rmf_tool – A library to Compute (Refined) Mean Field Approximation(s)

Sebastian Allmeier
Univ. Grenoble Alpes
Inria
Grenoble, France
sebastian.allmeier@inria.fr

Nicolas Gast
Univ. Grenoble Alpes
Inria
Grenoble, France
nicolas.gast@inria.fr

ABSTRACT

Mean field approximation is a powerful technique to study the performance of large stochastic systems represented as systems of interacting objects. Applications include load balancing models, epidemic spreading, cache replacement policies, or large-scale data centers, for which mean field approximation gives very accurate estimates of the transient or steady-state behaviors. In a series of recent papers [9, 7], a new and more accurate approximation, called the *refined* mean field approximation is presented. Yet, computing this new approximation can be cumbersome. The purpose of this paper is to present a tool, called *rmf_tool*, that takes the description of a mean field model, and can numerically compute its mean field approximations and refinement.

1. INTRODUCTION

Mean field approximation is widely applied to analyze the behavior of large stochastic systems. It applies to systems composed of n interacting objects. The idea of the approximation is to consider that objects within the system evolve independently. This transforms the study of a multi-dimensional stochastic process into much smaller stochastic processes that are weakly coupled. Under mild conditions, the mean field approximation is described by a finite set of deterministic ordinary differential equations (ODEs). As such, it can be simulated at low computational cost. Mean field approximation finds widespread use in fields such as epidemic spreading [14, 3], load balancing strategies [15, 12], the study of cache replacement strategies [8] or SSDs [17].

Classical models to which mean field approximation applies are the class of density dependent population processes (DDPPs, [11]), whose definition is recalled in Section 2 – epidemic spreading or load balancing models are typical examples of DDPPs. If X is a density dependent population process in d dimensions, its mean field approximation is the solution of a system of non-linear ODEs $\dot{x} = f(x)$ where $f : \mathbb{R}^d \to \mathbb{R}^d$ is called the drift of the system. Computing the mean field approximation can be easily automated, as the drift f can be expressed easily from the model's definition. Our tool incorporates this but, more importantly, allows going further.

Building on mean field approximation, the authors of [6, 7] introduce the notion of *refined* mean field approximation. This approximation consists in adding an expansion term to the original approximation. Denoting by x the value of the mean field approximation, it is shown in [5] that there exists a deterministic quantity $v(t)$ such that:

$$\mathbb{E}[X(t)] = \underbrace{x(t) + \frac{1}{n}v(t)}_{\text{refined m.f. approx.}} + O(\frac{1}{n^2}).$$

The quantity $v(t)$ is the solution of a time-inhomogeneous linear ODE. As shown in the aforementioned papers, the construction of this set of ODEs is direct from the model description but involves computing the derivatives of the drift, which can be cumbersome.

The purpose of *rmf_tool* – the refined mean field tool – is to make mean field and refined mean field approximation easily computable. Our tool is composed of a Python library. The tool takes as input a description of the system, which can be either a density dependent population process or a heterogeneous population model, and can be used to compute the mean field and refined mean field approximations numerically. The tool relies on standard libraries (like `numpy` and `scipy`) to construct and solve the corresponding ODEs. The tool is provided with a series of examples to demonstrate its expressiveness and the accuracy of the various approximations.

Related tools. There exist a large number of tools that provide methods to construct and simulate stochastic population models. Yet, to the best of our knowledge, the only tool that provides a way to analyze size expansion methods (which are essentially equivalent to our refined mean field approximation) is the iNA software of [16]. The iNA is a complete simulation toolbox (that includes its own graphical interface). Compared to this software, we use a more lightweight approach by providing a pure python library that can be easily integrated.

Roadmap. The paper is centered around the tool. We first describe the set of models to which the tool applies in Section 2, along with examples on how they can be defined within the tool. We then describe what are the mean field and refined mean field approximation, and how one can use the tool to compute them in Section 3. We detail some technical challenges in Section 4 and conclude in Section 5.

Reproducibility. Our tool is provided as an open-source software at `https://github.com/ngast/rmf_tool`. The code to reproduce the current paper along with all figures is available at `https://gitlab.inria.fr/gast/toolpaper_rmf`.

