = \{hl \,|\, l \in \mathcal{L}\}$, each associated with rate $\frac{1}{h} f_l(x) \mathbb{I}_{\{x+hl \in \mathcal{S}^h\}}$. We call the process X^h so defined the h-scaling of X.

Example. *Consider again the M/M/k queue introduced in Example 1 and assume that the queue starts with zero customers, i.e., $X(0) = 0$.*

The original state space is $\mathcal{S} = \mathbb{N}$ and $\mathcal{H}(R(\mathcal{S})) = \mathbb{R}_{\geq 0}$ so we will have that $\mathcal{S}^h$ it is still infinite, with $|\mathcal{S}| = |\mathcal{S}^h|$, but it is different from $\mathcal{S}$:

$$\mathcal{S}^h = \{hn \,|\, n \in \mathbb{N}\}.$$

The h-scaling is X^h, such that $X^h(0) = 0$ and X^h evolves according to the following transitions:

$$\begin{aligned} l_1^h &= h \quad \textit{at rate } \frac{\lambda}{h}, \\ l_2^h &= -h \quad \textit{at rate } \frac{\mu}{h} \min(x, k). \end{aligned} \tag{3.2}$$

Now consider the M/M/k/N queue from the same Example. In this case, $\mathcal{S} = \{0, 1, \ldots, N\}$ and the h-scaling for the M/M/k/N queue is defined by the transitions:

$$\begin{aligned} l_1^h &= h \quad \textit{at rate } \frac{\lambda}{h} \mathbb{I}_{\{x < h \lfloor \frac{N}{h} \rfloor\}}, \\ l_2^h &= -h \quad \textit{at rate } \frac{\mu}{h} \min(x, k). \end{aligned} \tag{3.3}$$

Therefore, in this case, $\mathcal{S}^h = \{0, h, \ldots, h\lfloor \frac{N}{h} \rfloor\}$ so we are indeed reducing the number of states as $|\mathcal{S}^h| = \lfloor \frac{N}{h} \rfloor + 1 \leq N + 1 = |\mathcal{S}|$.

Interpretation of h-scaling. When h is an integer, a physical interpretation of the process X^h is possible. For example, consider the $M/M/k$ of the example and set $h = 2$. X^2 is a process performing transitions $x \to x + 2$ with rate $\frac{\lambda}{2}$ and $x \to x - 2$ with rate $\frac{\mu}{2}\min(x, k)$. This means that in X^2 every event involves two perfectly synchronized agents that arrive and leave the queue together, and each event takes place after a time which is, on average, exactly two times the average time after which a single agent would perform that transition given the same initial conditions. For $h \in \mathbb{Q}$, the state space of X^h takes values in the real space. However, since the components of the state vector of an MPP represent population counts, non-integer values escape physical intelligibility. Nevertheless, we will show that rational values of h are useful to tune the accuracy of the approximation.

3.1.2 Static Scaling

A first approximation can be obtained simply by solving the Master Equation for X^h, yielding fewer equations than the one for X. This is possible when the state space is finite or when we consider a sufficiently large truncation of an infinite state space to contain most of the probability mass (Munsky and Khammash, 2006b). We will call this approximation *static scaling*.

Without loss of generality, let us consider the finite state space $\mathcal{S} = \{0, 1, \ldots, N_1\} \times \ldots \times \{0, 1, \ldots, N_m\}$, thus $|\mathcal{S}| = \prod_{i=1}^{m}(N_i + 1)$ is the number of equations of the Master Equation. Applying h-scaling for $h > 1$ gives as new state space $\mathcal{S}^h = \{0, h, \ldots, h\lceil \frac{N_1}{h} \rceil\} \times \ldots \times \{0, h, \ldots, h\lceil \frac{N_m}{h} \rceil\}$, with $|\mathcal{S}^h| = \prod_{i=1}^{m}\left(\lceil \frac{N_i}{h} + 1 \rceil + 1\right) \leq |\mathcal{S}|$. For all $x \in \mathcal{S}^h$ the Master Equation for X^h can be written as

$$\frac{dP^h(x)}{dt} = \sum_{l \in \mathcal{L}} \frac{1}{h} f_l(x - hl) P^h(x - hl; t) - \sum_{l \in \mathcal{L}} \frac{1}{h} f_l(x) \mathbb{I}_{\{x + hl \in \mathcal{S}^h\}} P^h(x; t). \tag{3.4}$$

We can then approximate $\mathbb{E}[X]$ by solving (3.4) and computing

$$\mathbb{E}[X^h] = \sum_{x \in \mathcal{S}^h} x P^h(x).$$

Example. *As we have seen, h-scaling applied to an $M/M/k/N$ queue yields* (3.3).

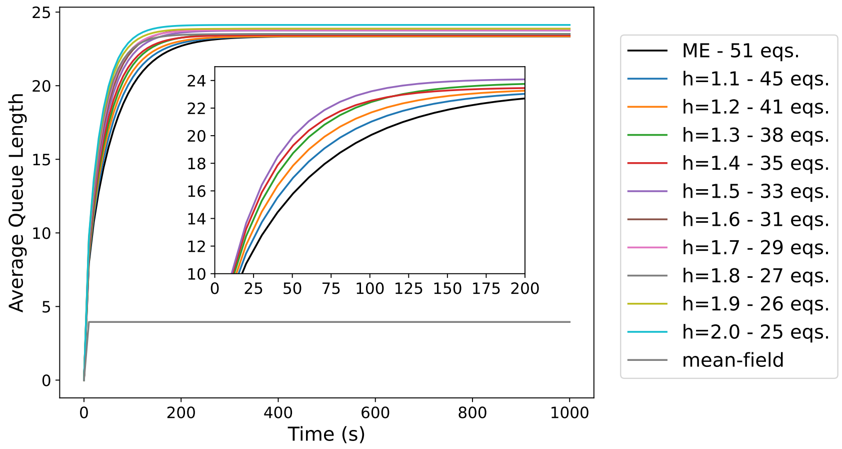


Figure 12: Application of static h-scaling to the $M/M/k/N$ queue. ME = true mean computed by numerically solving the master equation.

The Master Equation for X^h is then

$$\frac{dP^h(x)}{dt} = \begin{cases} -\frac{\lambda}{h}P^h(0;t) + \frac{\mu}{h}\min(h,k)P^h(h;t) & x = 0 \\ -\left(\frac{\lambda}{h} + \frac{\mu}{h}\min(x,k)\right)P^h(x;t) + \\ \qquad +\frac{\lambda}{h}P^h(x-h;t) + \\ \qquad\qquad +\frac{\mu}{h}\min(x+h,k)P^h(x+h;t) & x \neq 0, h\lfloor\frac{N}{h}\rfloor \\ -\frac{\mu}{h}\min\left(h\lfloor\frac{N}{h}\rfloor, k\right)P^h\left(h\lfloor\frac{N}{h}\rfloor;t\right) + \\ \qquad\qquad +\frac{\lambda}{h}P^h\left(h\left(\lfloor\frac{N}{h}\rfloor - 1\right);t\right) & x = h\lfloor\frac{N}{h}\rfloor \end{cases}$$

In Figure 12 we can see the results for $h = 1.2, 1.4, 1.6, 1.8, 2.0$ applied to an $M/M/k/N$ queue with parameters $k = 4, N = 50, \lambda = 3.95, \mu = 1$. We can see that the number of equations is progressively reduced up to 50% while the mean estimated using the h-scaling still keeps a low relative error with respect to the true mean (at steady-state no more than 4% of the true value).

3.1.3 Scaled Dynamic Boundary Projection

We can now combine h-scaling with DBP. We now assume that X evolves on a state space $\mathcal{S} \subseteq \mathbb{N}^m$ (not necessarily finite) and that DBP with parameter $n \in \mathbb{N}^m$ can be applied to X yielding system (2.14)-(2.15), consisting of $\mathcal{N}(n) + m$ equations. We will show how to apply DBP to the scaled process X^h and how this reduces the number of approximating equations.

In this case, our idea is to cover the portion of the state space inside a truncation $\mathcal{T}_y^{(n)}$ with a coarser grid. To do so, we define the truncations:

$$\mathcal{T}^h(n,y) = \{x = y + h(k_1 e_1 + \ldots + k_m e_m) \,|\, k_i \in \mathbb{N} \,\forall\, i = 1, \ldots, m\} \\ \cap H(\mathcal{T}_y^{(n)}). \quad (3.5)$$

Observe that the states in $\mathcal{T}^h(n,y)$ are not necessarily in $\mathcal{S}^h$ (although they are if $y \in \mathcal{S}^h$). Moreover, $\mathcal{T}^h(n,y)$ has $\mathcal{N}^h(n) = \prod_{i=1}^m \left(\lfloor \frac{n_i}{h} \rfloor + 1\right) \leq \mathcal{N}(n)$ states for any y. This implies that the number of equations in scaled DBP is reduced by a factor $\frac{1}{h^m}$, and we have relatively heavier reductions as the dimensionality of the state space increases. Once the definition of truncation in $\mathcal{S}^h$ is clarified, the derivation of the equations for scaled DBP follows, step-by-step, the one for the original process. It is detailed below.

The equations for scaled DBP with parameters n and h are given by:

$$\begin{aligned} \frac{dY^{(n,h)}}{dt} &= \sum_{l \in \mathcal{L}} \sum_{x \in \partial \mathcal{T}_l^h(n,0)} \mathcal{Y}_l^h(n,x) \frac{1}{h} f_l(x + Y^{(n,h)}(t)) P^{(n,h)}(x;t) \\ \frac{dP^{(n,h)}}{dt} &= Q^{(n,h)}(Y^{(n,h)}(t)) P^{(n,h)}(\,\cdot\,, t). \end{aligned} \quad (3.6)$$

Derivation of Scaled DBP

Having defined the truncations in Section 3.5, we proceed as in the derivation for DBP.

The border sets for the scaled truncations are defined as:

$$\partial \mathcal{T}_l^h(n,y) = \left\{x \in \mathcal{T}^h(n,y) : x + hl \notin \mathcal{T}^h(n,y)\right\}, \text{ for } l \in \mathcal{L},$$

$$\partial \mathcal{T}^h(n,y) = \bigcup_{l \in \mathcal{L}} \mathcal{T}_l^h(n,y) = \left\{x \in \mathcal{T}^h(n,y) : \exists\, l \in \mathcal{L} \text{ s.t. } x + hl \notin \mathcal{T}^h(n,y)\right\}$$

We can then define the boundary projection of X^h on $\mathcal{T}^h(n,y)$, in which every jump from $x \in \partial \mathcal{T}_l(n,y)$ to x' is redirected with same rate to x^* defined as:

$$x_i^* = \begin{cases} \min(y_i + h\lfloor \frac{n_i}{h} \rfloor, x_i') & \text{if } x_i' > x_i \\ \max(y_i, x_i') & \text{if } x_i' < x_i \\ x_i & \text{if } x_i' = x_i. \end{cases}$$

After performing the augmentation, we get the jump vectors $l^{(n,h)}(x)$ defined exactly as before. Then, letting $X_y^{(n,h)}$ be the boundary projection of X^h on $\mathcal{T}^h(n,y)$, its transition matrix $Q^{(n,h)}(y)$ can be written for $x, x' \in \mathcal{T}^h(n,0)$ as:

$$[Q^{(n,h)}(y)]_{x,x'} = \begin{cases} \sum_{l\in\mathcal{L}} \mathbb{I}_{\{x'+l^{(n,h)}(x')=x\}} \frac{1}{h} f_l(x'+y) & \text{if } x \neq x' \\ -\sum_{l\in\mathcal{L}} \mathbb{I}_{\{l^{(n,h)}(x)\neq 0\}} \frac{1}{h} f_l(x+y) & \text{if } x = x'. \end{cases}$$

So the Master Equation for $X_y^{(n,h)}$ as:

$$\frac{dP_y^{(n,h)}}{dt} = Q^{(n,h)}(y) P_y^{(n,h)}(\,\cdot\,;t)$$

where $P_y^{(n,h)}(\,\cdot\,;t)$ is an $\mathcal{N}^h(n)$-dimensional vector.

Again, to pass to DBP, we need to define the functions:

$$\Pi_i^{(n,h)}(x,y) = \begin{cases} x_i & x_i < y_i \\ y_i + h\lceil x_i - \big(y_i + \lfloor \frac{n_i}{h} \rfloor\big)\rceil & x_i > y_i + n_i \\ y_i & y_i \leq x \leq y_i + n_i. \end{cases} \quad \forall\, x, y \in \mathcal{S}^h$$

$$\mathcal{Y}_l^h(n,x) = \Pi^{(n,h)}(x+l, 0) \quad \forall\, l \in \mathcal{L}, \forall\, x \in \partial\mathcal{T}_l^h(n,0).$$

Observe that the second case in the definition of $\Pi^{(n,h)}(x,y)$ is motivated by the fact that x may not be in the form $y + h(k_1e_1 + \ldots + k_me_m)$, and, to mirror what happens in classic DBP, we want the function to return the closes y' in this form so that $\mathcal{T}^h(n,y')$ contains x.

Then the equations for scaled DBP with parameter n are given by:

$$\begin{aligned} \frac{dY^{(n,h)}}{dt} &= \sum_{l\in\mathcal{L}} \sum_{x\in\partial\mathcal{T}_l^h(n,0)} \mathcal{Y}_l^h(n,x) \frac{1}{h} f_l(x + Y^{(n,h)}(t)) P^{(n,h)}(x;t) \\ \frac{dP^{(n,h)}}{dt} &= Q^{(n,h)}(Y^{(n,h)}(t)) P^{(n,h)}(\,\cdot\,,t). \end{aligned} \tag{3.7}$$

Again, supposing $X(0) = x_0$ with probability 1, to define the initial condition we set:

$$\Big[Y^{(n)}(0)\Big]_i = \max\Big(0, x_{0,i} - h\Big\lfloor \frac{n_i}{h} \Big\rfloor\Big)$$

$$x_0^* = h\Big\lfloor \frac{x_0 - Y^{(n)}(0)}{h} \Big\rfloor$$

$$P^{(n)}(x;0) = \begin{cases} 1 & \text{if } x = x_0^*, \\ 0 & \text{else.} \end{cases}$$

Example. *We now consider the $M/M/k$ queue with $\lambda = 3.85, \mu = 1$ and $k = 4$. In principle, the model has an infinite state space, but it converges to its steady state distribution, so it is possible to select a finite truncation of the state space so that the probability mass outside it is arbitrarily small. To select the minimal truncation that we can take as ground truth, we start by considering the Master Equation with 500 states and progressively reduce the number of states so the error introduced is less than 0.001% of the Average Queue Length (AQL) at steady state. We obtain that we need 375 equations to capture the queue's dynamics correctly.*

DBP can be applied to this system to reduce the number of equations needed to approximate the system's dynamics. We see that using $n = 170$, the relative error between the DBP approximation and the solution of the Master Equation at steady state is less than 1%.

We can further reduce the number of equations by coupling DBP with h-scaling at the price of a bigger error; this yields the equations:

$$\frac{dY^{(n,h)}}{dt} = -\frac{\mu}{h}\min\left(Y^{(n,h}(t),k\right)P^{(n,h)}(0;t) + \frac{\lambda}{h}P^{(n,h)}\left(h\left(\left\lfloor\frac{N}{h}\right\rfloor - 1\right);t\right)$$

$$\frac{dP^{(n,h)}(x)}{dt} = \begin{cases} -\frac{\lambda}{h}P^{(n,h)}(0;t) + \frac{\mu}{h}\min\left(h + Y^{(n,h)}(t),k\right)P^{(n,h)}(h;t) & x = 0 \\ -\left(\frac{\lambda}{h} + \frac{\mu}{h}\min\left(x + Y^{(n,h)}(t),k\right)\right)P^{(n,h)}(x;t)+ & \\ \quad +\frac{\lambda}{h}P^{(n,h)}(x-h;t)+ & \\ \qquad +\frac{\mu}{h}\min\left(x + h + Y^{(n,h)}(t),k\right)P^{(n,h)}(x+h;t) & x \neq 0, h\lfloor\frac{N}{h}\rfloor \\ -\frac{\mu}{h}\min\left(h\lfloor\frac{N}{h}\rfloor + Y^{(n,h)}(t),k\right)P^{(n,h)}\left(h\lfloor\frac{N}{h}\rfloor;t\right)+ & \\ \qquad +\frac{\lambda}{h}P^{(n,h)}\left(h\left(\lfloor\frac{N}{h}\rfloor - 1\right);t\right) & x = h\lfloor\frac{N}{h}\rfloor \end{cases}$$

As seen from Figure 13 and Table 7, for the same value of h, scaled DBP performs better than h-scaling while yielding a heavier reduction in the number of equations. We will see that this is the case also for more complex examples.

3.2 Limit Behaviour

We now prove that for a fixed $h > 1$ the approximating process X^h shares the same limiting behaviour as X. To lighten the notation, we will assume $v_R = 0$.

3.2.1 Preservation of the Mean-Field Limit

Suppose that the original process X is part of a sequence $\left(X^N\right)_{N \geq N_0}$ satisfying the hypotheses of Theorem 1. Fix $h > 1$ and consider a new se-

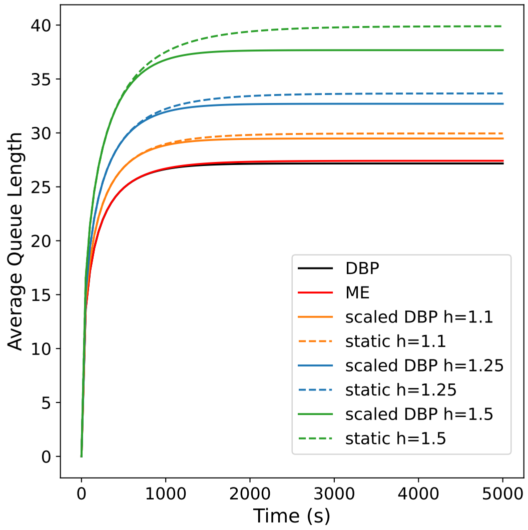


Figure 13: h-scaling and scaled DBP applied to the $M/M/k$ queue.

	h-scaling			scaled DBP		
h	err	# eqs.	red.	err	# eqs.	red.
1.0	-	376	-	0.91%	172	54.25%
1.1	9.25%	341	9.31%	7.54%	156	58.51%
1.25	22.78%	301	19.94%	19.29%	138	63.29%
1.5	45.50%	251	33.24%	37.4%	115	69.41%

Table 7: Approximated value of the AQL of the $M/M/k$ queue at steady state (t=5000), with relative error, number of equation and reduction in the number of equations when h-scaling and scaled DBP are applied. h-scaling with $h = 1$ is considered the ground truth.

quence $\left(X^{N,h}\right)_{N \geq N_0}$ where each $X^{N,h}$ is obtained applying the h-scaling to X^N.

It is immediate to observe that the new sequence $\left(X^{N,h}\right)_{N \geq N_0}$ still satisfies the hypotheses of the theorem, and, in particular, for every N

$$F^{N,h}(x) = \sum_{l \in \mathcal{L}} h\hat{l}^n \frac{1}{h} \hat{f}_l^N(x) = F^N(x),$$

which means that under the proposed scaling the drift function is preserved for every N and independent of h. This implies that the deterministic limit process $\hat{x}(t)$ defined by (1.4) is exactly the same for both sequences (in fact, observe that $\lim_N h\lfloor \frac{x_0^N}{h} \rfloor = x_0$, so also the limiting initial condition is the same).

This can be summed up in the following theorem:

Theorem 5. *Consider a sequence of processes $\left(X^N\right)_{N \geq N_0}$ and consider the sequence of approximating processes $\left(X^{N,h}\right)_{N \geq N_0}$ obtained applying h-scaling for a fixed $h > 1$. If the original sequence admits a deterministic limit $\hat{x}$ in the sense of Theorem 1, then the sequence of approximating processes admits the same limit.*

In the special case of density-dependent processes for all $l \in \mathcal{L}$ the scaled transitions have transition vectors $\frac{h}{\gamma_n} v_l$ and rate function $\frac{\gamma_N}{h} g_l(x)$. This is equivalent to saying that the sequence $\left(X^{N,h}\right)_{N \geq N_0}$ is a density-dependent family with respect to the parameter $\frac{\gamma_N}{h}$. This is the same observation that in Ciocchetta et al., 2009 led to prove the limit behaviour for $h \to 0$, and it is easily explained by the fact that we are approximating systems of size γ_N with systems of size $\frac{\gamma_N}{h}$ scaling coherently both the magnitude of the jumps and the transition rates, which are the two quantities involved in the density-dependence assumption.

Extension to Other Mean-Field Limit Results. The fact that the deterministic approximation is preserved with exactly the same limit drift F allows us to extend to $\left(X^{N,h}\right)_{N \geq N_0}$ other limit results, provided they hold for the original sequence. In particular, if E is compact and the ODE in (1.4) admits a globally asymptotically stable fixed point x^*, every sequence of invariant measures of X^N tends weakly to the Dirac distribution centered on x^* (Benaim and Le Boudec, 2008b). The same is true for any sequence of invariant measures of the sequence $\left(X^{N,h}\right)_{N \geq N_0}$.

Analogously, we can straightforwardly extend to $\left(X^{N,h}\right)_{N \geq N_0}$ results related to mean-field independence (Benaim and Le Boudec, 2008b), both in transient and steady-state, and to fast simulation (RWR Darling and J. R. Norris, 2008) provided the original sequence satisfies the required assumptions.

3.2.2 Preservation of the LNA

Similarly to the mean-field limit, we assume that our original process X belongs to a sequence of processes $\left(X^N\right)_{N \geq N_0}$ satisfying the hypotheses of Theorem 2. Again, we fix $h > 1$ and consider a sequence $\left(X^{N,h}\right)_{N \geq N_0}$ obtained by applying the h-scaling to each process of the original sequence. Then, the following result hold:

Theorem 6. *Suppose that for the sequence $\left(X^N\right)_{N \geq N_0}$ the hypotheses of Theorem 2 are verified and, in addition:*

- *equation* (1.4) *admits a globally asymptotically stable equilibrium x^*;*
- *for each N $X^N(0) = N\hat{x}_0$;*
- *for each N $\gamma_N = N$.*

Then, letting $\mu(t)$ and $\Sigma(t)$ denote the mean and the covariance matrix of limiting Gaussian process for the original sequence, we have that the sequence of approximating processes $\left(X^{N,h}\right)_{N \geq N_0}$ admits a Gaussian limiting process with mean $\mu^h(t)$ and covariance matrix $\Sigma^h(t)$ such that:

$$\lim_{t\to\infty} \mu^h(t) = \lim_{t\to\infty} \mu(t) = 0 \textit{ and } \Sigma^h(t) = \Sigma(t) \,\forall\, t \geq 0.$$

Proof. Theorem 2 guarantees that under the hypothesis $\mu(t)$ and $\Sigma(t)$ exist. The rest of the proof is obtained by following the same derivation used in Van Kampen, 1992 with the ansatz:

$$\hat{X}^{N,h}(t) = \hat{x}(t) + \sqrt{\frac{h}{N}}\xi^h(t). \tag{3.8}$$

and verifying that $\xi^h(t)$ is a Gaussian Process whose mean $\mu^h(t)$ and covariance $\Sigma^h(t)$ satisfy exactly the same ODEs as $\mu(t)$ and $\Sigma(t)$, i.e. (1.6) and (1.7).

Furthermore, in the sequence of the approximated processes, $\left(X^{N,h}\right)_{N \geq 0}$ we have redefined the initial conditions as $X^{N,h}(0) = h\lfloor \frac{N\hat{x}_0}{h} \rfloor$, while the initial condition for the deterministic process remains unchanged. Therefore, when setting the initial condition for $\mu^h(y)$ we need to take into account that for the ansatz to be valid at time $t = 0$ the Gaussian Limit Process possibly has a non-zero mean, namely $\mu^h(0) = \sqrt{\frac{h}{N}} \left(\lfloor \frac{N\hat{x}_0}{h} \rfloor - N\hat{x}_0 \right)$.

So, in general, $\xi^h(t)$, describing the fluctuations of $X^{N,h}$, is different from $\xi(t)$, describing the fluctuations of X^N, since $\mu^h(0) \neq \mu(0) = 0$ (observe that instead the covariance matrix is still the same, i.e., $\Sigma^h(t) = \Sigma(t)\, \forall\, t \geq 0$). However, equation (1.6) is exactly the variational equation associated with the ODEs defining the deterministic limit (1.4), so, regardless of its initial condition, its solution must tend to 0 as $\hat{x}(t)$ tends to the equilibrium x^*. This implies $\lim_{t\to\infty} \mu^h(t) = \lim_{t\to\infty} \mu(t) = 0$.

