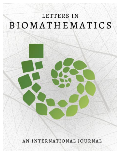

LETTERS IN BIOMATHEMATICS
An International Journal

Intercollegiate Biomathematics Alliance

RESEARCH ARTICLE

OPEN ACCESS

GillesPy2: A Biochemical Modeling Framework for Simulation Driven Biological Discovery

Sean Matthew[a], Fin Carter[b], Joshua Cooper[b], Matthew Dippel[b], Ethan Green[b], Samuel Hodges[c], Mason Kidwell[b], Dalton Nickerson[b], Bryan Rumsey[a], Jesse Reeve[b], Linda R. Petzold[d], Kevin R. Sanft[b], Brian Drawert[a]

[a]National Environmental Modeling and Analysis Center (NEMAC), University of North Carolina, Asheville, NC 28804; [b]Department of Computer Science, University of North Carolina, Asheville, NC 28804; [c]Department of Computer Science, North Carolina State University, NC 27695; [d]Department of Computer Science and Department of Mechanical Engineering, University of California, Santa Barbara, CA 93106

ABSTRACT

Stochastic modeling has become an essential tool for studying biochemical reaction networks. There is a growing need for user-friendly and feature-complete software for model design and simulation. To address this need, we present GillesPy2, an open-source framework for building and simulating mathematical and biochemical models. GillesPy2, a major upgrade from the original GillesPy package, is now a stand-alone Python 3 package. GillesPy2 offers an intuitive interface for robust and reproducible model creation, facilitating rapid and iterative development. In addition to expediting the model creation process, GillesPy2 offers efficient algorithms to simulate stochastic, deterministic, and hybrid stochastic-deterministic models.

ARTICLE HISTORY
Received October 15, 2022
Accepted April 28, 2023

KEYWORDS
Simulation, Modeling, Stochastic, Hybrid

1 Introduction

In 1976, Dan Gillespie first presented the Stochastic Simulation Algorithm (SSA) (Gillespie, 1976, 1977), which allows for efficient and accurate simulation of discrete stochastic reaction systems. This method has gained wide popularity in the simulation of complex biological and biochemical systems, and is widely used in the field of computational systems biology. It has inspired a plethora of software packages which implement this method, and a family of algorithms that enhance and extend it. Our team has been part of the development of many of these algorithms and related software. In particular, we developed StochKit2 (Sanft et al., 2011) and later a Python 2 wrapper GillesPy (Abel et al., 2016), which we named in honor of Dan Gillespie. We have continued our algorithm and software development with the next generation of simulation software, GillesPy2, which has been completely rewritten and is stand-alone. Our design goals are to provide software that is easy to use by novices but powerful enough in performance and features for even the most advanced users. GillesPy2 is distributed and used as a Python 3 (Van Rossum and Drake, 2009) package, and the computationally expensive solvers are written in C++ (for maximum performance) and Python (for maximum compatibility). It provides simulation interfaces for stochastic and deterministic systems, as well as a novel hybrid method which can switch automatically between them. GillesPy2 aims to be a modeling and simulation package that can provide everything needed for building and solving any spatially homogeneous biochemical reaction system.

Stochastic modeling and simulation has become a powerful and impactful tool for the study of mathematical, biochemical, and biological systems (Gillespie et al., 2013; El Samad et al., 2005; Elowitz et al., 2002). In addition, it has made an impact in the fields of conservation ecology (Drawert et al., 2017, 2022) and epidemiology (Jiang et al., 2021; Drawert et al., 2017). The process of understanding such systems via modeling and simulation is iterative, akin to the scientific method: starting with observations and data, a *user* (modeler) first develops a mathematical model. Most biological and similar dynamic models, such as many *ordinary differential equation* (ODE) models, do not have analytical solutions and, therefore, are "solved" by simulation. These simulations are a form of experimentation that produce results and data that the modeler can compare to the original observations and data to further refine the model. This iterative process is captured in the schematic shown in Figure 1(A). Software is an essential tool in this workflow. The software needs to be simple to use, computationally efficient, and it should provide features to accommodate studying a wide variety of systems. For example, Figure 1 shows simulation results from a discrete stochastic model (B), a model with *events* (C), and a model with coupled discrete stochastic and continuous components (D). GillesPy2 provides an integrated Python 3 platform that is designed to meet the needs of modelers. A brief summary of the GillesPy2 capabilities is shown in the boxes around the schematic in Figure 1(A).

