Masters Project

Extending the Stochastic Simulation Software Package StochPy with Stochastic Delays, Cell Growth and Cell Division

Author:
Maxim Moinat
Bioinformatics and Systems Biology
VU University Amsterdam

Supervisors:
Timo Maarleveld, MSc
Anne Schwabe, MSc
Prof. Dr. Frank Bruggeman

Department:
IBIVU Systems Bioinformatics
VU University Amsterdam

Date:
July 1, 2014
Abstract

Reactions at single-cell level and at low copy number are inherently stochastic. This results in single cell heterogeneity, an increasingly important research subject. Conventional deterministic models are unable to capture this stochasticity and therefore stochastic simulation algorithms were developed. Most notably is the Gillespie algorithm, simulating reactions with exponentially distributed reaction times. In previous research, StochPy was developed, a comprehensive and user-friendly stochastic modeling tool in Python. We present the implementation of several new features to StochPy. Firstly, two previously described delayed methods were implemented. Time delays have to be used to model some vital processes in cells, like transcription and translation, which have a delay between initiation and completion. In addition we develop a new stochastic simulation method, the Single Molecule Method (SMM). This method is capable of simulating reactions without non-exponential waiting times. In this research we show which situations are suitable for the SMM and which are for the delayed method. Another source for heterogeneity between cells is stochastic cell growth and division. We developed a cell growth and division module for StochPy, capable of stochastic simulation of a cell lineage. The new features were applied for simulations of a two-component signaling model, revealing behavior that would not have been observed in a deterministic model.
Acknowledgements

I want to thank Timo for his patience in the long process of making the report. Thanks for keep setting new deadlines, until finally one stuck. And thanks Anne for figuring out together how the long equations could be correctly translated to a script.

And of course special thanks to my fellow students, in particular Katy, Jacomien, Remco, Wouter and Rebecca for having a great time working side by side on our projects. And finally, Aafke, for her critically reading of the report.

Lastly, I want to thank dr. Frank Bruggeman for providing me this interesting and challenging internship.
1 Introduction

1.1 Stochastic Modeling

Traditionally, chemical kinetics have been modeled with deterministic ordinary differential equations (ODEs). This is appropriate for chemical systems with high copy numbers, where the species concentration behaves deterministically. With the recent advances in single cell measurements, it has become apparent that multiple important cell processes happen at low copy numbers [1]. These reactions are intrinsically stochastic and cause single cell heterogeneity. Deterministic methods like ODEs are not sufficient to describe single cell heterogeneity [2]. Therefore, stochastic models are necessary to model single cell processes.

Stochastic modeling treats systems with discrete species copy numbers. The (continuous) time to a new reaction is drawn from an exponential distribution. A stochastic simulation is exact in the sense that each run is an independent realization from the true underlying process [1].

Stochastic simulations have the capability to explain observed experimental data of complex biological process at low copy numbers. One of the advantages of discrete copy numbers is that it enables the stochastic simulations to supply uncertainty intervals for a species at every given time point. In addition, discrete copy numbers can go to zero. This in contrast to ODEs. Stochastic simulations can therefore give parameters like a distribution for the time to extinction, a feature of which deterministic methods do not support [3].

Stochpy (Stochastic Python) [4] was developed to perform stochastic simulations in Python. It is a flexible and interactive modeling tool and includes high level stochastic simulation functions. Statistical functions and advanced plotting functions are also provided.

1.2 Research Goal

Many processes in living cells do not occur instantly. Two common examples are transcription and translation. For typical eukaryotic cells, the transcription of an average gene takes about 2750 seconds and the translation of an average protein about 450 seconds [5]. There is a delay between the initiation and completion of the reaction, which should be included in the model (Figure 3).

In addition, real cells are not containers of fixed volume, cells grow and divide. For instance the partitioning of species is not exact, leading to a large cell to cell variability. Especially for species at low copy number. Stochastic modeling of cell growth and division is, therefore, essential to capture this source of cell heterogeneity.

The goal of this project is to extend the existing StochPy software with delayed reactions and a cell division module. In the methods section we will explain the algorithms behind the modules. The results section will show how to use the new modules.

2 Background

2.1 Gillespie Algorithm

The principal method for stochastic simulations is the Gillespie Algorithm, first presented in 1976 [6]. The algorithm numerically simulates the very Markov processes behind the model. The system is described by its state vector $X$, containing the copy numbers of each species, and the propensities $a_k(X)$ for each reaction $k$. A propensity can be described as the probability that a particular reaction will occur. The propensities are calculated from the rate equation (often called the propensity function) and the state vector. The sum of all the propensities is denoted by $a_0(X)$.

The Gillespie algorithm relies on the chemical master equation, which determines the probability that each species will have a specified copy number at a given time [7]. To eliminate spatial effects, it is assumed that the system is well-stirred. Gillespie derives two essential equations from the master equation, one to generate a waiting time $\tau$ until firing of the next reaction and one for the generation of the reaction $\mu$ that will fire. First, two random numbers $r_1$ and $r_2$, on the interval $[0,1]$, are drawn from
the standard uniform distribution, then

\[ \tau = -\frac{\ln(r_1)}{a_0(X)} \]  

(1)

\[ \mu = \text{the smallest integer satisfying } \sum_{i=1}^{\mu} a_i(X) > r_2 \cdot a_0(X) \]  

(2)

This leads to the following stochastic simulation algorithm [7] (Figure 1).

1. Initialization
   
   (a) Set \( t = t_{\text{init}} \).
   
   (b) Set the initial number of molecules in \( X \).
   
   (c) Calculate the propensity function of each reaction.

2. Draw two random numbers \( r_1 \) and \( r_2 \).

3. Calculate \( \tau \) and \( \mu \) according to Equations 1 and 2.

4. Execute reaction \( \mu \) by setting \( t = t + \tau \) and updating \( X \).

5. Evaluate all the propensities.

6. Go to step 2 or stop when maximum number of iterations reached.

---

**Figure 1:** The Gillespie algorithm on a timeline with three sequential reactions. After an exponential waiting time \( \tau_k \) a reaction \( \mu_k \) takes place.

### 2.2 Next Reaction Method

Recently, different versions and improvements of the Gillespie algorithm have been presented to improve the simulation performance. Two were developed by Gillespie, the first-reaction and first-family method [7]. In this section we give a small introduction to the Next Reaction Method (NRM), developed by Gibson and Bruck [8]. This method is presented as being more efficient, especially improving the speed for simulations of large systems. It will serve as a foundation of new algorithms presented in this report.

There are three major differences as compared to the original Gillespie algorithm. First of all, waiting times are absolute and not relative (time at which reaction occurs, not time until reaction). This time is generated and saved for each reaction separately. The reaction with the lowest waiting time is executed. The second major difference is the use of a dependency graph. In short, this graph indicates which propensities are affected by the firing of a particular reaction. This reduces the need to recalculate all the propensity functions. Lastly, after each step just one random number is drawn, instead of two. This number is used to recalculate the waiting time for the reaction that has fired. The waiting times for reaction that have not fired are reused. All these differences together can increase the speed of the stochastic simulation, without making an approximation.

### 2.3 Cell growth and division

Cell growth and division are stochastic processes that can have a large influence on the copy numbers in individual cells [9]. Therefore cell growth is another source of intrinsic noise in living cells. The first origin of stochasticity is a heterogeneity of cell volume. The three main reasons for this heterogeneity are [10]: 1) a heterogeneity in the mother cell volume at division, 2) an imprecise volume division from
mother to daughter cells and 3) a heterogeneity of growth rates (Figure 2). Secondly, the division of molecules between the two daughter cells is imprecise. This latter effect is especially dominant at low copy numbers.

Figure 2: Three sources of cell growth and division heterogeneity. The three sources are the volume at which a cell divides, the volume of the two daughter cells and the volume growth rate are all variable.
3 Methods

We implemented four different methods of modeling delayed reactions were implemented in StochPy (version 1.2.0). Two were previously described: the Delayed Direct Method by Cai [5] and the Delayed Next Reaction Method by Anderson [11]. These methods extend the Gillespie stochastic simulation algorithm (SSA) [7] to a system with delays. A delayed reaction consists of an exponential waiting time as initiation step with a subsequent delay time (Figure 3).

Two other methods were newly developed for StochPy: the Single Molecule Method (SMM) and the fast Single Molecule Method (fSMM). Both methods consider delayed reactions as reactions with a non-exponential waiting time. In contrast to the other two delayed methods, the reaction is not split up into initiation and completion, it consists of only a non-exponential waiting time. Although these methods do not model delays explicitly, they can be used instead of the delayed methods. The differences will be described in more detail in the results section.

3.1 Consuming and nonconsuming delayed reactions

To avoid confusion, we define two types of delayed reactions. Both Anderson and Cai differentiate two types of delayed reactions, but name them differently. Cai names them ‘consuming’ and ‘nonconsuming’, Anderson names them ‘initiation completion delayed’ (ICD) and ‘completion delayed’ (CD). The difference between the two types of delayed reactions are the events that happen at initiation and at completion of the delay. From the names it is not immediately apparent what is meant by the definitions. Below are four quotes that describe the types of delay for Cai and Anderson, respectively:

Cai:
Consuming: "... the reactants of an unfinished reaction cannot participate in a new reaction.”
Nonconsuming: "the reactants of an unfinished reaction can participate in a new reaction ...’ This implies, although not explicit, that the reactant is consumed at completion.”

Anderson:
ICD: "... the system is updated by losing the reactant species at the time of initiation (...) and is updated by gaining the product species at the time of completion.”
CD: "... the system is updated only at the time of completion (...) by losing the reactant species and gaining the product species.”

Figure 3: Timeline of a delayed reaction. The reaction is split up in a initiation and a completion event, which are separated by a delay time. At initiation reactants can be consumed at completion both consumption and production can take place. In other words, the reaction starts at initiation and ends at completion.

The definitions of Cai and Anderson are equivalent, ICD is the same as consuming and CD the same as nonconsuming. In the remainder of this report, we will use consuming and nonconsuming to distinguish between the two types of delayed reactions (Table 1). The difference between consuming and nonconsuming is not negligible and can affect the simulation result considerably. This is further illustrated by the simulations in Figure 4.
Table 1: Comparison of the events at initiation and completion of a reaction without delay, consuming delay and nonconsuming delay. A consuming reaction consumes its reactants at initiation of a delay, where a nonconsuming reaction consumes them at the completion of the delay. A reaction without delay has initiation and completion and the same time ($delay = 0$).

<table>
<thead>
<tr>
<th>Reaction type</th>
<th>Change at Initiation ($\Delta t = \tau$)</th>
<th>Change at Completion ($\Delta t = \tau + delay$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without delay</td>
<td>−reactants &amp; +products</td>
<td></td>
</tr>
<tr>
<td>Consuming delay</td>
<td>−reactants</td>
<td>+products</td>
</tr>
<tr>
<td>Nonconsuming delay</td>
<td></td>
<td>+products &amp; −reactants</td>
</tr>
</tbody>
</table>
3.2 Delayed Direct Method

The delayed direct method, developed by Cai, is one of the first efficient exact SSA supporting delays [5]. The method is directly based on the principles of the direct SSA by Gillespie. Other methods for a delayed SSA have been developed before, but are not exact or not as efficient as the one presented by Cai. One example is the rejection method by Barrio et al. [12], which needs up to 50% more random numbers to be generated due to an algorithm based on trial and error. In the next sections we will describe the delayed SSA as presented by Cai.

Figure 4: Difference between no delay (A,B), consuming delay (C,D) and nonconsuming delay (E,F). The simulation for one trajectory (A,C,E) and the average of 1000 trajectories (B,D,F) is shown. The consuming delay simulation has immediate consumption of X, but a delayed production of Y. The shape of the production of X stays the same (compare B and D), because in a consuming delay it is consumed immediately. With a nonconsuming delay, the consumption of X is also delayed. The consumption happens simultaneous with the production of Y. The total shape of the curve of the nonconsuming delayed simulation changes, because all the reactions are initiated with a X copy number of 20. This causes the reactions to be faster. In all simulations $k_1 = 0.5$ and the delay is fixed at 2 seconds.
3.2.1 Waiting time generation

The generation of the waiting times is, besides the delays itself, one of the largest additions of the delayed method as compared to the Gillespie Direct Method. The waiting times are generated with a different strategy, because amounts (and thus propensities of reactions) are subject to change when an ongoing delayed reaction ends. Cai therefore presents a new way of calculating the waiting times, which considers all ongoing delays.

The state of a system with delayed reactions is not only described by the state vector $X$ and propensity functions $a_i$, but also by the reactions that are pending to be completed. The relative times at which pending delayed reactions complete are stored in the vector $T_i$. This vector is initialized with the first element zero, $T_0 = 0$ and the last element infinite, $T_{-1} = \infty$. These first and last elements serve as breaking points. We assume that $T_i$ is ordered, such that $T_1 \leq T_2 \leq T_3 \leq \ldots$. The first delayed reaction to complete is at $t + T_1$, where $t$ is the current time. A possible state of the vector $T_i$ is shown in Figure 5.

![Timeline of delayed reactions and corresponding changes in the sum of propensities](image)

Figure 5: Timeline of delayed reactions and corresponding changes in the sum of propensities ($a_0(t)$) at a particular time point. Five delayed reactions will complete at $T_1$ to $T_5$ and at the completion of each delayed reaction the $a_0$ changes. Upon completion of a reaction, $a_0$ will almost always increase. This is because at completion a product is often formed, increasing the propensities and thus also the sum $a_0$.

