Network Modeling and Simulation

Ilaria Cherchi
Telegram: @ilariacherchi
Giacomo Fantoni
Telegram: @GiacomoFantoni
Elisa Pettinà
Telegram: @elispettina
Github: https://github.com/giacThePhantom/network-modelling-and-simulation

February 9, 2023

Contents

Part I

Stochastic models

Chapter 1

Introduction

1.1 Introduction

1.1.1 Cells

A cell is the basic unit of all known living organism. Different cell types have different roles which then organize to form higher levels of organization. The cell is a dynamical system whose behaviour is controlled and regulated by interaction between chemical species.

1.1.2 Biochemical reactions

The interactions between molecules in a cell are known as biochemical reactions. The chemical species in a cells are in constant movement and when they collide can cause a reaction if specific reaction conditions, like the activation energy, are satisfied. The outcome of a reaction is the consumption of some species and the production of new ones to help perform the necessary activities of the cell.

1.1.2.1 Reaction kinetics

The rate of a reaction is dependent on the species involved, the number of molecules present and a basal rate or affinity. The basal rate depends on the type and number of species involved and is often constant. The rates of reaction are determined by the reaction kinetics.

1.1.2.2 Pathways

Biochemical reactions are organized in pathways, a map showing the structural relationship of molecular species and their specific cellular response. They are involved in:

- Metabolism.
- Signal transmission.
- Gene expression regulation.

They are involved in different cellular purposes like:

- Cell growth.
- Proliferation.
- Differentiation.
- Apoptosis.

Explaining how a cellular function emerges from the molecular interactions needs a system-wide approach.

1.1.3 Systems biology

Systems biology is a new discipline that aims to understand how reactions give rise to specific cellular behaviours and a biological response. Its holistic view-point provide advantages in scientific and practical terms like:

- Drug discovery.
- Disease mechanism explanation.
- Hypothesis verification.

1.1.3.1 Challenges of system biology

The challenges of systems biology are due to the large number of possible reactions and their non-linear dynamics. In these cases the stationary and time-invariant assumptions are often violated: the species constantly evolves according to changes in the cellular environments. Moreover some molecular species are present in low copy number or population. Reaction between these cause a significant fluctuation in their population, or biological noise, which may propagate along the pathway.

1.1.3.2 Stochasticity

The stochasticity in biochemical reaction can be also due to the fact that after many non-reactive collisions between species the biological system could choose a different cellular functioning. This is called multistability: a number of separated stable equilibria points are separated by unstable equilibria. Bistability is the simplest example of multistability, with only two stable equilibria.

1.1.4 Computational tools

Computational tools play a crucial role in systems biology. Introducing a model to represent the biological system's species of interest or states and the reaction between them or state transition, knowledge of the biology can be written in a formal form, often mathematically.

1.1.4.1 Biological models

A biological model is an abstraction of the system, but it is useful to understand it. A direct way to describe a model is to write down the list of reactions between species. Modelling a reaction network by coupled reactions is simple and flexible.

1.1.4.2 Computer simulation

Given a model, a computer simulation is used to realize its temporal evolution. The dynamical interactions between species can reveal indirect implications or unexpected behaviours. A simulation based experiment is called an in silico experiment. If results of this experiment agree with experimental data, they can be used to provide predictions for the dynamics of the system.

1.1.4.3 Advantages and applications of in silico experiment

With respect to traditional experiment, an in silico simulation has a number of advantages:

- It takes less time.
- It is cheaper.
- They can detect indirect and hidden implications.
- It is possible to isolate vital genes from the cell, overcoming the necessity of maintaining a vital cell.

The results of an in silico experiment are used to:

- Test hypothesis.
- Suggesting new experiments.

The predictive feature of computer simulations makes it useful to perform quantitative analysis.

1.1.5 Synthetic biology

Biological models and simulation contribute to the design and implementation of synthetic biology by providing a design focused experiment framework where models are reused as basic building blocks in a large model. This component-based model building is more effective and less error-prone than a traditional from-scratch approach. Moreover this component approach allows to reprogram cellular functions to serve for special purposes of biological research.

1.2 Different levels of simulation

To cope with the inherent multi-physics and multi-scale natures of biochemical reactions, different levels of simulation detail have been adopted to investigate their dynamical behaviour.

1.2.1 Molecular dynamics

Molecular dynamics deals with the microscopic level. In this type of simulation the structures, positions, velocities and possible collisions of all molecules in the system are kept track of. Movement and reactions are governed by physical forces. A molecular dynamics simulation requires a very detailed knowledge of molecules in both time and space and requires a lot of computational power. Because of this it is used to simulate the system at the level of nanoscale of time and space.

1.2.2 Brownian dynamics

Brownian dynamics focuses on the dynamics of each individual species, skipping the molecular structure information and the weak long-range forces between species. The movement of a species is described as a random or Brownian walk among point-like structures. A reaction happens whenever the distance between two species is less than a predefined reaction radius. The time scale of a Brownian dynamics simulation is greatly improved over molecular dynamics, but it is still require too much computational power to deal with large scale models.

1.2.3 Deterministic simulation

A deterministic simulation is the highest coarse-grained approach which focuses on the macroscopic behaviour of biochemical reactions. Molecular species in the deterministic simulation approach are represented by their concentrations. The rate of change in the concentration of each species due to a reaction is directly proportional to the concentrations of the species involved in the reaction. The time evolution of a biochemical reaction network is described by a set of ordinary differential equations or ODEs. The deterministic simulation is fast, however its underlying assumption inherently oversimplifies biological reactions in which populations of molecular species are continuous variables and their changes due to single reaction firings are assumed to be negligible. The correctness of deterministic simulation is severely affected when stochasticity plays an important role in the dynamical behaviour of biochemical reactions.

1.2.4 Stochastic simulation

A stochastic simulation is a mesoscopic approach to provide a probabilistic description of the time evolution of biochemical reactions. It keeps track of a discrete count for the population, but abstracts all the detailed position and velocity information of each species. Each reaction in the network is assigned a non-negative chance to fire and to drive the system to a new state. The probability that a reaction occurs in a time interval is derived from the reaction kinetics. Each stochastic simulation step will select a reaction to fire according to its probability.

1.2.5 Problems with stochastic simulations

Although the stochastic simulation approach is faster than both a molecular dynamics or Brownian dynamics approach, it is often computational demanding for simulating large biological systems. Moreover:

- Biochemical reactions, due to their multi-scale nature, are separated by different time scales in which some fast reactions will occur at rates greater than other reactions. The dynamical behaviour of biochemical reactions, after the short fluctuation time at the beginning, will be determined by the dynamics of the slow reactions. However, most of the time the simulation samples the fast reactions to realize the dynamics which is not the expected behaviour.
- The population of some species involved in reactions may be larger than others by many orders of magnitude. This implies that the fluctuations of some species are more or less significant when involving reactions fire.
- Keeping track of large population species is less efficient, since a coarse-grained simulation method can be applied without loss of total simulation accuracy. A model can combine and mix all of these aspects making the fast reactions occur frequently and drive the system very fast into a stable state.
- Due to the stochastic behaviour in a single simulation, many simulation runs must be performed to ensure a statistical accuracy and this requires a high computational effort. These issues raise a computational challenge for developing and implementing efficient stochastic simulation methods.

1.3 Network modelling

1.3.1 Logic models

A logical model $(\mathcal{G}, \mathcal{K}$ of a regulatory network is defined by:

- A set of n regulatory components $\mathcal{G} = \{g_1, \ldots, g_n\}$, with each g_i associated with an integer value bounded by $\{0, \ldots, \max_i\}$. This defines a discrete mapping of the range of the component functional level. The state space $\mathcal{S}$ is the Cartesian product $\prod_{i=1}^{n} \{0, \max_i\}$. The model state is a vector $g = (g_1, \ldots, g_n)$.
- For each g_i a discrete function $\mathcal{K}_i$ defines its values depending on the state: $\mathcal{K}_i : \mathcal{S} \rightarrow \{0, \ldots, \max_i\}$. The transition function $\mathcal{K} : \mathcal{S} \rightarrow \mathcal{S}$ defines the model behaviour and the underlying regulatory graph as $\mathcal{K}(g) = (\mathcal{K}_1(g), \ldots, \mathcal{K}_n(g))$. In some cases a regulatory component may operate at different levels on distinct targets and requires multiple values whose maximum is greater than 1. The discrete functions are still logical functions.

1.3.1.1 Regulatory graph

The regulatory graph $(\mathcal{G}, \mathcal{R})$ encompasses nodes denoting model components in $\mathcal{G}$ and signed, directed edges denoting regulatory activations or inhibitions in $\mathcal{R}$. The logical rules encode these interactions: $(\mathcal{G}, \mathcal{R})$ can be deduced from $\mathcal{K}$. Several sets of logical rules can be compliant with a regulatory graph, which therefore defines a family of logical models.

1.3.1.2 Functional interaction

A functional interaction from g_j to g_i, or $(g_j, g_i) \in \mathcal{R}$ if and only if there exists a pair of neighbouring states that only differ on the value of g_j and for which $\mathcal{K}_i$ takes a different value.: a variation of g_j has an effect on g_i. Formally, assume that g_j is a boolean variable, $(g_i, g_j) \in \mathcal{R}$ if an only if there exist:

$$g = (g_1, \ldots, g_{j-1}, 1, g_{j+1}, \ldots, g_n)$$

$$\bar{g} = (g_1, \ldots, g_{j-1}, 0, g_{j+1}, \ldots, g_n)$$

Such that $\mathcal{K}_i(g) \neq \mathcal{K}_i(\bar{g})$. If $\mathcal{K}_i(g) < \mathcal{K}_i(\bar{g})$ this interaction is an activation, otherwise an inhibition.

1.3.1.3 Model dynamics

Given a stage g, $\mathcal{K}$ specifies the possible changes on the model variables: if $\mathcal{K}(g) \neq g$, there is at least one g_i called to update toward the target value $\mathcal{K}_i(g)$. Multi-valued variables are modified stepwise: the next value of g_i is changed by 1 for each step. If $\mathcal{K}(g) = g$ g is a stable state and each component value is maintained constant. Input components have no regulators and considered constant.

1.3.1.4 State transition graphs

State transition graphs or STG represents model dynamics. Nodes denote states, while directed edges state transitions. Since the number of states is finite, model simulation ends up in a stable state or a cyclic trajectory. These are called attractors and are defined as terminal strongly connected components SCC of the STG, the maximal sets of mutually exclusive reachable states, with no transitions leaving the set. The set of states from which trajectories lead exclusively to an attractor is called its strict basin of attraction. They are relevant since they define the reachable attractors depending on the initial state.

1.3.1.5 State updates

If at state g several variables are called to change their values, there is a need to specify how the changes should be performed.

1.3.1.5.1 Synchronous update According to the synchronous method, all the variable updates are performed synchronously. The resulting deterministic dynamics defines at each time step t the successor state of $g(t)$:

$$g(t+1) = (g_i(t) + sign(\mathcal{K}_i(g(t)) - g_i(t)))_{i=1,\dots,n}$$

Where:

$$sign(p) = \begin{cases} 1 & p > 0 \\ -1 & p < 0 \\ 0 & otherwise \end{cases}$$

A successor state is defined by increasing or decreasing 1 all the variables whose current value differ form the values specified by their logical function. This yields exactly one transition toward a successor state, which can be g itself.

1.3.1.5.2 Asynchronous update According to the asynchronous update each variable is modified independently, yielding as many transitions and successor states as the number of updated variables. At state $g(t)$, for all $g_i \in \mathcal{G}$ such that $\mathcal{K}_i(g(t)) \neq g_i(t)$, an asynchronous successor is defined as:

$$g_i(t+1) = g_i(t) + sign(\mathcal{K}_i(g(t)) - g_i(t))$$

$$g_j(t+1) = g_j(t) \qquad \forall j \neq i$$

According to this definition a stable state has no successors.

1.3.2 Petri nets

Petri nets are specific networks introduced in 1960s, with the idea to describe communication processes in the computer science field. In Petri nets two types of nodes can be found:

- Places: container of entities, which are represented as tokens.
- Transitions: nodes that describe the possibility to move one or more tokens to other places.

In order to model a chemical reaction, places can be associated to variables, tokens to molecules and transitions to chemical transformations.

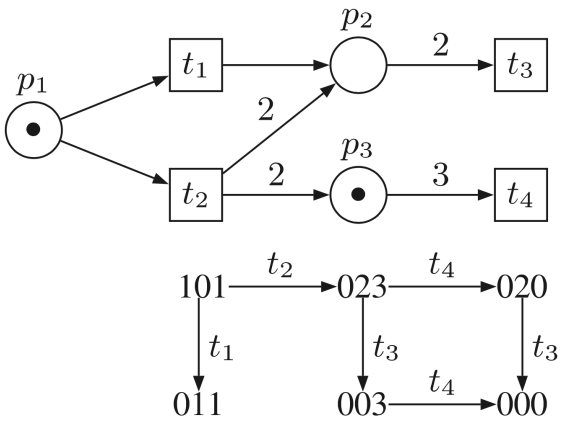

Figure 1.1: An example of a Petri Net

Figure 1.1 is an example of a Petri net. The network will evolve according to the transitions applied. A transition can be enabled or not to fire. For example there is one token in p_1, so t_1 and t_2 are enabled. Instead, t_4 cannot be enabled, as 3 tokens are required but only one is present. t_2 is taking the token from p_1 and creating 2 tokens in p_2 and p_3, allowing to fire t_4, since now p_3 has the required number of tokens. Moreover, looking at figure 1.2:

1. Panel (a) shows the initial marking before firing the enable transition t.
2. Panel (b) shows the marking after transition labelled reaction 1 fires. Places: hydrogen, oxygen and water. The stoichiometry of the reaction can be represented through the numbers on the edges and the numbers on the tokens.

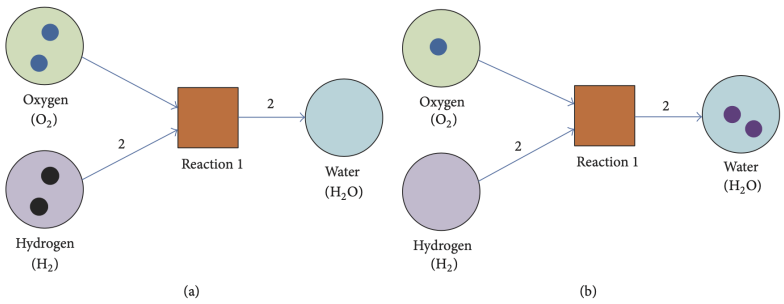

Figure 1.2: Transition in a Petri net

1.3.2.1 Advantages and disadvantages

In Petri nets there is no constraint on the data type, allowing to extend the number of items which can be associated to a model. However there is still no fixed rule to apply transition and reaction's complexity is still not being considered.

1.3.3 Discussion

Having a dynamics based on the integers can be quite useful, as if considering single chemical events the data is discrete. This is the same case as the exact stochastic algorithms. In differential equations real numbers are needed. Also the discretization of the time step might not be a limitation, as time can be discretized in reality. The main limitation of network models is the approximation of time: all the reactions take the same amount of time to happen.

1.3.4 Rewriting systems

By upgrading the notation more accurate representations can be achieved:

- Add inhibitory and test transitions.
- Differentiate between discrete and continuous transitions.

1.3.4.1 P systems

Popular rewriting systems are P systems, which are also called membrane systems. They are computational environments inspired to the structure of membranes. In particular, they define a hierarchy of membranes partitioning the space in different areas, similarly to a cell. In each region entities can be assigned and transformation rules applied. The rewriting rules change the value of each letter. These kind of systems tend to use non-determinism: they try to explore the full set of possibilities.

1.3.4.2 MP systems

The difference with standard P system is the association of functions to each reaction. In this way the complexity of the reaction can be reconstructed, since in the model all possible reaction are applied, which will produce an amount given by the function.

1.3.5 Equation-based approach

Equation-based approaches represent the system as a set of ordinary differential equations ODE. For example the mass action model:

$$A + E \underset{k_2}{\overset{k_1}{\rightleftharpoons}} A \mid E \xrightarrow{k_3} B + E$$

1.3.6 Simulation algorithms

For a simulation to be computed the necessary data are:

- The stoichiometric matrix.
- A vector of initial values.
- A stochastic rate.

1.3.6.1 Exact simulation algorithms

Exact simulation algorithms are computationally intensive, but provide the most accurate solution. For each step they try to define the faster strategies with the aim of compromising between accurate dynamics and a feasible solution. It is also possible to rely on a mixture of technologies to focus on different results. If the well-mixed assumption is not fulfilled, the compartments are partitioned into sub-compartments leading to an approximation or more sophisticated algorithms are needed.

1.3.6.2 The stoichiometric matrix

The stoichiometric matrix tells how the system evolves if one of the two functions is applied, but it is not enough for computing a simulation. There can be many reactions that arrive to the same definition of stoichiometric matrix. To compute the dynamics a series of states needs to be developed, requiring:

- τ: tells how much later the system will evolve to another state.
- μ: choose the reaction by considering the probability of execution of each reaction at the step.

1.3.6.3 Reaction propensity

The reaction propensity is a function needed for the derivation of probability. The higher the propensity, the higher will be the strength of the reaction. Naturally, a higher probability is found when a higher propensity is observed, but the two quantities are something different. The propensity is a property of the reaction, while the probability is a property of a reaction in the system: it takes into account other reactions. Instead of performing an in depth analysis of probability, a stochastic approach is undertaken. It is only necessary to compute one evolution of the system per time.

Chapter 2

Stochastic simulation of biochemical reaction systems

2.1 Introduction

Biochemical reaction network can be modelled through stochastic chemical kinetics. In this type of models the discreteness in population of species and the randomness of reactions are treated as an intrinsic part. The dynamical behaviour is described by the chemical master equation or CME.

2.2 Stochastic chemical kinetics

2.2.1 Rewriting biochemical reactions

Biochemical reactions are the building blocks to model biological systems. They provide a unifying notation with sufficient level of details to represent complex biological processes. Biochemical reactions decorated with reaction kinetics can be simulated by a simulation algorithm to generate a realization of their dynamics. Chemical species in a biological system move around and gain kinetic energy. Upon collisions with other species, they undergo reactions to modify and transform into different species.

2.2.1.1 Representing molecular reactions

Molecular reactions can be classified between elementary reactions and non-elementary ones. The former require one step to complete, while the latter require a higher order reactions or a multi-step one.

2.2.1.1.1 Some elementary reactions

2.2.1.1.1.1 Unimolecolar reaction An unimolecolar reaction involves the transformation of one species into another. It can be represented as:

$$A \to B$$

2.2.1.1.1.2 Degradation The degradation of a species is a special type of an unimolecolar reaction in which a species is degradated:

$$A \rightarrow \emptyset$$

Where the symbol $\emptyset$ represents a species that is not considered in the model, probably because it is large and does not change over time.

2.2.1.1.1.3 Synthesis The synthesis of a species is another type of unimolecolar reaction in which a species is synthesised:

$$\emptyset \rightarrow A$$

This reaction is used to model the effects of the outside environment on system dynamics.

2.2.1.1.1.4 Bimolecular reaction An example of a bimolecular reaction is an association reaction, in which a molecule can associated with another to produce a complex:

$$A + B \rightarrow C$$

Often this process is reversible and the inverse dissociation reaction can take place:

$$A + B \leftrightharpoons C$$

2.2.1.1.1.5 Dimerization The dimerization is a special bimolecular reaction in which two molecules of the same species are consumed to produce another species:

$$2A \rightarrow B$$

2.2.1.1.2 Some non-elementary reactions

2.2.1.1.2.1 Termolecular reaction The termolecular reaction is used to represent the polymerization of three molecules into another species. The original molecules can be of the same species:

$$3A \rightarrow B$$

Or of two different species:

$$2A + B \rightarrow C$$

2.2.1.1.2.2 Enzymatic reaction Another multi-step reaction is an enzymatic reaction, where the enzyme E that catalyses the rate of conversion is kept the same:

$$A + E \rightarrow B + E$$

2.2.1.2 Equation-based methods

The previous way to represent reactions does not give insight on the function of the equation. To shine light on this aspect equation-based methods need to be applied, like ordinary differential equation systems. In these the derivative with respect to time is specified for any of the variables in term of a function of the state. It can be noted how the stochastic simulation approach can be approximated by a deterministic one.

2.2.1.3 Creating a formal mathematical description of a biological system

To write a mathematical description of a biological system there is a need to:

1. Choose the biochemical reactions' representation.
2. Choose the type of variable to describe the state of the system.
3. Specify the likelihood of execution for each reaction.

A reaction will be faster when there is a huge amount of reactants. This is because in term of probability, the more a species is present, the more it is likely to interact and for the reaction is to happen. This is true only when the assumption of that there is a well-mixed reaction volume is made. This assumption entails that the distribution of molecules is uniform. Moreover, in order to approximate a complex system the main compartments can be partitioned in smaller sets where a discretization procedure can be applied. Therefore in this case Petri nets are a suitable representation for these kind of systems.

2.2.1.4 Well-mixed reaction volume

A well-mixed reaction volume is a reaction volume in which all the molecular species are homogeneously distributed and spatially indistinguishable. This is legitimate as non-reactive conditions are much more frequent than reactive ones. The biochemical reaction system with well-mixed volume thus satisfies the spatial homogeneity condition, where spatial distribution of molecular species can be ignored. Chemical species under the well-mixed assumption at a thermal equilibrium are uniformly distributed in the volume V and their velocities are thermally randomized according to the Maxwell-Boltzmann distribution.

2.2.1.5 Formal representation

The state of a spatially homogeneous biological system is determined by the population of each species, while the position and velocity of each individual molecule are ignored. Let $X_i(t)$ be the population of species S_i at a particular time t. The N-vector $X(t) = (X_1(t), ..., X_N(t))$, which determines the population of each species, constitutes the system state at the time t. Chemical species interact through M reactions $R_1, \ldots, R_M$. A general reaction R_j has a general scheme:

$$v_{j1}^{-} S_1 + ... + v_{jN}^{-} S_N \rightarrow v_{j1}^{+} S_1 + ... + v_{jN}^{-} S_N$$

Where:

- Species on the left side are reactants.
- Species on the right side are products.
- The non-negative integers v_{ji}^{-} and v_{ji}^{+} are the stoichiometric coefficients which denote the number of molecules of a reactant that

are consumed and the number of molecules of a product that are produced.

A reactant species that affects the speed of a reaction but is not consumed by it is called a catalyst. The sum of stoichiometric coefficients of reactants of a reaction R_j is called reaction order. For each reaction R_j, the net change in the population of species S_i involved in the reaction is equal to $v_{ji}^{+} - v_{ji}^{-}$, which can be positive, negative or zero.

2.2.1.6 Stoichiometric matrix

The net changes by all reactions are described by a stoichiometry matrix $\vec{v}$ of size MN. The jth row $\vec{v}_j$ of the stoichiometry matrix expresses the changes caused by reaction R_j and it is called the state change vector. Different system can lead to the same stoichiometric matrix. To describe a reaction two matrices are required:

- V^{+} provides the products.
- V^{-} provides the reactants.

$$V^{-} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, V^{+} = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 2 \\ 2 & 0 & 0 \end{bmatrix}, V = \begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix} \; V = V^{+} + V^{-}$$

Suppose that at a time t the state, or number of molecules in that given moment is $X(t)$. It is further assumed that the next reaction scheduled to fire at the next time $t + \tau$ is R_μ, which move the system accordingly to a new state $X(t + \tau)$. $\vec{x}$ is a simple notation to represent $X(t)$. Two important assumptions are made for the transition from state $X(t)$ to state $X(t + \tau)$:

- No changes occur in the system state during the time interval $[t, t + \tau[$, before the next reaction fires.
- The reaction occurs instantly after it is initiated.

These are called the Markov property. So the evolution of the system in a time step will be:

$$X(t + \tau) = X(t) + v_\mu$$

2.2.1.7 Summary

While specifying in computational terms the biological system of interest, the structure of at least one compartment has to be described, where it is assumed to find a chemical volume in which there are some entities that interact with each other. The well-mixed volume preliminary assumption is enforced, meaning that all the actors are present with equal availability in all the parts of the volume. For each compartment, the entities or how many molecules are available, and the reactions that provide the rules for transforming the chemicals in others along the time have to be specified. All these structures of reactions can be defined in mathematical terms using matrices: the number of columns is equal to the number of variables, while the number of rows is equal to the number of reactions and the stoichiometric coefficient of the transformation from reactant to products is specified.

2.2.2 Reaction propensity

Each reaction in stochastic chemical kinetics is considered as a *stochastic process* where each of its occurrences is a random event with an assigned probability distribution. It is impossible to predict the progress of reactions deterministically, but only stochastically with a probability. The propensity of a reaction is a formula that is computed on a state of a system. Through the reaction propensity the probability of execution of a reaction will be hinted.

2.2.2.1 Definition

The propensity a_j of a reaction R_j is defined such that $a_j(\vec{x})dt$ is the probability that a reaction R_j fires in the next infinitesimal time interval $[t, t+dt[$, given the state $X(t) = \vec{x}$ at time t. In a chemical setting, the probability of execution of one reaction will be proportional to the viability of the reactant.

2.2.2.2 Value of the propensity

The propensity $a_j(X(T))$ is a function of the state $X(t)$. The fact that a_j of a reaction depends on time t happens implicitly through the Markovian assumptions. At a particular time t, the value of the propensity $a_j(X(T))$ is a deterministic quantity. The propensity value of a reaction in state $X(t)$ is often used as a measure of how fast the reaction proceeds to move to a new state. Let $\mathcal{P}\{R_j \text{ fires } \in [t, t+dt[\}$ be the probability that reaction R_j fires in the time interval. Given the state $X(t) = \vec{x}$ at time t, then:

$$\mathcal{P}\{R_j \text{ fires } \in [t, t+dt[\} = a_j(\vec{x})dt + o(dt)$$

Where $o(dt)$ expresses that it asymptotically approaches zero faster than dt: $\lim_{dt\to 0} \frac{o(dt)}{dt} = 0$. In other words the probability that there are more than one firing of R_j in the time step i in order of $o(dt)$ and so it is negligible. The propensity of a reaction is an intrinsic property of the reaction and is linked to the phenomenon that the reaction is going to represent. The probability will depend on the propensity of the reaction and the other reactions that are competing for the same reactant. The propensity is not affected by the products.

$$a_1(0) = c_1 h_1(x(0))$$

$$h_2(x(0)) = \binom{100}{1}\binom{50}{1}\binom{30}{0}$$

$$x(0+\tau) = x(0) + V_1 = \begin{bmatrix} 99 & 51 & 30 \end{bmatrix}$$

$$x(0) = \begin{bmatrix} 100 & 50 & 30 \end{bmatrix}$$

Since the combination is being computed a^2 will not be taken into account as in the canonical law of mass action.

2.2.2.3 Mass action propensity

A precise formula for the propensity function a_j is dependent on the kinetics and a specific assumption about how the reaction occurs physically. This is referred as the fundamental premise of the stochastic chemical kinetics. For the standard mass action kinetics, the propensity a_j of reaction R_j is proportional to a stochastic reaction rate c_j and the number of its reactants. So given current state $X(t)$ at time t:

$$a_j(X(t)) = c_j h_j(X(t))$$

Where c_j is the stochastic reaction rate and $h_j(X(t))$ counts the number of distinct combination of reactants:

$$h_j(X(t)) = \prod_i \binom{X_i(t)}{v_{ji}^-} = \prod_i \frac{X_i(t)!}{v_{ij}^0!(X_i(t) - v_{ij}^-)!}$$

In the case of a synthesis reaction, where the stoichiometric coefficient of its reactants is 0 is set to $h_j(X(t)) = 1$.

2.2.2.3.1 The stochastic rate The stochastic rate c_j denotes the average probability per unit time that a particular combination of reactant molecules of reaction R_j reacts in the volume V and depends on the reaction type. In the case of a unimolecular reaction is independent of the volume size, while in the case of a bimolecular one it will be inversely proportional of it. The rate is constant provided that:

- The volume V is constant.
- The volume is well-mixed.
- The volume is thermally homogeneous.

2.2.2.4 Reaction propensity for reactions R_j with mass action kinetics

From now X_i will be used in place of $X_i(t)$, when t is irrelevant or clear from the context. Some reaction will be described by mass action kinetics by:

- Synthesis reaction ($\emptyset \rightarrow$ products): the number of combinations $h_j = 1$ and propensity $a_j = c_j$
- Unimolecular reaction ($S_i \rightarrow$ products): the number of combinations $h_j = X_i$ and propensity $a_j = c_j X_i$.
- Bimolecular reaction ($S_i + S_k \rightarrow$ products): the number of combinations $h_j = X_i X_k$ and propensity $a_j = c_j X_i X_k$.
- Dimerization reaction ($2S_i \rightarrow$ products): the number of combinations $h_j = \frac{1}{2} X_i(X_i - 1)$ and propensity $a_j = \frac{1}{2} c_j X_i(X_i - 1)$.
- Polymerization reaction ($3S_i \rightarrow$ products): the number of combinations $h_j = 16 X_i(X_i - 1)(X_i - 2)$ and propensity $a_j = 16 c_j X_i(X_i - 1)(X_i - 2)$.
- Termolecular reaction ($2S_i + S_k \rightarrow$ products): the number of combinations $h_j = \frac{1}{2} X_i(X_i - 1) X_k$ and propensity $a_j = \frac{1}{2} c_j X_i(X_i - 1) X_k$.

2.2.2.5 Complex reaction kinetics

Complex reaction kinetics can be used. The propensity a_j will show a complicated, non-linear dependence on the chemical species and may contain more than one rate constant.

2.2.2.5.1 Michaelis-Menten kinetics The Michaelis-Menten kinetics is used to approximate the mechanism of enzymatic reaction. Consider an enzymatic reaction R_j with form:

$$S_i + S_k \rightarrow S_i + S_j$$

Where:

- S_k is the substrate.
- S_i is the enzyme.

The reaction propensity will be defined as:

$$a_j = \frac{V_{\max}}{K_M + x_k} X_i X_k$$

Where:

- $V_{\max}$ is the maximum rate such that the substrate S_k is saturated.
- K_m or Michaelis constant is the substrate concentration at which the reaction rate is half of $V_{\max}$.

2.2.3 Chemical Master Equation

The chemical master equation or CME is the theoretical approach allowing to derive the complete set of probabilities of all the possible states of the system. Suppose that the biochemical reaction system starts with an initial state $X(t_0) = \vec{x}_0$. Let $t > t_0$ and the system at state $X(t) = \vec{x}$, The purpose of the stochastic chemical kinetics is to infer the probability:

$$\mathbb{P}\{\vec{x}, t|\vec{x}_0, t_0\}$$

2.2.3.1 Grand probability function

The probability function or grand probability:

$$\mathbb{P}\{\vec{x}, t|\vec{x}_0, t_0\}$$

Is the probability that the system state is $X(t) = \vec{x}$ at time t, given the initial state $X(t_0) = \vec{x}_0$ at time t_0. It is called so because it gives the probabilities of all reachable states of the system at time t given the initial state. Knowing it all the statistical properties can be computed for every species at any time.

2.2.3.2 Deriving the chemical master equation

To derive the time evolution of the grand probability, consider an infinitesimal time interval $[t, t+dt[$ so that there is at most one reaction in that interval. Suppose that at time $t+dt$ the system state is $X(t+dt) = \vec{x}$. There are two cases in order to reach the state $\vec{x}$ in the time-step given the current time:

- At time t the state is $X(t) = \vec{x} - \vec{v}_j$ and reaction R_j fires in the next time step leading to $X(t + dt) = \vec{x}$.
- At time t the state is already $X(t) = \vec{x}$ and no reaction fires in the time step.

Then the grand probability can be written as:

$$\begin{aligned}\mathbb{P}\{\vec{x}, t + dt|\vec{x}_0, t_0\} = &\sum_{j=1}^{M} \mathbb{P}\{R_j \text{ fires } \in [t, t + dt[\}\mathbb{P}\{\vec{x} - \vec{v}_j, t|\vec{x}_0, t_0\} + \\ &+ \mathbb{P}\{\text{No reaction fires } \in [t, t + dt[\}\mathbb{P}\{\vec{x}, t|\vec{x}_0, t_0\}\end{aligned}$$

Note that when the state vector $\vec{x}-\vec{v}$ gives negative populations, the probability $\mathbb{P}\{\vec{x}-\vec{v}_j, t|\vec{x}_0, t_0\}$ is zero because the populations of species must be positive. Now, the probability that no reaction fires in the time-step can be computed as:

$$\begin{aligned}\mathbb{P}\{\text{No reaction fires } \in [t, t + dt[\} &= \prod_{j=1}^{M}(1 - \mathbb{P}\{R_j \text{ fires } \in [t, t + dt[\}) = \\ &= \prod_{j=1}^{M}(1 - a_j(\vec{x})dt + o(dt)) = \\ &= 1 - \prod_{j=1}^{M} a_j(\vec{x})dt + o(dt)\end{aligned}$$

Then substituting the definition of the probability of a reaction firing and the probability of a reaction non firing just computed into the grand probability function:

$$\begin{aligned}\mathbb{P}\{\vec{x}, t + dt|\vec{x}_0, t_0\} = &\sum_{j=1}^{M} \mathbb{P}\{\vec{x} - \vec{v}_j, t|\vec{x}_0, t_0\}(a_j(\vec{x} - \vec{v}_j)dt + o(dt)) + \\ &+ \mathbb{P}\{\vec{x}, t|\vec{x}_0, t_0\}\left(1 - \sum_{j=1}^{M} a_j(\vec{x})dt + o(dt)\right)\end{aligned}$$

Subtracting $\mathbb{P}\{\vec{x}, t|\vec{x}_0, t_0\}$ from both sides, dividing by dt and considering the limit $dt \to 0$, the chemical master equation is obtained:

$$\frac{\mathbb{P}\{\vec{x}, t|\vec{x}_0, t_0\}}{dt} = \sum_{j=1}^{M}(a_j(\vec{x} - \vec{v}_j)\mathbb{P}\{\vec{x}, t|\vec{x}_0, t_0\}) - \mathbb{P}\{\vec{x}, t|\vec{x}_0, t_o\}\sum_{j=1}^{M} a_j(\vec{x})$$

The chemical master equation is a collection of differential equations in which each of them represents the probability of each possible state of the system at time t. It provides a complete description of the time evolution of the grand probability:

$$\mathbb{P}\{\vec{x}, t|\vec{x}_0, t_0\}$$

2.2.3.3 Limitations of the chemical master equation

The chemical master equation allows to compute the probabilities of all possible states at any time, however directly solving it require a lot of computational challenges:

- An analytical or numerical approach to solve CME in general is non-trivial to find.
- There is a huge number of differential equations: a^N, where N is the number of species and a is the values of the population of each species.

So it can be seen how the number of differential equations increases exponentially, making the state space explode in dimension, preventing direct approaches in solving CMEs.

2.3 Stochastic simulation

When applying the CME, all the possible state of the system are explored. With a stochastic simulation only one possible trajectory of the system is computed (to obtain the total description many simulations have to be run). The stochastic simulation works because only an idea of the most probable conditions of the system is needed. A stochastic simulation is faster than a real experiment and it can be run on a personal computer (also thousands of simulations can be performed). One of the crucial points of a stochastic simulation is that there is a need to be able to sample from this probability distribution function given the fact that the computer has few ways of generating something that is stochastic. The easiest way to do so is through a random number generator.

2.3.1 Probability density function

Stochastic simulations are an alternative approach to solve CME by producing possible realization of the grand probability function. It only explores possible states in the state space each time, so they can handle the biochemical reactions with very high-dimensional state space. The mathematical basis of a stochastic simulation is the reaction probability density function pdf:

$$p(\tau, \mu|\vec{x}, t)$$

Which is defined such that $p(\tau, \mu|\vec{x}, t)d\tau$ is the probability that a reaction R_μ fires in the next infinitesimal time interval $[t+\tau, t+\tau+d\tau[$, given the state $X(t) = \vec{x}$ at time t. The pdf $p(\tau, \mu|\vec{x}, t)$ is a joint distribution of two variables showing the index μ of the reaction firing R_μ at time τ of the firing respectively, knowing that the system is at state $X(t)$ at time t. The reaction index μ will vary from $1 \leq \mu \leq M$, while the next time τ will vary from $0 \leq \tau \leq \infty$.

2.3.1.1 Computing the probability density function

The probability $p(\tau, \mu|\vec{x}, t)dt$ can be computed as the product of:

- The probability that no reaction fires in the time interval $[t, t+\tau[$.
- The probability that a reaction R_μ fires in the next infinitesimal interval $[t+\tau, t+\tau+d\tau[$.

Let:

- $\mathbb{P}\{\text{No reaction fires } \in [t, t+\tau[\}$ be the probability that no reaction fires in the time interval $[t, t+\tau[$.
- $\mathbb{P}\{R_\mu \text{ fires } \in [t+\tau, t+\tau+d\tau[\}$ be the probability that reaction R_μ fires in the next infinitesimal time interval $[t+\tau, t+\tau+d\tau[$.

Then:

$$p(\tau, \mu|\vec{x}, t)d\tau = \mathbb{P}\{\text{No reaction fires } \in [t, t+\tau[\}\mathbb{P}\{R_\mu \text{ fires } \in [t+\tau, t+\tau+d\tau\}$$

To compute the probability that no reaction fires the time interval is divided into k non overlapping sub interval of length $\epsilon = \frac{k}{\tau}$. The probability that no reaction fires in the i-th interval $[t+(i-1)\epsilon, t+i\epsilon[\forall i = 1, \ldots, k$ is:

$$\mathbb{P}\{\text{No reaction fires } \in [t+(i-1)\epsilon, t+i\epsilon[\} = 1 - \sum_{j=1}^{M} a_j(\vec{x})\epsilon + o(\epsilon)$$

Hence the probability that no reaction fires in $[t, t+\tau[$ is the product of the probabilities that no reaction fires in k non-overlapping intervals:

$$\begin{aligned}
\mathbb{P}\{\text{No reaction fires } \in [t, t+\tau[\} &= \prod_{i=1}^{k} \mathbb{P}\{\text{No reaction fires } \in [t+(i-1)\epsilon, t+i\epsilon[\} = \\
&= \prod_{i=1}^{k} \left(1 - \sum_{j=1}^{M} a_j(\vec{x})\epsilon + o(\epsilon)\right) = \\
&= \left(1 - \sum_{j=1}^{M} a_j(\vec{x})\epsilon + o(\epsilon)\right)^k = \\
&= (1 - a_0(\vec{x})\epsilon + o(\epsilon))^k
\end{aligned}$$

Where $a_0(\vec{x})$ is the total propensity defined as:

$$a_0(\vec{x}) = \sum_{j=1}^{M} a_j(\vec{x})$$

In the limit case where $k \to \infty$:

$$\begin{aligned}
\mathbb{P}\{\text{No reaction fires } \in [t, t+\tau[\} &= \lim_{k\to\infty} (1 - a_0(\vec{x})\epsilon + o(\epsilon))^k = \qquad = \lim_{k\to\infty} \left(1 - \frac{a_0(\vec{x})k\epsilon + ko(\epsilon)}{k}\right)^k = \\
&= \lim_{k\to\infty} \left(1 - \frac{a_0(\vec{x})\tau + \tau(\frac{o(\epsilon)}{\epsilon}}{k}\right)^k = \\
&= e^{-a_0(\vec{x})\tau}
\end{aligned}$$

The last step is true because:

- $\frac{o(\epsilon)}{\epsilon} \to 0$ when $k \to \infty$.
- $\lim_{k\to\infty} \left(1 - \frac{a_0(\vec{x})\tau}{k}\right)^k = e^{-a_0(\vec{x})\tau}$.

The probability that a reaction fires, following the definition of the reaction propensity is computed by:

$$\mathbb{P}\{R_\mu \text{ fires } \in [t+\tau, t+\tau+d\tau[\} = a_\mu(\vec{x})d\tau + o(d\tau)$$

Now plugging everything in the definition of the probability:

$$p(\tau, \mu|\vec{x}, t)d\tau = e^{-a_0(\vec{x})\tau}(a_\mu(\vec{x})d\tau + o(d\tau))$$

Dividing both sides by $d\tau$ and taking the limit $d\tau \to \infty$ and remembering that $\frac{o(d\tau)}{d\tau} \to 0$, the pdf will have the formula:

$$p(\tau, \mu|\vec{x}, t) = a_\mu(\vec{x})e^{-a_o(\vec{x}\tau)}$$

Which is the joint probability density function of the next reaction index μ and the next firing time τ over their domains. This can be verified as:

$$\int_0^\infty d\tau \sum_{\mu=1}^{M} p(\tau, \mu|\vec{x}, t) = \sum_{\mu=1}^{M} a_\mu(\vec{x}) \int_0^\infty d\tau e^{-a_0(\vec{x})\tau} = 1$$

The pdf depends on the propensities of all reaction as well as on all species.

2.3.2 Stochastic simulation algorithm

$$p(\tau, \mu|\vec{x}, t) = a_\mu(\vec{x})e^{-a_o(\vec{x}\tau)}$$

Is the framework for a class of exact Monte Carlo simulation techniques originating from the stochastic simulation algorithm or SSA. It is a discrete event simulation: the state is updated by a random selected reaction R_μ and a discrete time τ sampled from the pdf. It exactly generates μ of the reaction and the firing time τ without introducing approximation in sampling the pdf. A general sketch of the SSA procedure is outlined in algorithm 1.

Algorithm 1: `Stochastic simulation algorithm (SSA) - General Sketch()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
while $t < T_{\max}$ **do**
    $a_0 = 0$
    foreach R_j **do**
        compute a_j
        $a_0 = a_0 + a_j$
    sample reaction R_μ and firing time τ from $p(\tau, \mu, |\vec{x}, t)$
    $\vec{X} = \vec{X} + \vec{v}_\mu$
    $t = t + \tau$

For each iteration in the while loop the algorithm computes the propensity a_j for each reaction and the total propensity. The next reaction R_μ and its firing time τ are sampled from the pdf, a step that may need the generation of randomly distributed random numbers. The state is updated and the time is advanced to $t = t+\tau$. The loop is repeated until $t > T_{\max}$. The propensities are updated at each iteration to reflect changes in the populations of species caused by reaction firings, but this can be skipped by employing an appropriate sampling technique. The result of the algorithm is a trajectory, that shows the evolution of the biological system over time. This is a collection of states $X(t)$ that denotes the state of the system at any time $0 \leq t \leq T_{max}$. It should be emphasized that because SSA is a discrete event simulation algorithm, the state changes only at discrete time instants when reactions fire. The state between two reaction firings is a constant.