A number of software packages are kindred to GillesPy2. These include COPASI (Hoops et al., 2006), GillespieSSA (Pineda-Krch, 2008), StochPy (Maarleveld et al., 2013), PySB (Lopez and Garbett, 2014), Biosimulator.jl (Landeros et al., 2018), BioNetGen (Harris et al., 2016), Tellurium and libRoadRunner (Choi et al., 2018; Somogyi et al., 2015), and others. While a complete feature review is beyond the scope of this paper, each of these packages provides mechanisms for defining or importing models and/or simulating models within a particular programming language, integrated development environment, or graphical user interface. For example, COPASI features a graphical user interface that allows users to define models with multiple compartments using a comprehensive set of rate laws. GillespieSSA and Biosimulator.jl allow users to build and simulate biochemical models in the R and Julia languages, respectively. PySB implements rule-based modeling and BioNetGen implements structure-based modeling in Python. StochPy and Tellurium and libRoadRunner provide text-based interfaces for defining biochemical models that can be simulated and visualized in Python. GillesPy2 aims to provide a comprehensive and flexible collection of simulation methods in an easy-to-use package. It provides an intuitive *object-oriented* Application Programming Interface (API) for the development of well-mixed biochemical reaction models that can include stochastic, deterministic, or coupled components. GillesPy2 is designed to integrate with Jupyter notebooks. GillesPy2 is also the computational engine for our powerful web-based IDE, StochSS (Drawert et al., 2016).

The remaining sections of this manuscript are structured as follows. In Section 2 we discuss biological modeling and various methods of simulation. In Section 3 we present the Tau-Hybrid Simulation Algorithm. Section 4 describes how GillesPy2 was designed and implemented, including details on our novel hybrid simulation algorithm. Section 5 provides several examples of building and simulating models with GillesPy2. In Section 6, we illustrate the performance of GillesPy2 and how it fits into the software ecosystem of StochSS (Drawert et al., 2016). We end with our conclusions in Section 7.

2 Background

Biological systems feature dynamics that operate on a range of spatial and temporal scales. This leads to many different types of mathematical models, each with their own modeling assumptions and approximations. Furthermore, most models require software to simulate or "solve" them, and it can be difficult to separate model development from simulation and software. For example, creating an ODE model implicitly makes assumptions about the continuity of the state variables. In this section we describe the classes of models and the simulation algorithms that are utilized in GillesPy2.

2.1 Biological modeling

While there are many classes of biological models, here we consider models where the *system state* is a time-dependent vector of *populations*. For example, the populations could be numbers of molecules of different types in a biochemical simulation or populations of different species in an ecological model. The models' populations change in time due to two primary mechanisms: discrete events and differential equations.

To demonstrate these mechanisms, consider the *Michaelis-Menten reaction set* (Michaelis and Menten, 1913)

$$R_1 : A + B \xrightarrow{k_1} C \qquad R_2 : C \xrightarrow{k_2} A + B \qquad R_3 : C \xrightarrow{k_3} B + D. \tag{1}$$

Reaction set (1) describes the enzyme-catalyzed conversion of substrate "A" into product "D" via enzyme "B". The model state is the vector of the $N = 4$ species' populations $X(t) = [A(t), B(t), C(t), D(t)]$. The populations evolve via $M = 3$ *reactions*, denoted R_j. For example, R_1 describes the binding of a molecule of A and a molecule of B into the complex C. Here, k_i are *rate constants* that influence the rate at which the reactions are occurring. A complete model description requires values for the rate constants and initial populations defined at time $t = 0$. We assume that reactions occur instantaneously.

The reaction set in (1) can be be formulated using *reaction rate equations*, as follows:

$$\begin{aligned} \frac{dA}{dt} &= -k_1AB + k_2C & \qquad \frac{dC}{dt} &= k_1AB - k_2C - k_3C \\ \frac{dB}{dt} &= -k_1AB + k_2C + k_3C & \qquad \frac{dD}{dt} &= k_3C. \end{aligned} \tag{2}$$

The model in (2) is deterministic and treats the populations as continuous quantities. This ODE model is appropriate if the effect of stochastic fluctuations is relatively small, which will generally be the case if all of the species populations are sufficiently large.