Consider the situation in Figure 5, and let $t$ be the current time and $a_0(t)$ as the sum of propensities at $t$. At this point two events can happen next: 1) a new reaction initiates at $t + \tau$ or 2) the first delayed reaction completes at $t + T_1$. The first event happens when the exponential waiting time is smaller than $T_1$, the second will happen otherwise. When the delayed reaction completes, the times change $t = t + T_i$ and $T_i = T_{i-1} - T_1$ for $i \in \{2, 3, 4, 5\}$. This process is repeated until a new reaction initiates, instead of the completion of a pending delayed reaction. At every completion $a_0(t)$ changes due to the formation of species. This also influences the exponential waiting times to be drawn, because this time depends on $a_0(t)$ (Equation 1).

To cope with the changing propensities, Cai defines the cumulative distribution function of $\tau$, $F_\tau(\tau)$ (Equation S3). The function can be used to calculate the probability that the waiting time is smaller than $\tau$. $F_\tau$ can be used to generate $\tau$ from a standard uniform random variable $u_2$. The first step is to find $i$ such that $F_\tau(T_i) \leq u_2 < F_\tau(T_{i+1})$. Thus $\tau \in [T_i, T_{i+1})$, which means that the reaction initiates between delayed event $T_i$ and $T_{i+1}$. Then $\tau$ can be directly calculated from
\[
\tau = T_i + \frac{-\ln(1-u_2) - \sum_{j=0}^{i-1} a_0(t+T_j) \cdot (T_{j+1} - T_j)}{a_0(t + T_i)} \quad \text{with } \tau \in [T_i, T_{i+1})
\]

Equation 3 sums the propensity after each completion of a delayed reaction (after a time \(T_i\)). Therefore the change of propensity is taken into account when generating \(\tau\). Cai also provides a pseudocode to generate \(\tau\). This pseudocode was found to be incorrect, on which we elaborate in the results section. The improved pseudocode (Pseudocode 1) works by first finding \(i\) such that \(\tau\) lies in between \(T_i\) and \(T_{i+1}\) and follows the steps shown in Figure 6. Then it calculates \(\tau\) from the integrated \(a_t\). The original incorrect pseudocode can be found in the SI (pseudocode S2).

**Data:** Propensity functions, state vector, completion times \(T\) with \(T_0 = 0\) and \(T_{N_{\text{delayed}}+1} = \infty\)

**Result:** Generates the event waiting time \(\tau\) according to equation 11 [Cai, 2007]

Algorithm 1 describes the delayed direct SSA. In this algorithm, \(M\) is the number of reactions, \(a_k\) are the propensities and \(X\) is the state vector, containing the copy number of all species. The structure \(T_{\text{struct}}\) is the vector \(T\) of waiting times of pending delayed reactions, and also contains the reaction index.

\begin{verbatim}
Pseudocode 1: Improved Pseudocode for generating \(\tau\) in a system with delays.

\begin{algorithm}
  \begin{algorithmic}
    \State \textbf{Data:} Propensity functions, state vector, completion times \(T\) with \(T_0 = 0\) and \(T_{N_{\text{delayed}}+1} = \infty\)
    \State \textbf{Result:} Generates the event waiting time \(\tau\) according to equation 11 [Cai, 2007]
    \Function{GenerateEvent}{\(\tau\)}
      \State Generate a realization of the standard uniform random variable, \(u_2\)
      \If{no ongoing delayed reaction}
        \State \(\tau = -\ln(u_2)/a_0(t)\)
      \Else
        \State \(i = 0\)
        \State \(a_{\text{prev}} = 0\)
        \State \(a_t = a_0(t) \cdot T_i\)
        \State \(F = 1 - \exp(-a_t)\)
        \While{\(F < u_2\)}
          \State \(i = i + 1\)
          \State \(a_{\text{prev}} = a_t\)
          \State Update the species vector due completion of the delayed reaction at \(t + T_i\)
          \State Calculate propensity functions with new species vector and calculate \(a_0(t + T_i)\)
          \State \(a_t = a_t + a_0(t + T_i) \cdot (T_{i+1} - T_i)\)
          \State \(F = 1 - \exp(-a_t)\)
          \State \(\tau = T_i - \ln(1-u_2) + a_{\text{prev}}/a_0(t + T_i)\)
        \EndWhile
      \EndIf
    \EndFunction
  \end{algorithmic}
\end{algorithm}
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.png}
\caption{Timeline with depiction of the workflow of Pseudocode 1. Here \(a_t\) corresponds to the summation of \(a_0\) in the sum of Equation 3, the current absolute time \(t = 0\) and \(F = 1 - \exp(-a_t)\). The steps in the while loop are indicated with red arrows. At every step an element is added to \(a_t\). At \(T_4\), \(F > u_2\), and the while loop is exited. Hence, \(\tau\) will be between \(T_3\) and \(T_4\) (in terms of Equation 3: \(i = 3\)). The last \(a_t\) term is incorrectly added and has to be removed (blue arrow), or let \(a_t = a_{\text{prev}}\) as in Pseudocode 1. Next, \(\ln(1-u_2)\) should be added to \(a_t\) and \(\tau\) can be calculated from \(\tau = T_3 - \frac{a_t}{a_0(T_3)}\).}
\end{figure}
of the delayed reaction. In this way the algorithm can remember which reaction has to be completed. This makes every element of Tstruct a set of delay waiting time and reaction index. Pseudocode 1 is used to calculate $\tau$, the time until the initiation of the next reaction. This can be a reaction with or without a delay. In the process of generating $\tau$, multiple pending delayed reactions can be completed.

**Algorithm 1. Delayed Direct Method [5]**

1. **Initialization**
   (a) Set $t = t_{\text{init}}$.
   (b) Set the initial number of molecules.
   (c) Calculate the propensity function of each reaction.
   (d) Set $T_{\text{struct}} = [(0, \text{nan}), (\text{inf}, \text{nan})]$.

2. **Generate $\tau$ from pseudocode 1.**
   (a) If delayed pending reactions have been completed, remove these from Tstruct.
   (b) Update Tstruct by subtracting the $\tau$ from every time in Tstruct.

3. **Generate $\mu$ from a standard uniform random variable $u_1$ by taking $\mu$ to be the integer for which**
   $\sum_{j=1}^{\mu-1} a_j(t + \tau) < u_1 a_0(t + \tau) \leq \sum_{j=1}^{\mu} a_j(t + \tau)$.
   (a) If reaction $\mu$ is a nondelayed reaction, update the state vector $X$ with the consumption of reactants and production of products.
   (b) If reaction $\mu$ is a consuming delayed reaction, update the state vector $X$ with the consumption of reactants.
   (c) If reaction $\mu$ is a nonconsuming delayed reaction, do not update the state vector.
   (d) In the last two cases, draw the delay time from the given distribution and add this time to the Tstruct. Sort Tstruct.

4. **Update the propensities.**

5. **Set** $t = t + \tau$.

6. **If** $t > t_{\text{end}}$ or steps $>$ endsteps, quit. Else go to step 2.

### 3.3 Delayed Next Reaction Method

A different approach for a delayed stochastic simulation algorithm has been developed by Anderson [11]. This method is based on a modified NRM, which is also introduced by Anderson. In this modification, the reaction waiting times are explicitly represented by the firing times of independent, unit rate Poisson processes $Y_k$. The method is extended to a system with delays. He reports that his delayed method is substantially faster than the delayed direct method by Cai and the rejection method by Barrio et al.

Algorithm 2 describes the Delayed Next Reaction Method by Anderson. Here, $X(t)$ is the state vector at time $t$, $M$ the number of reactions, $a_k(t)$ the propensities and $d_k$ is the delay of each reaction $k$. This delay time can be drawn from a given distribution. In this method, the reaction times are explicitly represented as the firing times of independent, unit rate Poisson processes with internal times given by integrated propensity functions [11]. A Poisson process gives the number of events in a given time interval, from which the next time until firing of the reaction can be calculated. The independent unit rate Poisson processes of reaction $k$ are represented by $Y_k$. Each reaction also has an internal time, $T_k$, which is defined by

$$T_k(t) = \int_0^t a_k(X(s))ds$$

This can be seen as the sum of the propensity $a_k(t)$ in the time, weighted by the time. $T_k$ describes how many times the reaction $k$ is expected to have fired up to time $t$. $a_k$ is discrete during a stochastic
simulation and each value is attained until the next reaction fires, after a waiting time $\tau$. The next firing time of $Y_k$ is given by $P_k = \min \{ t > T_k : Y_k(t) > Y(T_k) \}$, where $t$ is the current time. In practice, $P_k$ is calculated by

$$P_k = \sum_{0}^{n} \ln \left( \frac{1}{u_k} \right)$$

where $n$ is the number of reactions taken place and $u_k$ a random uniformly distributed number. The time until the next firing of reaction $k$ can then be calculated from

$$\Delta t_k = \frac{P_k - T_k}{a_k}$$

This has as advantage that the system can be simulated without conversion to a relative timescale. This theory forms the basis of Algorithm 2.

**Algorithm 2. Delayed Next Reaction Method [11]**

1. Initialization:
   (a) Set $t = t_{init}$.
   (b) Set the initial number of molecules.
   (c) For each $k \leq M$ set $P_k = 0$ $T_k = 0$.
   (d) Set $T_{struct} = [(0, \text{nan}), (\text{inf}, \text{nan})]$.
   (e) Calculate the propensity for each reaction.
   (f) Generate for each reaction a uniform random number $u_k$ and set $P_k = \ln \left( \frac{1}{u_k} \right)$.

2. For each $k$, set $\Delta t_k = \frac{P_k - T_k}{a_k}$.

3. Set $\tau = \min (\Delta t_k, T_{struct} - t)$.

4. Set $t = t + \tau$.

5. If we chose the completion of the delayed reaction $\mu$ ($T_{struct} - t < \Delta t_k$):
   (a) Update the system according to the completion of the reaction $\mu$.
   (b) Delete the second element of $T_{struct}$.

6. Elif reaction $\mu$ initiated and $\mu$ is not a delayed reaction:
   (a) Update the system according to reaction $\mu$.

7. Elif reaction $\mu$ initiated and $\mu$ is a consuming delayed reaction:
   (a) Update the system according to the initiation of reaction $\mu$.
   (b) Update $T_{struct}$ by inserting $t + d_\mu$ into $T_{struct}$ and resort $T_{struct}$.

8. Elif reaction $\mu$ initiated and $\mu$ is a nonconsuming delayed reaction:
   (a) Update $T_{struct}$ by inserting $t + d_\mu$ into $T_{struct}$ and resort $T_{struct}$.

9. For each $k$, set $T_k = T_k + a_k \cdot \tau$.

10. If reaction $\mu$ initiated, let $u$ be a uniform random number and set $P_\mu = P_\mu + \ln \left( \frac{1}{u_k} \right)$.

11. Recalculate each propensity function.

12. If $t > t_{end}$ or steps > endsteps, quit. Else go to step 2.
3.4 Single Molecule Method

Experimentally determined waiting times are not always exponentially distributed, whereas stochastic simulations do always assume exponential waiting times. The delayed methods can use non-exponential delays, but will always consist of an exponential initiation reaction time and a subsequent delay (Figure 3). We, however, developed a method we name the Single Molecule Method (SMM), which approaches reactions differently. Both the exponential step and a potential delay are replaced by one waiting time for the whole reaction (Figure 7). More importantly, the reaction time does not have to be drawn from an exponential distribution, but can be drawn from any distribution. Hence, we can give a particular reaction an experimentally determined reaction time distribution.

In this section we will explain the newly developed SMM. The implementation in StochPy will be explained in the results section. For the SMM we distinguish between waiting times and putative reaction times. A waiting time of a reaction is as described the time until that particular reaction fires. A putative reaction time is the time until a single molecule or combination of molecules will react in a particular reaction channel. A waiting time is always exponentially distributed and depends on the copy number of the reaction, where a putative reaction time can also be non exponentially distributed and does not depend on the copy number (it has fixed parameters).

Figure 7: Timeline of two Single Molecule Method reactions. First molecule X<sub>1</sub> reacts, followed by molecule X<sub>2</sub>. It is possible that both react in different reactions, depending on the model. The putative reaction time is specific for a single molecule of X and can, in contrast to the traditional Gillespie Algorithm, be non exponentially distributed.

3.4.1 Single Molecule Method

As noted already, the SMM approaches reaction waiting times differently than the methods derived from the Gillespie algorithm. Every single molecule that can react has a random putative reaction time, the time it takes for that molecule until it reacts. This putative reaction time can be from a distribution of choice, but should not allow negative values. At every step, the molecule with the smallest putative reaction time will react. If a molecule is a reactant in multiple reactions, it has a putative reaction time for each of these reactions. Again, the smallest putative reaction time will eventually be executed and the molecule will react in that reaction channel. The reaction times of this molecule will be deleted for both reactions. The SMM has similarities with the Next Reaction Method of Gibson and Bruck [8], with the difference that the SMM works with single molecules instead of single reactions.

For second order reactions, in which the reaction is dependent on two species, every combination of molecules should be considered. A simple example will illustrate this. Consider the reaction A + B → C, with two molecules of A and ten molecules of B. There are in total twenty different reactions possible; both molecules of A can react with ten different molecules of species B (Table 2). The reaction with the smallest putative reaction time will take place.
Table 2: Putative reaction times of a second order SMM reaction, \(A + B > C\). Two system consists of two molecules of species A and ten of B. A total of twenty randomly distributed putative reaction times are generated, one for each combination of single molecules. Any random distribution can be used. The next reaction will be the reaction with the smallest time, in this case \(A_1 \text{ with } B_5\). After \(A_1\) and \(B_5\) have reacted, the first row and fifth column will be deleted, because \(A_1\) and \(B_5\) cannot react anymore. Next this process is repeated again for the remaining molecules until the molecules are exhausted.