Observe that all the introduced hypotheses are needed for the correct application of the ansatz: the differentiability of the drifts is needed to apply the Taylor expansion as in Van Kampen, 1992, while the presence of a globally asymptotically stable equilibrium ensures that the ansatz remains valid for $t \in [0, +\infty)$. $\square$

3.3 Multi-scale Approximation

In some cases, it can be useful to consider, instead of a single scalar parameter h, an m-dimensional vector $\underline{h} \in \mathbb{R}^m$, $h_i \geq 1\, \forall\, i = 1, \ldots, m$. In this case we will re-scale the jumps in different components using different values h_i. This can be desirable when one component evolves on a much larger space than the others. However, we cannot arbitrarily choose $\underline{h}$, since we need to preserve the quantities transformed by the transitions. If in the original process a agents of class i transition into b agents of class j, the scaling parameters h_i and h_j must be chosen so that the corresponding transition preserves this conversion in the scaled process.

We can index the components of the original process with the set $I = \{1, \ldots, m\}$. We say that component i transitions to j, written $i \leftrightarrow j$, if a sequence of transitions transforms an agent of class i into an agent of class j. We require that when choosing $\underline{h}$, if $i \leftrightarrow j$, then $h_i = h_j$. We call a vector $\underline{h}$ satisfying this assumption a *valid* h-scaling vector.

Observe that the constraints on $\underline{h}$ depend not on the rate functions but only on the transition vectors l. The scalar case corresponds to $h_i = h_j\, \forall i, j \in I$.

All previous results can be extended to the multi-scale case, provided $\underline{h}$ is a valid vector. For an application, see Section 3.4.1.

Example. *Consider again the Egalitarian Processor Sharing system with K classes of customers of Example 4.*

In this system, agents cannot transition from one class into another so we can consider any h-scaling vectors with $h_i \neq h_j$ for each $i, j \in I$.

Example. *Consider the Malware Propagation model introduced in Example 3. Since $D \leftrightarrow A$, we need to set $h_D = h_A$ for each valid h-scaling vector, i.e., we cannot use a multi-scale approach in this case.*

For a valid h-scaling vector $\underline{h}$ we define the scaled state space $\mathcal{S}^{\underline{h}}$ as

$$\mathcal{S}^{\underline{h}} = \{x = v_R + (k_1 h_1 e_1 + \ldots + k_m h_m e_m) | k_i \in \mathbb{N} \, \forall \, i = 1, ..., m\} \cap \mathcal{H}(R(\mathcal{S})).$$

The scaled process $X^{\underline{h}}$ will have initial condition $X^{\underline{h}}(0)$ with components $X^{\underline{h}}_i(0) = (v_R)_i + h_i \lfloor \frac{(x_0)_i - (v_R)_i}{h_i} \rfloor \in \mathcal{S}^{\underline{h}}$.

Finally, we observe that, by definition of valid h-scaling vector, for each transition vector $l \in \mathcal{L}$, the scaling parameters h_i associated with non-null components of l, i.e. components i such that $l_i \neq 0$, are all equal to a certain value h_l. Then, for all $l \in \mathcal{L}$, we define the transitions of $X^{\underline{h}}$ as having transition vector $h_l l$ and rate function $\frac{1}{h_l} f_l(x) \mathbb{I}_{\{x + h_l l \in \mathcal{S}^{\underline{h}}\}}$.

3.4 Examples

We look again at the Egalitarian Processor Sharing and the Malware Propagation Model. While in the previous chapter, the average over a sufficient number of simulations was taken as ground truth, we now compare our results with the solution of the Master Equation (truncated to a sufficient number of states when necessary). We compute the error as the L^1 norm of the difference between the vector $\mathbb{E}[X(t)]$, the average evolution of X at time t computed by the (truncated) Master Equation, and the same quantity computed using h-scaling or scaled DBP. Results show while computing the mean from the original Master Equation requires a prohibitive amount of time this can be reduced significantly by applying h-scaling and even more efficiently using DBP and its scaled version. All experiments were performed on a laptop with a 2.8 GHz Intel i7 quad-core processor and 16 GB RAM.

Effect of h scaling on simulation time.

One could wonder if any computational gain can be achieved by simulating the rescaled process X^h instead of the original process. Observe that (3.8) implies that $\mathrm{Var}(X^h(t)) = h\mathrm{Var}(X(t))$, so the approximating process obtained by h-scaling has higher variance. This has an unlucky consequence: the number of simulations needed to achieve a certain accuracy when we use Gillespie's Stochastic Simulation Algorithm to estimate a functional of the process will also increase! In particular, if we require that the confidence interval for the estimated quantity is smaller than a certain percentage of the estimated mean, we expect the number of simulations to increase proportionally to the variance, i.e., proportionally to h.

On the other hand, it is known that the time needed for a single simulation scales with the inverse of the average time needed for the next transition to occur (Anderson and Koyama, 2012). Such average time is the inverse of the sum of all transition rates computed in the current state. Since in h-scaling all rates are divided by h, the time needed for each simulation scales with $\frac{1}{h}$.

Overall, we expect that the time for simulating a sufficient number of $X^h(t)$ trajectories is comparable to that needed to achieve the same accuracy by simulating $X(t)$. So, our scaling does not achieve any computational advantage when we deal with simulations on infinite state spaces. However, it can be coupled with Dynamic Boundary Projection to speed up the method while still achieving a low approximation error.

3.4.1 Egalitarian Processor Sharing

For the Egalitarian Processor Sharing model we set the parameters to $N = 5, \lambda_1 = N \cdot 0.5, \lambda_2 = N \cdot 0.4, \mu = N$ and the initial condition to $x_{Q_1}(0) = x_{Q_2}(0) = 0$.

The considered model has an infinite state space. However, choosing a truncation of the type

$$\{(x_{Q_1}, x_{Q_2}) | x_{Q_1} \leq N_1, x_{Q_2} \leq N_2\}$$

for sufficiently large parameters N_1, N_2, it is possible to approximate the mean dynamics with arbitrary accuracy.

We start by setting $N_1 = 150, N_2 = 120$ and progressively reduce the state space until the error introduced exceeds the 0.001% of the AQL at steady state. We find that to keep the error below the chosen threshold we need to set $N_1 = 135, N_2 = 108$. This truncation corresponds to an approximated Master Equation with almost 15000 equations, so the time needed to solve it is considerably high ($\approx$ 8 hours). We can apply

Table 8: Approximated values of the Egalitarian Processor Sharing model at steady state (t=1000), with relative error, number of equations, reduction in the number of equations, and computational time (in seconds) when h-scaling and scaled DBP are applied. h-scaling with $h = 1$ is considered the ground truth.

	h-scaling					scaled DBP				
h	AQL	err	# eqs.	red.	time	AQL	err	# eqs.	red.	time
$(1.0, 1.0)$	53.97	-	14824	-	$3.09e4$	53.46	0.94%	4049	72.70%	$4.44e3$
$(1.5, 1.2)$	57.23	6.04%	8281	44.14%	1.67e4	55.36	2.57%	2211	85.01%	$1.73e3$
$(2.0, 1.6)$	61.19	13.38%	4624	68.81%	$5.43e3$	57.12	5.83%	1298	91.26%	$5.20e2$
$(2.5, 2.0)$	64.95	20.35%	3025	79.59%	$1.40e3$	58.26	7.94%	843	94.32%	$1.43e2$
$(3.0, 2.4)$	68.39	26.71%	2116	85.72%	$8.99e2$	58.82	8.97%	578	96.11%	$1.18e2$

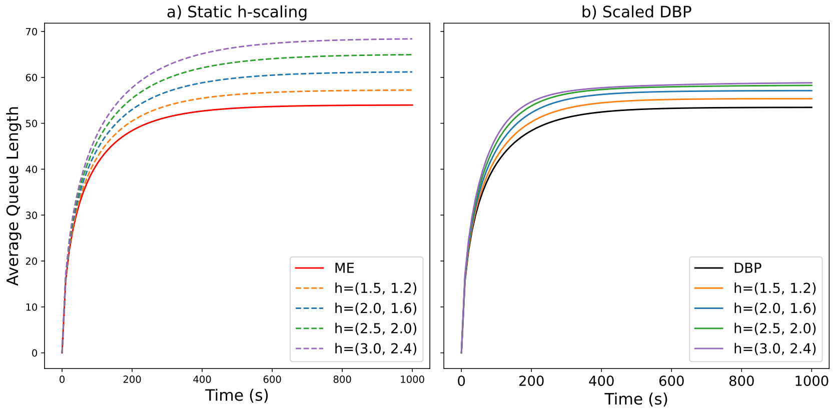


Figure 14: Application of h-scaling and scaled DBP to Egalitarian Processor Sharing with two classes of customers.

static h-scaling to reduce the number of equations and consequently the computational time, but this comes at the cost of a relative error that can reach 27% of the AQL at steady state (see Table 8 and Figure 14a). In general the computational time is proportional to the number of equations for static h-scaling is $\mathcal{N}^h(n) = \prod_{i=1}^{m}(\lfloor \frac{n_i}{h} + 1 \rfloor)$, while for scaled DBP is $\mathcal{N}^h(n) + 2$.

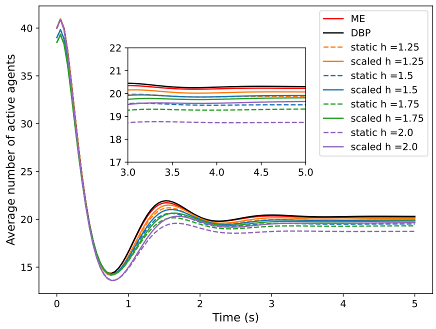
For this example, DBP already provides a significative advantage by allowing us to achieve an error of less than 1% by choosing $n = (70, 56)$ that corresponds to roughly 4000 equations. This reduces the computational time significantly. Moreover using scaled DBP with the same h used for the static case shows that a smaller relative error can be achieved with a smaller number of equations by reducing the computational time to less than 2 minutes with a relative error of less than 10% (see Table 8 and Figure 14b).

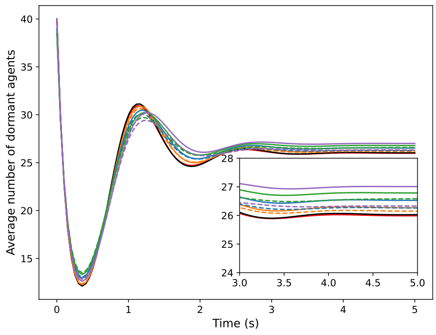
3.4.2 Malware Propagation

For the Malware Propagation Model we consider $N = 80$ agents and initial conditions $x_D(0) = x_A(0) = 40$.

Since this system has a finite state space given by $\{(x_D, x_A) | x_A + x_D \leq N\}$, its Master Equation can be solved exactly. Taking into account the constraint, the Master Equation yields $\frac{(N+1)\cdot(N+2)}{2}$ equations, instead of $(N+1)^2$, and the same is true for static h-scaling ($\frac{1}{2}(\lfloor \frac{N}{h} \rfloor + 1)(\lfloor \frac{N}{h} \rfloor + 2)$ equations). Analogously, in DBP and scaled DBP the only states of the truncation with probability greater than 0 will be the states (x_A, x_B) such that $x_A + Y_A^{(n,h)}(t) + x_B + Y_B^{(n,h)}(t) \leq 80$. However since the component $Y^{(n,h)}$ varies in time we write the equations for all $(n+1)^2$ states.

We measure the error as the sum of the relative errors on the three components S, A, D at steady state. Again, we choose n for DBP by requiring that the error is below 1%. This happens when we set $n = 50$ and significantly reduce the number of equations and the computational time. Moreover, using scaled DBP allows us to outperform static h-scaling in terms of accuracy and further reduce the computational times while keeping the error below 10% (see Figures 15 and Table 9).





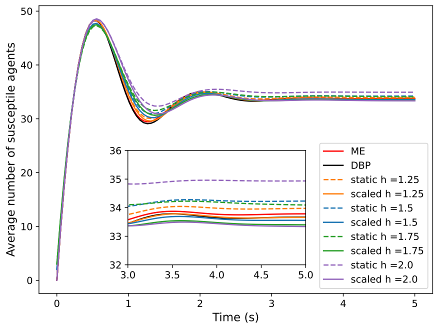


Figure 15: From above: average number of active, dormant and susceptible agents in the Malware Propagation model computed using h-scaling and scaled DBP.

Table 9: Total relative error in the mean dynamics of the Malware Propagation model at steady state (t=5), with the number of equations, reduction in the number of equations and computational time (in seconds) when h-scaling and scaled DBP are applied. h-scaling with $h = 1$ is considered the ground truth

	h-scaling				scaled DBP			
h	err	# eqs.	red.	time	err	# eqs.	red.	time
1.0	-	3321	-	5.19e3	0.87%	2601	21.68%	2.38e3
1.25	2.92%	2145	35.41%	1.73e3	2.22%	1681	49.38%	1.00e3
1.5	5.91%	1485	55.28%	6.61e2	4.28%	1156	65.19%	6.18e2
1.75	7.68%	1081	67.45%	3.20e2	6.23%	841	74.68%	1.63e2
2.0	12.1%	861	74.07%	2.48e2	8.06%	676	79.64%	1.00e2

Chapter 4

Inference of Probabilistic Programs With Moment-Matching Gaussian Mixtures

This chapter and the next one deal with the inference problem for bounded probabilistic programs. In particular, in this chapter, we introduce Gaussian Semantics, a family of approximating semantics whose practical implementation is the object of the next chapter. In Section 4.1, we introduce the state-of-the-art and our approach using a motivating example. In Section 4.2, we report some background material specific to the next two chapters. The control-flow syntax and exact semantics for probabilistic programs analyzed are introduced in Section 4.3, while Gaussian Semantics are introduced in Section 4.4. Our main theoretical result, the universal approximation theorem, is stated and proved in Section 4.5.

4.1 State-of-the-Art

Probabilistic programming languages are programming languages augmented with primitives expressing probabilistic behaviours (Gordon et al., 2014). Examples are random assignments (“program variable x is distributed according to the probability distribution D”), probabilistic choices ("do P_1 with probability p else P_2) or conditioning ("variable x is distributed according to D, under the constraint that it can only take positive values"). This has enabled a variety of applications, such as the analysis of randomized algorithms, machine learning and biology (Gordon et al., 2014).

Given a probabilistic program, there are different equivalent ways in which its semantics can be defined (Kozen, 1983). Following Kozen's Semantics 2 (Kozen, 1979), here we see a program as a transformer: given an initial joint distribution over the program variables, each instruction in the program transforms that joint distribution into a possibly different one, for example, due to the presence of probabilistic assignments or conditional statements. In this framework, we are interested in the *inference problem*: given a program P and an initial distribution D over program variables, what is the distribution over program variables after executing P? Borrowing from Bayesian inference, we will sometimes refer to the initial distribution D as the *prior distribution* over program variables and to the distribution obtained after executing P as the *posterior distribution*. Then, the inference problem boils down to computing the posterior.

Over the years, many approaches have tackled this problem: numerical methods based on Monte Carlo Markov chain (MCMC) sampling (Hastings, 1970; Nori et al., 2014; Goodman et al., 2008; V. Mansinghka, Selsam, and Perov, 2014; Pfeffer, 2001; A. Chaganty, Nori, and Rajamani, 2013), variational inference (VI) (Bingham et al., 2019; M. I. Jordan et al., 1999; Kucukelbir et al., 2015), symbolic execution (Gehr, Misailovic, and Vechev, 2016; Narayanan et al., 2016; Saad, Rinard, and V. K. Mansinghka, 2021), volume computation (Holtzen, Van den Broeck, and Millstein, 2020; Filieri, Păsăreanu, and Visser, 2013; Huang, Dutta, and Misailovic, 2021), and approaches based on moment-based invariants (Barthe et al., 2016; Chakarov and Sankaranarayanan, 2014; Katoen et al., 2010; Bartocci, Kovács, and Stankovič, 2020; Moosbrugger et al., 2022). Overall, each method presents its own pros and cons, and is tailored to particular classes of programs.

4.1.1 Motivating Example

As a motivating example, consider the *Tracking_n* model reported in Algorithm 1 and adapted from Wu et al. (2018). It describes a Gaussian process evolving on a bi-dimensional space for n steps and starting from coordinates (2, -1) (lines 1-5). A radar is positioned in (0,0) and can sense the process if it is at a squared distance ($dist$, line 6) of less than 10 units

Algorithm 1 *Tracking_n* adapted from (Wu et al., 2018)

```
1: $x = 2, y = -1$
2: **for** $i$ **in** $range(n)$ **do**
3:     $x = x + gauss(0, 1)$
4:     $y = y + gauss(0, 1)$
5: **end for**
6: $dist = x^2 + y^2$
7: **if** $dist > 10$ **then**
8:     $out = 1$
9: **else**
10:    $out = 0$
11: **end if**
12: **if** $out = 1$ **then**
13:    $obs\_dist = 10$
14: **else**
15:    $obs\_dist = gauss(dist, 1)$
16: **end if**
17: **observe**$(out == 1)$
18: **return** $obs\_dist$
```

from the radar. Therefore, the process can be either out of scope ($out = 1$, line 8) or in scope ($out = 0$, line 10). When the process is out of scope, the radar returns an observed distance of 10 (line 13) and a noisy measurement of the true distance else (line 15). Therefore, the distribution over *obs_dist* is a mixture of δ_{10}, i.e., a Dirac delta centered in 10, and a Gaussian with mean *dist*. However, if we observe that the process is out of scope (line 17), the posterior over *obs_dist* is just δ_{10} because any continuous distribution puts zero mass on a single point. Therefore, the exact posterior over *obs_dist* is a distribution placing probability 1 on 10.

While this program may seem quite simple, performing inference may be challenging. Using PSI (Gehr, Misailovic, and Vechev, 2016), an exact symbolic solution returns a formula for the posterior mean of *obs_dist* in less than a second, which, however, contains several non-simplified integrals. This is because, in line 6, computing *dist* requires computing the probability density function (pdf) of the product of two continuous distributions, and this requires symbolic integration. Attempting to integrate it numerically using Mathematica [Wolfram Research, Inc.] did not terminate after 30 minutes on a common machine.

One can resort to approximate approaches; however, many methods, such as STAN's MCMC sampling (Carpenter et al., 2017), Pyro's VI (Bingham et al., 2019) and AQUA's quantization (Huang, Dutta, and Misailovic, 2021), do not support discrete posteriors, therefore this particular program cannot be encoded in their syntax. BLOG is a probabilistic programming language relying on probabilistic relational model representation and likelihood weighting sampling (Milch, Marthi, and Russell, 2004), that has been extended by Wu et al. (2018) for mixtures of continuous and discrete distribution such as the one in our example. It computes the exact posterior in 0.516 s for $n = 1$ and about 5 s for $n = 100$. A similar behaviour is exhibited by applying Pyro's variable elimination (Obermeyer et al., 2019), which computes the exact posterior in 0.192 s for $n = 1$ and about 9 s for $n = 100$ (see Table 16).

4.1.2 Proposed Approach

We present an overview of our approach by examining separately: the choice of representation for the distributions, how the approximation is performed at each program location, the asymptotic correctness and the particular implementation presented in the next chapter. Finally, we consider again the motivating example, to show how our approximation method performs inference on it.

Representation. The difficulty in performing inference on the previous program stems from various factors: PSI's exact engine returns non-simplified integrals, requiring computationally expensive numerical integration. STAN's sampling, Pyro's VI and AQUA's quantization cannot be applied in this case, but in general, can incur long computational times and out-of-memory errors (see Section 5.2). BLOG's and Pyro's ad hoc sampling perform best, but increasing the number of steps hinders scalability. To complement all these techniques, we present a new *approximate analytical* method that does not require integration or sampling and that relies on a compact representation of the joint distribution using moment-matching Gaussian mixtures (GMs). Our choice of representation is based on some desirable properties of GMs, and, in particular, the following three: i) they can encode both continuous and discrete distributions (using degenerate GMs); ii) their moments can be computed exactly and efficiently; iii) they are universal approximators, so we can always increase the number of components in our representation to get

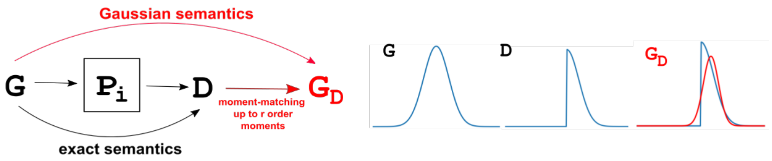


Figure 16: Left part: general approximation scheme used in Gaussian Semantics. In program location P_i, the exact semantics of the program transforms a GM G into a non-GM distribution D. In the same program location, Gaussian Semantics transform G into another GM G_D, that approximates D using moment-matching. Right part: concrete example. The Gaussian distribution G is transformed by the exact semantics into a truncated Gaussian D and by the second-order Gaussian Semantics into the red Gaussian distribution G_D.

a better approximation. These considerations lead to the definition of a family of approximating semantics called *Gaussian Semantics*.

Local approximation. We define Gaussian Semantics so that it is closed with respect to the class of (degenerate) GMs, meaning that, at every program location, the Gaussian Semantics of a program transforms a GM into a GM. In particular, we proceed as in the general approximation scheme proposed by Boyen and Koller (1998): given a GM G, the exact semantics of a program location would transform it in a different distribution D, which is not necessarily a GM. However, we approximate D with a new GM G_D and define the Gaussian Semantics as semantics that transform G into G_D at that program location. This process is represented in Figure 16. Performing this at every program location approximates the whole program semantics. In particular, we choose to approximate D with G_D using moment-matching, meaning that G_D is a GM having the same moments of D up to a certain order r. This is convenient for two reasons: first, it avoids computing the full pdf of D, as only its first r moments are needed to find G_D; second, since D is obtained as a transformation of a GM, it can be expressed as a linear combination of transformed Gaussians, and its moments can be computed analytically using the results summarized in Table 10.

To sum up, in Gaussian Semantics, each program location is associated with an integer r, and the semantics acts on a GM G performing two steps: first, it computes the first r order moments of the transformed distribution D, using the results in Table 10; then, it finds a new GM

Operation	**Theoretical Result**	**Computes moments for:**
Sum of Gaussians	Closed w.r.t linear transformations Billingsley, 2008	$c_1\mathcal{N}(\mu_1, \Sigma_1) + c_2\mathcal{N}(\mu_1, \Sigma_2)$
Conditioning Gaussians to $x_i == c$	Closed w.r.t. conditioning Bishop and Nasrabadi, 2006	$\mathcal{N}(\mu, \Sigma \mid x_i == c)$
Conditioning Gaussians to $x \in [a, b]$	Iterative formulas Kan and Robotti, 2017	$\mathcal{N}(\mu, \Sigma \mid x \in [a, b])$
Product of Gaussian	Isserlis' Theorem Wick, 1950	$\mathcal{N}(\mu_1, \Sigma_1)\mathcal{N}(\mu_2, \Sigma_2)$

Table 10: Summary of the theoretical results used to compute the moments of transformed Gaussian Mixtures. $\mathcal{N}(\mu, \Sigma)$ denotes a Gaussian distribution with mean μ and covariance matrix Σ, c is any real constant and a, b are vectors in $\mathbb{R}^d$ defining the hyper-rectangle $[a, b] = \{x \in \mathbb{R}^d : a_i \leq x_i \leq b_i\}$.

G_D having same moments as D up to order r. More than one moment-matching GM G_D can exist, therefore, we give a heuristic to determine a unique G_D for any r. We base our heuristics, called *max entropy matching*, on the maximum entropy principle (Kullback and Leibler, 1951).

Asymptotic correctness. Our first technical contribution is theoretical: we provide a universal approximation result stating that, under mild conditions, when the order of moments matched at each program location grows, the family of Gaussian Semantics converges to the exact probabilistic semantics. While our result exploits the well-known universal approximation power of GMs (Lo, 1972), it is a non-trivial consequence of it. The density of GMs guarantees the existence of a GM arbitrarily close to a target distribution; however, for a probabilistic program, the target distribution is generally not known. Here, we give a *constructive method* to build the approximating GM.

Implementation. Besides the definition of Gaussian Semantics, we look at how they can be practically computed. Unfortunately, it turns out that while the formulas in Table 10 allow us to compute moments up to any order, finding a moment-matching GM is a hard task. In fact, finding a moment-matching GM for moment orders higher than two requires the solution of a constrained system of polynomial equations, for which no analytical solution is known (Lasserre, 2009). Despite this, when only the first two orders of moments are matched, our matching boils down to using a single Gaussian distribution with a given mean and covariance, and no system of equations needs to be solved. We call this particular instance *Second Order Gaussian Approximation* (SOGA) and present an algorithm that implements it.