TOSME 2021 Online Conference

2. MODELS

The tool that we develop accepts three kinds of models: homogeneous population processes (HomPP), density dependent population processes (DDPPs) and heterogeneous population models (HetPP). First, we describe the notion of the HomPP of which DDPP and HetPP are generalizations.

2.1 Homogeneous population process

Population Processes are widely used to describe the evolution of a number of interacting objects (or individuals). A homogeneous population model consists of n interacting objects that each evolves in a finite state space $\{1 \dots d\}$. All objects have similar transition rates which are a combination of unilateral and pairwise interactions, i.e. objects can change their state with or without interacting with one other member of the population. Let $X_s(t)$ be the fraction of objects that are in state s at time t. We assume that $\mathbf{X} = (X_1 \dots X_d)$ is a continuous time Markov chain whose transitions are such that for all state $s, s', \tilde{s}, \tilde{s}'$:

(Uni.) An object in state s moves to state s' at rate $a_{s,s'}$.

(Pair.) A pair of objects in state $(s, \tilde{s})$ moves to state $s', \tilde{s}'$ at rate $b_{s,\tilde{s},s',\tilde{s}'}/n$.

Note that for pairwise interactions, the rate is scaled by $1/n$ as the number of pairs of objects is n times larger than the number of objects.

Example: the SIS model. One of the simplest examples of population process is the epidemic model called the SIS model. In an SIS model, each object can be in one of the two states S (susceptible) or I (infected). Susceptible objects can become infected from an external source (unilateral transition) or when meeting an infected individual (pairwise transition). An infected individual can recover and become susceptible again (unilateral transition). We assume that an individual becomes infected at rate α by an external source, and recovers at rate β. Moreover, assume that the rate at which two individuals meet is γ/n and that when a susceptible meets an infected individual, the susceptible gets infected.

With our tool, we define a class called HomPP for which we specify the transition rates and an initial state. For the SIS model above, with $\alpha, \beta, \gamma = 1$, we write:

```
import rmf_tool as rmf
model = rmf.HomPP()
d, S, I = 2, 0, 1
A, B = np.zeros((d, d)), np.zeros((d, d, d, d))
A[S, I] = 1                    # \alpha
A[I, S] = 1                    # \beta
B[S, I, I, I] = 1              # \gamma
model.add_rate_tensors(A, B)
```

The specified model can be used to simulate stochastic trajectories of the underlying process for various population sizes. It can also be used to compute the mean field approximation and the refinements (see Section 3). For instance, if one wants to simulate a trajectory for a population of size $n = 1000$, where all individuals are susceptible in the initial state, one would write:

```
model.set_initial_state([1,0])
t, X = model.simulate(N=1000, time=2)
```

State representation. Recall that $X_s(t)$ is the fraction of objects in state s at time t. The transitions of such a model can be expressed[1] as:

(Uni.) When an object moves from s to s', this changes $\mathbf{X}$ into $\mathbf{X} + \frac{1}{n}(\mathbf{e}_{s'} - \mathbf{e}_s)$. As $nX_s(t)$ is the number of objects in state s, this transition occurs at rate $na_{s,s'}X_s(t)$.

(Pair.) When a pair moves from $(s, \tilde{s})$ to $(s', \tilde{s}')$, this changes $\mathbf{X}$ into $\mathbf{X} + \frac{1}{n}(\mathbf{e}_{s'} + \mathbf{e}_{\tilde{s}'} - \mathbf{e}_{\tilde{s}} - \mathbf{e}_s)$. This transition occurs at rate $nb_{s,\tilde{s},s',\tilde{s}'}X_s(t)X_{\tilde{s}(t)}$.