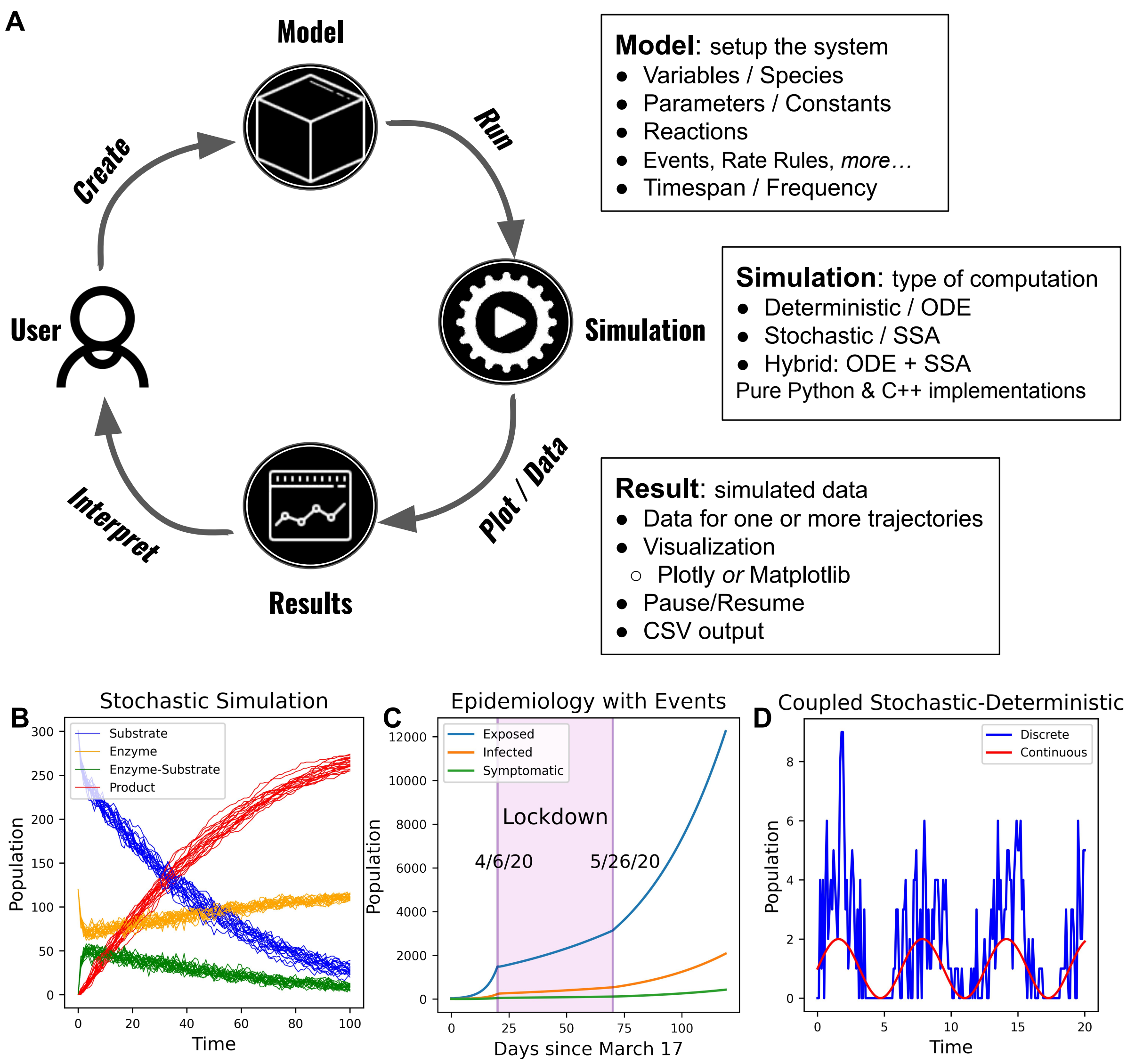

Figure 1: (A) Diagram representing the GillesPy2 workflow, and the three major components: Model, Simulation, and Result. (B) Simulation of the classic Michaelis-Menten model (Michaelis and Menten, 1913). (C) Simulation of Covid-19 infections in a population with lockdown and re-opening events. (D) Simulation of a discrete stochastic birth-death process coupled to a time-varying continuous variable. Visualizations of simulations were all performed with GillesPy2.