In general, the posterior computed by SOGA is a GM whose number of components grows exponentially in the number of conditional statements. To help cope with this, we introduce a pruning strategy that keeps the number of components in the GMs below a user-specified threshold by merging components with minimal cost. Using a prototype implementation, we compare SOGA on a corpus of benchmarks against state-of-the-art tools representative of different inference methods: MCMC sampling (STAN), symbolic execution (PSI), quantization (AQUA), VI (Pyro). Even when it is not the best-performing method, it still provides the flexibility to model both continuous and discrete posteriors, unlike STAN and AQUA, which only support the former. Additionally, it enables reaching numerical solutions in reasonable runtimes when PSI returns non-simplified integrals that demand computationally prohibitive times for numerical integration. When applied to the analyzed benchmarks, pruning significantly reduced the computational time without incurring noticeable approximation errors.

Importantly, we highlight that SOGA is particularly useful for performing inference on two classes of programs: those involving mixtures of continuous and discrete distributions and collaborative filtering models. Most state-of-the-art approaches do not support the first class, even though it is known that this kind of distribution arises in various application domains (Gao et al., 2017; Kharchenko, Silberstein, and Scadden, 2014; Pierson and Yau, 2015). Thanks to its GM representation, SOGA can easily encode these distributions. When tested on benchmarks introduced specifically for this problem, SOGA is able to perform inference faster than dedicated methods such as (Wu et al., 2018), while identifying the exact posterior. Collaborative filtering models are an established framework to model recommendation systems and have been extensively investigated in the machine learning community (Koren, Rendle, and Bell, 2021). SOGA can deal with a large number of variables without incurring large computational times or out-of-memory errors, as happens with alternative methods.

Application to Motivating Example. In our motivating example, at line 6, to approximate the distribution of $dist$ after the assignment $dist = x^2 + y^2$, SOGA proceeds as follows. It first computes the means and covariance matrices of x^2 and y^2 using Isserlis' theorem (Wick, 1950) (observe that x and y are Gaussian, but x^2 and y^2 are not). Then, it approximates the distributions of x^2 and y^2 with two Gaussians having the computed means and covariance matrices. Finally, it exploits the closedness of Gaussians with respect to sum to approximate the distribution of $dist$ with the sum of the Gaussians approximating x^2 and y^2. Therefore, while in the exact semantics, after line 6, $dist$ does not have a GM distribution, in SOGA it does. This significantly simplifies the subsequent computations. Indeed, when entering the if statement at line 7, $dist$ is conditioned to $dist > 10$. Performing conditioning in the exact semantics requires computing the integral of the pdf of $dist$ over the set of vectors satisfying $dist > 10$. Instead, in SOGA $dist$ is Gaussianly distributed, therefore we can compute the moments of the conditioned distribution using the formulas from Kan and Robotti (2017), and then approximate the conditioned distribution with a Gaussian having given mean and covariance matrix. Overall, for Algorithm 1 SOGA computes the output, which in this case is exact, in 0.042 s for $n = 1$ and in 0.192 s for $n = 100$, performing significantly better than BLOG and Pyro.

4.1.3 Other Related Work

In addition to the techniques already mentioned, volume computation can be quite efficient for discrete models (Filieri, Păsăreanu, and Visser, 2013; Holtzen, Van den Broeck, and Millstein, 2020); however, it cannot be applied to continuous distributions. Moreover, while all the above methods use a pdf representation of the distributions, more recently, representations using generating functions have been investigated, but only for discrete distributions (Chen et al., 2022).

The approximation-by-Gaussian approach is also common to Laplace approximation (Tierney and Kadane, 1986). Laplace approximation is a mode-matching strategy and is computationally more intense than VI, as it is based on an optimization process to find the mode. Generally, it is inferior to VI (Bishop and Nasrabadi, 2006). Another kind of Gaussian approximation is Gaussian Smoothing (Chaudhuri and Solar-Lezama, 2010; Chaudhuri and Solar-Lezama, 2011), although it does target neither probabilistic programs nor the inference problem.

Finally, our approach can be ascribed to the practice, common in many branches of mathematics, of studying *universal approximators*. One shows that a given function belonging to a certain class is shown to be approximated, arbitrarily closely, by another family of (parameterized) functions. Notable examples are polynomials (Pérez and Quintana, 2006), and neural networks (Hornik, Stinchcombe, and White, 1989; D.-X. Zhou, 2020).

4.2 Background

We now introduce some of the notation and the concepts that will be used in this and in the next chapter.

Notation. Given a Boolean value B, $\neg B$ denotes its negation. For a vector $x \in \mathbb{R}^d$, $x \setminus x_i$ denotes a vector in $\mathbb{R}^{d-1}$ obtained from x by suppressing the i-th component; $x[x_i = E(x)]$ denotes a vector $x \in \mathbb{R}^d$ in which the i-th component is replaced with the expression $E(x)$; $\|x\|$ denotes the 2-norm; $diag(d_1, \ldots, d_s)$ denotes the $\mathbb{R}^{s \times s}$ diagonal matrix having $d_1, \ldots, d_s$ as diagonal elements.

Probability Distributions. We always deal with distribution over $\mathbb{R}^d$ and use $x \sim D$ to denote that the stochastic vector x is distributed according to distribution D. We always assume that a distribution D can be specified by its probability density function (pdf) $f_D : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$. For $x \sim D$, and a set $A \subseteq \mathbb{R}^d$, the probability of A under D, denoted by $P_D(A)$, can be expressed as the Lebesgue integral $P_D(A) = \int_A f_D(x)dx$. Sometimes, we will find it more convenient to refer to the probability measure induced by D on the measurable space $(\mathbb{R}^d, \mathcal{B}(\mathbb{R}^d))$, where $\mathcal{B}(\mathbb{R}^d)$ is the Borel σ-algebra on $\mathbb{R}^d$. By probability measure, we mean a function $m : \mathcal{B}(\mathbb{R}^d) \to [0, 1]$ that satisfies the following two properties: i) $m(\emptyset) = 0$ and $m(\mathbb{R}^d) = 1$; ii) $m(\cup_{i\in\mathbb{N}} A_n) = \sum_{i\in\mathbb{N}} m(A_n)$ for any countable collection of disjoint sets $A_n \subset \mathbb{R}^d$. For a distribution D the associated measure as $m_D(A)$ is given by $m_D(A) = P_D(A) = \int_A f_D(x)\,dx$ for every $A \in \mathcal{B}(\mathbb{R}^d)$. Moreover, due to the presence of conditional branches and observe statements, we must consider distributions conditioned to subsets of $\mathbb{R}^d$. Letting $\mathbb{I}_A$ be the characteristic function of set $A \subset \mathbb{R}^d$, $D|A$ will denote the *distribution of D truncated (or conditioned) to A*, whose pdf is given by $f_{D|A} = \frac{1}{P_D(A)} f_D \mathbb{I}_A$. Observe that $f_{D|A}$ is obtained by setting f_D to 0 outside A and then by dividing it by $P_D(A)$ so that the induced measure is still a probability measure. Given a d-dimensional random vector $x \sim D$ and a subvector $x' = (x_{i_1}, \ldots, x_{i_s})$

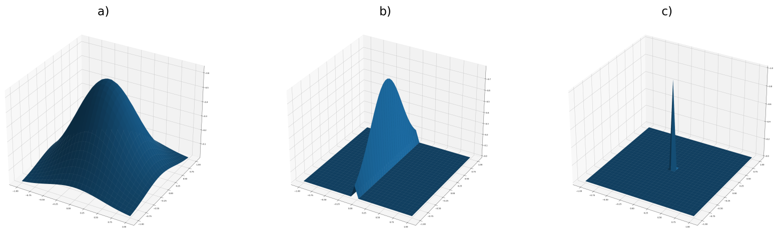


Figure 17: Plot of: a) a non-degenerate 2-dimensional Gaussian; b) a degenerate Gaussian whose covariance matrix has rank 1; c) a degenerate Gaussian with null covariance matrix (Dirac delta). Considering mixtures of possibly degenerate Gaussians, allow us to capture (mixtures of) both continuous and discrete distributions.

with $i_1, \ldots, i_s \in \{1, \ldots, d\}$, we denote by $Marg_{x'}(D)$ the marginal distribution of D, obtained integrating out the components not in x', i.e. $x' \sim Marg_{x'}(D) = \int_{\mathbb{R}^{d-s}} f_D(x) d(x \setminus x')$.

Gaussian Distributions and Mixtures. Gaussian distributions with mean μ and covariance matrix Σ are denoted by $\mathcal{N}(\mu, \Sigma)$ and have pdf:

$$f_{\mathcal{N}(\mu,\Sigma)} = \frac{1}{\sqrt{(2\pi)^d det(\Sigma)}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

If the rank of the covariance matrix is d_0 such that $0 < d_0 < d$, we can consider the following density:

$$f_{\mathcal{N}(\mu,\Sigma)} = \frac{1}{\sqrt{(2\pi)^{d_0} det^*(\Sigma)}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{+}(x-\mu)\right)$$

where det^* is the pseudo-determinant defined as ($\mathbb{I}$ is the identity matrix)

$$det^*(\Sigma) = \lim_{\alpha \to 0} \frac{det(\Sigma + \alpha\mathbb{I})}{\alpha^{d-d_0}}$$

and Σ^+ is the generalized inverse (also called Moore-Penrose pseudoinverse), defined as the matrix Σ^+ satisfying the following properties

$$\Sigma\Sigma^+\Sigma = \Sigma, \quad \Sigma^+\Sigma\Sigma^+ = \Sigma^+,$$
$$(\Sigma\Sigma^+)^T = \Sigma\Sigma^+, \quad (\Sigma^+\Sigma)^T = \Sigma^+\Sigma.$$

If the rank of the covariance matrix is 0, we interpret the Gaussian as a Dirac delta distribution centered in μ. For further details, we refer the reader to Florescu, 2014.

We refer to *mixtures* as the scalar products of two vectors $(p_1, \dots, p_C)$ and $(D_1, \dots, D_C)$ such that $\sum_{i=1}^{C} p_i = 1$, $0 < p_i \leq 1$, and D_i is the distribution of the i-th *component*, with $i = 1, \dots, C$. The numbers p_i are called *mixing coefficients*. We will denote a mixture as $M = p_1 D_1 + \dots + p_C D_C$, thus indicating that M has pdf $f_M(x) = p_1 f_{D_1}(x) + \dots + p_C f_{D_C}(x)$. When $C = 1$, we recover the case of a single distribution. A special case is given by Gaussian Mixtures (GMs) in which $D_i = \mathcal{N}(\mu_i, \Sigma_i)$ for $i = 1, \dots, C$, with mean vectors and covariance matrices μ_i, Σ_i. We assume $(\mu_i, \Sigma_i) \neq (\mu_j, \Sigma_j)$ for $i \neq j$. The set of GMs is dense in the set of probability distributions with respect to the weak topology (Lo, 1972), meaning that for any probability distribution, one can always find a GM that approximates it arbitrarily close with respect to a particular metric, the Levy-Prokhorov distance. Since we consider Dirac deltas as particular Gaussian distributions, discrete distributions over a finite set of values are included in the set of GMs.

Weak Convergence For a sequence of random vectors $X_n \sim D_n$ with cdfs F_{D_n} we say that D_n *converge weakly to D*, with F, if for every continuity point x of F (i.e. points for which $\lim_{y \to x} F(y) = F(x)$) it holds:

$$\lim_n F_{D_n}(x) = F_D(x).$$

We denote weak convergence with $D_n \xrightarrow{n \to \infty} D$. Equivalently, the corresponding measures converges weakly, denoted by $m_{D_n} \xrightarrow{n \to \infty} m_D$. Interestingly, the space of distributions with the weak topology is metrizable, i.e. we can define a metric such that weak convergence is equivalent to convergence in the metric. This metric is the Levy-Prokhorov distance that for two measures m, m' on $(\mathbb{R}^d, \mathcal{B}(\mathbb{R}^d))$ is defined as:

$$d_{LP}(m, m') = \inf\{\epsilon > 0 \mid m(A) \leq m'(A^\epsilon) + \epsilon \text{ and } m'(A) \leq m'(A^\epsilon) + \epsilon, \\ \forall\, A \in \mathcal{B}(\mathbb{R}^d)\} \qquad (4.1)$$

where for $A \subseteq \mathbb{R}^d$, $A^\epsilon = \{x \in \mathbb{R}^d \mid \exists\, y \in A \text{ s.t. } \|x - y\| \leq \epsilon\}$.

Let $B_\epsilon(x)$ be the ball of radius ϵ centered in x, defined as $B_\epsilon(x) = \{y \in \mathbb{R}^d \mid \|x - y\| < \epsilon\}$. For a set $A \subseteq \mathbb{R}^d$ we define the *interior* of A as the set:

$$int(A) = \{x \in A \mid \exists\, \epsilon > 0 \text{ s. t. } B_\epsilon(x) \subset A\}.$$

The *closure* of A as the set

$$\bar{A} = \{x \in \mathbb{R}^d \,|\, \forall\, \epsilon > 0\, B_\epsilon(x) \cap A \neq \emptyset\}.$$

The *boundary* of A as the set $\partial A = \bar{A} \setminus int(A)$.

Given a measure m on $(\mathbb{R}^d, \mathcal{B}(\mathbb{R}^d))$ and $A \in \mathcal{B}(\mathbb{R}^d)$ we say that A is an m-continuity set if $m(\partial A) = 0$.

The following theorem on weak convergence will be used to prove our main result. Observe that, in particular, if h is discontinuous on the border of an m-continuity set, Theorem 7 holds for h.

Theorem 7 (Mapping Theorem). *Suppose $h : \mathbb{R}^d \to \mathbb{R}^d$ is measurable and that the set H of its discontinuities is such that $m_D(H) = 0$. If $D_n \to D$ in the Levy-Prokhorov metric, then $h(D_n) \to h(D)$.*

Distributions Determined by Their Moments. Let $x \sim D$. For $r = (r_1, \ldots, r_d)$, with $r_i \in \mathbb{N}$, $\forall\, i$ define the r-th central moment of X as

$$\mathbb{E}[x^r] = \int_{\mathbb{R}^d} x_1^{r_1} \ldots x_d^{r_d} f_D(x) dx.$$

Letting r vary over all vectors in $\mathbb{N}^d$ such that $r_1 + \ldots + r_d = r$, we obtain the set of all r-th order central moments of D. Observe that, for any D, $\mathbb{E}[x^0] = 1$. Since the construction of our semantics relies on the Method of Moments, we need to ensure that this converges to the correct distribution. This is true only if no other distribution has all moments equal to those of the target one (Billingsley, 2008). We say that in this case, the target distribution is *determined by its moments*, formalized next.

Definition 1. *A distribution D is determined by its moments, if for any other distribution D' such that for all $r_1, \ldots, r_d \geq 0$*

$$\int_{\mathbb{R}^d} x_1^{r_1} \ldots x_d^{r_d} f_D(x) dx = \int_{\mathbb{R}^d} x_1^{r_1} \ldots x_d^{r_d} f_{D'}(x) dx$$

it holds that $m_D = m_{D'}$.

The following theorem relates the moment-matching of distributions determined by their moments with weak convergence.

Theorem 8 (Billingsley (2008)). *Suppose $X \sim D$, $X_n \sim D_n$, and D is a distribution determined by its moments, while D_n has moments of all orders. If for all $r > 0$*

$$\lim_n \mathbb{E}[X_n^r] = \mathbb{E}[X]$$

then $D_n \to D$ in the Levy-Prokhorov metric.

4.3 Syntax and Exact Probabilistic Semantics

4.3.1 Syntax

Following Kozen (1979), we will consider probabilistic programs as transformers over distributions D defined over a vector of variables taking values in $\mathbb{R}^d$. Similarly to Chaudhuri and Solar-Lezama (2011), we represent programs in a control flow-graph (cfg) syntax (P. Cousot and R. Cousot, 1977). We use as explanatory example the simple program in Algorithm 2.

A program is a directed graph $P = (V, E)$ where V is set of nodes and E is the set of edges. Specifically, we consider directed acyclic graphs (DAGs) of bounded depth. Each node belongs to one of five types in $\Gamma =$ {*entry, state, test, observe, exit*}. We denote the fact that a node $v \in V$ is of a given type $\gamma \in \Gamma$ with $v\colon \gamma$. The nodes satisfy the following properties.

- A node $v\colon \mathit{entry}$ has no incoming edge and one outgoing edge.
- A node $v\colon \mathit{state}$ has any number of incoming edges and one outgoing edge. A function *cond* is defined on the set of state nodes, such that $\mathit{cond} : \{v \in V \mid v\colon \mathit{state}\} \to \{\mathit{true}, \mathit{false}, \mathit{none}\}$ and $\mathit{cond}(v) = \mathit{none}$ if and only if the parent of v is not a test node.
- A node $v\colon \mathit{test}$ has one incoming edge and two outgoing edges toward state nodes v_1, v_2 such that $\mathit{cond}(v_1) = \mathit{true}$ and $\mathit{cond}(v_2) =$ *false*.
- A node $v\colon \mathit{observe}$ has one incoming edge and one outgoing edge.
- A node $v\colon \mathit{exit}$ has any number of incoming edges and no output edge.

Moreover, for each program, there is exactly one $v \in V$ such that $v\colon \mathit{entry}$ and one $v \in V$ such that $v\colon \mathit{exit}$, and they correspond to the root and the only leaf of the DAG representing the program, respectively. The control-flow syntax for Algorithm 2 is represented in Figure 18.

Variables are defined as

$$\mathtt{z} := \mathtt{x} \mid \mathtt{g} \qquad \mathtt{g} := \mathtt{gm}([\pi_1, \dots, \pi_\mathtt{s}], [\mu_1, \dots, \mu_\mathtt{s}], [\sigma_1, \dots, \sigma_\mathtt{s}]) \mid \mathtt{gauss}(\mu, \sigma)$$

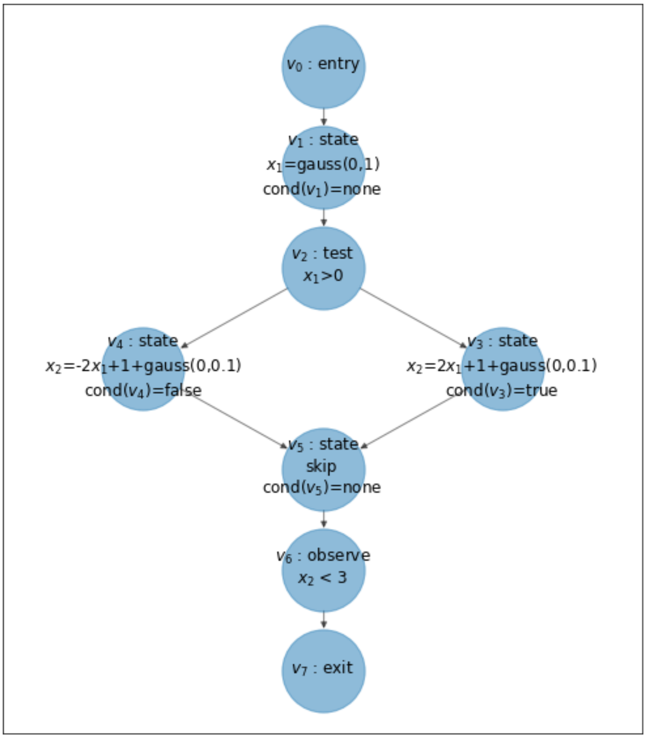


Figure 18: Control-flow graph representation of Algorithm 2

Algorithm 2 Example

```
entry                                        {$v_0$ : $entry$}
$x_1 = gauss(0, 1)$                          {$v_1$ : $state$}
if $x_1 > 0$                                 {$v_2$ : $test$}
    $x_2 = 2x_1 + 1 + gauss(0, 0.1)$         {$v_3$ : $state$}
else
    $x_2 = -2x_1 + 1 + gauss(0, 0.1)$        {$v_4$ : $state$}
end if
skip                                         {$v_5$ : $state$}
observe($x_2 < 3$)                           {$v_6$ : $observe$}
exit                                         {$v_7$ : $exit$}
```

where $\mathtt{x}$ is an *output variable*, i.e., a variable on which to compute the posterior distribution, and $\mathtt{g}$ denotes a fresh *read-only variable* distributed according to univariate GMs with mixing coefficients π_i, means μ_i and variances σ_i^2, $i = 1, \dots, s$. For the sake of brevity, in our examples, we will also use read-only variables denoted by $\mathtt{gauss}(\mu, \sigma)$ which is syntactic sugar for $\mathtt{gm}([1], [\mu], [\sigma])$. We use read-only variables to perform random assignments, as it is done in lines 2, 4, and 6 of Algorithm 2 and to encode Boolean conditions depending on arbitrary distributions.

The vector of output variables is denoted by $x = (x_1, \dots, x_d)$. The vector z augmented with read-only variables instead is denoted by $z = (x_1, \dots, x_d, g_1, \dots, g_{n-d})$. We denote the distribution of the augmented vector with D_z. We assume read-only variables are dropped after the assignment is performed or the condition is evaluated, marginalizing them out. For example, in line 4 of Algorithm 2, the assignment $x_2 = 2x_1+1+gauss(0, 0.1)$ is performed by augmenting the vector $x = (x_1, x_2)$ to $z = (x_1, x_2, g)$, with g being an independent standard Gaussian, and assigning x_2 with $2x_1 + 1 + g$. After the new posterior on z is computed, g is marginalized out, returning to the vector $x = (x_1, x_2)$.

State nodes are labelled by either skip or assignment instructions of the type $\mathtt{x_i} = \mathtt{E(z)}$ where $\mathtt{E(z)}$ is an expression of the following form:

$$\mathtt{E(z)} := \mathtt{c_1} \cdot \mathtt{z_1} + \dots + \mathtt{c_n} \cdot \mathtt{z_n} + \mathtt{c} \mid \mathtt{z_{i_1}} \cdot \mathtt{z_{i_2}} \tag{4.2}$$

where $\mathtt{c}, \mathtt{c_1}, \dots, \mathtt{c_n}$ are scalar.

Test and observe nodes are labeled by linear Boolean conditions (LBCs)

$$\begin{aligned}
x = Bernoulli(y) \quad &\rightarrow \begin{cases} z = Uniform(0,1) \\ if\, z < y \,\{x = 1\}\, else\, \{x = 0\} \end{cases} \\
x = Normal(y, z) \quad &\rightarrow \quad x = y + z \cdot Normal(0,1) \\
x = Uniform(y, z) \quad &\rightarrow \quad x = y + (z - y) \cdot Uniform(0,1) \\
x = Laplace(y, c) \quad &\rightarrow \quad x = y + Laplace(0, c) \\
x = Exponential(c/y) \quad &\rightarrow \quad x = y \cdot Exponential(c)
\end{aligned}$$

Figure 19: Reparametrizations for transforming random assignments involving distributions depending on variable parameters y, z into assignments only involving distributions with constant parameters.

of the following form:

$$\mathtt{B(z)} := \mathtt{true} \mid \mathtt{false} \mid \mathtt{c_1} \cdot \mathtt{z_1} + \ldots + \mathtt{c_n} \cdot \mathtt{z_n} \bowtie \mathtt{c} \mid \mathtt{z_i} \,\square\, \mathtt{c} \tag{4.3}$$

where $\mathtt{c}, \mathtt{c_1}, \ldots, \mathtt{c_n}$ are real scalar constants, $\bowtie \in \{<, \leq, \geq, >\}$ and $\square \in \{==, !=\}$. We associate an LBC with the set, defined on the space of augmented variables

$$[\![\mathtt{B(z)}]\!] = \{z \in \mathbb{R}^n \textit{ s.t. } \mathtt{B}(z) \textit{ holds } \}. \tag{4.4}$$

where $[\![\mathtt{true}]\!] = \mathbb{R}^n$ and $[\![\mathtt{false}]\!] = \emptyset$. Observe that an expression or LBC can have at most d output variables but any finite number of read-only variables.

4.3.2 Supported Programs

Our syntax rules out general distributions depending on non-constant parameters, unbounded loops, and non-polynomial functions. We briefly comment on the limitations of this approach, how they can be mitigated, and when they are shared by other techniques.

Probabilistic Assignments. Probabilistic assignments assign read-only variables to output variables. This is not a limitation since dependence between variables can be encoded using multiple assignments. For what concerns the restriction on GM distributions, instead, we exploit the already discussed density of GMs in the space of distribution (Lo, 1972), and assign a GM arbitrarily close to the target distribution. In this sense, we will assume that we are approximating non-GM distributions with a GM whenever we refer to non-GM distributions.

Finally, probabilistic assignments will involve only distributions depending on constant parameters. This restriction is more difficult to overcome and is shared with other tools based on moment-based techniques, such as Bartocci, Kovács, and Stankovič (2020) and Moosbrugger et al. (2022). This is because it is not always possible to derive how the moments change if one or more parameters of a distribution are probabilistic. As in Moosbrugger et al. (2022), this limitation can be mitigated by performing suitable reparametrizations, as the one reported in Figure 19.