Written in a compact way, those transitions are:

$$\mathbf{x} \to \mathbf{x} + \frac{1}{n}(\mathbf{e}_{s'} - \mathbf{e}_s) \qquad \text{at rate } na_{s,s'}x_s \quad (1)$$

$$\mathbf{x} \to \mathbf{x} + \frac{1}{n}(\mathbf{e}_{s'} + \mathbf{e}_{\tilde{s}'} - \mathbf{e}_{\tilde{s}} - \mathbf{e}_s) \qquad \text{at rate } nb_{s,\tilde{s},s',\tilde{s}'}x_s x_{\tilde{s}}. \quad (2)$$

2.2 Density dependent populations process

The class of homogeneous population model that we define is a subclass of density dependent population processes (DDPPs) that are introduced by Kurtz in the 70s [11]. For a given n, a DDPP defines a stochastic process $\mathbf{X} \in \mathbb{R}^d$. The transitions of the process are specified by a finite set of vectors $\mathcal{L} \subset \mathbb{R}^d$, and a set of corresponding rate functions $\beta_\ell : \mathbb{R}^d \to \mathbb{R}^+$ for all $\ell \in \mathcal{L}$. The process $\mathbf{X}$ jumps from $\mathbf{x}$ to $\mathbf{x} + \ell/n$ at rate $n\beta_\ell(\mathbf{x})$.

It should be clear from Equation (1)-(2) that HomPP is a special case of DDPP. DDPPs generalize HomPP since they allow to choose arbitrary transition rates as opposed to combinations of unilateral and pairwise transition. In the case where $nX_s(t)$ denotes the number of individuals in state s at time t, the vector $\ell \in \mathcal{L}$ indicates how many individuals are created or destroyed by a transition. For instance, if $d = 3$, $\ell = (1, -1, 0)$ corresponds to having one additional individual in state 1 and one less in state 2 (this occurs typically when one individual moves from state 2 to state 1), $\ell = (0, 0, 2)$ corresponds to the creation of two additional individuals in state 3.

The SIS model as a DDPP. To illustrate the relation between DDPPs and HomPP, consider the SIS model defined in the previous section and recall that $(X_S(t), X_I(t))$ is the fraction of susceptible and of infected individuals. The transitions $\ell \in \mathcal{L}$ and their corresponding rates β_ℓ are:

Event	Transition ℓ	Rate $\beta_\ell(\mathbf{x})$
infection from ext. source	(-1,1)	$\alpha\mathbf{x}_S$
recovery	(1,-1)	$\beta\mathbf{x}_I$
infection from a meeting	(-1,1)	$\gamma\mathbf{x}_S\mathbf{x}_I$

Within our tool, we define a class called `DDPP` that can be used to define DDPPs directly from their mathematical definition. For the above SIS example, we would write:

```
import rmf_tool as rmf
model = rmf.DDPP()
alpha, beta, gamma = 1,1,1
model.add_transition([−1,1], lambda x: alpha*x[0])
model.add_transition([1,−1], lambda x: beta*x[1])
model.add_transition([−1,1], lambda x: gamma*x[0]*x[1])
```

[1]The notation $\mathbf{e}_s \in \{0,1\}^d$ correspond to a vector of size d whose s entry is equal to 1, all others being 0.

As for the HomPP, the model can then be used to simulate the stochastic process, to compute the mean field approximation and the refinements. The syntax is identical. If one wants to run a simulation with a population of $n = 1000$ where at the beginning all individuals are in the first state (susceptible), one would write:

```
model.set_initial_state([1,0])
t, X = model.simulate(N=1000, time=2)
```

2.3 Heterogeneous population process

In [1], the authors extend the notion of the HomPPs to deal with populations of heterogeneous objects. As before, the heterogeneous population model consists of n interacting objects which each evolve in a finite state space $\{1 \dots d\}$. Each object has a specific transition rate which is a combination of unilateral or pairwise interactions. In contrast to the HomPP, transition rates are object dependent:

- The object k moves from state s to state s' at rate $a_{k,s,s'}$.
- The pair (k, k') moves from states $(s, \tilde{s})$ to states $(s', \tilde{s}')$ at rate $b_{k,k',s,\tilde{s},s',\tilde{s}'}/n$.