<table>
<thead>
<tr>
<th>(B_1)</th>
<th>(B_2)</th>
<th>(B_3)</th>
<th>(B_4)</th>
<th>(B_5)</th>
<th>(B_6)</th>
<th>(B_7)</th>
<th>(B_8)</th>
<th>(B_9)</th>
<th>(B_{10})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A_1)</td>
<td>0.15</td>
<td>1.22</td>
<td>0.66</td>
<td>0.42</td>
<td>0.13</td>
<td>0.48</td>
<td>0.45</td>
<td>0.96</td>
<td>0.32</td>
</tr>
<tr>
<td>(A_2)</td>
<td>1.00</td>
<td>0.75</td>
<td>0.96</td>
<td>0.69</td>
<td>0.45</td>
<td>0.37</td>
<td>0.77</td>
<td>0.64</td>
<td>1.09</td>
</tr>
</tbody>
</table>

Zero-order reactions form a special case. A zero order reaction has no reactant, so the putative reaction time cannot be associated with a molecule. We assume that the putative reaction time distribution of a zero order reaction is the time between subsequent firings of this reaction. Upon firing, a new putative reaction time is drawn from the distribution. For example if the putative reaction time is fixed at 5 seconds, the reaction will fire every five seconds.

In contrast to the exponential reaction times in traditional stochastic simulation algorithms, the drawn putative reaction times do not depend on the copy number. The time until a reaction fires does depend on the copy numbers, due to more drawn putative reaction times. More copy numbers gives more putative reaction times and a higher probability for a low minimum putative reaction time. So a higher copy number leads, in general, to a faster next firing of the reaction.

3.4.2 Algorithm Single Molecule Method

Every reaction \(k\) has a structure, \(Mstruct_k\), which contains all putative reaction times for every molecule, drawn from the putative reaction time distribution of reaction \(k\). If \(k\) is a zero-order reaction, \(Mstruct_k\) contains one putative reaction time and if reaction \(k\) is first-order it contains one row of putative reaction times. If \(k\) is a second-order reaction, \(Mstruct_k\) is a 2D matrix (Table 2). The matrix is shaped by the copy numbers of both reactants. Assume reaction \(k\) has two reactant species, \(X\) and \(Y\), with copy numbers \(m\) and \(n\) respectively. Then \(Mstruct_k\) will be a matrix with \(m\) rows and \(n\) columns. Every element is a unique combination of a molecule of each species. If a particular species is a reactant in multiple reactions, it has same the place in every matrix. This is important for deleting the correct molecule if the molecule reacts in one of the reactions.

A special second-order reaction is homodimerization, e.g. \(A + A > C\). \(Mstruct_k\) is constructed the same way as for all second-order reactions, but the putative reaction times on diagonal are set to infinite and will never take place (Table 3). A molecule of \(A\) cannot dimerize with itself, because it needs another molecule of \(A\). If the copy number of species \(A\) is one, the matrix will only contain an infinite. Please note that there are two putative reaction times for each combination, e.g. both the combination \(A_1, A_2\) and the combination \(A_2, A_1\) exist. We could have chosen to only leave one combination in the matrix, but we have not done this to allow the same behavior as ‘traditional’ simulations with identical settings. A traditional rate equation for the homodimerization is \(k_1 \cdot A \cdot (A - 1)\). So the number of putative reaction times needed in the SMM is \(A \cdot (A - 1)\), which is equal to the number of elements in a \(A \times A\) square matrix without the diagonal.

Table 3: Putative reaction times of a SMM second-order homodimerization reaction, \(A + A > C\). The diagonal is set to infinite, because a molecule can never react with itself (e.g. \(A_2\) with \(A_2\)).

<table>
<thead>
<tr>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A_1)</td>
<td>(\infty)</td>
<td>0.66</td>
</tr>
<tr>
<td>(A_2)</td>
<td>1.00</td>
<td>(\infty)</td>
</tr>
<tr>
<td>(A_2)</td>
<td>0.69</td>
<td>0.45</td>
</tr>
</tbody>
</table>

The reaction times in \(Mstruct_k\) are the absolute times of firing. After reaction \(k\) fired, the putative reaction time of the molecule is deleted from \(Mstruct_k\). The possible putative reaction times of this molecule in other reactions are also deleted.

1. **Initialize:**
   
   (a) Set \( t = t_{\text{init}} \).
   
   (b) Set the initial number of molecules.
   
   (c) Initialize \( M_{\text{struct}} \) for each reaction \( k \).

2. Find the smallest putative reaction time \( \tau \) in \( M_{\text{struct}} \). Let \( \mu \) be the place where \( \tau \) is found in \( M_{\text{struct}} \).

3. Set \( t = \tau \)

4. Change the number of molecules according to execution of reaction \( \mu \).

5. For each molecule reacted in reaction \( \mu \) and every reaction \( k \) the molecule is reactant in:
   
   Remove the molecule from the matrices \( M_{\text{struct}} \). If first-order, remove one element. If second-order, remove the entire row or column.

6. For each product \( P \) of reaction \( \mu \):
   
   For each reaction \( i \) where \( P \) is a reactant:
   
   (a) If first-order, generate a random \( r \) from the distribution of \( i \). Add \( r + t \) to \( M_{\text{struct}} \).
   
   (b) If second-order, generate a row of random \( R_{\tau} \) for every row (or column) in \( M_{\text{struct}} \). Add \( R_{\tau} + t \) to \( M_{\text{struct}} \).

7. If \( \mu \) is a zero-order reaction, generate a random \( r \) from the distribution of \( \mu \). Set \( M_{\text{struct}} = r + t \).

8. If \( t > t_{\text{end}} \) or steps > endsteps, quit. Else go to step 2.

Third-order SMM reactions are not described in this report. In theory a third-order reaction can be implemented in the same way as second-order reactions, but with a three dimensional matrix instead of two dimensional. This, however, makes the SMM algorithm computationally expensive. An addition of a single molecule to the system leads to the generation of a matrix of random putative reaction times, for every reaction the molecule is in. In addition, finding the minimum putative reaction time will take more time. Because most biological events can be simulated with second and lower order reactions, we chose not to support third-order reactions.

3.4.3 fast Single Molecule Method

The Single Molecule Method described above has a large cost: the simulations are slow, especially for larger models. This is due to the large \( M_{\text{struct}} \) structure, which contains a matrix for each reaction. Finding the smallest \( \tau \) in such a large structure is a slow procedure. This is in particular true for systems with many reactions and multiple second-order reactions. This causes the number of putative reaction times in \( M_{\text{struct}} \) to explode. One molecule can have a waiting time for multiple reactions and if a reaction is second-order, it will also have multiple putative reaction times for this reaction. Even at relatively low copy numbers this has a noticeable affect the performance.

To improve the SMM performance, we developed the fast Single Molecule Method (fSMM). This method combines the Next Reaction Method and the SMM. The main advantage is that it does not simulate all reactions as single molecule reactions (SMRs). Only reactions which are given a particular putative reaction time distribution, the SMRs, are treated as described in the SMM algorithm. Often, the majority of reactions in a model are exponential reactions and can be simulated with a traditional stochastic simulation algorithm. This has as advantage that the \( M_{\text{struct}} \) contains a smaller number of SMRs. Thus the number of waiting times stored in \( M_{\text{struct}} \) will be significantly decreased, resulting in a faster search of the smallest \( \tau \).

In this description of the fSMM, we assume zero- and first-order SMRs. Second and higher order reactions can be modeled in the traditional way, with propensities. By not allowing second-order SMRs, he performance of the fSMM will be better. Second-order SMRs are more computationally expensive due to the creation of a matrix of putative reaction times. However, this description can be used as blueprint to construct a fSMM with second and higher order SMRs.
For an additional performance boost, the fSMM also employs dependency graphs and an indexed priority queue. Both are previously described and applied in the Next Reaction Method [8]. A dependency graph allows for a speed increase in updating the system. The fSMM makes uses of three dependency graphs, $G$, $H$ and $I$, each devoted to a specific type of system update (Table 4).

The dependency graph $G$ is the same dependency graph as used in the Next Reaction Method. The graph $G_k$ contains for each reaction $k$, the indices of the reactions which propensities are affect if reaction $k$ fires. SMR do not depend on a propensity function, and their indices are therefore not included in this dependency graph. A SMR can influence a propensity of a normal reaction. So if reaction $k$ is a single molecule reaction, then $G_k$ can contain indices of (non single molecule) reactions. For example, reaction 3 in Table 4 changes the copy number of species $E$ ($D$ is both reactant and product and therefore does not change), which both affects reactions 4 and 5. But because reaction 4 is a SMR, only reaction index 5 is present in $G_3$.

The other two dependency graphs, $H$ and $I$, are dedicated to the single molecule reactions. Dependency graph $H_k$ contains for every reaction $k$ the single molecule reactions that are affected by the loss of reactants due to firing of reaction $k$. A waiting time in $Mstruct_k$ is then deleted from all affected single molecule reactions $k$. In this way the reaction ‘feels’ the loss of a reactant molecule. The dependency graph is retrieved from $Reactants(k) \cap Reactants(l)$, where reactions $l$ are only SMRs and $k \neq l$. For example reaction 5 in table has two reactants, $E$ and $F$. If this reaction fires, the reaction 4 (which is a SMR) loses a molecule of reactant $E$. Thus reaction index 4 is present in $G_5$. Note that due to the given definition, the reaction index of the reaction itself is never present. If a SMR fires, it always loses reactants and this is handled separately in the algorithm (Algorithm 4).

Dependency graph $I$ is used for the creation of reactants of SMRs. If a reaction produces a molecule that is reactant in a SMR, new putative reaction times would be generated. It provides for every reaction $k$ the SMRs that are affected by the formation of products due to firing of reaction $k$. A putative reaction time is then added to $Mstruct_k$ of all affected single molecule reactions $k$. Dependency graph $I$ is constructed by taking $Products(k) \cap Reactants(l)$, where reactions $l$ are only SMRs. Note that if the intersection produces multiple matches, the reaction index will occur twice (or more) in the dependency graph for that reaction. We again take reaction 5 in Table 4 as example. The reaction has two molecules of species $D$ as products. $D$ is the reactant in the SMR and $I_5$ contains two times index 3. This results in two additions of putative reaction times in $Mstruct_5$. In contrast to $H$, the own reaction index can be present in $I$ (see reaction 3 in Table 4).

The indexed priority queue, $P_i$, is directly adapted from the Next Reaction Method. Every reaction $i$ has one next reaction time in this queue. The next reaction time for a single molecule reaction $i$, is the minimum of $Mstruct_i$.

3.4.4 Algorithm fast Single Molecule Method

Algorithm 4. fast Single Molecule Method (fSMM):

1. Initialize:
   
   (a) Set $t = t_{init}$.
   (b) Set the initial number of molecules.
(c) Generate the dependency graph $G$, $H$ and $I$.

(d) For each reaction $k$: 
   
   i. If $k$ is a single molecule reaction:
      
      A. If $k$ is a zero-order reaction, generate $r$ from the distribution of $k$. Let $\tau_k = r + t$.
      
      B. Else if $k$ is a first-order reaction, generate $r$ for every copy number of the reactant. Calculate $\tau = r + t$ and put all $\tau$ in the Mstruct$_k$. Let $\tau_k$ be the smallest number in Mstruct$_k$, and remove it from Mstruct$_k$.
   
   ii. Else, calculate the propensity function $a_k$ and generate a waiting time $\tau_k$ according to an exponential distribution with parameter $a_k$.

(e) Initialize the indexed priority queue $P_k$ with the values $\tau_k$.

2. Let $\mu$ be the reaction whose waiting time $\tau_\mu$, stored in $P$, is least.

3. Set $t = \tau_\mu$.

4. Change the number of molecules according to execution of reaction $\mu$.

5. If $\mu$ is a single molecule reaction, replace the old value $\tau_\mu$ in $P$ with the smallest value in Mstruct$_\mu$ (and delete the value from Mstruct$_\mu$). If $\mu$ is also a zero-order reaction, draw a random number $r$ from the distribution of reaction $\mu$. Replace the old $\tau_\mu$ value in $P$ with $\tau_\mu = r + t$.

6. For each $\alpha$ in $G_\mu$:
   
   (a) Update $a_\alpha$.
   
   (b) If $\alpha \neq \mu$, set $\tau_\alpha = \frac{a_{\alpha,old}}{a_{\alpha,new}} \cdot (\tau_\alpha - t) + t$.
   
   (c) If $\alpha = \mu$, generate a random number, $r$, according to an exponential distribution with parameter $a_\mu$ and set $\tau_\alpha = r + t$.
   
   (d) Replace the old $\tau_\alpha$ value in $P$ with the new value.

7. For each $\beta$ in $H_\mu$:
   
   (a) Randomly delete one value from Mstruct$_\beta$ or $P_\beta$.
   
   (b) If the $P_\beta$ value is deleted, replace it by the smallest value in Mstruct$_\beta$.

8. For each $\gamma$ in $I_\mu$:
   
   (a) Generate a random number $r$ from the distribution of reaction $\gamma$ and set $\tau = r + t$.
   
   (b) If $\tau$ is smaller than the waiting time in the indexed priority queue, replace the old $\tau_\gamma$ in $P$ by $\tau$ and append the old $\tau_\gamma$ to Mstruct$_\gamma$.
   
   (c) Else, append $\tau$ to Mstruct$_\gamma$.

9. If $t > t_{end}$ or steps > endsteps, quit. Else go to step 2.
3.5 Cell Division Module

This section explains how cell growth and division are implemented in StochPy. The sources of stochasticity included are: heterogeneity of mother cell volumes, imprecise volume division and imprecise species division. The heterogeneity of growth rates is not included. The whole cell growth and division process is shown in Figure 8 and explained below.