Iterations. We restrict our attention to loops bounded by deterministic constants (as in our illustrating example in Algorithm 1), similarly to Gehr, Misailovic, and Vechev (2016), Huang, Dutta, and Misailovic (2021), Holtzen, Van den Broeck, and Millstein (2020), Albarghouthi et al. (2017) and Nori et al. (2014). If guarantees on almost sure termination can be given, the true distribution of the loop can be approximated by a bounded unrolled loop with a sufficiently large number of iterations (Kozen, 1979).

Polynomial Programs. Differently from Gehr, Misailovic, and Vechev (2016), Huang, Dutta, and Misailovic (2021), and Carpenter et al. (2017), we consider programs involving only the arithmetic operations $+, -, ^*$, and ^. This assumption is common to other approaches relying on moment-based techniques such as Bartocci, Kovács, and Stankovič (2020) and Moosbrugger et al. (2022), due to the fact that non-polynomial functions (such as the logarithm) may generate distributions that are not determined by their parameters. We remark that from expressions such as (4.2) and (4.3), general polynomial expressions and boolean conditions can be obtained, respectively, by chaining state nodes and nesting conditional statements. A probabilistic choice, i.e., x is assigned e_1 with probability p or e_2 with probability $1-p$, is encoded using the LBC $y < q$ where y is a standard Gaussian and q is the Gaussian p-quantile.

4.3.3 Exact Probabilistic Semantics

The "exact" semantics follows Kozen's *Semantics 2* (Kozen, 1979). Since we are using the control-flow syntax of P. Cousot and R. Cousot (1977), we are close to the *collecting semantics* in Chaudhuri and Solar-Lezama (2011): we combine the semantics of the nodes to define the semantics of the paths, then define the semantics of the program as a sum over the semantics of the paths. This semantics is particularly convenient for our method because it gives the posterior distribution as a mixture, similarly to Y. Zhou et al. (2020).

Given a program $P = (V, E)$, we define a path π as an ordered sequence of nodes $\pi = v_0 \cdots v_n$ with $v_0\colon entry$, $v_n\colon exit$ and $(v_{i-1}, v_i) \in E$, $\forall\, i = 1, \dots n$. The successor of node v_i in path π is denoted as $s_\pi(v_i) = v_{i+1}$. The set of all paths of P is denoted by Π^P. We define the semantics of a path π, denoted by $[\![\pi]\!]$, as a pair (p, D), where $p \geq 0$ and D is a probability distribution on $\mathbb{R}^d$. The semantics $[\![\pi]\!]$ composes the semantics of the nodes along path π, i.e., $[\![\pi]\!] = [\![v_n]\!]_\pi \circ \dots \circ [\![v_0]\!]_\pi$. The semantics of each node is defined as follows.

- The *entry* node outputs the pair $(1, \delta_0)$, δ_0 being a Dirac delta centered on the zero vector:

$$\text{if } v\colon entry, [\![v]\!]_\pi = (1, \delta_0).$$

- A *state* node v takes as input a pair (p, D) and returns a pair (p, D') depending on its label. If it is labelled by `skip`, it returns (p, D). If it is labelled by $\mathtt{x_i} = \mathtt{E(z)}$, it returns (p, D') with D' the distribution of the vector $x[x_i = E(z)]$:

$$\text{if } v\colon state, [\![v]\!]_\pi(p, D) = \begin{cases} (p, D) & \text{if } v \text{ is labelled by } \mathtt{skip} \\ (p, D') & \text{if } v \text{ is labelled by } \mathtt{x_i} = \mathtt{E(z)}. \end{cases}$$

- A *test* node v labelled by $\mathtt{B(z)}$ takes as input a pair (p, D) returns (p', D') depending on the value of $cond(s_\pi(v))$. In particular, first, the augmented vector z and its distribution D_z are considered. If $cond(s_\pi(v)) = true$, then the node computes the probability of the boolean condition evaluating to true, i.e. $P_{D_z}([\![\mathtt{B(z)}]\!])$. Then, it conditions the current distribution to such event, i.e. $D_z \mid [\![\mathtt{B(z)}]\!]$. The result is the output pair $(p \cdot P_{D_z}([\![\mathtt{B(z)}]\!]), Marg_x(D_z \mid [\![\mathtt{B(z)}]\!]))$. Similarly, if $cond(s_\pi(v)) = false$ the output of the node is the pair $(p \cdot P_{D_z}([\![\neg\mathtt{B(z)}]\!]), Marg_x(D_z \mid [\![\neg\mathtt{B(z)}]\!]))$. To overcome conditioning with respect to zero-probability events we assume that whenever

$P_{D_z}(\llbracket \mathtt{B(z)} \rrbracket) = 0$ (resp.$P_{D_z}(\llbracket \neg\mathtt{B(z)} \rrbracket) = 0$) the output pair is $(0, D)$:

$$\text{if } v\colon \mathit{test}, \llbracket v \rrbracket_\pi(p, D) = \begin{cases} (p \cdot P_{D_z}(\llbracket \mathtt{B(z)} \rrbracket), \mathit{Marg}_x(D_z \mid \llbracket \mathtt{B(z)} \rrbracket)) \\ \qquad \mathit{cond}(s_\pi(v)) = \mathit{true} \wedge P_{D_z}(\llbracket \mathtt{B(z)} \rrbracket) \neq 0 \\ (p \cdot P_{D_z}(\llbracket \neg\mathtt{B(z)} \rrbracket), \mathit{Marg}_x(D_z \mid \llbracket \neg\mathtt{B(z)} \rrbracket)) \\ \qquad \mathit{cond}(s_\pi(v)) = \mathit{false} \wedge P_{D_z}(\llbracket \neg\mathtt{B(z)} \rrbracket) \neq 0 \\ (0, D) & \mathit{else}. \end{cases}$$

- For an *observe* node v labelled by $\mathtt{B(z)}$ we condition the current distribution to $\llbracket \mathtt{B(z)} \rrbracket$. Observe that if $\mathtt{B(z)}$ only contains read-only variables, conditioning does not affect the distribution of the output variables x. If $B(z) = \mathtt{x_i} == \mathtt{c}$ the node returns a distribution D' having pdf $\frac{1}{I} f_D(x, x_i = c)\delta_c(x_i)$ with $I = \int_{\mathbb{R}^{d-1}} f_D(x, x_i = c) d(x \setminus x_i)$. In all other cases conditioning is treated as usual:

$$\text{if } v\colon \mathit{observe}, \llbracket v \rrbracket_\pi(p, D) = \begin{cases} (p \cdot I, D') & B(z) = \mathtt{x_i} == \mathtt{c} \\ (p \cdot P_{D_z}(\llbracket \mathtt{B(z)} \rrbracket), D \mid \llbracket \mathtt{B(z)} \rrbracket) \\ \qquad B(z) \neq \mathtt{x_i} == \mathtt{c} \wedge P_{D_z}(\llbracket \mathtt{B(z)} \rrbracket) > 0 \\ (0, D) & \mathit{else}. \end{cases}$$

- The *exit* node v takes as input (p, D) and outputs the same pair (p, D):
$$\text{if } v\colon \mathit{exit}, \llbracket v \rrbracket_\pi(p, D) = (p, D).$$

The semantics of the program P is then defined as:

$$\llbracket P \rrbracket = \sum_{\substack{(p,D) = \llbracket \pi \rrbracket : \\ \pi \in \Pi^P}} \frac{p}{\sum_{\substack{(p',D') = \llbracket \pi \rrbracket \\ \pi \in \Pi^P}} p'} D. \tag{4.5}$$

Example 5. *For the program in Algorithm 2, we have only two paths, $\pi_T = v_0v_1v_2v_3v_5v_6v_7$ and $\pi_F = v_0v_1v_2v_4v_5v_6v_7$ corresponding to evaluations of the conditional statement as true or false, respectively. To compute the semantics of π_T we start from v_0, that outputs $(1, \delta_0)$. This pair is taken as input by v_1, which is a state node assigning $gauss(0, 1)$ to x_1, so its output is $(1, \mathcal{N}(0, \Sigma_1))$ with $\Sigma_1 = diag(1, 0)$ (corresponding to the distribution in Figure 2b). This pair is taken as input by the test node v_2. Since we are considering π_T, for which $s_{\pi_T}(v_2) = v_3$, and $cond(v_3) = true$, the semantics of v_2 in this path* *conditions $\mathcal{N}(0,\Sigma_1)$ to $x_1 > 0$. Therefore in this path the output of v_2 is $(0.5, \mathcal{N}(0,\Sigma_1)|[\![x_1 > 0]\!])$, where $0.5 = P_{\mathcal{N}(0,\Sigma_1)}([\![x_1 > 0]\!])$. This pair is taken as input by v_3, which updates the distribution of x_2 and therefore outputs a new pair $(0.5, D_3)$. We can proceed until we compute the output of v_7, which gives the final pair $[\![\pi_T]\!] = (p_T, D_T)$. In the same way, we compute $[\![\pi_F]\!] = (p_F, D_F)$, and finally, the semantics of the whole program as the mixture $(p_T D_T + p_F D_F)/(p_T + p_F)$.*

4.4 Gaussian Semantics

Gaussian Semantics is a family of semantics closed with respect to GMs. Each node takes as input and returns a GM over the program variables. This is done by composing the exact semantics of a node with an operator T_r^{GM} acting on the output distribution of $[\![v]\!]_\pi$. In particular, T_r^{GM} transforms any distribution D into a GM G, having the same moments of D up to order r. Therefore, we call T_r^{GM} the *moment-matching operator*. Formally, we use a map $R : V \to \mathbb{N}_0$ to associate each node $v \in V$ with the highest order of moments that will be matched at v. The semantics of a node is then:

$$[\![v]\!]_\pi^R = \begin{cases} [\![v]\!]_\pi & \text{if } v\colon \mathit{entry}, \mathit{exit} \\ \left(\mathbb{I}, T_{R(v)}^{GM}\right) \circ [\![v]\!]_\pi & \text{if } v\colon \mathit{state}, \mathit{test}, \mathit{observe} \end{cases} \tag{4.6}$$

where $\mathbb{I}$ is the identity acting on the first element of the pair (p, D), and $T_{R(v)}^{GM}$ is the moment-matching operator. The Gaussian Semantics of paths and programs are defined similarly to the exact semantics as:

$$[\![\pi]\!]^R = [\![v_n]\!]_\pi^R \circ \ldots \circ [\![v_0]\!]_\pi^R \quad \text{and} \quad [\![P]\!]^R(D_0) = \sum_{\substack{(p,D)=[\![\pi]\!]^R \\ \pi\in\Pi^P}} \frac{p}{\sum_{\substack{(p',D')=[\![\pi]\!]^R \\ \pi\in\Pi^P}} p'} D.$$

Example 6. *We have seen in Algorithm 2 that the exact semantics is not closed with respect to GMs. For example, node v_2 takes as input a Gaussian but returns a truncated Gaussian, which is not a GM. If, instead, we consider Gaussian Semantics with $R(v_2) = 3$, the output of v_2 will be a GM matching the first three order moments of $\mathcal{N}(0,\Sigma_1) \,|\, [\![x > 0]\!]$. Two steps are required to compute $[\![v_2]\!]_\pi^R$. In the first step, we compute the first $R(v_2) = 3$ order moments of the output distribution of $[\![v_2]\!]_\pi$. Therefore, we compute the first order moments $\mathbb{E}[x_1] = 0.7979$ and $\mathbb{E}[x_2] = 0$, the second order* *moments $\mathbb{E}[x_1^2] = 1, \mathbb{E}[x_1 x_2] = \mathbb{E}[x_2^2] = 0$ and the third order moments $\mathbb{E}[x_1^3] = 1.5958, \mathbb{E}[x_1^2 x_2] = \mathbb{E}[x_1 x_2^2] = \mathbb{E}[x_2^3] = 0$. Observe that, thanks to the results in Table 10, this is significantly easier than computing the whole pdf of the output distribution. The second step involves finding a GM having the computed moments. This is generally more complex and is performed by the operator T_r^{GM}. In the rest of the section, we will assume that the computation of the moments is done using the aforementioned formulas, and we focus on the definition of the operator T_r^{GM} and the derivation of its properties.*

Moment-Matching Operator In general, the operator $T_r^{GM}(D)$ acts on distributions $D = \sum_{i=1}^{C} p_i D_i$ that are mixtures (possibly of a single component). The moments of D are computed as a linear combination of the moments of its components: if $x \sim D = \sum_{i=1}^{C} p_i D_i$ and $x_i \sim D_i$, then $\mathbb{E}[x^n] = \sum_{i=1}^{C} p_i \mathbb{E}[x_i^n]$. Therefore, when computing the moments of D, we first compute the moments of every component D_i. Then, it makes sense to define $T_r^{GM}(D)$ so that when it acts on a mixture D ($C > 1$), it recursively acts on each component of the mixture, moment-matching each of them. When, instead, T_r^{GM} acts on a non-mixture distribution D ($C = 1$), it returns a GM having moments up to order r equal to those of D. This second action is encoded by a second operator $\mathbf{match}_r$.

$$T_r^{GM}(D) = \begin{cases} \sum_{i=2}^{C} p_i T_r^{GM}(D_i) & \text{if } D = \sum_{i=1}^{C} p_i D_i \text{ and } C > 1 \\ \mathbf{match}_r(D) & \text{otherwise.} \end{cases} \tag{4.7}$$

We require that $\mathbf{match}_r(D)$ satisfies the following two conditions:

R1) for any distribution D, $\mathbf{match}_r(D)$ is a GM;

R2) $\mathbf{match}_r(D)$ has central moments up to order r equal to those of D.

The existence of the operator $\mathbf{match}_r$ is guaranteed by the following result, derived from Schmüdgen (2017, Theorem 17.2), stating that that, for any finite sequence of moments, there exists a moment-matching discrete distribution putting positive mass on a number of points smaller than or equal to the number of matched moments. Since discrete distributions are GMs, the Proposition holds.

Proposition 5. *For any $r \in \mathbb{N}_0$, there exists an operator $\mathbf{match}_r$ satisfying R1 and R2.*

Proof. We need to show that for any distribution D we are able to find a GM matching the first r-th order moments of D. To do this, consider a d-dimensional random variable $x \sim D$ and let $r \in \mathbb{N}_0$ be fixed. Let us define the set

$$\mathcal{N}(r) = \left\{ \alpha = (\alpha_1, \dots, \alpha_d) : \sum_{i=1}^{d} \alpha_i \leq r \right\}$$

and the associated *truncated moment sequence* $(s_\alpha)_{\alpha \in \mathcal{N}(r)}$, with $s_\alpha = \mathbb{E}[X^\alpha]$. By Theorem 17.2 in Schmüdgen, 2017, there exists a C-atomic positive measure (i.e. a discrete measure placing positive probability mass on C points), with $C \leq |\mathcal{N}(r)|$, such that

$$\int_{\mathbb{R}^d} x^\alpha dm = s_\alpha \, \forall \, \alpha \in \mathcal{N}. \tag{4.8}$$

Since $\mathcal{N}(r)$ is finite for any r, there exist a C-atomic measure m_r satisfying (4.8) for every $\alpha \in \mathcal{N}(r)$ for any truncated moment sequence $(s_\alpha)_{\alpha \in \mathcal{N}(r)}$. Moreover, since $s_0 = 1$, m_r is a probability measure, and therefore it is induced by a finite mixture of Dirac deltas. Since any finite mixture of Dirac deltas is a GM, the proof concludes. □

Example 7. *In our example, we want to match a total of* 10 *moments, a zeroth-order moment (which is always 1), two first-order, three second-order, and four third-order moments. Theorem 17.2 in Schmüdgen (2017) ensures that there exists at least one discrete distribution (therefore a GM) with* $C \leq 10$ *components that has exactly the given moments.*

Proposition 5 ensures the existence of at least one GM matching the moments of D up to order r. In general, letting $GM_r(D)$ denote the set of all finite GMs matching the moments of D up to order r, we may have that $|GM_r(D)| > 1$. For $\mathbf{match}_r$ to be well-defined, we need to uniquely identify a moment-matching GM in $GM_r(D)$. This can be done using different heuristics: we propose one based on the principle of maximum entropy, which we call the max entropy matching (MEM).

Max Entropy Matching MEM can be summed up as follows: whenever $|GM_r(D)| > 1$ we choose the GM G having the least number of components (in order to minimize the number of parameters to be fit) and minimizing a certain cost function. Any remaining tie is resolved by comparing the vectors of parameters P that identify the GMs with respect to the lexicographic ordering detailed below. We select our cost function as the sum of the opposite of the differential entropy plus a penalty term, where the differential entropy for a distribution D with pdf f_D is defined as (Cover, 1999)

$$H(D) = -\int_{\mathbb{R}^d} f_D(x) \log(f_D(x))\, dx. \tag{4.9}$$

Intuitively, the *principle of maximum entropy* asserts that the distribution maximizing entropy is the one that minimizes the number of assumptions on the distribution (Cover, 1999). Therefore, by maximizing $H(D)$, we are, in some sense, choosing the most general moment-matching distribution. We add to $H(D)$ a penalty term to avoid uncontrolled growth of the parameter values.

For inducing a lexicographic ordering on GMs, consider a Gaussian mixture $p_1 D_1 + \ldots + p_C D_C$ where $D_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, $i = 1, \ldots, C$. Since it is uniquely identified by its parameters we can order them in a vector $(p_i, \mu_i, \Sigma_i)_{i=1,\ldots,C}$ in the following way:

- $p_1 \geq p_2 \geq \cdots \geq p_C$;
- if $p_i = p_{i+1}$ then either of the following two conditions holds:
  - there exists $j \in \{1, \ldots, d\}$ such that $\mu_i(s) = \mu_{i+1}(s) \,\forall\, s < j$ and $\mu_i(j) > \mu_{i+1}(j)$ (means are ordered according to the lexicographic order);
  - if $\mu_i = \mu_{i+1}$ there exists $j \in \{1, \ldots, d^2\}$ such that $\Sigma_i(s) = \Sigma_{i+1}(s) \,\forall\, s < j$ and $\Sigma_i(j) > \Sigma_{i+1}(j)$, where Σ is converted into a vector using lexicographic ordering, i.e.

$$\Sigma = (\Sigma(0,0), \ldots, \Sigma(0,c^*), \Sigma(1,0), \ldots, \Sigma(1,c^*), \ldots, \Sigma(c^*,c^*))$$

This procedure allows us to consider a set of parameters P as a vector

$$\begin{aligned} P = (&p_1, p_2, \ldots, p_C, \mu_1(0), \mu_1(1), \ldots, \mu_C(d), \\ &\Sigma_1(0,0), \ldots, \Sigma_1(d,d), \ldots, \Sigma_C(0,0), \ldots, \Sigma_C(d,d)). \end{aligned}$$

For two set of parameters P_1, P_2 we say that $P_1 \succ P_2$ if P_1 is greater then P_2 according to the lexicographic ordering, i.e. if exists i such that $P_1(j) = P_2(j) \,\forall j < i$ and $P_1(i) > P_2(i)$. Observe that $\succ$ is a total order, i.e. if $P_1 \neq P_2$ necessarily either $P_1 \succ P_2$ or $P_2 \succ P_1$.

Then, the procedure to compute $\mathbf{match}_r(D)$ is the following.

1) Find $c^* = \min\left\{c : \exists\, G = \sum_{s=1}^{c} p_s \mathcal{N}(\mu_s, \Sigma_s) \in GM_r(D)\right\}$.

2) Find the set $\mathcal{P}$ such that $P = (p_1, \dots, p_{c^*}, \mu_1, \dots, \mu_{c^*}, \Sigma_1, \dots, \Sigma_{c^*}) \in \mathcal{P}$ if and only if the GM with parameters P matches the moments of D up to the r-th order.

3) Find the set $\mathcal{P}^*$ given by:

$$\mathcal{P}^* = \arg\min_{\mathcal{P}} \left\{ -H\left(\sum_{i=1}^{c^*} p_i \mathcal{N}(\mu_i, \Sigma_i)\right) + \sum_{i=1}^{c^*} \left(\|\mu_i\|_2^2 + \|\Sigma_i\|_2^2\right) \right\}. \tag{4.10}$$

4) If $|\mathcal{P}^*| > 1$ choose $P^* \in \mathcal{P}^*$ maximum with respect to $\succ$.

The following proposition guarantees that MEM leaves us with a well-defined operator $\textbf{match}_r$. It is again proved using Theorem 17.2 from Schmüdgen (2017), and by noticing that $\mathcal{P}$ is a compact set, therefore Eq. (4.10) is well-defined.

Proposition 6. *For any D and r the max entropy matching uniquely identifies $\textbf{match}_r(D)$.*

Proof. By Proposition 5 $GM_r(D)$ is non-empty, and by Theorem 17.2 in Schmüdgen, 2017 there is at least one moment-matching mixture such that $C < |\mathcal{N}(r)|$, therefore c^* is well-defined. Once c^* is fixed, the set of parameters $\mathcal{P}$ is the set of solutions of a system of polynomial equations equating the moments of the mixture of c^* components, expressed as functions of p_i, μ_i and Σ_i, to the moments of D (S. Wang, A. T. Chaganty, and Liang, 2015). Being the set of solutions of a polynomial system, $\mathcal{P}$ is closed. Moreover, since by Example 12.2.8 in Cover, 1999 for fixed moments the entropy is bounded from above, $-H$ is bounded from below, and we can choose $M > 0$ so that $\mathcal{P}^* \subseteq \mathcal{P} \cap \left([0,1]^{c^*} \times [0, M]^{dc^* + \frac{1}{2}d(d+1)c^*}\right)$. Then, $\mathcal{P}^*$ is compact. Finally, the maximum with respect to the lexicographic ordering can be seen as maximizing projections of the vector of parameters P on different coordinates in a given order. Since $\mathcal{P}^*$ is compact, the set of maximals with respect to the lexicographic ordering is non-empty, but since the lexicographic ordering is a total order the set of maximals can have only one element which is uniquely defined. □

We remark that the choice of MEM is arbitrary, as other cost functions could be introduced. However, it has various benefits. *(i)* To guarantee that Proposition 6 holds, one needs a bounded cost function. *(ii)* Using entropy leads to a parallelism with VI: SOGA itself can be seen as a form of VI since it involves the minimization of the reverse differential entropy (Kullback and Leibler, 1951). However, correspondence with VI is lost when higher-order moments are considered because the minimizer of the reverse differential entropy is not analytically expressible for GMs. *(iii)* In the spirit of minimizing the number of assumptions on the approximating distribution, the approach looks more pleasing mathematically.

Example 8. *While Schmüdgen (2017) ensures that we can find a moment-matching GM with 10 components, it is easy to check that $c^* = 2$ is the minimum number of components required to match three order moments. In fact, $c^* > 1$, since for a single Gaussian, given the mean and the covariance matrix, all the other moments are fixed (so, we can match the first two order moments but not the third). For $c = 2$ instead we can consider the GM G with pdf $p\mathcal{N}(m, S) + (1-p)\mathcal{N}(m', S')$ such that m, m', S, S' satisfy the system:*

$$\begin{aligned}
&pm_i + (1-p)m'_i = \mathbb{E}[x_i] && \textit{for } i = 1,2\\
&p(m_i^2 + S_{i,i}) + (1-p)(m_i'^2 + S'_{i,i}) = \mathbb{E}[x_i^2] && \textit{for } i = 1,2\\
&p(m_1 m_2 + S_{1,2}) + (1-p)(m'_1 m'_2 + S'_{1,2}) = \mathbb{E}[x_1 x_2] &&\\
&p(m_i^3 + 3m_i S_{i,i}) + (1-p)(m_i'^3 + S'_{i,i}) = \mathbb{E}[x_i^3] && \textit{for } i = 1,2\\
&p(m_i S_{j,j} + 2m_j S_{i,j} + m_i m_j^2) + &&\\
&\quad + (1-p)(m'_i S'_{j,j} + 2m'_j S'_{j,j} + m'_i m_j'^2) = \mathbb{E}[x_i x_j^2] && \textit{for } i,j = 1,2, i \neq j\\
&0 < p < 1, \quad S_{1,1}S_{2,2} - S_{1,2}^2 \geq 0, \quad S'_{1,1}S'_{2,2} - S_{1,2}'^2 \geq 0 &&
\end{aligned}$$

In the system, we equate the moments of G (l.h.s) with those of $x \sim D$ (r.h.s, computed in Example 6). Moreover, we look for solutions such that $0 < p < 1$ and S, S' are positive semidefinite (last line). Since the system is polynomial, using SMT solvers over reals we can check it is satisfiable; therefore, $c^* = 2$. Now, we should determine the set $\mathcal{P}$ of all solutions and find those that minimize the cost function. Since finding all solutions is generally impossible, we directly proceed to optimize our cost function numerically, constraining the variables to satisfy the previous system. We find the approximate solution $p = 0.572$, $m = (0.471, 0)$, $m' = (1.236, 0)$ and $S = diag(0.066, 0)$, $S' = diag(0.426, 0)$. The approximating GM is shown in the green line of Figure 20, while the blue line shows the true non-Gaussian distribution.