Note that the difference between a homogeneous population process and a heterogeneous population process is that the rate tensors a and b depend on the object id k. As a result, the process $\mathbf{X} = (X_1 \dots X_d)$ where $X_s(t)$ is the fraction of objects in state s is not a Markov process. Let the stochastic process $\mathbf{Z} \in \{0,1\}^{n\times d}$ describe the evolution of the population where $Z_{k,s} = 1$ indicates that object k is in state s and $Z_{k,s} = 0$ if it is not. The process $\mathbf{Z}$ is a Markov process whose transitions are:

$$\mathbf{z} \mapsto \mathbf{z} - \mathbf{e}_{k,s} + \mathbf{e}_{k,s'} \qquad \text{at rate } a_{k,s,s'} z_{k,s}$$

$$\mathbf{z} \mapsto \mathbf{z} - \mathbf{e}_{k,s} + \mathbf{e}_{k,s'} - \mathbf{e}_{\tilde{k},\tilde{s}} + \mathbf{e}_{\tilde{k},\tilde{s}'} \quad \text{at rate } \frac{1}{n} b_{k,\tilde{k},s,\tilde{s},s',\tilde{s}'} z_{k,s} z_{\tilde{k},\tilde{s}}.$$

These transitions generalize (1)-(2).

Example: Heterogeneous SIS model. To set up a heterogeneous version of the previous SIS model we use the HetPP class of the toolbox. In contrast to the HomPP and DDPP class, the model can not be defined independent of the system size, *i.e.*, n and d have to be defined to initialize the model. For instance, to set up a SIS model where objects are almost identical but some recover slower than others, we can use the code:

```
import rmf_tool.src.heterogeneous_rmf_tool as hrmf
model = hrmf.HetPP()
N, d = 20, 2
S, I = 0, 1
A, B = np.zeros((N, d, d)), np.zeros((N, N, d, d, d, d))
A[:, S, I] = np.ones((N))
A[:, I, S] = np.random.rand(N)   # Hetero. recovery rates
B[:, :, S, I, I, I] = (1/N) * np.ones((N, N))
model.add_rate_tensors(A, B)
```

Here, the tensor A and B specify the transition rates where A[k,s,s'] $= a_{k,s,s'}$ and B[k,$\tilde{k}$,s,$\tilde{s}$,s',$\tilde{s}$'] $= \frac{1}{n} b_{k,\tilde{k},s,\tilde{s},s',\tilde{s}'}$. The corresponding transition vectors of the model are derived from the non zero rates of the tensors. The methods of the HetPP class are coherent to the HomPP and DDPP class.

3. MEAN FIELD APPROXIMATIONS AND REFINEMENTS

3.1 Mean field approximation (homogeneous)

For a given DDPP, and a given state $\mathbf{x} \in \mathbb{R}^d$, we define the drift in state $\mathbf{x}$ as

$$f(\mathbf{x}) = \sum_{\ell \in \mathcal{L}} \ell \beta_\ell(\mathbf{x}).$$

The drift corresponds to the average variation of the model, as it is the sum of state changes (ℓ) weighted by the rate at which these changes occur.

For a given initial condition $\mathbf{x}(0)$, the mean field approximation of a DDPP is the solution of the ODE:

$$\dot{\mathbf{x}} = f(\mathbf{x}).$$

The same holds true for any HomPP since the class of DDPP is essentially a scaled generalization of the former. Thus, all methods which are available for DDPPs are available for HomPP as well. Within our tool, the mean field approximation can be easily computed with:

```
t, X = model.ode(time=2)
```

It is known from [11] that under very general conditions (essentially that f is Lipschitz-continuous), the stochastic trajectories of $\mathbf{X}$ converge to the mean field approximation $\mathbf{x}$ as the scaling parameter n goes to infinity. We illustrate the accuracy of the mean field approximation in Figure 1, where we compare two stochastic trajectories of the system for populations of $n = 100$ and $n = 1000$ individuals, with the mean field approximation.