2.4 Simulation output analysis

The trajectory obtained from a SSA run represents a possible realization of the grand probability $\mathbb{P}\{\vec{x}, t|\vec{x}_0, t_0\}$. Many independent run started with the same initial conditions are needed to have a reasonable statistical estimation.

2.4.1 Confidence interval estimation

Let K be the number of simulation and $\vec{X}^r$, with $r = 1, \ldots, K$ be a realization of the state $\vec{X}$ obtained at time t by the r-th independent run of SSA. The statistical properties can be derived from the ensemble of the trajectories, that are ensured to approach the exact solution of the CME as $K \to \infty$. Let $\langle\vec{X}\rangle$ be the sample mean and s^2 the unbiased variance. They are computed as:

$$\langle\vec{X}\rangle = \frac{\sum_{r=1}^{K} \vec{X}^r}{K} \qquad s^2 = \frac{\sum_{r=1}^{K} (\vec{X}^r - \langle\vec{x}\rangle)^2}{K-1}$$

The sample mean and variance will approach the mean and variance of the random variable $\vec{X}$ when K tends to infinity:

$$\mathbb{E}[\vec{X}] = \lim_{K\to\infty} \langle\vec{X}\rangle \qquad Var[\vec{X}] = \lim_{K\to\infty} s^2$$

The convergence of the estimation is measured by the size of the confidence interval as the number of simulation runs is limited:

$$d = \frac{zs}{\sqrt{K}}$$

Where z is a confidence interval, denoting the percentage of the range of estimated values expected to include the true value. When it is fixed, the probability that the mean lies in the interval $[\langle\vec{x}\rangle - d, \langle\vec{x}\rangle + d]$ is $2\phi(z) - 1$, where ϕ is the cumulative distribution function or cdf of the normal distribution $norm(0, 1)$. The required number of simulation runs to achieve a specified confidence interval size can be computed:

$$K = \frac{z^2 s^2}{d^2}$$

K depends reciprocally on the square of the confidence interval size d and on the sample variance s^2, which is unknown. One approach to circumvent this problem is to perform first a small number of trial runs to estimate s^2_{trial}, which is applied to compute the number of simulation runs necessary:

$$K = \frac{z^2 s_{trial}^2}{d^2}$$

2.4.2 Probability distribution equation

Bistability is a form of multistablity where two separated stable equilibrium points are separated by an unstable equilibrium, the average population of species might not provide enough information for their dynamical behaviour. The probability distribution must be used to quantitatively analyse the simulation results. The probability distribution can be estimated by using the histogram or empirical distribution function of the samples. It is ensured to converge to the exact probability distribution given a large number of simulation runs K. The following assumes X to be a scalar value, but it can be easily extended to the general case. To compute the histogram the state X at time t obtained by K simulation runs of SSA is supposed to be bounded into an interval $[X_{\min}, X_{\max}]$, chosen arbitrarily. Then the interval is divided into B bins, such that the i-th bin I_i is defined as the subinterval:

$$\left[X_{\min} + \frac{(i-1)L}{B}, X_{\min} + \frac{iL}{B}\right]$$

Where $L = X_{\max} - X_{\min}$. The histogram h_X of state X is then defined as:

$$h_X(I_i) = \frac{B}{KL} \sum_{r=1}^{K} \mathcal{X}(X^r, I_i)$$

Where X^r is the realization of X by the r-th simulation and $\mathcal{X}(X^r, I_i)$ is defined as:

$$\mathcal{X}(X^r, I_i) = \begin{cases} 1 & X^r \in I_i \\ 0 & otherwise \end{cases}$$

The histogram therefore gives the average probability of X in interval I_i. Let now p_X be the probability distribution of state X. When $K \to \infty$ and $B \to \infty$, I_i reduces to a point and h_X converges to p_X at this point. Formally:

$$p_X = \lim_{K,B \to \infty} h_X$$

Chapter 3

Implementation of Stochastic Simulation Algorithms

3.1 Introduction

3.1.1 Non-deterministic vs stochastic

Working under the assumption of using the same model and parameters:

- A deterministic system does not show randomness and the same result is always obtained.
- A non-deterministic system shows some degree of uncertainty: different runs have different results.

3.1.1.1 Exact stochastic simulation

In an exact stochastic simulation, if some hypotheses are satisfied the system will behave like the biological one. Although the probability function could be computed, this does not make the method deterministic: uncertainty is intrinsic in the model. Theoretically there is no insight on the execution of the reactions in a stochastic setting, but a high level of accuracy can be reached thanks to the probability function.

3.1.2 Advantages of a non-deterministic approach

The reasoning behind the employment of a non-deterministic approach lies in the fact that to model a biological system there is a need to compromise between time and complexity. In non-deterministic polynomial time algorithms don't have an efficient solution, but it seems possible to find it. A non-deterministic setting allows us to understand whether an algorithm can be solved in polynomial time by step-wise guessing.

3.1.3 Categories of the exact simulation algorithms

A summary of the main exact stochastic simulation algorithms is reported in figure 3.1.

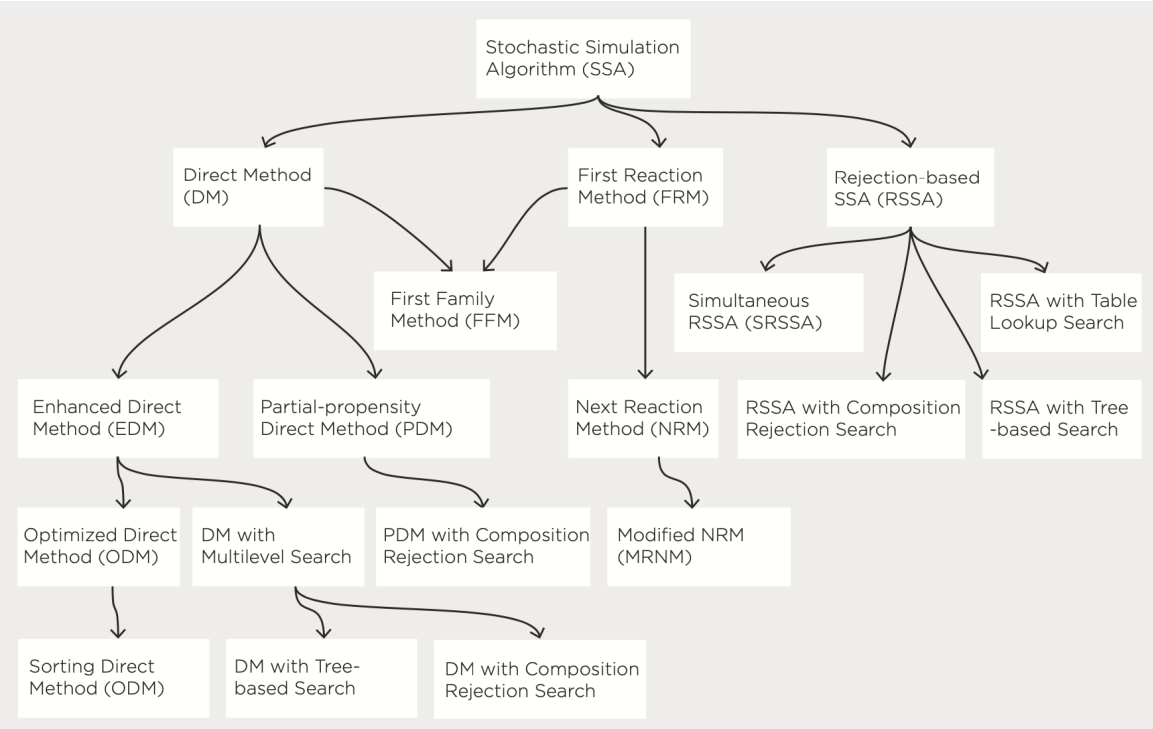

Figure 3.1: Main stochastic simulation algorithm.

3.2 Direct method

Gillespie's direct method defines a couple of formulae able to understand how the system will execute in terms of time τ and reactions μ. Since each time step is infinitesimal each reaction occurs and ends exactly at time τ, hence there cannot be multiple reactions firing simultaneously. Let a_0 be the sum of all propensities in the system, then the algorithm works as follow:

1. Sample one random number from the distribution $a_0 = \sum_{j=1}^{M} a_j \rightarrow V_1 = U(0,1)$.
2. Scale it to the maximum $V_1 \cdot a_0 = U(0, a_0)$.
3. See where this number will point over the different propensities (Figure 3.2).

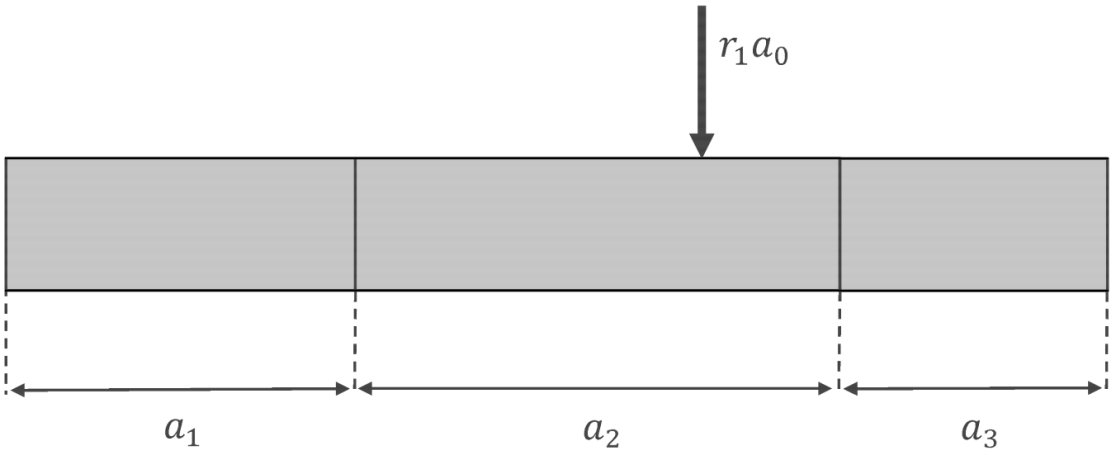

Figure 3.2: Boundaries on the propensities.

4. Generate another random number $V_2 = U(0, 1)$
5. $\tau \sim exp(a_0)$
6. $\tau = \frac{1}{a_0} ln(\frac{1}{V_2})$.

3.2.1 Mathematical discussion

Gillespie's direct method is used to sample the pdf $p(\tau, \mu|\vec{x}, t)$. The direct method partitions the joint probability density function into the product of two one-variable probability functions, one for τ and one for μ that can be sampled independently. The pdf can be factorized by the chain rule of probability as:

$$p(\tau, \mu|\vec{x}, t) = p_1(\tau|\vec{x}, t)p_2(\mu|\vec{x}, t)$$

Where:

- p_1 is the probability density function of the firing time τ.
- p_2 is the probability density function of the reaction with index μ that fires at $t + \tau$.

So that $p_1(\tau|\vec{x}, t)d\tau$ is the probability that a reaction will fire in the next time interval $[t + \tau, t + \tau + d\tau[$. This marginal probability is obtained by summing the probability $p(\tau, \mu|\vec{x}, t)d\tau$ over the domain of all possible values of reaction index μ:

$$p(\tau|\vec{x}, t) = \sum_{\mu=1}^{M} p(\tau, \mu|\vec{x}, t) = \sum_{\mu=1}^{M} a_\mu e^{a_0 \tau} = a_0 e^{-a_0 \tau}$$

Where a_0 is the total propensity. Plugging this and recalling the formula of the joint pdf:

$$p_2(\mu|\tau, \vec{x}, t) = \frac{p(\tau, \mu|\vec{x}, t)}{p_1(\tau|\vec{x}, t)} = \frac{a_\mu}{a_0}$$

It can be seen how p_2 is independent of τ, so it can be written as:

$$p_2(\mu|\vec{x},t) = p_2(\mu|\tau,\vec{x},t) = \frac{a_\mu}{a_0}$$

To verify that these two equation are part of the pdf:

$$\int_0^\infty p_1(\tau|\vec{x},t)d\tau = \int_0^\infty a_0 e^{-a_0\tau}d\tau = 1 \qquad \wedge \qquad \sum_{\mu=1}^{M} p_2(\mu|\vec{x},t) = \sum_{\mu=1}^{M} \frac{a_\mu}{a_0} = 1$$

The direct method uses $p_1(\tau|\vec{x},t)$ to sample the firing time τ and $p_2(\mu|\vec{x},t)$ to sample the reaction index μ. Since the two pdfs are independent the firing time and the reaction index can be sampled independently, so that the order of sampling does not effect the exactness of the direct method. The generated firing time τ and the next reaction firing R_μ are ensured to have the pdf $p(\tau,\mu|\vec{x},t)$ specified by the stochastic simulation algorithm, so that the generated trajectories are exact.

3.2.1.1 Choice of the reaction

The selection of the next reaction index μ has probability $\frac{a_\mu}{a_0}$. Given M discrete probabilities $\frac{a_j}{a_0}$ with $j = 1,\ldots,M$, the choice of the next reaction index:

$$\mu = \arg\min_{\mu\in j=1,\ldots,M} \sum_{j=1}^{M} \frac{a_j}{a_0} \geq r_1 =$$

$$= \arg\min_{\mu\in j=1,\ldots,M} \sum_{j=1}^{M} a_j \geq r_1 a_0$$

Where r_1 is a uniformly distributed random number $norm(0,1)$. To select the next reaction firing R_μ, the direct methods accumulates the sum $\sum_{j=1}^{\mu} a_j$ until it finds the smallest index μ that satisfies that inequality.

3.2.1.2 Selection of the firing time

For the reaction of the firing time τ, consider its pdf $p_1(\tau|\vec{x},t)$. It can be noted how it is an exponential distribution with rate a_0. So the firing time can be generated as:

$$\tau = \frac{1}{a_0}\ln\frac{1}{r_2}$$

Where r_2 is a uniformly distributed random number $norm(0,1)$.

3.2.2 The algorithm

The independent sampling of the firing time and of the reaction index are the basis of each simulation step of the direct method, outlined in algorithm 2.

Algorithm 2: `Direct Method (DM)()`

1 **Input**: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
2 **Output**: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
3 $t = 0$
4 $\vec{X} = \vec{x}_0$
5 **while** $t < T_{\max}$ **do**
6 $a_0 = 0$
7 **foreach** R_j **do**
8 compute a_j
9 $a_0 = a_0 + a_j$
10 generate two random numbers $r_1, r_2 \sim norm(0, 1)$
11 select R_μ with the smallest index μ such that $\sum_{j=1}^{\mu} a_j \geq r_1 a_0$
12 $\tau = \frac{1}{a_0} \ln \frac{1}{r_2}$
13 $\vec{X} = \vec{X} + \vec{v}_\mu$
14 $t = t + \tau$

Lines 10-12 implement the sampling of the joint reaction probability density function of the next reaction firing and its firing time. The simulation needs two random number, the first is used to select the next reaction firing with probability $\frac{a_\mu}{a_0}$, while the second for the firing time. The state is then advanced to the new one and the time is moved to $t + \tau$.

3.2.2.1 Time complexity

The computational cost for the generation of random numbers, the firing time and the update of simulation time are constant. Moreover the update of the state can be considered constant as only a few species are involved in a reaction. Because of this the computational cost of the algorithm arises due to:

- The computation of reaction propensities due to state changes at lines 7-9.
- The selection of the next reaction firing at line 11.

The direct method computes M reaction propensities for each simulation step, so the time complexity for this step is of $O(M)$. The search for the next reaction in the worst case requires to sum up all the M reaction propensities, making the cost for searching the next reaction firing $O(M)$. Summing up the time complexity for each simulation step of the direct method is $O(M)$.

3.2.3 Enhanced direct method

The enhanced direct method EDM reduces the number of propensity computation for each simulation iteration, recomputing only the propensity of reactions that change. The detection of changes in the reaction propensity is based on the fact that the propensity of a reaction changes only when the population of the reactants involved in the reaction changes only then the population of the reactants involved are changed by the reaction firing. Only the propensity of reaction that have reactant

population changed are recomputed. This is decided by analysing the dependency relationship between reactions.

3.2.3.1 Reaction dependency graph

A reaction R_j is dependent on a reaction R_μ if its propensity a_j is changed when R_μ fires. This relationship is collected and presented in a reaction dependency graph.

3.2.3.1.1 Some definitions

3.2.3.1.1.1 Reactants and products set For each reaction R_j, with $j = 1, \dots M$, define:

$$Reactants(R_j) = \{S_i | S_i \text{ is a reactant of } R_j\} \qquad \wedge \qquad Products(R_j) = \{S_i | S_i \text{ is a product of } R_j\}$$

3.2.3.1.1.2 Affects set The set of species involved in the computation of the propensity a_j of a reaction R_j is:

$$Affects(R_j) = \{S_i | a_j \text{ changes if population of } S_i \text{ changes}\}$$

3.2.3.1.1.3 Mass action kinetics For mass action kinetics, because the mass action propensity a_j of reaction R_j is proportional to its reactants:

$$Affects(R_j) = Reactants(R_j)$$

3.2.3.1.1.4 AffectedBy set The set of species whose population changes by firing reaction R_j is:

$$AffectedBy(R_j) = \{S_i | \text{Population of } S_i \text{ is changed if firing } R_j\}$$

3.2.3.1.1.5 Population of AffectedBy For each reaction R_j it is:

$$AffectedBy(R_j) \subseteq Reactants(R_j) \cup Products(R_j)$$

This is because $AffectedBy(R_j)$ includes species that are consumed and produced by reaction R_j excluding any species whose population is conserved.

3.2.3.1.2 Definition of the reaction dependency graph Let $\mathcal{R}$ be the set of reactions in the biochemical reaction network. The reaction dependency graph $G(V, E)$ is a directed graph with the vertex set $V = \mathcal{R}$ and the edge set E contains directed edges $e(R_j, R_k)$ from a reaction R_j to another reaction R_k if:

$$AffectedBy(R_j) \cap Affects(R_k) \neq \emptyset$$

All self-edges $e(R_j, R_j)$ belong to E.

3.2.3.1.3 Dependent reactions The set of reactions that are dependent on reaction R_j by the reaction dependency graph G is defined such that:

$$Dependents(R_j) = \{R_k | \exists \text{ a directed edge } e(R_j, R_k) \in G\}$$

The reaction dependency graph G determines the reaction for which propensities must be re-computed after firing. The number of reaction in the *Dependents* set is equal to the out-degree of the reaction in the dependency graph.

3.2.3.2 Algorithm

In the EDM algorithm the reaction dependency graph is the first thing built. This will be a static structure independent on the time evolution of the system and will be stored with a cost $O(M^2)$. The computation of propensity of all the reaction is performed only at the beginning of the simulation. For each iteration the selection is the same as in DM, then the new propensity for each reaction $R_j \in Dependents(R_\mu)$ is computed. The procedure is presented in algorithm 3.

Algorithm 3: `Enhanced Direct Method (EDM)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
build the reaction dependency graph G
$t = 0$
$\vec{X} = \vec{x}_0$
foreach R_j **do**
    compute a_j
    $a_0 = a_0 + a_j$
while $t < T_{\max}$ **do**
    generate two random numbers $r_1, r_2 \sim norm(0, 1)$
    select R_μ with the smallest index μ such that $\sum_{j=1}^{\mu} a_j \geq r_1 a_0$
    $\tau = \frac{1}{a_0} \ln \frac{1}{r_2}$
    $\vec{X} = \vec{X} + \vec{v}_\mu$
    $t = t + \tau$
    foreach $R_j \in Dependents(R_\mu)$ **do**
        compute a_j^{new}
        $a_0 = a_0 + (a_j^{new} - a_j)$
        $a_j = a_j^{new}$

In this way the propensity updates become local. Let D be the average number of reactions depending in a reaction, the cost of the propensity update for a simulation loop becomes $O(D)$, considering that $D < M$, so the propensity update in EDM is more efficient than in DM.

3.2.4 Improvements for Direct Method

3.2.4.1 Direct method with sorted reaction

The principle of the direct method with sorted reaction is to reduce the search depth of the direct method by re-indexing reactions, reducing the search depth of reactions that happens more frequently, improving simulation performance.

3.2.4.1.1 Optimized direct method The optimized direct method reduces the average search depth of the next reaction firing. This is done because in many biochemical networks, some reactions fire much more frequently than others.

3.2.4.1.1.1 Average search depth The average search depth S_m is the average number of operation performed for the selection of the next reaction firing:

$$S_M = \frac{\sum_{j=1}^{M} j n_j}{\sum_{j=1}^{M} n_j}$$

Where:

- j is the search index of reaction R_j.
- n_j is the number of times that R_j fires during the simulation.

These two values are not known so to order the reactions $\langle n_j \rangle$ the average estimation of n_j is used to order reaction. This is computed by some short pre-simulation runs.

3.2.4.1.1.2 Algorithm Optimized direct method is implemented as in algorithm 4.

Algorithm 4: `Optimized Direct Method (ODM)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
build the reaction dependency graph G
perform a few DM pre-simulation runs to estimate $\langle n_j \rangle$ of each reaction
order reaction indices such that $j < k$ if $\langle n_j \rangle > \langle n_k \rangle$
$t = 0$
$\vec{X} = \vec{x}_0$
foreach R_j **do**
    compute a_j
    $a_0 = a_0 + a_j$
while $t < T_{\max}$ **do**
    generate two random numbers $r_1, r_2 \sim norm(0, 1)$
    select R_μ with the smallest index μ such that $\sum_{j=1}^{\mu} a_j \geq r_1 a_0$
    $\tau = \frac{1}{a_0} \ln \frac{1}{r_2}$
    $\vec{X} = \vec{X} + \vec{v}_\mu$
    $t = t + \tau$
    foreach $R_j \in Dependents(R_\mu)$ **do**
        compute a_j^{new}
        $a_0 = a_0 + (a_j^{new} - a_j)$
        $a_j = a_j^{new}$

3.2.4.1.1.3 Discussion In the case of a fixed number of bits the sum of the biggest propensities placed in the front of the search list may be not enough to account for the rest: the reactions with very small propensity will never fire. Moreover the pre-simulation introduces an additional computational burden to the simulation. Moreover ODM assumes that the reaction order determined by the pre-simulation runs will characterize the long-term reaction behaviour, which could not be true.

3.2.4.1.2 Sorting direct method The sorting direct method SDM is a variant of ODM that does not use pre-simulation runs by maintaining an approximately sorted order of reaction. The ordering is built dynamically during the simulation run: the index of a reaction whenever it is selected to fire is dynamically bubbled up one step ahead in the reaction list. The reactions that have just occurred are put towards the top of the search list.

3.2.4.1.2.1 Algorithm The sorting direct method is implemented as in algorithm 5.

Algorithm 5: `Sorting Direct Method (SDM)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
build the reaction dependency graph G
foreach R_j **do**
    compute a_j
    $a_0 = a_0 + a_j$
while $t < T_{\max}$ **do**
    generate two random numbers $r_1, r_2 \sim norm(0, 1)$
    select R_μ with the smallest index μ such that $\sum_{j=1}^{\mu} a_j \geq r_1 a_0$
    $\tau = \frac{1}{a_0} \ln \frac{1}{r_2}$
    $\vec{X} = \vec{X} + \vec{v}_\mu$
    $t = t + \tau$
    foreach $R_j \in Dependents(R_\mu)$ **do**
        compute a_j^{new}
        $a_0 = a_0 + (a_j^{new} - a_j)$
        $a_j = a_j^{new}$
    if $\mu > 1$ **then**
        Swap R_μ and $R_{\mu-1}$ in the reaction list

3.2.4.1.2.2 Discussion The swapping step adds overhead to each simulation step, but it is negligible. SDM is thus suited to deal with the simulation of networks where the propensities change sharply.

3.2.4.2 Direct method with Multi-level search

The main bottleneck of DM is that the search for next reaction firing is slow in large reaction models. The multi-level search is an effort to reduce the time complexity of DM for large systems. The search problem is divided into smaller sub-problem partitioning the M reactions into L groups $G_1, \ldots, G_L$. Each group G_l contains k_l reactions. Let a^l be the sum of propensities of reactions in group G_l:

$$a^l = \sum_{R_j \in G_l} a_j$$

It is obvious that:

$$a_0 = \sum_{l=1}^{L} a^l$$

The selection of the next reaction firing is in two steps. First a group G_α is selected with probability $\frac{a^\alpha}{a_0}$. Then the next reaction firing R_μ is selected with probability $\frac{a_\mu}{a^\alpha}$ conditioning on the selected group G_α.

3.2.4.2.1 Exactness of the multi level search The next reaction R_μ in the group G_α that is selected by the multi-level search has probability $\frac{a_\mu}{a_0}$. Let $\mathbb{P}\{R_\mu\}$ be the probability of selecting the reaction R_μ. This can be expanded as:

$$\mathbb{P}\{R_\mu\} = \mathbb{P}\{G_\alpha\}\mathbb{P}\{R_\mu|G_\alpha\} = \frac{a^\alpha}{a_0}\frac{a_\mu}{a^\alpha} = \frac{a_\mu}{a_0}$$

3.2.4.2.2 Implementation An implementation to select the group index and the reaction index requires two random numbers:

$$\alpha = \arg\min_{\alpha \in l=1,\ldots,L} \sum_{l=1}^{\alpha} a^l \geq r_1 a_0$$

And:

$$\mu = \arg\min_{\mu \in k=1,\ldots,M} \sum_{\substack{k=1 \\ G_\alpha=\{R_j,\ldots,R_{j+k\alpha}\}}}^{\mu} a_k \geq r_2 a^\alpha$$

The need for r_2 can be avoided by recycling r_1:

$$\frac{r_1 a_0 - \sum_{l=1}^{\alpha-1} a^l}{a^\alpha}$$

Is a uniformly distributed random number. So r_1 is rescaled to select the next reaction firing in the group.

3.2.4.2.3 Algorithm The direct method with multi-level search is implemented as in algorithm 6.

Algorithm 6: `Direct method with multi level search()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
build the reaction dependency graph G
partition M reactions into L groups $\{G_1, \ldots, G_L\}$
$t = 0$
$\vec{X} = \vec{x}_0$
foreach G_l **do**
    $a^l = 0$
    foreach $R_j \in G_l$ **do**
        compute a_j
        $a^l = a^l + a_j$
    $a_0 = a_0 + a^l$
while $t < T_{\max}$ **do**
    generate two random numbers $r_1, r_2 \sim norm(0, 1)$
    select G_α with the smallest index α such that $\sum_{l=1}^{\alpha} a^l \geq r_1 a_0$
    $r_1 = \frac{r_1 a_0 - \sum_{l=1}^{\alpha-1} a^l}{a^\alpha}$
    select R_μ with the smallest index μ such that $\sum_{\substack{k=1 \\ G_\alpha = \{R_j, \ldots, R_{j+n}\}}}^{\mu} a_k \geq r_1 a^\alpha$
    $\tau = \frac{1}{a_0} \ln \frac{1}{r_2}$
    $\vec{X} = \vec{X} + \vec{v}_\mu$
    $t = t + \tau$
    foreach $R_j \in Dependents(R_\mu)$ **do**
        compute a_j^{new}
        $a^l = a^l + (a_j^{new} - a_j)$
        $a_j = a_j^{new}$

3.2.4.2.4 Discussion To analyse the time complexity of the multi-level search assume that M reactions are partitioned into $L = \left[\frac{M}{k}\right]$ groups and each group contains $k_l = k$ reactions. The time complexity has two parts:

- Searching for a group $O\left(\frac{M}{k}\right)$.
- Searching for a reaction within the group $O(k)$.

The total time complexity is then

$$O\left(\frac{M}{k}\right) + O(k) = O\left(\max\left\{\frac{M}{k}, k\right\}\right)$$

The total time is minimized by taking $k = c\sqrt{M}$, so that the minimal time complexity per reaction event is $O(\sqrt{M})$. The multi-level search can be further expanded partitioning the groups into sub-groups, introducing the multi-dimensional search method.

3.2.4.3 Direct method with tree-based search

The tree-based search refines the multi-level search. The finest partitioning of reaction is when the lowest level has at most two reaction creating a binary tree structure. Each node has two children or zero and the leaves hold reaction propensity, while internal node hold the sums of the values in their child nodes. The root of the tree holds a_0.

3.2.4.3.1 Dimension of a complete tree A complete binary tree with M leaves has $2M-1$ nodes. Let P be the number of internal nodes. In a complete tree each internal node has two children, hence the number of edges is $2P$. Also the edges are $M+P-1$, so $P = M-1$, so in conclusion the number of nodes is $P+M = 2M-1$.

3.2.4.3.2 Implementation of the tree The tree can be implemented by an array with $2M-1$ elements. The number of reaction M has to be even, and if it is not a dummy node with propensity 0 is added to the end of the array. Algorithm 7 outlines how the tree is built: it happens recursively from the leaves to the root, observing that a node in i will have children in position $2i$ and $2i+1$.

Algorithm 7: `build_tree(`*position*`)`

Input: an array TREE with $2M-1$ elements where elements from M to $2M-1$ are filled with M reaction propensities and a starting position.
Output: The complete binary tree represented by the array TREE.
if *position* $< M$ **then**
    `build_tree(`2·*position*`)`
    `build_tree(`2·*position*+1`)`
    $TREE[position] = TREE[2 \cdot position] + TREE[2 \cdot position + 1]$

3.2.4.3.3 Tree-based search The tree-based search for the next reaction firing R_μ given $s = ra_0$ starts by selecting the next branch of the tree by comparing the search value s with the value stored in the left child of the current node. Then the search selects the left branch if the value is less than the one stored in the left child of the node, otherwise the search chooses the right branch. If the right branch is selected the search value is subtracted by the value stored in the current node. This proceeds recursively until it reaches a leaf and the reaction stored in that leaf is returned with the correct probability $\frac{a_\mu}{a_0}$. This procedure is outlined in algorithm 8.

Algorithm 8: `search_tree(`*position, s*`)`

Input: A complete binary tree represented by the array TREE, and integer position and a search value s
Output: The leaf of the complete binary tree which stores the next reaction firing.
if $position \geq$ **then**
    return position
else if $TREE[2 \cdot position] \geq s$ **then**
    `search_tree(`$2 \cdot position$, s`)`
else
    $s = TREE[position] - s$
    `search_tree(`$2 \cdot position + 1$, s`)`

3.2.4.3.4 Updating the tree The system is updated after the selected reaction fires. The nodes of the tree will update their propensity value. For each reaction depending on the reaction firing according to the dependency graph G, its new propensity is computed and the difference is propagated for every of their paths. To optimize this implementation, reactions dependent on each other should be placed as close as possible on the tree. This procedure is outlined in algorithm 9.

Algorithm 9: `update_tree(`*position, c*`)`

Input: A complete binary tree represented by the array TREE.
Output: The complete binary tree updated by the reaction firing.
$TREE[position] = TREE[position] + C$
if *position is not root* **then**
    `update_tree(`$\lfloor \frac{i}{2} \rfloor$, c`)`

3.2.4.3.5 Algorithm The whole procedure is implemented in the algorithm 10.

Algorithm 10: `Direct method with tree-based search()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
build the reaction dependency graph G
$t = 0$
$\vec{X} = \vec{x}_0$
foreach R_l **do**
    compute a_j
build $TREE$ structure for M reaction propensities with 7
while $t < T_{\max}$ **do**
    generate two random numbers $r_1, r_2 \sim norm(0, 1)$
    select next reaction firing R_μ by algorithm 8 with $s = r_1 a_0$
    $\tau = \frac{1}{a_0} \ln \frac{1}{r_2}$
    $\vec{X} = \vec{X} + \vec{v}_\mu$
    $t = t + \tau$
    foreach $R_j \in Dependents(R_\mu)$ **do**
        compute a_j^{new}
        update the TREE by algoritm 9 with $c = a_j^{new} - a_j$
        $a_j = a_j^{new}$

3.2.4.3.6 Discussion The search and updated are related to the height of the tree, logarithmic in the number of reactions. So the total computational cost for each reaction event is $O(\log(M))$.

3.2.4.3.7 Tree with optimal height The computational cost for selecting the next reaction firing in a complete tree is not the optimal average-case performance. Let C be a tree structure.

3.2.4.3.7.1 Average number of comparison The average number of comparison performed during the search in tree C is:

$$T_m(C) = \sum_{j=1}^{M} w_j D_j$$

Where:

- M is the total number of reactions in the leaves.
- w_j is a weight related to the probability that reaction R_j is selected to fire.
- D_j is the depth of leaf R_j.

3.2.4.3.7.2 Complete tree When the tree C is complete the depth D_j is the same for all j. This leads to the fact that picking a fast reaction requires the same computational power of picking a slow one, leading to a non-optimal $T_M(C)$.

3.2.4.3.7.3 Huffman tree The minimization of $T_M(C)$ leads to the construction of the Huffman tree. The leaves in this type of tree with large propensity values will be closer than the leaves with small values. This is built by merging trees in a forest, populated initially by trees with one node. At each step the two trees with roots p and q having the smallest weight w_p and w_q are merged creating the new root pq with weight $w_{pq} = w_p + w_q$. The process stops when only one tree is found such that $D_{pq} + 1 = D_q = D_q$, where p, q, pq are the nodes involved in a merge such hat:

$$\begin{aligned} T_M(C) &= \sum_{\substack{j=1 \\ j\neq p,q}}^{M} w_j D_j + w_p D_q = \\ &= \left(\sum_{\substack{j=1 \\ j\neq p,q}}^{M} w_j D_j + w_{pq} D_{pq} \right) + w_{pq} = \\ &= T_{M-1}(C) + w_{pq} \end{aligned}$$

This derivation allows to determine that the Huffman tree gives the minimum value of $T_M(C)$. This is proven by induction on M. Consider the base case with $M = 2$, this is easy to check. By the inductive hypothesis, the Huffman tree for $M - 1$ gives the optimum value for $T_{M-1}(C)$. Suppose by contradiction that the Huffman tree for M is not optimal, so there is some tree having the total number of comparison $T'_M(C)$ such that $T'_M(C) < T_M(C)$. Without loss of generality the smallest weights are placed at the lowest level. Let p and q be the nodes with the smallest weights and label their parent pq. Using the derivation this gives:

$$T'_{M-1}(C) + w_{pq} < T_{M-1}(C) + w_{pq}$$

Then $T'_{M-1}(C) < T_{M-1}(C)$, contradicting the inductive hypothesis.

3.2.4.3.7.4 Building the Huffman tree To build a Huffman tree an array with size $2M-1$ is considered and each node has two children. However M does not need to be even. The leaves are between M and $2M - 1$. Each element in the array points to its left and right child and an additional field parent points to the parent of the node. To extract the nodes with minimal weights a binary heap is used, such that each element is (i, w_i), where i is the index of a node in the tree and the weight w_i is used as the key for ordering the heap. A heap is a tree-based data structure that satisfies the heap property: the key of a parent node is smaller than the key of its child nodes. The implementation for this procedure is presented in algorithm 11.

Algorithm 11: `build_huffman_tree(`*position*`)`

Input: an array TREE with $2M-1$ elements where elements from M to $2M-1$ are filled with M reaction propensities
Output: The Huffman tree represented by the array TREE.
build binary heap H with elements $(M, w_1), \ldots, (2M-1, w_M)$ ordered according to W_j
for *position* $= M-1$ **to** 1 **do**
    extract top element (p, w_p) from H
    extract top element (q, w_q) from H
    $TREE[position].VALUE = TREE[p].VALUE + TREE[q].VALUE$
    $TREE[position].LEFT = p$
    $TREE[position].RIGHT = q$
    `insert(`*position,* $w_p + w_q$`)` into H
    $TREE[p].PARENT = position$
    $TREE[q].PARENT = position$

The same binary search and propagation update are applied to search and update the propensities of the reactions.

3.2.4.3.7.5 Selecting the weight The weight function w_j can be the propensity function a_j because it allows to reduce the time spent to find the next reaction, however reaction firing could make the tree no longer optimal, so it should be rebuilt. To balance the expensive operation of re-building with the non-optimal tree, the non-optimal tree is used for some number of steps. The choice for this step number only affect performance and not exactness. There are two approaches to do so:

- Fixed time tree rebuilding.
- Adaptive time tree rebuilding.

3.2.4.3.7.6 Fixed time tree rebuilding In fixed time tree rebuilding the tree structure is built every k steps. To predict the changes in the reaction propensities during the k steps the weights can be modified assigning a higher weight to those reaction that are more likely to change.

Conflicts and Favours set For a reaction R_j define:

$$Conflicts(R_j) = \{R_k | (R_j \in Dependents(R_k)) \wedge (Reactants(R_k) \cap Reactants(R_j) \neq \emptyset)\}$$

$$Favours(R_j) = \{R_k | (R_j \in Dependents(R_k)) \wedge (Products(R_k) \cap Reactants(R_j) \neq \emptyset)\}$$

Dependency graph In terms of the dependency graph:

$$|Conflicts(R_j)| + |Favours(R_j)| = in\ degree\ of\ R_j$$

Estimating changes of propensity After a reaction firing, the probability that R_j will increase or decrease is estimated as $\frac{|Conflicts(R_j)|}{M}$ or $\frac{|Favours(R_j)|}{M}$. For k simulation steps, the estimated weight of reaction R_j is:

$$w_j(a_j, k) = a_j + \alpha_1 k \frac{|Favours(R_j)|}{M} + \alpha_2 k \frac{|Conflicts(R_j)|}{M}$$

Where α_1 and α_2 are parameters denoting the amount of average change.

3.2.4.3.7.7 Adaptive time tree rebuilding In adaptive time tree rebuilding the tree is rebuilt when a significant change has occurred. To detect the abrupt change in propensities a predefined value δ, the acceptance threshold defines the largest change which does not require an immediate tree rebuilding. The difference in propensity after a reaction R_j firing is $c_j = a_j^{new} - a_j$. If $c_j \geq \delta$ the Huffman tree should be rebuilt. To account for many small changes a cumulative sum of all the propensity changes s_j since the last rebuilt is computes and compared against the acceptance threshold to decide whether to rebuild the tree.

3.2.4.4 Direct method with composition-rejection search

The composition-rejection CR search employs the partitioning of reaction into groups, but the selection of the next reaction is performed through an acceptance-rejection sampling. The reactions are partitioned into L groups so that $R_j \in G_l$ if a_j satisfies $2^{u_l - 1} \leq a_j \leq 2^{u_l}$ in which u_l is selected such that $u_l = \lceil \log_2(a_j) \rceil$. If $a_{\min}$ and $a_{\max}$, the global minimum and maximum propensities, are known, then $L = \left\lceil \log_2 \left(\frac{a_{\max}}{a_{\min}} \right) \right\rceil$ for the whole simulation. These two bounds on a can be estimated through physical reasoning. Where this is not possible L must be increased during the simulation.

3.2.4.4.1 Search for the next reaction Let $a^l = \sum_{R_j \in G_l} a_j$ be the sum of the propensity in group G_l. The total propensity a_0 can be computed as:

$$a_0 = \sum_{l=1}^{L} a^l$$

The search is composed of two steps.

3.2.4.4.1.1 First step In the first step a group G_α is selected with probability $\frac{a^l}{a_0}$. This can be performed accumulating values a^l until the smallest index α is found such that:

$$\sum_{l=1}^{\alpha} a^l \geq r_1 a_0$$

The tree-based search can be applied to select the group.

3.2.4.4.1.2 Second step The second step is done through an acceptance-rejection sampling with the chosen envelope 2^{u_α}. A random and uniform reaction index $\mu \in G_\alpha$ is computed: $\mu = [r_2 |G_\alpha|]$, where $|G_\alpha|$ is the size of G_α and r_2 is a random number. The selected reaction is tested to accept with probability $\frac{a_\mu}{2^{u_\alpha}}$: a random number r_3 is generated and compared against $\frac{a_\mu}{2^{u_\alpha}}$. r_2 can be recycled noting that $r_3 = r_2 |G_\alpha| - \mu$ is uniformly distributed in $[0, 1]$. If $\frac{a_\mu}{2^{u_\alpha}} \leq r_3$ holds, R_μ is accepted to fire. Otherwise the reaction is rejected and a new random reaction index is generated and the check is performed again. This is repeated until there is an accepted R_μ. The acceptance probability is bound to $\frac{1}{2}$: $\frac{a_\mu}{2^{u_\alpha}} \geq \frac{1}{2}$ by definition.

3.2.4.4.2 Algorithm An implementation of this method is found in algorithm 12. The rejection test is repeated on average two times: the acceptance rate is bounded by $\frac{1}{2}$. Moreover after a reaction firing the propensity must be updated and it could be that a new reaction falls outside of the current bound, so it must be moved to an appropriate group.

Algorithm 12: `Direct Method with Composition-Rejection Search()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
build the dependency graph G
partition M reactions into L groups $\{G_1, \ldots, G_L\}$ such that $R_j \in G_l$ if $2^{u_l - 1} \leq a_j \leq 2^{u_l}$
$a_0 = 0$
foreach G_l **do**
    $a^l = 0$
    foreach $R_j \in G_l$ **do**
        compute a_j
        $a^l = a^l + a_j$
    $a_0 = a_0 + a^l$
while $t < T_{\max}$ **do**
    generate a random number $r_1 \sim norm(0, 1)$
    select G_α with the smalles group index α such that $\sum_{l=1}^{\alpha} a^l \geq r_1 a_0$
    repeat
        generate a random number $r_2 \sim norm(0, 1)$
        $\mu = [r_2 |G_\alpha|]$
        $r_2 = r_2 |G_\alpha| - \mu$
    until $r_2 \leq \frac{a_\mu}{2^{u_\alpha}}$
    generate a random number $r_3 \sim norm(0, 1)$
    $\tau = \frac{1}{a_0} \ln \frac{1}{r_2}$
    $\vec{X} = \vec{X} + \vec{v}_\mu$
    $t = t + \tau$
    foreach $R_j \in Dependents(R_\mu)$ **do**
        update a_j
        if $a_j \notin [2^{u_l - 1}, 2^{u_l}]$ **then**
            move R_j from G_l to an appropriate G_m
            updated a^l and a^m
        else
            update a^l
        update a_0

3.2.4.4.3 Discussion The selected base 2 in the partition can be chosen arbitrarily, with the dimension of groups and the number of rejections increasing with the base. Moreover efficient data structures are needed to implement the movement of reactions between groups: this should support dynamic memory allocation operation. In addition a hash table should be used to support fast lookup of a reaction in a group. For adding a reaction to a group, the group size is increased and

the reaction is added at the end of the group. When deleting a reaction, the reaction at the end of the group replaces it and the group size is decremented. After each of this two operations, the hash table is updated.