The example shows that the difficult step in computing a Gaussian Semantics of arbitrary order is 2). Finding the parameters of a moment-matching GM requires the solution of a system of polynomial equations, like the one in the example. This problem is notoriously hard to solve, as no analytical solution exists (Lasserre, 2009). Performing numerical optimization can solve the problem approximately, but is in general numerically unstable and requires relatively long computational times (in our example, using scipy (Virtanen et al., 2020), it took sround 7 s to match a single Gaussian!). While we leave open the problem of solving 2) efficiently in the general case, the following lemma gives two important properties of $\mathbf{match}_r$, which will be used to derive our second-order approximation.

Lemma 1. *The following two properties hold:*

i) when $r = 2$, $\boldsymbol{match}_2(D)$ is a single Gaussian variable with mean and covariance matrix equal to those of D;

ii) if D is Gaussian, for any $r \geq 2$ $\boldsymbol{match}_r(D) = D$.

Proof. The first point follows observing that, since we want to match the first two order moments, a single Gaussian variable can be used, so $c^* = 1$. Moreover, since we have a single component with mean and covariance matrix fixed, the set $\mathcal{P}$ has a single set of parameters, and MEM reduces to approximating D with a Gaussian having the same mean and covariance matrix.

On the other hand, if D is Gaussian, for $r \geq 2$ we always have $c^* = 1$, and the set $\mathcal{P}$ has a single set of parameters, so $\mathbf{match}_r(D) = D$. □

We conclude the section with a consequence of Lemma 1. It follows from ii) that T_r^{GM} has the desirable property of leaving GMs unaltered, i.e. if M is a GM $T_r^{GM}(M) = M$ for all $r \geq 2$. As a consequence, Gaussian Semantics coincides with the exact semantics for programs only involving GMs, and in particular, for programs involving only discrete distributions (mixtures of deltas).

Proposition 7. *Let $P = (V, E)$ be such that every read-only variable in the program is a finite discrete distribution. Then, for any R, $[\![P]\!]^R = [\![P]\!]$.*

Proof. Since truncations, linear combinations and products of discrete distributions are discrete distributions, only discrete distributions are generated in the execution of $[\![P]\!]$. By Lemma 1 applying the moment-matching operator T_r^{GM} to them leaves them unaltered, so the conclusion follows. □

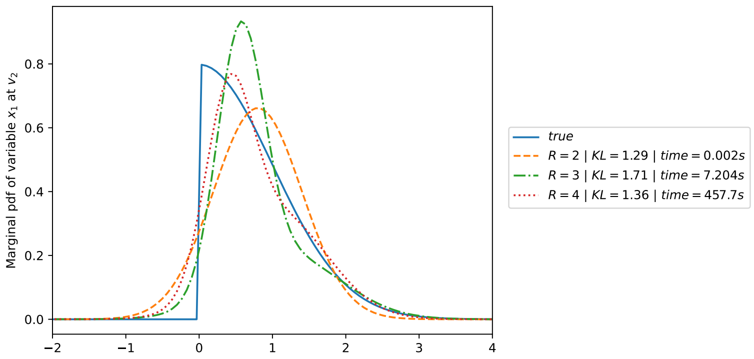


Figure 20: Marginal pdf of x_1 at node v_2 in Algorithm 2 given by exact semantics ($true$) and the Gaussian Semantics with $R = 2, 3$ and 4. In the legend, we report the KL divergence with the true distribution and the time needed to compute the approximating GM.

Example 9. *Consider a second Gaussian semantics that maps v_2 to $R(v_2) = 2$. In this case, we want to match only the first two order moments, namely $\mathbb{E}[x_1], \mathbb{E}[x_2], \mathbb{E}[x_1^2], \mathbb{E}[x_2^2], \mathbb{E}[x_1x_2]$. As noticed before, in this case $c^* = 1$, since we can take the Gaussian $\mathcal{N}(\mu, \Sigma)$ with $\mu = (0.7979, 0)$ and $\Sigma = diag(1, 0)$ and it will have required moments. Observe that we do not need to solve any system or optimization problem.*

In general, for a fixed number of moments matched, we expect Gaussian Semantics to approximate reasonably well the matched moments but not necessarily the whole distribution (see Section 5.2.2 for further discussion). Indeed, let us compare the exact posterior distribution with the ones obtained distribution when $R = 2, 3, 4$, respectively, using Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951). The KL divergence between two distributions P and Q is a standard way to evaluate the error committed in approximating P with Q. In our case, we take as P the truncated Gaussian and as Q the GMs obtained matching different order moments. The respective values are 2.29 (R=2), 1.71 (R=3) and 1.36 (R=4), so the higher-order Gaussian semantics indeed improves the approximation. Figure 20 compares the true marginal pdf of x_1 at node v_2 (blue solid line) with the second- (orange dashed), third- (green dash-dotted) and fourth (red dotted) approximations. The advantages of fitting only a finite number of moments are mainly computational. Indeed, it can be seen from the legend that as the number of moments matched grows, increasing KL accuracy comes at increasing computational cost.

4.5 Universal Approximation Theorem

Our main convergence result states that, for well-behaved programs, it is possible to find a map R so that the output distribution yielded by the semantics $[\![\cdot]\!]^R$ is arbitrarily close to that of $[\![\cdot]\!]$ in the Levy-Prokhorov metric (Ethier and Kurtz, 2009b). By "well-behaved" we mean that the distributions in the exact semantics are determined by their moments and that they induce measures m such that sets in the form (4.4) are m-continuity sets. Formally,

Theorem 9. *Assume that $P = (V, E)$ is a program such that for each $v \in V$ and each path $\pi \in \Pi^P$ the output distribution D of $[\![v]\!]_\pi$ satisfies the following:*

H1) D is determined by its moments;

H2) if D is the input distribution for a test or observe node v', then the set defined by the LBC labelling v' is an m_{D_z}-continuity set.

Then there exists a sequence of maps $(R^k : V \to \mathbb{N}_0)_{k \in \mathbb{N}}$ such that:

$$[\![P]\!]^{R^k} \xrightarrow{k \to \infty} [\![P]\!]. \tag{4.11}$$

where the convergence is intended in the weak topology, or equivalently, in the Levy-Prokhorov metric.

4.5.1 Satisfaction of the Hypotheses

Before giving an outline of the proof, we briefly comment on the hypotheses.

First, observe that H1 and H2 are sufficient but not necessary. In particular, if the hypotheses of Proposition 7 are satisfied, convergence holds trivially, even when H1 or H2 are violated.

Hypothesis H1 is common to other works considering moment-based approximation, such as in Bartocci, Kovács, and Stankovič (2020) and Moosbrugger et al. (2022) and is needed to guarantee that the method of moments converges to the true distribution (Billingsley, 2013). For a given program, it is possible to perform static analysis to check whether the arising distributions are determined by their moments, exploiting known results on moment determinacy (see, for example, the moment-generating function characterization in Billingsley, 2013). Notably, to apply these results is not necessary to compute the exact pdf of the arising distributions, but it is sufficient to keep track of their type. In fact, for some classes of distributions, moment-determinacy is established: this is true for finite discrete distributions, Gaussians, uniforms, Poissons, exponentials, truncations and mixtures thereof (Billingsley, 2013). The case studies analyzed in this thesis feature such distributions. On the contrary, log-normal distributions are not determined by their moments. However, as long as moments are computable, Gaussian Semantics can still be applied: in this case no formal guarantee of convergence towards the true distribution is given, but the method still provides an analytical approximation for the moments.

Hypothesis H2 guarantees that when distributions are conditioned to sets in the form (4.4), weak convergence is preserved. This requirement can be falsified if D has degenerate components that place positive probability mass on the boundary of the set defined by the LBC. This could happen, for instance, if a component is a Dirac measure centered on any point of $\partial[\![\mathtt{B}(\mathtt{x})]\!]$. For example, consider line 12 of *Tracking_n* in Section 4.1.1, where *out* can be 1 or 0 with probability > 0. This falsifies H2. However, such cases can be statically detected, and the program can be transformed into one that uses the equivalent condition as $out > 0.5$, so that H2 holds. More in general, continuity corrections such as those performed in Laurel and Misailovic (2020) can be adopted.

Example 10. *Algorithm 2 satisfies H1 since the joint at each location is either a mixture of Gaussians or truncated Gaussians, for which moment-determinacy is known. Moreover, the two LBCs checked in the program are $x_1 > 0$ (line 3) and $x_2 < 3$ (line 9). Before checking $x_1 > 0$, x_1 has non-degenerate Gaussian distribution, and therefore the set $x_1 = 0$ (border of $[\![x_1 > 0]\!]$) has measure 0. Similarly, before line 9, x_2 is Gaussian-distributed with $\sigma > 0$, therefore $x_2 = 3$ (border of $[\![x_2 < 3]\!]$) has again probability 0. We conclude that also H2 is verified.*

Algorithm 1 in Section 4.1.1, each marginal is obtained by Gaussians, performing sums, squares, or conditioning. Here, to check moment-determinacy, we use a result in Billingsley (2008), which states that if the moment-generating function (mgf) of a distribution is defined in an interval of 0, then the distribution is determined by its moments. Using symbolic integration, we can compute the mgf of the product of two Gaussians and verify that it is defined in an interval of 0. Therefore, the distribution is determined by its moments and H1 holds. For H2, we have already shown how to correct the condition in line 12 so that H2 is verified. For the if statement in line 7, observe that before entering it, the marginal w.r.t. to $dist$ is the distribution of $x^2 + y^2$, which is continuous, and therefore the point 10 has measure 0 with respect to it. Therefore also H2 holds.

4.5.2 Preliminary Results

The proof of the main theorem relies on two auxiliary results: first, we show that the exact semantics preserves weak convergence (Lemma 2); second, we prove that, given a weakly converging sequence of distributions D_n, it is always possible to choose a sequence of integers r_n such that $T^{GM}_{r_n}(D_n)$ converges to the same limit (Lemma 3).

Lemma 2. *Let P be a program, $\pi = (v_0, \dots, v_n) \in \Pi^P$ and $v_i \in \pi$ be fixed. Suppose (p_n, D_n) is a sequence of pairs such that the following conditions are satisfied:*

- $0 \leq p_n \leq 1 \,\forall\, n$ *and* $p_n \xrightarrow{n\to\infty} p$ *in* $\mathbb{R}$*;*
- $D_n \xrightarrow{n\to\infty} D$*;*
- *D is determined by its moments;*
- *if v_i : $test$ or v_i : $observe$ and v_i is labelled by an LBC defining the set* $[\![\mathtt{B(z)}]\!]$, $[\![\mathtt{B(z)}]\!]$ *is an m_{D_z}-continuity set;*
- $[\![v_i]\!]_\pi(p, D) = (p', D')$ *such that* $p' \neq 0$.

Then $[\![v_i]\!]_\pi(p_n, D_n) \to [\![v_i]\!]_\pi(p, D)$.

Proof. Let us consider separately the possible types of v_i.

If v_i : $entry, exit$ there is nothing to prove.

Suppose v_i : $state$. If v_i is indexed by `skip`, there is nothing to prove. If it is indexed by an assignment instruction $\mathtt{x_j} = \mathtt{E(z)}$ the conclusion follows from Theorem 7 with $h : \mathbb{R}^n \to \mathbb{R}^d$ such that

$$[h(z)]_k = \begin{cases} x_k & \text{if } k \neq j \\ E(z) & \text{if } k = j. \end{cases}$$

Suppose v_i : $test$ and $cond(v_i) = true$ and set $[\![v_i]\!]_\pi(p_n, D_n) = (p'_n, D'_n)$ and $[\![v_i]\!]_\pi(p, D) = (p', D')$. By hypothesis $p' \neq 0$. First observe that $p'_n = p_n \cdot P_{D_{n,z}}([\![\mathtt{B(z)}]\!]) \to p \cdot P_{D_z}([\![\mathtt{B(z)}]\!])$, again because of Theorem 7 and the fact that by hypothesis the set $[\![\mathtt{B(z)}]\!]$ must be an m_{D_z}-continuity set. From this it follows that, starting from some $n_0 > 0$, $p'_n > 0$, $\forall\, n > n_0$.

Let $z^* \in \mathbb{R}^n$ be such that $m_{D_{n,z}}(z^*) = 0 \,\forall\, n$. We can then define

$$h(z) = \begin{cases} z & \text{if } z \in [\![\mathtt{B(z)}]\!] \\ z^* & \text{else.} \end{cases}$$

h is m_{D_z}-measurable and its set of discontinuity points is given by $\partial[\![\mathtt{B(z)}]\!]$, so that $m_{D_z}(\partial[\![\mathtt{B(z)}]\!]) = 0$ because we are assuming that the sets $[\![\mathtt{B(z)}]\!]$ are m_{D_z}-continuity sets. So applying again Theorem 7 we have that

$$\frac{1}{p'_n} m_{D_{n,z}} \circ h^{-1} \to \frac{1}{p'} m_{D_z} \circ h^{-1}.$$

Moreover, $\mathbb{R}^{n-d}$ is a continuity set for any measure (since it has no border), so when applying the operator $Marg_x$, the weak convergence is preserved. The conclusion follows observing that $Marg_x(\frac{1}{p'_n} m_{D_n} \circ h^{-1}) = m_{D'_n}$ and $Marg_x(\frac{1}{p'} m_D \circ h^{-1}) = m_{D'}$. Convergence for $cond(v_i) = false$ follows from the same argument.

If v_i : *observe*, we can apply the same argument used for v : *test*. □

Lemma 3. *Given a weakly converging sequence of distributions $D_n \to D$ for each $n \in \mathbb{N}$ it is possible to find an integer $r_n \in \mathbb{N}_0$ such that*

$$T^{GM}_{r_n}(D_n) \xrightarrow{n\to\infty} D.$$

Proof. Consider the distribution space over $\mathbb{R}^d$ with the Levy-Prokhorov metric d_{LP}. Consider the family of sequences $T^{GM}_r(D_n)$ where $n, r \in \mathbb{N}$. We want to show that for each n, it is possible to fix r_n such that

$$\forall\, \epsilon > 0\, \exists\, n_0 \text{ s.t. } \forall\, n \geq n_0 \quad d_{LP}\left(T^{GM}_{r_n}(D_n), D\right) < \epsilon.$$

Let $\epsilon > 0$ be fixed and ϵ_n be a real sequence such that $\epsilon_n \to 0$ and $|\epsilon_n| < \frac{\epsilon}{2}\, \forall\, n$. By Theorem 8, $T^{GM}_r(D_n) \xrightarrow{r\to\infty} D_n\, \forall\, n$, so there exists $\bar{r}_n$ such that $\forall\, r > \bar{r}_n\; d_{LP}\left(T^{GM}_r(D_n), D_n\right) < \epsilon_n$. Moreover let n_0 be such that $\forall\, n > n_0$ $d_{LP}(D_n, D) < \frac{\epsilon}{2}$. Then, for each n we can choose $r_n > \bar{r}_n$ and we have that $\forall\, n > n_0$:

$$d_{LP}\left(T^{GM}_{r_n}(D_n), D\right), < d_{LP}\left(T^{GM}_{r_n}(D_n), D_n\right) + d_{LP}(D_n, D) < \epsilon_n + \frac{\epsilon}{2} < \epsilon.$$

□

4.5.3 Proof of Theorem 9

We first prove that the theorem is true for programs P such that $\forall\, \pi \in \Pi^P$ if $[\![\pi]\!] = (p', D')$ it holds $p' \neq 0$. Then we prove that, given this, the conclusion generalizes to any P in the hypotheses of the theorem.

Suppose that P is such that $\forall\, \pi \in \Pi^P$ such that $[\![\pi]\!] = (p', D')$ it holds $p' \neq 0$. Let $\pi = v_0 \cdots v_n \in \Pi^P$ be fixed such that $[\![\pi]\!] = (p', D')$ with $p' \neq 0$. We want to prove that:

$$[\![P]\!]^{R^k} = \sum_{\substack{(p_k, D_k) = [\![\pi]\!]^{R^k} \\ \pi \in \Pi^P}} \frac{p_k}{\displaystyle\sum_{\substack{(p'_k, D'_k) = [\![\pi]\!]^{R^k} \\ \pi \in \Pi^P}} p'_k} D_k \to \sum_{\substack{(p,D) = [\![\pi]\!]: \\ \pi \in \Pi^P}} \frac{p}{\displaystyle\sum_{\substack{(p', D') = [\![\pi]\!] \\ \pi \in \Pi^P}} p'} D = [\![P]\!]. \tag{4.12}$$

Observe that since $|\Pi^P| < \infty$ this is implied by:

$$[\![\pi]\!]^{R^k} \to [\![\pi]\!]. \tag{4.13}$$

By definition of path semantics, we can prove (4.13) by showing that for every $i = 0, \dots, n$ it is possible to choose $R^k(v_i) \,\forall\, k \in \mathbb{N}$ such that the output of $[\![v_i]\!]_\pi^{R^k}$ converges to the output $[\![v_i]\!]_\pi$.

For $i = 0$, we can set $R^k(v_0)$ to any value, as this does not affect the final distribution. In fact $[\![v_0]\!]_\pi^{R^k} = [\![v_0]\!]_\pi = (1, \delta_0)$.

For $i = 1$ if $v_1 : exit$ there is nothing to prove. If not we set $R^k(v_1) = k$ and let (p'_0, D'_0) be $[\![v_1]\!]_\pi(1, \delta_0)$. Then:

$$\begin{aligned}[\![v_1]\!]_\pi^{R^k}(1, \delta_0) = \left(\mathbb{I}, T_k^{GM}\right) \circ [\![v_1]\!]_\pi(1, \delta_0) = \left(\mathbb{I}, T_k^{GM}\right)(p'_0, D'_0) = \\ = \left(p'_0, T_k^{GM}(D'_0)\right)\end{aligned}$$

and by Theorem 8 $T_k^{GM}(D'_0) \to D'_0$.

So we have proved that the statement holds for $i = 0$ and $i = 1$. Now suppose that it holds for some $i > 1$ and let us prove that it holds for $i + 1$.

If $v_{i+1} : exit$ there is nothing to prove. If not let (p_k, D_k) be output of $[\![v_i]\!]_\pi^{R^k}$. By inductive hypothesis $(p_k, D_k) \to (\bar{p}, \bar{D}) = [\![v_i]\!]_\pi$. Then:

$$[\![v_{i+1}]\!]_\pi^{R^k}(p_k, D_k) = \left(\mathbb{I}, T^{GM}_{R^k(v_{i+1})}\right) \circ [\![v_{i+1}]\!]_\pi(p_k, D_k).$$

Let (p'_k, D'_k) be $[\![v_{i+1}]\!]_\pi(p_k, D_k)$. By hypothesis $p'_k \neq 0$ (or $[\![\pi]\!](D) = (0, D')$) so we can apply Lemma 2 to get $(p'_k, D'_k) \to [\![v_{i+1}]\!]_\pi(\bar{p}, \bar{D}) = (\bar{p}', \bar{D}')$, so

$$[\![v_{i+1}]\!]_\pi^{R^k}(p_k, D_k) = \left(\mathbb{I}, T^{GM}_{R^k(v_{i+1})}\right)(p'_k, D'_k) = \left(p'_k, T^{GM}_{R^k(v_{i+1})}(D'_k)\right).$$

Then, by Lemma 3 we can choose a sequence of integers s_k such that setting $R^k(v_{i+1}) = s_k$ $T^{GM}_{R^k(v_{i+1})}(D'_k) \to \bar{D}'$. Thus, we have set $R^k(v_{i+1}) = s_k$ so that $[\![v_{i+1}]\!]_\pi^{R^k}(p_k, D_k) \to [\![v_{i+1}]\!]_\pi(\bar{p}, \bar{D})$.

Now suppose that for $[\![\pi]\!] = (0, D')$ for some π, we want to prove that (4.12) still holds. In this case the path π does not contribute to the output distribution of computed by $[\![P]\!](D)$. Moreover, there exist i such that at v_i the output pair is $(0, D'')$ while for all $j < i$ the output at v_j is $(p^{(j)}, D^{(j)})$ with $p^{(j)} \neq 0$. The statement then holds up to node v_i, that takes in input a sequence $(p_k, D_k) \to (p'', D'')$. Letting $[\![v_i]\!]_{\pi}^{R^k}(p_k, D_k) = (p'_k, D'_k)$ and using the same argument as in the proof of Lemma 2 we can prove $p'_k \to 0$. So (4.12) will hold even if (4.13) does not.

Chapter 5

Second Order Gaussian Approximation

As discussed in Examples 8 and 9 of the previous chapter, while implementing an arbitrary order Gaussian Semantics may be difficult, it is straightforward to compute the Gaussian Semantics associated with $R = 2$. In this chapter, we propose *Second Order Gaussian Approximation (SOGA)*, an algorithm that implements this particular case. A prototype implementation of SOGA can be found at `https://zenodo.org/records/10026970`. In Section 5.1, we introduce an overview of the algorithm and its computational cost, while in Section 5.2 we present the numerical evaluation of the algorithm.

5.1 Second Order Gaussian Approximation

5.1.1 Overview

SOGA implements Second-Order Gaussian Semantics, i.e. the Gaussian Semantics that at each node of the control-flow graph matches the first two order moments (mean and covariance matrix).

In our implementation, SOGA accepts programs in a Python-like syntax, then compiled into a formal control-flow graph. SOGA recursively visits the nodes of the control-flow graph in a breadth-first fashion to compute the semantics of all paths.

We assume that each node in the control-flow graph has two attribute lists of *children* and *parents*, whose elements point, respectively, to children and parent nodes. Furthermore, each node has two attributes, p and $dist$: p is a non-negative scalar proportional to the probability of reaching that node, while $dist$ stores the output distribution (in the form of a GM) computed by that semantics of the node. Nodes have type-specific attributes: nodes of type *test* and *observe* have an attribute *LBC* storing an LBC expression; nodes of type *state* have an attribute *cond* taking value *true, false* or *none* and an attribute *expr* storing an assignment expression.

To apply SOGA we create a queue $visit\_queue$ containing the *entry* node. Then we apply iteratively SOGA on *pop*$(visit\_queue)$. When called on a new node, the algorithm first accesses the attributes p and $dist$ of its parents, and invokes *merge_dist* on the list of pairs (p, D). Then, computes the semantics corresponding to the node type as follows:

- if $v\colon entry$, it initializes $node.p$ to 1, $node.dist$ to δ_0 and $node.trunc$ to $none$;

- if $v\colon observe$, it saves the LBC in $node.trunc$, and calls the function *approx_trunc*;

- if $v\colon test$, it does nothing;

- if $v\colon state$, it checks if $node.cond = true$ or $node.cond = false$ and in that case retrieves the LBC condition from the parent $test$ node. Then it calls the function *approx_trunc*. This results in a new pair $(p, dist)$ on which the function *apply_rule* is applied. Finally, the output is stored in $node.p$, $node.dist$;

- if $v\colon exit$, after merging the resulting distribution is returned as the approximated output distribution of the whole program.

After executing the semantics of the node the queue is updated by pushing the children nodes of the current node. This is detailed in Algorithms 3-7. The computation of the moments of the transformed distributions is performed by the functions $apply\_rule(input\_dist, expr)$ and $approx\_trunc(input\_dist, trunc)$, which are detailed below.

Algorithm 3 *SOGA*$(node\colon\ entry)$:

```
$node.p = 1;$
$node.dist = [d \cdot gm([1],[0],[0])];$
$node.trunc = none$
for $child$ in $node.children$ do
    push($visit\_queue$, $child$)
end for
```

Algorithm 4 *SOGA*$(node\colon\ observe)$:

```
$input\_list = []$
for $par$ in $node.parents$ do
    $input\_list.append((par.p, par.dist))$
end for
$node.p, node.dist =$ merge_dist($input\_list$)
$node.trunc = node.LBC$
$I, node.dist =$ approx_trunc($node.dist, node.trunc$)
$node.p = node.p \cdot I$
for $child$ in $node.children$ do
    push($visit\_queue$, $child$)
end for
```

Algorithm 5 *SOGA*$(node\colon\ test)$:

```
for $par$ in $node.parents$ do
    $input\_list.append((par.p, par.dist))$
end for
$node.p, node.dist =$ merge_dist($input\_list$)
for $child$ in $node.children$ do
    push($visit\_queue$, $child$)
end for
```

Algorithm 6 *SOGA*($node\colon\ state$):

```
for $par$ in $node.parents$ do
    $input\_list.append((par.p, par.dist))$
end for
$node.p, node.dist$ = merge_dist($input\_list$)
if $node.cond == true$
    $trunc = node.parent.LBC$
else if $node.cond == false$
    $trunc$ = not $node.parent.LBC$
end if
if $node.cond! = none$
    $p', node.dist$ = approx_trunc($node.dist, trunc$)
    $node.p = node.p \cdot p'$
end if
$node.dist$ = apply_rule($node.dist, node.expr$)
for $child$ in $node.children$ do
    push($visit\_queue$, $child$)
end for
```

Algorithm 7 *SOGA*($node\colon\ exit, p, dist, trunc$):

```
for $par$ in $node.parents$ do
    $input\_list.append((par.p, par.dist))$
end for
$node.p, node.dist$ = merge_dist($input\_list$)
return $node.dist$
```

Function *apply_rule*. It implements the semantics of a *state* node. It takes as input the current mixture $input\_dist$ and an expression $expr$ of type (4.2). It returns a new distribution $dist$ obtained applying $expr$ and T_2^{GM} to $input\_dist$. To compute the moments of the transformed distribution $dist$, and therefore its second-order approximation, it uses the results in Table 10: when $expr$ is a linear transformation, it applies the formulas for the sum of multivariate Gaussians (Billingsley, 2013); when $expr$ involves products, it applies Isserlis' theorem (Wick, 1950).