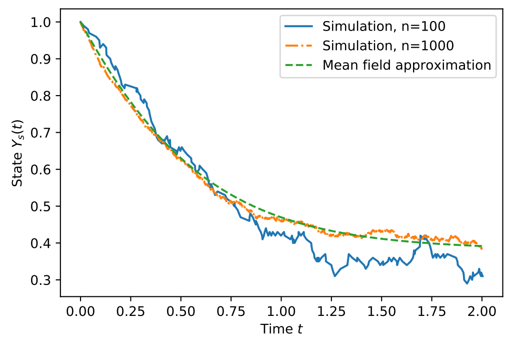


Figure 1: Example: Simulation of the SIS (DDPP model)

3.2 The refined mean field approximation

It is shown in [6, 7] that when the drift of the DDPP is twice differentiable, there exists a time varying vector $\mathbf{v}$ and a time varying matrix $\mathbf{w}$ such that:

$$\mathbb{E}[\mathbf{X}(t)] = \mathbf{x}(t) + \frac{1}{n}\mathbf{v}(t) + O(\frac{1}{n^2});$$

$$\text{Var}[\mathbf{X}(t)] = \frac{1}{n}\mathbf{w}(t) + O(\frac{1}{n^2}),$$

where Var $[\mathbf{X}(t)]$ is the covariance matrix of the stochastic process $\mathbf{X}$.

The above equation holds for any finite time. It is shown in [7] that, for the transient regime, $\mathbf{v}$ and $\mathbf{w}$ satisfy a time-inhomogeneous linear ODEs. If there exists a point $\mathbf{x}(\infty)$

such that for all initial condition $\mathbf{x}(0) \in \mathbb{R}^d$, the solution of the ODE converges to $\mathbf{x}(\infty)$ exponentially fast, then this equation holds uniformly in time and in particular is also true for the steady-state $t = +\infty$. In the latter case, the following linear equation (that is called a Lyapunov equation) is satisfied:

$$\mathbf{w}J + J^T\mathbf{w} + \mathbf{q} = 0, \tag{3}$$

where J is the Jacobian of the drift f evaluated at $\mathbf{x}(\infty)$ and $\mathbf{q} = \sum_\ell \ell \otimes \ell \beta_\ell(\mathbf{x}(\infty))$. The vector $\mathbf{v}(\infty) = J^{-1}(D \cdot \mathbf{w})$, where D is the second derivative of f evaluated at $\mathbf{x}(\infty)$ and $\cdot$ denotes a tensor product: $(D \cdot \mathbf{w})_i = \sum_{jk} D_{i,jk} w_{jk}$ and $D_{i,jk} = (\partial^2 f / \partial x_j \partial x_k)$ evaluated in $x(\infty)$.

This means that they can be easily solved numerically. The tool provides methods to automatically compute these constants for the transient or the steady-state regime. These functions rely on `scipy`'s functions: for the transient regime it uses the `solve_ivp` from `scipy` and for the steady-state the function `solve_continuous_lyapunov`. An example of the tool is:

```
t, x, v, _ =\
        model.meanFieldExpansionTransient(order=1,time=2)
x_inf, v_inf,_ =\
        model.meanFieldExpansionSteadyState(order=1)
x_simu, _ = ddpp.steady_state_simulation(N=n, time=20000)
```

where the last line estimates $\mathbb{E}[\mathbf{X}(\infty)]$ by simulating a trajectory of 20000 events and computes the average over the end of the trajectory.

This result is illustrated in Table 1, where we compare the mean field approximation, the refined mean field approximation and an estimation of the steady-state probability $\mathbb{E}[X_s(\infty)]$ computed by simulation. We observe that if the mean field approximation is already very accurate, its refined version is close to being exact.

n	M-f $\mathbf{x}(\infty)$	Refined $\mathbf{x}(\infty) + \frac{1}{n}\mathbf{x}(\infty)$	Simulation
10	0.382	0.394	0.394 ± 0.004
20	0.382	0.388	0.389 ± 0.003
30	0.382	0.386	0.386 ± 0.002

Table 1: SIS model: Illustration of the accuracy of the mean field and refined mean field approximations for steady-state.

Note that the tool also allows to compute the second order refinement term as defined in [7]. This can be done by changing the `order=1` into `order=2` in the code. The time to compute this approximation is much larger than the time to compute the refined mean field approximation (that corresponds to a first order expansion).