3.2.4.4.3.1 Computational cost The computational cost is composed by the search for the group, proportional to the number of groups $O(L)$ and the cost for selecting the next reaction, which is constant. Because the average number of rejection tests is bound by 2, the time is $O(L)$ and is independent of the number of reactions M. If $L \ll M$ and is bounded by a small constant the search for the next reaction firing is $O(1)$.

3.2.5 Partial-propensity direct method

The partial propensity direct method PDM requires that reactions must be elementary and their propensities follow the mass action kinetics. Mass action propensities are factorized and the partial propensities related to common reactants are grouped to facilitate the selection of the next reaction firing. Let π_j^i the partial propensity of a reaction R_j with respect to reactant S_i, this is defined as the propensity per molecule of reactant S_i.

3.2.5.1 Partial propensities of elementary reactions

- Synthesis reaction ($\emptyset \rightarrow$ products): propensity $a_j = c_j$ and partial propensity $\pi_j^0 = c_j$.
- Unimolecular reaction ($S_i \rightarrow$ products): propensity $a_j = c_j X_i$ and partial propensity $\pi_j^i = c_j$.
- Bimolecular reaction ($S_i + S_k \rightarrow$ products): propensity $a_j = c_j X_i X_k$ and partial propensity $\pi_j^i = c_j X_k$ and $\pi_j^{(k)} = c_j X_i$.
- Dimerization reaction ($2S_i \rightarrow$ products): propensity $a_j = \frac{1}{2} c_j X_i (X_i - 1)$ and partial propensity $\pi_j^i = \frac{1}{2} c_j (X_i - 1)$.

3.2.5.2 Storing the partial propensities

The partial propensities related to a species S_i are grouped into a group Π_i, such that the structure:

$$\Pi = \{\Pi_i\}_{i=0}^{N}$$

Stores all of them and is a matrix implemented as an array of arrays. It has $N + 1$ rows in which the i-th row stores the partial propensities related to species S_i, while the 0-th row stores all the partial propensities for synthesis reactions. In the case of a bimolecular reaction one of the two partial propensities has to be dropped.

3.2.5.2.1 Minimizing updates To minimize updates the partial propensities are stored with respect to the reactant involved in a larger number of reaction: before building Π, the species are re-indexed such that for each pair of them S_i and S_k, $i < k$ if the number of reactions involving S_i is greater than S_k. Then PDM stores partial propensity of a bimolecular reaction with respect to the reactant with smaller index.

3.2.5.3 Group-sum array

The sum $\Lambda_i = \Sigma_j \Pi_{i,j}$ gives the sum of partial propensities of reactions R_j sharing common reactant S_i. This is the group-sum array and is used to store the sum of partial propensities in group. $\Omega_i = X_i \Lambda_i$, where X_i is the population of S_i will be the sum of propensities of reaction having species S_i as the common reactant. The array $\Omega = \{\Omega_i\}_{i=0}^{N}$ to store the sum of propensities of groups. The total propensity is computed as:

$$a_0 = \sum_{i=0}^{N} \Omega_i$$

3.2.5.4 Reaction lookup

A reaction is identified by the group index i and the element j in a group Π_i. To facilitate the lookup of a reaction given the element j, a lookup table $\mathbf{L}$ is used to store the reaction indices of corresponding partial propensities in Π. It has the same structure as Π ans is implemented as an array of arrays. The index of reaction with element index j in group i of Π is identified as $\mathbf{L}_{ij}$, moreover three additional lookup table are used to facilitate the update of Π, Λ and Ω after a reaction firing:

- $\mathbf{U}^{(1)}$ is an array of M arrays in which array j contains the indices of species involved in R_j.
- $\mathbf{U}^{(2)}$ is an array of M arrays in which the array j contains the amount of change in population of the corresponding species in $\mathbf{U}^{(1)}$.
- $\mathbf{U}^{(3)}$ is an array of N arrays in which array k contains pairs of group indices and element indices of all entry in Π that depend on species k. Each element in the row k is a pair denoting the partial propensity $\Pi_{i,j}$ dependent in X_k.

3.2.5.5 Selecting the reaction firing

PDM selects R_μ in two steps. Let r_1 b a uniformly distributed random number in $norm(0, 1)$.

3.2.5.5.1 First step Search for the group index p, with $0 \leq p \leq N$:

$$p = \arg\min_{i \in p} \sum_{i=0}^{p} \Omega_i \geq r_1 a_0$$

3.2.5.5.2 Second step Search for an element index q, with $q \geq 1$ such that:

$$q = \arg\min_{i \in q} \left(X_p \sum_{j=1}^{q} \Pi_{p,j} + \sum_{i=0}^{p} \Omega_i - \Omega_p \right) \geq r_1 a_0$$

Or:

$$q = \arg\min_{i \in q} \Pi_{o,j} \geq \Psi$$

Where:

$$\Psi = \frac{r_1 a_0 - \sum_{i=0}^{p} \Omega_i + \Omega_p}{X_p}$$

Then p and q are used to retrieve the reaction firing index $\mu = \mathbf{L}_{p,q}$.

3.2.5.6 Exactness of PDM

The next reaction firing R_μ is selected by PDM having probability $\frac{a_\mu}{a_0}$. The selection of the reaction is performed by DM as:

$$\mu = \arg\min_{\mu \in j} \sum_{j=1}^{\mu} s_j \geq r_1 a_0$$

PDM identifies a reaction by a pair (p, q) where p is the group index and q the element index in Π by $\mu = \mathbf{L}_{p,q}$, then the equation can be re-written as:

$$(p, q) = \arg \min_{p \in i \wedge q \in j} \sum_{i=0}^{p} \sum_{j} a_{\mathbf{L}_{p,j}} \geq r_1 a_0$$

That can be broken down in:

$$p = \arg\min_{p \in i} \sum_{i=0}^{p} \sum_{j} a_{\mathbf{L}_{i,j}} \geq r_1 a_0$$

$$q = \arg\min_{q \in j} \sum_{i=0}^{p} \sum_{j} a_{\mathbf{L}_{i,j}} + \sum_{j=1}^{q} a_{\mathbf{L}_{p,j}} \geq r_1 a_0$$

Plugging into this two the definitions of Ω and Π they are equivalent to the one for the PDM seen before.

3.2.5.7 Algorithm

PDM is implemented as in algorithm 13.

Algorithm 13: `Partial-Propensity Direct Method (PDM)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$ with mass action kinetics
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
build Π, Λ, Ω and $\mathbf{L}$, $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$ and $\mathbf{U}^{(3)}$
$a_0 = 0$
foreach $i \in \Omega$ **do**
    $a_0 = a_0 + \Omega_i$
while $t < T_{\max}$ **do**
    generate two random numbers $r_1, r_2 \sim norm(0, 1)$
    select the smallest group index p such that $\sum_{i=0}^{p} \Omega_i \geq r_1 a_0$
    $\Psi = \frac{r_1 a_0 - \sum_{i=0}^{p} \Omega_i + \Omega_p}{X_p}$
    select R_μ with the smallest index q such that $\sum_{j=1}^{q} \Pi_{p,j} \geq \Psi$
    $\mu = \mathbf{L}_{p,q}$
    $\tau = \frac{1}{a_0} \ln \frac{1}{r_2}$
    $\Delta a = 0$
    foreach $k \in \mathbf{U}_\mu^{(1)}$ **do**
        $l = \mathbf{U}_{\mu,k}^{(1)}$
        $X_l = X_l + \mathbf{U}_{\mu,k}^{(2)}$
        foreach $m \in \mathbf{U}_l^{(3)}$ **do**
            $(i, j) = \mathbf{U}_{l,m}^{(3)}$
            $\mu' = \mathbf{L}_{i,j}$
            if $l \neq i$ **then**
                $\Pi_{i,j} = \Pi_{i,j} + c_{\mu'} \mathbf{U}_{\mu,k}^{(2)}$
                $\Lambda_i = \Lambda_i + c_{\mu'} \mathbf{U}_{\mu,k}^{(2)}$
            else if $l = i$ **then**
                $\Pi_{i,j} = \Pi_{i,j} + \frac{1}{2} c_{\mu'} \mathbf{U}_{\mu,k}^{(2)}$
                $\Lambda_i = \Lambda_i + \frac{1}{2} c_{\mu'} \mathbf{U}_{\mu,k}^{(2)}$
            $\Omega_{temp} = \Omega_i$
            $\Omega_i = X_i \Lambda_i$
            $\Delta a = \delta a + \Omega_i - \Omega_{temp}$
        $\Delta a = \Delta a + X_k \Lambda_l - \Omega_l$
        $\Omega_l = X_l \Lambda_l$
    $a_0 = a_0 + \Delta a$
    $t = t + \tau$

3.2.5.7.1 Discussion

3.2.5.7.1.1 Time complexity The time complexity of the search for the next reaction firing has two parts:

- Selecting the group, which in the worst case travels $N+1$ groups, in time $O(N)$.
- Selecting the element in the group, which is proportional to the number of reactions sharing the same reactant. This is model dependent and bounded by a small constant. For elementary reactions the worst case is N, so the cost for this step is $O(N)$.

In total the time complexity for the search for the next reaction firing in PDM is $O(N)$.

3.2.5.7.1.2 Limitations The major limitation of PDM is that it works for a class of reactions involving at most two reactants with factorisable reaction propensities. For models with high-order reactions the propensity is not factorizable and they must be broken down into elementary reactions and the propensity computation has to be modified.

3.2.5.8 Partial propensity direct method with composition-rejection search

The partial propensity direct method with composition-rejection search PDM-CR is a variant of PDM where the selection of p and q are done through the composition-rejection approach. Ω is grouped into L groups such that G_l stores group index i such that $2^{u_l-1} \leq \Omega_i < 2^{u_l}$ with $u_l = \lceil \log_2(\Omega_i) \rceil$. The sum of propensities in G_l is:

$$a^l = \sum_{i \in G_l} \Omega_i$$

For Π each i row is partitioned into K_i groups such that Q_k^i stores element index j such that $2^{v_k^i-1} \leq \Pi_{i,j} < 2^{v_k^i}$ with $v_k^i = \lceil \log_2(\Pi_{i,j}) \rceil$. The sum of partial propensity in Q_k^i is:

$$b_i^k = \sum_{j \in Q_k^i} \Pi_{i,j}$$

For each row of Π the relation $\sum_{k=1}^{K_i} b_i^k = \sum_j \Pi_{i,j} = \Lambda_i$ holds.

3.2.5.8.1 Selection of the next reaction firing The selection of the next R_μ consists of two CR searches: the first selects the group index p and the second the element index q.

3.2.5.8.1.1 Selecting group index To select the group index p two random numbers $r_1, r_2 \sim norm(0,1)$ are sampled. Then a group G_α is selected with probability $\frac{a^\alpha}{a_0}$ with $a_0 = \sum_{l=1}^{L} a^l = \sum_{i=0}^{N} \Omega_i$ accumulating a^l until the smallest α is found such that:

$$\sum l = 1^\alpha a^l \geq r_1 a_0$$

Then r_2 is used to accept the group index p in G_α through an acceptance-rejection test with probability $\frac{\Omega_p}{2^{u_\alpha}}$.

3.2.5.8.1.2 Selecting element index Once p has been selected q is selected through a second CR search. This requires two random numbers $r_3, r_4 \sim norm(0,1)$. A group Q_β^p is selected with probability $\frac{b_p^\beta}{\Lambda_p}$ by a linear search. Then the element index q in the groups is selected through an acceptance-rejection test with probability $\frac{\Pi_{p,q}}{2^{v_\beta^p}}$.

3.2.5.8.2 Algorithm The PDM-CR implementation is outlined in algorithm 14.

Algorithm 14: `Partial-Propensity Direct Method with Composition-Rejection Search (PDM-CR)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$ with mass action kinetics
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
build Π, Λ, Ω and $\mathbf{L}$, $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$ and $\mathbf{U}^{(3)}$
partition Ω into L groups $G_1, \ldots, G_L$ such that G_l contains Ω_l if $2^{u_l - 1} \leq \Omega_i < 2^{u_l}$
$a_0 = 0$
foreach $G_l \in \{G_1, \ldots, G_L\}$ **do**
    $a^l = \sum_{i \in G_l} \Omega_i$
    $a_0 = a_0 + a^l$
for $i = 0 \Rightarrow N$ **do**
    partition Π_i into K_i groups $Q_1^i, \ldots, Q_{K_i}^i$ such that Q_k^i contains $\Pi_{i,j}$ if $2^{v_k^i - 1} \leq \Pi_{i,j} < 2^{v_k^i}$
    for $i = 1 \rightarrow K_i$ **do**
        $b_i^k = \sum_{\Pi_{i,j} \in Q_k^i} \Pi_{i,j}$
while $t < T_{\max}$ **do**
    generate a random number $r_1 \sim norm(0,1)$
    select the smallest group G_α such that $\sum_{l=1}^{\alpha} a^l \geq r_1 a_0$
    $\Psi = \frac{r_1 a_0 - \sum_{i=0}^{p} \Omega_i + \Omega_p}{X_p}$
    select R_μ with the smallest index q such that $\sum_{j=1}^{q} \Pi_{p,j} \geq \Psi$
    repeat
        generate a random number $r_2 \sim norm(0,1)$
        $p = [r_2 |G_\alpha|]$
        $r_2 = r_2 |G_\alpha| - p$
    until $r_2 < \frac{\Omega_p}{2^{u_\alpha}}$
    generate a random number $r_3 \sim norm(0,1)$
    select the smallest group Q_β^p such that $\sum_{k=1}^{\beta} b_p^k \geq r_3 \Lambda_p$

Contd-While
  repeat
    generate a random number $r_4 \sim norm(0,1)$;
    $q = [r_4|Q^p_\beta|]$;
    $r_4 = r_4|Q^p_\beta| - q$;
  until $r_4 < \frac{\Pi_{p,q}}{2^{v^p_\beta}}$;
  $\mu = \mathbf{L}_{p,q}$;
  $\tau = \frac{1}{a_0} \ln \frac{1}{r_2}$;
  $\Delta a = 0$;
  foreach $k \in \mathbf{U}^{(1)}_\mu$ **do**
    $l = \mathbf{U}^{(1)}_{\mu,k}$;
    $X_l = X_l + \mathbf{U}^{(2)}_{\mu,k}$;
    foreach $m \in \mathbf{U}^{(3)}_l$ **do**
      $(i,j) = \mathbf{U}^{(3)}_{l,m}$;
      $\mu' = \mathbf{L}_{i,j}$;
      if $l \neq i$ **then**
        $\Pi_{i,j} = \Pi_{i,j} + c_{\mu'}\mathbf{U}^{(2)}_{\mu,k}$;
        $\Lambda_i = \Lambda_i + c_{\mu'}\mathbf{U}^{(2)}_{\mu,k}$;
      else if $l = i$ **then**
        $\Pi_{i,j} = \Pi_{i,j} + \frac{1}{2}c_{\mu'}\mathbf{U}^{(2)}_{\mu,k}$;
        $\Lambda_i = \Lambda_i + \frac{1}{2}c_{\mu'}\mathbf{U}^{(2)}_{\mu,k}$;
      $\Omega_{temp} = \Omega_i$;
      $\Omega_i = X_i\Lambda_i$;
      $\Delta a = \delta a + \Omega_i - \Omega_{temp}$;
      updated group G_i and Q^i_j;
    $\Delta a = \Delta a + X_k\Lambda_l - \Omega_l$;
    $\Omega_l = X_l\Lambda_l$;
  updated group G_l;
  $a_0 = a_0 + \Delta a$;
  $t = t + \tau$;

3.3 First reaction method

The first reaction method FRM is an alternative method to implement the Monte Carlo step of SSA. The next reaction firing R_μ and firing time τ are exact, so they are ensured to be distributed by the pdf $p(\tau, \mu|\vec{x}, t)$.

3.3.1 Tentative time

In FRM the reaction selected to fire is the one with the smallest tentative time. The tentative time is the firing time of the reaction assuming that no other reaction fires before. Let τ_j be the tentative

time to the firing of R_j assuming that no other reaction fires before. Let $p(\tau_j|\vec{x},t)$ be the pdf of τ_j such that $p(\tau_j|\vec{x},t)d\tau_j$ gives the probability that R_j fires in the next infinitesimal time interval $[t+\tau_j, t+\tau_j+d\tau_j[$ assuming that no other reaction fires before. The formula of $p(\tau_j|\vec{x},t)$, noting that there is only one R_j involved in the calculation is:

$$p(\tau_j|\vec{x},t) = a_j e^{-a_j\tau_j}$$

From this the tentative time can be generated applying the inverse transforming method:

$$\tau_j = \frac{1}{a_j}\ln\left(\frac{1}{r_j}\right)$$

Where $r_j \sim norm(0,1)$.

3.3.2 Exactness of the first reaction method

Let R_μ be the reaction having the smallest tentative time $\tau = \min_{j=1}^{M}\{\tau_j\}$ where each τ_j with $j = 1,\ldots,M$ is distributed according to the pdf $p(\tau_j|\vec{x},t) = a_j e^{-a_j\tau_j}$. Let $\tilde{p}(\tau,\mu|\vec{x},t)d\tau$ be the probability that R_μ fires at time τ, then:

$$\tilde{p}(\tau,\mu|\vec{x},t) = a_\mu e^{-a_0\tau}$$

The probability of R_μ which has the smallest time $\tau = \min_{j=1}^{M}\{\tau_j\}$ to fire at time τ is:

$$\tilde{p}(\tau,\mu|\vec{x},t)d\tau = \mathbb{P}\{\tau < \tau_\mu < \tau + d\tau\}\mathbb{P}\{\tau_j > \tau \forall j \neq \mu\}$$

Where:

- $\mathbb{P}\{\tau < \tau_\mu < \tau + d\tau\}$ is the probability that R_μ with smallest tentative fire τ_μ fires in $[\tau, \tau + d\tau[$.
- $\mathbb{P}\{\tau_j > \tau \forall j \neq \mu\}$ is the probability that τ_j of R_j is greater than τ.

The first probability is directly derived from the definition of τ_μ as:

$$\mathbb{P}\{\tau < \tau_\mu < \tau + d\tau\} = a_\mu e^{-a_\mu\tau}d\tau$$

The second one is derived as follows:

$$\begin{aligned}
\mathbb{P}\{\tau_j < \tau \forall j \neq \mu\} &= \mathbb{P}\left\{\frac{1}{a_j}\ln\left(\frac{1}{r_j}\right) \forall i \neq \mu\right\} = \\
&= \mathbb{P}\{r_j < e^{-a_j\tau} \forall j \neq \mu\} = \\
&= \prod_{i=1 \wedge j \neq \mu}^{M} \mathbb{P}\{r_j < e^{-a_j\tau}\} = \\
&= \prod_{i=1 \wedge j \neq \mu}^{M} e^{-a_j\tau}
\end{aligned}$$

In which the generation of τ_j is considered, and the third step follows from the fact that r_js are all independent and identically distributed random numbers. The last holds because the probability that a uniformly distribute random number from a unit interval is less than a number is equal to that number. Now, plugging the last to into $\tilde{p}$ and recalling the definition of total propensity, the probability distribution of the next reaction firing is:

$$\tilde{p}(\tau, \mu|\vec{x}, t) = a_\mu e^{-a_\mu \tau} \left(\prod_{j=1 \wedge j \neq \mu} e^{-a_j \tau} \right) = a_\mu e^{-a_0 \tau}$$

3.3.3 Algorithm

An implementation of the steps of FRM is presented in algorithm 15. For each simulation iteration M uniformly distributed random numbers r_j are generated and used to compute the tentative time τ_j for all reactions. Then the reaction with the smallest time is selected to fire. Once the reaction has fired, the time and state are updated accordingly.

Algorithm 15: `First Reaction Method (FRM)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
while $t < T_{\max}$ **do**
    foreach R_j **do**
        compute a_j
        generate a random number $r_j \sim norm(0, 1)$
        $\tau_j = \frac{1}{a_j} \ln\left(\frac{1}{r_j}\right)$
    select R_μ with the smallest tentative time $\tau = \min_{j=1}^{M}\{\tau_j\}$
    $\vec{X} = \vec{X} + \vec{v}_\mu$
    $t = t + \tau$

3.3.3.1 Time complexity

The time complexity is due to:

- Computing the tentative time of reactions. This, for M reaction, is done in $O(M)$ time.
- Searching for the reaction with the smallest tentative time. This is also $O(M)$ as they can be linearly compared as they are generated.

The time complexity is thus $O(M)$ for a time step and this algorithm is easy to parallelize. However it is often slower than DM because a large number of random numbers are required at each iterations. FRM is therefore slower than DM if the number of reactions $M \geq 2$.

3.3.4 First family method

The first family method FFM is a generalization of DM and FRM methods. It partitions M reactions into L families. Each F_l contains k_l reactions, that can be different between families. L and k_l are tunable parameters. Note that because the partition is complete $\sum_{l=1}^{L} k_l = M$.

3.3.4.1 Next reaction event

The next reaction event in FFM is a pair (α, μ) denoting R_μ in family F_α. The selection is performed in two steps:

- Selection of the family.
- Selection of the reaction.

3.3.4.1.1 Selection of the family The selection is performed in two steps: a family with the smallest tentative time is selected. The tentative time of F_l is generated from an exponential distribution with rate equal to the sum of reaction propensities in the family. Let $a^l = \sum_{R_j \in F_l} a_j$ be the sum of propensities, then the tentative time will be computed as:

$$\tau_l = \frac{1}{a^l} \ln\left(\frac{1}{r_l}\right)$$

Where $r_l \sim norm(0, 1)$.

3.3.4.1.2 Selection of the reaction Let F_α be the family with the smallest tentative time: $\tau = \min_{l=1}^{L}\{\tau_l\}$. Conditioning on the selected F_α, R_μ is selected with probability $\frac{a_\mu}{a^l}$. A DM search is applied to find the next reaction:

$$\mu = \arg\min_{k \in F_\alpha} \sum_{\substack{k=j \\ F_\alpha = \{R_j, \ldots, R_{j+k\alpha}\}}}^{\mu} a_k \geq r a^\alpha$$

Where $r \sim norm(0, 1)$.

3.3.4.2 Algorithm

An implementation of the FFM is presented in algorithm 16. For each simulation step, L random number are required to compute the tentative times and an additional one is used to select the next reaction firing.

Algorithm 16: `First Family Method (FFM)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
partition M reactions into L families $\{F_1, \ldots, F_L\}$
while $t < T_{\max}$ **do**
    foreach F_l **do**
        $a^l = 0$
        foreach R_j **do**
            compute a_j
            $a^l = a^l + a_j$
        generate a random number $r_l \sim norm(0, 1)$
        $\tau_l = \frac{1}{a^l} \ln\left(\frac{1}{r_l}\right)$
    select F_α with smallest tentative time $\tau = \min_{l=1}^{L}\{\tau_l\}$
    generate a random number $r \sim norm(0, 1)$
    select $R_\mu \in F_\alpha$ with the smallest index μ such that $\sum_{\substack{k=j \\ F_\alpha = \{R_j, \ldots, R_{j+k\alpha}\}}}^{\mu} a_k \geq r a^\alpha$
    $\vec{X} = \vec{X} + \vec{v}_\mu$
    $t = t + \tau$

3.3.4.2.1 Discussion

3.3.4.2.1.1 Performance It can be seen how $L + 1$ random number are generated by FFM, so it has better performance than FRM when the number of reactions $M \gg L$ thanks to the smaller number of random number generation. This algorithm is still parallelizable for the families.

3.3.4.2.1.2 Comparison with DM and FRM DM and FRM are special cases of FFM obtained by tuning L and k_l. If $L = 1$ the algorithm is reduced to the DM. If $L = M$ and $k_l = 1$ the algorithm reduces to FRM. Great attention has to be posed to the propensity of the families: if one is much greater than the other the algorithm could choose only that family.

3.4 Next reaction method

The next reaction method NRM improves FRM:

- Avoids recomputing propensities of all reaction after a firing. It recomputes the propensity a_j of R_j only if it actually changes extracting them from a reaction dependency graph G.

- Switches to absolute tentative time instead of relative and reuses the time when appropriate. For each simulation step it generates the new time for the reaction firing, while the other are updated and reused, reducing the number of random number used in the simulation.
- it employs efficient data structures to store and retrieve putative firing time of reaction, making the selection of the next reaction fast and efficient.

3.4.1 Absolute tentative time

Let τ_j be the tentative time to the firing of R_j with pdf $p(\tau_j|\vec{x},t) = a_j e^{-a_j\tau_j}$. Let $\tau_m = \min_{k=1}^{M} \tau_j$. The residual $\tau_j - \tau_m$ is transformed for all $j \neq \mu$ to compute the new tentative time for R_j.

3.4.1.1 Distribution of the residual time

$\tau_j - \tau_\mu$ is exponentially distributed with rate a_j: let X be a random variable with an exponential density function:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Where $\lambda > 0$ is a parameter, then, $\forall s > t \geq 0$:

$$\mathbb{P}\{X > s | X > t\} = \mathbb{P}\{X > s - t\}$$

Expanding the left-hand side:

$$\begin{aligned} \mathbb{P}\{X > s | X > t\} = \frac{\mathbb{P}\{X > s \wedge X > t\}}{\mathbb{P}\{X > t\}} = \frac{\mathbb{P}\{X > s\}}{\mathbb{P}\{X > t\}} = \frac{1 - \mathbb{P}\{X \leq s\}}{1 - \mathbb{P}\{X \leq t\}} = \\ = \frac{1 - \int_{-\infty}^{s} f(x)dx}{1 - \int_{-\infty}^{t} f(x)dx} = \frac{1 - \int_{0}^{s} \lambda e^{-\lambda x}dx}{1 - \int_{0}^{t} \lambda e^{-\lambda x}dx} = \frac{e^{-\lambda s}}{e^{-\lambda t}} = \\ = e^{-\lambda(s-t)} \end{aligned}$$

And the right-hand side:

$$\begin{aligned} \mathbb{P}\{X > s - t\} = 1 - \mathbb{P}\{X \leq s - t\} = 1 - \int_{-\infty}^{s-t} f(x)dx = \\ = 1 - \int_{0}^{s-t} \lambda e^{-\lambda x}dx = e^{-\lambda(s-t)} \end{aligned}$$

Obtaining the required equality.

3.4.1.2 Relationship between absolute and relative tentative time

Let t_j the absolute tentative time: the time from the start of the simulation to the firing of R_j. The relationship between the absolute and tentative time is:

$$t_j = t + \tau + j$$

Where t is the current simulation time. The reaction with the smallest absolute time is the reaction with the smallest relative time because t is fixed. However using it NRM does not need the random number necessary to generate the new tentative times.

3.4.1.3 Computing the absolute time

Let R_μ the reaction with smallest t_μ. After it fires and t is advanced to t_μ, the new times for reactions have to be generated. For R_μ a new tentative time is generated from an exponential distribution $t_\mu^{new} = e^{a_\mu^{new}}$ and the absolute time is updated to:

$$t_\mu^{new} + t_\mu + \tau_\mu^{new}$$

For each R_j with $j \neq \mu$, let a_j^{new} and τ_j^{new} be the new propensity value and new relative time. In the case that R_j does not depend on R_μ and the propensity of the reaction does not change by firing, the difference $\tau_j - \tau_m = t_j - t_\mu$ can be used as the new relative tentative time τ_j^{new} of the reaction, so that the absolute time does not change:

$$t_j^{new} = \tau_j^{new} + t_\mu = t_j - t_\mu + t_\mu = t_j$$

If $R_j \in Depenents(R_\mu)$ and $j \neq \mu$, the propensity a_j changes to a_j^{new}. A new relative tentative time, an exponential random number with rate a_j^{new} has to be computed.

3.4.1.3.1 Computing the new tentative time Let X be a random variable with exponential density function:

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Where $\lambda > 0$ is a parameter. Let Y be a random variable such that $Y = cX$, where $c > 0$ is a constant, then the probability density function of Y is:

$$f_Y(x) = \begin{cases} \frac{\lambda}{x} e^{-\frac{\lambda}{c}x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Let $F_Y(x)$ be the cdf of Y, then:

$$f_Y(x) = \frac{dF_y(x)}{dx} = \frac{d\mathbb{P}\{Y \leq x\}}{dx} = \frac{d\mathbb{P}\{cX \leq x\}}{dx} = \frac{d\mathbb{P}\left\{X \leq \frac{x}{c}\right\}}{dx} = \\ = \frac{d\left(\int_{-\infty}^{\frac{x}{c}} f_X(s)ds\right)}{dx}$$

If $x < 0$, then $f_Y(x) = 0$. If $x \geq 0$:

$$f_Y(x) = \frac{d\left(\int_0^{\frac{x}{c}} \lambda e^{-\lambda s} ds\right)}{dx} = \frac{d\left(1 - e^{-\frac{\lambda}{c}x}\right)}{dx} = \\ = \frac{\lambda}{c} e^{-\frac{\lambda}{c}}$$

This ensures that:

$$\tau_j^{new} = \frac{a_j}{a_j^{new}}(t_j - t_\mu)$$

Is exponentially distributed with rate a_j^{new} as desired, so the new value for the absolute tentative time for R_j is:

$$t_j^{new} = \tau_j^{new} + t_\mu = \frac{a_j}{a_j^{new}}(t_j - t_\mu) + t_\mu$$

3.4.1.4 Conclusion

In conclusion NRM does not generate new random numbers to compute the new times for all the reactions: the old time is reused to compute the new one.

3.4.2 Data structures

To speed up the selection of the minimum t_μ, NRM employs a binary heap to index the absolute putative times t_j of reactions R_j. Each node is a pair (t_j, R_j) in which t_j is a key that prioritize the node. It maintains a partial order between nodes: the parent have a smaller time than its child. The selection of the reaction with smallest time is constant: it is on the top of the heap. The heap is has to be updated for each reaction whose time is changed. To do this the node containing the reaction is updated with the new time and is swept up and down to maintaining the order of the heap. This costs $O(\log(M))$.

3.4.3 Algorithm

An implementation of the next reaction method is presented in algorithm 17.

Algorithm 17: `Next Reaction Method (NRM)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
build the reaction dependency graph G
foreach R_j **do**
    compute a_j
    generate a random number $r_j \sim norm(0, 1)$
    $t_j = \frac{1}{a_j} \ln\left(\frac{1}{r_j}\right)$
build the binary heap H for M tentative times t_j
while $t < T_{\max}$ **do**
    extract the node with smallest t_μ and reaction R_μ from H
    $t = t_\mu$
    $\vec{X} = \vec{X} + \vec{v}_\mu$
    foreach $R_j \in Dependents(R_\mu)$ **do**
        compute a_j^{new}
        if $j \neq \mu$ **then**
            $t_j = \frac{a_j}{a_j^{new}}(t_j - t) + t$
        else if $j = \mu$ **then**
            generate a random number $r \sim norm(0, 1)$
            $t_\mu = t + \frac{1}{a_\mu^{new}} \ln \frac{1}{r}$
        $a_j = a_j^{new}$
        replace the old time t_j in H with the new value t_j^{new} and maintain the heap H

3.4.3.1 Discussion

3.4.3.1.1 Time complexity The computation cost of NRM scales as the logarithm of M. For each simulation step, the extraction of the smallest time, advancing the simulation to the new time and updating the state are constant. The update cost in the heap for the time dominates the total cost. It iterates over all the dependent reaction to calculate their new times and perform the heap update, for one reaction this costs $O(\log(M))$. Let D be the average number of reaction that are dependent on the reaction firing, then the total cost of update is $O(D \log(M))$. If D is small and is bounded by a constant, then the total cost is $O(\log(M))$. Moreover only one random number is necessary for each simulation step, so the number of random numbers used by NRM is optimal.

3.4.3.1.2 Inactive reaction When a reaction is inactive $a_j = 0$ before firing R_μ and becomes active after the reaction, the new time of R_j will be $t_j^{new} = t_\mu$, which is not possible. A solution for the implementation of this step is to generate a new putative time τ_j^{new} by sampling $e^{a_j^{new}}$ rather than applying the transformation.

3.4.4 Modified next reaction method

The modified next reaction method MNRM is a variant of NRM where the firing time of reaction are represented as independent Poisson processes with rates given by their integrated propensities.

3.4.4.1 Poisson process

Let $Y(t), t \geq 0$ be a process that counts the number of events by time t. $Y(t)$ is a Poisson process with rate $\lambda > 0$ if:

- $Y(0) = 0$.
- $Y(t)$ has the stationary increment property: $\forall [t, t+\Delta t], Y(t) - Y(t+\Delta t)$ has the same distribution as $Y(\Delta t)$.
- $Y(t)$ has the independent increment property: $\forall [t, t+\Delta t] \wedge [t', t'+\Delta t']$ such that $[t, t+\Delta t] \cap [t', t'+\Delta t'] = \emptyset$, $Y(\Delta t)$ is independent of $Y(\Delta t')$.
- The probability of observing one event in the infinitesimal $[t, t+dt[$ is $\mathbb{P}\{Y(t+dt) - Y(t) = 1\} = \lambda dt + o(dt)$

Now let $Y(t)$ be a Poisson process with rate λ, then:

- The distribution $Y(t), t \geq 0$ is a Poisson distribution $poi(\lambda t)$.
- The time to the next event of the Poisson process is an exponential distribution $exp(\lambda)$.

3.4.4.1.1 $Y(t)$ is a Poisson distribution To prove the first claim let $\mathbb{P}\{Y(t) = k\}$ be the probability that there are k events in $[0, t]$. Suppose that this is divided into intervals $[\frac{(i-1)t}{n}, \frac{it}{n}]$ of length $\frac{t}{n}$ such that there is at most one event in each subinterval. The number of events in $[0, t]$ is the sum of the events observed in the subintervals. By definition of $Y(t)$, the probability of observing an event in the subinterval is $\frac{\lambda t}{n}$. The probability $\mathbb{P}\{Y(t) = k\}$ follows a binomial distribution with success probability $\frac{\lambda t}{n}$:

$$\mathbb{P}\{Y(t) = k\} = \binom{n}{k} \left(\frac{\lambda t}{n}\right)^k \left(1 - \frac{\lambda i}{n}\right)^{n-k} = \frac{n!}{k!(n-k)!} \left(\frac{\lambda t}{n}\right)^k \left(1 - \frac{\lambda i}{n}\right)^{n-k}$$

Expanding the factorial and taking the limit $n \to \infty$:

$$\mathbb{P}\{Y(t) = k\} = \lim_{n\to\infty} \frac{n}{n}\frac{n-1}{n} \cdots \frac{n-k-1}{n} \left(1 - \frac{\lambda t}{n}\right)^{-k} \frac{(\lambda t)^k}{k!} \left(1 - \frac{\lambda t}{n}\right)^n = \\ = \frac{(\lambda t)^k e^{-\lambda t}}{k!}$$

Considering that $\lim_{n\to\infty} \left(1 - \frac{\lambda t}{n}\right)^n = e^{-\lambda t}$. In conclusion then the probability $\mathbb{P}\{Y(t) = k\}$ denotes a Poisson distribution $poi(\lambda t)$.

3.4.4.1.2 The time to the next event is an exponential distribution Let T be the time to the next event of the Poisson process $Y(t)$. By definition of the Poisson process, it only needs to consider the time to the first event after 0. Let F_T be the cdf of T:

$$F_T(t) = \mathbb{P}\{T \leq t\} = \mathbb{P}\{Y(t) \geq 1\} = 1 - \mathbb{P}\{Y(t) = 0\} = 1 - \frac{(\lambda t)^0 e^{-\lambda t}}{0!} = 1 - e^{-\lambda t}$$

Considering that $\mathbb{P}\{Y(t) \geq 1\} = \mathbb{P}\{T \leq t\}$. This shows that the time T to the next event follows an exponential distribution $exp(\lambda)$.

3.4.4.2 Unit Poisson process

The Poisson process with rate 1 is called a unit Poisson process. If $Y(t)$ denotes a unit Poisson process, then $T(\lambda t)$ is a Poisson process with rate λ.

3.4.4.3 Random time change representation

Let $C_j(t)$ be the number of times that R_j fires up to t. $C(t)$ satisfies the conditions of the Poisson process. The probability that R_k fires in $[t, t + dt[$ by definition of a_j is:

$$\mathbb{P}\{C_j(t + dt) - C_j(t) = 1 | X(s), s \leq t\} = a_j(X(t))dt + o(dt)$$

$C_j(t)$ denotes a Poisson process with rate $a_j(X(T))$. Let $Y_j(t)$ be an independent unit Poisson process. $C_j(t)$ is represented in MNRM in term of $Y_j(t)$. It can be noted how:

$$C_j(t) = Y_j\left(\int_0^t a_j(X(s))ds\right)$$

Or the random time change RTC representation.

3.4.4.4 Internal time

The internal time I_j of Y_k associated with R_j is:

$$I_j(t) = \int_0^t a_j(X(s))ds$$

$I_j(t)$ given t shows the amount of time that Y_k passed before it expires due to the firing of R_j. Each reaction can be seen as carrying its own internal clock, with a rate given by the integration of the propensity. There are $M + 1$ time frames in which the first is the physical time t and the last M are for M Poisson processes Y_j. The internal time is a dimensionless quantity.

3.4.4.4.1 Computing the tentative time Let t be the current time, the system state is $X(t) = \vec{x}$. The propensity a_j and the internal time is $T_j = I_j(t)$. The internal time at time $t = 0$ is $I_j(0) = 0$. Let P_j be the next internal event time of Y_j with corresponding physical time $t_j > t$, $P_j = I_j(t)$. The relationship between T_j and P_j is:

$$P_j = I_j(t_j) = I_j(t) + a_j(t_j - t) = T_j + a_j\tau_j$$

Where $\tau_j = t_j - t$ is the relative time to the firing of R_j. The amount of internal time to the firing is $a_j\tau_j$. Because Y_j is a unit Poisson process, the time to the next firing follows a distribution $exp(1)$: the amount $P_j - T_j$ is an exponentially distributed random number with rate 1. If the current internal time T_j and the next internal time P_j are tracked, the tentative time to the firing, given that no other reaction fires before is:

$$\tau_i = \frac{P_j - T_j}{a_j}$$

The selection with minimum τ_μ will be selected to fire.

3.4.4.4.2 Updating the firing time Suppose that the reaction R_μ fires at time $t_\mu = t+\tau_\mu$. The next internal event time P_j of Y_μ must be generated because it expired, the new time is computed as:

$$\tau_i = \frac{P_j - T_j}{a_j}$$

For reaction R_j, with $j \neq \mu$, the updated internal time of process Y_j at time $t + \tau_\mu$ is:

$$I_j(t + \tau_\mu) = I_j(t) + a_j\tau_\mu$$

Let a_j^{new} be the new propensity and τ_j^{new} be the new tentative time to firing R_j after the firing of R_μ. $a_j^{new}\tau_j^{new}$ is the remaining amount of internal time to the next firing of Y_j. Because the processes are independent, the next internal event time P_j of Y_j does not change:

$$P_j = I_j(t + \tau_\mu) + a_j^{new}\tau_j^{new} = I_j(t) + a_j\tau_\mu + a_j^{new}\tau_j^{new}$$

Comparing it with the other computation of P_j:

$$\tau_j^{new} = \frac{a_j}{a_j^{new}}(\tau_j - \tau_\mu) = \frac{a_j}{a_j^{new}}(t_j - t_\mu)$$

Then the absolute time to the next firing of R_j:

$$t_j^{new} = t_\mu + \tau_j^{new} = \frac{a_j}{a_j^{new}}(\tau_j - \tau_\mu) + t_\mu$$

Which is the transformation used by NRM: the selection of the next reaction firing is exact.

3.4.4.5 Algorithm

An implementation of MNRM is outlined in algorithm 18.

Algorithm 18: `Modified Next Reaction Method (MNRM)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
build the reaction dependency graph G
foreach R_j **do**
    $T_j = 0$
    generate a random number $r_j \sim norm(0, 1)$
    $P_j = \ln \frac{1}{r_j}$
    compute a_j
while $t < T_{\max}$ **do**
    foreach R_j **do**
        $\tau_j = \frac{1}{a_j}(P_j - T_j)$
    select R_μ with the smallest time $\tau = \min_{j=1}^{M}\{\tau_j\}$
    $\vec{X} = \vec{X} + \vec{v}_\mu$
    $t = t + \tau$
    foreach R_j **do**
        $T_j = T_j + a_j\tau$
        if $j = \mu$ **then**
            generate a random number $r \sim norm(0, 1)$
            $P_\mu = P_\mu + \ln \frac{1}{r}$
        if $R_j \in Dependents(R_\mu)$ **then**
            compute new a_j

3.4.4.5.1 Discussion The simulation of MNRM is equivalent to NRM. MNRM explicitly works with internal time arising from the RTC representation, while NRM works with physical time. This makes MNRM more flexible to handle complex propensity function and helps make a smooth connection between exact stochastic simulation and the class of Poisson approximation, like the τ-leaping algorithm.

3.5 Rejection Based stochastic simulation algorithm

The rejection based stochastic simulation algorithm or RSSA is an exact simulation algorithm that tries to reduce the number of propensity updates during the simulation. Each simulation iteration selects a R_μ with probability $\frac{a_\mu}{a_0}$ and its firing time is exponentially distributed according to a_0. The selection of the next reaction firing in RSSA is an acceptance-rejection sampling technique that allows to skip the propensity updates in most of the iterations. They are recomputed only when necessary. Because of this it is useful for reaction in which propensities are complex.