Two distributions are at the heart of the simulation: the distribution of mother volumes, \( \phi(V_m) \), and the distribution of the volume partitioning, \( K(p) \). Both \( \phi \) and \( K \) are denote probability density functions (PDF) from which the mother volume \( (V_m) \) and partitioning ratio \( (p) \) are drawn, respectively. The cell volume grows until \( V_m \) is reached. At that point it will divide into two daughter cells. The partitioning ratio determines how the volume is distributed over the two daughter volumes \( (V_d) \). One daughter will receive a volume of \( V_{d1} = p \cdot V_m \), where the other will receive the other \( V_{d2} = (1 - p) \cdot V_m \). \( K \) should be symmetrically distributed around a mean of 0.5, otherwise a bias for one daughter is created.

Volume growth is modeled as either exponential or linear growth. The exponential growth law is \( V(t) = V_d \cdot e^{g \cdot t_{gen}} \) and linear growth law \( V(t) = V_d + g \cdot t_{gen} \), where \( V_d \) is the daughter volume, \( g \) the growth rate and \( t_{gen} \) the generation time or age \( (t_{gen} = t_{division} - t) \). The time for a certain daughter cell to reach division is called the inter division time (IDT). The IDT can be calculated from the growth law, given the daughter volume \( V_d \) and next mother volume \( V_m \). For exponential and linear growth laws the equations are respectively: \( t = \frac{\ln(V_d/V_0)}{g} \) and \( t = \frac{V_m-V_d}{g} \).

The division of molecules between the two daughter cells is also a stochastic process and depends on the daughter cell volumes. A daughter with a larger volume has a tendency to inherit more molecules than the other daughter. The probability that a specific molecule is inherited by daughter \( d1 \) is modeled as \( \frac{V_{d1}}{V_m} \). The number of molecules, with copy number \( n \), inherited by daughter \( d1 \) can thus be drawn as a random sample from a binomial distribution with \( n \) number of trials and success probability \( \frac{V_{d1}}{V_m} \). The process is repeated for each species. When a delayed method is used, the pending delayed reactions are also divided between the cells. This division is identical to the division of normal molecules.

It should be noted that some species should not be binomially divided between the daughter cells. Molecules like DNA is partitioned actively by the cell. Each daughter cell gets one copy of the chromosome in a normal cell division event. These species divide exact and should be treated separately.

![Diagram of cell growth and division](image)

Figure 8: The cell growth and division module workflow as incorporated in StochPy. Blue triangles and red squares represent molecules which are stochastically partitioned at cell division between both daughters. In this example, the largest daughter cell—which also has a larger chance to be selected—is selected for the next generation. The stochastic simulation is interrupted at each cell division and stops when (1) the number of generations is reached, (2) the desired end time is reached, (3) the desired number of time steps is reached, or (4) all reactions are exhausted.
3.5.1 Volume dependency

Diffusion limited reactions have a high dependency on the volume of the cell. When the cell grows, the reacting species need more time to find each other and thus the reaction waiting time increases. This creates a volume dependency of second and higher order reactions. The reaction rate and thus the propensity $a(t)$ of these reactions changes when the cell volume changes. This is implemented by inserting the volume as a variable in the propensity function:

$$a_V(t) = a(t) \cdot v(t)^{\text{order} - 1}$$

with $\text{order} \geq 2$. For $\text{order} \in 0, 1$, the propensity is unaffected by the volume: $a_V(t) = a(t)$.

3.5.2 Simulating a single lineage

Simulating a whole cell population is computationally expensive. With typical inter division time distributions, it takes about 15-18 generations until a population started with a single cell reaches a state approximating balanced growth. This means that a simulation of the complete population history will involve simulation at least $2^{15}$ cells. Simulation of a single lineage (Figure 9) has a large potential to speed up simulations [10].

![Cell lineage and cell population](image)

Figure 9: Cell lineage and cell population [9]. A cell lineage follows one cell, choosing at each division one of the daughters to follow. A cell population on the other hand follows all daughters after each cell division event.

When certain conditions are met, a simulation of a lineage corresponds to a sample of mother cells. The conditions are deterministic and balanced volume growth, independent mother and daughter volume distributions and determining which daughter to follow with a certain probability. The latter two are described in the following paragraphs.

The first condition that needs to be met is that the mother and daughter cell volume distributions are independent. The mother volume is dependent on the daughter volume if a daughter is larger than the smallest mother. The mother volume has to be larger than this volume and is thus dependent on the previous daughter volume. Independent volumes can be achieved by choosing the mother cell distribution ($\phi$) and partition distribution ($K$) such that there is no overlap possible between mother and daughter cell volumes. We assume that both the mother and daughter distributions are beta distributed. Also the alpha and beta parameters of the beta distributions are equal. This gives bounded and symmetrical distributions. Call the mean of the mother volume $\phi_{\text{mean}}$, then by using symmetry and the definition that a beta distribution is spread over a domain of length one, the smallest mother volume $x$ is $\phi_{\text{mean}} - 0.5$ and the largest mother volume is $\phi_{\text{mean}} + 0.5 = x + 1$. To prevent overlapping of daughter and mother volumes, the largest mother should not divide into a cell that is larger than $x$. This gives that the maximum partitioning value is $\frac{x}{x + 1}$. By symmetry, the smallest partitioning value is $\frac{1}{x + 1}$. Given a beta
distributed random variable \( r \), the volume partitioning ratio \( p \) is calculated from:

\[
p = r \cdot \frac{x - 1}{x + 1} + \frac{1}{x + 1} = \frac{r \cdot (x - 1) + 1}{x + 1}
\]  

(7)

This assures that the mother volume will never depend on the daughter volume.

The second condition is that the single lineage is representative for the whole population. Therefore the decision which of the two daughters to follow after division has to be made according to how many descendants each daughter cell would contribute to a later extant population [10]. If daughter 1 has \( n_1 \) descendants and daughter 2 \( n_2 \) descendants, the probability \( p \) to choose daughter 1 is

\[
p = \frac{n_1}{n_1 + n_2}
\]  

(8)

Let \( t_2 \) be a later time point and \( t_1 \) the inter division time of daughter one, then the number of expected descendants can be calculated using the population growth law: \( n_1 = e^{\mu(t_2 - t_1)} \), where \( \mu \) is the population growth rate. And exponential volume growth is assumed. For daughter 2 the same equation applies: \( n_2 = e^{\mu(t_2 - t_2)} \). Inserting these in Equation 8, gives,

\[
p = \frac{e^{\mu(t_2 - t_1)}}{e^{\mu(t_2 - t_1)} + e^{\mu(t_2 - t_2)}} = \frac{e^{\mu\Delta t}}{1 + e^{\mu\Delta t}}
\]  

(9)

where \( \Delta t = t_2 - t_1 \). In short, the larger daughter cell is more probable to be chosen, because this cell will reach the next mother volume faster and is therefore likely to have more descendants.

### 3.5.3 Extant cell population

Under the conditions described above, the statistics for a sample of an extant cell population can be calculated from the simulation of a single lineage, by using Equation 10 [10].

\[
p(N_x = n) = \int_0^{\tau_{\text{max}}} \int_0^{a_{\text{max}}} f_m(\tau) \cdot \mu \cdot e^{\mu(\tau - a)} \sum_j I(n, a, \tau) \cdot d\tau
\]  

(10)

Here \( I(n, a, \tau) \) is an indicator function equal to the number of occurrences of copy number \( N_x = m \) at cell age \( a \) in a cell with inter division time \( \tau \). Furthermore \( \mu \) is the specific growth rate and \( f_m(\tau) \) is the inter division time distribution of a sample of mother cells. The latter can be directly obtained from the simulation, as the lineage corresponds to a sample of mother cells. Obtaining the statistics for an extant cell population consists of two main steps. First, in order to work with the indicator functions, the simulation time series needs to be binned. The binning needs to be performed at regular time intervals. This is not trivial, because at some points the reactions can be fast and at some point slow, making the time gaps between points irregular. Secondly, a double integral has to be taken. For infinitesimal small steps of \( a \) and \( \tau \) is a slow procedure. Therefore an approximation of the integral is applied, using the the trapezoidal rule. This speeds up the calculation.

The specific growth rate \( k \) can be obtained by solving equation 11 [13].

\[
\int_0^{x_{\text{max}}} \lambda(x) = \int_0^{x_{\text{max}}} e^{-R(x)} \cdot \int_0^{x_{\text{max}}} \frac{\mu e^{R(x)}[2\psi(x) - \phi(x)]}{V(x)} dx + C \cdot e^{-R(x)} dx = 1
\]  

(11)

where

\[
R(x) = \int_0^{x_{\text{max}}} \frac{V'(x) + \mu}{V(x)} dx
\]  

(12)

Where \( x \) is the cell volume, \( \lambda(x) \) the distribution of extant cell volumes, \( \psi(x) \) the distribution of daughter cell volumes, \( \phi(x) \) the distribution of mother cell volumes and \( V(x) \) the differential of the formula of cell volume growth \( V(x) = g \cdot x^t \) for exponential growth and \( V(x) = g \) for linear volume growth, with \( g \) the volume growth rate. We know \( \phi(x), V(x), V'(x) \) and \( C \) from the model parameters. \( \psi(x) \) can be calculated from \( \phi(x) \) and the partition distribution \( K(x) \) using equation 13.

\[
\psi(x) = \int_x^{\infty} \frac{\phi(\theta)}{\theta} \cdot K\left(\frac{x}{\theta}\right) d\theta
\]  

(13)
A solution for equation 11 is approximated by using the secant or Newton-Rhapson method.

**Algorithm 5. Cell Growth and Division**

*Model settings:*

- Initial volume (first daughter volume) \( V_0 \).
- Volume growth law. Either exponential \( (V = V_0 \times e^{g \cdot t}) \) or linear \( (V = V_0 + g \cdot t) \).
- Mother volume distribution \( \phi \).
- Partitioning distribution \( K \).
- An \( \text{end}_\text{time} \), \( \text{end}_\text{steps} \) and \( \text{end}_\text{generations} \) to stop the simulation if finished.

1. **Initialization**
   - (a) Calculate the specific growth rate \( k \) from \( \phi \), \( K \), growth law.
   - (b) Generate the first cell division volume \( V_m \) from \( \phi \).
   - (c) Set the initial number of molecules.
   - (d) Set \( V_d = V_0 \).

2. Calculate the interdivision time \( IDT \) from the growth law, \( V_m \) and \( V_d \).

3. Execute the volume dependent SSA until \( IDT \) or until \( \text{end}_\text{steps} \) or \( \text{end}_\text{time} \) is reached. In the latter case, quit.

4. Divide \( V_m \) in two \( V_{d1} \) and \( V_{d2} \), according to partitioning distribution \( K \).

5. Binomial partitioning of the molecules and delayed reactions, weighted by \( V_{d1} \) and \( V_{d2} \).

6. Choose \( d1 \) or \( d2 \) to follow:
   - (a) Draw the next mother volume \( V_m \) from \( \phi \).
   - (b) Calculate the time up to the next division at \( V_m \) for both daughter cells, and their difference \( \Delta t \).
   - (c) Pick daughter cell \( d1 \) with probability \( p = \frac{e^{k \cdot \Delta t}}{1 + e^{k \cdot \Delta t}} \). Let \( i \) be the index of the chosen daughter.

7. Set \( V_d = V_{di} \).

8. If number of generations exceeds \( \text{end}_\text{generations} \), quit. Else go to step 2.
4 Results

This section elaborates on the details of implementation of the new simulation methods were implemented in the existing StochPy package. We will discuss the difficulties encountered during the implementation and the way the new methods are called in StochPy. In addition we show the results of a number of different simulations, and will elaborate on the difference between the delayed and single molecule methods. At the end we introduce a biologically relevant application of the delayed and cell division methods.

4.1 Pseudocode Delayed Direct Method

4.1.1 Error original pseudocode

As stated in the methods section, a problem was encountered during the implementation of the delayed direct method of Cai. The pseudocode S2 in the Supplementary Information does not return values for \( \tau \) that are in accordance to Equation 3. For some settings, the pseudocode also returns negative values. Upon investigation of the pseudocode, we discovered that the equation in line 16 needs an additional term.

The source of the error is depicted in Figure 10. The original pseudocode does one step more before exiting the while loop, as compared to the improved pseudocode. This adds an additional term, that has to be subtracted (dotted blue line). This subtraction, however, is not performed in the original pseudocode. Adding this correction in line 16 already solves this problem (Pseudocode S3). With this modification, the pseudocode does return \( \tau \) values in agreement with equation 3. In addition to this major error, a smaller problems were also alleviated in the corrected pseudocode (Pseudocode S3). The last two lines should be in the 'Else' block. Otherwise the \( \tau \) assigned at line 3 is always overwritten by the \( \tau \) assignment on the last line.

\[
T_{0}=0 \quad T_{1} \quad T_{2} \quad T_{3} \quad T_{4} \quad T_{5} \quad T_{6}=\infty
\]

\[
at = a_{0}(0) \cdot T_{1} + a_{0}(T_{1}) \cdot (T_{2}-T_{1}) + a_{0}(T_{2}) \cdot (T_{3}-T_{2}) + a_{0}(T_{3}) \cdot (T_{4}-T_{3})
\]

\[-a_{0}(T_{3}) \cdot (T_{4}-T_{3}) + \ln(1-u^{2}) + a_{0}(T_{4}) \cdot (T_{5}-T_{4}) - a_{0}(T_{4}) \cdot (T_{5}-T_{4}) \]

![Figure 10: Timeline with workflow of the (incorrect) Pseudocode S2. The situation is identical to the one in Figure 6. \( a_{t} \) corresponds to the summation of \( a_{0} \), the current absolute time \( t = 0 \) and \( F = 1 - \exp(-a_{t}) \). The steps in the while loop of the pseudocode are indicated with red arrows. At every step an element is added to \( a_{t} \). At \( T_{5} \), \( F > u^{2} \), and the while loop is exited. This is one step later than when using the improved pseudocode (Pseudocode 1 and Figure 6). To move back to \( \tau \), the algorithm needs to subtract the last two additions, one for the step to \( T_{4} \) and one to \( T_{5} \). The last subtraction, from \( T_{5} \) to \( T_{4} \) (dotted blue line), is not done in the original pseudocode, giving erroneous values of \( \tau \). The corrected pseudocode (Pseudocode S3), does include this additional subtraction. The waiting time \( \tau \) is then calculated from \( \tau = T_{3} - \frac{a_{t}}{a_{0}(T_{3})} \).]