3.3 Heterogeneous mean field approximation and refinements

The heterogeneous mean field approximation and its refinement differs from the homogeneous case in the sense that transitions are dependent on the state of single objects. For the stochastic process this is taking into account by considering an object dependent representation. The intuition of the mean field approximation is as before, for the drift we consider the sum over all transitions weighted by their transition rate. Let the drift in state z be denoted by $f^{het}(z)$, then, the mean field approximation is again the solution to the ode having f^{het} as drift with initial condition $z(0)$. If both, $a_{k,s,s'}$ and $b_{k,\tilde{k},s,\tilde{s},s',\tilde{s}'}$ are uniformly bounded, it holds, as shown in [1] that the adapted mean field and refined mean field approximation capture the probability of the objects to be in a state with an accuracy of $O(1/n)$ and $O(1/n^2)$, i.e.

$$\mathbb{E}[Z_{k,s}(t)] = \mathbb{P}(Z_{k,s}(t) = 1) = z_{k,s}(t) + O(1/n), \tag{4}$$
$$\mathbb{E}[Z_{k,s}(t)] = \mathbb{P}(Z_{k,s}(t) = 1) = z_{k,s}(t) + v_{k,s}(t) + O(1/n^2).$$

The term $v_{k,s}(t)$ refers to the adapted refinement term whose precise definition can be found in [1, Appendix B].

Simulations of stochastic trajectories, mean field and the refinement methods can be calculated by calling the same functions as for the homogeneous case. Note that second order refinement methods are note available for the current version since they are computationally too expensive.

Due to the setup of the heterogeneous population process, single simulation trajectories are not close to the mean field approximation but close to the sample mean of the stochastic system, that is, (4) holds but $Z_{k,s}(t)$ does not converge to $z_{k,s}(t)$ as n goes to infinity (contrary to what appends to the DDPP case for which one can show [11] that $X_s(t)$ converges in probability to its mean field approximation $x_s(t)$, which is what is observed in Figure 1). Hence, to study the accuracy of the mean field and refined mean field approximation in the heterogeneous context, we provide the additional methods **`sampleMean`**, **`sampleMeanVariance`** with which the sample mean and sample variance can be calculated. To calculate an approximated mean with 100 samples, we set the initial state to have only susceptible objects and write:

```
model.set_initial_state(np.ones((N,d))*np.array([1,0]))
t_mean, mean, var =\
        model.sampleMeanVariance(time=2, samples=100)
```

In order to compare the results to a one of the homogeneous models one should consider the sum $Y_s(t) = \frac{1}{n}\sum_{k=1}^n Z_{(k,s)}(t)$, which is a density representation of the heterogeneous population process. It can be shown that $Y_s(t)$ converges in probability to its mean field approximation $y_s(t)$, as the number of objects grows.

4. IMPLEMENTATION CHALLENGES

Most of the toolbox functionality is a direct implementation of the equations defined in [10, 7], with the use of functions from `numpy` or `scipy` to integrate differential equations or solve linear equations. Yet, there are some implementation challenges among which we list two here: how to automatically compute the drift's derivatives (Section 4.1), and how to deal with model that do not satisfy the exact assumptions of [10] needed for the steady-state (Section 4.2).

4.1 Automatic differentiation

To compute the refined mean field approximation, one needs to compute the first and second derivatives of the drift function f. We implement three different methods. The first is to use a finite difference method. $\partial f_i(\mathbf{x})/\partial x_j \approx (f_i(\mathbf{x} + \varepsilon \mathbf{e}_j) - f_i(\mathbf{x}))/\varepsilon$. This is the most robust method but is relatively slow and has a limited precision due to the choice of ε. The second method that we implement is to use the package `simpy` that allows for symbolic computation and can be used to compute derivatives. The third method is a method based on `autograd` from `jax` that uses automatic differentiation. These two methods are both faster and more accurate than finite difference methods. Yet, they cannot differentiate all functions. For instance, if the drift involves a sinus function and if the DDPP model is defined using the `numpy.sin` function, then the `simpy` will not be able to differentiate this function as it does not understand the `numpy` function. Here, `autograd` will work.