3.5.1 Fluctuation interval

For each species S_i RSSA abstracts its exact population $X_i(t)$ with a fluctuation interval $[\underline{X_i}, \overline{X_i}]$. The interval can be chosen arbitrarily around the current population without affecting the correctness of the algorithm. The fluctuation interval can be defined as:

$$[\underline{X_i}, \overline{X_i}] = [(1 - \delta_i)X_i(t), (1 + \delta_i)X_i(t)]$$

Where δ_i is the parameter called fluctuation rate. A good choice for real biological model is from 10 to 20% of current population of species. Considering abstraction interpretation terminology, $X(t)$ is called the concrete state, while $[\underline{X}, \overline{X}]$ is the abstract state. For each species holds:

$$\underline{X} \leq X(t) \leq \overline{X}$$

3.5.2 Abstract propensity value

For each R_j an abstract propensity value $[\underline{a_j}, \overline{a_j}]$ is computed, this is an interval encompassing all its possible values, including the exact one $a_j(X(t))$. The bounds are derived by minimizing and maximizing a_j over the fluctuation interval. For the standard mass action or Michaelis-Menten kinetics, a_j is a monotonic function of the state X, so that $\underline{a_j} = a_j(\underline{X})$ and $\overline{a_j} = a_j(\overline{X})$. If it is non-monotonic, a numerical optimization technique or interval analysis can be applied to recover the bounds. The exact bounds are not needed as the tight bounds of a_j over the fluctuation interval are sufficient.

3.5.3 Selection of the next reaction

The selection of the next reaction has two steps:

- Simulation of the abstract model.
- Acceptance of the candidate reaction.

3.5.3.1 Simulation of the abstract model

In the first step RSSA simulates the abstract model assigning to each R_j $\frac{\overline{a_j}}{\overline{a_0}}$, where $\overline{a_0} = \sum_{j=1}^{M} \overline{a_j}$. R_μ is randomly selected with probability $\frac{\overline{a_\mu}}{\overline{a_0}}$ as a candidate. The realization of R_μ is performed accumulating propensity upper bounds until the smallest μ is selected such that:

$$\sum_{j=1}^{\mu} \overline{a_j} \geq r_1 \overline{a_0}$$

Where $r_1 \sim norm(0, 1)$.

3.5.3.2 Acceptance of the abstract model

In the second step RSSA checks whether R_μ is accepted to occur in the concrete model through a rejection test with success probability $\frac{a_\mu}{\overline{a_\mu}}$. Since the exact value of the propensity is not known, a random number $r_2 \sim norm(0, 1)$ is drawn to check whether:

$$r_2 \leq \frac{a_\mu}{\overline{a_\mu}}$$

If this succeeds R_μ is accepted because $r_2 \leq \frac{\frac{a_\mu}{\overline{a_\mu}}}{\frac{a_\mu}{a_\mu}} \leq \frac{a_\mu}{a_\mu}$. When this fails a_μ is computed and r_2 is tested against $\frac{a_\mu}{a_\mu}$. This happens infrequently when $\frac{\frac{a_\mu}{\overline{a_\mu}}}{\frac{a_\mu}{a_\mu}}$ is close to 1. If R_μ is accepted the firing time is computed, otherwise a new reaction is selected and tested again.

3.5.4 Advancement of the simulation

To remain exact RSSA has to advance the simulation at every attempt of rejection by an exponentially distributed variable with parameter $\overline{a_0}$. Assuming $k-1$ reactions and following the acceptance, the simulation has to advance by a quantity equal to the sum of k exponential random numbers:

$$\frac{1}{\overline{a_0}} \ln \frac{1}{u_1} + \frac{1}{\overline{a_0}} \ln \frac{1}{u_2} + \cdots + \frac{1}{\overline{a_0}} \ln \frac{1}{u_k}$$

Where $u_i \sim norm(0,1)$ are independent and identically distributed random numbers. This sum is the Erlang distribution $erlang(k, \overline{a_0})$.

3.5.5 Exactness of RSSA

For each simulation iteration of RSSA, R_μ is selected to fire with probability $\frac{a_\mu}{a_0}$ and its firing time τ follows an exponential distribution with rate a_0. Let $\mathbb{P}\{R_\mu\}$ be the probability that R_μ is selected and accepted to fire in a single attempt. This can be expressed as the product of the probability of being selected and being accepted:

$$\mathbb{P}\{R_\mu\} = \frac{\overline{a_\mu}}{\overline{a_0}} \frac{a_\mu}{\overline{a_0}} = \frac{a_\mu}{\overline{a_0}}$$

Let $\mathbb{P}\{R\}$ be the probability that some reaction is accepted in a single attempt, this is:

$$\mathbb{P}\{R\} = \frac{\sum_{j=1}^{M} a_j}{\overline{a_0}} = \frac{a_0}{\overline{a_0}}$$

R_μ being accepted after any number of rejections is a conditional probability of accepting R_μ knowing some reaction is accepted:

$$\mathbb{P}\{R_\mu | R\} = \frac{\frac{a_\mu}{\overline{a_0}}}{\frac{a_0}{\overline{a_0}}} = \frac{a_\mu}{a_0}$$

Demonstrating that the reaction is selected with the correct probability. Now let F_V be the cdf and f_V the pdf of V. Let k be the random variable for the number of attempts performed before accepting R_μ. k is geometrically distributed with success probability $\mathbb{P}\{R\}$. Let τ be a random variable corresponding to the simulation time advancement due to the firing of R_μ. Let $\mathbb{P}(\tau \leq x)$ be the probability that $\tau \leq x$ given a reaction is accepted after some trials, the pdf of τ:

$$
\begin{aligned}
f_\tau(x) &= \frac{\partial}{\partial x}\mathbb{P}\{\tau \leq x\} = \\
&= \frac{\partial}{\partial x}\sum_{k_0=1}^{\infty}\mathbb{P}\{\tau \leq x | k = k_0\}\mathbb{P}\{k = k_0\} = \\
&= \frac{\partial}{\partial x}\sum_{k_0=1}^{\infty} F_{erlang(k_0,\overline{a_0})}(x)\frac{a_0}{\overline{a_0}}\left(1-\frac{a_0}{\overline{a_0}}\right)^{k_0-1} = \\
&= \sum_{k_0=1}^{\infty}\frac{\partial}{\partial x}F_{erlang(k_0,\overline{a_0})}(x)\frac{a_0}{\overline{a_0}}\left(1-\frac{a_0}{\overline{a_0}}\right)^{k_0-1} = \\
&= \sum_{k_0=1}^{\infty} f_{erlang(k_0,\overline{a_0})}(x)\frac{a_0}{\overline{a_0}}\left(1-\frac{a_0}{\overline{a_0}}\right)^{k_0-1} = \\
&= \sum_{k_0=1}^{\infty}\frac{\overline{a_0}^{k_0}x^{k_0-1}e^{-\overline{a_0}x}}{(k_0-1)!}\frac{a_0}{\overline{a_0}}\left(\frac{\overline{a_0}-a_0}{\overline{a_0}}\right)^{k_0-1} = \\
&= a_0 e^{-\overline{a_0}x}\sum_{k_0=1}^{\infty}\frac{(\overline{a_0}-a_0)^{k_0-1}x^{k_0-1}}{(k_0-1)!} = \\
&= a_0 e^{-\overline{a_0}x}e^{x(\overline{a_0}-a_0)} = a_0 e^{-a_0 x}
\end{aligned}
$$

Noting that:

- Partitioning $\mathbb{P}(\tau < x)$ according to k is done because for a fixed k, τ is an Erlang distribution with parameters k and $\overline{a_0}$.
- Applying the closed form of the pdf of Erlang.
- $e^x = \sum_{n=0}^{\infty}\frac{x^n}{n!}$.

This shows how the firing time τ follows and exponential distribution $exp(a_0)$.

3.5.5.1 Bounds on the acceptance probability

The acceptance probability of a single attempt $\mathbb{P}\{R\}$ is bounded by:

$$\frac{\underline{a_0}}{\overline{a_0}} \leq \mathbb{P}\{R\} = \frac{a_0}{\overline{a_0}} \leq 1$$

Tighter lower or upper bounds yield a better acceptance probability: if $[\underline{X}, \overline{X}]$ is reduced to the concrete state $X(t)$, the acceptance probability is 1:

$$\underline{a_j} = \overline{a_j} = a_j$$

And RSSA is reduced to DM.

3.5.6 Evolution of the state

After firing $X(t)$ is updated and RSSA checks whether the new concrete state is compatible with the abstract state: $\underline{X_i} \leq X_i(t) \leq \overline{X_i}$ holds for each species. This is often the case as a reaction only

affects a few molecules. When this is true RSSA does not recompute the abstract propensities: the inequality $\underline{a_j} \leq a_j \leq \overline{a_j}$ still holds for all reaction and the next reaction step can be performed. If $X(t)$ falls outside the abstract state the abstract propensities have to be updated by redefining a new abstract state around the new concrete one and deriving the corresponding propensities. This operation can be made cheaper by observing that only the reaction affected by the species for which a new fluctuation interval has been redefined require an update of the propensity bounds. This are determined by a species-reaction RS dependency graph.

3.5.6.1 Species-Reaction dependency graph

Let $\mathcal{S}$ and $\mathcal{R}$ be the set of species and reactions in the biochemical network. The species-reaction SR dependency graph is the direct bipartite graph $\mathcal{G}(V, E)$ having:

$$V = \mathcal{S} \cup \mathcal{R}$$
$$E = \{(s, r) \in \mathcal{S} \times \mathcal{R} | s \in Reactants(r)\} \cup \{(r, s) \in \mathcal{R} \times \mathcal{S} | s \in Products(r)\}$$

This is a bipartite graph that shows the dependency of reactions on species. This allows RSSA to decide which reaction should recompute their propensity bounds. For a species S_i if its population moves out of the fluctuation interval RSSA recomputes propensity bounds of a reaction R_j if there is a directed edge from S_i to R_j in SR. The number of reaction that needs to recompute propensity bounds is equal to the out-degree of S_i. The reactions that have to recompute the propensity bounds if $X_i(t) \notin [\underline{X_i}, \overline{X_i}]$ are defined in term of the SR as:

$$ReactionAffectedBy(S_i) = \{R_j | \exists (S_i, R_j) \in \mathcal{G}\}$$

3.5.7 Algorithm

An implementation of RSSA is outlined in algorithm 19.

Algorithm 19: `Rejection-Based SSA (RSSA)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
build the species reaction SR dependency graph $\mathcal{G}$
foreach $S_i \in$ *Species* **do**
    define a new $[\underline{X_i}, \overline{X_i}]$ around X_i
$\overline{a_0} = 0$
foreach $R_j \in$ *Reactions* **do**
    compute $\overline{a_j}$ and $\underline{a_j}$
    $\overline{a_0} = \overline{a_0} + \overline{a_j}$
while $t < T_{\max}$ **do**
    repeat
        $u = 1$
        accepted = **false**
        repeat
            generate three random numbers $r_1, r_2, r_3 \sim norm(0, 1)$
            select R_μ with minimum μ such that $\sum_{j=1}^{\mu} \overline{a_j} \geq r_1 \overline{a_0}$
            if $r_2 \leq \frac{\underline{a_\mu}}{\overline{a_\mu}}$ **then**
                accepted = **true**
            else
                evaluate a_μ with state X
                if $r_2 \leq \frac{a_\mu}{\overline{a_\mu}}$ **then**
                    accepted = **true**
            $u = u \cdot r_3$
        until *accepted*
        $\tau = \frac{1}{\overline{a_0}} \ln(u)$
        $\vec{X} = \vec{X} + \vec{v}_\mu$
        $t = t + \tau$
    until $\exists(X_i \notin [\underline{X_i}, \overline{X_i}])$
    foreach $X \notin [\underline{X_i}, \overline{X_i}]$ **do**
        define a new $[\underline{X_i}, \overline{X_i}]$ around X_i
        foreach $R_j \in ReactionsAffectedBy(S_i)$ **do**
            compute new propensity bounds $\overline{a_j}$ and $\underline{a_i}$
            update $\overline{a_0}$

3.5.7.1 Discussion

3.5.7.1.1 Time complexity The time complexity is composed of the cost to realize a candidate reaction and the number of rejection tests. RSSA accumulates propensity upper bounds linearly until it finds a candidate reaction: selecting the candidate is $O(M)$. Let $\alpha = \frac{\overline{a_0}}{a_i}$ be the average number of times the search is performed until a candidate is accepted: the cost for the selection of firing is $O(\alpha M)$. This cost is compensated by a huge reaction in propensity updates: let T_{DM}^{update} be the cost for propensity updates in DM. In RSSA is $\frac{T_{DM}^{update}}{\beta}$, where β is the average number of skipped updates and is the average frequency of $X(t) \in [\underline{X}, \overline{X}]$, providing a significant improvement for the simulation performance.

3.5.8 Simultaneous RSSA

Simultaneous RSSA SRSSA is a variant of RSSA that generates multiple independent trajectories in a simulation run. The propensity bounds in SRSSA are computed once and shared across the simulations, reducing the memory requirements to store the propensity bounds. The recomputing is performed collectively in a single operation which reduces the total number of propensity updates and improves the simulation performance.

3.5.8.1 Global fluctuation interval

Let K be the number of trajectories and X^r the system state of the r-th realization with $r = 1, \ldots, K$. Let a_j^r the propensity of reaction R_j in the r-th realization. A lower and upper bound for each R_j such that $\underline{a_j} \leq a_j^r \leq \overline{a_j} \forall r = 1, \ldots, K$ and uses these for all K realization. It stores only M propensity bounds, which are derived by first defining a global fluctuation interval $[\underline{X}, \overline{X}]$ which bounds all possible population in all K states X^r. Then a_j is minimized or maximized a_j on the global interval, which can be defined as $X_i^{\min} = \min(X_i^1, \ldots, X_i^K)$ and $X_i^{\max} = \max(X_i^1, \ldots, X_i^K)$ respectively to be the minimum and maximum of species S_i in all K states. Then the population interval:

$$[\underline{X_i}, \overline{X_i}] = [(1 - \delta_i)X_i^{\min}, (1 + \delta_i)X_i^{\max}]$$

Will bound al population S_i in K states, where δ_i is the fluctuation rate of the species. The new global population intervals are redefined only when all K trajectories are stopped.

3.5.8.2 Algorithm

An implementation of SRSSA is outlined in algorithm 20.

Algorithm 20: `Simultaneous RSSA (SRSSA)()`

1 **Input**: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$ and the number of generated trajectories K

2 **Output**: K independent trajectory of the biochemical reaction network, which is a collection of states $X^r(t)$ for time $0 \leq t \leq T_{\max}$

foreach *Trajectory*$\in K$ **do**
  $t^r = 0$
  $\vec{X}^r = \vec{x}_0$
build the species reaction SR dependency graph $\mathcal{G}$
foreach $S_i \in$ *Species* **do**
  define a new $[\underline{X_i}, \overline{X_i}]$ around X_i, such that $\underline{X_i} \leq X_i^1, \ldots, X_i^K \leq \overline{X_i}$
$\overline{a_0} = 0$
foreach $R_j \in$ *Reactions* **do**
  compute $\overline{a_j}$ and $\underline{a_j}$
  $\overline{a_0} = \overline{a_0} + \overline{a_j}$
repeat
  $UpdateSpeciesSet = \emptyset$
  foreach $r = 1 \rightarrow K$ **do**
    repeat
      $u = 1$
      accepted = **false**
      repeat
        generate three random numbers $r_1, r_2, r_3 \sim norm(0, 1)$
        select R_μ with minimum μ such that $\sum_{j=1}^{\mu} \overline{a_j} \geq r_1 \overline{a_0}$
        if $r_2 \leq \frac{\underline{a_\mu}}{\overline{a_\mu}}$ **then**
          accepted = **true**
        else
          evaluate a_μ^r with state X^r
          if $r_2 \leq \frac{a_\mu}{\overline{a_\mu}}$ **then**
            accepted = **true**
        $u = u \cdot r_3$
      until *accepted*
      $\tau^r = \frac{1}{\overline{a_0}} \ln(u)$
      $\vec{X}^r = \vec{X}^r + \vec{v}_\mu$
      $t^r = t^r + \tau^r$
    until $\exists (X_i^r \notin [\underline{X_i}, \overline{X_i}]) \vee t^r \geq T_{\max}$
    foreach S_i *where* $X_i^r \notin [\underline{X_i}, \overline{X_i}]$ **do**
      $UpdateSpeciesSet = UpdateSpeciesSet \cup \{S_i\}$
  foreach $S_i \in UpdateSpeciesSet$ **do**
    define a new $[\underline{X_i}, \overline{X_i}]$ such that $\underline{X_i} \leq X_i^1, \ldots, X_i^K \leq \overline{X_i}$
    foreach $R_j \in ReactionsAffectedBy(S_i)$ **do**
      compute new propensity bounds $\overline{a_j}$ and $\underline{a_i}$
      update $\overline{a_0}$
until $t^r \geq T_{\max} \forall r = 1, \ldots, K$

3.5.9 Improvements for RSSA

The search for a candidate reaction in RSSA is linear and becomes a computational bottleneck for large reaction networks.

3.5.9.1 RSSA with tree-based search

The tree-based search can be used to reduce the time complexity of the search for the next reaction to logarithmic time. The tree stores the propensity upper bounds in the RSSA case: the search time for the candidate reaction is $O(\log M)$ and a tree updated is performed in $O(\log M)$. The time complexity of the algorithm is $O(\log M)$.

3.5.9.1.1 Algorithm An implementation of RSSA with tree-based search can be found in algorithm 21.

Algorithm 21: `Rejection-Based SSA with Tree-based search()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
build the species reaction SR dependency graph $\mathcal{G}$
foreach $S_i \in$ *Species* **do**
    define a new $[\underline{X_i}, \overline{X_i}]$ around X_i
$\overline{a_0} = 0$
foreach $R_j \in$ *Reactions* **do**
    compute $\overline{a_j}$ and $\underline{a_j}$
    $\overline{a_0} = \overline{a_0} + \overline{a_j}$
build TREE structure for M propensity upper bounds $\overline{a_j}$ by `build_tree`
while $t < T_{\max}$ **do**
    repeat
        $u = 1$
        accepted = **false**
        repeat
            generate three random numbers $r_1, r_2, r_3 \sim norm(0, 1)$
            select R_μ by `search_tree`
            if $r_2 \leq \frac{\underline{a_\mu}}{\overline{a_\mu}}$ **then**
                accepted = **true**
            else
                evaluate a_μ with state X
                if $r_2 \leq \frac{a_\mu}{\overline{a_\mu}}$ **then**
                    accepted = **true**
            $u = u \cdot r_3$
        until *accepted*
        $\tau = \frac{1}{\overline{a_0}} \ln(u)$
        $\vec{X} = \vec{X} + \vec{v}_\mu$
        $t = t + \tau$
    until $\exists (X_i \notin [\underline{X_i}, \overline{X_i}])$
    foreach $X \notin [\underline{X_i}, \overline{X_i}]$ **do**
        define a new $[\underline{X_i}, \overline{X_i}]$ around X_i
        foreach $R_j \in ReactionsAffectedBy(S_i)$ **do**
            compute new propensity bounds $\overline{a_j}$ and $\underline{a_i}$
            update the Tree bu algorithm *update_tree*
            update $\overline{a_0}$

3.5.9.2 RSSA with composition-rejection search

RSSA with composition-rejection search RSSA-CR employs composition-rejection search. The reaction are partitioned into L groups using the propensity bounds. R_j is put into G_l if $2^{u_l-1} \leq \overline{a_j} < 2^{u_l}$ where $u_l = \lceil \log_2(\overline{a_j}) \rceil$. Let $p_l = \sum_{R_j \in G_l} \overline{a_j}$ and $p_o = \sum_{l=1}^{L} p_l = \sum_{j=1}^{M} \overline{a_j} = \overline{a_0}$.

3.5.9.2.1 Selection of the next reaction The selection of the next reaction is a two step process:

- Selection of the group.
- Selection of a reaction.

3.5.9.2.1.1 Selection of a group A group G_α is selected with probability $\frac{p_\alpha}{p_0}$ by linearly accumulating p_l until a minimum α such that $\sum_{l=1}^{\alpha} p_l \geq r_1 p_0$ is found, where $r_0 \sim norm(0,1)$.

3.5.9.2.1.2 Selection of a reaction The selection of R_μ has two consecutive acceptance-rejection tests: the first randomly selects $R_\mu \in G_\alpha$ and accepts it with probability $\frac{\overline{a_\mu}}{2^{\mu\alpha}}$. This is repeated until R_μ is accepted. Then a second test with acceptance probability $\frac{a_\mu}{\overline{a_\mu}}$ is performed. If the test fails both R_μ and G_α are rejected.

3.5.9.2.2 Firing time The firing time is generated by sampling $erlang(k, p_0)$, but the number of trials k counts only the second rejection test.

3.5.9.2.3 Algorithm An implementation of RSSA with composition-rejection search can be found in algorithm 22.

Algorithm 22: `Rejection-Based SSA (RSSA)()`

1 **Input**: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
2 **Output**: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$

```
1  t = 0
2  X⃗ = x⃗_0
3  build the species reaction SR dependency graph G
4  foreach S_i ∈ Species do
5      define a new [X_i, X̄_i] around X_i
6  ā_0 = 0
7  foreach R_j ∈ Reactions do
8      compute ā_j and a_j
9      ā_0 = ā_0 + ā_j
10 group M reaction into L groups so that R_j ∈ G_l if 2^{u_l − 1} ≤ ā_j < 2^{u_l}
11 p_0 = Σ_{l=1}^{K} p_l
12 while t < T_max do
13     repeat
14         u = 1
15         accepted = false
16         repeat
17             generate three random numbers r_1 ∼ norm(0, 1)
18             select minimum group index α such that Σ_{l=1}^{α} p_l ≥ r_1 p_0
19             repeat
20                 generate a random number r_2 ∼ norm(0, 1)
21                 μ = [r_2 · |G_α|]
22                 r_2 = r_2|G_α| − μ
23             until r_2 = r_2|G_α| − μ
24             generate two random numbers r_3, r_4 ∼ norm(0, 1)
25             if r_3 ≤ a_μ / ā_μ then
26                 accepted = true
27             else
28                 evaluate a_μ with state X
29                 if r_3 ≤ a_μ / ā_μ then
30                     accepted = true
31             u = u · r_4
32         until accepted
33         τ = (1/p_0) ln(u)
34         X⃗ = X⃗ + v⃗_μ
35         t = t + τ
36     until ∃(X_i ∉ [X_i, X̄_i])
37     foreach X ∉ [X_i, X̄_i] do
38         define a new [X_i, X̄_i] around X_i
39         foreach R_j ∈ ReactionsAffectedBy(S_i) do
40             compute new propensity bounds ā_j and a_i
41             update G_l with its p_l and sum p_0
```

3.5.9.2.4 Discussion

3.5.9.2.4.1 Time complexity Te time complexity consist of the cost of selecting the group in $O(L)$ and the average number of times that the validation test is performed to accept a reaction is $\alpha = \frac{2\overline{a_\mu}}{\underline{a_\mu}}$, which is dependent on the ratio of the propensity bounds which can be tuned through the fluctuation interval. The total computational cost for the selection of a reaction is $O(L)$.

3.5.9.3 RSSA with table-lookup search

The alias table lookup search is a constant time search but it requires an expensive pre-processing step to build the lookup tables. Any discrete probability distribution over M probability values can be expressed as an equi-probable mixture of M two-points distribution. The M probabilities are $\frac{\overline{a_j}}{\overline{a_0}}$. The setup requires to build two tables implemented as arrays of size M in which the first, the cut-off table Q, stores the first values of the two point mixture and the second, the alias table A, contains the alias to the second part.

3.5.9.3.1 Building the tables When building the tables the objective is to transform the M probabilities into a square histogram. The probabilities greater than average are stored in the *Greater* set and the smaller in the *Smaller* one. For each loop an element of *Greater* and an element of *Smaller* are selected. The element from *Greater* transfer a part of its value to the smaller one from *Smaller* to make it average. This step implies that for $l \in Smaller$ such that $Q_l < 1$ there is no alias. This is repeated until all elements in *Smaller* are processed. To show that it will never reach a deadlock consider the invariant of the while loop, the sum of element in *Greater* and *Smaller* is:

$$Total = \sum_{j=1}^{M} Q_j = M$$

The average value will be 1. For each loop $l \in Smaller$ with value Q_l is removed and value Q_k of $k \in Greater$ is reduced by $1 - Q_l$. The total sum is reduced by 1, so the loss after the i-th loop is:

$$Loss = \sum_{k=1}^{i} 1 = i$$

The total number of elements in the sets is $M - i$: in each loop one element from *Smaller* is removed. The average value of elements in the sets after the i-th loop is:

$$\frac{Total - Loss}{M - i} = 1$$

For each loop, if *Smaller* is not empty there is at least one element in *Greater*, proving the claim. This procedure is outlined in algorithm 23.

Algorithm 23: `build_alias_table()`

Input: M probabilities $\frac{\overline{a_j}}{\overline{a_0}}$
Output: alias table A with size M storing reaction indices and cut-off table Q with size M storing cut-off probabilities
foreach $j = 1 \to M$ **do**
    $Q_j = M\frac{\overline{a_j}}{\overline{a_0}}$
$Greater = \emptyset$
$Smaller = \emptyset$
foreach $j = 1 \to M$ **do**
    if $Q_j \geq 1$ **then**
        add j to $Greater$
    else
        add j to $Smaller$
while $Greater \neq \emptyset \wedge Smaller \neq \emptyset$ **do**
    take $k \in Greater$ and $l \in Smaller$
    $A_l = k$
    remove l from $Smaller$
    $Q_k = Q_k - (1 - Q_l)$
    if $Q_k < 1$ **then**
        move k from $Greater$ to $Smaller$

3.5.9.3.2 Search for the candidate reaction The search considers a random number $r \sim norm(0, 1)$ as the parameter. First a random index $\mu = [Mr]$ is computed, where $[-]$ is the truncation operator. r is rescaled and compared against Q_μ the probability cut-off. If $r < Q_\mu$ μ is returned, otherwise the reaction index in A_μ is returned. This procedure is outlined in algorithm 24.

Algorithm 24: `search_alias_table(`r`)`

Input: alias array A with size M storing reaction indices and cut-off array Q with size M storing cut-off probabilities, and a random number $r \sim norm(0, 1)$
Output: a candidate reaction R_μ with probability $\frac{\overline{a_\mu}}{\overline{a_0}}$
$\mu = [Mr]$
$r = Mr - \mu$
if $r < Q_\mu$ **then**
    return μ
else
    return A_μ

3.5.9.3.3 Algorithm An implementation of RSSA with table-lookup search is outlined in algorithm 25.

Algorithm 25: `Rejection-Based SSA (RSSA)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
build the species reaction SR dependency graph $\mathcal{G}$
foreach $S_i \in$ *Species* **do**
    define a new $[\underline{X_i}, \overline{X_i}]$ around X_i
$\overline{a_0} = 0$
foreach $R_j \in$ *Reactions* **do**
    compute $\overline{a_j}$ and $\underline{a_j}$
    $\overline{a_0} = \overline{a_0} + \overline{a_j}$
build alias tables for M probabilities $\frac{\overline{a_j}}{\overline{a_0}}$ by `build_alias_table()`
while $t < T_{\max}$ **do**
    repeat
        $u = 1$
        accepted = **false**
        repeat
            generate three random numbers $r_1, r_2, r_3 \sim norm(0, 1)$
            select R_μ by `build_alias_table(`r_1`)`
            if $r_2 \leq \frac{\underline{a_\mu}}{\overline{a_\mu}}$ **then**
                accepted = **true**
            else
                evaluate a_μ with state X
                if $r_2 \leq \frac{a_\mu}{\overline{a_\mu}}$ **then**
                    accepted = **true**
            $u = u \cdot r_3$
        until *accepted*
        $\tau = \frac{1}{\overline{a_0}} \ln(u)$
        $\vec{X} = \vec{X} + \vec{v}_\mu$
        $t = t + \tau$
    until $\exists(X_i \notin [\underline{X_i}, \overline{X_i}])$
    foreach $X \notin [\underline{X_i}, \overline{X_i}]$ **do**
        define a new $[\underline{X_i}, \overline{X_i}]$ around X_i
        foreach $R_j \in \overline{ReactionsAffectedBy}(S_i)$ **do**
            compute new propensity bounds $\overline{a_j}$ and $\underline{a_i}$
            update $\overline{a_0}$
    rebuild alias tables for M probabilities $\frac{\overline{a_j}}{\overline{a_0}}$ by `build_alias_table()`

3.5.9.3.4 Discussion The alias table lookup takes one comparison and at most two memory accesses, so $O(1)$. The constant time search is affected by a large computational cost for rebuilding the lookup tables, which is $O(M)$, but the average number of times the lookup tables rebuild is controllable through the fluctuation interval.

Chapter 4

Approximation algorithms

4.1 Introduction

Exact simulation of complex biological system is often to expensive due to their stochastic and multi scale nature. These lead to the development of approximate algorithms, which improve the simulation efficiency by sacrificing their accuracy. Multiple firings are coalesced and performed together in one simulation step with a huge speed up. Approximate methods should be used because:

- It might be the only possible and feasible solution to solve a problem.
- reality is affected by error, so even when using exact stochastic simulation algorithms a small degree of approximation should be considered. A certain deal of error could aid in retrieving a more realistic result.

In this case, each algorithm (Figure 4.1) approximates in a different way, assuming different approximations. Of the three main computational strategies presented here, there is one which is much more popular with respect to the others the τ leaping method.

To improve the exact simulations algorithms, one should consider to:

- Work on the number of reaction events, grouping them in such a way to reduce the number of reactions in the system. The τ leaping methodology is working in this direction.
- Improve the computation of the propensity. For instance, the probability-weighted Monte Carlo is particularly promising in situations in which there is a huge differences in propensities among reactions.

4.2 Probability-Weighted Dynamic Monte Carlo Method

The probability-weighted dynamic Monte Carlo method (PW-DMC) is an approximation approach for improving the computational efficiency of stochastic simulations of reaction networks where some reactions have propensities significantly larger than other reactions. This is because fast reactions (with large propensities) occur frequently and dominate the simulation, while slow reactions (with small propensities) occur less frequently. The events from the slow reaction are rare and their statistical estimation is unreliable. PW-DMC attempts to equalize the propensities of reactions so that a larger increment time step can be chosen, improving simulation's performance.

]H]

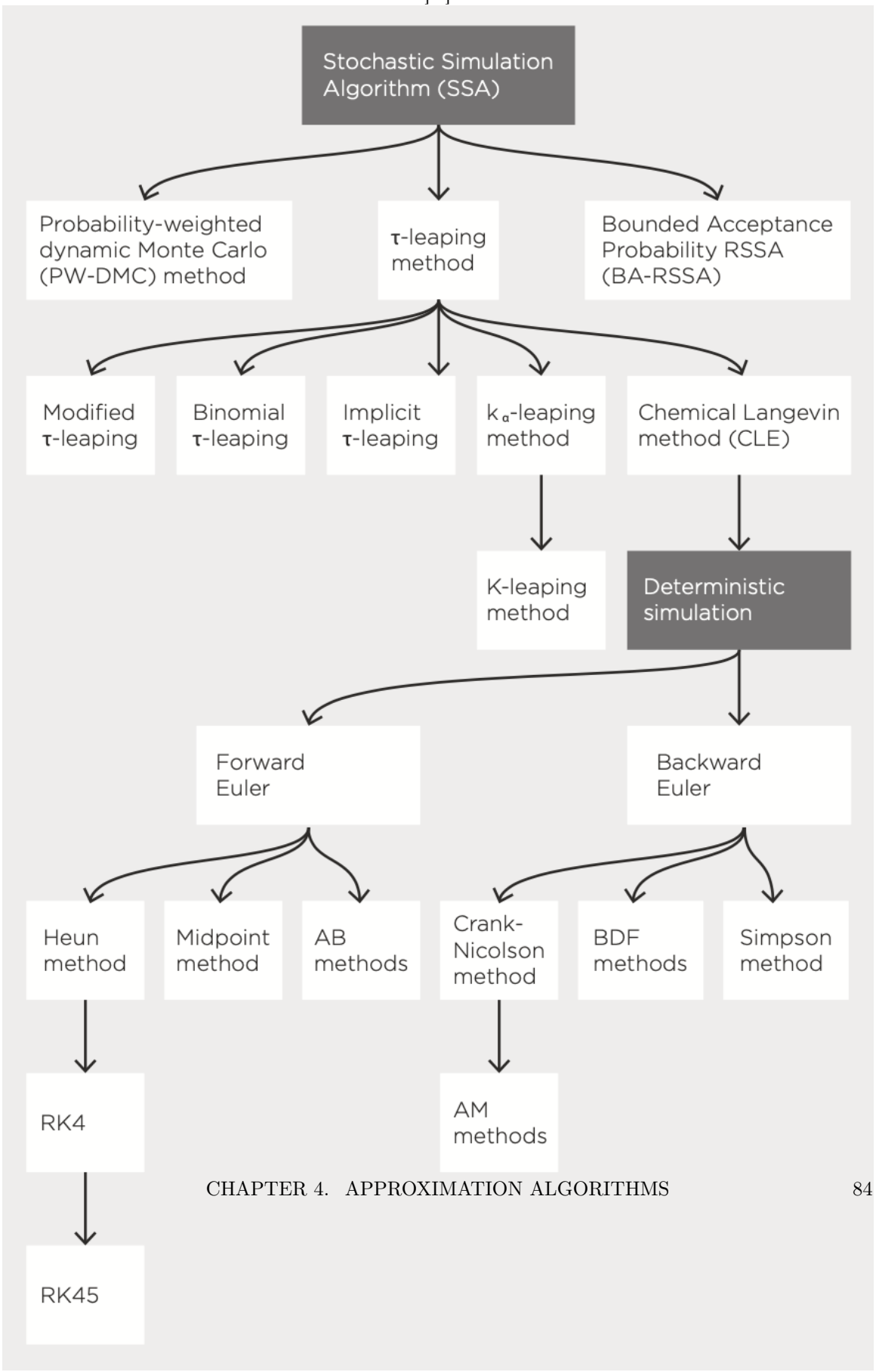

Figure 4.1: Relationship between stochastic simulation algorithms

4.2.1 Weighted sampling

The principle of this algorithm is a sort of modification of the probability distribution of the next reaction firing through weighted sampling. The propensity a_j of R_j is scaled by a biasing weight w_j defined as the number of firing of R_j at each step. To compute it the unweighted probability $\frac{a_j}{a_0}$ is discretized into integer valued histogram bins according to a size b.

4.2.1.1 Effective propensity

The effective propensity a_j^w is computed as:

$$a_j^w = \frac{a_j}{w_j}$$

These are then used for the selection of R_μ. The chance that a slow reaction fires is increased and so is the frequency of rate events.

4.2.2 Realization of the reaction firing

The realization of the next reaction firing has two step:

- Selection of the reaction.
- Correction of the firing time.

4.2.2.1 Selection of the reaction

R_μ is selected with probability $\frac{a_\mu^w}{a_0^\mu}$, where $a_0^\mu = \sum_{j=1}^{M} a_j^w$. This can be done by linearly accumulating a_μ^w as:

$$\mu = \arg\min_{j \in \mu} \sum_{j=1}^{\mu} a_j^w \geq r_1 a_0^w$$

Where $r_1 \sim norm(0, 1)$.

4.2.2.2 Correction of the firing time

The firing time τ is corrected to account for the bias. τ is generated from an exponential distribution with rate a_0^w:

$$\tau = \frac{1}{a_0^w} \ln\left(\frac{1}{r_2}\right)$$

Where $r_2 \sim norm(0, 1)$. Then the state at time $t+\tau$ is updated assuming there are w_μ consecutive firings of R_μ in $[t, t + \tau[$ and the state at time $t + \tau$ is updated as:

$$X(t + \tau) = X(t) + w_\mu \vec{v}_\mu$$

The weight of the reactions have to be updated accordingly.

4.2.3 Bounds on the weights

w_j should be an integer value because the population of a species involved in a reaction is an integer. Its magnitude is constrained in order to bound the error in the results. Each time R_j is selected it fires w_j times. The change of each species S_i in R_j is w_j, the fluctuation of the population is thus $\frac{w_j}{X}$. This ratio must be less than a predefined tolerance ϵ to ensure the statistical uncertainty of the estimation of X_i. Moreover the ratio would be negligibly small when the population of species is large, and the chosen weight does not affect the simulation result. However the weight introduces an approximation to the temporal dynamics when the population of species is low and in this case w_j must be adjusted to maintain $w_j < \epsilon X_i$.

4.2.4 Algorithm

An implementation of PW-DMC can be found in algorithm 26.

Algorithm 26: `Probability-Weighted Dynamic Monte Carlo (PW-DMC)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$, the size b for discretizing probability of reacitons and tolerance ϵ for constraining the fluctuation of species.
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
build the reaction dependency graph G
compute propensity a_j for each reaction R_j
while $t < T_{\max}$ **do**
    compute w_j for each R_j
    compute $a_j^w = \frac{a_j}{w_j}$ for each R_j
    $a_0^w = \sum_{j=1}^{M} a_j^w$
    generate two random numbers $r_1, r_2 \sim norm(0, 1)$
    select R_μ with the smallest index μ such that $\sum_{j=1}^{\mu} a_j^w \geq r_1 a_0^w$
    $\tau = \frac{1}{a_0^w} \ln \frac{1}{r_2}$
    $\vec{X} = \vec{X} + w_\mu \vec{v}_\mu$
    $t = t + \tau$
    foreach $R_j \in Dependents(R_\mu)$ **do**
        update a_j

4.2.5 Discussion

4.2.5.1 Time complexity

The speed-up gain in PW-DM is achieved by multiple firings of a reaction in each step. The weight can be tuned to produce a significant gain in computational performance, while keeping accuracy. The frequency of rare events is increased, helping explore more the biochemical systems.

4.2.5.2 Limitations

PW-DMC skews the probability distribution of the state, because the weight sampling groups reaction of the same type in bundles and fires them together, loosing the order of reaction. PW-DMC could misdescribe the fluctuation of species in the result. ϵ has to be set for constraining fluctuation of species to a reasonably small value in order to bound the accuracy of the simulation.

4.3 Bounded acceptance probability RSSA

The bounded acceptance probability RSSA BA-RSSA focuses on the simulation of reactions involved species with both small and large population. These will have a large propensity. Many firings can occur in time interval and quickly deplete the small population species. To bound the error updates must be performed frequently, degrading the simulation performance, especially if the small population is a hub species. The simulation of this reactions is accelerated by bounding the acceptance of a candidate reaction selected by RSSA. It accepts a candidate reaction without validation if its acceptance probability is greater than a user-defined probability, reducing the computational cost for both selecting of reaction firing and propensity updates.

4.3.1 Defining the bounds

Let $0 \leq \alpha \leq 1$ be a constant defined as a lower bound for the acceptance probability and R_j the selected reaction. BA-RSSA guarantees that the probability that R_j is accepted to fire is greater than α. The validation step of standard RSSA accepts R_j to fire with probability $\frac{a_j(X(t))}{\overline{a_j}}$, the goal of BA-RSSA is to ensure:

$$\frac{a_j(X(t))}{\overline{a_j}} \geq \alpha$$

This is difficult to assess because it depends on $X(t)$. Anytime the state changes a_j has to be re-evaluated. To cope with this BA-RSSA exploits the fact that $a_j(X(t)) \geq \underline{a_j}$ when $X(t) \in [\underline{X}, \overline{X}]$, therefore if:

$$\frac{\underline{a_j}}{\overline{a_j}} \geq \alpha$$

Holds for each reaction $\frac{a_j(X(t))}{\overline{a_j}} \geq \alpha$ is automatically satisfied.

4.3.2 Defining the fluctuation interval

To enforce $\frac{\underline{a_j}}{\overline{a_j}} \geq \alpha$, $[\underline{X}, \overline{X}]$ has to be defined so that $\frac{\underline{a_j}}{\overline{a_j}}$ of each R_j within the fluctuation interval is bounded by α. A fluctuation rate δ_i for each S_i involved in R_j has to be selected so that when S_i fluctuates in $[(1-\delta_i)X_i(t), (1+\delta_i)X_i(t)]$ the inequality is satisfied. Only a range of δ_i can be chosen given the ratio of propensity bounds.

4.3.2.1 Computing the maximum fluctuation rate

The maximum fluctuation rate δ_i computation is reaction dependent.

4.3.2.1.1 Synthesis reaction For a synthesis reaction R_j, a_j is independent of $X(t)$ and is equal to x_j. Also the bounds are constant, the ratio is equal to 1 and the inequality is satisfied.

4.3.2.1.2 Unimolecolar reaction For a unimolecular reaction R_j, $\overline{a_j} = c_j(1+\delta_i)X_i$ and $\underline{a_j} = c_j(1-\delta_i)X_i$, so that the inequality becomes:

$$\frac{1-\delta_i}{1+\delta_i} \geq \alpha$$

The maximum value of δ_i then becomes:

$$\delta_i = \frac{1-\alpha}{1+\alpha}$$

4.3.2.1.3 Bimolecular reaction For a bimolecular reaction R_j $\overline{a_j} = c_j(1+\delta_j)(1+\delta_k)X_iX_k$ and $\underline{a_j} = c_j(1-\delta_i)(1-\delta_k)X_iX_k$, so that:

$$\frac{(1-\delta_i)(1-\delta_k)}{(1+\delta_i)(1+\delta_k)} \geq \alpha$$

Which is a quadratic equation of two independent variables, which can be split in to part so that:

$$\frac{1-\delta_i}{1+\delta_i} \geq \sqrt{\alpha} \wedge \frac{1-\delta_k}{1+\delta_k} \geq \sqrt{\alpha}$$

So that the maximum values for the fluctuation rates are:

$$\delta_i = \delta_k = \frac{1-\sqrt{\alpha}}{1+\sqrt{\alpha}}$$

4.3.2.1.4 Dimerization reaction For a dimerization reaction R_j by a similar derivation as the bimolecular one:

$$\frac{(1-\delta_i)((1-\delta_i)X_i-1)}{(1+\delta_i)((1+\delta_i)X_i-1)} \geq \alpha$$

So that the maximum δ_i:

$$\delta_i = \frac{1-\sqrt{\alpha}}{1+\sqrt{\alpha}}\left(1-\frac{1}{X_i}\right)$$

4.3.3 Selecting the propensity

Once the state is confined in a fluctuation interval that satisfies the bounds, any $b_j \in [\underline{a_j}, \overline{a_j}]$ can be chosen as the propensity. The extreme values may bias the selection step, so the average can be used:

$$b_j = \frac{\underline{a_j} + \overline{a_j}}{2}$$

This choice requires two evaluation of the propensity function. Another choice is to compute the value of the propensity at the central point of the fluctuation interval so to have one evaluation of the propensity function:

$$b_J = a_j\left(\frac{\underline{X} + \overline{X}}{2}\right) = a_j(X(t))$$

4.3.4 Algorithm

An implementation of BA-RSSA can be found in algorithm 27.