4.1.2 Improved pseudocode

The corrected pseudocode does generate the waiting times correctly, but it is not efficient and not straightforward to implement. The pseudocode proposed in the methods section is more efficient and has a few implementation advantages (Pseudocode 1).

Firstly, the propensity function is calculated in each iteration of the corrected pseudocode (line 11), without changing the species vector. However, the propensity function is almost always dependent on the species vector. The species vector is not updated until the next loop, due to the completion of a delayed reaction at \( i - 1 \) (line 14, Pseudocode S2). This has as consequence that to calculate the propensity function, one has to temporarily update the species vector or create a new species vector especially for
this purpose. The improved pseudocode updates the species vector right before the propensity function is calculated. This simplifies the implementation of this improved pseudocode with respect to the original and corrected pseudocodes.

In addition some other difficulties with the original pseudocode are relieved, among which accessing a non-existing index of T and losing data by adding infinitely large numbers to very small numbers. The workflow of the improved pseudocode is also more obvious. The while loop is entered only if a delayed reaction completes and \( \tau \) surpasses the completion of a delayed reaction. The pseudocode therefore takes just one step ahead (these steps are illustrated in figure 6), instead of two in the original pseudocode.

The last improvement is a speed increase of the improved pseudocode with respect to the corrected pseudocode. The speed increase was tested on a small system with only a set of values for T, a set of propensities and a set of fixed random numbers (SI section 6.3). This small test system was executed without a full stochastic simulation, only the both pseudocodes were run with the data from the test system. The time taken for 10,000 runs of this test system was recorded. This was repeated 30 times, yielding an average and standard deviation of the execution time. Table 5 shows a speed improvement of about 37% percent. So the improved pseudocode does not only give a more straightforward, but also a faster way to generate \( \tau \) in a system of delayed reactions.

Table 5: Mean and standard deviation for time taken for 10,000 runs of a small test system for calculating \( \tau \), performed by both the Improved Pseudocode (1) and the Corrected, original, Pseudocode (Pseudocode S3). The mean and standard deviation were taken over 30 executions (of each 10,000 runs).

<table>
<thead>
<tr>
<th></th>
<th>Mean Run time</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved Pseudocode</td>
<td>0.34</td>
<td>0.05</td>
</tr>
<tr>
<td>Corrected Pseudocode</td>
<td>0.47</td>
<td>0.07</td>
</tr>
</tbody>
</table>

4.2 Example usage Delayed Methods

We use a illustrative example to show how the delayed method is implemented in StochPy. The isomerization model (Figure 11) consists of one reaction with a delay. In this reaction a molecule of A is converted into B in reaction R1. This reaction has a fixed, consuming, delay of 1 second. The resulting plot is shown in Figure 12.

Figure 11: The Isomerization model, species A is isomerized into B with a delay between initiation and completion of the reaction. The Pysces model can be found in the SI, section 6.4.

The StochPy commands to simulate the described system are:

```python
import stochpy
smod = stochpy.SSA()
smod.Model( "Isomerization.psc" )
smod.SetDelayParameters( delay_distributions = {"R1": ("fixed",1)},
nonconsuming_reactions = [] )
```
smod.DoDelayedStochSim( method = "DelayedDirect" )
smod.PlotSpeciesTimeSeries()

The `smod.SetDelayParameters()` function is used to add delay parameters to the model. This function is necessary before a delayed stochastic simulation is performed, otherwise a warning is raised. The function takes two arguments: the delay distributions (`delay_distributions`) and, optionally, the nonconsuming reactions (`nonconsuming_reactions`). The delay distributions argument is entered as a Python dictionary with as keys the names of the reactions that are delayed. The values are sets of distribution name and the parameters. The distribution name can be any from the numpy.random module (e.g. ‘gamma’, ‘normal’, ‘beta’) with the addition of ‘fixed’ (parameter is one value). The latter always yields the given, fixed delay time. The nonconsuming reactions has as default an empty list, implying that all delayed reactions are consuming. In this example the nonconsuming reactions parameter can also be left out. A nonconsuming reaction can be set by specifying this explicitly, e.g.

```
smod.SetDelayParameters( {"R1": ("fixed",1)},nonconsuming_reactions = ["R1"] )
```

As an alternative to the reaction name, the index of the reaction can also be used. The index follows the order of occurrence in the model file, starting at 0. In this case, ‘R1’ could be replaced by the index 0, as shown below.

```
smod.SetDelayParameters( { 0 : ("fixed",1)},nonconsuming_reactions = ["R1"] )
```

The function `smod.DoDelayedStochSim()` has as parameter ‘method’, with which either the Delayed Direct Delayed method or Delayed NRM can be selected. By default the Direct Delayed method of Cai is used. The Anderson delayed next reaction method can be selected either by `smod.Method("DelayedNRM")` or `smod.DoDelayedStochSim(method = "DelayedNRM")`. An overview of the functions and options of the delayed methods is given in Table S1.

Figure 12: Time series of a simulation of the isomerization model. After initiation of this reaction, A is consumed. One second later, B is formed. The steps down of A is similar to the steps up of B. The simulation stops after exhaustion of A as the reaction cannot fire anymore.
A few aspects need to be taken into account when using the delayed methods. Firstly, when a new model is loaded or the current one reloaded, the delay parameters have to be set again with the \texttt{SetDelayParameters} function. This means that any previous delay parameters are forgotten. To prevent errors, one should always first load the model and then set the delay parameters.

Secondly, the Numpy distributions use the scale parameter instead of the rate. The scale parameter is the inverse of the rate (\(\text{scale} = \frac{1}{\text{rate}}\)). In general, consult the Numpy documentation \([14]\) before using a Numpy distribution, to get an overview of the possible parameters and in which order they should be given. This overview also lists the size parameter of the distribution, but this parameter is used internally by StochPy and should not be given by the user in the \texttt{SetDelayParameters} function.

Thirdly, be careful with distributions that can return negative delay times. An example is the normal distribution. When a negative delay time is drawn, StochPy will try to redraw a new delay time from the same distribution. This slows down the algorithm and distorts the delay time distribution. After ten subsequent negative delay times, the algorithm interrupts the simulation to prevent an infinite loop.

### 4.3 A system with two reaction channels

Both Cai \([5]\) and Anderson \([11]\) use a simple example to demonstrate their delayed methods. We will recreate their figures using StochPy, using our implementations of the delayed methods. The model used is a dimerization reaction between \(X\) and \(Y\), forming \(Z\) (Figure 13). This dimerization reaction has a fixed, consuming delay of 0.1. In a second, non-delayed, reaction \(Z\) is degraded. Following Cai and Anderson, we chose \(X(0) = Y(0) = 1000\), \(Z(0) = 0\), \(c_1 = 0.001\) and \(c_2 = 0.001\). The system was simulated from \(t=0\) to \(t=1\). The simulation was ran \(1.1 \cdot 10^5\) times with and without delay and the end copy numbers of \(X\) (which is equal to \(Y\)) and \(Z\) were recorded.

Figure 14 shows the histograms of the copy number of \(X\) and \(Z\). For comparison, the simulation was also performed without a delay. There is no difference in the distribution of the copy number of \(X\) is the same for a system with and without a delay. This was expected, because \(X\) (and \(Y\)) is not affected by the delay; at initiation the copy number of \(X\) immediately updates. The histogram of the copy number of \(Z\), on the other hand, shows that the copy number distribution of \(Z\) is affected. The delay causes less \(Z\) to be produced and the copy number of \(Z\) is distributed around a lower mean (around 475). The formation of \(Z\) is delayed by 0.1, compared to the consumption of \(X\) and \(Y\). The simulation still ends at 1, meaning that less \(Z\) can be formed in this time span. In other words, some molecules of \(Z\) are in their delay time and waiting to be produced.

Please note that when the delay would have been nonconsuming, the distribution of the copy number of \(X\) would also have been shifted by the delay. In that case, the consumption of \(X\) is delayed and more \(X\) will be left at the end of the simulation. For more details about consuming and nonconsuming, see the methods section.

![Figure 13: The Cai model. An initial amount of X and Y react to Z (with a delay) and Z subsequently degrades with a first order reaction. The Pysces model can be found in the SI, section 6.4.](image)
4.4 Burst Model

The goal of this section is to validate the implemented delayed methods in a quantitative way. The results of the delayed simulations are compared to analytic distributions. For this purpose a simple burst model was used (Figure 15). In this model a molecule switches from the on state to the off state with a delay. During the delay, and only during the delay, a protein $X$ is produced. An example of one time series is shown in Figure 16. The statistic under interest is the number of molecules of $X$ that are produced during the delay time (e.g. 6 in Figure 16). The distribution of this number can be solved analytically. Namely, the distribution is a probability mixture of the delay distribution and the product formation distribution (a Poisson distribution). The distribution of the number of $X$ molecules was simulated with StochPy by 10,000 runs for both delay methods (Mathematica was used to calculate the analytic distribution).

Figure 17 shows the results of the simulations. The analytical distribution almost perfectly matches the histograms of both the Delayed Direct method and Delayed NRM. Small deviations can be attributed to the stochastic nature of the simulations. So the results are analytically correct for the given model and delay distributions. This indicates reason that both delayed methods are implemented correctly in StochPy.
Figure 15: Adapted burst model. The on state reacts to off with a (consuming) delay. During this delay, \( X \) is produced. The Pysces model can be found in the SI, section 6.4.

Figure 16: One time series of the burst model. The on state (blue) reacts to off (green) with a delay. During the delay \( X \) (red) is formed stochastically with rate 1.
Figure 17: Distribution of number of molecules of X produced in the burst model with four different delay distributions; A) Gamma(1,1), B) Gamma(9,0.5), C) Normal(5,1) and D) Exponential(0.1). 10,000 runs were performed with both the Delayed Next Reaction Method (blue) and the Delayed Direct Method (green). The black line and diamonds mark the analytical solution for each distribution.

4.5 Example Usage Single Molecule Method

To illustrate the usage of the Single Molecule Method, the Transcription with intermediate model is used (Figure 18). This is a transcription model with an explicit intermediate form of polymerase, polymerase moving. Polymerase starts moving when reaction $R_1$ fires, and attains the state ‘Polymerase Moving’. The next reaction, $R_2$ has a fixed putative reaction time of 100 seconds. This results in a systems where mRNA (and free polymerase) is always formed 100 seconds after firing of reaction $R_1$. In other words, it takes the polymerase 100 seconds to transcribe an mRNA molecule. mRNA is subsequently degraded.

The StochPy commands to simulate the described system are:

```python
import stochpy
smod = stochpy.SSA()
smod.Model("TranscriptionIntermediate.psc")
smod.SetPutativeReactionTimes({"R2":("fixed",100)})
smod.DoSingleMoleculeStochSim(end = 1000, mode = "time")
smod.PlotSpeciesTimeSeries()
```
In Stochpy, the fSMM is used as default method to perform a single molecule simulation. The normal Single Molecule Method can be selected by `smod.Method( "SMM" )` or `smod.DoSingleMoleculeStochSim(method = "SMM")`. The function `.SetPutativeReactionTimes` accepts a dictionary with reaction name or index as key and distribution and parameters as value (e.g. `{1:("gamma",3,1)}`). This is exactly the same as the delay distributions in the `.SetDelayParameters` function and the notes on the Numpy distributions listed in section 4.2 also apply to the putative reaction time distributions. An overview of all SMM functions can be found in Table S2.

Note that the rate equations must be constructed carefully. The rate equation should contain the species on which the reaction is dependent. In the most simple case, this only lists the reactants (separated by *). For example the following model definition of a Single Molecule Reaction (SMR). Note that in the fSMM this reaction should also be set as a SMR by using `.SetPutativeReactionTimes`.

R1:

\[ X + Y \rightarrow Z \]
\[ X*Y \]

The rate equation is not used as a propensity function for a SMR, only the presence of the species is important. Another example will underline the importance. Consider the same reaction as above, but with \(Y\) as catalyst instead of reactant: \(X \rightarrow Z\). Species \(Y\) should then still be mentioned in the rate equation, \(X*Y\). This copies the behavior of the other methods, where the rate of the reaction is determined by the rate equation.

One consequence of the above is that a SMR always has a linear dependence on the species named in the rate equation. The construction of the formula is completely ignored. For example a rate equation like \(0.5*k1*X*(1-Y)\) will produce the same result as \(X*Y\) or \(X/Y\). Note that the rate equation has to be a valid equation to be read correctly (e.g. \(X,Y\) is not accepted).