Function *approx_trunc* It implements the semantics of a *test* or an *observe* node. It takes as input the current mixture $input\_dist$ and a set $trunc$ defined by an LBC of type (4.3). It returns the probability mass p, given by the probability that $input\_dist$ satisfies $trunc$, and a new mixture distribution $dist$, representing the GM approximating $input\_dist$ conditioned to $trunc$. Again, it applies the results in Table 10 to compute $dist$: in particular, when the LBC expresses inequality constraints the formulas in Kan and Robotti (2017) are used; when instead the LBC has the form $\mathtt{x_i} == \mathtt{c}$ it uses the formulas from Bishop and Nasrabadi (2006).

Function *merge_dist* Merging is performed whenever a node is accessed prior to applying its semantics: *merge_dist* collects all the output pairs (p, D) computed at the parent nodes, and merges them together in a single GM. Given the set of s parents' pairs $(p_1, D_1), \ldots, (p_s, D_s)$, the function returns probability mass $input\_p = p_1 + \ldots + p_s$ and a new GM $input\_dist = \frac{1}{input\_p}(p_1 D_1 + \ldots + p_s D_s)$. For an *exit* node, the output of this function is the output distribution of the program.

5.1.2 Distributivity of Transfer Functions

SOGA explores the control-flow graph in a breadth-first fashion, performing merges when required. On the other end, the exact and the Gaussian semantics are defined as a sum over all execution paths, leading to an apparent discrepancy. To ensure that SOGA indeed computes the Gaussian Semantics associated with the map $R(v) = 2$ we show in Proposition 8 that the transfer function of the exact semantics is distributive with respect to the merge operation.

To do this, for a set of pairs (p_i, D_i) we define the merge operator

$$merge((p_1, D_1), \ldots, (p_s, D_s)) = \left(\sum_{i=1}^{s} p_i, \sum_{i=1}^{s} \frac{p_i}{\sum_{j=1}^{s} p_j} D_i \right) = (P, D).$$

We show that computing the semantics of a node after performing a merge gives the same output distribution as computing the semantics of each pair and then merging the results. This distributivity transfers straightforwardly to Gaussian Semantics, since it is computed by composing the exact semantics with the operator T_r^{GM}, which is distributive with respect to merging by Definition 4.7. This, in turn, justifies computing the semantics exploring the control-flow graph in a breadth-first fashion as SOGA does.

Proposition 8. *Let (p_i, D_i) be pairs with $p_i \geq 0$ and D_i a distribution for $i = 1, \ldots, s$. Let v be a node of type state, test, observe or exit. Then*

$$merge([\![v]\!](p_1, D_1), \ldots, [\![v]\!](p_s, D_s)) = [\![v]\!](merge((p_1, D_1), \ldots, (p_s, D_s))) \tag{5.1}$$

Proof. Let $(\tilde{p}_i, \tilde{D}_i) = [\![v]\!](p_i, D_i)$ for $i = 1, \ldots, s$. Then the L.H.S. of Equation 5.1 becomes

$$merge([\![v]\!](p_1, D_1), \ldots, [\![v]\!](p_s, D_s)) = \left(\sum_{i=1}^{s} \tilde{p}_i, \sum_{i=1}^{s} \frac{\tilde{p}_i}{\sum_{j=1}^{s} \tilde{p}_j} \tilde{D}_i \right) = (\tilde{P}, \tilde{D}).$$

For the R.H.S. let:

$$[\![v]\!](merge((p_1, D_1), \ldots, (p_s, D_s))) = [\![v]\!](P, D) = (\hat{P}, \hat{D}).$$

Let us show for each type of node that $\tilde{P} = \hat{P}$ and $\tilde{D} = \hat{D}$. We observe that for $v\colon$ *state*, with v labeled by `skip`, and for $v\colon$ *exit* conclusion follows trivially. We examine the remaining cases separately.

- Let $v\colon$ *state* and suppose v is labelled by $\mathtt{x_k} := \mathtt{E(z)}$ then $\tilde{p}_i = p_i$ and $\tilde{D}_i$ is the distribution of $x[x_k = E(z)]$ where $x \sim D_i$. Then $\tilde{P} = P$ and $\tilde{D} = \sum_{i=1}^{s} \frac{p_i}{P} \tilde{D}_i$. On the other hand $\hat{P} = P = \tilde{P}$ and $\tilde{D}$ is the distribution of $y[y_k = E(z)]$ where $y \sim D = \sum_{i=1}^{k} \frac{p_i}{P} D_i$. Therefore $\hat{D} = \sum_{i=1}^{k} \frac{p_i}{P} \tilde{D}_i = \tilde{D}$.

- Let $v\colon$ *test*. To ease the notation, let us assume $cond(s_\pi(v)) = true$ and $\mathtt{B(z)} = \mathtt{B(x)}$, but the argument works analogously in the other cases. In this case $\tilde{p}_i = p_i \cdot P_{D_i}([\![\mathtt{B(x)}]\!])$ and $\tilde{D}_i = D_i \mid [\![\mathtt{B(x)}]\!]$. Therefore $\tilde{P} = \sum_{i=1}^{s} p_i \cdot P_{D_i}([\![\mathtt{B(x)}]\!])$ and $\tilde{D} = \sum_{i=1}^{s} \frac{p_i \cdot P_{D_i}([\![\mathtt{B(x)}]\!])}{\tilde{P}} (D_i \mid [\![\mathtt{B(x)}]\!])$. On the other hand $\hat{P} = P \cdot P_D([\![\mathtt{B(x)}]\!]) = P \cdot \sum_{i=1}^{s} \frac{p_i}{P} P_{D_i}([\![\mathtt{B(x)}]\!]) = \sum_{i=1}^{s} p_i P_{D_i}([\![\mathtt{B(x)}]\!]) = \tilde{P}$. Moreover, $\hat{D} = D \mid [\![\mathtt{B(x)}]\!]$. Therefore, $\hat{D}$ has density

$$\frac{f_D(x) \mathbb{I}_{[\![\mathtt{B(x)}]\!]}}{P_D([\![\mathtt{B(x)}]\!])} = \sum_{i=1}^{s} \frac{p_i P_{D_i}([\![\mathtt{B(x)}]\!])}{P_D([\![\mathtt{B(x)}]\!])} \frac{f_{D_i}(x) \mathbb{I}_{[\![\mathtt{B(x)}]\!]}}{P_{D_i}([\![\mathtt{B(x)}]\!])}$$

  which is the same as the one of $\sum_{i=1}^{s} \frac{p_i P_{D_i}([\![\mathtt{B(x)}]\!])}{\tilde{P}} (D_i \mid [\![\mathtt{B(x)}]\!]) = \tilde{D}$. Observe that we have assumed that for at least one i $\tilde{p}_i \neq 0$. However, if that is not the case $P_D([\![\mathtt{B(x)}]\!]) = 0$ and the conclusion still holds.

- Let v: *observe*. If `B(x)` has a probability greater than zero conclusion follows as in the previous case. If `B(x)` has the form $\texttt{x}_\texttt{k}$ == `c` we can use the same argument, but we need to replace $P_{D_i}(\llbracket B(x) \rrbracket)$ with the normalization constant $I_i = \int_{\mathbb{R}^{d-1}} f_{D_i}(x, x_i = c) d(x \setminus x_i)$.

□

5.1.3 SOGAprune

To improve the scalability of SOGA we propose a second version of the algorithm, called SOGAprune, in which the user can introduce at script level the instruction **prune**(K), K being an integer number. When the script is compiled in a cfg, the prune instruction is compiled in a new node of type *prune*. When accessed, the function node_semantics invokes the function $prune\_dist(input\_dist, K)$.

Function *prune_dist* It prunes the current distribution $input\_dist$ to keep the number of its components below a user-specified bound K. The pruning is performed similarly to Chaudhuri and Solar-Lezama (2010). In particular, for each pair of components i, j in the input distribution *input_dist*, having mixing coefficients π_i, π_j, means μ_i, μ_j and covariance matrices Σ_i, Σ_j, we compute the mean $\mu' = \frac{\pi_i \mu_i + \pi_j \mu_j}{\pi_i + \pi_j}$ and the cost $cost(i, j) = \pi_i \|\mu' - \mu_i\| + \pi_j \|\mu' - \mu_j\|$.

After computing the cost for all pairs (i, j) such that $i \neq j$ and $i, j < K$, the pair (i, j) with minimal cost is substituted with a single component having mean μ' and covariance matrix Σ' with

$$\Sigma' = \frac{\pi_i}{\pi_i + \pi_j}\left(\Sigma_i + \mu_i^T \mu_i\right) + \frac{\pi_j}{\pi_i + \pi_j}\left(\Sigma_j + \mu_j^T \mu_j\right) - \mu'^T \mu'.$$

Observe that μ' and Σ' are exactly the mean ad the covariance matrix of the mixture $\frac{\pi_i}{\pi_i + \pi_j}\mathcal{N}(\mu_i, \Sigma_i) + \frac{\pi_j}{\pi_i + \pi_j}\mathcal{N}(\mu_j, \Sigma_j)$. This produces the best possible approximation of the two components (Chaudhuri and Solar-Lezama, 2010). The procedure is iterated until the number of components is less than K (observe that after the first two components have been merged into a new one, we need to recompute the cost only for the pairs in which the new component appears).

A summary of how the semantics of each node is implemented is reported in Table 11.

Type	*Function*	*Input*	*Computing*
$state$	*apply_rule*	$input\_dist$, $expr$	First two order moments of the components of the distribution obtained applying $expr$ to $input\_dist$
$text$, $observe$	*approx_trunc*	$input\_dist$, $trunc$	Probability mass (or normalization constant) and first two order moments of the components of the distribution obtained truncating $input\_dist$ to $trunc$
$prune$	*prune_dist*	$input\_dist$, K	Distribution $input\_dist$ iteratively pruned until the number of components is $\leq K$

Table 11: Function implementing node semantics in SOGA and SOGAprune. The input arguments $input\_p$, $input\_dist$ are retrieved by the parent nodes' attributes p, $dist$. The arguments $expr$, $trunc$ and K are stored in node attributes when the cfg is compiled from the program script.

5.1.4 Computational Cost

We first compute the computational cost without pruning, then we discuss how pruning affects it.

Cost without Pruning. Let $|V|$ denote the total number of nodes, $|T|$ the number of *test* nodes, $|TO|$ the number of *test* and *observe* nodes and $|S|$ the number of *state* nodes. W.l.o.g. we assume for simplicity that all read-only variables are pushed to an initial distribution D_0 over $\mathbb{R}^n$; thus the output of the entry node is $(1, D_0)$ and all assignments only use output variables. By doing this we compute an upper bound on the true computational cost since the dimensions corresponding to read-only variables are dropped after marginalization. Letting C_0 denote the number of components of D_0, the output distribution will have at most $C \leq C_{max} = 2^{|T|}C_0$ components.

We consider the cost to access a node and perform elementary operations, such as assignments and products, constant. Expressions $expr$ and $trunc$ are assumed to be stored in suitable data structures accessible in constant time, so storage and reading of them are also considered elementary operations. Overall, elementary operations contribute to the total computational cost with a term $O(|V|)$, which is however dominated by the computational cost of executing *approx_trunc*, *apply_rule* and *merge_dist*. We examine their cost separately.

The function *approx_trunc* is invoked once when an *observe* node is accessed and twice when a *test* node is accessed, for the true and the false branch respectively. When $\mathtt{B(z)}$ is in the form $\mathtt{c_1} \cdot \mathtt{z_1} + \ldots + \mathtt{c_n} \cdot \mathtt{z_n} \bowtie \mathtt{c}$

a singular value decomposition is performed to change coordinates, so that in the new set of coordinates the truncation set is a hyper-rectangle (cost $O(n^3)$, (Gu and Eisenstat, 1995)). Then, a new mixing coefficient has to be computed for each component to convert the truncated GM into a mixture of truncated Gaussians (cost $O(n)$). Finally, for each truncated Gaussian, the first two order moments are computed using the formulas in Kan and Robotti (2017) (cost $O(n^4)$, see paragraph below). When `B(z)` is in the form `x_i == c`, to apply the formulas in Bishop and Nasrabadi (2006), matrix multiplication must be performed, amounting to cost $O(n^2)$ (Skiena, 2008). Overall, we have a cost of $O\left(|TO|C_{max}n^4\right)$.

The function *apply_rule* is invoked every time a *state* node is accessed. Since affine transformations require matrix multiplication (cost $O(n^3)$), the total cost is $O\left(|S|C_{max}n^3\right)$.

Finally, the function *merge_dist* is invoked whenever a node is accessed and performs a scalar product. It contributes for a cost $O\left(|V|C_{max}\right)$.

The total cost of SOGA is therefore

$$O\left(|TO|C_{max}n^4\right) + O\left(|S|C_{max}n^3\right) + O\left(|V|C_{max}\right) \leq O(|V|2^{|T|}C_0n^4), \tag{5.2}$$

that is, linear in the number of nodes $|V|$ and in the initial number of components C_0, polynomial in the dimensionality of the augmented input space n and exponential in the number of *test* nodes $|T|$, i.e., linear in the number of paths.

Moments of Truncated Gaussians We derive the computational cost of computing the first two order central moments of a d-dimensional Gaussian distribution truncated to a hyper-rectangle $[\underline{a}, \underline{b}] = [a_1, b_1] \times \ldots \times [a_d, b_d]$ in the special case in which $a_1 > -\infty$, $a_i = -\infty \, \forall \, i = 2, \ldots, d$ and $b = \infty \, \forall \, i = 1, \ldots, d$. Observe that this case and the symmetric one with $b_1 < \infty$ are the only ones arising in executing SOGA, because we restrict conditional branches to have the form in 4.3. To carry out the computation we use the recursive formulas from Kan and Robotti, 2017 reported below.

Let $\underline{r} = (r_1, \ldots, r_d) \in \mathbb{N}_0^d$. We define:

$$F_{\underline{r}}^d(\underline{a}, \underline{b}, \mu, \Sigma) = \int_{[\underline{a},\underline{b}]} x^{\underline{r}} f_{\mathcal{N}(\mu,\Sigma)}(x)dx.$$

If X is a Gaussian with mean μ and covariance matrix Σ truncated to

$[\underline{a}, \underline{b}]$ we have that

$$\mathbb{E}[X^{\underline{r}}] = \frac{F^d_{\underline{r}}(\underline{a}, \underline{b}, \mu, \Sigma)}{F^d_{\underline{0}}(\underline{a}, \underline{b}, \mu, \Sigma)}$$

so, if we compute $F^d_{\underline{r}}$ for all $\underline{r}$ such that $\sum_{i=1}^d r_i \leq 2$ we can retrieve the first two order moments of the truncated Gaussian in $O(d^2)$ operations.

Observe that due to the particular form of our hyper-rectangles $F^d_{\underline{0}}$ can be computed in costant time, as if $d = 1$.

To compute $F^d_{\underline{r}}$ for other value of $\underline{r}$ we use the recursive formula:

$$F^d_{\underline{r}+e_i}(\underline{a}, \underline{b}, \mu, \Sigma) = \mu_i F^d_{\underline{r}}(\underline{a}, \underline{b}, \mu, \Sigma) + e_i^T \Sigma c_{\underline{r}} \tag{5.3}$$

where

$$c_{\underline{r},j} = k_j F^d_{\underline{r}-e_j}(\underline{a}, \underline{b}, \mu, \Sigma) + a_j^{k_j} f_{\mathcal{N}(\mu_j, \Sigma_{j,j})}(a_j) F^{d-1}_{\underline{r}_{(j)}}(\underline{a}_{(j)}, \underline{b}_{(j)}, \tilde{\mu}^a_j, \tilde{\Sigma}_j) \tag{5.4}$$

$$\tilde{\mu}^a_j = \mu_{(j)} + \Sigma_{(j),j} \frac{a_j - \mu_j}{\Sigma_{j,j}} \tag{5.5}$$

$$\tilde{\Sigma}_j = \Sigma_{(j),(j)} - \frac{1}{\Sigma_{j,j}} \Sigma_{(j),j} \Sigma_{j,(j)} \tag{5.6}$$

and for a vector $\underline{v}$ the notation $\underline{v}_{(j)}$ denotes the vector obtained from $\underline{v}$ suppressing the index j. Moreover, it is understood that when $a_j = -\infty$ the second term at the right hand side of (5.4) is 0.

To compute moments of order 1, i.e. $F^d_{e_i}(\underline{a}, \underline{b}, \mu, \Sigma)$ for $i = 1, \ldots, d$, we set $\underline{r} = \underline{0}$ in (5.3). We first compute $c_{\underline{0}}$ for which we have

$$c_{\underline{0},1} = f_{\mathcal{N}(\mu_i, \Sigma_{1,1})}(a_1) F^{d-1}_{\underline{r}_{(1)}}(\underline{a}_{(1)}, \underline{b}_{(1)}, \tilde{\mu}^a_1, \tilde{\Sigma}_1) = f_{\mathcal{N}(\mu_i, \Sigma_{1,1})}(a_1)$$

since $[\underline{a}_{(1)}, \underline{b}_{(1)}] = \mathbb{R}^{d-1}$ and

$$c_{\underline{0},j} = 0.$$

Therefore $c_{\underline{0}}$ is computed in constant time and the only computational cost in computing the first order moments is due to the $(d \times d) \cdot (d \times 1)$ matrix multiplication $e_i^T \Sigma c_{\underline{0}}$, which is $O(d^2)$. Since we need to compute d first order moments, the total cost is $O(d^3)$.

To compute moment of order 2, we set $\underline{r} = e_s$ and compute $F^d_{e_s+e_i}(\underline{a}, \underline{b}, \mu, \Sigma)$ as $s, i = 1, \ldots, d$. We first need to compute c_{e_s} for which we have:

$$c_{e_s,1} = \begin{cases} F^d_{\underline{0}}(\underline{a}, \underline{b}, \mu, \Sigma) + a_1 f_{\mathcal{N}(\mu_1, \Sigma_{1,1})}(a_1) & \text{if } s = 0 \\ f_{\mathcal{N}(\mu_1, \Sigma_{1,1})}(a_1) F^{d-1}_{e_{s_{(1)}}}(\underline{a}_{(1)}, \underline{b}_{(1)}, \tilde{\mu}^a_1, \tilde{\Sigma}_1) & \text{if } s \neq 0. \end{cases} \tag{5.7}$$

$$c_{e_s,j} = \begin{cases} F^d_{\underline{0}}(\underline{a}, \underline{b}, \mu, \Sigma) & \text{if } s = j \\ 0 & \text{if } s \neq j. \end{cases}$$

As $s = 2, \ldots, d$ we need to compute $F^{d-1}_{e_{s_{(1)}}}(\underline{a}_{(1)}, \underline{b}_{(1)}, \tilde{\mu}^a_1, \tilde{\Sigma}_1)$. However since this are the first order moments of a gaussian with mean μ^a_1 in $[\underline{a}_{(1)}, \underline{b}_{(1)}] = \mathbb{R}^{d-1}$, computing the previous quantity amounts to computing $\tilde{\mu}^a_1$ which can be done in $O(d)$ operations. Once this is done, $c_{e_s,j}$ can be computed for every s and j in $O(d^2)$ operations. Finally, we need to perform again the matrix multiplication $e_1^T \Sigma c_{e_s}$, this time for $\frac{d(d-1)}{2}$ times, for a total computational cost of $O(d^4)$.

Effect of Pruning. Let us now consider the effect of introducing some **prune**(K) instructions. Let $|P|$ be the number of *pruning* nodes and $|T|_{bet}$ be the maximum number of subsequent *test* nodes without *pruning* instructions between them. Then $|T|_{bet} \leq |T|$ and $|T|_{bet} = |T|$ if no pruning instructions have been introduced in the program. Then, the maximum number of components a mixture can have before pruning occurs is $C_{max} = K2^{|T|_{bet}}$ (assuming w.l.o.g. $|C_0| < K$).

The function *prune_dist* is invoked at most $|P|$ times. When invoked, it first computes the cost for all possible pairs of components, which is at most $C_{max}(C_{max} - 1)$. The computation of the cost function for each pair has cost $O(n)$, while the computation of the covariance matrix (cost $O(n^2)$) is performed for a single pair. At its first iteration, the computational cost of *prune_dist* is, therefore, $O(C^2_{max}n)$. After this, new costs are computed for at most $C_{max} - K$ times, but each time only for $C < C_{max}$ pairs of components. The whole cost of the function is therefore $O(C^2_{max}n)$.

Substituting in (5.2) one gets that the computational cost with pruning is bounded by:

$$O(|V|K2^{|T|_{bet}}n^4) + O(K^2 2^{2|T|_{bet}}n) \leq O(|V|K^2 2^{2|T|_{bet}}n^4). \tag{5.8}$$

Comparing (5.2) with (5.8) one can conclude that pruning is only effective in reducing the computational cost when the overhead introduced by pruning ($K^2 2^{2|T|_{bet}}$) is less demanding than dealing with the full space of paths ($C_0 2^{|T|}$). To keep the overhead contained one could use small values of K while keeping also $|T|_{bet}$ small (e.g. by introducing many *pruning* instructions). However, this introduces an additional level of approximation which can hinder the accuracy of SOGA.

5.2 Numerical Evaluation

We split the numerical evaluation into four parts. In Section 5.2.1 we compare SOGA with four baseline tools representative of different inference methods for estimating the posterior mean: STAN for MCMC (Carpenter et al., 2017), PSI for exact symbolic analysis (Gehr, Misailovic, and Vechev, 2016), AQUA for quantization of posterior distributions (Huang, Dutta, and Misailovic, 2021) and Pyro for VI (Bingham et al., 2019). In Section 5.2.2 we compare SOGA against Pyro in performing Maximum A Posteriori (MAP) estimation (Gelman et al., 2013), to test how well our method is able to capture the posterior distribution, in addition to its moments. Finally, in Sections 5.2.3 and 5.2.4, we evaluate SOGA's performance on two applications that have been extensively studied in the literature, owing to their significant practical impact. The first application is inference on models involving mixtures of continuous and discrete distributions, as in Kharchenko, Silberstein, and Scadden (2014), Pierson and Yau (2015), and Gao et al. (2017); the second application is Bayesian inference on collaborative filtering (Zhao, Zhang, and J. Wang, 2013).

5.2.1 Posterior Mean Estimation

We start by comparing SOGA with STAN for MCMC (Carpenter et al., 2017), PSI for exact symbolic analysis (Gehr, Misailovic, and Vechev, 2016), and AQUA for quantization of posterior distributions (Huang, Dutta, and Misailovic, 2021) and Pyro for VI (Bingham et al., 2019). We consider the case studies from these tool's reference papers (Carpenter et al., 2017; Gehr, Misailovic, and Vechev, 2016; Huang, Dutta, and Misailovic, 2021), excluding those which could not be encoded in our syntax. This choice is intended to stress SOGA in the analysis of programs that were not designed to enhance its properties. Overall out of 31 total models, 13 were left out: 9 because of non-parametrizable distributions depending on variable parameters, and 4 because of the presence of non-polynomial functions (taken from: STAN - 1, PSI - 3, AQUA - 8, Pyro - 1). The remaining 18 models can be found in Carpenter et al. (2017) (*Bernoulli*), Gehr, Misailovic, and Vechev (2016) (*BayesPointMachine, Burglar, ClickGraph, ClinicalTrial, CoinBias, DigitRecognition, Grass, MurderMistery, NoisyOr, SurveyUnbias, TrueSkills, TwoCoins*) and Huang, Dutta, and Misailovic (2021) (*Altermu, Altermu2, NormalMixtures, RadarQuery, TimeSeries*).

The considered programs are listed in Table 12. Pruning was applied after every test and observe nodes (repeating it only once if they occur subsequently) for programs whose computation time was greater than 1 s and at least ten times larger than the worst performing tool. We set $K = 0.015C_{max}$ except for *NormalMixtures*; there, since C_{max} exceeded the tens of thousands, we set $K = 30$. With this strategy, the pruning algorithm was invoked only in 4 out of the 18 considered programs.