Algorithm 27: `Buonded Acceptance Probability RSSA (BA-RSSA)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$ and the bon=und of the acceptance probability $0 \leq \alpha \leq 1$
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
build the species reaction SR dependency graph $\mathcal{G}$
define $\delta_i \forall S_i$ involved in R_j to ensure that the acceptance of R_j is bounded by α
compute the fluctuation interval $[\underline{X_i}, \overline{X_i}] = [(1 - \delta - i)X_i, (1 + \delta_i)(X_i)]$ for each species S_i around its current population X_i
compute b_j for each R_j
$b_0 = \sum_{j=1}^{M} b_j$
$\overline{a_0} = 0$
foreach $R_j \in$ *Reactions* **do**
    compute $\overline{a_j}$ and $\underline{a_j}$
    $\overline{a_0} = \overline{a_0} + \overline{a_j}$
while $t < T_{\max}$ **do**
    repeat
        generate two random numbers $r_1, r_2 \sim norm(0, 1)$
        select minimum index μ such that $\sum_{j=1}^{\mu} b_\mu \geq r_1 b_0$
        $\tau = \frac{1}{b_0}\ln(r_2)$
        $\vec{X} = \vec{X} + \vec{v}_\mu$
        $t = t + \tau$
    until $\exists(X_i \notin [\underline{X_i}, \overline{X_i}])$
    foreach $X_i \notin [\underline{X_i}, \overline{X_i}]$ **do**
        define a new $[\underline{X_i}, \overline{X_i}] = [(1 - \delta - i)X_i, (1 + \delta_i)(X_i)]$
        foreach $R_j \in ReactionsAffectedBy(S_i)$ **do**
            update b_j and b_0

4.3.5 Discussion

When $\alpha = 1$ the algorithm returns to the exact case.

4.3.5.1 Time complexity

BA-RSSA reduces the selection cost for the next reaction firing and avoids a large number of the propensity updates. The selected reaction firing is ensured to fire with probability greater than a threshold when the population is confined in its fluctuation interval. The propensity updates are performed infrequently and only locally.

4.4 τ-Leaping method

The aim of the τ-leaping method is to discretize the time axis into intervals and to approximate the number of reaction firing in each one. The simulation then leaps form one interval to the next with many reaction firing performed simultaneously.

4.4.1 Simulation time

The simulation time is discretized into time intervals of length τ, the leap time. This is adaptively defined during the simulation. Consider a time interval $[t, t+\tau[$. The joint probability $\mathbb{P}\{k_1, \ldots, k_M|\tau, \vec{x}, t\}$ gives the number of firing of reactions during the time interval given the state at time t. And is defined as the probability that there are k_j firings of R_j during the time interval. Finding an exact formula is as difficult as solving the CME, an approximation can be derived by assuming that changes in propensities due to reaction firing are insignificant, called the leap condition.

4.4.1.1 Leap condition

The leap condition states that there exists a leap $\tau > 0$ such that the change in a_j of each reaction R_j during the time interval $[t, t+\tau[$ is negligibly small. Let now $[t, t+\tau[$ be the time interval in which the leap condition is satisfied. The propensity of R_j during this interval given the state $X(t) = \vec{x}$ is a constant value $a_j(\vec{x})$. The probability that R_j fires in $dt \in [t, t+\tau[$ is constant and equal to $a_j(\vec{x})dt$, regardless if other reaction fire. Let $\mathbb{P}\{k_j|\tau, \vec{x}, t\}$ be the probability that k_j firing of R_j happen in the interval. It can be shown that:

$$\mathbb{P}|\{k_j|\tau, \vec{x}, t\} = \frac{(a_j(\vec{x})\tau)^{k_j}}{k_j!} e^{-a_j(\vec{x})\tau}$$

Which is a Poisson distribution: k_j is a Poisson distributed random number $poi(a_j(\vec{x})\tau)$. Because the M probabilities are statistically independent, the joint probability is:

$$\mathbb{P}\{k_1, \ldots, k_M|\tau, \vec{x}, t\} = \prod_{j=1}^{M} \mathbb{P}\{k_j|\tau, \vec{x}, t\}$$

This allows to implement the τ-leaping.

4.4.2 Advancing the simulation time

Let $X(t) = \vec{x}$ at time t and $[t, t+\tau[$ that satisfies the leap condition. k_j is generated by sampling $poi(a_j(\vec{x})\tau)$. k_j are ensured to distribute with the joint probability. Knowing the firing times of the reaction, the method leaps down time t by τ to the new time $t+\tau$ and updates the state:

$$X(t+\tau) = \vec{x} + \sum_{j=1}^{M} k_J \vec{v}_j = \vec{x} \sum_{j=1}^{M} poi(a_j(\vec{x}\tau)\vec{v}_j)$$

If the number of firing during the time interval is sufficiently large the τ-leaping method is faster than the exact simulation.

4.4.3 Issues

The τ-leaping method exposes many issues that must be addressed for a practical implementation.

4.4.3.1 Efficiency and accuracy

The efficiency and accuracy are dependent on how to choose a leap τ. If propensities are independent of the state any τ satisfies the leap condition, making it an exact method. If the propensities are state dependent, the selection of the leap is a trade-off between simulation accuracy and its performance. If τ is too large, the simulation is fast but less accurate, if it is too little the simulation is slow. There is a need to write a procedure to determine the largest τ approximately satisfying the leap condition.

4.4.3.2 Negative population of reactant species

The simulation needs to make sure that the generated random number do not cause the firing of reactions to result in negative population of species.

4.4.3.3 Switch to exact simulation

It needs a robust condition to switch to exact simulation when τ is very small because the cost for generating the random numbers becomes expensive, making exact SSA more efficient.

4.4.4 Leap selection

The leap selection procedure tries to determine a leap approximately satisfying the leap condition by bounding the change in propensity during the leap by an error parameter. Let $0 < \epsilon \ll 1$ be the error parameter and $\Delta a_j(\vec{x}) = a_j(X(t+\tau)) - a_j(X(t))$, the change in propensity after the leap, the leap selection will select τ such that the propensity change is bounded by the error parameter ϵ.

4.4.4.1 Postleap τ selection

The postleap selection starts with a predefined small τ value. A trial leap is performed and the difference in propensity is computed and compared against ϵ, checking $|\Delta a_j(\vec{x})| \leq \epsilon$. If this holds for all reaction τ is accepted, if it fails it is reduced and the procedure is repeated.

4.4.4.1.1 Issues This procedure poses many issues: it is not robust, the starting value is dependent on the model. Moreover a lot of random numbers may be wasted during the simulation, degradating the simulation performance. Furthermore, the selection may bias infrequent reactions from large changes.

4.4.4.2 Preleap τ selection

The leap selection estimates the changes in propensities of reactions. The selection of the leap can be directly through bounding changes in propensity values or through bounding changes in species population.

4.4.4.2.1 Bounding changes in propensities The approach determines the leap by forcing the propensity change to be bounded by a fraction of the total propensity. The condition for enforcing the leap condition is:

$$|\Delta a_j(\vec{x})| \leq \epsilon a_0(\vec{x})$$

Let λ be the net change vector in which each element denotes the change in population X_i of species S_i due to the firing:

$$\lambda = X(t+\tau) - \vec{x} = \sum_{j=1}^{M} poi(a_j(\vec{x})\tau)\vec{v}_j$$

Using the first-order Taylor expansion of $\Delta a_j(\vec{x})$ by using the net change vector, it can be approximated as:

$$\Delta a_j(\vec{x}) \approx \lambda \cdot \nabla a_j(\vec{x}) = \sum_{i=1}^{N} \lambda_i \frac{da_j(\vec{x})}{dX_i}$$

Define now M^2 functions such that:

$$f_{jl}(\vec{x}) = \sum_{i=1}^{N} \frac{da_j(\vec{x})}{dX_i}\vec{v}_{lj}$$

Where j, l run over the index set of reactions. Plugging this into the previous renaming the running index to l and rearranging the orders of summation:

$$\Delta a_j(\vec{x}) \approx \sum_{l=1}^{M} f_{jl}(\vec{x}) poi(a_l(\vec{x})\tau)$$

Showing that $\Delta a_j(\vec{x})$ is a linear combination of M independent Poisson-distributed random number and denotes a random variable with mean:

$$\mathbb{E}[\Delta a_j(\vec{x})] \approx \sum_{l=1}^{M} f_{jl}(\vec{x})\mathbb{E}[poi(a_l(\vec{x})\tau)] = \sum_{l=1}^{M} f_{jl}(\vec{x})(a_l(\vec{x})\tau)$$

And variance:

$$Var[\Delta a_j(\vec{x})] \approx \sum_{l=1}^{M} f_{jl}^2(\vec{x}) Var[poi(a_l(\vec{x})\tau)] = \sum_{l=1}^{M} f_{jl}^2 f(\vec{x})(a_l(\vec{x})\tau)$$

Allowing to approximate the random variable, considering a conservative approximation:

$$\Delta a_j(\vec{x}) \approx \mathbb{E}[\Delta a_k(\vec{x})] + \sqrt{Var[\Delta a_j(\vec{x})]}$$

Now considering the error parameter:

$$|\mathbb{E}[\Delta a_j(\vec{x})]| + \sqrt{Var[\Delta a_j(\vec{x})]} \leq \epsilon a_0(\vec{x})$$

To satisfy this the to terms are constrained separately:

$$|\mathbb{E}[\Delta a_j(\vec{x})]| \leq \frac{\epsilon a_0(\vec{x})}{2} \quad \wedge \quad \sqrt{Var[\Delta a_j(\vec{x})]} \leq \frac{\epsilon a_0(\vec{x})}{2}$$

Ensuring the constraint. The scaling factor $\frac{1}{2}$ is a tunable parameter. The largest τ that satisfies the leap condition is:

$$\tau = \min_{j=1}^{M} \left(\frac{\epsilon a_0(\vec{x})}{2|\mu_j|}, \frac{(\epsilon a_0(\vec{x}))^2}{4\sigma_j^2} \right)$$

Where:

$$\mu_j(\vec{x}) = \sum_{l=1}^{M} f_{jl}(\vec{x}) a_l(\vec{x}) = \sum_{l=1}^{M} \sum_{i=1}^{N} \frac{da_j(\vec{x})}{dX_i} \vec{v}_{li} a_l(\vec{x})$$

And:

$$\sigma_j^2(\vec{x}) = \sum_{l=1}^{M} f_{jl}^2(\vec{x}) a_l(\vec{x}) = \sum_{l=1}^{M} \sum -i = 1^N \left(\frac{da_j(\vec{x})}{dX_i} \vec{v}_{li} \right)^2 a_l(\vec{x})$$

4.4.4.2.2 Bounding changes in species population The bounding changes in propensities is refined in this method: although $a_0(\vec{x})$ does limit the change in propensity, it might produce less accurate τ. Moreover the need to evaluate M^2 partial derivative is removed. The τ-selection bounds the change in propensity $\Delta a_j(\vec{x})$ of each R_j by its current $a_j(\vec{x})$ instead of the total. The condition $|\Delta a_j(\vec{x})| \leq \epsilon a_j(\vec{x})$ when a_j approaches 0, forces the leap time to be zero, halting the simulation. A minimum amount of changes in propensity of each reaction is enforced, observing that propensity changes only by discrete amounts. For R_j the minimum amount of change can be selected to be rate constant c_j, so the leap condition is written as:

$$|\Delta a_j(\vec{x})| \leq \max\{\epsilon a_j(\vec{x}, c_j)\}$$

Moreover instead of directly enforcing the leap condition, the change in the population is bound such that if the change in the population of a species is bounded, then the change in propensity of the corresponding reaction given by the previous condition is satisfied, implying that it is not needed to evaluate the M^2 partial derivatives. Let $\Delta X_i = X_i(t+\tau) - X_i(t)$ be the change in the population of S_i after the lap. The τ selection bounds the population change such that:

$$|\Delta X_i| \leq \max\{\epsilon_i X_i, 1\}$$

Where ϵ_i is dependent from ϵ and $X_i(t)$ such that the equation is satisfied and the change in propensity is approximately satisfied, enforcing the leap condition.

4.4.4.2.2.1 Unimolecolar reaction For a unimolecolar reaction $a_j = c_j X_i$ and the propensity change is $\Delta a_j = c_j \Delta X_i$. The relative change in propensity is:

$$\frac{\Delta a_j}{a_j} = \frac{\Delta X_i}{X_i}$$

If the relative change in population S_i is bounded by $\epsilon_i = \epsilon$, the relative change in propensity of R_j is bounded by ϵ.

4.4.4.2.2.2 Bimolecular reaction For a bimolecular reaction $a_j = c_j X_i X_k$, and the change in propensity can be approximated as:

$$\Delta a_j \approx c_j X_i \Delta X_k + c_j X_k \Delta X_i$$

So the relative change in propensity:

$$\frac{\Delta a_j}{a_j} \approx \frac{\Delta X_i}{X_i} + \frac{\Delta X_j}{X_k}$$

The relative change in population of S_j is bounded by $\epsilon_j = \frac{\epsilon}{2}$ with $j = i, k$, the relative change in propensity of R_j is bounded by ϵ.

4.4.4.2.2.3 Dimerization reaction For a dimerization reaction $a_j = \frac{1}{2} c_j X_i (X_i - 1)$. The change in propensity:

$$\Delta a_j \approx \frac{1}{2} c_j (X_i - 1) \Delta X_i + \frac{1}{2} c_j X_i \Delta X_i$$

And the relative change in propensity:

$$\frac{\Delta a_j}{a_j} \approx \frac{\Delta X_i}{X_i} + \frac{\Delta X_i}{X_i - 1} = \frac{\Delta X_i}{X_i} \left(2 + \frac{1}{X_i - 1} \right)$$

If the relative change in population is bounded by $\epsilon_i = \frac{\epsilon}{g}$ with $g = \frac{2+1}{X_i - 1}$, the relative change in propensity of the reaction will be bounded by ϵ.

4.4.4.2.2.4 Approximating the change in population Knowing ϵ_i, the last step is approximating ΔX_i by knowing the net change in population of species:

$$\Delta X_i = \lambda_i = \sum_{j=1}^{M} poi(a_j(\vec{x})\tau)\vec{v}_{ji}$$

Substituting this into the leap condition and bounding the expected value and variance of ΔX_i, the largest τ that satisfies the lap condition is:

$$\tau = \min_{i=1}^{N} \left(\frac{\max\{\epsilon_i X_i, 1\}}{2|\hat{\mu}_i(\vec{x})|}, \frac{\max\{\epsilon_i X_i, 1\}^2}{4\hat{\sigma}_i^2(\vec{x})} \right)$$

Where:

$$\hat{\mu}_i(\vec{x}) = \sum_{j=1}^{M} \vec{v}_{ji} a_j(\vec{x}) \quad \wedge \quad \hat{\sigma}_i^2(\vec{x}) = \sum_{j=1}^{M} v_{ij}^2 a_j(\vec{x})$$

4.4.5 Avoiding the negative population problem

The number of firings k_j follows a Poison distribution that could lead to negative population of reactants. This could be due to two situations.

4.4.5.1 The number of firings is greater than the current population of reactants

k_j could be greater than the current population of reactants because k_j is unbounded.

4.4.5.2 Simultaneous firing of multiple reactions

Although the population of S_i could be greater than k_j, it could be that S_i is a common reactant and the total number of firings of reactions sharing it could be grater than the population.

4.4.5.3 Handling the negative population problem

A simple strategy to handle this problem is to monitor the population of each species and each time there is a species whose population is negative a flag is set and the current leap is rejected and the simulation is rolled back. A new leap trial is performed with a smaller leap value $\alpha\tau$ where α is a reduction factor. This process is repeated until no negative populations are found.

4.4.6 Switching to exact simulation

If τ is smaller than a few multiple of $\frac{1}{a_0(\vec{x})}$, the expected time to the firing of the next reaction in the exact simulation, it is likely that only some of the k_j are 1, while the others are 0. This method than gains little over the exact strategy, so it is better to use exact SSA. To handle this case let k be an integer denoting a multiplicative factor of the expected time to the firing of a reaction and p the number of exact SSA steps performed. $\frac{k}{a_0(\vec{x})}$ is defined as threshold for switching to exact SSA, performing p exact steps if τ is smaller than the threshold before trying a new leap.

4.4.7 The τ-leaping algorithm

An implementation of the τ-leaping method can be found in algorithm 28.

Algorithm 28: `τ-leaping method()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$, the error control paramter $0 < \epsilon \ll 1$, the reduction factor $\alpha < 1$, the threshold paramter k and the exact number of exact SSA steps parameter p

Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$

$t = 0$

$\vec{X} = \vec{x}_0$

$a_0 = 0$

foreach $R_j \in$ *Reactions* **do**

    compute a_j

    $a_0 = a_0 + a_j$

while $t < T_{\max}$ **do**

    $threshold = \frac{k}{a_0}$

    determine τ satisfying the leap condition with one of the leap selection procedures

    acceptedLeap =**false**

    repeat

        acceptedLeap = **true**

        if $\tau > threshold$ **then**

            generate M Poisson-distributed random numbers $k_j \sim poi(a_j(\vec{x})\tau)$

            $X = X + \sum_{j=1}^{M} k_j \vec{v}_j$

            $t = t + \tau$

        else

            perform p SSA simulation steps

        if $\exists$ *a species in X whose population* $X_i < 0$ **then**

            roll back state $X = X - \sum_{j=1}^{M} k_j \vec{v}_j$ and time $t = t - \tau$

            $\tau = \alpha\tau$

            acceptedLeap = **false**

    until *acceptedLeap*

4.4.8 Improvements for τ-leaping

4.4.8.1 Modified τ-leaping

The modified τ-leaping method efficiently handles the negative population problem. If the population of a species is low, the probability that it becomes negative is higher when reaction involving it fire. During the simulation, the reaction involving the low population species will be marked as critical reaction and monitored because their reactant species are likely to be exhausted.

4.4.8.1.1 Permitted firings Let L_j be the number of permitted firings of R_j during τ. L_j dependents on the reactants of R_j, but it is difficult to determine because the population of them

can change due to other reactions. This methods estimate L_i by assuming that the reactant of reactions are independent during the simulation. In this way the maximum number of permitted firings of R_j involving S_i is equal to the population of S_i divided by its stoichiometric coefficient v_{ji}^0. The minimum of these value will give the maximum number of permitted firings:

$$L_j = \min_{S_i \in Reactants} \left[\frac{X_i}{v_{ij}^-} \right]$$

Where $[-]$ is the truncation operator.

4.4.8.1.2 Critical reactions Let n_c be the critical value for classifying reactions. The reaction set is partitioned into the critical reaction set $\mathcal{R}^c$ and the non critical one $\mathcal{R}^{nc}$. $\mathcal{R}^c$ contains reactions with $L_l \leq n_c$ and $\mathcal{R}^{nc}$ the rest:

$$\mathcal{R}^x = \{R_j | L_j \leq n_c\} \quad \wedge \quad \mathcal{R}^{nc} = \{R_j | L_j > n_c\}$$

The partition has to be updated regularly because L_j changes after every reaction firing.

4.4.8.1.3 Firing of reactions The firing of reaction in $\mathcal{R}^c$ is done through SSA, while $\mathcal{R}^{nc}$ with τ-leaping. Let τ^{nc} be the largest time that satisfies the leap condition for reaction in $\mathcal{R}^{nc}$ and τ^c the next firing time of a reaction in $\mathcal{R}^c$. The actual leap time will be:

$$\tau = \min(\tau^{nc}, \tau^c)$$

Because at most one reaction in $\mathcal{R}^c$ is allowed to fire in the leap so that the population of their species never becomes negative.

4.4.8.1.4 Algorithm The modified t-leaping method is outlined in algorithm 29.

Algorithm 29: `Modified` τ`-leaping method()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$, the error control paramter $0 < \epsilon \ll 1$, the reduction factor $\alpha < 1$, the threshold paramter k and the exact number of exact SSA steps parameter p,, the critical value n_c

Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$

$t = 0$

$\vec{X} = \vec{x}_0$

while $t < T_{\max}$ **do**

  $a_0 = 0$

  foreach $R_j \in$ *Reactions* **do**

    compute a_j

    $a_0 = a_0 + a_j$

  compute L_j for every reaction R_j

  partition reaction into critical $\mathcal{R}^c$ and non-critical $\mathcal{R}^{nc}$ according to n_c; determine leap time τ^{nc} satisfying the leap condition for $\mathcal{R}^{nc}$

  $threshold = \frac{k}{a_0}$

  acceptedLeap =**false**

  repeat

    acceptedLeap = **true**

    if $\tau^{nc} > threshold$ **then**

      compute the firing time τ^c of the next reaction in $\mathcal{R}^c$ according to SSA

      $\tau = \min(\tau^{nc}, \tau^c)$

      foreach $R_j \in$ *Reactions* **do**

        if $R_j \in \mathcal{R}^{nc}$ **then**

          generate Poisson-distributed random number $k_j \sim poi(a_j(\vec{x})\tau)$

        if $R_j \in \mathcal{R}^c$ **then**

          $k_j = 0$

      if $\tau^{nc} > \tau^c$ **then**

        select reaction firing $R_\mu \in \mathcal{R}^c$ by SSA and set $k_\mu = 1$

      generate M Poisson-distributed random numbers $k_j \sim poi(a_j(\vec{x})\tau)$

      $X = X + \sum_{j=1}^{M} k_j \vec{v}_j$

      $t = t + \tau$

    else

      perform p SSA simulation steps for all reactions

    if $\exists S_i \in X, X_i < 0$ **then**

      roll back state $X = X - \sum_{j=1}^{M} k_j \vec{v}_j$ and time $t = t - \tau$

      $\tau = \alpha\tau$

      acceptedLeap = **false**

  until *acceptedLeap*

4.4.8.1.5 Discussion A non-critical reaction could achieve a negative population, so there is still a need to check it and roll back when necessary, but the frequency is decreased and is controlled through n_c. If $n_c \to \infty$ the modified τ-leaping converges to SSA, while is $n_c = 0$ it converges to the original τ-leaping.

4.4.8.2 Binomial τ-leaping

In binomial τ-leaping the Poisson-distributed random numbers are approximated with a binomial-distributed one. Let k_j be the number of firing of a reaction given τ and L_j the maximum number of permitted firings. The negative population problem is avoided if $k_j \leq L_j$, a requirement that can guaranteed by the binomial distribution.

4.4.8.2.1 The binomial distribution The number of firings during a leap is approximated as a series of L_j trials such that the probability that R_j fires during a trial is:

$$p_j = \frac{a_j(\vec{x})\tau}{L_j}$$

And the rejection probability is $1 - p_j$. The number of firings follows a binomial distribution $k_j \sim bin(p, L_j)$. This bounds R_j to not fire more than L_j times.

4.4.8.2.2 Species involved in many reactions This does not resolve the negative population problem because a reactant species may be involved in many reactions. To solve the problem and additional N-vector $\tilde{X}$ is used to track the population of reactants of reactions during a leap. At time t set $\tilde{X}(t) = X(t)$, each time a reaction fire $\tilde{X}(t)$ updates the population of its reactants: let $\vec{v_j}$ be the change in population of reactants by the firing of R_j, $\tilde{X} = \tilde{X} - k_j\vec{v_j}$. The maximum number of permitted firing of the next reaction is updated to reflect the change:

$$L_j = \min_{S_i \in reactants(R_j)} \left\lfloor \frac{\tilde{X}_i}{v_{ji}} \right\rfloor$$

4.4.8.2.3 Algorithm An implementation of the binomial τ-leaping method is outlined in algorithm 30.

Algorithm 30: `Binomial τ-leaping method()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$, the error control paramter $0 < \epsilon \ll 1$, the reduction factor $\alpha < 1$, the threshold paramter k and the exact number of exact SSA steps parameter p

Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$

$t = 0$

$\vec{X} = \vec{x}_0$

while $t < T_{\max}$ **do**

  $a_0 = 0$

  foreach $R_j \in$ *Reactions* **do**

    compute a_j

    $a_0 = a_0 + a_j$

  determine τ satisfying the leap condition

  $threshold = \frac{k}{a_0}$

  if $\tau^{nc} > threshold$ **then**

    $\tilde{X} = X$

    foreach $R_j \in$ *Reactions* **do**

      compute L_j fo reaction R_j using $\tilde{X}$

      $p_j = \frac{a_j \tau}{L_j}$

      generate binomial distributed random number $k_j \sim bin(P_j, L_j)$

      $\tilde{X} = \tilde{X} + k_j \vec{v_j}$

    $X = X + \sum_{j=1}^{M} k_j \vec{v}_j$

    $t = t + \tau$

  else

    perform p SSA simulation steps for all reactions

4.4.8.2.4 Discussion This methods imposes some constraint on the simulation: the leap time is restricted such that $\tau < \frac{L_j}{a_j(\vec{x})}$. Moreover the expected k_j is the same as the Poisson distribution, but the variance is smaller than the Poisson one:

$$Var[bin(p_j, L_j)] = a_j(\vec{x})\tau \left[1 - \frac{a_j(\vec{x})\tau}{L_j}\right]$$

Finally the order of execution of reactions depends on the current availability of reactants and affects the variance of k_j, biasing the trajectories. To limit the bias the order of execution could be chosen randomly.

4.4.8.3 Implicit τ-leaping

In the implicit τ-leaping method performance are improved for biochemical reactions with highly diverse reactions rates. The leap time is very small, yielding a small number of firings of each reaction

in a leap, degrading simulation efficiency. This is improved by allowing to choose an arbitrary large τ value through an implicit approximation form.

4.4.8.3.1 Correction term The state update formula is complemented with a correction term to account for the change in propensity of each reaction over $[t, t+\tau[$. If the propensity of a reaction changes after the leap, the number of firing is changed accordingly. So k_j is the sum of $poi(a_j(\vec{x})\tau)$ and the second part is a zero-mean random variable $[a_j(X(t+\tau)) - a_j(\vec{x})]\tau$. The propensity $a_j(X(t+\tau))$ in the correction part is a function of the unknown random state, the number of firing during the leap is:

$$k_j = poi(a_j(\vec{x})\tau) + [a_j(X(t+\tau)) - a_j(\vec{x})]\tau$$

And the state update becomes:

$$X(t+\tau) = \vec{x} + \sum_{j=1}^{M} \left(poi(a_j(\vec{x})\tau) + [a_j(X(t+\tau)) - a_j(\vec{x})]\tau\right) \vec{v}_j$$

A root finding method like the Newton-Raphson can be applied to find the next state and the population of species in the unknown vector has to be cast to the nearest integer.

4.4.8.3.2 Algorithm An implementation of the implicit τ-leaping method is outlined in algorithm 31.

Algorithm 31: `Implicit τ-leaping method()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$ and the leap τ
Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$
$t = 0$
$\vec{X} = \vec{x}_0$
choose a leap τ value
while $t < T_{\max}$ **do**
    $a_0 = 0$
    foreach $R_j \in$ *Reactions* **do**
        compute a_j
        $a_0 = a_0 + a_j$
    generate M Poisson distributed random number $k_j \sim poi(a_j\tau)$
    $X(t+\tau) = \vec{x} + \sum_{j=1}^{M} (k_j + [a_j(X(t+\tau)) - a_j]\tau)\, \vec{v}_j$
    $t = t + \tau$

4.4.8.3.3 Discussion Since τ is fixed it is a time-stepping algorithm. The greater efficiency is achieved from being able to choose a large leap for each leap. This tends to dampen fluctuation of species. To solve this downshifting could be employed: the τ leap is interfaced with a sequence of smaller steps simulated using SSA to retain the damped fluctuation while still achieving the computational efficiency of the implicit approach.

4.5 $\mathbf{k}_\alpha$-leaping method

The K_α-leaping method is a variant of τ leaping. It leaps down by a predetermined number of firing for a predetermined R_α and may be more convenient in some circumstance. Let k_α be the number of firing of R_α given $X(t) = \vec{x}$. Let τ be the length so that at $t + \tau$ the k_αth firing of R_α occurs. This method assumes that the leap condition is satisfied for all reactions in $[t, t + \tau[$. The number of firings k_j for R_j, $j \neq \alpha$ in the time interval still follows $poi(a_j(\vec{x})\tau)$.

4.5.1 Computing the time length

To compute τ consider R_α at time t. Each firing of R_j is an exponentially distributed random number $exp(a_\alpha(x))$. τ in which there are k_α firing of R_α is the sum of k_α exponential distribution with the same rate $exp(a_\alpha(x))$. So the time length τ is an Erlang random number $erlang(k_\alpha, a_\alpha)$ with shape k_α and rate a_α.

4.5.2 Selecting the number of firings

To select the largest k_α satisfying the leap condition a control parameter $0 < \epsilon \ll 1$ is used. Let τ^{temp} be the largest time selected by the τ selection procedure. $k_\alpha \sim poi(a_\alpha(\vec{x})\tau^{temp})$, so the average number of firing is:

$$k_\alpha = [a_\alpha(\vec{x})\tau^{temp}]$$

Where $[-]$ is the truncation operator. k_α is then used for the simulation and τ^{temp} is discarded.

4.5.3 Algorithm

An implementation of the k_α leaping method is outlined in algorithm 32.

Algorithm 32: k_α-`leaping method()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$, the error control paramter $0 < \epsilon \ll 1$

Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$

$t = 0$

$\vec{X} = \vec{x}_0$

while $t < T_{\max}$ **do**

  acceptedLeap =**false**

  repeat

    acceptedLeap = **true**

    compute a_j for each R_j

    determine k_α satisfying the leap condition

    generate $\tau \sim erlang(k_\alpha, a_\alpha)$

    foreach $R_j, j \neq \alpha$ **do**

      generate Poisson-distributed random numbers $k_j \sim poi(a_j(\vec{x})\tau)$

    $X = X + \sum_{j=1}^{M} k_j \vec{v}_j$

    $t = t + \tau$

    if $\exists$ *a species in* X *whose population* $X_i < 0$ **then**

      roll back state $X = X - \sum_{j=1}^{M} k_j \vec{v}_j$ and time $t = t - \tau$

      reduce k_α

      acceptedLeap = **false**

  until *acceptedLeap*

4.5.4 Discussion

This method has the same performance as τ-leaping, but it provides a more flexible way to choose the reaction and to adjust its number of firings to enforce the leap condition. In particolar if R_α is selected such that $a_\alpha = \max_{j=1}^{M} a_j$ and bounds k_α to an upper value, the number of firings of the other reactions are bounded by this, making the leap condition easier to enforce by tuning this bound.

4.5.5 K-leaping method

The K-leaping method or R-leaping method is a generalization of the k_α-leaping. It leaps down the simulation by total K firings of reactions chosen satisfying the leap condition. Let τ be the time length to that there are K reaction firings in $[t, t + \tau[$, given $X(t) = \vec{x}$. The firing time of a reaction is $exp(a_0)$, τ is the sum of K exponential distribution with rate a_0 and so is $erlang(K, a_0(\vec{x}))$.

4.5.5.1 Number of firing

Let $\mathbb{P}\{k_1, \ldots, k_M | K, \tau, \vec{x}, t\}$ be the joint probability that there are k_j firings for each R_j given the $X(t) = \vec{x}$. The explicit formula under the leap assumption can be derived considering that the probability that R_j fires in the interval is:

$$p_j = \frac{a_j}{a_0}$$

Moreover:

$$\sum_{j=1}^{M} k_j = K$$

The joint probability is then a multinomial distribution $multi(K, p_1, \ldots, p_M)$, with formula:

$$\mathbb{P}\{k_1, \ldots, k_M | K, \tau, \vec{x}, t\} = \frac{K!}{k_1! \cdots k_M!} p_1^{k_1} \cdots p_m^{k_M}$$

The number of firings is obtained by sampling $multi(K, p_1, \ldots, p_M)$.

4.5.5.2 Selecting K

To select the largest K satiating the leap condition with control parameter $0 < \epsilon \ll 1$, consider the preleap selection whit bounding $\Delta a_j(\vec{x}) \leq \epsilon a_0(\vec{x})$. The expected value and variance can be obtained using the properties of the multinomial distribution:

- $\mathbb{E}[k_j] = Kp_j = \frac{Ka_j(\vec{x})}{a_0(\vec{x})}$.
- $Var[k_j] = Kp_j(1 - p_j) = \frac{Ka_j(\vec{x})}{a_0(\vec{x})} \left(1 - \frac{a_j(\vec{x})}{a_0(\vec{x})}\right)$.
- $cov(k_j, k_m) = -Kp_j p_m = -\frac{Ka_j(\vec{x})a_m(\vec{x})}{a_0^2(\vec{x})}$.

Therefore:

$$\mathbb{E}[\Delta a_j(\vec{x})] \approx \sum_{l=1}^{M} f_{jl}(\vec{x}) \mathbb{E}[k_l] = K \frac{\mu_j(\vec{x})}{a_0(\vec{x})}$$

And:

$$Var[\Delta a_j(\vec{x})] \approx \sum_{l=1}^{M} f_{jl}^2(\vec{x}) Var[k_l] + \sum_{l=1}^{M} \sum_{l'=1}^{M} f_{jl}(\vec{x}) f_{jl'}(\vec{x}) cov(k_l, k_{l'}) =$$
$$= K \left(\frac{\sigma_j^2(\vec{x})}{a_0(\vec{x})} - \frac{\mu_j^2(\vec{x})}{a_0^2(\vec{x})} \right)$$

Where $\mu_j(\vec{x})$ and σ_j^2 are defined as in the preleap selection for the τ-leaping method. The largest K is then:

$$K = \left[a_0(\vec{x}) \min_{k=1}^{M} \left(\frac{\epsilon a_0(\vec{x})}{2|\mu_j(\vec{x})|}, \frac{(\epsilon a_0(\vec{x}))^2}{4\left(\sigma_j^2(\vec{x}) - \frac{\mu_j^2(\vec{x})}{a_0(\vec{x})}\right)} \right) \right]$$

Where $[-]$ is the truncation operator.

4.5.5.3 Algorithm

An implementation of the K-leaping method is found in algorithm 33.

Algorithm 33: K-`leaping method()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$, the error control paramter $0 < \epsilon \ll 1$

Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$

$t = 0$

$\vec{X} = \vec{x}_0$

while $t < T_{\max}$ **do**

  acceptedLeap =**false**

  repeat

    acceptedLeap = **true**

    compute a_j for each R_j

    $a_0 = \sum_{j=1}^{M} a_j$

    determine K satisfying the leap condition

    generate $\tau \sim erlang(k_\alpha, a_\alpha)$

    $p_j = \frac{a_j}{a_0} \forall j = 1, \ldots, M$

    generate $k_j \sim multi(K, p_1, \ldots, p_M) \forall j = 1, \ldots, M$

    $X = X + \sum_{j=1}^{M} k_j \vec{v}_j$

    $t = t + \tau$

    if $\exists$ *a species in* X *whose population* $X_i < 0$ **then**

      roll back state $X = X - \sum_{j=1}^{M} k_j \vec{v}_j$ and time $t = t - \tau$

      reduce K

      acceptedLeap = **false**

  until *acceptedLeap*

4.5.5.4 Discussion

The K-leaping differs from the k_α method in two points:

- τ is generated from $erlang(K, a_0)$.
- $k_j \sim multi(K, p_1, \ldots, p_m)$ with $p_j = \frac{a_j}{a_0}$

This method handles the negative population easier than τ-leaping, improving the accuracy. K is a deterministic number and represent an upper bound on the number of firings on each reaction. If the negative population problem happens, decreasing the K reduces the change of it.

4.6 Chemical Langevin method

The chemical Langevin method is an approximation of the τ-leaping. Let τ be the time satisfying the leap condition. k_j of R_j in the leap follows $poi(a_j(\vec{x})\tau)$. Assume that the expected value of the Poisson distribution is large enough. CLE assumes that there exists a small $\tau > 0$ such that the change in propensity for each R_j during $[t, t+\tau[$ is negligibly small and:

$$a(\vec{x})\tau \gg 1$$

Now $poi(a_j(\vec{x})\tau)$ can be approximated to a normal distribution with same mean and variance:

$$poi(a_j(\vec{x})\tau) \approx norm(a_j(\vec{x})\tau, a_j(\vec{x})\tau) = a_j(\vec{x})\tau + \sqrt{a_j(\vec{x})\tau}norm(0,1)$$

4.6.1 State update

The state update after the leap is approximated by:

$$\begin{aligned} X(t+\tau) &\approx \vec{x} + \sum_{j=1}^{M} poi(a_j(\vec{x})\tau)\vec{v}_j \approx \\ &\approx \vec{x} + \sum_{j=1}^{M} a_j(\vec{x})\vec{v}_j\tau + \sum_{j=1}^{M} \sqrt{a_j(\vec{x})\tau}norm(0,1)\vec{v}_j \end{aligned}$$

This is the chemical Langevin equation CLE. The state is no longer an integer vector due to the square root.

4.6.2 Algorithm

An implementation of the CLE is outlined in algorithm 34.

Algorithm 34: `Chemical Langevin method (CLE)()`

Input: a biochemical reaction network of M reactions in which each reaction R_j, $j = 1, \ldots, M$ is accompanied with the state change vector $\vec{v}_j$ and the propensity a_j, the initial state $\vec{x}_0$ at time 0 and the simulation ending time $T_{\max}$, the error control paramter $0 < \epsilon \ll 1$

Output: a trajectory of the biochemical reaction network, which is a collection of states $X(t)$ for time $0 \leq t \leq T_{\max}$

$t = 0$

$\vec{X} = \vec{x}_0$

while $t < T_{\max}$ **do**

  compute a_j for each R_j

  determine τ satisfying the leap condition and $a_j \tau \gg 1$

  generate M unit normal-distributed random number $n_j \sim norm(0, 1)$

  $X = X + \sum_{j=1}^{M} a_j \tau \vec{v}_j + \sum_{j=1}^{M} n_j \vec{v}_j \sqrt{a_j \tau}$

  $t = t + \tau$

4.6.3 Discussion

This method does not handle the negative population explicitly, but it is relaxed because the population of species is very large in order to satisfy the CLE assumption and can be approximated to a continuous state. The CLE method is faster because generating random normal distributed number is easier than the Poisson distribution and $k_j \gg 1$.

Chapter 5

Deterministic simulations

5.1 introduction

When the last noise term of equation:

$$X(t+\tau) \approx \vec{x} + \sum_{j=1}^{M} a_j(\vec{x})\vec{v}_j\tau + \sum_{j=1}^{M} \sqrt{a_j(\vec{x})\tau} norm(0,1)\vec{v}_j$$

Becomes negligibly small compared with the second one, a deterministic way to computing the dynamics of a system can be applied. This happens in the limiting case $a_j(\vec{x})\tau \to \infty$ and the deterministic simulation produces an average behaviour of the system close to the one obtained by averaging an infinite number of stochastic simulations of the system starting from the same initial state. Deterministic simulation are faster than exact stochastic ones because reaction events are executed as the simultaneous application of a set of reactions. A single run is sufficient because stochasticity of the system is not considered any more. One way to simulate a biological system according to a deterministic approach is to translate it into a set of ordinary differential equations or ODEs.

5.2 From biochemical reactions to ODEs

Ordinary differential equations can be used to simulate a biochemical system that satisfied the spatial homogeneity and the continuum hypothesis.

5.2.1 Starting hypothesis

5.2.1.1 Continuum hypothesis

A biochemical system satisfies the continuum hypothesis if the number of molecules for each species is large enough to safely approximate molecular abundances by concentrations that vary continuously.

5.2.1.2 Effect of starting hypothesis

Spatial homogeneity allows to randomize spatial information: the rate of each reaction is space independent. The continuum hypothesis allows to approximate discrete changes in molecule number

by continuous changes in concentration, moreover individual reactions are infinitesimal changes in molecule abundance. This holds for species with molecules counts of thousands or more. In the cases where molecule abundance is low, its changes will have to be treated like discrete steps in population size and stochastic simulation should be preferred.

5.2.2 Law of mass action

When the two hypothesis are satisfied, a biochemical reaction system can be translated into a set of ODEs by the law of mass action. The law of mass action states that the deterministic rate of a chemical reaction is proportional to the product of the concentrations of its reactants. Now let $[A] = \frac{\#A}{N_A V}$ be the molar concentration of species A in a chemical volume V and N_A Avogadro's number. Some example of conversion of chemical reactions into ODEs are outlined in table 5.1.

Reaction type	Reaction	Rate	ODEs
Zero-order reaction	$\emptyset \xrightarrow{k} A$	k	$\frac{d[A]}{dt} = k$
First-order reaction	$A \xrightarrow{k} B$	$k[A]$	$\frac{d[A]}{dt} = -k[A]; \frac{d[B]}{dt} = k[A]$
Second-order reaction	$A + B \xrightarrow{k} C$	$k[A][B]$	$\frac{d[A]}{dt} = \frac{d[B]}{dt} = -k[A][B]; \frac{d[C]}{dt} = k[A][B]$
Second-order reaction same reactant	$A + A \xrightarrow{k} B$	$k[A]^2$	$\frac{d[A]}{dt} = -k[A]^2; \frac{d[B]}{dt} = k[A]^2$
Third-order reaction	$A + B + C \xrightarrow{k} D$	$k[A][B][C]$	$\frac{d[A]}{dt} = \frac{d[B]}{dt} = \frac{d[C]}{dt} = -k[A][B][C]$ $\frac{d[D]}{dt} = k[A][B][C]$

Table 5.1: Conversion of biochemical reactions into ODEs

Taking into consideration that the deterministic rate k is not the stochastic reaction rate constant c_j but will be computed as in table 5.2.