Due to the linear dependence of SMRs on one or two species, these reactions can only be simulated by mass action kinetics. Non-linear rate equations, like Michaelis-Menten, cannot be simulated with a SMR. The formula in the rate equation is not used, only the species present in this equation are read. Thus for non-linear reactions, a SMR will give different results than a traditional Gillespie reaction, which evaluates the rate equation to calculate the propensity. However, the fSMM is capable of handling non-linear rate equations. These reactions should then not be manually set as a SMR, by not naming these reactions in the `.SetPutativeReactionTimes` function. Only the reactions which are given a putative reaction time are treated as SMRs, the other reactions are handled as in the Next Reaction Method. The normal SMM will treat every reaction as a SMR.

The normal Single Molecule Method considers all reactions as single molecule reactions. The reactions that are not given a manual putative reaction time distribution, are given an exponential distribution. The rate parameter for this distribution is ‘guessed’ by evaluating the rate equation. All species copy numbers are first set to 1000, then the rate equation is evaluated. The resulting number is then divided by 1000 to
the power of the number of species there were. In the case of a simple linear rate equation, this method will get the rate constant (e.g. $k_2 \cdot X \cdot Y$). The choice for entering 1000 for the species and dividing by 1000 again, and not by 1, was made to deal with rate equations like $k_1 \cdot X \cdot (X-1)$. Here we expect the rate to be $k_1$, but entering 1 will give 0. By following the procedure as described above, the result will approximate $k_1$.

As already noted in the results section, the SMMs do not accept reactions with an order higher than two. Due to computationally restrictions, every reaction can be dependent on maximally two species. A third or higher order reaction should therefore be split up in multiple first or second-order reactions. Alternatively, if this particular reaction has a exponential waiting time, the fast SMM can be used to model it like in the NRM, without restrictions on the order.

### 4.6 Comparison Delayed Method and Single Molecule Method

In the next section we will compare the Delayed methods and SMMs. We will illustrate the differences, but more importantly also show how the same results can be retrieved with both methods.

As stated in the methods section, the main difference between the delayed methods and the SMM is that a reaction in the delayed methods consists two steps and in the SMM of one step. The delayed method has an exponential initiation step followed by a random delay, the SMM only has a putative reaction time from a given random distribution.

The system used in this section is a linear reaction of molecule $A$ to a Product with four sequential steps. This can be a simplistic model for transcription, where a polymerase takes hundreds of steps to form a mRNA molecule. The four reactions are modeled with an exponential waiting time distribution, with mass-balance kinetics. The model is shown in Figure 19A, and will be referred to as the fine grained model. This model will be simulated using the Gillespie Direct Method.

The model can also be coarse grained, where the system is modeled as one reaction (Figure 19B). To reflect the underlying process of four sequential steps, this reaction has gamma distributed waiting time. The gamma distribution has two parameters, the shape and scale, and gives the distribution of the sum of multiple identical exponential distributions. The shape parameter gives the number of underlying exponential reactions, and the scale parameter the scale of each exponential reaction. In this case the shape is 4 and scale is $\frac{1}{k_1}$ (if $k_1 = k_2 = k_3 = k_4$). This coarse grained model is simulated by both the delayed method and single molecule method. Coarse graining has the advantage that a) less parameters need to be known and b) the model becomes simpler (less reactions). In addition the simulation time will decrease, because less simulation steps are needed. Figure 19C shows the last variation of the model, with one intermediate between $A$ and Product. In the transcription example, the intermediate is a state where the polymerase is moving over the DNA.
The model without intermediate will be simulated by the delayed method and by the single molecule method. The model with intermediate will only be simulated by the single molecule method. The model with and without intermediate will illustrate the difference between the two methods. The fine grained model is simulated with the Direct Gillespie Method will serve as the golden standard, because it describes the underlying process. An overview of the models and simulation methods can be found in Table 6.

Table 6: Overview of the models and simulation methods used for each model.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Model type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Gillespie Direct Method</td>
<td>A) Fine grained</td>
</tr>
<tr>
<td>2 Delayed Direct Method</td>
<td>B) Coarse grained without intermediate</td>
</tr>
<tr>
<td>3 Single Molecule Method</td>
<td>B) Coarse grained without intermediate</td>
</tr>
<tr>
<td>4 Single Molecule Method</td>
<td>C) Coarse grained with intermediate</td>
</tr>
</tbody>
</table>

The four different methods were used in three different model parameter settings, two without only the linear model without branch, of which one with identical rate constants and one with a fast initial step. The third model setting includes the branch at $A$ with identical rate constants. We will call these model parameter settings ‘situations’. All situations start with one molecule of $A$ and the simulation ends when Product is formed. For each method 50,000 simulations (100,000 for the situation with branch) are performed and the time until Product is formed is recorded.

4.6.1 Identical rate constants

In the first situation, the four sequential reactions are all identical; they all have the same rate constant. The settings for each model are given in Table 7. Similar results were retrieved for all four simulation...
methods (Figure 20).

To accomplish similar results, the choice of the delay and putative reaction time distributions is important. A gamma distribution was chosen, because it can be used to represent a sequence of exponential reactions. The scale parameter is set to one, because every exponential reaction has a scale (and also rate) of one. The shape, determined by the number of underlying reactions, is different for each method. For the Delayed Method, the exponential step is the first reaction and the delay represents the other three. So the delay is given a gamma distribution with a shape of three. The SMM without intermediate has no exponential step, so the shape is set to four to represent all four exponential reactions. With an intermediate, the shape is again set to three, because after the intermediate there are three reactions to the product left.

Table 7: Parameters for the identical rate constants situation. For the delayed method the rate constant and delay distribution is given. The two numbers behind the distribution name indicate the scale and shape respectively. For the SMM’s the putative waiting time distribution replaces the rate constants (k1 or k2) completely. The rate of reaction 5 is set to 0, such that no branching will occur.

<table>
<thead>
<tr>
<th>Reaction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine Grained</td>
<td>k1=1</td>
<td></td>
<td>k2=1</td>
<td>k3=1</td>
<td>k4=1</td>
</tr>
<tr>
<td>Delayed Method</td>
<td>k1=1,</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>k5=0</td>
</tr>
<tr>
<td></td>
<td>delay = “gamma” (3, 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMM</td>
<td>“gamma” (4, 1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>k5=0</td>
</tr>
<tr>
<td>SMM with intermediate</td>
<td>k1=1</td>
<td>“gamma” (3, 1)</td>
<td>-</td>
<td>-</td>
<td>k5=0</td>
</tr>
</tbody>
</table>
4.6.2 Fast initial step

The second situation includes a fast first reaction, followed by three identical reactions. Again the situation is modeled using four different methods (Table 8).

Table 8: Parameters for the fast initial step situation. For the SMM’s the putative waiting time distribution replaces the rate constants (k1 or k2) completely. The two numbers behind the distribution name indicate the scale and shape respectively. Note that a scale of 0.5 corresponds to a rate of 2. The rate of reaction 5 is set to 0, such that no branching to Z will occur.

<table>
<thead>
<tr>
<th>Reaction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine Grained</td>
<td>k1=5</td>
<td></td>
<td>k3=1</td>
<td>k4=1</td>
<td>k5 = 0</td>
</tr>
<tr>
<td>Delayed Method</td>
<td>k1=5, “gamma” (3, 1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>k5 = 0</td>
</tr>
<tr>
<td>SMM</td>
<td>“gamma” (4, 0.5)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>k5 = 0</td>
</tr>
<tr>
<td>SMM with intermediate</td>
<td>k1=5</td>
<td>“gamma” (3,1)</td>
<td>-</td>
<td>-</td>
<td>k5 = 0</td>
</tr>
</tbody>
</table>
Figure 21: Distribution of time until formation of Product for the fast initial step situation (50000 trajectories). The parameters are as named in Table 8. Panel A shows the underlying fine grained process, panels B, C and D represent three ways to coarse grain the model: delayed method (B), SMM without delay (C) and SMM with delay (D). The black line is the expected distribution of total time until Product is formed, which is a hypoexponential distribution with the rates 5, 1, 1 and 1, describing the underlying process.

Figure 21 shows that only the SMM without intermediate deviates from the other models. This is due to the gamma distribution, which does not correctly describe the underlying time distribution. A gamma distribution assumes identical rate constants of the underlying exponential process, while the current model settings includes a high rate for the initial reaction. The gamma distribution with a scale of 0.5 and shape of four, is equivalent to four reactions with a rate of two. Although the average rate is the same in this case, it does not give the same distribution.

The Delayed Method and the SMM with intermediate do give the correct time distributions and are thus in accordance with the fine grained model. In these two methods, the initial step is modeled implicitly and explicitly, respectively. The three subsequent reactions have identical rates (all one) and can be modeled with the gamma distribution.

To use the SMM to model the system without intermediate, a different waiting time distribution should be used. This can be achieved with an hypoexponential distribution. A hypoexponential distribution takes a list of exponential rates of the underlying process and returns a distribution of the total waiting time. The black line in Figure 21 is the hypoexponential distribution given the sequential rates 5, 1, 1 and 1. This line was produced by a small script provided in SI section 6.5. The SMM without intermediate was not modeled using this hypoexponential distribution, because it is not present in the numpy library.

4.6.3 Branch

In the last situation we activate the branch (reaction 5) in the system. The rate constant for the branch is $k_5 = 1$, all other parameters are as in the situation of the identical rate constants (Table 9). The simulation
ends if \textbf{Product} or \textbf{Z} is formed. Only the time to form product is stored and used for analysis.

Table 9: Parameters for the situation with a branch path. For the SMM’s the putative waiting time distribution replaces the rate constants (k1 or k2) completely. The two numbers behind the distribution name indicate the scale and shape respectively.

<table>
<thead>
<tr>
<th>Reaction</th>
<th>\textbf{1}</th>
<th>\textbf{2}</th>
<th>\textbf{3}</th>
<th>\textbf{4}</th>
<th>\textbf{5}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine Grained</td>
<td>k1=1</td>
<td>k2=1</td>
<td>k3=1</td>
<td>k4=1</td>
<td>k5=1</td>
</tr>
<tr>
<td>Delayed Method</td>
<td>k1=1, delay = “gamma” (3, 1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>k5=1</td>
</tr>
<tr>
<td>SMM</td>
<td>“gamma” (4, 1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>k5=1</td>
</tr>
<tr>
<td>SMM with intermediate</td>
<td>k1=1</td>
<td>“gamma” (3, 1)</td>
<td>-</td>
<td>-</td>
<td>k5=1</td>
</tr>
</tbody>
</table>

Figure 22: Distribution of time until formation of \textbf{Product} for the situation with a branch (100,000 trajectories). The parameters are as given in Table 9. Panel A shows the underlying fine grained process, panels B, C and D represent three ways to coarse grain the model; delayed method (B), SMM without delay (C) and SMM with delay (D). The black line is a gamma distribution with shape = 4 and scale = 1 and given for comparison.

The first observation is that, compared to the gamma distribution (black line in Figure 22A,B,D), the histograms are slightly shifted to smaller waiting times. This gamma distribution is the expected waiting time distribution without a branch. The observed shift is due to the ‘competition’ between the reaction to \textbf{Product} and to \textbf{Z}. The reactions have the same rate, so a molecule of \textbf{A} has an equal chance react to \textbf{B} (and ultimately form product) or react to \textbf{Z}. Indeed, except for the model without intermediate, in about half of the simulations the \textbf{Product} is formed (Table 10). The shift of waiting time distribution is observed, because the \textbf{Product} is only formed if the waiting time to form \textbf{Product} is less than the waiting time to react to \textbf{Z}. Therefore the smaller waiting times are more frequent, resulting in a distribution around
smaller waiting times.

The waiting time distribution of the fSMM without intermediate deviates significantly from the other three histograms (Figure 22C). This is due to lack of the initial exponential step to an intermediate. The reaction from A to Product is one step, with a relatively long waiting time (gamma distributed). In most simulations the reaction to Z has a lower waiting time and Z is formed. This is also emphasized by Table 10. Product was formed in just 1.23% of the 100,000 simulations. The histogram in Figure 22C is build out of fewer data points, explaining the less smooth appearance of the histogram. In addition the histogram is even more shifted to the left than the other three histograms. The reason for this difference is that only the very small waiting times ‘win’ and form Product. This leaves just a few, but very small, times to form Product. For the other three models, only the first reaction to B or the intermediate should be fast. After that step, the simulation is locked in this branch and the next reaction times or delays are allowed to much higher.

Table 10: Percentage of the 100,000 simulations in which Product is formed, for the situation with branch. In the other simulation, Z is formed. Note that for model 1,3 and 4 in about half of the simulations Product is formed, which one would expect in an equally branching model. For model 2 on the other hand, just in 1.23% of the simulations the Product is formed.

<table>
<thead>
<tr>
<th>Method</th>
<th>Direct</th>
<th>SMM without intermediate</th>
<th>SMM with intermediate</th>
<th>Delayed Direct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentage of simulations in which Product is formed</td>
<td>53.4%</td>
<td>1.23%</td>
<td>53.3%</td>
<td>46.6%</td>
</tr>
</tbody>
</table>

### 4.7 Cell Division

#### 4.7.1 Example Usage

Below is a sample code using the Cell Division module. As an example, the Immigration Death model (Figure 23) is used. It simulates five generations and plots both the copy number and volume time series (Figure 24). Note that the volume is discrete. This is also how it is handled in the module itself. Only upon updating the species vector, the volume vector is updated. Some of the cell division parameters are manually set. The cell volume is set to start at 1.0 and grows to a volume drawn from the phi distribution, which is given as a beta distribution with a mean of 2.0. A complete overview of all cell growth and division setting is given in Table S3.

```python
import stochpy

cmod = stochpy.CellDivision()
cmod.Model("ImmigrationDeath")

# Set cell division parameters
cmod.SetGrowthFunction(growth_rate = 0.1, growth_type = "exponential")
cmod.SetInitialVolume(initial_volume = 1.0)
cmod.SetVolumeDistributions(phi = ("beta",5,5), K = ("beta",2,2), phi_beta_mean = 2.0)
cmod.SetVolumeDependencies(IsVolumeDependent = True)

# Simulation and plotting
cmod.DoCellDivisionStochSim(end=5, mode="generations")
cmod.PlotSpeciesVolumeTimeSeries(plottype = "amounts")
```
Figure 23: The immigration death model. The Pysces model can be found in the SI, section 6.4.