The experiments were performed on a laptop equipped with a 2.8 GHz Intel i7 quad-core processor and 16 GB RAM, CmdStan v2.30.1 and Wolfram Mathematica 13.1 (Wolfram Research, Inc.), setting a time-out threshold at 600 s.

Results

Table 12 collects the results where *time* refers to the average runtimes (in seconds) out of 10 executions and *value* refers to the computed expected value of a target variable in the model. For each model we specify the kind of distributions involved: B=Bernoulli, Be=Beta, D=Discrete, G(*) =Gaussian (with non-constant mean), U=Uniform. For STAN, we indicate the time needed to obtain a 5% confidence interval whose amplitude is contained in 1% of the mean (up to a maximum of 10^5 samples). For PSI we report the sum of the time needed to generate the symbolic formula and that needed to integrate it when in the presence of non-simplified integrals (observe that in the original paper, only the time for symbolic computation was considered). For VI, due to high sensitivity with respect to the hyperparameters (Hoffman et al., 2013), we proceed using three different learning rates (0.01, 0.005, 0.001). To ease the comparison, we report the most accurate estimation in Table 12, while the full set of experimental results is reported in Table 13. The number of iterations of the stochastic gradient descent is increased from a minimum of 100 to a maximum of 10k, stopping the optimization if the difference between the estimated mean posterior and the mean posterior estimated 100 steps before is less than 1% of the current estimation. For SOGA, runtimes labeled with * indicate that the pruning algorithm was invoked. Finally, we highlight the fastest method with a grey background. For accuracy evaluation, we consider PSI's results as ground truth when available (i.e., when PSI terminates and the integration is successfully computed within the timeout threshold). We made this choice since PSI is an exact method and the only guaranteed to be exact among the evaluated tools.

Only in one example, *BayesPointMachine*, SOGA performs poorly in

Model	*Dist.*	*STAN*		*AQUA*		*Pyro (VI)*		*PSI*		*SOGA*			
		time	*value*	*time*	*value*	*time*	*value*	*time*	*value*	*time*	*value*	*C*	*d*
Bernoulli	B,U	0.17	0.250	0.84	0.247	4.51	0.250	0.38	0.250	1.28*	0.252	27	2
BayesPointMachine	G*	51.0	0.056	mem		60.49	0.046	err		2.20	0.011	1	9
Burglar	B	—		—		—		0.12	0.003	0.06	0.003	4	6
ClickGraph	B,U	102	0.540	mem		3.13	0.566	1.10	0.614	208*	0.630	35	6
ClinicalTrial	B,U	—		—		—		0.97	0.755	92.2*	0.753	23	5
CoinBias	B,Be	0.07	0.420	0.91	0.383	0.91	0.419	0.34	0.417	0.61	0.415	64	2
DigitRecognition	D	—		—		—		err		4.46	4.453	10	2
Grass	B	—		—		—		0.08	0.708	0.09	0.708	28	10
MurderMistery	B	—		—		—		0.12	0.016	0.01	0.016	2	2
NoisyOr	B	—		—		—		0.16	0.814	0.16	0.814	256	10
SurveyUnbias	B,G,U	0.10	0.800	1.08	0.567	2.89	0.770	18.5	0.800	1.56	0.799	128	4
TrueSkills	G*	0.04	104.0	mem		1.30	101.4	to		0.05	104.7	1	6
TwoCoins	B	—		—		—		0.10	0.333	0.01	0.333	3	3
Altermu	G*	19.0	0.009	1.32	0.000	33.1	0.030	to		0.16	0.000	1	5
Altermu2	G*, U	15.0	0.170	0.79	0.155	5.50	0.098	284	0.155	0.36	0.156	4	3
NormalMixtures	G*, U	0.38	0.286	1.27	0.286	104.89	0.295	to		50.4*	0.298	30	4
RadarQuery	B,G*,U	144	5.000	0.90	6.333	err		7.75	6.333	6.34	5.940	2016	8
TimeSeries	G*,U	0.37	-1.600	1.67	-1.575	26.15	-1.701	to		3.79	-1.590	19	4

Table 12: Results using STAN, PSI, AQUA and SOGA. '—': discrete posterior not supported; '*mem*': out of memory error; '*err*': tool returns error state. For SOGA, C: final number of components; d: dimensionality of the output vector.

	Pyro (VI)				*SOGA*		*True value*
Model	*l.r.*	*steps*	*value*	*time*	*value*	*time*	
Bernoulli	0.01	2000	0.247	4.51			
	0.005	500	0.282	1.438	0.252	1.28	0.25
	0.001	1700	0.306	4.44			
BayesPointMachine	0.01	5800	0.046	60.49			
	0.005	not converged			0.011	2.20	0.056*
	0.001	not converged					
ClickGraph	0.01	200	0.566	3.13			
	0.005	200	0.504	3.03	0.63	208	0.614
	0.001	400	0.490	6.36			
CoinBias	0.01	200	0.419	0.91			
	0.005	1100	0.425	5.233	0.41	0.61	0.41
	0.001	900	0.403	4.36			
SurveyUnbias	0.01	500	0.770	2.89			
	0.005	800	0.743	4.42	0.80	1.56	0.80
	0.001	900	0.701	5.02			
TrueSkills	0.01	200	101.4	1.30			
	0.005	200	100.79	1.31	104.7	0.05	104*
	0.001	200	100.16	1.48			
Altermu	0.01	not converged					
	0.005	1400	0.030	33.1	0.000	0.16	0*
	0.001	not converged					
Altermu2	0.01	1300	8.713	29.52			
	0.005	5700	-9.624	150.39	0.156	0.36	0.155
	0.001	200	0.098	5.50			
NormalMixtures	0.01	400	0.344	28.80			
	0.005	1700	0.295	104.89	0.298	50.4	0.286*
	0.001	300	0.500	17.77			
TimeSeries	0.01	2800	-1.832	59.128			
	0.005	900	-2.257	19.87	-1.590	3.79	-1.600*
	0.001	1100	-1.701	26.15			

Table 13: Comparison between Pyro's Variational Inference, SOGA and true values of the models of Table 3 with continuous posterior. For Pyro's VI we report the values and the runtimes for 3 different learning rates (l.r.), together with the number of steps needed to meet our stopping criterion (steps). By 'not converged', we mean that the stopping criterion was not met after 10k steps of gradient descent. True values are obtained using PSI or, when not available, using STAN or AQUA, denoted by *.

terms of accuracy, estimating a value of 0.011 for a parameter estimated by STAN as $0.054 \pm 2\mathrm{e}{-4}$. We remark, however, that this program turned out to be particularly difficult to solve for AQUA (which issued an out-of-memory error) and PSI (which was not able to complete the symbolic computation of the posterior). On the other examples, SOGA yields very good accuracy, with a relative error below 7% across all comparable models. We now discuss a detailed comparison of runtimes against each tool method.

STAN STAN does not support discrete posteriors; hence it could not analyze eight models. For the models that can be analyzed by both, SOGA outperforms STAN in terms of runtimes on *Altermu*, *Altermu2*, and *RadarQuery*. By contrast, STAN outperforms SOGA in *Bernoulli* and *NormalMixtures*. We attribute this to the presence of non-Gaussian priors and a large number of observations, resulting in a high number of components and truncations to be computed. Both have similar performance on the remaining models.

AQUA SOGA is more flexible than AQUA in that it supports discrete posteriors. On *ClickGraph*, and *TrueSkills* AQUA issued an out-of-memory error while SOGA could approximate the posterior mean. We ascribe this issue to the fact that AQUA uses tensors, whose dimension rapidly increases with the number of distributions. In particular, in AQUA each distribution must be stored in the tensor, while SOGA can use fresh read-only variables which are dropped once the marginal over the output variables is evaluated. Notably, SOGA outperforms AQUA also on *Altermu*, *Altermu2* and *TimeSeries* proposed in the AQUA paper (Huang, Dutta, and Misailovic, 2021). Instead, AQUA is more efficient than SOGA in *RadarQuery*, *Bernoulli* and *NormalMixtures*, for the same reasons explained for STAN.

Pyro Being a gradient-based method, Pyro's VI offers limited support for discrete variables,[1] so that, similarly to STAN and AQUA, we were not able to encode models with discrete posterior. In addition, we found that the encoding of *RadarQuery* incurred runtime errors. For the remaining models, VI is comparable to SOGA when not less accurate and taking longer runtimes. Noticeable exceptions are *BayesPointMachine*,

[1] `https://pyro.ai/examples/enumeration.html`

Model	*Time*	*Value*	*C*
Bernoulli	11.97	0.252	1774
ClickGraph	to		2304
ClinicalTrial	to		1508
NormalMixtures	to		97714

Table 14: Application of pruning to models whose computation time was greater than 1 s and at least ten times larger than the worst performing competitor. For models yielding a time-out the number of components C is the one reached before time-out.

where, as already noticed, SOGA is not able to achieve a good accuracy and *ClickGraph*, where SOGA incurs long runtimes even with pruning. On the other hand, VI exhibits a significant sensitivity with respect to the choice of the hyperparameters, which can result in non-convergence and sloppy approximations for poor choices of the parameters, as shown in Table 13.

PSI Similarly, PSI outperforms SOGA on *Bernoulli*, *ClickGraph*, and *ClinicalTrial*. However, on six models (*SurveyUnbias, TrueSkills, Altermu, Altermu2, RadarQuery,* and *TimeSeries*) PSI timed out or resulted in long runtimes. This behavior can be explained by the presence of distributions dependent on variable parameters (*SurveyUnbias, Trueskills, Radar*) or by the high number of observations (*Altermu, Altermu2, TimeSeries*). In *Altermu*, PSI could not compute a symbolic formula within the time-out threshold, while in *BayesPointMachine, TrueSkills,* and *TimeSeries* the formula contained non-simplified integrals, whose integration in Mathematica took longer than the time-out threshold. Notably, for models involving only Bernoulli distributions (*Burglar, Grass, MurderMistery, NoisyOr, TwoCoins*), for which both tools are exact, their performance is comparable.

Performance of Pruning. In Table 14 we report the runtimes, values, and number of components C for SOGA without pruning applied to the four models that required the application of pruning (*Bernoulli, ClickGraph, ClinicalTrial, NormalMixtures*). All models share the occurrence of GM distributions with more than 1000 components. For *Bernoulli,* pruning allowed comparable runtimes with respect to the best-performing tool, while base SOGA was about 9 times slower (11.97 s). In addition, base SOGA computed an output indistiguishable from the pruned version up to the third decimal digit. For the other three cases base SOGA was unable to compute a numerical result within the time out threshold. Applying pruning allowed SOGA to complete the computation before the time out while achieving excellent accuracy with respect to ground truth.

5.2.2 Maximum a Posteriori Estimation

Since SOGA approximates the posterior with a Gaussian mixture, it can also compute the Maximum a Posteriori (MAP) estimate by simply returning the mean of the GM component with the largest weight. Here we compare its performance against Pyro, in which MAP estimation can be performed using a different parametrizing distribution than the one used for the mean posterior inference[2]. To get a baseline for the MAP value, we first generate the symbolic posterior using PSI and then optimize it numerically. For models in which PSI is not able to compute the exact posterior, we estimate the ground truth by taking 10k samples from the posterior and binning them into 50 intervals; then, MAP is the midpoint of the interval with the most samples. We tested the same models with continuous posterior reported in Table 12, except *Altermu2*, since, by visual inspection, we found that it has a flat posterior.

Results are reported in Table 15. Due to Pyro's sensitivity to hyperparameters observed in the previous section, we tested three different values of learning rate, confirming the sensitivity issues. These experiments show that SOGA performs relatively worse than in the estimation of the posterior mean. This is expected because SOGA is designed to match means and variances, but it does not necessarily approximate the whole distribution. However, compared to Pyro, it is still able to obtain the closest estimation for *BayesPointMachine*, *ClickGraph*, *TrueSkills*, *Altermu* and *NormalMixtures*, while it is outperformed by Pyro in *Bernoulli*, *CoinBias*, *SurveyUnbias* and *TimeSeries*. Finally, we note that analyzing *Bernoulli* with SOGAprune degrades the MAP estimation, unlike in the posterior mean.

[2] `https://pyro.ai/examples/mle_map.html`

	Pyro				*SOGA*		*True value*
Model	*l.r.*	*steps*	*value*	*time*	*value*	*time*	
Bernoulli	0.01	200	0.200	0.20			
	0.005	200	0.200	0.20	0.220	11.97	0.200
	0.001	800	0.200	0.83			
Bernoulli (P)	0.01	200	0.200	0.20			
	0.005	200	0.200	0.20	0.290	1.28	0.200
	0.001	800	0.200	0.83			
BayesPointMachine	0.01	900	0.000	7.83			
	0.005	not converged			0.011	2.20	0.032 ± 0.002*
	0.001	not converged					
ClickGraph	0.01	700	0.417	9.51			
	0.005	300	0.490	4.23	0.861	208	1.000
	0.001	200	0.501	2.98			
CoinBias	0.01	200	0.400	0.63			
	0.005	200	0.400	0.65	0.493	0.61	0.400
	0.001	700	0.398	2.32			
SurveyUnbias	0.01	700	0.964	3.46			
	0.005	1000	0.943	4.88	0.755	1.56	1.000
	0.001	2200	0.847	11.53			
TrueSkills	0.01	200	101.6	0.99			
	0.005	200	100.8	0.97	104.7	0.05	104.8 ± 0.681*
	0.001	200	100.2	1.03			
Altermu	0.01	not converged					
	0.005	not converged			0.000	0.16	0.114 ± 0.092*
	0.001	not converged					
NormalMixtures (P)	0.01	1100	0.236	49.48			
	0.005	300	0.477	14.98	0.276	50.4	0.275 ± 0.005*
	0.001	200	0.500	9.72			
TimeSeries	0.01	3100	-1.564	55.37			
	0.005	1100	-1.497	26.64	-1.494	3.79	-1.604 ± 0.021*
	0.001	2800	-1.347	2800			

Table 15: Comparison between Pyro and SOGA for MAP estimation. Models with '(P)' were pruned when SOGA was applied. True values are derived optimizing the exact posterior, or from samples (denoted with '*').

	Runtimes (s)			
Model	*SOGA*	*PSI*	*BLOG*	*VE*
IndianGPA	0.099	0.180	0.516	0.192
Scale	0.013	0.120	0.810	0.150
Tracking_1	0.042	to	0.803	0.143
Tracking_5	0.046	to	1.044	0.394
Tracking_10	0.046	to	1.330	0.670
Tracking_50	0.110	to	2.886	4.095
Tracking_100	0.192	to	5.054	8.885
Tracking_150	0.271	to	6.602	13.723

Table 16: Runtimes (in seconds) for the models proposed Wu et al. (2018) comprising mixtures of discrete and continuous distributions. We do not report values, since all methods identify the exact posterior.

5.2.3 Mixtures of Continuous and Discrete Distributions

Mixtures of continuous distributions and discrete probability masses appear in different domains such as in Kharchenko, Silberstein, and Scadden (2014), Pierson and Yau (2015), and Gao et al. (2017). Languages such as STAN and AQUA do not support them. Ad hoc methods have been proposed in Tolpin et al. (2016) and Nitti, De Laet, and De Raedt (2016). More recently Wu et al. (2018) extended the sampling techniques used in BLOG for more accurate inference. We test SOGA on the three benchmarks proposed by Wu et al. (2018) and compare its runtimes against PSI, BLOG, and variable elimination (VE) as implemented in Pyro (Obermeyer et al., 2019). *IndianGPA* and *Scale* are reported exactly as in the original paper, while the *Tracking_n* example from Section 4.1.1 is adapted since it was originally cast as a control problem. All examples have a Dirac delta posterior, which is computed exactly by all. However, SOGA is the fastest and the one which scales better as the number of steps n increases.

5.2.4 Bayesian Inference for Collaborative Filtering

Collaborative filtering models are well-known in machine learning for applications to recommendation systems (Koren, Rendle, and Bell, 2021). We target the problem of Bayesian inference on the latent factor model

		SOGA		*STAN*		*AQUA*		*VI*	
k	*Ground truth*	*time*	*value*	*time*	*value*	*time*	*value*	*time*	*value*
1	2	0.16	1.86	0.94	1.90	1.64	1.83	18.90	1.79
2	25	0.18	24.28	4.87	24.00	mem		25.10	23.93
3	-5	0.19	-5.79	7.63	-5.80	mem		26.40	-5.82
5	-30	0.22	-31.98	7.22	-32.00	mem		23.19	-31.47
10	151	0.30	149.75	5.42	150.00	mem		20.04	146.39
20	70	0.60	73.76	14.10	74.00	mem		23.18	69.92

Table 17: Runtimes (in seconds) for the collaborative filtering model $\mathcal{N}(cf_k, 1)$.

proposed in Hofmann and Puzicha (1999), which arises after a singular value decomposition and serves as the basis for solving an optimization problem (Zhao, Zhang, and J. Wang, 2013). The model assumes noisy observations sampled from $\mathcal{N}(cf_k, 1)$ where cf_k has the form $cf_k = a_1b_1 + \ldots + a_kb_k + c$, where a_i, b_i, and c are unknown latent variables. As noticed in Nishihara, Minka, and Tarlow (2013), performing Bayesian inference on these models is particularly difficult due to non-identifiability (Tsiatis, 1975) and symmetry (Neal, 1999) of the parameters. For example, switching the distributions of a_i and b_i will result in the same distribution for cf_k, which is the only one observed. In some cases, one may still want to model each parameter separately to allow for more flexibility. In this particular case, though not solving the problem of symmetry and non-identifiability, SOGA can estimate the distribution of cf_k faster than its competitors. Results are shown in Table 17 for various values of k. PSI results are not reported because the tool was able to produce a symbolic formula only up to $k = 3$; however, even in these cases, numerical integration of the non-simplified integrals required more than 600 s. Although STAN's estimates are accurate and close to SOGA's ones, its runtimes are longer due to the increased cost of sampling, which is exponential in the number of variables. As above, we attribute AQUA's out-of-memory error to its tensor based representation. For VI, we report results for the learning rate 0.005, which we found to be the one performing best in average, among the tested ones. A full set of experiment results are shown in Table 18. VI exhibits an accuracy comparable to SOGA's, but significantly longer runtimes. We observe, however, that thanks to vectorization, VI's runtimes do not significantly increase with k. Overall, the excellent runtime performance of SOGA is due the

		SOGA		*Pyro*		
k	*Ground truth*	*time*	*value*	*l.r.*	*time*	*value*
1	2	0.16	1.86	0.01	12.86	1.79
				0.005	18.90	1.84
				0.001	20.21	0.51
2	25	0.18	24.28	0.01	14.07	23.49
				0.005	25.10	23.93
				0.001	30.49	6.75
3	-5	0.19	-5.79	0.01	15.72	-5.51
				0.005	26.40	-5.82
				0.001	30.61	-1.86
5	-30	0.22	-31.98	0.01	18.49	-32.21
				0.005	23.19	-31.47
				0.001	31.49	-6.21
10	151	0.30	149.75	0.01	11.16	148.92
				0.005	20.04	146.39
				0.001	30.71	9.83
20	70	0.60	73.76	0.01	10.94	68.51
				0.005	23.18	69.92
				0.001	30.73	6.87

Table 18: Comparison between Pyro and SOGA for Variational Inference on Collaborative Filtering models.

particular structure of the models, which exhibit Gaussian posteriors on variables combined in a scalar product without introducing truncations that could slow down the computations.

Chapter 6

Conclusion

The present thesis presents two methods for the efficient and accurate analysis of two classes of transparent generative models, namely Markov population processes and probabilistic programs.

The first two chapters presents a method, called Dynamic Boundary Projection (DBP), that can be used to refine mean-field approximations of Markov population processes, i.e., Markov Chains representing the evolution of systems of N interacting agents. It is based on a system of differential equations for the transient probability distribution of a state space truncation, which is modulated by a mean-field approximation that essentially shifts such truncation across the whole state space. DBP is parameterized by a vector n that defines the size of the state space truncation. Thus, it leads to a family of approximations indexed by n for a given Markov population process. Importantly, for each distinct n, one needs to solve a different system of DBP equations in general. In this respect, it is different from related work on this subject. In particular, the method by Gast *et al.* computes the constants associated with the terms $1/N$ and $1/N^2$ of expansions of the mean-field equation for density-dependent Markov processes (Gast, Bortolussi, and Tribastone, 2019), where N is the scaling parameter. These constants correct the mean-field approximations for all N. Another difference with respect to the related work is that DBP neither makes scaling assumptions nor requires differentiability of the drift of the mean-field approximation.

Theoretically, asymptotic convergence to the original population process is proved when n tends to infinity (uniformly over a finite time horizon). As with many convergence results, however, this does not give directly useful insights as to the behavior of the approximation for finite n. Hence natural questions that arise may be based on establishing analogies with the literature on mean-field refinements, in particular, to study the following: i) whether DBP can be used to compute mean estimates of an arbitrary functional of a Markov population process; ii) whether suitable conditions hold to extend convergence to the steady state; iii) whether it is possible to establish rates of convergence or error bounds. Tackling these bounds is particularly difficult due to the non-linear nature of the systems yielded by DBP, for which a straightforward adaption of classic results does not work.

The present formulation of DBP is proved to be asymptotically correct only for systems exhibiting a bounded and Lipschitz drift. Relaxing these hypotheses might be of great interests to broaden the scope of applicability of DBP, especially since unbounded drift (in the form of mass-action rate functions) are very common in biological systems and can be the cause of unstable behaviours, oscillations and high-variance distributions Van Kampen, 1992.

The advantage of DBP is that the freedom to choose n can be exploited to improve the accuracy of the approximations. This has been used in the analysis of the malware propagation model (Gast, Bortolussi, and Tribastone, 2019; Benaim and Le Boudec, 2008a; Khouzani, Sarkar, and Altman, 2012) where DBP has been shown to avoid instabilities exhibited with $1/N$ and $1/N^2$ size expansions. The main disadvantage is that, since DBP relies on truncations of the state space, it is still subject to the well-known *curse of dimensionality* that affects Markov population processes. For systems with many dimensions, the values of the parameter n must be kept small to avoid large computational times. However, the numerical results have shown that, even for modest truncations, the classical mean-field estimations are improved. Suitable heuristics for the choice of n can help mitigate this problem, as we discussed in the example of the queuing system with egalitarian processor sharing. However, these heuristics are model-dependent, and in general, it is unclear how to fix n to achieve better approximations. In this regard, deriving exact error bounds depending on n could be particularly interesting.

This motivates the extension of the method proposed in Chapter 3. To lessen the curse of dimensionality we propose to study the dynamics of a rescaled Markov population process. This method sees the state space of the original process as a multi-dimensional grid of size $h = 1$ and covers it with a coarser grid with length $h > 1$, rescaling transition rates by a factor $\frac{1}{h}$. The resulting process has an exponentially smaller state space with respect to the number of dimensions. Moreover, it allows for different components of the original process to be rescaled with different parameters as long as they preserve some conservation properties. Examples show that the ME of the rescaled process can be solved in less time than the original one, where the difference in seconds can be up to orders of magnitude. The scaling can be coupled with DBP, resulting in a double advantage: the number of equations needed to apply DBP is reduced, and, using the same h, better accuracy can be achieved than applying h-scaling directly.

A similar idea to the one proposed in this chapter was proposed in Ciocchetta et al., 2009 in the context of biochemical networks modeling using the process algebra Bio-PEPA. There the terminology *CTMC with levels* was used to denote different discretization of the same Bio-PEPA model, obtained by applying the equivalent of different scaling as the ones proposed here. While the re-scaling of jump magnitudes and rate function is the same, our work substantially differs from the one in Ciocchetta et al., 2009 for several reasons. First, we use the proposed scaling for approximating a given system, while in Ciocchetta et al., 2009 the different discretizations are treated as systems on their own. Second, our scaling can be applied to a generic Markov population process, abstracting away from a process algebraic description. Third, we propose the introduction of different re-scaling factors for different classes of agents, stating formally under which conditions this is possible. Theoretically, in Ciocchetta et al., 2009, it is proved that as the scaling parameter h tends to 0, the associated family of systems satisfies the density-dependent assumption, so the associated process's dynamics tend to the mean-field limit. Here we study the case $h > 1$ and its convergence properties to a mean-field or to an LNA, proving that these limits are preserved in the re-scaled sequence.

Again, a natural question is whether it is possible to establish an error bound depending on h or at least some convergence rate toward such limits. While this is unclear, we leave these questions open for future work.

Chapters 4 and 5 deal with a different kind of generative models, namely probabilistic programs. We see a PP as a distribution transformer and propose an approximate analytical method to infer the final distribution carried by a bounded PP, once the initial distribution is fixed.