Reaction type	Reaction	Rate	Unit
Zero-order reaction	$\emptyset \xrightarrow{k} A$	$k = \frac{c}{N_A V}$	$concentration \cdot time^{-1}$
First-order reaction	$A \xrightarrow{k} B$	$k = c$	$time^{-1}$
Second-order reaction	$A + B \xrightarrow{k} C$	$k = cN_A V$	$concentration^1 time^{-1}$
Second-order reaction same reactant	$A + A \xrightarrow{k} B$	$k = \frac{cN_A V}{2}$	$concentration^1 time^{-1}$
Third-order reaction	$A + B + C \xrightarrow{k} D$	$k = c(N_A V)^2$	$concentration^2 time^{-1}$

Table 5.2: Conversion of biochemical reactions into ODEs

5.2.3 Building the set of ODEs

Consider now a biochemical reaction system with N species $S_1, \ldots, S_N$ interacting trough M reactions $R_1, \ldots R_M$ and a stoichiometric matrix:

$$\vec{v} = \vec{v}^{+} - \vec{v}^{-}$$

The deterministic rate constant of each reaction is:

$$k_j = \frac{c_j (N_A V)^{Order_j - 1}}{\prod_{i=1}^{N} \vec{v_{ij}}!}$$

Where $Order_j$ is the order of reaction R_j. The set of ODEs modelling the evolution of the species is:

$$\frac{d[S_i]}{dt} = \sum_{j=1}^{M} \left(k_j \vec{v}_{ji} \prod_{l=1}^{N} [S_l]^{\vec{v}_{jl}} \right), \forall i = 1, \ldots, N$$

Other methods can be used to approach this translation like the Michaelis-Menten kinetics or the Hill kinetics. The former can be used to quantify cooperative binding, the phenomena of binding of a ligand to a macromolecule enhancing when another molecule is attached to the same macro-one.

5.2.4 Michaelis-Menten kinetics

Michaelis-Menten (MM) kinetics are used to model enzymatic reactions:

$$A \xrightarrow{E} B$$

Enzyme accelerate the reaction, and to transform this reaction into a law of mass action it needs to be expanded into:

$$A + E \underset{k_2}{\overset{k_1}{\rightleftharpoons}} AE \xrightarrow{k_3} B + E$$

Describing the process more accurately. The translation to a set of ODE of the reaction requires the definition of four differential equation, where also $[E]$ and $[EA]$ are considered. The MM kinetics allow to simplify the model by reducing the number of equation, so the equation is translated into two odes considering the variation of concentration of A and B:

$$\frac{d[A]}{dt} = -\frac{d[B]}{dt} = -V_{MAX} \frac{[A]}{K_M + [A]}$$

Where:

- V_{MAX} is the maximum velocity of the enzymatic reaction.
- K_M, Michaelis constant, is the concentration of the substrate at which the reaction rate is half of its maximum.

The effect of the enzyme is modelled, where:

$$K_M = \frac{k_2 + k_3}{k_1} \qquad \wedge \qquad V_{MAX} = k_{cat}[E_T]$$

Where $[E_T]$ is the enzyme available to the system. This kinetics can be used also in the context of stochastic simulation, so that the propensity for this type of reactions will be:

$$a(\vec{x}) = \frac{V_{MAX} A}{K_M + A}$$

Where V_{MAX} and K_M are scaled to consider molecule abundances.

5.3 Numerical solution of ODEs

The simulation of a system of ODE is addressed by solving the initial value or Cauchy problem. This corresponds to finding the solution of a set of differential equations that satisfies the initial condition corresponding to the initial concentration of the species. An exact solution is usually to complex, so suitable numerical methods need to be used. This will produce approximations of the solution at specified time points. Some interpolation methods can be used to obtain intermediate values. Even when an exact solution if found, the dynamics of the biochemical system is an approximation.

5.3.1 Finding a solution

Consider the system with N species:

$$\frac{d[X]}{dt} = \vec{F}(t, [X])$$

Where:

- $\vec{F} : \mathbb{R} \times \mathbb{R}^N \rightarrow \mathbb{R}^N$ is the vector of N functions providing the time derivatives of species concentration.
- $[X]$ is the current state of the system expressed in molecular concentrations.

Let $I = (0, T_{\max})$ be the integration interval of the system and:

$$t_n = nh, h > 0 \wedge n = 0, \ldots, N_h$$

Be the sequence of discretization of I into subintervals $I_n = [t_n, t_{n+1}]$, where:

- N_h is the maximum integer such that $t_{N_h} \leq T_{max}$.
- h is the discretization stepsize.

Numerical methods compute a sequence of states $[X_n]$. Approximating the trajectory in terms of concentrations along the time steps starting from $[X_0]$. These methods can be explicit or implicit. They are called explicit if $[X_{n+1}]$ can be computed directly from the previous state $[X_k]$. They are called implicit if $[X_{n+1}]$ depends implicitly on itself through $\vec{F}$.

5.3.2 Forward/Backward Euler method

The Forward Euler method is an explicit method, while the backward one is an implicit. The are:

$$\text{Forward Euler: } [X_{n+1}] = [X_n] + h\vec{F}(t_n, [X_n])$$
$$\text{Backward Euler: } [X_{n+1}] = [X_n] + h\vec{F}(t_{n+1}, [X_{n+1}])$$

Comparing this with the chemical Langevin equation, when stochasticity is negligible, the CLE reduces a forward Euler with $\tau = h$.

5.3.2.1 Forward Euler algorithm

An implementation of the forward Euler algorithm is found in algorithm 35.

Algorithm 35: `Forward Euler method()`

Input: a system of ODEs $\frac{d[X]}{dt} = \vec{F}(t, [X])$ corresponding to a biochemical reaction system, the initial state $[X_0]$ of the system with species concentrations at time 0, the simulation ending time $T_{\max}$ and the discretization stepsize h
Output: a trajectory of the biochemical system expressed in terms of molecule concentrations with discretization stepsize h
$t = 0$
$[X] = [X_0]$
while $t < T_{\max}$ **do**
    $[X] = [X] + h \cdot \vec{f}(t, [X])$
    $t = t + h$

5.3.2.2 Backward Euler algorithm

An implementation of the backward Euler algorithm is found in algorithm 36.

Algorithm 36: `Backward Euler method()`

Input: a system of ODEs $\frac{d[X]}{dt} = \vec{F}(t, [X])$ corresponding to a biochemical reaction system, the initial state $[X_0]$ of the system with species concentrations at time 0, the simulation ending time $T_{\max}$ and the discretization stepsize h
Output: a trajectory of the biochemical system expressed in terms of molecule concentrations with discretization stepsize h
$t = 0$
$[X] = [X_0]$
while $t < T_{\max}$ **do**
    estimate $[X_{new}] = [X] + h \cdot \vec{F}(t, [X])$
    $t = t + h$
    $[X] = [X] + h \cdot \vec{f}(t, [X_{new}])$

5.3.2.3 Discussion

Implicit methods are less intuitive because they need to compute an estimation of the state. This can be done by computing a first approximation of the next state which is used then to compute the actual one.

5.4 Improving the accuracy of numerical methods

The accuracy of the computation depends on the discretization stepsize and on the properties of the numerical method. The general form of one step for an explicit method is:

$$[X_{n+1}] = [X_n] + h\mathbb{F}(t_n, [X_n], \vec{F}(t_n, [X_n])lh) + h\epsilon_{n+1}(h)$$

Where:

- $n = 0, \ldots, N_h$.
- $h > 0$.
- $\mathbb{F}$ is the increment function.
- $\epsilon_{n+1}(h)$ is the local truncation error LTE at t_{n+1} of the numerical method. This provides a measure of how distant the estimation is from the exact value.

A global truncation error is required to evaluate the accuracy of a numerical method.

5.4.1 Global truncation error

Consider a numerical method with local truncation error $\epsilon_{n+1}(h)$. The global truncation error is:

$$\epsilon(h) = \max |\epsilon_{n+1}(h)|, n = 0, \ldots, N_h$$

5.4.2 Consistency of a numerical method

A numerical method with global truncation error $\epsilon(h)$ is consistent with the initial value problem if:

$$\lim_{h \to 0} \epsilon(h) = 0$$

From now on only consistent numerical methods will be considered.

5.4.3 Order of a numerical method

A numerical method with global truncation error $\epsilon(h)$ has order p if:

$$\forall t \in]0, T_{\max}[: \epsilon(h) = O(h^p), h \to 0$$

A Taylor expansion shows that the forward Euler has order 1. To increase the accuracy of a simulation the discretization stepsize or increase the order of the numerical methods need to be decreased. Decreasing the discretization stepsize implies to compute more simulation steps, while increasing the method order the complexity of each step increases.

5.4.4 Heun method

An example of a second order numerical method is the implicit trapezoidal or Crank-Nicolson method, which updates the system by:

$$[X_{n+1}] = [X_n] + \frac{h}{2}[\vec{F}(t_n, [X_n]) + \vec{F}(t_{n+1}, [X_{n+1}])]$$

The gain in accuracy is balanced by the increased complexity in the update formula, which requires the evaluation of two $\vec{F}$ at each step. This can be transformed into the explicit alternative Heun method, which updates the system by:

$$[X_{n+1}] = [X_n] + \frac{h}{2}[\vec{F}(t_n, [X_n]) + \vec{F}(t_{n+1}, [X_n] + h\vec{F}(t_n, [X_n]))]$$

5.4.4.1 Algorithm

An implementation of the Heun algorithm can be found in algorithm 37.

Algorithm 37: `Heun method()`

Input: a system of ODEs $\frac{d[X]}{dt} = \vec{F}(t, [X])$ corresponding to a biochemical reaction system, the initial state $[X_0]$ of the system with species concentrations at time 0, the simulation ending time $T_{\max}$ and the discretization stepsize h
Output: a trajectory of the biochemical system expressed in terms of molecule concentrations with discretization stepsize h
$t = 0$
$[X] = [X_0]$
while $t < T_{\max}$ **do**
    $[X] = [X] + \frac{h}{2}[\vec{F}(t, [X]) + \vec{F}(t, [X] + h\vec{F}(t, [X]))]$
    $t = t + h$

5.4.5 Runge-Kutta methods

The Runge-Kutta methods are a family of numerical methods that can be written as:

$$[X_{n+1}] = [X_n] + h\mathbb{F}(t_n, [X_n], h; \vec{F})$$

Where:

- $n = 0, \ldots, N_h$.
- $h > 0$.
- $\mathbb{F}$ is the increment function of the method.

In particular:

$$\mathbb{F}(t_n, [X_n], h; \vec{F}) = \sum_{i=1}^{s} b_i K_i$$

$$K_i = \vec{F}\left(t_n + c_i h, [X_n] + h\sum_{j=1}^{s} a_{ij} K_j\right)$$

Whit s being the number of stages of the method and a_{ij}, b_i and c_i are suitable numbers that characterize the RK method. These method can be explicit or implicit depending on the values of a_{ij}. The Heun method is an explicit RK method, because a_{12} and a_{22} are zeros. The number of stages and the order of the methods are related: the minimum number $s_{\min}$ required to get an explicit RK method of corresponding order is described in table 5.3.

Order	1	2	3	4	5	6	7	8
s_{min}	1	2	3	4	6	7	9	11

Table 5.3: RK order and s_{min} relationship

4 is the maximum number of stages for which the order is not less than $s_{\min}$. For this a four-stage explicit RK method is the more convenient way to solve an initial-value problem.

5.4.5.1 Fourth order RK method

An example of an update of a fourth order RK method is:

$$[X_{n+1}] = [X_n] + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + k_4)$$

Where;

- $K_1 = \vec{F}(t_n, [X_n])$.
- $K_2 = \vec{F}(t_n + \frac{h}{2}, [X_n] + \frac{h}{2}K_1)$.
- $K_3 = \vec{F}(t_n + \frac{h}{2}, [X_n] + \frac{h}{2}K_2)$.
- $K_4 = \vec{F}(t_{n+1}, [X_n] + hK_3)$.

This is called RK4 and is one of the most used numerical methods for deterministic simulations.

5.4.5.1.1 Algorithm An implementation of the RK4 method can be found at algorithm 38.

Algorithm 38: `RK4 method()`

Input: a system of ODEs $\frac{d[X]}{dt} = \vec{F}(t, [X])$ corresponding to a biochemical reaction system, the initial state $[X_0]$ of the system with species concentrations at time 0, the simulation ending time $T_{\max}$ and the discretization stepsize h
Output: a trajectory of the biochemical system expressed in terms of molecule concentrations with discretization stepsize h
$t = 0$
$[X] = [X_0]$
while $t < T_{\max}$ **do**
    $K_1 = \vec{F}(t, [X])$
    $K_2 = \vec{F}(t + \frac{h}{2}, [X] + \frac{h}{2}K_1)$
    $K_3 = \vec{F}(t + \frac{h}{2}, [X] + \frac{h}{2}K_2)$
    $K_4 = \vec{F}(t + h, [X] + hK_3)$
    $[X] = [X] + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + k_4)$

5.5 Multistep methods

A numerical method for the approximation of the initial-value problem is a one step method if $\forall n \geq 0$, the computation of $[X_{n+1}]$ depends only on $[X_n]$, otherwise the scheme is called a multistep method. Multistep (MS) schemes require one functional evaluation at each step and their accuracy can be increased at the expense of increasing the number of steps. Thy can be implicit or implicit and have an order of accuracy.

5.5.1 Linear multistep numerical method

A linear $(s + 1)$-step method is a multistep method whose update formula fits the scheme:

$$[X_{n+1}] = \sum_{j=0}^{s} a_j [X_{n-j}] + h \sum_{j=0}^{s} b_j \vec{F}(t_{n-j}, [X_{n-j}]) + hb_{-1}\vec{F}(t_{n+1}, [X_{n+1}])$$

Where:

- $n \geq s \geq 0$.
- a_j, b_j are numbers that characterize the method. When $b_{-1} = 0$ the method is explicit, otherwise implicit.

5.5.1.1 Midpoint method

The midpoint method is a second order, two step linear explicit method, which updates the system by:

$$[X_{n+1}] = [X_{n-1}] + 2h\vec{F}(t_n, [X_n])$$

They rely on the fact that the formula depend on some previous state of the system to increase accuracy. History dependency does not require additional functional evaluation because previous state are stored during the simulation. These reduces the simulation runtime, while increasing the complexity in space of the algorithm. The length of the time series of states that needs to be stored depends on the update formula and increases with the order.

5.5.1.1.1 Algorithm An implementation of the midpoint method can be found in 39.

Algorithm 39: `Midpoint method()`

Input: a system of ODEs $\frac{d[X]}{dt} = \vec{F}(t, [X])$ corresponding to a biochemical reaction system, the initial state $[X_0]$ of the system with species concentrations at time 0, the simulation ending time $T_{\max}$ and the discretization stepsize h
Output: a trajectory of the biochemical system expressed in terms of molecule concentrations with discretization stepsize h
$t = 0$
$[X_{old}] = [X_0]$
$[X] = [X_{old}] + \frac{h}{2}\left[\vec{F}(t, [X_{old}]) + \vec{F}(t + h, [X_{old}] + h \cdot \vec{F}(t, [X_{old}]))\right]$
$t = h$
while $t < T_{\max}$ **do**
    $[X_{new}] = [X_{old}] + 2h \cdot \vec{f}(t, [X])$
    $[X_{old}] = [X]$
    $[X] = [X_{new}]$
    $t = t + h$

5.5.1.1.2 Discussion In order to preserve the order of accuracy of the MS algorithm, the one-step method used in the preliminary phase must have at least the same order of the MS method.

5.5.1.2 Simpson method

The Simpson method is a two-step implicit linear model which updates the system by:

$$[X_{n+1}] = [X_{n-1}] + \frac{h}{3}[\vec{F}(t_{n-1}, [X_{n-1}]) + 4\vec{F}(t_n, [X_n]) + \vec{F}(t_{n+1}, [X_{n+1}])]$$

5.5.1.3 General linear multistep algorithm

An implementation of a generic $(s + 1)$-step method that requires a preliminary phase where a one-step method is used to compute the first s step of the simulation can be found in algorithm 40.

Algorithm 40: `Linear` $(s+1)$`-step method()`

Input: a system of ODEs $\frac{d[X]}{dt} = \vec{F}(t, [X])$ corresponding to a biochemical reaction system, the initial state $[X_0]$ of the system with species concentrations at time 0, the simulation ending time $T_{\max}$ and the discretization stepsize h and the coefficient values a_j, b_j
Output: a trajectory of the biochemical system expressed in terms of molecule concentrations with discretization stepsize h
$t = 0$
$[X_{old}] = [X_0]$
compute the first s steps of the dynamics by a one-step numerical method of order at least equal to the implemented multistep method
while $t < T_{\max}$ **do**
    if $b_{-1} \neq 0$ **then**
        approximate $[X_{t+h}]$
    $[X] = \sum_{j=0}^{s} a_j[X_{n-j}] + h \sum_{j=0}^{s} b_j \vec{F}(t_{n-j}, [X_{n-j}]) + hb_{-1}\vec{F}(t_{n+1}, [X_{n+1}])$
    $t = t + h$

5.5.2 Adams methods

Adams methods are linear multistep methods that update the system state:

$$[X_{n+1}] = [X_n] + h \sum_{j=-1}^{s} b_j \vec{F}(t_{n-j}, [X_{n-j}])$$

Where:

- $n \geq s \geq 0$.
- b_j are numbers that characterize the method. $b_{-1} = 0$ implies that the method is explicit and is called the Adams-Bashforth method, when it is implicit is the Adams-Moulton method.

5.5.3 BDF methods

BDF methods are implicit linear multistep methods that update the system by:

$$[X_{n+1}] = \sum_{j=0}^{s} a_j[X_{n-j}] + hb_{-1}\vec{F}(t_{n+1}, [X_{n+1}])$$

Where:

- $n \geq s \geq 0$.
- a_j and $b_{-1} \neq 0$ fully characterize the system.

5.6 Adaptive methods

Adaptive numerical methods change the discretization stepsize at each simulation step so that h remains close to the greatest value which keeps the global truncation error within a bound. They do

not set h but a threshold for a maximum value of the global truncation error. The output is a time series with time points not equally spaced. They require the implementation of an error estimator. Here one step methods are considered, particularly RK schemes.

5.6.1 Estimating the local truncation error

RK schemes are well suited to provide an efficient estimator of the local truncation error and to adapt the stepsize according to it. An a posteriori error estimator is used to estimate the local truncation error. This can be built in two ways:

- By comparing the output of the same RK method with two different stepsizes $2h$ and h.
- By comparing the output of two RK methods of different order.

In both cases the error estimator computes a posteriori, after the simulation step. The first case a considerable increase of computational cost is required, while the second do not by using two different RK methods with s stages of order p and $p+1$, which share the same set of values.

5.6.2 Runge-Kutta Fehlberg

The Runge-Kutta Fehlberg method of fourth order, or RK45 method updates the system as:

$$[X_{n+1}] = [X_n] + \frac{25}{216}K_1 + \frac{1408}{2565}K_3 + \frac{2187}{4104}K_4 - \frac{1}{5}K_5$$

Coupled with a fifth order RK method:

$$[\tilde{X}_{n+1}] = [X_n] + \frac{16}{135}K_1 + \frac{6656}{12825}K_3 + \frac{28561}{56430}K_4 = \frac{9}{50}K_5 + \frac{2}{55}K_6$$

$K_1, \ldots, K_6$ are shared between methods and are computed as:

$$\begin{aligned}
K_1 &= h\vec{F}(t_n, [X_n]) \\
K_2 &= h\vec{F}\left(t_n + \frac{h}{4}, [X_n] + \frac{1}{4}K_1\right) \\
K_3 &= h\vec{F}\left(t_n + \frac{3h}{8}, [X_n] + \frac{3}{32}K_1 + \frac{9}{32}K_2\right) \\
K_4 &= h\vec{F}\left(t_n + \frac{12h}{13}, [X_n] + \frac{1932}{2197}K_1 - \frac{7200}{2197}K_2 + \frac{7296}{2197}K_3\right) \\
K_5 &= h\vec{F}\left(t_n + h, [X_n] + \frac{439}{216}K_1 - 8K_2 + \frac{3680}{513}K_3 - \frac{845}{4104}K_4\right) \\
K_6 &= h\vec{F}\left(t_n + \frac{h}{2}, [X_n] - \frac{8}{27}K_1 + 2K_2 - \frac{3544}{2565}K_3 + \frac{1859}{4104}K_4 - \frac{11}{40}K_5\right)
\end{aligned}$$

The fourth order version is used to compute the dynamics of the system, while the fifth order one is used to estimate the local truncation error:

$$\Delta_{n+1} = \frac{\left|[\tilde{X}_{n+1}] - [X_{n+1}]\right|}{h}$$

The error estimate is compared to the error threshold ϵ_t. If it is less the local truncation error is assumed smaller than the threshold and the state is accepted, in the other case the new state is rejected and evaluated again using a smaller h. In both cases h is updates as:

$$h_{n+1} = h_n \sigma$$
$$\sigma = \left(\frac{\epsilon_t}{2\Delta_{n+1}} \right)^{\frac{1}{4}} \approx 0.84 \left(\frac{\epsilon_t}{\Delta_{n+1}} \right)^{\frac{1}{4}}$$

This is derived from $\sigma = \left(\frac{\epsilon_t}{\Delta_{n+1}} \right)^{\frac{1}{p}}$, defining how to update h for an adaptive one step method of order p.

5.6.2.1 Algorithm

An implementation of the RK45 method can be found at algorithm 41.

Algorithm 41: `RK45 algorithm()`

Input: a system of ODEs $\frac{d[X]}{dt} = \vec{F}(t, [X])$ corresponding to a biochemical reaction system, the initial state $[X_0]$ of the system with species concentrations at time 0, the simulation ending time $T_{\max}$, the initial discretization stepsize h_0 and a threshold for the maximum local truncation error ϵ_t

Output: a trajectory of the biochemical system expressed in terms of molecule concentrations with discretization stepsize h_0

$t = 0$

$[X] = [X_0]$

$h = h_0$

while $t < T_{\max}$ **do**

  $K_1 = h\vec{F}(t_n, [X_n])$

  $K_2 = h\vec{F}\left(t_n + \frac{h}{4}, [X_n] + \frac{1}{4}K_1\right)$

  $K_3 = h\vec{F}\left(t_n + \frac{3h}{8}, [X_n] + \frac{3}{32}K_1 + \frac{9}{32}K_2\right)$

  $K_4 = h\vec{F}\left(t_n + \frac{12h}{13}, [X_n] + \frac{1932}{2197}K_1 - \frac{7200}{2197}K_2 + \frac{7296}{2197}K_3\right)$

  $K_5 = h\vec{F}\left(t_n + h, [X_n] + \frac{439}{216}K_1 - 8K_2 + \frac{3680}{513}K_3 - \frac{845}{4104}K_4\right)$

  $K_6 = h\vec{F}\left(t_n + \frac{h}{2}, [X_n] - \frac{8}{27}K_1 + 2K_2 - \frac{3544}{2565}K_3 + \frac{1859}{4104}K_4 - \frac{11}{40}K_5\right)$

  $[X_{new}] = [X] + \frac{25}{216}K_1 + \frac{1408}{2565}K_3 + \frac{2187}{4104}K_4 - \frac{1}{6}K_5$

  $[\tilde{X}_{new}] = [X] + \frac{16}{135}K_1 + \frac{6656}{12825}K_3 + \frac{28561}{56430}K_4 = \frac{9}{50}K_5 + \frac{2}{55}K_6$

  $\Delta = \frac{\left|[\tilde{X}_{new}] - [X_{new}]\right|}{h}$

  if $\Delta \leq \epsilon_t$ **then**

    $[X] = [X_{new}]$

    $t = t + h$

  $\sigma = 0.84\left(\frac{\epsilon_t}{\Delta}\right)^{\frac{1}{4}}$

  if $\sigma < 0.1$ **then**

    $\sigma = 0.1$

  if $\sigma > 4$ **then**

    $\sigma = 4$

  $h = h\sigma$

5.6.2.2 Discussion

Even though the computation of the simulation step is more demanding than the RK4 method the possibility of changing h often decreases the simulation runtime.

5.6.3 Stiffness

An expected result of the adaptive method is that h is small in region where the solution curves display large variation and large in region when the solution have a near zero slope. This doesn't always happen: the stepsize h is forced to be small to an unacceptable level in region where solution curves are very small. This phenomenon is known as stiffness. A system of ODE is said to be stiff when a numerical solution of its initial value problem forces the numerical method to employ a discretization stepsize excessively small with respect to the smoothness of the exact solution. This

is a property of the couple of ODE system and numerical scheme used to solve. It could appear or disappear changing the numerical scheme. It doesn't arise only in adaptive methods, but it is important because the algorithm may lose control on the update of h with an impact on the runtime. In this cases the problem is solved by changing the numerical algorithm. Usually BDF methods are used to simulate a system of ODEs that exhibits stiffness with adaptive numerical simulation.

5.7 Issues of deterministic simulation

Regardless of the chosen numerical method the deterministic simulation provides only an approximation of the biochemical dynamics, this is often not a problem: the gain in simulation runtime compensates the loss in accuracy, especially for higher-order numerical methods. However there are some specific conditions in which deterministic simulation fails to compute the real behaviour of the system.

5.7.1 Spatial homogeneity and continuum hypothesis

In particular spatial homogeneity and the continuum hypothesis are not satisfied for biological processes involving low numbered chemical species because the dynamics could be partially or totally driven by few stochastic events impossible to observe by considering averaged dynamics. Even when the two hypothesis are satisfied some properties are impossible to observed.

5.7.2 Multistability

One of them is multistability: this is never observed by an averaged dynamics of the system because the deterministic simulation computes the same averaged dynamics when starting from the same initial state.

5.7.3 Steady state

Another condition that lead to important differences arises when the biochemical system is simulated from a steady state, a solution of the set of ODEs when all derivative are set to 0. This makes the averaged dynamics stationary. The asynchronous application of reaction firing in stochastic simulation could cause an exit from the equilibrium leading to non-stationary dynamics.

Chapter 6

Hybrid simulation approaches

6.1 Introduction

Hybrid simulation approaches refer to a class of methods that combines the advantages of complementary simulation approaches: the system is partitioned into subsystems that are simulated with different methods. When exact stochastic simulation is not a realistic option approximate strategies have to be considered together with their accuracy, so they often require the fulfilment of specific conditions. The state space associated with biochemical reaction networks can be partitioned into regions depending on the nature of the system, according to the number of molecules and the frequency of reaction events. This helps evaluate which approximations are reasonable. The essential elements are the abundance of the species and the frequency of reaction. Threshold variables demarcate the different partitions and they are model dependents as seen in figure 6.1:

- t_1 indicates whether a reaction is considered fast.
- t_2 indicates whether a population is abundant or not.
- t_3 and t_4 the border between stochastic variation and deterministic behaviour.

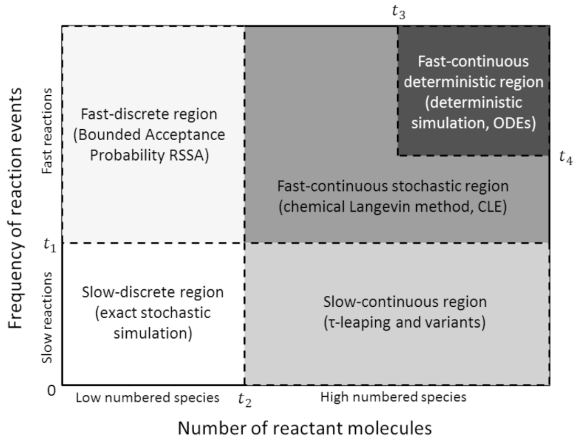

Figure 6.1: Partitions of the state space

6.1.1 Slow-discrete region

The system is in a slow-discrete region when the reaction are slow and low numbered species are present. The species must be represented with an integer and discrete changes are allowed. Moreover the reaction events are rare and correspond to significant changes in the system. This region should be treated with the highest accuracy method available.

6.1.2 Fast-discrete region

The system is in the fast-discrete region whenever molecular populations are small and the reactions happen frequently enough that exact simulation could be intractable. This region can be treated with stochastic approximate algorithms developed to work with large reaction propensities and small populations like BA-RSSA.

6.1.3 Slow-continuous region

The system is in the slow-continuous region whenever molecular population are large and reaction are slow. The molecular populations can be assumed continuous and each reaction occurrence does not change significantly its species concentration. The simulation of many reaction occurrences can be skipped without affecting the accuracy. This region can be treated by the τ-leaping algorithm.

6.1.4 Fast continuous stochastic and deterministic region

The system starts in the fast continuous stochastic and goes into the deterministic whenever it has fast reactions and high number species. In the first case the CLE can be considered for the simulation, while in the second deterministic simulations can be used.

6.1.5 Dimensionality explosion

The system is two dimensional only when it has one species and one reaction. When these two sets increase the criteria above can be applied only when all of them belong to the same region. This is a very strict condition that must be preserved along all of the simulation process. Hybrid simulation approaches try to solve this problem by dividing the system into parts that fit with the regions described which are then simulated by ad hoc simulation methods to provide the best compromise between accuracy and runtime.

6.2 Reaction-based system partitioning

The reaction based system partitioning partitions the system into subnetwork that require to be simulated by the same strategy. The reaction are divided into $\mathcal{R}^s$ and $\mathcal{R}^f$ of slow and fast reactions. Fast reaction have high propensities and can be simulated by CLE or deterministic simulation while slow one have low propensities. When deterministic simulation is employed a preliminary step is required to transform the set of reactions in ODEs. Slow reactions require a more accurate stochastic simulation approach.

6.2.1 Classifying fast reactions

Reactions can be fast when it occurs many times in a small time interval and the effect of each reaction on the number of reactants and products is small, so quantitatively:

$$a_j \tau^f \geq \theta \ll 1$$

And:

$$\vec{x}_i > \lambda |\vec{v}_{ij}|, \forall S_i \in Reactants(R_j) \cup Products(R_j)$$

θ and λ define the minimum number of time a fast reaction can be applied within τ^f and how fine grained the species must be in order for them to appear as continuous. Usually for practical models $\theta = 10$ and $\lambda = 100$.

6.2.2 Partitioning algorithm

An implementation of this reaction-based partitioning can be found in algorithm 42.

Algorithm 42: `Two class reaction-based partitioning()`

Input: a biochemical reaction system with stoichiometrix matrix $\vec{v}$ and current state $\vec{x}$, the time increment τ^f for the approximate simulation of fast reaction, θ defining the minimum number of times that a fast reaction has to be applied on average within τ^f and λ indicating how fine grained the reactant and product species must be in order to be approximated by continuous numbers
Output: the sets $\mathcal{R}^s$ and $\mathcal{R}^f$
$\mathcal{R}^s = \emptyset$
$\mathcal{R}^f = \emptyset$
foreach $R_j \in \mathcal{R}$ **do**
    if $a_j \tau^f < \tau$ **then**
        $R_j \in \mathcal{R}^s$
    else if $\exists S_i \in Reactants(R_j) \cup Products(R_j) : \vec{x}_i \leq \lambda |\vec{v}_{ij}|$ **then**
        $R_j \in \mathcal{R}^s$
    else
        $R_j \in \mathcal{R}^f$

6.2.3 Dynamic partitioning

Because reaction propensities and state vector change in time the partitioning has to be evaluated during the simulation. This is dynamic partitioning. It is important when the system state varies considerably over time. A fixed partitioning can be used when the subsystems are different and these differences remain constant during the simulation.

6.2.4 Different constraints

Some partitioning strategies depends only on the second condition and relax it to include only the reactants. This is to speed up the simulation by taking out the stochastically simulated fast reaction subsystem, while particle-number based partitioning is concerned with correctly treating reactions with small concentrations. Moreover τ^f can change at each simulation step and can be not considered: the reaction are divided into a tentative partitioning and then a fixed point iterative scheme happens, tuning the partitioning so that the propensity of each fast reaction will be at least

λ times larger than the fastest slow reaction. This procedure, outlined in algorithm 43 is more computationally demanding but has less parameters.

Algorithm 43: `Two class reaction-based partitioning with fixed point iterations()`

Input: a biochemical reaction system with stoichiometrix matrix $\vec{v}$ and current state $\vec{x}$, the time increment τ^f for the approximate simulation of fast reaction, γ indicating how fine grained the reactant and product species must be in order to be approximated by continuous numbers abd λ indicating how many times the propensity of the slowest fast reaction has to be larger than the propensity of the fastest slow reaction
Output: the sets $\mathcal{R}^s$ and $\mathcal{R}^f$
$a^s_{\max} = 0$
$\mathcal{R}^s = \emptyset$
$\mathcal{R}^f = \emptyset$
foreach $R_j \in \mathcal{R}$ **do**
    if $\exists S_i \in Reactants(R_j) \cup Products(R_j) : \vec{x}_i \leq \gamma |\vec{v}_{ij}|$ **then**
        $R_j \in \mathcal{R}^s$
        $a^s_{\max} = \max\{a^s_{\max}, a_j\}$
    else
        $R_j \in \mathcal{R}^f$
if $\mathcal{R}^s \neq \emptyset \wedge \mathcal{R}^f \neq \emptyset$ **then**
    repeat
        fxdpt = **true**
        foreach $R_j \in \mathcal{R}^f$ **do**
            if $a_j < \lambda a^s_{\max}$ **then**
                move R_j from $\mathcal{R}^f$ to $\mathcal{R}^s$
                $a^s_{\max} = \max\{a^s_{\max}, a_j\}$
                fxdpt = **false**
    until $fxdpt$ =**true**

6.2.5 Four class reaction partitioning

The reaction partitioning in two classes reduces at minimum the simulation strategies implemented in the hybrid algorithm, simplifying its implementation, but reducing the accuracy of the results. A higher number of reaction classes are introduced to be more consistent with the regions. A way of doing so is to divide it into fours set:

- Very slow reactions $\mathcal{R}^{vs}$ requiring exact simulation.
- Slow reactions $\mathcal{R}^s$ requiring a τ-leaping method.
- Medium reactions $\mathcal{R}^m$, requiring CLE.
- Fast reactions $\mathcal{R}^f$, requiring deterministic simulation.

At the basis of this strategy is the computation of τ, providing the next firing time of a model reaction and imposing the following restraints:

- $a_j \tau \leq 1$ R_j very slow.

- $a_j\tau > 1 \wedge a_j\tau \not\gg 1$ R_j is slow.
- $a_j\tau \gg 1 \wedge \sqrt{a_j\tau} \not\gg 1$ R_j is medium.
- $\sqrt{a_j\tau} \gg 1$ R_j is fast.

6.2.5.1 Algorithm

An implementation of the four class reaction based partitioning is outlined in algorithm 44, introducing θ to implement the $\gg$ constraint.

Algorithm 44: `Four class reaction-based partitioning()`

Input: a biochemical reaction system with current staet $\vec{x}$, the firing time τ of the next model reaction and the threshold parameter θ
Output: the sets $\mathcal{R}^{vs}$, $\mathcal{R}^{s}$, $\mathcal{R}^{m}$ and $\mathcal{R}^{f}$
$\mathcal{R}^{vs} = \emptyset$
$\mathcal{R}^{f} = \emptyset$
$\mathcal{R}^{m} = \emptyset$
$\mathcal{R}^{f} = \emptyset$
foreach $R_j \in \mathcal{R}$ **do**
    if $a_j\tau \leq 1$ **then**
        $R_j \in \mathcal{R}^{vs}$
    else if $a_j\tau < \theta$ **then**
        $R_j \in \mathcal{R}^{s}$
    else if $\sqrt{a_j\tau} < \theta$ **then**
        $R_j \in \mathcal{R}^{m}$
    else
        $R_j \in \mathcal{R}^{f}$

6.3 Synchronization of exact and approximate simulations

When the hybrid strategy moves forward one simulation step, slow reaction are simulated by exact stochastic simulation while less accurate strategies are used for faster reactions. The approximate simulation of faster reactions is constrained to not exceed the firing time of the first slow reaction. This is a synchronization to preserve the accuracy of the simulation. Even if the simulation of slow reactions is exact, their firing is not guaranteed to be exact when faster reactions are simulated in an approximate way. Since the approximation is difficult to obtain many hybrid algorithm implement a simulation for slow reactions that is not exact, but less approximate than the one of the fast reactions.

6.3.1 Algorithm

An implementation of a four class hybrid simulation strategy is implemented in algorithm 45.

Algorithm 45: `Four class hybrid simulation strategy()`

Input: a biochemical reaction system with initial state X_0, the simulation ending time $T_{\max}$ and the threshold parameter $\theta > 1$
Output: a trajectory of the biochemical system
$t = 0$
$X = X_0$
while $t < T_{\max}$ **do**
    compute τ according to the τ-leaping method
    repeat
        $updateNeeded =$ **false**
        foreach $R_j \in \mathcal{R}$ **do**
            $\mathcal{R}^{vs}, \mathcal{R}^s, \mathcal{R}^m, \mathcal{R}^f =$ `partitioning(`$\mathcal{R}$`, `θ`, `τ`)`
        foreach $R_j \in \mathcal{R}^{vs}$ **do**
            compute τ_f using exact simulation algorithm
        if $\mathcal{R}^{vs} = \mathcal{R} \wedge \tau \neq \min\{\tau_j\}, R_j \in \mathcal{R}^{vs}$ **then**
            $\tau = \min\{\tau_j\}, R_j \in \mathcal{R}^{vs}$
        if $\tau > \min\{\tau_j\}, R_j \in \mathcal{R}^{vs}$ **then**
            $\tau = \min\{\tau_j\}, R_j \in \mathcal{R}^{vs}$
            $updateNeeded =$ **true**
        if $\neg updateNeeded$ **then**
            compute X^{new} at $t + \tau$ approximating the firing of reactions in $\mathcal{R}^s.\mathcal{R}^m, \mathcal{R}^f$ accorrding to:
            τ-leaping method for slow reactions
            CLE for medium reactions
            deterministic simulation for fast reactions
            update X^{new} by applying $R_j \in \mathcal{R}^{vs}$, such that $\tau_j = \tau$
            if *any* $X_i^{new} < 0$ **then**
                $\tau = \frac{\tau}{2}$
                $updateNeeded =$ **true**
            else
                $X = X^{new}$
                $t = t + \tau$
    until $\neg updateNeeded$

6.3.2 Preserving the exactness

In order to preserve the exactness of the simulation of slow reactions, the reaction probability density has to be extended to consider time-varying transition propensities account for the propensities of slow reactions that change due to the simulation of fast reactions. The pdf of the next firing of a slow reaction $R_\mu \in \mathcal{R}^s$ becomes:

$$p^s(\tau, \mu|\vec{x}, t) = a_\mu(X(t+\tau))e^{-\int_t^{t+\tau} a_0^s(X(t'))dt'}$$

Where $X(t+\tau)$ is the system at time $t+\tau$, $\vec{x}$ is the current system state and:

$$a_0^s = \sum_{R_j \in \mathcal{R}^s} a_j$$

The firing time of the next slow reaction is obtained by solving:

$$\int_t^{t+\tau} a_0^s(X(t'))dt' = -\ln(r)$$

This solution is difficult to compute: the system state is changed by fast reactions during the time interval. The hybrid simulation has to evaluate it simultaneously with the simulation of the fast reaction to generate the next slow reaction event.

6.3.2.1 Event detection

In order to satisfy the time integral the simulation of fast reaction has to include an event detection part. The easiest way to do so is to relax the constraint of the equation to find a time instant along the simulation such that:

$$\left| \int_t^{t+\tau} a_0^x(X(t'))dt' + \ln(r) \right| \leq \epsilon$$

Where $\epsilon \approx 0$ is a positive error threshold. This makes the simulation of slow reaction depend on ϵ. This correspond to adding to the set of deterministic ODEs an additional equation:

$$\frac{dRES}{dt} = a_0^x RES(0) = \ln(r)$$

This is numerically integrated together with the set of ODE until $|RES(t+\tau)| \leq \epsilon$. The computation of the firing time of the next slow reaction will be not exact because it depends on the order of the numerical method used to solve the initial value problem.

6.3.2.2 Algorithm

A two class hybrid strategy with exact simulation of slow reactions is outlined in algorithm 46.

Algorithm 46: `Two class hybrid strategy with exact simulation of slow reactions()`

Input: a biochemical reaction system with initial state X_0, the simulation ending time $T_{\max}$
Output: a trajectory of the biochemical system
$t = 0$
$X = X_0$
while $t < T_{\max}$ **do**
  foreach $R_j \in \mathcal{R}$ **do**
    $\mathcal{R}^s, \mathcal{R}^f =$ `partitioning(`$\mathcal{R}$`)`
  generate two uniform random numbers $r_1, r_2 \sim norm(0,1)$
  compute X_{new} from X by approximating the firing of fast reactions until $t+\tau$ is reached that satisfied the integral $\int_t^{t+\tau} a_0^s(X(t'))dt' + \ln(r) = 0$
  select the slow reaction $R_\mu \in \mathcal{R}^s$ with the smallest μ such that $\sum_{j=1}^{\mu} a_j > r_2 a_0^s$
  update X_{new} by applyinh R_μ
  $t = t + \tau$

6.4 Hybrid Rejection-Based SSA

The hybrid rejection based stochastic simulation algorithm HRSSA is built on top of RSSA. The simulation strategy relies on the concept of fluctuation interval:

$$[\underline{\vec{x}}, \overline{\vec{x}}] = [(1-\delta)\vec{x}, (1+\delta)\vec{x}], 0 < \delta < 1$$

Used to obtain exact stochastic simulation of slow reactions without computing the integral, improving simulation accuracy and runtime.

6.4.1 Reaction partitioning

Reactions are partitioned in two classes and the partition is updated only when the current system does not fit any more in its fluctuation interval: the partitioning is done considering the lower bond $\underline{\vec{x}}$. This does not affect the accuracy of the classification, but it imposes more stringent constraints that increase the number of reactions classified as slow.