Figure 24: Time series of the copy number (A) and volume (B) of the immigration death model. Cell volume starts at 1 and grows exponentially with a growth rate of 0.1. Note that the volume growth is not continuous, but increases step wise.

4.7.2 Volume dependencies

The reaction propensities can be made dependent on the volume. Also the order of a reaction can be manually given. If no manual settings are given, the module will automatically determine the order of each reactions and implement the volume dependency accordingly. The order is determined by
the number of species names in the rate equation. Extracellular species, set manually by the option SpeciesExtracellular in the cmod.SetVolumeDependencies function, are subtracted from this number.

The volume dependency is not an exact implementation, but an approximation. In a real cell, volume growth is continuous, but in the module it is simulated with discrete growth steps (Figure 24B). At each reaction of the system, for instance when reaction 1 fires, the volume is updated. The volume dependent propensities are also only recalculated when the volume is updated. This has as a consequence that the propensities do not ‘feel’ any volume growth until the next reaction. This is a good approximation if the reactions are fast relative to the generation time, because the volume is then updated regularly and discrete increases in volume are small. For slow reactions relative to the generation time, the implemented volume dependency is not a proper approximation of the real dependency.

4.7.3 Extant cell population

The analysis can be called by using cmod.AnalyzeExtantCells. This function extrapolates the species statistics for a population of extant cells for the last simulation from the last stochastic simulation. The conditions described in the methods section need to be fulfilled, otherwise the analysis will exit with a warning. Two options can be set, n_bins and n_samples. The first gives the number of bins to use for binning of the inter division time and age and the latter gives the number of (equally spaced) samples to take from the simulation. The higher these numbers, the more precise the result. This is, however, at the cost of calculation speed. The speed of the analysis can be affected dramatically due to the high cost of the integrating functions.

Note that the analysis needs the specific growth rate. In the case of exponential volume growth, the given volume growth rate equals the specific growth rate. In the case of a linear volume growth, the specific growth rate can be calculated by the module. This can only be done if beta distributions for phi and K are given, because in this case the distributions have well defined bounds. The specific growth rate is also needed during the simulation, to choose the daughter to follow at cell division. The specific growth rate can namely be used to determine how many offspring each daughter is expected to have and thus with what probability to select each the daughter. After the analysis is finished, the result can be visualized by using cmod.PlotExtantSpeciesDistribution. This command plots the extant species copy number probability mass function for every species (Figure 25).
Figure 25: Extant mRNA copy number distribution after 5000 generations of the immigration death model. From the simulated cell lineage (the first five generations are shown in Figure 24), we expect the (extant) cell population to have the given mRNA copy number distribution. The simulation settings are described in the example code at the start of section 4.7.1.

4.7.4 Additional plotting functions

The cell division module has some additional plotting functions. Besides plotting the standard species copy number time series, it is also possible to plot a volume and species concentration time series. Concentration is calculated by dividing, at each time point, the copy number by the volume. Next to the new time series plotting functions, a range of distribution plots were added. An overview of all the addition plotting functions can be found in Table S4.

4.8 Two-component signaling model

In a parallel study, the new StochPy cell division module and delay algorithms were applied on a two-component signaling model [15] (Figure 26). This system is prevalent in the membranes of prokaryotic cells (e.g. the PhoP/PhoQ system), which enables the cell to detect extracellular ligands (L). It consists of two key species, a transmembrane histidine kinase sensor protein (S) and a cytosolic response regulator protein (R). Both species can be (de)phosphorylated and can transfer a phosphate between each other. The phosphorylated response regulator, RP, is a transcription factor, influencing the transcription of genes in response to the signal L. This system typically operates at low copy numbers, making it sensitive to noise. It is therefore especially interesting to model this system with stochastic simulations.

The system was expanded with the autoregulatory transcription and translation of the R and S proteins (Figure 27). The transcription is under control of two promoters, one controlled by the phosphorylated R dimer (RP) and one constitutive. R and S lie on the same operon, where the transcription of R is always followed by the transcription of S. The transcription and translation times are modeled...
with delays. The delays are distributed with a gamma distribution. The mean delay times are shown in the diagram of Figure 27. The delays were modeled using the direct delayed method and the exact parameters are reported in Table S5. One challenge was to correctly model the sequential transcription of R followed by S. This can be implemented in multiple ways, for instance with the single molecule method, but it was chosen to model it with only delays. In the first transcription reaction R was produced with a delay, which immediately triggered the initiation of the transcription of S. After each transcription event, a translation event started to produce the corresponding protein. All reactions described here were modeled with a delay.

Figure 26: Two-component signaling model. S is the sensor, R the response regulator, L the ligand or signal and P is phosphate. RP is a transcription factor (TF) inducing gene expression.
Figure 27: Transcription and translation of R and S, induced by a RP dimer. The mean time for transcription and translation are given. Both events were modeled with a gamma distributed delay time after initiation of transcription.

To obtain a realistic model, we also accounted for stochastic cell growth and division. This was important, because mathematical analysis showed that the autoregulatory system has a high sensitivity on changes of the protein degradation rate [15]. Both cell growth and division have a high impact on this rate. The cell doubles in size in one generation time, significantly diluting the protein content. In systems where simulation time exceeds the generation time, the dilution is often the primary source of protein concentration degradation. This is the case for the two-component signaling model, because the timescale to settle into steady state, on the order of tens of minutes, is similar to the generation time of a typical prokaryote (around 20 minutes for E. coli). Cell division halves the contents of the cell binomially, especially impacting molecules at low copy numbers. The RP concentration typically ranges around tens to hundreds of copies, making the concentration dependent on the binomial species division.

The cell growth and division module of StochPy, which is described in the previous sections, was used to simulate the two-component system. The inserted parameters for cell volume growth and cell division are reported in Table S6. The volume growth was chosen such that it led to a generation time of around 40 minutes, typical for bacteria such as E. coli.

The stochastic simulation was run for multiple generations (Figure 28). Previous simulations showed bistable behavior of RP, therefore one simulation was started in the low state and one in the high state. Both the low and high state are shown to be stable, leading to the conclusion that the system has bistable steady-state RP concentrations. Due to stochastic fluctuations, the state also occasionally switches spontaneously from the low to the high state. The reverse was not observed. This can be explained by a higher energy barrier to switch from high to low than to low to high. The simulation illustrates the power of stochastic simulations. Bistable switching cannot be observed in deterministic models, where noise is not present.
Figure 28: Time series of the copy number of RP. The system was both started in the low state (A) and in the high state (B). In the low state a few trajectories switch to the high state can be observed.
Discussion and Conclusion

The first goal of this project was to extend the existing StochPy software with delayed reactions, to give StochPy the capability to simulate processes like translation and transcription. We successfully implemented two different delayed methods, the delayed direct method of Cai [5] and the delayed next reaction method of Anderson [11]. A correction for the delayed direct method was given, both correcting an error and reduce the simulation time. We showed that both methods produce the same results in StochPy as originally reported by both Cai and Anderson.

In addition, the Single Molecule Method (SMM) was newly developed and implemented. This method, or an equivalent, has to our knowledge not been described before. It enables the StochPy user to choose non-exponential distributions for the reaction times. The main motivation behind the method was that the delayed methods split a reaction into two steps, an initiation reaction and a delay until completion. This is not always desirable, because experimental waiting time distributions often only give one distribution, not a combination with an exponential distribution. The separate initiation and delay are not observable. An alternate version of the SMM, the fast SMM (FSMM), was also described and implemented. The FSMM has improved performance and allows traditional reactions besides the single molecule reactions.

The difference between the SMM and the delayed method was explained by making a coarse grain model of a system with four sequential reactions. The SMM needs an explicit intermediate to get the same results as for the delayed methods. This is because the delayed method has an implicit intermediate, which exists between initiation and completion. Using a single molecule method without intermediate causes different results in more complicated models, e.g. with branches. This cannot be corrected with a different waiting time distribution, as was done for the situation with fast initial step. A model with sequential reactions needs a (relatively fast) reaction that commits the molecule to the sequence. In the delayed methods this commitment is achieved via an exponential step before the delay, creating an implicit intermediate. In the single molecule method, the intermediate should be explicitly modeled. Concluding, the choice to use the either a delayed method or single molecule method depends on the way one wants to model the system. If the system is modeled without intermediate states, one should choose a delayed method. If the system is modeled with intermediate states (e.g. moving Polymerase, moving ribosomes), one should choose a single molecule method. Using the delayed method with intermediates or single molecule method without, is considered a wrong model.

Both the SMM and FSMM have a few disadvantages. As described above, the model should be carefully constructed. The algorithms are also both computationally expensive. Although the FSMM outperforms the SMM, it is still performs worse than other stochastic simulation algorithms. The SMM has two more limitations, it cannot handle third or higher order reactions and can only simulate mass action kinetics. These problems are partly alleviated in the FSMM, where higher order and non-mass action kinetics can be simulated with traditional (non single molecule) reactions. But the FSMM can only handle single molecule reactions up to first order. In future work, the limitation on the order of the reaction can be removed. This is, however, at the cost of performance, as larger arrays of waiting times are needed. The limitation to mass actions kinetics for single molecule reactions, on the other hand, is fundamental for the single molecule method and cannot be overcome.

The second part of this project compromised the implementation of an extensive cell growth and division module. This module was also successfully implemented in StochPy. Stochasticity in cell volume arises from a distribution of volumes at division (mother volumes) and stochastic volume partitioning distribution. At division the species divide binomially over the two daughter cells. At each division, one daughter is tracked, creating a cell lineage. Several plotting option are added to offer easy visualization of the simulated cell lineage. An advanced cell population statistics analysis is available to extrapolate statistics of the extant cell population from the simulation of the cell lineage.

There are several features with which the cell division module can still be extended. In the current module, volume is considered a deterministic process and the volume grows either exponentially or linearly. In reality, the volume growth rate is heterogeneous, resulting in stochastic volume growth. This was previously modeled by Gomez et al. [16], considering volume growth as an additional reaction channel. Besides a heterogeneous growth rate, various factors also affect volume growth and division in a cell. Species (e.g. a growth factor) can actively change the volume growth rate or induce cell division. A last modification is volume or cell age triggered reactions. Certain reactions take place in a certain stage of the cell cycle, for instance DNA replication.

The new stochastic delays and stochastic cell division module create opportunities to simulate a wide
variety of systems. Delayed reactions are particularly important to model systems with transcription and translation, two vital components in living cells. One example of an elaborate system is provided; the two-component signaling model, using delays for the production of protein within a model growing and dividing cell.
References


6 Supplementary Information

6.1 Cumulative distribution function \( F \) of \( \tau \). Equation 10 of cai.

\[
F_\tau(\tau) = 1 - \exp \left( - \sum_{j=0}^{i-1} [a_0(t + T_j) \cdot (T_{j+1} - T_j)] - a_0(t + T_i) \cdot (\tau - T_i) \right)
\]  

(S1)

Where \( \tau \in [T_i, T_{i+1}), i = 0, \ldots, N_d \)  

(S2)

6.2 Original and corrected pseudocodes

Both the presented pseudocodes were not used in StochPy. For illustration purposes only.

**Data:** Propensity functions, state vector, \( T_0 = 0 \) and \( T_{N_{\text{delayed+1}}} = \infty \)

**Result:** Generates the event waiting time \( \tau \) according to equation 3

1. Generate a realization of the standard uniform random variable, \( u_2 \)

2. if no ongoing delayed reaction then

3. \[ \tau = \frac{-\ln(u_2)}{a_0(t)} \]

4. else

5. \( i = 0 \)

6. \( a_t = a_0(t) \cdot T_1 \)

7. \( F = 0 \)

8. while \( F < u_2 \) do

9. \( F = 1 - \exp(-a_t) \)

10. \( i = i + 1 \)

11. Calculate propensity functions due to the completion of the delayed reaction at \( t + T_i \) and calculate \( a_0(t + T_i) \)

12. \( a_t = a_t + a_0(t + T_i) \cdot (T_{i+1} - T_i) \)

13. if \( i > 1 \) then

14. Update the species vector due the completion of the delayed reaction at \( t + T_{i-1} \)

15. \( i = i - 1 \)

16. \( a_t = a_t - a_0(t + T_i) \cdot (T_{i+1} - T_i) \)

17. \( \tau = T_i - \frac{\ln(1 - u_2) + a_t}{a_0(t + T_i)} \)

**Pseudocode S 2:** Original pseudocode, as initially described by Cai [5], for generating the waiting time \( \tau \) in a system with delays.
Data: Propensity functions, state vector, $T_0 = 0$ and $T_{N_{delayed}+1} = \infty$

Result: Generates the event waiting time $\tau$ according to equation 3

Generate a realization of the standard uniform random variable, $u_2$

if no ongoing delayed reaction then

\[
\tau = \frac{-\ln(u_2)}{a_0(t)}
\]

else

\[
i = 0
\]

\[
a_t = a_0(t) \cdot T_1
\]

\[
F = 0
\]

while $F < u_2$ do

\[
i = i + 1
\]

Calculate propensity functions due to the completion of the delayed reaction at $t + T_i$ and calculate $a_0(t + T_i)$

\[
a_t = a_t + a_0(t + T_i) \cdot (T_{i+1} - T_i)
\]

if $i > 1$ then

\[
i = i - 1
\]

\[
a_t = a_t - a_0(t + T_i) \cdot (T_{i+1} - T_i) - a_0(t + T_{i+1}) \cdot (T_{i+2} - T_{i+1})
\]

\[
\tau = T_i - \frac{\ln(1 - u_2) + a_t}{a_0(t + T_i)}
\]

Pseudocode S 3: Corrected Pseudocode for generating $\tau$
6.3 New Pseudocode speed improvement test set

The following set of variables were used to test the performance of the improved pseudocode. A fixed set of T_struct’s, propensities sums sim_a_mu_zeros and randoms were used to create a controllable testing environment.

\[
\begin{align*}
T_{\text{struct}1} &= [(0, \text{"nan"}), (0.005, 1), (0.03, 0), (0.06, 2), (0.425, 4), (0.72, 3), (\infty, \text{"nan"})] \\
T_{\text{struct}2} &= [(0, \text{"nan"}), (0.005, 1), (0.03, 0), (0.06, 2), (0.425, 4), (0.72, 3), (1e4, \text{"nan"})] \\
T_{\text{struct}3} &= [(0.005, 1), (0.03, 0), (0.06, 2), (0.425, 4), (0.72, 3)] \\
T_{\text{struct}4} &= [(0, \text{"nan"}), (0.005, 1), (0.03, 0), (0.06, 2), (0.425, 4), (0.85, 3), (1e4, \text{"nan"})] \\
sim_a\_mu\_zeros &= [20, 22, 10, 9, 1, 10] \\
randoms &= [0.01, 0.32, 0.99, 0.21, 0.55, 0.67]
\end{align*}
\]

6.4 Models

The ‘Isomerization’ model

\[
R1: \quad A > B \\
\text{k1}\ast A
\]

# Parameters

k1 = 0.5

# Initial species values

A = 20 \\
B = 0

With .SetDelayParemeters("R1":('fixed',1)).