4.2 Dimension reduction

In order for the equation (3) to have a unique solution, the assumption used in [10] is that all solutions of the mean field ODE $\dot{\mathbf{x}} = f(\mathbf{x})$ converge to the same fixed point $\mathbf{x}(\infty)$, regardless of the initial condition $\mathbf{x}(0) \in \mathbb{R}^d$. Yet, in practice, many models are naturally described as d-dimensional DDPP but evolve in a smaller dimensional space $\mathcal{X} \subset \mathbb{R}^d$. This is for instance the case for the SIS model of Section 2 that evolves in a space of dimension 1 because $x_S + x_I = 1$. It further implies that the Lyapunov equation (3) does not have a unique solution. As such, one cannot apply directly the theorem of [10] to this SIS model.

A mathematical solution to this is to redefine our SIS model to obtain a model in dimension 1. By replacing the occurrences of x_I by $1 - x_S$ in all equations. Yet, if this is easily done for reducing a 2D model to a 1D model, it can be cumbersome when going from a 20D to a 15D model. Our tool allows doing this automatically. This is how we can obtain Table 1 while using the DDPP defined in Section 2.

Our approach to this problem is to compute the rank of the set of transitions $\mathcal{L}$. If this rank is $d' < d$, this means that the model evolves in a d'-dimensional state space. In particular, the jacobian A used in Equation (3) has dimension at most d'. Our code uses the SVD decomposition of A to transform the d-dimensional Lyapunov equation (3) into d'-dimensional equation. This is particularly useful for heterogeneous models composed of n objects that each evolve in a S dimensional state: a natural description of the model is to construct a nS-dimensional DDPP, but that evolves in a subset of dimension $n(S-1)$ or even smaller. For instance, the cache replacement policy studied in [1, 2, 8] with n objects and S lists is naturally described as a $n(S+1)$ process but evolves in fact in a $nS - S$ state space. Using the automated dimension reduction greatly simplifies the definition of the model in the tool.

5. CONCLUSION AND DISCUSSION

In this paper, we present a tool, called *rmf_tool*, that can be used to define and study mean field interaction models. The tool is build-in with a stochastic simulator, and methods to compute the mean field approximation and refined approximation of a given model. The tool consists on a Python library and models can be directly be defined as python objects. In the present paper, we illustrate how the tool can be used by using a simple SIS model. Below, we discuss in more detail the applicability of the tool by giving a few examples of application, and by analyzing the computation time.

5.1 To which model does this apply?

The tool is provided with a number of examples that demonstrates the use of the tool and the accuracy of the approximation. These examples include:

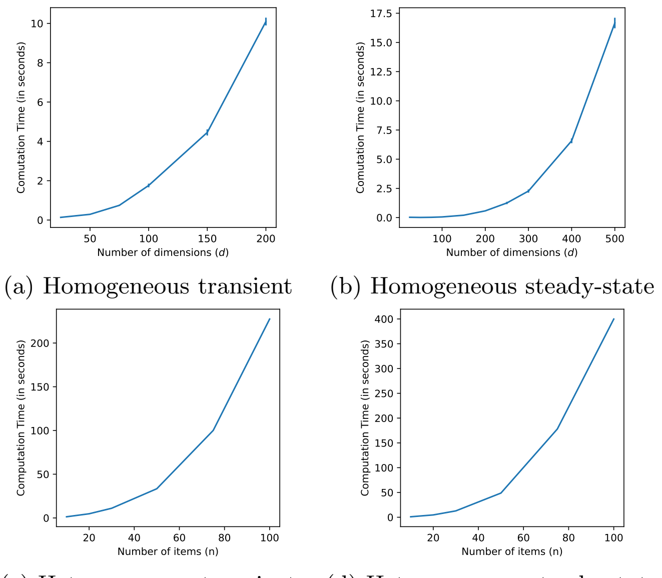
- The power of two choice model of [13]. This example models a simple, yet powerful, load balancing strategy in a system composed of n servers.
- The bike-sharing model of [4]. It models a city where Cn bikes moves in a city composed of n stations.
- A epidemic model called the SIR model (that is a generalization of the SIS model presented in the paper).

Although they are not directly provided as examples in the repository, the tool is also used in [1, 2] to analyze the performance of cache replacement policies. These cache replacement policies are examples of non-homogeneous population models.