Alternatively, the reaction set in (1) could be formulated as a discrete stochastic system, where the populations are discrete and the reactions are random events. In the discrete stochastic representation, the evolution of the system state is probabilistic and can be described by the *Chemical Master Equation* (CME) (Gillespie et al., 2013):

$$\frac{\partial P(X,t)}{\partial t} = \sum_{j=1}^{M} \left[a_j(X-\nu_j)P(X-\nu_j,t \mid X_0,t_0) - a_j(X)P(X,t \mid X_0,t_0) \right], \tag{3}$$

where ν_j is the *stoichiometry vector* that describes how the state variable $X(t)$ changes when reaction R_j occurs (or *fires*), and a_j is the *propensity function* that describes the rate at which R_j is firing. The CME is huge. It contains one equation for each possible value of the population X. Thus, it cannot be solved directly for most models (Munsky and Khammash, 2006). Instead of solving the CME directly, the common alternative approach is to approximate the solution using Monte Carlo simulation.

The Systems Biology Markup Language (SBML) provides a way to describe many biological models such as the reaction set in (1) (Hucka et al., 2003). SBML uses *tags* to describe the *Species* and *Reactions* in a model. Reactions are comprised of *Reactants* (species that are consumed) and *Products* (species that are produced) and a *KineticLaw*, which may use *Parameters*, to describe differential equation or discrete stochastic state changes. SBML also defines other mechanisms such as *rate rules* and *events*. Rate rules describe additional differential equation state-change mechanisms, and events are state changes that happen when a condition known as a *trigger* occurs. For the purposes of this work, we will follow the SBML nomenclature and use the terms “reaction”, “rate rule”, and “event” to refer to the various state-change mechanisms in biological models.

We note that SBML includes several features that we will not consider. In particular, SBML defines *Compartment* components that define the region in which the species populations exist. Compartments can be used to describe spatially inhomogeneous models by tracking separate species populations within each compartment and by treating “diffusion” as a reaction that transfers populations between compartments. However, such spatial models can benefit from additional geometry information that SBML does not currently provide (however, an SBML spatial package is in development). Other modeling and simulation software is better suited to modeling systems that require detailed geometry information. In this work we consider only spatially homogeneous models.

2.2 Deterministic simulation

The theory for simulating ODE models is well established and mature (Ascher and Petzold, 1998). Numerically solving an initial value problem involves starting with the initial state, evaluating the derivatives, and repeatedly simulating over small, finite time intervals until an end time is reached. Algorithms exist that use dynamic step size selection to efficiently approximate the true solution. These algorithms can generally be categorized as *explicit* or *implicit*. Implicit methods perform better on models that exhibit *stiffness*, which arises in models that contain multiple timescales, as is common in biological systems. There are algorithms and software that can detect the presence of stiffness and choose an appropriate algorithm automatically (Petzold, 1983).

GillesPy2 uses the LSODA method (Petzold, 1983) for deterministic simulation. We use the implementation from the SciPy Python package (Virtanen et al., 2020) for the Python solvers, and the implementation from the SUNDIALS package (Hindmarsh et al., 2005) for the solvers implemented in C++. In addition, the Python solvers provide options for selecting other methods implemented by the SciPy package, which we recommend only for advanced users.

2.3 Stochastic simulation

Gillespie’s Stochastic Simulation Algorithm (SSA) generates statistically exact samples from the CME (3) (Gillespie et al., 2013). However, there are many variants of the SSA. The assumptions underlying the CME, namely the Markovian assumption and the spatially homogeneous assumption, require that each reaction type fires according to an exponential distribution. The original SSA *direct method* was derived naturally from the joint density:

$$Pr(\tau, j \mid X, t) = a_j(X) e^{-\sum_{i=1}^{M} a_i(X)\tau}, \tag{4}$$

where $Pr(\tau, j \mid X, t)$ is the probability that reaction R_j will fire in the infinitesimal interval $[t+\tau, t+\tau+dt)$. Any algorithm that produces samples of the next reaction time τ and next reaction index j from Eq. (4) can be used to generate samples of the CME. Therefore, the SSA is better viewed as a family of algorithms. Many algorithms have been proposed that use various data structures to achieve different performance and scaling properties (Gibson and Bruck, 2000; Slepoy et al., 2008; Ramaswamy et al., 2009; Mauch and Stalzer, 2011; Sanft and Othmer, 2015).