Our method is based on the definition of Gaussian Semantics, a family of approximations parametrized by the moment order to match against a Gaussian mixture at each location of a probabilistic program. The universal approximation theorem states that such a family converges to the true semantics. Although, in principle, any program location could be treated with different moment-order matching, in practice, this is a difficult problem that requires solving a system of nonlinear equations.While the system is guaranteed to have a solution, finding it using SMT solvers over reals or numerical methods yields poor results, due to long computational times and numerical instability. Therefore we leave open the general problem of implementing Gaussian Semantics for any order of moments. However, we provide an analytical method that matches second-order moments of the exact probabilistic semantics (SOGA). The numerical results for the case studies demonstrate high quality of the approximation and that SOGA complements state-of-the-art methods for probabilistic inference and in particular for inference on models with mixtures of discrete and continuous distributions and for Bayesian inference on collaborative filtering models. Due to the efficiency shown by SOGA, we believe that in these cases our method can effectively be used as an alternative to sampling. Exploring other tailored applications of our method is left for future work.

While SOGA performed satisfactorily on all tested benchmarks, it could not be applied to some of the models from the same repositories, due to the limitations of our syntax. Extending the latter to include general distributions depending on non-constant parameters, unbounded loops and non-polynomial functions would widen its scope of applicability. A possible way to overcome the former restriction could be learning offline the approximating distributions as a function of the variable parameters, but how to do this efficiently is currently not clear, even though of great interest. For what concerns unbounded loops, we observed that for almost surely terminating programs, the loops can be unrolled for a finite number of iterations so that the error committed in the approximation is arbitrarily small. This suggests that increasing the number of unrolled iterations together with the number of moments matched should preserve our convergence theorem, even in the case of almost surely terminating unbounded programs. Similarly, one could exploit convergence results for polynomial approximations to extend the convergence result to sequences of polynomial programs that approximate programs featuring non-polynomial functions, similarly to what has been done in Kofnov et al., 2022; Kofnov et al., 2023. We leave the possibility to explore these extensions of our convergence results in future work.

To improve the accuracy of the approximation, one might devise algorithms for higher-order moments. While an extension to *exact* higher-order moment matching seems hard, a relaxed moment problem could be defined as an optimization problem, as proposed in Hansen, 2010.

Finally, another interesting direction is given by differentiability of Gaussian Semantics, which was not formally proved in the current paper. Differentiable programming has been gaining popularity in the past few years thanks to its wide applicability in machine learning application Baydin et al., 2018. Providing an approximating differentiable semantics for probabilistic programs would be of great interest to perform gradient based optimization on programs, as done for example in Cui and H. Zhu, 2021.

Bibliography

Albarghouthi, Aws et al. (2017). “Fairsquare: probabilistic verification of program fairness”. In: *Proceedings of the ACM on Programming Languages* 1.OOPSLA, pp. 1–30.

Anderson, David F and Masanori Koyama (2012). “Weak error analysis of numerical methods for stochastic models of population processes”. In: *Multiscale Modeling & Simulation* 10.4, pp. 1493–1524.

Barthe, Gilles et al. (2016). “Synthesizing probabilistic invariants via Doob’s decomposition”. In: *International Conference on Computer Aided Verification*. Springer, pp. 43–61.

Bartocci, Ezio, Laura Kovács, and Miroslav Stankovič (2020). “Mora-automatic generation of moment-based invariants”. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, pp. 492–498.

Baydin, Atilim Gunes et al. (2018). “Automatic differentiation in machine learning: a survey”. In: *Journal of Marchine Learning Research* 18, pp. 1–43.

Benaim, Michel and Jean-Yves Le Boudec (2008a). “A class of mean field interaction models for computer and communication systems”. In: *Performance evaluation* 65.11-12, pp. 823–838.

— (2008b). “A class of mean field interaction models for computer and communication systems”. In: *Performance evaluation* 65.11-12, pp. 823–838.

Billingsley, Patrick (2008). *Probability and measure*. John Wiley & Sons.

— (2013). *Convergence of probability measures*. John Wiley & Sons.

Bingham, Eli et al. (2019). “Pyro: Deep universal probabilistic programming”. In: *The Journal of Machine Learning Research* 20.1, pp. 973–978.

Bishop, Christopher M and Nasser M Nasrabadi (2006). *Pattern recognition and machine learning*. Vol. 4. 4. Springer.

Bolch, Gunter et al. (2005). *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley.

Bortolussi, Luca and Richard A Hayden (2013). “Bounds on the deviation of discrete-time Markov chains from their mean-field model”. In: *Performance Evaluation* 70.10, pp. 736–749.

Bortolussi, Luca, Jane Hillston, et al. (2013a). “Continuous approximation of collective system behaviour: A tutorial”. In: *Performance Evaluation* 70.5, pp. 317–349.

— (2013b). “Continuous approximation of collective system behaviour: A tutorial”. In: *Performance Evaluation* 70.5, pp. 317–349.

Boyen, Xavier and Daphne Koller (1998). “Tractable inference for complex stochastic processes”. In: *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pp. 33–42.

Bramson, Maury, Yi Lu, and Balaji Prabhakar (2010). “Randomized Load Balancing with General Service Time Distributions”. In: *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 275–286.

Cardelli, Luca et al. (2017). “ERODE: A tool for the evaluation and reduction of ordinary differential equations”. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, pp. 310–328.

Carpenter, Bob et al. (2017). “Stan: A probabilistic programming language”. In: *Journal of statistical software* 76.1.

Cecchi, Fabio, Sem C Borst, and JSH van Leeuwaardena (2015). “Mean-field analysis of ultra-dense csma networks”. In: *ACM SIGMETRICS Performance Evaluation Review* 43.2, pp. 13–15.

Chaganty, Arun, Aditya Nori, and Sriram Rajamani (2013). “Efficiently sampling probabilistic programs via program analysis”. In: *Artificial Intelligence and Statistics*. PMLR, pp. 153–160.

Chaintreau, Augustin, Jean-Yves Le Boudec, and Nikodin Ristanovic (2009). “The age of gossip: spatial mean field regime”. In: 37.1, pp. 109–120.

Chakarov, Aleksandar and Sriram Sankaranarayanan (2014). “Expectation invariants for probabilistic program loops as fixed points”. In: *International Static Analysis Symposium*. Springer, pp. 85–100.

Chaudhuri, Swarat and Armando Solar-Lezama (2010). “Smooth interpretation”. In: *ACM Sigplan Notices* 45.6, pp. 279–291.

— (2011). “Smoothing a program soundly and robustly”. In: *International Conference on Computer Aided Verification*. Springer, pp. 277–292.

Chen, Mingshuai et al. (2022). “Does a program yield the right distribution? Verifying probabilistic programs via generating functions”.

In: *International Conference on Computer Aided Verification*. Springer, pp. 79–101.

Ciocchetta, Federica et al. (2009). "Some investigations concerning the CTMC and the ODE model derived from Bio-PEPA". In: *Electronic Notes in Theoretical Computer Science* 229.1, pp. 145–163.

Cousot, Patrick and Radhia Cousot (1977). "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints". In: *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pp. 238–252.

Cover, Thomas M (1999). *Elements of information theory*. John Wiley & Sons.

Cui, Guofeng and He Zhu (2021). "Differentiable synthesis of program architectures". In: *Advances in Neural Information Processing Systems* 34, pp. 11123–11135.

Darling, R.W.R. and J.R. Norris (2008). "Differential equation approximations for Markov chains". In: *Probab. Surveys* 5, pp. 37–79. DOI: 10.1214/07-PS121.

Darling, RWR and James R Norris (2008). "Differential equation approximations for Markov chains". In: *Probability surveys* 5, pp. 37–79.

Dijk, Dick van, Timo Teräsvirta, and Philip Hans Franses (2002). "Smooth transition autoregressive models—a survey of recent developments". In: *Econometric reviews* 21.1, pp. 1–47.

Dinh, Khanh N and Roger B Sidje (2016). "Understanding the finite state projection and related methods for solving the chemical master equation". In: *Physical biology* 13.3, p. 035003.

Ethier, Stewart N and Thomas G Kurtz (2009a). *Markov processes: characterization and convergence*. Vol. 282. John Wiley & Sons.

— (2009b). *Markov processes: characterization and convergence*. Vol. 282. John Wiley & Sons.

Filieri, Antonio, Corina S Păsăreanu, and Willem Visser (2013). "Reliability analysis in symbolic pathfinder". In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, pp. 622–631.

Florescu, Ionut (2014). *Probability and stochastic processes*. John Wiley & Sons.

Gao, Weihao et al. (2017). "Estimating mutual information for discrete-continuous mixtures". In: *Advances in neural information processing systems* 30.

Gast, Nicolas (2017). “Expected values estimated via mean-field approximation are $1/N$-accurate”. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1.1, pp. 1–26.

Gast, Nicolas, Luca Bortolussi, and Mirco Tribastone (2019). “Size expansions of mean field approximation: Transient and steady-state analysis”. In: *Performance Evaluation* 129, pp. 60–80.

Gast, Nicolas and Gaujal Bruno (2010). “A mean field model of work stealing in large-scale systems”. In: *ACM SIGMETRICS Performance Evaluation Review* 38.1, pp. 13–24.

Gast, Nicolas and Benny Van Houdt (2015). “Transient and steady-state regime of a family of list-based cache replacement algorithms”. In: *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 123–136.

— (2017). “A refined mean field approximation”. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1.2, pp. 1–28.

Gatys, Leon, Alexander Ecker, and Matthias Bethge (2016). “A Neural Algorithm of Artistic Style”. In: *Journal of Vision* 16.12, pp. 326–326.

Gehr, Timon, Sasa Misailovic, and Martin Vechev (2016). “PSI: Exact symbolic inference for probabilistic programs”. In: *International Conference on Computer Aided Verification*. Springer, pp. 62–83.

Gelman, Andrew et al. (2013). *Bayesian data analysis*. CRC press.

Gillespie, Daniel T (2007). “Stochastic simulation of chemical kinetics”. In: *Annu. Rev. Phys. Chem.* 58, pp. 35–55.

Goodfellow, Ian et al. (2014). “Generative adversarial nets”. In: *Advances in neural information processing systems* 27.

Goodman, Noah D et al. (2008). “Church: a language for generative models”. In: *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 220–229.

Gordon, Andrew D et al. (2014). “Probabilistic programming”. In: *Future of Software Engineering Proceedings*, pp. 167–181.

Grima, Ramon (2010). “An effective rate equation approach to reaction kinetics in small volumes: Theory and application to biochemical reactions in nonequilibrium steady-state conditions”. In: *The Journal of chemical physics* 133.3.

Gu, Ming and Stanley C Eisenstat (1995). “A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem”. In: *SIAM Journal on Matrix Analysis and Applications* 16.1, pp. 172–191.

Gupta, Ankit, Jan Mikelson, and Mustafa Khammash (2017). “A finite state projection algorithm for the stationary solution of the chemical master equation”. In: *The Journal of chemical physics* 147.15, p. 154101.

Hansen, Lars Peter (2010). “Generalized method of moments estimation”. In: *Macroeconometrics and Time series Analysis*. Springer, pp. 105–118.

Hart, Andrew G and Richard L Tweedie (2012). “Convergence of invariant measures of truncation approximations to Markov processes”. In.

Hastings, W Keith (1970). “Monte Carlo sampling methods using Markov chains and their applications”. In.

Hoffman, Matthew D et al. (2013). “Stochastic variational inference”. In: *Journal of Machine Learning Research*.

Hofmann, Thomas and Jan Puzicha (1999). “Latent class models for collaborative filtering”. In: *IJCAI*. Vol. 99. 1999.

Holtzen, Steven, Guy Van den Broeck, and Todd Millstein (2020). “Scaling exact inference for discrete probabilistic programs”. In: *Proceedings of the ACM on Programming Languages* 4.OOPSLA, pp. 1–31.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5, pp. 359–366.

Huang, Zixin, Saikat Dutta, and Sasa Misailovic (2021). “Aqua: Automated quantized inference for probabilistic programs”. In: *International Symposium on Automated Technology for Verification and Analysis*. Springer, pp. 229–246.

Jordan, Michael I et al. (1999). “An introduction to variational methods for graphical models”. In: *Machine learning* 37.2, pp. 183–233.

Kan, Raymond and Cesare Robotti (2017). “On moments of folded and truncated multivariate normal distributions”. In: *Journal of Computational and Graphical Statistics* 26.4, pp. 930–934.

Katoen, Joost-Pieter et al. (2010). “Linear-invariant generation for probabilistic programs”. In: *International Static Analysis Symposium*. Springer, pp. 390–406.

Kharchenko, Peter V, Lev Silberstein, and David T Scadden (2014). “Bayesian approach to single-cell differential expression analysis”. In: *Nature methods* 11.7, pp. 740–742.

Khouzani, MHR, Saswati Sarkar, and Eitan Altman (2012). “Maximum damage malware attack in mobile wireless networks”. In: *IEEE/ACM Transactions on Networking* 20.5, pp. 1347–1360.

Kingma, Durk P et al. (2014). “Semi-supervised learning with deep generative models”. In: *Advances in neural information processing systems* 27.

Kofnov, Andrey et al. (2022). “Moment-based invariants for probabilistic loops with non-polynomial assignments”. In: *International Conference on Quantitative Evaluation of Systems*. Springer, pp. 3–25.

Kofnov, Andrey et al. (2023). "Exact and Approximate Moment Derivation for Probabilistic Loops With Non-Polynomial Assignments". In: *arXiv preprint arXiv:2306.07072*.

Koren, Yehuda, Steffen Rendle, and Robert Bell (2021). "Advances in collaborative filtering". In: *Recommender systems handbook*, pp. 91–142.

Kozen, Dexter (1979). "Semantics of probabilistic programs". In: *20th Annual Symposium on Foundations of Computer Science (FOCS 1979)*. IEEE, pp. 101–114.

— (1983). "A probabilistic pdl". In: *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pp. 291–297.

Kucukelbir, Alp et al. (2015). "Automatic variational inference in Stan". In: *Advances in neural information processing systems* 28.

Kullback, Solomon and Richard A Leibler (1951). "On information and sufficiency". In: *The annals of mathematical statistics* 22.1, pp. 79–86.

Kuntz, Juan et al. (2019). "Stationary distributions of continuous-time Markov chains: a review of theory and truncation-based approximations". In: *arXiv preprint arXiv:1909.05794*.

Kurtz, Thomas G (1970). "Solutions of ordinary differential equations as limits of pure jump Markov processes". In: *Journal of applied Probability* 7.1, pp. 49–58.

— (1978). "Strong approximation theorems for density dependent Markov chains". In: *Stochastic Processes and their Applications* 6.3, pp. 223–240.

Lasserre, Jean Bernard (2009). *Moments, positive polynomials and their applications*. Vol. 1. World Scientific.

Laurel, Jacob and Sasa Misailovic (2020). "Continualization of probabilistic programs with correction". In: *European Symposium on Programming*. Springer, Cham, pp. 366–393.

Liu, Xin, Lei Ying, et al. (2020). "Beyond Scaling: Calculable Error Bounds of the Power-of-Two-Choices Mean-Field Model in Heavy-Traffic". In: *arXiv preprint arXiv:2012.06613*.

Liu, Yuanyuan and Wendi Li (2018). "Error bounds for augmented truncation approximations of Markov chains via the perturbation method". In: *Advances in Applied Probability* 50.2, pp. 645–669.

Liu, Yuanyuan, Wendi Li, and Hiroyuki Masuyama (2018). "Error bounds for augmented truncation approximations of continuous-time Markov chains". In: *Operations Research Letters* 46.4, pp. 409–413.

Lo, J (1972). "Finite-dimensional sensor orbits and optimal nonlinear filtering". In: *IEEE Transactions on information theory* 18.5, pp. 583–588.

Mansinghka, Vikash, Daniel Selsam, and Yura Perov (2014). "Venture: a higher-order probabilistic programming platform with programmable inference". In: *arXiv preprint arXiv:1404.0099*.

Massoulié, Laurent and Milan Vojnovic (2005). "Coupon replication systems". In: *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and modeling of computer systems*, pp. 2–13.

Masuyama, Hiroyuki (2017). "Error bounds for last-column-block-augmented truncations of block-structured Markov chains". In: *Journal of the Operations Research Society of Japan* 60.3, pp. 271–320.

Milch, Brian, Bhaskara Marthi, and Stuart Russell (2004). "BLOG: Relational modeling with unknown objects". In: *ICML 2004 workshop on statistical relational learning and its connections to other fields*, pp. 67–73.

Minnebo, Wouter and Benny Van Houdt (2013). "A fair comparison of pull and push strategies in large distributed networks". In: *IEEE/ACM Transactions on Networking* 22.3, pp. 996–1006.

Mitzenmacher, Michael (2001). "The power of two choices in randomized load balancing". In: *IEEE Transactions on Parallel and Distributed Systems* 12.10, pp. 1094–1104.

Moosbrugger, Marcel et al. (2022). "This is the moment for probabilistic loops". In: *Proceedings of the ACM on Programming Languages* 6.OOPSLA2, pp. 1497–1525.

Munsky, Brian and Mustafa Khammash (2006a). "The finite state projection algorithm for the solution of the chemical master equation". In: *The Journal of Chemical Physics* 124.4, p. 044104.

— (2006b). "The finite state projection algorithm for the solution of the chemical master equation". In: *The Journal of chemical physics* 124.4, p. 044104.

— (2007). "A multiple time interval finite state projection algorithm for the solution to the chemical master equation". In: *Journal of Computational Physics* 226.1, pp. 818–835.

Narayanan, Praveen et al. (2016). "Probabilistic inference by program transformation in Hakaru (system description)". In: *International Symposium on Functional and Logic Programming*. Springer, pp. 62–79.

Neal, Radford M (1999). "Erroneous results in "Marginal likelihood from the Gibbs output"". In: *minmeo, University of Toronto*.

Ng, Andrew and Michael Jordan (2001). "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes". In: *Advances in neural information processing systems* 14.

Nishihara, Robert, Thomas Minka, and Daniel Tarlow (2013). "Detecting parameter symmetries in probabilistic models". In: *arXiv preprint arXiv:1312.5386*.

Nitti, Davide, Tinne De Laet, and Luc De Raedt (2016). "Probabilistic logic programming for hybrid relational domains". In: *Machine Learning* 103.3, pp. 407–449.

Nori, Aditya et al. (2014). "R2: An efficient MCMC sampler for probabilistic programs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 28. 1.

Obermeyer, Fritz et al. (2019). "Tensor variable elimination for plated factor graphs". In: *International Conference on Machine Learning*. PMLR, pp. 4871–4880.

Ohno, Hiroshi (2020). "Auto-encoder-based generative models for data augmentation on regression problems". In: *Soft Computing* 24.11, pp. 7999–8009.

Olmedo, Federico et al. (2018). "Conditioning in probabilistic programming". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 40.1, pp. 1–50.

Parekh, Abhay K and Robert G Gallager (1993). "A generalized processor sharing approach to flow control in integrated services networks: the single-node case". In: *IEEE/ACM Transactions on Networking* 3, pp. 344–357.

— (1994). "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case". In: *IEEE/ACM Transactions on Networking* 2.2, pp. 137–150.

Pérez, Dilcia and Yamilet Quintana (2006). "A survey on the Weierstrass approximation theorem". In: *arXiv preprint math/0611038*.

Perko, Lawrence (2013). *Differential equations and dynamical systems*. Vol. 7. Springer Science & Business Media.

Pfeffer, Avi (2001). "IBAL: A probabilistic rational programming language". In: *IJCAI*. Citeseer, pp. 733–740.

— (2009). "Figaro: An object-oriented probabilistic programming language". In: *Charles River Analytics Technical Report* 137, p. 96.

Pierson, Emma and Christopher Yau (2015). "ZIFA: Dimensionality reduction for zero-inflated single-cell gene expression analysis". In: *Genome biology* 16.1, pp. 1–10.

Randone, Francesca, Luca Bortolussi, and Mirco Tribastone (2021). "Refining mean-field approximations by dynamic state truncation". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5.2, pp. 1–30.

Randone, Francesca, Luca Bortolussi, and Mirco Tribastone (2022). "Jump Longer to Jump Less: Improving Dynamic Boundary Projection with h-Scaling". In: *Quantitative Evaluation of Systems: 19th International Conference, QEST 2022, Warsaw, Poland, September 12–16, 2022, Proceedings*. Springer, pp. 150–170.

Saad, Feras A, Martin C Rinard, and Vikash K Mansinghka (2021). "SPPL: probabilistic programming with fast exact symbolic inference". In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 804–819.

Schmüdgen, Konrad (2017). *The moment problem*. Vol. 9. Springer.

Seneta, E (1967). "Finite approximations to infinite non-negative matrices". In: *Proc. Camb. Phil. Soc.* Vol. 63. 4, p. 983.

— (1968). "Finite approximations to infinite non-negative matrices, II: refinements and applications". In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 64. 2. Cambridge University Press, pp. 465–470.

Singh, Abhyudai and Joao Pedro Hespanha (2006). "Lognormal moment closures for biochemical reactions". In: *IEEE 45th Conference on Decision and Control (CDC)*, pp. 2063–2068.

Skiena, SS (2008). *The Algorithm Design Manual. Springer Publishing Company.*

Song, Yang et al. (2021). "Maximum likelihood training of score-based diffusion models". In: *Advances in Neural Information Processing Systems* 34, pp. 1415–1428.

Tierney, Luke and Joseph B Kadane (1986). "Accurate approximations for posterior moments and marginal densities". In: *Journal of the american statistical association* 81.393, pp. 82–86.

Tolpin, David et al. (2016). "Design and implementation of probabilistic programming language anglican". In: *Proceedings of the 28th Symposium on the Implementation and Application of Functional programming Languages*, pp. 1–12.

Tsiatis, Anastasios (1975). "A nonidentifiability aspect of the problem of competing risks." In: *Proceedings of the National Academy of Sciences* 72.1, pp. 20–22.

Tsitsiklis, John N and Kuang Xu (2011). "On the power of (even a little) centralization in distributed processing". In: *ACM SIGMETRICS Performance Evaluation Review* 39.1, pp. 121–132.

Tweedie, Richard L (1971). "Truncation procedures for non-negative matrices". In: *Journal of Applied Probability* 8.2, pp. 311–320.

Tweedie, Richard L (1973). “The calculation of limit probabilities for denumerable Markov processes from infinitesimal properties”. In: *Journal of Applied Probability*, pp. 84–99.

Van Houdt, Benny (2013). “A mean field model for a class of garbage collection algorithms in flash-based solid state drives”. In: *ACM SIGMETRICS Performance Evaluation Review* 41.1, pp. 191–202.

Van Kampen, N. G. (2007). *Stochastic Processes in Physics and Chemistry*. 3rd. Elsevier.

Van Kampen, Nicolaas Godfried (1992). *Stochastic processes in physics and chemistry*. Vol. 1. Elsevier.

Vasantam, Thirupathaiah and Ravi R Mazumdar (2019). “Fluctuations Around the Mean-Field for a Large Scale Erlang Loss System Under the SQ (d) Load Balancing”. In: *2019 31st International Teletraffic Congress (ITC 31)*. IEEE, pp. 1–9.

Virtanen, Pauli et al. (2020). “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Wang, Sida, Arun Tejasvi Chaganty, and Percy S Liang (2015). “Estimating mixture models via mixtures of polynomials”. In: *Advances in Neural Information Processing Systems* 28.

Wick, Gian-Carlo (1950). “The evaluation of the collision matrix”. In: *Physical review* 80.2, p. 268.

Wolfram Research, Inc. (2022). *Mathematica*. Version 13.2. URL: https://www.wolfram.com/mathematica.

Wu, Yi et al. (2018). “Discrete-continuous mixtures in probabilistic programming: Generalized semantics and inference algorithms”. In: *International Conference on Machine Learning*. PMLR, pp. 5343–5352.

Xie, Qiaomin et al. (2015). “Power of d choices for large-scale bin packing: A loss model”. In: *ACM SIGMETRICS Performance Evaluation Review* 43.1, pp. 321–334.

Ying, Lei (2016). “On the approximation error of mean-field models”. In: *ACM SIGMETRICS Performance Evaluation Review* 44.1, pp. 285–297.

— (2017). “Stein’s method for mean field approximations in light and heavy traffic regimes”. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1.1, pp. 1–27.

Zhao, Xiaoxue, Weinan Zhang, and Jun Wang (2013). “Interactive collaborative filtering”. In: *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 1411–1420.

Zhou, Ding-Xuan (2020). “Universality of deep convolutional neural networks”. In: *Applied and computational harmonic analysis* 48.2, pp. 787–794.

Zhou, Yuan et al. (2020). “Divide, conquer, and combine: a new inference strategy for probabilistic programs with stochastic support”. In: *International Conference on Machine Learning*. PMLR, pp. 11534–11545.

Zhu, Lulai, Giuliano Casale, and Iker Perez (2020). “Fluid approximation of closed queueing networks with discriminatory processor sharing”. In: *Performance Evaluation* 139, p. 102094.