5.2 Analysis of the computation time

To give an idea of the time needed to compute the refined approximation, we report in Figure 2 the time taken by the tool to compute the refined mean field approximation as a function of the system size. The first line corresponds to a homogeneous model of dimension d: in this model, we consider the power of two choice model of [13] where we bound the queue length by d. We report the numbers of [7]. We observe that for this model, we can solve the problem for a few hundreds of dimension in less than a few tens of seconds. Note that for this example, the jacobian and the second derivative can be computed in close form. Hence, the reported time does not include the time that would be taken if one were to use symbolic differentiation.

In the second line of Figure 2, we report the time taken to solve the heterogeneous SIS model defined in Section 2.3 with n different objects. For this example, we use the `HetPP` class. Note that this class does not use symbolic differentiation since the derivative can be directly computed by using the A and B tensors. The model here is a $2n$ dimensional model. We observe that the time taken here for a model with $2n$ dimension is larger than the time for a homogeneous model of dimension $2n$. We believe that the run time could be improved by using sparse tensor multiplications and will consider this question for future work.



(a) Homogeneous transient (b) Homogeneous steady-state

(c) Heterogeneous transient (d) Heterogeneous steady-state

Figure 2: Analysis of the computation time: for the transient regime, we compute $\mathbf{v}(t)$ for $t \in [0, 10]$. For the steady-state, we compute $\mathbf{v}(\infty)$.

Acknowledgments

This work is supported by the French National Research Agency (ANR) through REFINO Project under Grant ANR-19-CE23-0015.

6. REFERENCES

[1] Sebastian Allmeier and Nicolas Gast. Mean field and refined mean field approximations for heterogeneous systems: It works!, 2021.

[2] Giuliano Casale and Nicolas Gast. Performance analysis methods for list-based caches with non-uniform access. *IEEE/ACM Transactions on Networking*, 2020.

[3] Guilherme Ferraz de Arruda, Francisco A. Rodrigues, and Yamir Moreno. Fundamentals of spreading processes in single and multilayer complex networks. *Physics Reports*, 756:1–59, October 2018.

[4] Christine Fricker and Nicolas Gast. Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity. *Euro journal on transportation and logistics*, 5(3):261–291, 2016.

[5] Nicolas Gast. Expected values estimated via mean-field approximation are 1/n-accurate. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):17, 2017.

[6] Nicolas Gast. Refined Mean Field Tool. 2018.

[7] Nicolas Gast, Luca Bortolussi, and Mirco Tribastone. Size expansions of mean field approximation: Transient and steady-state analysis. *Performance Evaluation*, 129:60–80, February 2019.

[8] Nicolas Gast and Benny Van Houdt. 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 - SIGMETRICS '15*, pages 123–136, Portland, Oregon, USA, 2015. ACM Press.

[9] Nicolas Gast and Benny Van Houdt. A Refined Mean Field Approximation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):33:1–33:28, December 2017.

[10] Nicolas Gast and Benny Van Houdt. A refined mean field approximation. *Proc. ACM Meas. Anal. Comput. Syst*, 1, 2017.

[11] Thomas G Kurtz. Strong approximation theorems for density dependent markov chains. *Stochastic Processes and Their Applications*, 6(3):223–240, 1978.

[12] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, Oct./2001.

[13] Michael David Mitzenmacher. *The Power of Two Random Choices in Randomized Load Balancing*. PhD thesis, PhD thesis, Graduate Division of the University of California at Berkley, 1996.

[14] Antonio Montalbán, Rodrigo M. Corder, and M. Gabriela M. Gomes. Herd immunity under individual variation and reinfection. *arXiv:2008.00098*, November 2020.

[15] Arpan Mukhopadhyay and Ravi R. Mazumdar. Analysis of Load Balancing in Large Heterogeneous Processor Sharing Systems. *arXiv:1311.5806*, February 2015.

[16] Philipp Thomas, Hannes Matuschek, and Ramon Grima. Intrinsic noise analyzer: a software package for the exploration of stochastic biochemical kinetics using the system size expansion. *PloS one*, 7(6):e38518, 2012.

[17] Benny Van Houdt. A mean field model for a class of garbage collection algorithms in flash-based solid state drives. *ACM SIGMETRICS Performance Evaluation Review*, 41(1):191–202, June 2013.