It is important to note that the time step size is not an algorithm parameter in the SSA. However, if the values of the propensities do not change much in a time step $\tau = \Delta t$, then the number of firings of reaction R_j can be well approximated as a Poisson random variable with mean $a_j(X)\tau$. This approximation is the basis for the *Tau-Leaping* simulation algorithm:

$$X(t+\tau) = X(t) + \sum_{j=1}^{M} Poisson\big(a_j(X)\tau\big)\nu_j, \tag{5}$$

where each *Poisson* is an independent Poisson random number. In practice, the step size τ is determined adaptively to control the error in the simulation (Gillespie, 1977; Cao et al., 2006).

When the populations are sufficiently large such that the mean of all the *Poisson* random variables in (5) is large, then the Poisson random variables can be approximated as Normal random variables. If one makes this approximation and replaces the finite step size τ with the infinitesimal dt, the resulting algorithm is

$$X(t+dt) - X(t) = \sum_{j=1}^{M} \mathcal{N}\Big(\mu = a_j(X)dt,\ \sigma = \sqrt{a_j(X)dt}\,\Big)\nu_j, \tag{6}$$

where each $\mathcal{N}(\mu, \sigma)$ is an independent Gaussian random variate (Gillespie et al., 2013). Equation (6) is a stochastic differential equation known as the *Chemical Langevin Equation* (CLE). The CLE is often written as:

$$dX = \nu a(X)dt + \nu\sqrt{a(X)}\mathcal{N}_M(\mu = 0,\ \sigma = I)\sqrt{dt}, \tag{7}$$

where ν is the *stoichiometric matrix*, $a(X)$ is the vector of propensities, and $\mathcal{N}_M$ is an M-dimensional normal random variable. In moving from Tau-Leaping to the CLE, the state X transitions from a discrete population to a continuous population. It is interesting to note that as the population X increases, the noise term becomes negligible and the CLE (7) approaches the reaction rate equation ODE, hence the ODE model arises in the large population (thermodynamic) limit of the discrete stochastic model (Kurtz, 1972).

2.4 Hybrid simulation

In this context we define hybrid modeling and simulation to be the simulation of a reaction system that includes both deterministic and stochastic components. This involves partitioning a model into two subsystems, where one is simulated deterministically and the other is simulated stochastically (Pahle, 2009; Helms et al., 2018). The partitioning can be based on the populations (i.e., large and small) or the reaction rates (i.e., fast and slow). Reactions that involve only large population species may be approximated as continuous, while reactions that fire less often or involve small population species can be simulated via an SSA variant or Tau-Leaping. With a hybrid model, one may gain computational performance while still capturing the stochasticity of the system (Ahmadian et al., 2017). On the other hand, hybrid simulation introduces computational overhead to maintain additional data structures, in which case the performance will typically be slower than specialized solvers for models that do not include both deterministic and stochastic components. In these situations, the benefit of hybrid simulation is to allow modelers more flexibility when defining their models by specifying whether a species should be simulated as only continuous, only discrete, or be able to switch between the two.

3 The Tau-Hybrid Simulation Algorithm

Here, we introduce our novel Tau-Hybrid simulation algorithm, which is a hybrid of ODE deterministic simulation of continuous species and the Tau-Leaping (Gillespie, 2001) method for stochastic simulation of discrete species. The Tau-Hybrid algorithm features two methods for combining continuous and discrete simulation: the coupling of discrete and continuous species in a reaction network, and the automatic switching of a species from discrete to continuous (and vice versa). It is important to note that in the case where all of the propensities of the system are dependent only on discrete state variables (no time-dependent propensities, or continuous species), this method is statistically equivalent to Tau-Leaping (note, it is not the same method).

3.1 Simulation of a coupled discrete-continuous system

We will first discuss the combined simulation method for coupled discrete and continuous species in a reaction network. We partition the reactions and species of the system. Reactions are either part of the *deterministic* (*det*) or *stochastic* (*stoch*) sets, species S are either *continuous* ($S_c \in \mathbb{R}_0^+$) or *discrete* ($S_d \in \mathbb{Z}_0^+$). We evolve the state of the system $X(t) = [S_1, ..., S_N]$ forward by continuously integrating the *deterministic* reactions and firing discontinuous jump events for the *stochastic* reactions. Reactions are *deterministic* if they consume and produce only *continuous* species, otherwise they are *stochastic*. The partitioning of the species as *discrete* or *continuous* can be done *a priori* by the modeler or automatically by the method (see next section).