6.4.2 Firing time

The sum of the upper propensity bounds of slow reactions is computed:

$$\overline{a_0}^s = \sum_{R_j \in \mathcal{R}^s} \overline{a}_j = \sum_{R_j \in \mathcal{R}^s} a_j(\overline{\vec{x}})$$

The firing time of a candidate slow reaction is:

$$\tau = -\frac{\ln r}{\overline{a}_0^s}$$

Where $r \sim norm(0,1)$. If the system state remains in the fluctuation interval $\overline{a}_0^s$ is not dependent on time in $[t, t+\tau]$, allowing to simulate fast reactions over this interval without an side effect on slow reactions.

6.4.3 Updating the system

After the simulation of fast reaction in $[t, t+\tau]$, a slow reaction is chosen and validated to fire. To preserve the exactness, the feasibility of the current system must be preserved during the simulation of the fast reactions. Every time the system state exists from its bounds the simulation has to be stopped and the fluctuation interval is updated. Fast reactions are simulated in $[r, t+\tau']$, where $t+\tau'$ is the last computed time step $\tau' \leq \tau$ that does not violate the feasibility of the current system state. When $\tau' < \tau$, the simulation process is stopped and the fluctuation interval of the system state is updated, ensuring correct synchronization. When $\tau' = \tau$, the state can be updated by applying one slow reaction R_j, selected according to the RSSA simulation strategy.

6.4.4 Algorithm

An implementation of HRSSA is outlined in algorithm 47.

Algorithm 47: `Hybrid Rejection-Based SSA (HRSSA)()`

Input: a biochemical reaction system with initial state X_0, δ to compute the fluctuation interval, τ^f, θ, γ to partition the system and simulation ending time $T_{\max}$.
Output: a trajectory of the biochemical reaction system
$t = 0$
$\vec{X} = \vec{x}_0$
while $t < T_{\max}$ **do**
  $[\underline{X}, \overline{X}] = [(1-\delta)X, (1+\delta)X]$
  foreach $R_j \in \mathcal{R}$ **do**
    compute $\overline{a_j}$ and $\underline{a_j}$
    $\mathcal{R}^s, \mathcal{R}^f$ = `partition(`$\underline{X}$, γ, θ, τ^f`)`
    bound the system state $\underline{X}$ according to γ, θ, τ^f
  $\overline{a}_0^s = \sum_{R_j \in \mathcal{R}^s} \overline{a}_j$
  updateNeeded = **false**
  while $t < T_{\max} \wedge \neg updateNeeded$ **do**
    $r \sim norm(0,1)$
    $\tau = -\frac{\ln r}{\overline{a}_0^s}$
    compute $X(t+\tau')$ by simulating $\mathcal{R}^f$ at time steps of maximum length τ^f, where $t+\tau'$ is the last computed step such that $\tau' \leq \tau \wedge X(t+\tau') \in [\underline{X}, \overline{X}]$
    if $\tau' = \tau$ **then**
      generate $r_1, r_2 \sim norm(0,1)$
      accepted = **false**
      select $R_\mu \in \mathcal{R}^s$ with smallest μ such that $\sum_{j=1}^{\mu} \overline{a}_j > r_i \overline{a}_0^s$
      if $r_2 \leq \frac{\underline{a}_\mu}{\overline{a}_\mu}$ **then**
        accepted = **true**
      else
        compute $a_\mu(X(t+\tau'))$
        if $r_2 \leq \frac{a_\mu(X(t+\tau'))}{\overline{a}_\mu}$ **then**
          accepted = **true**
      if *accepted* = **true then**
        update $X(t+\tau')$ by applying R_μ
      if $X(t+\tau') \notin [\underline{X}, \overline{X}]$ **then**
        updateNeeded = **true**
    else
      updateNeeded = **true**
    $X = X(t+\tau')$
    $t = t+\tau'$

6.4.5 Correctness of the simulation of slow reactions

HRSSA simulates slow reactions exactly under the hypothesis that the system state remains in its fluctuation interval during the simulation. Let $p_1^s(\tau|\vec{x},t)$ be the pdf of the firing time τ such that $p_1^s(\tau|\vec{x},t)d\tau$ gives the probability that the slow reaction is selected to fire at time $t+\tau$. Let $p_2^s(\mu|\tau,\vec{x},t)$ be the probability that the slow reaction is R_μ assuming that $\vec{x} \in [\underline{X},\overline{X}]$. The multiplication of p_1^s and p_2^s gives the joint probability density function of the next slow reaction R_μ firing at $t+\tau$, sampled by HRSSA. To compute $p_1^s(\tau|\vec{x})$ let $\mathbb{P}\{R^s\}$ be the probability that some slow reaction is accepted by HRSSA at $t+\tau$:

$$\mathbb{P}\{R^s\} = \frac{\int_t^{t+\tau} \sum\limits_{R_j \in \mathcal{R}^s} a_j(X(t'))dt'}{\overline{a}_0^s} = \frac{\int_t^{t+\tau} a_0^s(X(t'))dt'}{\overline{a}_0^s \tau}$$

If k is the number of attempts before a reaction is accepted, k is geometrically distributed with success probability $\mathbb{R}\{R^s\}$. For a fixed k, τ is an Erlang distribution of parameters k and $a_0^s(\vec{\overline{x}})$. Summing over all possible k:

$$p_1^s(\tau|\vec{x},t) = \sum_{k=1}^{\infty} f_{erlang(k,\overline{a}_0^s)}(\tau)\mathbb{P}\{R^s\}(1-\mathbb{P}\{R^s\})^{k-1}$$

Where $f_{erlang(k,\overline{a}_0^s)}$ is the pdf of the Erlang distribution. Substituting the formula of $\mathbb{P}\{R^s\}$ and recalling:

$$f_{erlang(k,\overline{a}_0^s)} = \frac{(\overline{a}_0^s)^k \tau^{k-1} e^{-\overline{a}_0^s \tau}}{(k-1)!}$$

Then:

$$\begin{aligned}
p_1^s(\tau|\vec{x},t) &= \sum_{k=1}^{\infty} \frac{(\overline{a}_0^s)^k \tau^{k-1} e^{-\overline{a}_0^s \tau}}{(k-1)!} \frac{\int_t^{t+\tau} a_0^s(X(t'))dt'}{\overline{a}_0^s \tau} \left(1 - \frac{\int_t^{t+\tau} a_0^s(X(t'))dt'}{\overline{a}_0^s \tau}\right)^{k-1} = \\
&= \frac{\int_t^{t+\tau} a_0^s(X(t'))dt'}{\tau} e^{-\overline{a}_0^s \tau} \sum_{k=1}^{\infty} \frac{(\overline{a}_0^s \tau - \int_t^{t+\tau} a_0^s(X(t'))dt')^{k-1}}{(k-1)!} = \\
&= \frac{\int_t^{t+\tau} a_0^s(X(t'))dt'}{\tau} e^{-\overline{a}_0^s \tau} e^{\overline{a}_0^s \tau - \int_t^{t+\tau} a_0^s(X(t'))dt'} = \\
&= \frac{\int_t^{t+\tau} a_0^s(X(t'))dt'}{\tau} e^{-\int_t^{t+\tau} a_0^s(X(t'))dt'}
\end{aligned}$$

Now let $\mathbb{R}\{R_\mu\}$ be the probability that R_μ is selected and accepted at $t+\tau$ given that some reaction is accepted at that time $\mathbb{P}\{R_\mu\}$ is the multiplication of the probability that R_μ is selected and is accepted:

$$\mathbb{P}\{R_\mu\} = \frac{\overline{a}_\mu}{\overline{a}_0^s} \frac{a_\mu(X(t+\tau))}{\overline{a}_\mu} = \frac{a_\mu(X(t+\tau))}{\overline{a}_0^s}$$

Which combining with $\mathbb{P}\{R^s\}$:

$$p_2^s(j|\tau,\vec{x},t) = \frac{\mathbb{P}\{R_\mu\}}{\mathbb{P}\{R\}} = \frac{\frac{a_\mu(X(t+\tau))}{\overline{a}_0^s}}{\frac{\int_t^{t+\tau} a_0^s(X(t'))dt'}{\overline{a}_0^s \tau}} = \frac{a_\mu(X(t+\tau))\tau}{\int_t^{t+\tau} a_0^s(X(t'))dt'}$$

Now:

$$p_1^s(\tau|\vec{x},t)p_2^s(j|\tau,\vec{x},t) = a_\mu(X(t+\tau))e^{-\int_t^{t+\tau} a_0^s(X(t'))dt'}$$

Which is equal to $p^s(\tau,\mu|\vec{x},t)$ proving the correctness of HRSSA.

6.5 Hybrid simulation with stiffness

Fast reactions have highly reactive species which may have a low population which quickly equilibrates to a stochastic equilibrium distribution on the time scales of slow reactions. This is kept changing over time due to the firings of reactions, but its distribution remains. This equilibrium could be altered by slow reactions. This behaviour is called stiffness and it often arises from the quasi-steady state (QSS) or the reaction partial equilibrium (PE). The former concentrates on the state, while the latter on the reactions. QSS assumes that the changes in populations of fast species is fast with respect to slow species, while PE assumes that fast reactions involving only fast species quickly equilibrate to a stationary distribution in the time scale of slow reactions. Under stiffness the behaviour of species at equilibrium is averaging and the dynamics are formulated conditionally on the limiting of these species, skipping the detailed simulation of the fast reactions.

6.5.1 Formulation of reactions with stiffness

Consider a biochemical network where species are partitioned into a fast $\mathcal{S}^f$ and slow $\mathcal{S}^s$. If a species is highly active is a fast or intermedie species in $\mathcal{S}^f$, otherwise it is a slow or primary species. The state vector is $X = (X^f, X^s)$ and the state change vector of R_j is $\vec{v}_j = (\vec{v}_j^f, \vec{v}_j^s)$. Let now $X(t) = (X^f(t), X^s(t)) = (\vec{x}^f, \vec{x}^s)$ the state at t. The CME equation can be written as:

$$\frac{d\mathbb{P}\{\vec{x}^f,\vec{x}^s,t\}}{dt} = \sum_{j=1}^{M}(a_j(\vec{x}^f-\vec{v}_j^f,\vec{x}^s-\vec{v}_j^s)\mathbb{P}\{\vec{x}^f-\vec{v}_j^f,\vec{x}^s-\vec{v}_j^s,t\}) - \mathbb{P}\{\vec{x}^f,\vec{x}^s,t\}\sum_{j=1}^{M}a_j(\vec{x}^f,\vec{x}^s)$$

The initial state in the condition has been omitted to simplify the notation. The probability can be factorized as:

$$\mathbb{P}\{\vec{x}^f,\vec{x}^s,t\} = \mathbb{P}\{\vec{x}^s,t\}\mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}$$

Differentiating both sides with respect to time:

$$\frac{d\mathbb{P}\{\vec{x}^f,\vec{x}^s,t\}}{dt} = \mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}\frac{d\mathbb{P}\{\vec{x}^s,t\}}{dt} + \mathbb{P}\{\vec{x}^s,t\}\frac{d\mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}}{dt}$$

Resulting in:

$$\begin{aligned}\mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}\frac{d\mathbb{P}\{\vec{x}^s,t\}}{dt} + \mathbb{P}\{\vec{x}^s,t\}\frac{d\mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}}{dt} = \\ = \sum_{j=1}^{M}(a_j(\vec{x}^f-\vec{v}_j^f,\vec{x}^s-\vec{v}_j^s)\mathbb{P}\{\vec{x}^f-\vec{v}_j^s,t\}\mathbb{P}\{\vec{x}^s-\vec{v}_j^s,t\})+ \\ \mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}\mathbb{P}\{\vec{x}^s,t\}\sum_{j=1}^{M}a_j(\vec{x}^f,\vec{x}^s)\end{aligned}$$

Now summing both sides over all possible value of $\vec{x}^f$ and noting that:

- $\sum_{x^f} \mathbb{P}\{\vec{x}^f|\vec{x}^s\} = 1$
- $\sum_{x^f} \frac{d\mathbb{P}\{\vec{x}^f|\vec{x}^s\}}{dt} = \frac{d\sum_{x^f}\mathbb{P}\{\vec{x}^f|\vec{x}^s\}}{dt} = 0$

$$\frac{d\mathbb{P}\{\vec{x}^s,t\}}{dt} = \sum_{j=1}^{(} \hat{a}_j(\vec{x}^s - \vec{v}_j^s)\mathbb{P}\{\vec{x}^s - \vec{v}_j^s, t\}) - \mathbb{P}\{\vec{x}^s,t\}\sum_{j=1}^{M}\hat{a}_j(\vec{x}^s)$$

Where:

$$\hat{a}_j(\vec{x}_s) = \sum_{x^f} a_j(\vec{x}^f,\vec{x}^s)\mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}$$

Is the slow scale propensity: it is the average propensity of a reaction over a fast species. This is the conditional mean of propensity $a_j(\vec{x}^f,\vec{x}^s)$ with respect to $\mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}$. This equation defines a CME for a slow species, To simulate it the QSS assumption or the PE assumption have to be considered: the slow scale propensity is affected by the time-dependent probability $\mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}$. In QSSA the net change in the population of fast species is assumed to be approximately zero. PEA assumes that the fast reactions that reach equilibrium remain in it. These allow to approximate the slow-scale propensity and a stochastic simulation algorithm can then be adapted to simulate the dynamics of slow species.

6.5.1.1 Quasi-steady state assumption

Under QSSA the slow scale propensity is approximated by replacing the probability by a time-invariant one:

$$\hat{a}_j(\vec{x}^s) \approx \sum_{\vec{x}^f} a_j(\vec{x}^f,\vec{x}^s)\mathbb{P}\{\vec{x}^f|\vec{x}^s\}$$

This is obtained by imposing:

- The dynamics of the state X^f conditional on X^s is Markovian: $\mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}$ satisfies the CME equation.
- The net change $\mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}$ is approximately zero: $\frac{d\mathbb{P}\{\vec{x}^f|\vec{x}^s,t\}}{dt} = 0$.

Let $\mathcal{R}^f$ the set of fast reactions that drive $\mathcal{S}^f$ into an equilibrium. Note that the fast reactions can involve a slow species state. The time-invariant probability distribution is the solution of:

$$\sum_{R_j\in\mathcal{R}^f}(a_j(\vec{x}^f - \vec{v}_j^f,\vec{x}^s)\mathbb{P}\{\vec{x}^f - \vec{v}_j^f|\vec{x}^s\}) - \mathbb{P}\{\vec{x}^f,\vec{x}^s\}\sum_{j=1}^{M}a_j(\vec{x}^f,\vec{x}^s) = 0$$

6.5.1.2 Partial equilibrium assumption

The partial equilibrium assumption PEA is an alternative to approximate slow-scale propensity. It is done by uncoupling the dynamics of X^f from X^s in the computation of its stationary distribution. The equilibrium distribution $\mathbb{P}\{\vec{x}^f|\vec{x}^s,t\} \approx \mathbb{P}\{\vec{x}^f(\infty)\}$. The slow scale propensity then is computed as:

$$\hat{a}_j(\vec{x}^s) \approx \sum_{\vec{x}^f} a_j(\vec{x}^f, \vec{x}^s)\mathbb{P}\{\vec{x}^f(\infty)\}$$

Reactions are first provisionally partitioned into $\mathcal{R}^f$ and $\mathcal{R}^s$. A reaction is put into $\mathcal{R}^f$ if its propensity is much larger than a reaction in $\mathcal{R}^s$. Species of reactions in $\mathcal{R}^f$ are put into $\mathcal{S}^f$ with state X^f, while for slow one they are put into $\mathcal{S}^s$ with state X^s. Fast species can participate as a reactant in a slow reaction while slow species cannot participate in a reactant in a fast one. The behaviour of fast species can be affected by slow ones. To uncouple slow species from the fast one, fast reactions are assumed to quickly reach an equilibrium distribution. The dynamics of X^f can uncouple from X^s by defining a virtual fast process $\hat{X}^f$ composed of X^f evolved through fast reactions only. This virtual fast process is only dependent on fast reactions and its probability distribution satisfies a CME, and its probability reaches a stationary distribution quickly:

$$\lim_{t\to\infty} \mathbb{P}\{\vec{x}^f, t\} = \mathbb{P}\{\vec{x}^f(\infty)\}$$

The stationary distribution of fast species is the solution of:

$$\sum_{R_j \in \mathcal{R}^f} (a_j(\vec{x}^f - \vec{v}_j^f, \vec{x}^s)\mathbb{P}\{\vec{x}^f - \vec{v}_j^f\}) - \mathbb{P}\{\vec{x}^f\}\sum_{j=1}^{M} a_j(\vec{x}^f, \vec{x}^s) = 0$$

6.5.2 Slow-scale stochastic simulation algorithm

The slow scale stochastic simulation algorithm ssSSA simulates the dynamical behaviour of slow species. The species have to be partitioned into slow and fast and the fast equilibrate to a stationary distribution under QSSA or PE. It supposes to be able to generate value for fast species state X^f and to compute $\hat{a}_j$ assuming that there exists an explicit formula for $\mathbb{P}\{\vec{x}^f|\vec{x}^s\}$ or $\mathbb{R}\{\vec{x}^f(\infty)\}$. For each slow reaction where the sum over all possible values of X^f given its stationary distribution is:

$$\hat{a}_j(\vec{x}^s) \approx a_j(\langle X^f \rangle, \vec{x}^s)$$

This approximation is exact it the propensity is linear with respect to fast species.

6.5.2.1 Algorithm

An implementation of ssSSA is outlined in algorithm 48.

Algorithm 48: `Slow-scale SSA (ssSSA)()`

Input: a biochemical reaction network of M reactions and N species. Partition the state into X^f and X^s. Find the stationary distribution for X^f as the mehod for genertaing a random value for the corresponding distribution
Output: a trajectory of the biochemical reaction network
$t = 0$
$\vec{X} = \vec{x}_0$
while $t < T_{\max}$ **do**
  compute $\langle X_f \rangle$ from its stationary probability distribution
  $\hat{a}_j(\vec{x}^s) \approx a_j * \langle X^f \rangle, \vec{x}^s$
  $\hat{a}_i = \sum_{j=0}^{M} \hat{a}_j$
  $r_1, r_2 \sim norm(0, 1)$
  select R_μ with minimum μ such that $\sum_{j=1}^{\mu} \hat{a}_j \geq r_1 \hat{a}_0$
  $\tau = \frac{1}{\hat{a}_0} \ln \frac{1}{r_2}$
  $X = X + \vec{v}_\mu$
  $t = t + \tau$

6.5.3 Nested stochastic simulation algorithm

Finding an explicit analytical expression is often difficult, so is a direct computation of $\hat{a}_j$. The nested stochastic algorithm nSSA is an alternative approach that approximates $\hat{a}_j$ on the fly. Two nested SSA simulations are used in which the inner loop simulates the fast reaction and approximates $\hat{a}_j$, while the other simulates the slow subset. Let $X(t) = (\vec{x}^f, \vec{x}^s)$, be the state at t. The simulation is two consecutive SSAs. The first computes $\hat{a}_j$ for a slow reaction while it simulates the fast reactions as:

$$\hat{a}_j(\vec{x}^s) \approx \frac{1}{K} \sum_{i=1}^{K} \frac{1}{\Delta_f t} \int_t^{t+\Delta_f t} a_j(\vec{x}^f, \vec{x}^s) dt$$

Here K and $\Delta_f t$ are predefined number of simulations and time interval for the fast reactions on which the error depends. When they reach ∞ the computation converges to an exact value. The time interval can be chosen small because the fast reactions under stiffness quickly approach the equilibrium. Then the second step is used to select the next slow reaction in the slow reaction subset.

6.5.3.1 Algorithm

An implementation of nSSA is outlined in algorithm 49.

Algorithm 49: `Nested SSA (nSSA)()`

Input: a biochemical reaction network of M reactions and N species. Partition the state into X^f and X^s, the simulation ending time $T_{\max}$, the time discretization $\Delta_f t$ and the number of simulation runs K for simulating fast reactions.

Output: a trajectory of the biochemical reaction network

$t = 0$

$\vec{X} = \vec{x}_0$

while $t < T_{\max}$ **do**

  simulate only the fast reaction subset by K runs of SSA from t to $t + \Delta_f t$

  $\hat{a}_j(\vec{x}^s) \approx \frac{1}{K} \sum_{i=1}^{K} \frac{1}{\Delta_f t} \int_t^{t+\Delta_f t} a_j(\vec{x}^f, \vec{x}^s) dt$

  $\hat{a}_0 = \sum_{R_j \in \mathcal{R}^f} \hat{a}_j$

  $r_1, r_2 \sim norm(0, 1)$

  select R_μ with minimum μ such that $\sum_{j=1}^{\mu} \hat{a}_j \geq r_1 \hat{a}_0$

  $\tau = \frac{1}{\hat{a}_0} \ln \frac{1}{r_2}$

  $X = X + \vec{v}_\mu$

  $t = t + \tau$

Part II

Parameter optimization

Chapter 7

Introduction

7.1 Systems

A system is a set of integrated and interacting components or entities that form a whole with definite boundaries and surrounding environment. It has a goal to achieve by performing one or more functions or tasks. Systems can be aggregated into a hierarchy. A system at a given level of detail can be a component of another at a higher level of detail.

- A complex (non-linear) system is a system that does not satisfy the principle of superposition: the behaviour of the system cannot be inferred from the behaviour of its components.
- A dynamical system is a system where fixed rules define the time dependencies of the system in a geometrical space. Dynamical systems have a space and time dimension because they change their characteristics over time. Picking snapshots of the system at different time points, different configurations of the system (data) are observed

7.1.1 Configuration

A configuration or state of the system refers to the current condition of the system and stores enough information to predict its next move. A state is characterized by the position of its components in a geometrical space and by the values of the attributes of its components like concentration or number of each elements involved. Systems change their state over time by changing the location of some of their components or changing the attributes of some of their components. Some particular configuration are:

- Steady state: some of the attributes of the system will no longer change in the future.
- Transient state: the system stays in this state for time needed to reach the steady state.

7.1.2 Determinism, nondeterminism and stochasticity

- Deterministic systems always react in the same way to the same set of stimuli. These systems are completely determined by the initial state and the input set. The essence of deterministic systems is that each event is causally related to previous events and choices are always resolved in the same way in the same context.
- Nondeterministic systems are systems in which multiple different outcomes can be generated from the same input in different observation, and so the output cannot be predicted from the input.
- Stochasticity is the quality of lacking any predictable order or plan and stochastic systems possess some inherent randomness. It is possible to transform a nondeterministic system into a stochastic one by attaching probabilities to the selection points so that nondeterministic choices are turned into probabilistic choices.

7.1.3 Computational complexity

Complexity arises when interacting components self-organize to form evolving structures that exhibit a hierarchy of emergent system properties. An emergent behaviour can be originated by a collection of components that interact in the absence of a centralized point of control to produce something that has not been designed or programmed in the system construction or evolution. Computational complexity is the amount of resources, measured as a function of the size of the input, needed to execute an algorithm. It is divided into:

- Computational space complexity: the amount of memory needed during the execution
- Computational time complexity: the number of instructions to be executed.

7.2 Models

A representation is a set of symbols used to convey information and knowledge about a system. It is either physical as a cell or an ecosystem, or artificial as a computer network or an economic market. An abstraction is a representation that ignores some aspects of a system which are not of interest for the current investigation. A *model* is an abstraction of a system. A model has its own interacting components that are characterized by the attributes that are under investigation. The set of all the attributes in a model is the experimental frame.

- A dynamic model aims at predicting the behaviour of the system in time and space through what if analysis. What if analysis investigates how a change in some attributes affects the behaviour of the modelled system.
- A computational model is a model that can be manipulated by a computer to observe properties of the corresponding system.

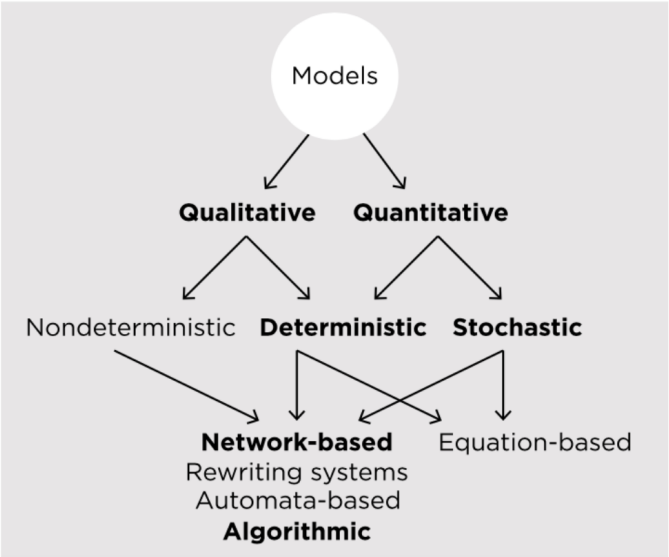

Figure 7.1: From a model to methods

7.2.1 Checking the validity of a model

Validity is a fundamental property of models and witnesses the capacity of a model of making good predictions. Assessing the validity of a model is a fundamental step to predict the behaviour of the system of interest. Assume that M is a model for the system S and $\underline{M}$ is the modelling process. Let $s(t)$ and $m(t)$ be the state of the system and of the model at time t, and f_s and f_m the state transition functions of the system and of the model respectively. Finally, let $I_s(t)$ and $O_s(t)$ be the input and output of the system at time t. Similarly, the input and output are defined also for the model $I_m(t)$ and $O_m(t)$. Going from one state to another a function for the system and one for the model are found. In particular in a mathematical model f_m is integrated to investigate the transition phase, while for the real system f_s is knot known. Because of this the model needs to be input/output validated: known input and output are used to investigate it, where input and output are generalized concepts. For example, an input can be any perturbation of the system, while an output any observable property causally related to the input. Now a model M can be said to be valid for a system S if:

$$f_m(\underline{M}(s(t_0))) = \underline{M}(f_s(s(t_0))) = m(t_1)$$

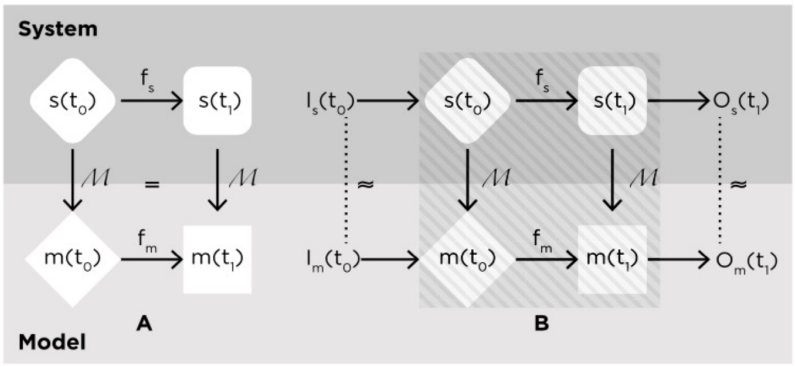

Figure 7.2: Validation

An input/output validity check can be performed by comparing data-set from the system and from the model. This checking process could cause overfitting:

- A model is well tuned to a specific dataset used to build the model.
- It performs poorly on other datasets.

7.2.1.1 Cross validation

Cross validation is a process that checks overfitting by testing the model on data sets different from the ones used to build and calibrate or train the model. These concepts, even if usually referred to computational models, may apply to general models or representations of a system.

7.2.2 Building a model

The first step to build a model is to define:

- The system to be modelled.
- The property of interest of the system.
- The expected output.
- The objective of the model.

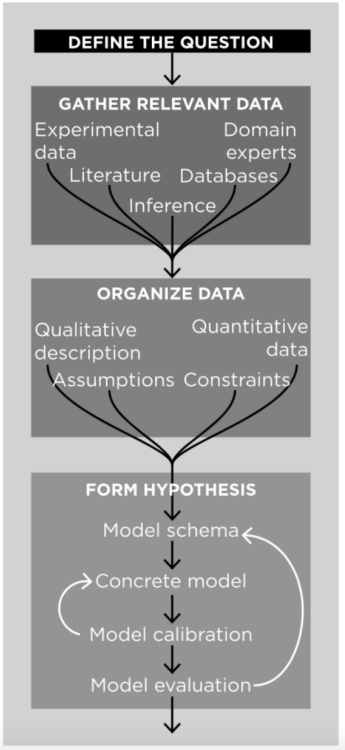

Figure 7.3: Workflow

After defining these things the model needs to be built and calibrated, so to check if it can recapitulate data. If it does not, the model can be wrong or some parameters need to be tuned. It is important to note that different parameters can lead to dramatic changes in dynamics, like for example, in the Lotka-Volterra model in figure 7.4. It can be noted how this example shows periodic oscillation, with the same amount of preys and predators, on the left, while on the right, after parameter change, it shows higher peaks and a lower predator presence.

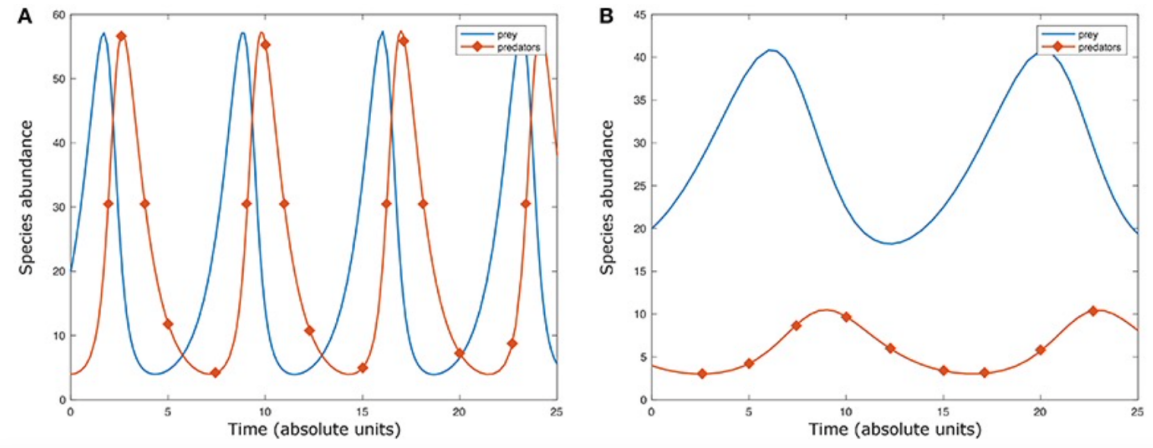

Figure 7.4: Lotka-Volterra model

Chapter 8

Optimization problem

8.1 Introduction

An optimization problem is a problem in which the objective is to maximize or minimize a function. The optimization objective is called an objective or cost function that depends on a variable or a vector of variables called unknowns, parameters or parameter estimates. These variables may be subject to certain constraints.

8.1.1 Definition of a minimum

A point x^* is called a minimum of a function f if:

$$\exists \varepsilon > 0 : \forall x : ||x - x^*|| < \varepsilon \Rightarrow f(x) \geq f(x^*)$$

Respectively, for the maximum:

$$f(x) \leq f(x^*)$$

A minimum is called global if is valid $\forall x \in \mathbb{R}^n$. Or, in this domain:

$$f(x) \geq f(x^*)$$

In general it is not easy to compute the global minimum or maximum, especially if there are a lot of dimensions. To find minima or maxima, $f'(x) = 0$ should be imposed. A stationary point $\bar{x}$ is defined as the point for which:

$$f'(\bar{x}) = 0$$

8.1.2 Finding the minimum

In general in an optimization problem f and f' are not known, so a numerical approximation is used. The idea is to look at the derivative f' and try to move in the point space so to set $f' = 0$. This is at the base of gradient methods.

8.1.3 Constraints

The optimization problem could have some constraints on the values that x can have, so they have to be considered. The problem now becomes:

$$\begin{cases} \max\limits_{x \in \mathbb{R}^n} f(x) \\ g_i(x) = 0 \quad i \in \mathcal{I} \\ x \in \mathbb{R}^n \end{cases}$$

Where:

- $\mathcal{I} = 1, \ldots, m$ iterates over the constraints.
- The constraints g_i could be equality or inequality ones.

8.2 Parameter optimization of a model

A model is a function that given a certain interval or time, initial conditions and parameters, returns the variables at that time the end of the interval. Assuming a deterministic condition, the model m can be defined as:

$$m : \mathbb{R} \times \mathbb{R}^{(n+1)} \times \mathbb{R}^m \to \mathbb{R}^N$$

Where $n + 1$ accounts for the dimension of variables and time:

$$(t_1, (\vec{x}_0, t_0), \theta) \to \vec{x}_1$$

Where θ is the vector of parameters. Assuming that there are k observation for the model:

$$(t_i, \vec{y}_i) \quad i = 1, \ldots, k$$

Where $t_i \in \mathbb{R}^+, \vec{y}_i \in \mathbb{R}^\ell$. It should be noted how in complex systems, where some variables are often not observed, y could be a subset of dimension $\ell < N$.

8.2.1 Distance

The distance is the measure of how the output of the model is far from the observed labels. It will be used to determine how to change the parameters of the model. The objective becomes to change the parameters to minimize the distance between predicted values and observed ones. Assume that $\ell = N$ and that:

$$m\left(t_1, (\vec{x}_0, t_0), \theta\right) \in \mathbb{R}^N = m(t_i, \theta)$$

Can be computed. The distance will be:

$$d_i = \vec{y}_i - m(t_i, \theta)$$

Where d_i is any type of distance. The point-wise distance between the model and the observed labels is computed. A common distance measure is the Euclidean one:

$$d_\varepsilon = \sqrt{\sum_{i=1}^{k} (\vec{y}_i, -m(t_i, \theta))^2} = \\ = \sqrt{\sum_{i=1}^{K} \sum_{j=1}^{N} \left(y_{ij} - m_j\left(t_i, \theta\right)\right)^2}$$

8.2.2 Weights

A weight can be added to scale the components of the output of the model as to make them more comparable and to make the variables unit of measurement agnostic, unbiasing the results. A weight is a scalar that multiplies the component in the measure of distance transforming it as a multiplicative factor. For example in the least square algorithm the distance is computed as:

$$d_\varepsilon = \sqrt{\sum \sum W_{ij} \left(y_{ij} - m_j\left(t_i, \theta\right)\right)}$$

8.2.3 Observations

When minimizing the zero of the function is not the final objective, because if the residual error is zero, the data is being overfitted. Furthermore a great attention has to be posed to the construction of the model. During optimization θ, the space of the parameters is being manipulated, modifying the output in the space of observations. In this way abstract values are being linked to observations.

Chapter 9

Gradient methods

9.1 Introduction

When the model needs to integrate to find an ODE solution, the output is points in that integrated function. f and f' are not known of the optimization problem, so a numerical approximation is needed. The idea of gradient methods is to set $f' = 0$. Gradient methods can be applied to both constraint and unconstrained models. Integrating constraints the problem becomes:

$$\begin{cases} \max f(x) \\ g_i(x) = 0, \quad i \in \mathcal{I} \\ x \in \mathbb{R}^N \end{cases}$$

Where $\mathcal{I} = 1, ..., m$. The traditional way to solve this problem is to translate this system to another function.

9.1.1 Lagrangian function

The Lagrangian function is used to take into account the constraints. A Lagrangian function is defined as the function:

$$L : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$$

Such that:

$$L(x) = f(x) + \lambda g(x) = f(x) + \sum_{j=1}^{m} \lambda_j g_j(x)$$

9.1.2 Lagrangian Multipliers Theorem

If x^* is a stationary point for a Lagrangian function, then $\exists \lambda^*$ such that (x^*, λ^*) is a stationary point for L. This is a necessary condition that increases the dimensionality of the problem from $\mathbb{R}^N$ to $\mathbb{R}^N \times \mathbb{R}^m$, but it allows to search solutions using the stationary points. This can be generalized to $g_i(x) \leq 0$ constraints. Stationary points are not necessarily minima and maxima. This is enforced by checking through second derivations or evaluating the function in other points.

9.1.3 Definition of a gradient

Let $f : \mathbb{R}^N \to \mathbb{R}$ a differentiable function, the gradient of f is:

$$\nabla f : \mathbb{R}^N \to \mathbb{R}^N$$

Where:

$$\nabla f_i = \frac{\partial}{\partial x_i} f(x)_i$$

And:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \vdots \\ \frac{y}{\partial x_N} f(x) \end{bmatrix}$$

The objective is looking for points for which the derivative vanishes:

$$x^* : \nabla f(x^*) = 0$$

9.1.3.1 Some examples

Two functions used to test optimization algorithms are the Rosenbrock's function:

$$f(x, y) = (1 - x)^2 + 100 \left(y - x^2\right)^2$$

And the Eggholder function:

$$f(x, y) = -(y + 47) \sin\left(\sqrt{\left|\frac{x}{2} + (y + 47)\right|}\right) - x \cdot \sin\left(\sqrt{|x - (y + 47)|}\right)$$

Which are used to test optimization algorithms. Solving analytically these two problems is hard, so a numerical solution has to be employed.

9.1.4 Limitations of gradient descent methods

One of the major limitations of these algorithms is that there is no guarantee to reach the global minimum. Furthermore it could be the case that variables cannot be optimized together, so there is a need to make a trade-off between them. Moreover gradient methods cannot be applied to stochastic simulations as their functions are not continuous.

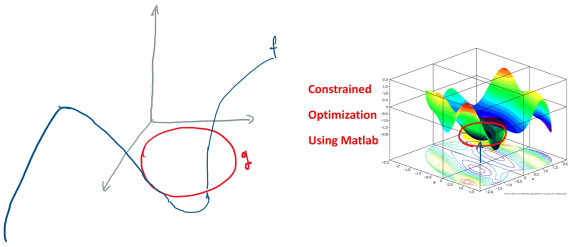

Figure 9.1: Blue = function, red = constraint

Figure 9.1 is an example of an objective function. If there is an equality constraint only the points meeting the boundary, in red, are to be considered. In an inequality constraint instead everything inside the red circle is considered. Generally constraints reduce the search space: the Lagrangian determines that the minimum point with some multipliers will give a solution of the Lagrangian. Finding the solution of the original condition. These could not be solutions of the original conditions, but they are ideal candidates that can be checked. To perform an evaluation of the distance the model should be integrated. This is computational expensive, so the measure of computation is the number of times the model has to be simulated per iteration.

9.2 Gradient approximation with Taylor formula

In most cases the gradient is unknown and it should be approximated using the Taylor formula. Let $]a,b[\in\mathbb{R}, x_0 \in]a,b[$ and $f_i]a,b[\to\mathbb{R}$ be differentiable $(n-1)$ times in $]a,b[$ and $f^{(n)}$ continuous in x_0. Then let $x \in]a,b[$:

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + f''(x_0)\frac{(x-x_0)^2}{2!} + ... + f^{(n)}(x_0)\frac{(x-x_0)^n}{n!} + R_n(x)$$

Such that:

$$\lim_{x\to x_0}\frac{R_n(x)}{(x-x_0)^n} = 0$$

Focussing on the first terms:

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + R_2(x) = 0$$

$$f'(x_0) = \frac{f(x)-f(x_0)}{x-x_0} + \left(\frac{R_2(x)}{(x-x_0)}\right) \approx \frac{f(x)-f(x_0)}{x-x_0} + R_1(x)$$

This trick can be used for any $N > 1$. Now let $f : \mathbb{R}^N \to \mathbb{R}$ and $e_i = (0,\dots,0,1,0,\dots,0)$. Consider $x_1, x+\varepsilon\ell_i, x-\varepsilon\ell_i$ so that the gradient is being explored in one direction:

$$\begin{aligned} f(x+\varepsilon e_i) &= f(x) + \varepsilon\frac{\partial f}{\partial x_i}(x) + \frac{1}{2}\varepsilon^2\frac{\partial^2 f}{\partial {x_i}^2}(x) + R_3(x) \\ f(x-\varepsilon e_i) &= f(x) - \varepsilon\frac{\partial f}{\partial x_i}(x) + \frac{1}{2}\varepsilon^2\frac{\partial^2 f}{\partial {x_i}^2}(x) + R_3(x) \\ \Rightarrow f(x+\varepsilon e_i) - f(x-\varepsilon e_i) &= +2\varepsilon\frac{\partial f}{\partial x_i}(x) + R_3(x) \\ \Rightarrow \frac{\partial f}{\partial x_i}(x) &= \frac{f(x+\varepsilon e_i) - f(x-\varepsilon e_i)}{2\varepsilon} + R_2(x) \end{aligned}$$

This is an approximation of the first derivative with improved accuracy. This applies to only one derivative, so it has to be performed at least twice as 2 function evaluation are needed for each $i \to 2W$. In order to obtain a decent gradient, a lot of computations are required, but these are fast as they have a low number of iterations. The algorithm converges when $\Delta f = 0$.

9.2.1 Vanishing gradient

There might be points where the gradient vanishes which are not the final destination. Gradient methods may tend to overfit, but they are effective. The main issue is that since the gradient is approximated, it cannot be trusted everywhere.

9.3 Line search

The line search is a class of algorithms that follow a direction along the gradient.

9.3.1 Newton's direction

Considering Taylor's formula and letting x_k be the starting point, $\alpha \in \mathbb{R}^+$ the step length and p the direction such that $(x_k, p \in R^n)$. For $n = 1$:

$$f(x_k + \alpha p) = f(x_k) + \alpha p f'(x_k) + \frac{\alpha^2 p^2}{2} f''(x_k) + r(p^3)$$

For simplicity set $\alpha = 1$ and truncate the formula:

$$f(x_n + p) = f(x_n) + p f'(x_n) + \frac{p^2}{2} f''(x_n) = m_k(p)$$

Now instead of f the simple polynomial $m_k(p)$ is minimized:

$$\frac{d}{dp} m_k(p) = f'(x_n) + p f''(x_n) = 0$$

From this equation the direction is:

$$p = -\frac{f'(x_k)}{f''(x_k)}$$

Or the Newton direction, which is the best direction to move the gradient. When $n > 1$:

$$p = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

Where:

$$\nabla^2 f(x) = \left[\frac{\partial}{\partial x_i} \left(\frac{\partial}{\partial x_j} f(x) \right) \right]_{i,j}$$

Is the Hessian matrix. Finding the inverse of a matrix is not straightforward, so next algorithm will try to approximate it. The second derivative progressively shrinks when the algorithm approaches convergence to reduce the step size.