The ‘Cai’ model

\[
\begin{align*}
R1: & \quad X + Y > Z \\
& \quad c1\ast X\ast Y \\
R2: & \quad Z > \$pool \\
& \quad c2\ast Z
\end{align*}
\]

# Parameters

\[
\begin{align*}
c1 &= 0.001 \\
c2 &= 0.001
\end{align*}
\]

# Initial species values

\[
\begin{align*}
X &= 1000 \\
Y &= 1000 \\
Z &= 0
\end{align*}
\]

With .SetDelayParemeters("R1":('fixed',1)).

The ‘Burstmodel’

\[
\begin{align*}
r1: & \quad \text{On} > \text{Off} \\
& \quad \text{k1}\ast \text{On} \\
r2: & \quad \$pool \ast X \\
& \quad \text{k2}\ast (1\ast \text{Off})\ast (1\ast \text{On})
\end{align*}
\]

49
# Parameters
k1 = 1

# Initial species values
On = 1
Off = 0
X = 0

With .SetDelayParameters("r2":('fixed',5)).
The burst model is constructed in such a way that X is only produced during the delay, when there is no On state and no Off state. Note that the delay needs to be consuming in order for this behavior to occur. Otherwise On is consumed at the same time Off is produced, namely after the delay.

The ‘Transcription Intermediate’ model

R1:
   Polymerase > PolymeraseMoving
ekini*Polymerase

R2:
   PolymeraseMoving > mRNA + Polymerase
   0

R3:
   mRNA > $pool
   kdeg*mRNA

# Parameters
kini = 1
kdeg = 1/600 #1/10 min

# Initial species values
mRNA = 0
Polymerase = 1
PolymeraseMoving = 0

The Single Molecule Example

R1:
   X > Z
   X*Y

# Initial species values
X = 10
Y = 5
Z = 0

With smod.SetPutativeReactionTime("R1": ("gamma",5,5) ).

Comparison SMM and delayed methods

Fine Grained model

R1:
   A > B
   k1*A
R2:
   B > C
\[ k_2 B \]
\[ R_3: \]
\[ C \rightarrow D \]
\[ k_3 C \]
\[ R_4: \]
\[ D \rightarrow \text{Product} \]
\[ k_4 D \]
\[ R_5: \]
\[ A \rightarrow Z \]
\[ k_5 A \]

# Initial species values
\[ A = 1 \]
\[ B = 0 \]
\[ C = 0 \]
\[ D = 0 \]
\[ \text{Product} = 0 \]
\[ Z = 0 \]

Coarse grained without intermediate

\[ R_1: \]
\[ A \rightarrow \text{Product} \]
\[ k_1 A \]
\[ R_5: \]
\[ A \rightarrow Z \]
\[ k_5 A \]

# Initial species values
\[ A = 1 \]
\[ \text{Product} = 0 \]
\[ Z = 0 \]

Coarse grained model with intermediate

\[ R_1: \]
\[ A \rightarrow \text{A}_{\text{intermediate}} \]
\[ k_1 A \]
\[ R_2: \]
\[ \text{A}_{\text{intermediate}} \rightarrow \text{Product} \]
\[ k_2 \text{A}_{\text{intermediate}} \]
\[ R_5: \]
\[ A \rightarrow Z \]
\[ k_5 A \]

# Initial species values
\[ A = 1 \]
\[ \text{A}_{\text{intermediate}} = 0 \]
\[ \text{Product} = 0 \]
\[ Z = 0 \]

The values for the rate constants are mentioned in tables 7, 8 and 9 of the main text.

6.5 Hypoexponential distribution PDF

Python code used to calculate the probability density function of a hypoexponential distribution.

```python
import numpy as np
```

51
from scipy import linalg

def hypoexp_pdf(x, rates_list):
    n_rates = len(rates_list)

    alpha = np.array([1] + [0]*(n_rates-1))
    theta = np.zeros((n_rates,n_rates))
    for i in range(n_rates-1):
        rate = rates_list[i]
        theta[i][i] = -rate
        theta[i][i+1] = rate
    theta[-1][-1] = -rates_list[-1]

    exp_xtheta = linalg.expm(x*theta)

    column_vector = np.ones((n_rates,1))
    sumtheta = theta.dot(column_vector)

    result = - alpha.dot(exp_xtheta.dot(sumtheta))
    return result[0]

6.6 Delay Method Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>smod.SetDelayParameter()</td>
<td>delay_distributions</td>
<td>Dictionary of reaction names or indices (key) with delay distributions and parameters (value). e.g. {&quot;R1&quot;: (&quot;gamma&quot;,5,5), &quot;R2&quot;: (&quot;fixed&quot;,2)}</td>
</tr>
<tr>
<td></td>
<td>nonconsuming_reactions</td>
<td>List of names or indices of nonconsuming delayed reactions.</td>
</tr>
<tr>
<td>smod.DoDelayedStochSim()</td>
<td>method</td>
<td>Either ‘DelayedDirect’ or ‘DelayedNextReactionMethod’</td>
</tr>
<tr>
<td></td>
<td><em>parameters of DoStochSim()</em></td>
<td></td>
</tr>
</tbody>
</table>

Table S1: Functions to used for the Delayed methods with their parameters.

6.7 Single Molecule Method Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>smod.SetPutativeReactionTimes()</td>
<td>distributions</td>
<td>Dictionary of reaction names or indices with putative reaction time distributions and parameters. e.g. &quot;R1&quot;: (&quot;gamma&quot;,5,5), &quot;R2&quot;: (&quot;exponential&quot;,2)</td>
</tr>
<tr>
<td>smod.DoSingleMoleculeStochSim()</td>
<td>method</td>
<td>‘fSMM’ or ‘SMM’</td>
</tr>
<tr>
<td></td>
<td><em>parameters of DoStochSim()</em></td>
<td></td>
</tr>
</tbody>
</table>

Table S2: Functions of the Single Molecule methods with their parameters.
### 6.8 Cell Division Module Functions

#### Stochastic Simulation

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmod.SetVolumeDependencies()</td>
<td>IsVolumeDependent</td>
<td>Boolean</td>
</tr>
<tr>
<td></td>
<td>VolumeDependencies</td>
<td>List of the order of reactions, determining the amount of volume dependency.</td>
</tr>
<tr>
<td></td>
<td>SpeciesExtracellular</td>
<td>List of names of extracellular species, which are not volume dependent.</td>
</tr>
<tr>
<td>cmod.SetInitialVolume()</td>
<td>initial_volume</td>
<td>Float, volume of the first daughter. To prevent errors, this value should be smaller than the smallest possible mother cell (see ‘phi’).</td>
</tr>
<tr>
<td>cmod.SetGrowthFunction()</td>
<td>growth_rate</td>
<td>Type of growth, ‘linear’ or ‘exponential’</td>
</tr>
<tr>
<td></td>
<td>growth_type</td>
<td></td>
</tr>
<tr>
<td>cmod.SetVolumeDistributions()</td>
<td>phi</td>
<td>Tuple of distribution and parameters, specifying the mother end volume.</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>Tuple of distribution and parameters, specifying the</td>
</tr>
<tr>
<td></td>
<td>phi_beta_mean</td>
<td>Float. If ‘phi’ is a beta distribution, then phi_beta_mean is the mean of the distribution</td>
</tr>
<tr>
<td>cmod.SetNonDividingSpecies()</td>
<td>species</td>
<td>List of the names of species that do not divide upon cell division.</td>
</tr>
<tr>
<td>cmod.SetExactDividingSpecies()</td>
<td>species</td>
<td>List of the names of species that divide equally upon division, e.g.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chromosomes.</td>
</tr>
<tr>
<td>cmod.DoCellDivisionStochSim()</td>
<td>mode</td>
<td>Besides the options ‘time’, ‘steps’ also ‘generations’ to simulate a certain amount of generations to be taken.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmod.PlotSpeciesTimeSeries()</td>
<td>plottype</td>
<td>Either ‘amounts’ or ‘concentrations’ to plot respectively species copy numbers or species concentration.</td>
</tr>
<tr>
<td>cmod.PlotVolumeTimeSeries()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmod.PlotSpeciesVolumeTimeSeries()</td>
<td></td>
<td>Plot of both the species and volume time series, in one window.</td>
</tr>
<tr>
<td>cmod.PlotVolumeMotherDistribution()</td>
<td></td>
<td>Histogram of volume of mother cells.</td>
</tr>
<tr>
<td>cmod.PlotVolumeDaughterDistribution()</td>
<td></td>
<td>Histogram of volume of daughter cells.</td>
</tr>
<tr>
<td>cmod.PlotSpeciesMotherDistribution()</td>
<td></td>
<td>Histogram of species of mother cells.</td>
</tr>
<tr>
<td>cmod.PlotSpeciesDaughterDistribution()</td>
<td></td>
<td>Histogram of species of daughter cells.</td>
</tr>
<tr>
<td>cmod.InterdivisionTimeDistribution()</td>
<td></td>
<td>Histogram of the inter division times.</td>
</tr>
<tr>
<td>cmod.CalculateSpecificGrowthRate()</td>
<td></td>
<td>Calculates the population growth rate from the given volume growth rate and distributions.</td>
</tr>
<tr>
<td>cmod.AnalyzeExtantCells()</td>
<td></td>
<td>Calculates the extant cell population from a simulation.</td>
</tr>
<tr>
<td>cmod.PlotExtantSpeciesDistribution()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table S3: Functions of the Cell Division module with their parameters.

### Plotting and Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmod.PlotSpeciesTimeSeries()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmod.PlotVolumeTimeSeries()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmod.PlotSpeciesVolumeTimeSeries()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmod.PlotVolumeMotherDistribution()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmod.PlotVolumeDaughterDistribution()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmod.PlotSpeciesMotherDistribution()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmod.PlotSpeciesDaughterDistribution()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmod.InterdivisionTimeDistribution()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmod.CalculateSpecificGrowthRate()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmod.AnalyzeExtantCells()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmod.PlotExtantSpeciesDistribution()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table S4: Plotting and analysis functions of the Cell Division module.
6.9 Two-component signaling model

<table>
<thead>
<tr>
<th>Reaction</th>
<th>Distribution</th>
<th>Mean</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>R transcription</td>
<td>gamma(0.018,672)</td>
<td>12.096</td>
<td>8128.512</td>
</tr>
<tr>
<td>S transcription</td>
<td>gamma(0.018,1461)</td>
<td>26.298</td>
<td>38421.378</td>
</tr>
<tr>
<td>R translation</td>
<td>gamma(0.059,223)</td>
<td>13.157</td>
<td>2934.011</td>
</tr>
<tr>
<td>S translation</td>
<td>gamma(0.059,486)</td>
<td>28.674</td>
<td>13935.564</td>
</tr>
<tr>
<td>Rp2</td>
<td>gamma(0.018,672)</td>
<td>12.096</td>
<td>8128.512</td>
</tr>
</tbody>
</table>

Table S5: An overview of all reactions that were modeled with a delay. In all cases, a gamma-distribution was used; values shape and scale parameters are given in brackets.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial cell volume</td>
<td>1</td>
</tr>
<tr>
<td>Volume Growth type</td>
<td>Exponential</td>
</tr>
<tr>
<td>Volume Growth rate</td>
<td>0.00025 s(^{-1})</td>
</tr>
<tr>
<td>(\phi) (mother cell volume)</td>
<td>Beta(2,2) + 2</td>
</tr>
<tr>
<td>(K) (partition distribution)</td>
<td>Beta(5,5)</td>
</tr>
<tr>
<td>Non-dividing species</td>
<td>DNA</td>
</tr>
</tbody>
</table>

Table S6: An overview of the settings used in the two-component cell division simulations. Note that an exponential growth rate of 0.00025 s\(^{-1}\) corresponds to a doubling time of about 46 minutes.