9.3.1.1 Algorithm

An algorithm for line search with Newton's direction for function minimization can be found in 50.

Algorithm 50: `Line search()`

Input: the function f to be minimized and x_k starting point
Output: the minimized function
while $||\nabla f(x_k)|| > \epsilon$ **do**
    $p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
    select α_k
    $x_{k+1} = x_k + \alpha_k p_k = x_k - \alpha_k (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
    $k = k + 1$

The stopping point is when the gradient is equal to zero, but there is a need to apply a threshold ϵ on it. Now at each iteration sometimes approximation is needed to compute the inverse of the matrix. Moreover all these gradients are computationally expensive to compute. Nevertheless this process of going in one direction is smart. The smartest part, or the computation of p_k can be approximated to make the computation less expensive.

9.3.2 Quasi Newton's direction

To avoid the computation of $\nabla^2 f$, a different matrix B_k is built, such that

$$B_{k+1} \cdot (x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k)$$

The second derivate is being approximated, the gradient is exploited to prevent extra computation. The Hessian where will not be computed. So the quasi-Newton direction will be:

$$p_{k+1} = -B_{k+1}^{-1} \nabla f_{k+1}$$

The progressive shrinking of the second derivative indicates that the step needs to be reduced. The step α should be changed according to the Armijio condition. Consider in particular that each time that an operation on a matrix is performed precision is lost.

9.3.3 Steepest descent direction

In steepest descent the direction chosen is the one that reduces the gradient the most:

$$p = -\alpha \frac{\nabla f}{||\nabla f||}$$

The second derivative is being ignored and only the gradient computation is needed. This method converges more slowly with respect to Newton and Quasi Newton direction methods.

9.3.4 Selecting α

To determine the step length α consider:

$$\phi(\alpha) = f(x_k + \alpha p_k)$$

The objective is to define $\bar{\alpha}$ such that:

$$\phi(\bar{\alpha}) = \min_{\alpha > 0} \phi(\alpha)$$

This is another optimization problem, which could be avoided by selecting an α that satisfies other conditions

9.3.4.1 Armijio condition

The Armijio condition forces a selection of α such that:

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f(x_n)^T p_k \qquad c_1 \in]0, 1[$$

9.3.4.2 Curvature condition

The curvature condition forces a selection of α such that:

$$\nabla f(x_k + \alpha p_k)^T p_k \geq c_2 \nabla^T f_k p_x \qquad c_2 \in]\epsilon_1, 1[$$

9.3.5 Convergence of a method

The order of convergence of a method is a constant ℓ, such that if x^* is a solution, the limit:

$$\lim_{k \to \infty} \frac{||f(x_{k+1}) - f(x^*)||}{||f(x_k) - f(x^*)||^\ell} = L > 0$$

Exists. The new iteration is compared to the old one. The limit should go to 0 and the exponent is the speed at which this happens: the higher ℓ the faster the numerator go to zero with the respect to the denominator, so the higher the exponent, the faster the convergence. For the methods discussed previously:

- Steepest descent: $\ell = 1$, linear convergence.
- Quasi-Newton: $\ell \in (1, 2)$, superlinear convergence.
- Newton: $\ell = 2$, quadratic convergence.

9.4 Trust region

Trust region is a class of algorithms that create an approximation of the problem and they solve it in a small trustable region. The objective is to optimize a problem such that close to a starting point x_k the model is close to the function.

9.4.1 Trust region steepest descent

Let now $f_k = f(x_k)$. The model can be approximated through a Taylor expansion:

$$m(x_p + \alpha p) = f(x_k) + \alpha p^T \nabla f(x_k) + \frac{1}{2} \alpha^2 p^T B_k p$$

Where B_k can be the Hessian matrix or its approximation. This problem can be solved in a region such that:

$$||p||_2 < \delta_k \qquad \delta_k > 0$$

Now assume for simplicity that $\alpha = 1$ and that $B_k = 0$. The model of the function will be now:

$$m_k(p) = f_k + p^T \nabla f_k$$

To minimize this, since f_k is a constant there is a need to work only on the second term:

$$p^T \nabla f_k = ||p|| \cdot ||\nabla f_k|| \cdot \cos\theta$$

The minimum is reached when $\cos\theta = -1$ and $||p|| = \delta_k$, so that $\bar{p}$:

$$\bar{p}^T \nabla f_k = -\delta_k ||\nabla f_k|| \Rightarrow \bar{p} = -\delta_k \frac{\nabla f_k}{||\nabla f_k||}$$

This result is exactly the equation from steepest descent. A condition on the region with δ_k is being applied. This direction and the whole approach is called trust region steepest descent. The same idea could be followed by applying Newton or Quasi-Newton.

9.4.2 Evaluating the trust regions

To evaluate the trust regions, the actual reduction is defined as:

$$\rho_k = \frac{f(x_k) - f(x_k + p)}{m(x_k) - m(x_k + p)}$$

By definition $m(x_k + p) \leq m(x_k)$, so the denominator is always > 0.

- If $\rho_k < 0$, reject p, as the real problem is not being improved. δ_k should be reduced, typically by $\frac{1}{4}\delta_k$.
- The closer ρ_k is to 1 the closer the model m_k to f, so bigger steps can be taken increasing δ_k.

The value of δ can be tuned according to the needs of the problem. The approach is similar to the RK method. A grey area between 0 and 1 is found, so a threshold is defined such that $\rho_k < \eta$ and$\rho_k > \eta$.

9.4.3 Trust region algorithm

An implementation of the trust region method is outlined in algorithm 51.

Algorithm 51: `Trust region()`

Input: the function f to be minimized and x_k starting point, $\hat{\delta}$ the maximum accepted region, the starting $\delta \in]0, \hat{\delta}[$ and $\eta \in \left[0, \frac{1}{4}\right]$ the minimum actual reduction for which the direction is accepted.
Output: the minimized function
repeat
    obtain p_k such that $p_k = \arg \min_{\substack{p \in \mathbb{R}^n \\ ||p|| \leq \delta_k}} m(x_k + p)$
    compute ρ_k
    if $\rho_k < \frac{1}{4}$ **then**
        $\delta_{k+1} = \frac{1}{4}\delta_k$
    else if $\rho_k > \frac{3}{4} \wedge ||p_k|| = \delta_k$ **then**
        $\delta_{k+1} = \min(2\delta_k, \hat{\delta})$
    else
        $\delta_{k+1} = \delta_k$
    if $\rho_k > \eta$ **then**
        $x_{k+1} = x_k + p_k$
    else
        $x_{k+1} = x_k$
until $||\nabla f(x_k)|| < \epsilon$

The stopping point is when the gradient is sufficiently small. The focus is on computing p_k, then ρ_k is evaluated to adjust the parameters.

Chapter 10

Least squares problems

10.1 Introduction

Let $\{y_i\}_{i=1,\dots,n}$ be the observed data and $m(t_i, \vec{\theta})_{i=1,\dots,n}$ be the corresponding model predition. Then the objective function is defined as:

$$f(\vec{\theta}) = \frac{1}{2}\sum_{j=1}^{n} r_j(\vec{\theta}) = \frac{1}{2}||r(\vec{\theta})||_2^2$$

Where:

- $r_j(\vec{\theta}) = y_j - m(t_j\vec{\theta})$.
- $r(\vec{\theta}) = (r_1(\vec{\theta}), \dots, r_n(\vec{\theta}))^T$ is called the residual.

In general the residual is a matrix such that:

$$r(\vec{\theta}) = [r_{ij}(\vec{\theta})]_{\substack{i=i,\dots,n \\ j=1,\dots,m}}$$

Where:

- i iterates over n residuals.
- j iterates over m observations.

The matrix can be reshaped into a vector, considering a vector of residuals. Defining:

$$J_k = J(\theta_k)\left[\frac{\partial r_I}{\partial \theta_i}\right]_{\substack{i=1,\dots,n \\ I=1,\dots n}}$$

So that:

$$f(\vec{\theta}) = \frac{1}{2}||r(\vec{\theta})||_2^2 = \frac{1}{2}\sum_{j=1}^{n} r_j^2(\vec{\theta})$$

$$\nabla f(\vec{\theta}) = \sum_{j=1}^{n} r_j \nabla r_j(\vec{\theta}) = J(\vec{\theta})^T r(\vec{\theta})$$

$$\nabla^2 f(\vec{\theta}) = J^T(\vec{\theta})J(\vec{\theta}) + \sum_{j=1}^{n} r_j \nabla^2 r_j(\vec{\theta})$$

Where $J^T(\vec{\theta})J(\vec{\theta})$ is easy to compute as it is known without new derivative computations and it can be used to approximate $\nabla^2 f(\vec{\theta})$.

10.1.1 Linear problem

The linear problem is the simplest case. Consider to describe a variable $y = A_{\vec{\theta}}$, where A is a matrix, $\vec{\theta}$ is the vector of parameters and $\bar{y}$ the observations. The residuals can be defined as:

- $r(\vec{\theta}) = A_{\vec{\theta}} - \bar{y}$.
- $f(\vec{\theta}) = ||A_{\vec{\theta}} - \bar{y}||^2$.
- $\nabla f(\theta) = A^T(A_{\vec{\theta}} - \bar{y})$.
- $\nabla^2 f = A^T A$.

If f is convex then:

$$\exists \theta^* \text{such that} \nabla f(\vec{\theta^*}) = 0 \Leftrightarrow A^T A_{\vec{\theta^*}} = A^T \bar{y}$$

Reaching a normal equation with a linear system.

10.1.2 Non linear least squares problem

For non-linear least square problem the first approach is a modified line-search Newton method. The problem of quantifying the distance between model and data can be formalized as:

$$f(\theta) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(\theta)\nabla f(\theta) = J(\theta)^T r(\theta)\nabla f(\theta) = J(\theta)^T J(\theta) + \sum_{j=1}^{m} r_j \theta \nabla^2 r_3 \theta$$

The sum can be ignored as:

- It contains second order derivatives, which are expensive to compute.
- The Newton direction is a quasi vector.

At every iteration k the approximated problem to be solved is:

$$J(\theta_k)^T J(\theta_k)p = -J(\theta_k)r(\theta_k)J_k^T J_k p = -J_k^T r_k$$

This is a linear system that is solved as:

$$f(\theta_k + p) = \frac{1}{2}||r(\theta_k + p)||^2 \simeq \frac{1}{2}||J_k p + r_k||^2$$

This is the normal equation for a normal least squares problem. Under certain hypotheses the method converges quadratically.

10.1.2.1 Summary

Recalling that the Newton method is quadratic convergent, the matrix is not singular. The system should be solved starting with a problem in the form of sum of squares. Thanks to this, the problem can be rewritten in a simpler way and an approximation can be performed, leading to solving a linear system at each iteration. This approximation guides rapidly to a solution. Linear search is being applied, by defining a direction and solving a new problem at each iteration. The biggest issue could be the non invertible matrix, but this can be solved. It should be noted that the approximation of the second order derivative could cause a over-estimation of the term.

10.2 The Levenberg-Marquardt method

The Levenberg-Markquardt method is a combination of the Gauss Newton with trust region. At each iteration the problem to be solved is:

$$\min_{||p|| \leq \delta_k} \frac{1}{2}||J_k p - r_k||^2$$

A solution can be inside inside the trust region δ_k or on the border, which is not the minimum, but the smallest value that can be reached. If a solution is inside the region is accepted. Otherwise if $||p|| \leq \delta_k$ p is a solution if and only if:

$$\exists \lambda \geq 0 \text{ such that } (J^T J + \lambda I)p = -J^T r$$

The general problem that needs to be solved is:

$$\begin{cases} (J^T J + \lambda I)p = -J^T r \\ \lambda(\delta_k - ||p||) = 0 \end{cases}$$

- if β is a solution and $||p|| \leq \delta_k$ the algorithm ends. — if $\exists \lambda > 0$:
- if $||p|| = \delta_k$, then $\bar{p}$ is a solution if and only — $(J^T J - \lambda I) = -J^T r + \lambda(\delta_k - ||p||) = 0$

If moving a bit from zero and the problem is still solved, this can be thought as a solution. If the boundary is being hit: $\lambda(\delta_k - ||p||) = 0$, the matrix has to be moved a bit from the singularity. The Levenberg-Marquardt is one of MATLAB default solvers and it converges quadratically when t is close to the solution, while if the residuals are big it does not perform well.

10.2.1 Bounds

The Lagrangian can be exploited for the equality constraint, but other boundaries can be had. Other approaches allow to not lose the approximation advantage. For example variable transformation can be applied. Some example of variable transformations are:

$$x \mapsto e^x \qquad \mathbb{R} \mapsto \mathbb{R}_0^+$$
$$x \mapsto \frac{e^x}{1+e^x} \qquad \mathbb{R} \mapsto [0,1]$$
$$x \mapsto a + (b-a)\frac{e^x}{1+e^x} \qquad \mathbb{R} \mapsto [a,b]$$

Another solution could be to include the bound in the trust region. There is a need to check at every iteration if the bounds are being satisfied and the actual reduction. Evolutionary algorithms follow the evolution of the solution: at the beginning a solution is built a penalty for the boundaries is added. This is called in Matlab the trust region reflective and it is the default method when defining a problem with bounds.

10.2.2 Solving global minimum problem

This method has no guarantee to converge to the global minimum. To solve the global minimum problem a multi-start approach could be taken. In this approach the parameter optimization is started from different starting point and it is checked whether the same minimum is obtained.

10.3 Gauss-Newton method

The global optimization problem is solved with a multi-start approach, so the procedure is repeated with different starting point. Let $N \in \mathbb{N}, \varepsilon > 0, J$ as defined before. Select randomly N vectors:
$\theta^1, \ldots, \theta^n \in \mathbb{R}^n$
The multi start approach for Gauss-Newton method is outlined in algorithm 52.

Algorithm 52: `Multi start approach for Gauss-Newton()`

Input: the Jacobian J and the N vectors of θ
Output: the optimal vector Θ
foreach $i = 1, \ldots, N$ **do**
  repeat
    compute q such that $\bar{J}^i \bar{J} q = -\bar{J} r(\theta^i)$
    $\vartheta = \theta^i + q$
    $\epsilon = \left|\left| \frac{\theta - \vartheta}{\theta} \right|\right|$, the relative increase or $\epsilon = ||\bar{J}(\vartheta)$
    $\theta^i = \vartheta$
  until $\epsilon < \delta$
  store θ_i
return θ_i such that $r(\theta^i = \min\{||r(\theta^1)||, \ldots, ||r(\theta^N)\})$

This algorithm does not guarantee that the global minimum is obtained.

10.3.1 Determining the initial points

Randomly selecting the points may lead to have cluster of points and parts of the space not considered, which could lead to unsuccessful optimization. To better spread the points two strategies can be applied.

10.3.1.1 Latin hypercube

The Latin hypercube divides the space in equiprobable intervals or subspaces. It assures that for each interval row and column there is only one point selected. This is the most used method.

10.3.1.2 Orthogonal cube

The orthogonal cube divides the space in a equi-probable sub spaces and inside each sub space there are equally dense points. The total set of points is a Latin hypercube.

10.3.2 Termination criteria

Different termination criteria could be employed.

- Gradient equal to zero:

$$||\nabla f|| = 0 \qquad \vee \qquad ||\nabla f|| < \epsilon$$

- Relative step size:

$$\left|\left|\frac{\theta^i - \vartheta}{\theta^i}\right|\right| < \epsilon$$

- Relative function change:

$$\left|\left|\frac{f(\theta^i) - f(\vartheta)}{f(\theta^i)}\right|\right| < \epsilon$$

- Maximum number of iterations.
- Maximum number of function evaluations.

10.4 Matlab

In Matlab, when an optimizer stops, it gives as output the function converged to the solution or why it stopped. Termination criteria are:

- Gradient.
- Incremental step: if the change was less than a tolerance the residual is less.
- Number of iteration.

According to different starting points, different results will be obtained.

10.4.1 Quantifying the fit

To quantify how much the model is distant from the real data consider the output of the normal simulation: this is a set of points, A linear interpolation for time and values can be performed, giving how much is each of the simulated variables at the requested time. Once the values at the right time are computed, the residuals can be calculated from the sum of squares.

10.4.2 Optimizing

To optimize in Matlab the function `lsqnonlin` can be used to improve the initial conditions.

10.4.3 Matlab multi-start

Matlab multi-start is a wrapper working with different algorithms. It will automatically parallelize the problem. It requires to insert starting points and tolerance. Then the bounds are defined, the problem is created and the initial input is given as input. The output contains the result of parallel lsqnonlin and parameters, somehow similar to what we saw before. The bounds can be reduced, but solution 1 of figure 10.1 is still the best. The main limitation is that this algorithm heavily depends on the initial points.

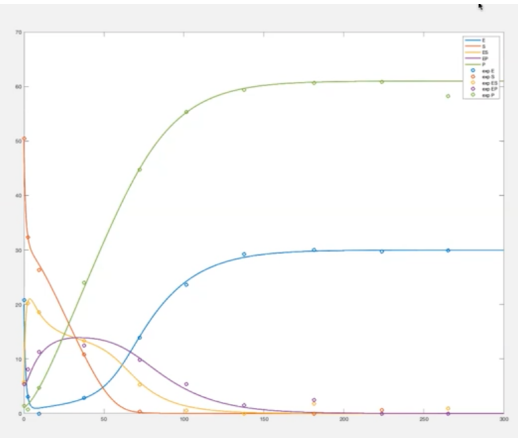

Figure 10.1: MATLAB multi-start lsqnonlin

Chapter 11

Stochastic methods for parameter estimation

11.1 Introduction

Stochastic methods are often used when the gradient cannot be computed. With the same parameters, stochastic methods can produce dramatically different results, as it can be seen in figure 11.1.

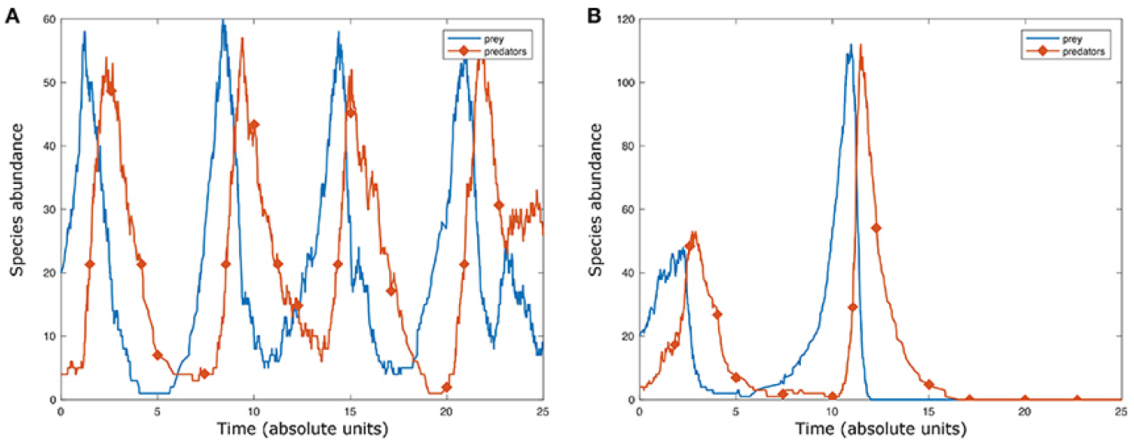

Figure 11.1: Lotka-Volterra stochastic simulation results. In B we witness preys and predators extinction, it is the output of a simulation performed with the same model and parameters as A.

Instead of following the gradient informations are collected in a manner resembling natural selection. Monte Carlo Methods are dated around 1949, the name comes from casinos. In order to use MCM to do inference, Metropoli and Hastings developed a specific algorithm in 1970.

11.2 Markov Chain Monte Carlo (MCMC)

MCMC is a chain of events where the current state depends on the previous one and on the transition probability. Running MCMC long enough they will reach a stable point. $x^{(i)}$ is a random variable or stochastic process that takes only discrete values $\{x_1, \ldots, x_s\}$. Let $p(x)$ be the probability distribution of x.

11.2.1 Markov chain

$x^{(i)}$ is a Markov Chain if:

$$p(x^{(i)}|x^{(i-1)}, \ldots, x^{(1)}) = T(x^{(i)}|x^{(i-1)})$$

11.2.1.1 Homogeneity

A Markov chain is homogenous if $T(x^{(i)})x^{(i-1)} = T$, So that the transition matrix is constant.

11.2.2 Example

Let 3 states as in figure 11.2.

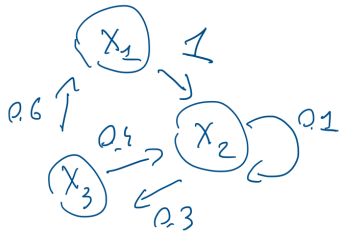

Figure 11.2: MCMC example

The homogeneous transition matrix:

$$T = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0.1 & 0.9 \\ 0.6 & 0.4 & 0 \end{bmatrix}$$

And the states:

$$\pi_1 = \begin{pmatrix} 0.5 & 0.2 & 0.3 \end{pmatrix}$$

The next probability of being in the three states is given by

$$\pi_1 \cdot T = \begin{pmatrix} 0.3 \cdot 0.6, & 0.5 + 0.02 + 0.12, & 0.18 \end{pmatrix} = \begin{pmatrix} 0.18, & 0.64, & 0.18 \end{pmatrix}$$

Iterating enough time a fixed distribution will be reached.

11.2.3 Invariant distribution

The invariant distribution if a fixed distribution reached after enough Monte Carlo iterations. The objective is to build a Markov chain whose invariant distribution is the distribution of the unknown parameters. For example, in the previous example, the invariant distribution will be:

$$p(x) = \cdots = \begin{pmatrix} 0.2213 \\ 0.4098 \\ 0.3689 \end{pmatrix}^T$$

11.2.4 Irreducible matrices

A stochastic matrix is irreducible if its graph does not contain unconnected sub-graphs. If T is an irreducible transition matrix and is aperiodic the Markov chain has an invariant distribution. The probability distribution has to be linked with the model.

11.2.5 Monte Carlo sampling

Monte Carlo refers to a family of methods developed for sampling. In particular, the output of a Monte Carlo method are independent and identically distributed random variables $\{x^{(i)}\}_{i>0}$ from a known density $p(X)$. The samples can be used to "estimate or approximate" a target density or distribution. Known distributions are exploited to approximate the unknown distribution of interest. Given enough samples, p can be approximated as:

$$p_N(x) = \frac{1}{N} \sum_{i=1}^{N} \delta_{x^{(i)}}(x)$$

With:

$$\delta_{x^{(i)}}(x) = \begin{cases} 1, & x = x^{(i)} \\ 0, & \text{elsewhere} \end{cases}$$

This is often used to approximate "hard" integrals. An integral with a finite sum of values, can be obtained as:

$$I_N(f) = \frac{1}{N} \sum_{i=1}^{N} f(x^{(i)})$$

The function is being evaluated as some points and the mean is an integration of the integral. This equation converges to the real integral by the law of large numbers:

$$I_N(f) = \frac{1}{N} \sum_{i=1}^{N} f(x^{(i)}) \xrightarrow[N \to \infty]{} I(f) = \int_x f(x) p(x) dx$$

This is better than the output of the least square problem as it gives information about the shape of the distribution.

11.3 Sampling a distribution

Let p be a known probability distribution up to a proportionality constant. It is usually preferred to sample from a well-known distribution, like for example the one in 11.3.

$$q(x) \text{ such that } \exists M > 0 : \qquad p(x) \leq Mq(x)$$

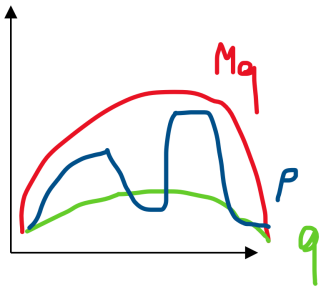

Figure 11.3: Example: bimodal distribution

Values from q are extracted if they are smaller than Mq. Each time information about p are collected from q.

11.3.1 Rejection sampling algorithm

A rejection sampling implementation is outlined in algorithm 53.

Algorithm 53: `Rejection sampling()`

Input: M, $p(x)$ and $q(x)$, N the number of numbers to be sampled
Output: a vector of random numbers distributed according to p
$i = 1$
repeat
  sample $x^{(i)} \sim q$, $u \sim norm(0, 1)$
  if $u < \frac{p(x^{(i)})}{Mq(X^i)}$ **then**
    accept and store $x^{(i)}$
    $i = i + 1$
  else
    reject $x^{(i)}$
until $i = N$

If the point is in the area between p and Mq is accepted. The further the ratio is from 1, the smaller the chance of keeping the point. A known distribution p is being used, then gradually what is known about it is removed in the following iterations.

11.4 Metropolis Hastings

Let X be the current state vector, q the proposal distribution, p the target distribution. p can be an unnormalized density ($\int p > 1$ but $\int p < \infty$). Ideally p should be known, but if it is not the procedure still works. Let h be the non negative and positive integral of the normalized density distribution. The trick of MH is to avoid relying on the probability: a function is used instead. Let x^* be the new candidate point, then the metropolis ratio is defined as:

$$r_M(x, x^*) = \frac{h(x^*)}{h(x)}$$

And the Hasting ratio:

$$r_H(x, x^*) = \frac{h(x^*) \cdot q(x^*|x)}{h(x) \cdot q(x|x^*)}$$

From the distribution, the objective is to measure how likely the previous value is given the current one. The choices is being weighted to the likeliness.

11.4.1 Algorithm

An implementation of the Metropolis Hastings method is outlined in algorithm 54

Algorithm 54: `Metropolis Hastings()`

Input: N, r and $q(x)$
Output: a random variable distributed according to p
guess $x^{(0)}$
foreach $i = 0, \ldots, N-1$ **do**
    sample $x^{(*)} \sim q(\cdot|x^{(i)})$ and $u \sim norm(0, 1)$
    if $u < \min\{1, r_H(x, x^*)\}$ **then**
        $x^{(i+1)} = x^{(*)}$
    else
        $x^{(i+1)} = x^{(i)}$

So in this algorithm the proposal distribution is sampled and the sample is accepted if it is more likely, if it is not it is accepted with a certain chance.

11.4.2 Visualization of MCMC

MATLAB plot:

- right: histogram with the known distribution
- left: MCMC oscillating between 0 and 10, recapitulates the distribution

The procedure is really fast, takes less than a second. If instead of extracting u and v from random distribution with an index, the result is the same → "thanks MATLAB, it is not necessary to pre-locate anymore"

If we employ a Gaussian distribution, the result is still good but not as accurate as the previous one. We are adding complexity by introducing the variance; by choosing a different value we

can improve the result. Pay attention to the fact that if we sample from a narrow distribution, we risk focusing only on one of the two points.

Everything we do has consequences!
`http://chi-feng.github.io/mcmc-demo/app.html?algorithm=RandomWalkMH&target=banana`
Green: accept, red: reject

Target distribution = standard

- GibbsSampling: collects points according to a certain direction from a starting point, it tries to rebuild a 2D Normal distribution.
- AdaptiveMH: sample from a starting mean and accept or reject new points.
- Random walk: more or less like MH. Even if the target distribution is not that difficult (bell shape), there are a lot of rejections initially.
- DE-MCMC-Z: produces vectors.

Target distribution = donut

- SVGO: stochastic vector gradient descent, intermediate between gradient and stochastic.
- EfficientNUTS (No-U-Turn samples): it creates many points, more complex but one of the best. In the end it will converge quite fast.
- RandomWalk: needs more time to converge with respect to standard distribution.

Target distribution = multimodal

- RandomWalk: three initial points, it proposes a random perturbation.
- AdaptiveMH: changes some of the parameters. Differently from the previous approach, the shape changes: instead of having a fixed search area, it evolves and adapts.

11.5 How to link data and Metropolis-Hasings

In least square methods, the function used to link data and prediction that was used was:

$$f(\theta) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(\theta) = \frac{1}{2}\sum_{j=1}^{m}(y_j - m_j(t_i, \theta))^2$$

The residuals can be weighted for their uncertainty:

$$f_w(\theta) = \frac{1}{2}\sum_{j=1}^{m} \frac{r_j^2(\theta)}{\vartheta_j^2}$$

ϑ_j can be, for example, the standard deviation of that measured point. Let Y be the vector of observations, a function that can be used to link the parameters and the output can be the non negative:

$$L(\theta|Y) = e^{-f_w(\theta)} \geq 0$$

This can be used as the h in the MH algorithm, so that the Metropolis ratio:

$$r_H(\theta^*, \theta) = \frac{L(\theta^*)}{L(\theta)} = \frac{e^{-f_w(\theta^*)}}{e^{-f_w(\theta)}} = e^{-f_w(\theta^*)+-f_w(\theta)}$$

Since $f_w(\theta) \geq 0$, if $f_w(\theta^*) > f_w(\theta) \Rightarrow r_H > 1$, the number is accepted, otherwise it is accepted according to a certain probability. This function can be used to link new parameters with data. Another example of this type of function is the log-likelihood, considering:

$$f_w(\theta^*) \qquad \wedge \qquad f_w(\theta)$$

So that $f_w(\cdot)$ is the h of the Metropolis-Hastings algorithm.

11.6 Random walk MCMC

Assume that θ^* is sampled from:

$$\mathcal{N}(0, \mathcal{C}) + \theta = \mathcal{N}(\theta, \mathcal{C})$$

Now a perturbation is added with $\mathcal{C}$, the covariance matrix, which can be fixed or adapted along the iterations. From the initial point there is a new candidate according to a multivariate normal distribution. If the candidate is accepted, the evaluation is restarted from it. This allows to collect informations about the shape of the target distribution.

11.6.1 Algorithm RW-MCMC (C known)

An implementation of the RW-MCMC is oulined in algorithm 55.

Algorithm 55: `Random walk Markov Chain Monte Carlo (RW-MCMC)()`

Input: L, θ
Output: a random variable distributed according to p
initialize matrix D_θ and vector V_L
generate randomly θ_1
$L(\theta_1) = L_1$
foreach $i = 1, \ldots, N$ **do**
    $Z \sim norm(D, \mathcal{C})$
    $\theta_2 = \theta_1 + Z$
    $L_2 = L(\theta_2)$
    $ratio = \frac{L_1}{L_2}$
    $u \sim norm(0, 1)$
    if $u < ratio$ **then**
        $L_1 = L_2$
        $\theta_1 = \theta_2$
    if $i > warm\_up$ **then**
        $D_\theta = [D_\theta; \theta_1]$
        $V_L = [V_L; L_1]$

11.6.2 Discussion

To reach a satisfactory convergence a good accept-reject tradeoff should be obtained. The advantages of RW-MCMC are:

- The object function is evaluated once per iteration.
- The target distribution can be built form the samples.
- Only the last point is necessary for the algorithm.
- Info on the model can be collected: the variability on the output and the sensitivity on parameters.
- Random selection helps to avoid local minima.

The disadvantages of RW-MCMC are:

- Convergence may be slow.
- The sampling distribution may affect the results.
- Each MCMC cannot be parallelized.
- Burn-time.
- Diagnostics are heuristics.

11.6.3 Diagnostics

Diagnostics are used to checked the goodness of the parameters and whether the iterations have finished. Consider for example figure 11.4:

- chain 1: expected MCMC plot.
- chain 2: requires a longer burn-in, as the target distribution has not been reached.
- chain 3: there aren't enough information because the number of iterations is too small.

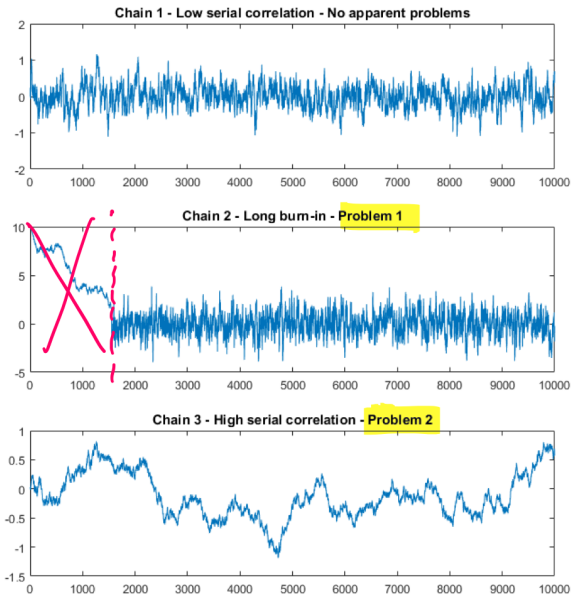

Figure 11.4: Diagnostic 1

The desired output of MCMC should look like the plots in figure 11.5.

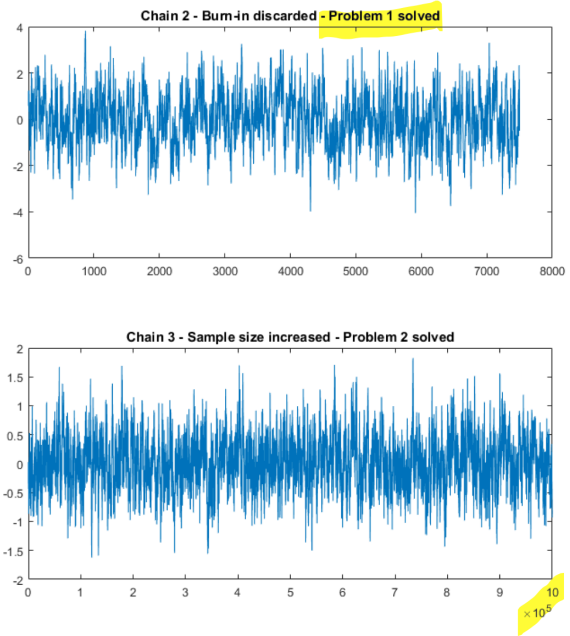

Figure 11.5: Diagnostic 2

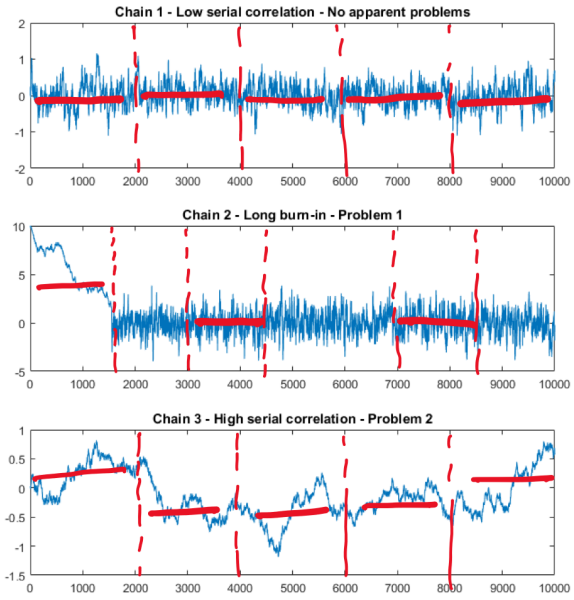

Figure 11.6: Diagnostic 3

11.6.3.1 Sample split

Sample split is a more analytical approach in which the plots are split in different regions and it is checked whether the sub-regions have the same mean. It can be seen in figure 11.6 there is no longer a need for warm-up in the second plot, while in the third more samples are still needed.

11.6.3.2 Conclusions

Differently from gradient methods, here things are a bit more hard to interpret. Eventually oscillations around the global optimum will be reached, but there is no guarantee. Diagnostics are based on the observation of the results, so they might be biased by the observers' believes and they do not provide an easy way to read a "certificate" of convergence. Another approach may be to run more MCMC in parallel and see if they all converge to the same distribution. However, it is again not a certificate of convergence to the global optimum. Constraints and bounds can be included easily in the proposal distribution or in the likelihood.

Chapter 12

Heuristics and the genetic algorithm

12.1 Introduction

Heuristics mimic natural selection: they follow nature inspired procedures. At the growing of computational power, they become feasible ways to solve optimization problems. There are no warranties on the exactness of the solutions, but often times the results are of high quality. In addition, heuristic algorithms are easy to implement, are general and can include even complex constraints.

12.1.1 An example

Consider for example the timetable for plane departures. Deciding when a plane lands is not only a function of flight time, as different things should be taken into account as:

- Delays.
- Passengers.
- Luggages.
- Crew.

As constraints. In biological problems relationships among variables which should be satisfied can be included, allowing for a high flexibility.

12.1.2 Famous heuristic algorithms

There are many famous heuristic algorithms:

- Simulated annealing: used in physics, match thermal dispersion.
- Ant colonies: ant are able to solve many problems like the supply chain one.
- Ccovariance matrix adaptation evolution strategy: performs similarly to adaptive MH algorithms.

12.1.3 Evolution strategy and population

All these methods have in common an evolution strategy. To observe evolution, these methods have in common an evolution strategy and a population of candidate solution. These solutions are selected according to their fitness, or objective function. The changes in populations occur as a results of variations on the current population.

12.2 The genetic algorithm

The genetic algorithm (GA) is a family of evolutionary strategies, introduced in 1975 by John Holland. It encodes tentative solutions in chromosomal like structures. Evolution occurs as reproductive opportunities for the fittest. External variation is introduced through mutations.

12.2.1 Outline of a genetic algorithm

A genetic algorithm encompasses the following fundamental steps:

1. Encoding of the chromosomes.
2. Generation of an initial population.
3. Fitness evaluation.
4. Parents selection.
5. Reproduction with crossover.
6. Mutation.
7. New population.

This steps are repeated from the new population to fitness evaluation. There is a need to encode the problems in a specific way and how to perform selection, mutation and reproduction.

12.2.2 Chromosome encoding

Chromosome encoding is performed through bit strings. A long entity θ, or chromosome, is divided into sections, or genes.

12.2.3 Generation of an initial population

Analogously to the starting points of a multi-start procedure, the generation of an initial population can be done in different ways:

- Random.
- Latin hypercube.
- Orthogonal sampling.

12.2.4 Fitness evaluation

The objective function for the current population could be picked from known functions like the sum of squares, the likelihood or more general formulations in the form of:

$$\theta^i \mapsto f^i = f(\theta^i) \qquad i = 1, \ldots, N$$

As long as there is a connection between the fitness number and candidate selection, the function is fine. MCMC, for example, was using one candidate at a time at each iteration, while gradient methods were computing the gradient using information from integration. In this case, the size of the population will determine the number of calls.

12.2.5 Parent selection

The parents can be selected through a threshold based selection (selecting the best k parameters) or through a random based selection. The random based selection, as an example is similar to Gillespie, where fitness is used instead of propensity:

1. From $f^1, \ldots, f^N$ compute $\sum_{i=1}^{N} f_i = f_0$.
2. Generate the random number $j \sim \mathcal{U}(0, 1)$.
3. Select the smallest k such that $\sum_{i=1}^{k} f_i > jf_0$.
4. Clone θ^k in an intermediate population.

Another way to select the parents is through "Roulette selection": the area of a circle is covered by each chromosome fitness proportionally. The idea is to spin the wheel and select the chromosome where it stops.

12.2.6 Reproduction

From the intermediate population two individuals u, v are randomly selected and a gene for cross over t. Parent chromosomes are recombined and new offspring is added to the new population. This procedure is only inheriting information from the previous generation, and mutations are not considered.

12.2.7 Mutation

The offspring may or may not mutate according to a certain probability. Let p be the probability that a gene of the new offspring mutates.

12.2.8 Algorithm

The procedure of a genetic algorithm can be visualized in figure 12.1.

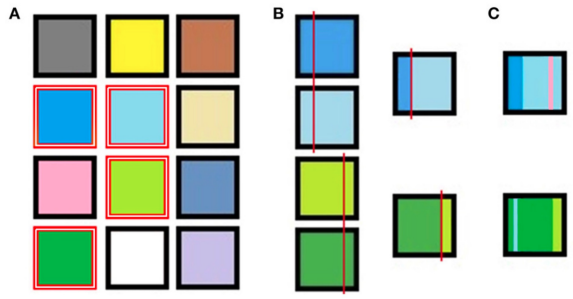

Figure 12.1: GA procedure example

An implementation of GA can be found in algorithm 56.

Algorithm 56: `Genetic algorithm()`

Input: a fitness function c that measures the goodness of the fit, the population size N, the rate of mutation σ and the number of generations G
Output: the best candidate solution $\tilde{p}$ after g generations
map the parameters into strings of lenth l
generate an initial population of strings $P = \{p_1, \dots, p_N\}$
foreach $G = 1, \dots, G$ **do**
    $P' = \emptyset$
    $f_i = c(p_i), i = 1, \dots, N$
    $f_0 = \sum_{i=1}^{N} f_i$
    foreach $N = 1, \dots, N$ **do**
        $j \sim norm(0, 1)$
        determine the smallest k such that $\sum_{i=1}^{j} f_i > j f_0$
        $P' = P' \cup p_k$
    $P = \emptyset$
    foreach $N = 1, \dots, N$ **do**
        $m, n \sim norm(1, N)$
        select $p_m, p_n \in P'$
        $t \sim norm(1, l)$
        $\tilde{p} = \{p_m\{1 : t\}, p_n\{t + 1 : l\}\}$
        for $i = 1, ;l$ **do**
            randomly change $\tilde{p}_i$ with probability σ
        $P = P \cup \tilde{p}$
return the best solution $\tilde{p}$ such that $c(\tilde{p}) = \min_{p \in P} c(p)$

12.2.9 Discussion

Defining the number of generation is the starting point when building this algorithm. Moreover the size of the population is decided through a function. The cost function is as general as possible. Furthermore genetic algorithms are used for hyperparameters tuning. A neural network can be trained inside it. This is not feasible with Markov Chains or gradient methods.

12.2.10 Conclusion

12.2.10.1 Disadvantages

The genetic algorithm is computationally demanding, as for N times the likelihood,cost or fitness is computed just for selecting new parents. Next, a direction is not chosen directly: the reduction of the fitness function is not driven by the dimension of the population. Close to the optimum this algorithm converges slowly.

12.2.10.2 Advantages

In the genetic algorithm each one of the computations can be parallelized. It is very general and obtains good results. f can vary sometimes; the algorithm can use an objective function without the constraints, that can be added as penalties after some generations.