We define the time evolution of the system due to the *deterministic* reactions as

$$\frac{d}{dt} S_i = \sum_{j}^{j \in det} \nu_{ij}\, a_j\big(X(t), t\big) \tag{8}$$

for all *continuous* species i. Note that in this formulation we are allowing propensity functions to be explicitly dependent on time. For *stochastic* reactions, we need to find the firing time t_j for all reactions $j \in stoch$. We find this using the integral form of the next-reaction SSA as described in (Salis and Kaznessis, 2005):

$$\int_{t_0}^{t_0+t_j} a_j\big(X(t'), t'\big)\, dt' + \log(URN) = 0, \tag{9}$$

where t_0 is the current time and URN is an independent uniform random number $URN \in (0, 1)$. Note that in the case where $a_j(X, t)$ is constant over the interval $[t_0, t_0 + t_j]$ (i.e., discrete stochastic systems), this is simplified to: $t_j = \frac{-\log(URN)}{a_j(X)} + t_0$, which is the time to the next reaction in the SSA method (Gillespie, 1976). Following the method in (Salis and Kaznessis, 2005) for solving (9), we define an indicator variable r_j for each reaction $j \in stoch$. We forward integrate these variables to find the time t_j at which $r_j = 0$, which is the time at which the reaction fires. We do this by converting (9) to a differential equation initialized by an independent uniform random number:

$$\frac{d}{dt} r_j = a_j(X, t) \qquad r_j \big|_{t=t_0} = \log(URN). \tag{10}$$

Note r_j will always start negative since $log(URN) < 0$, and r_j is monotonically increasing since $a_j >= 0$. The time at which r_j crosses zero is then a statistically exact sample of the firing time of reaction j for a jump process with time-varying intensity $a_j(X, t)$. If the numerical method used is an ODE solver to integrate (8) and (10) simultaneously, and using a root-finder to determine the reaction event times, we then have the hybrid ODE-SSA method as described in (Salis and Kaznessis, 2005).

However, integrating (8) and (10) with root-finding to find the reaction event times is quite computationally expensive. Following the idea from Tau-Leaping that tolerating a small amount of error can result in a large computational gain, instead of using root finding to determine the time of each reaction event, we take larger simulation steps by aggregating multiple firings of each *stochastic* reaction channel into a single event. All *stochastic* reaction channel firing events are processed at the end of each time step τ. Over the time period [t, t+τ], *deterministic* reactions evolve the system using (8), and the reaction indicator variables r_j evolve using (10) (however, the r_j variables may now have positive values). At the end of each time step τ we examine each r_j. If it is positive, we determine the number of reaction events that have fired in the time step by finding the smallest number n such that

$$\arg\min_{n} \left(r_j + \sum_{1}^{n} \log(URN) \right) < 0, \tag{11}$$

i.e., the sum of the log of n samples of URN until the value of r_j becomes negative. Since the initial condition (10) after each firing is $\log(URN)$, and we assume that $a_j(X, t)$ is constant over [t, t+τ], Eq. (11) can be efficiently implemented using the Poisson distribution:

$$n = 1 + Poisson(r_j), \tag{12}$$

for all reaction indicators r_j that are positive at the end of an integration step. Once we know that reaction j has fired n times, we update the state: $X \mathrel{+}= n\, \nu_j$, and then re-initialize $r_j = log(URN)$. In this way, the simulation continues time-discontinuously, integrating (8) and (10) from t to $t + \tau$ and using (12) to update the state of the system at the end of the step. We call this hybrid Tau-Leaping/ODE simulation method the Tau-Hybrid Algorithm.

Figure 2 shows a visual representation of a simulation using the Tau-Hybrid algorithm. The populations of the two species are shown, along with the values of the reaction indicator variables. Note that the value of r_3 moves in straight lines between the discontinuous jumps that arise due to stochastic reactions firing. This is due to the propensity function being constant. In contrast, the value of r_1 moves in curved lines, with a positive second derivative. This is because the propensity function $a_1(X)$ depends on the population of A, which is continuously increasing. In this example, the value of r_2 is ignored, as it is a deterministic reaction. Thus, the value of A changes continuously using the ODE integration over the step $[t, t + \tau]$, and then discontinuously (discretely) at the end of each step.

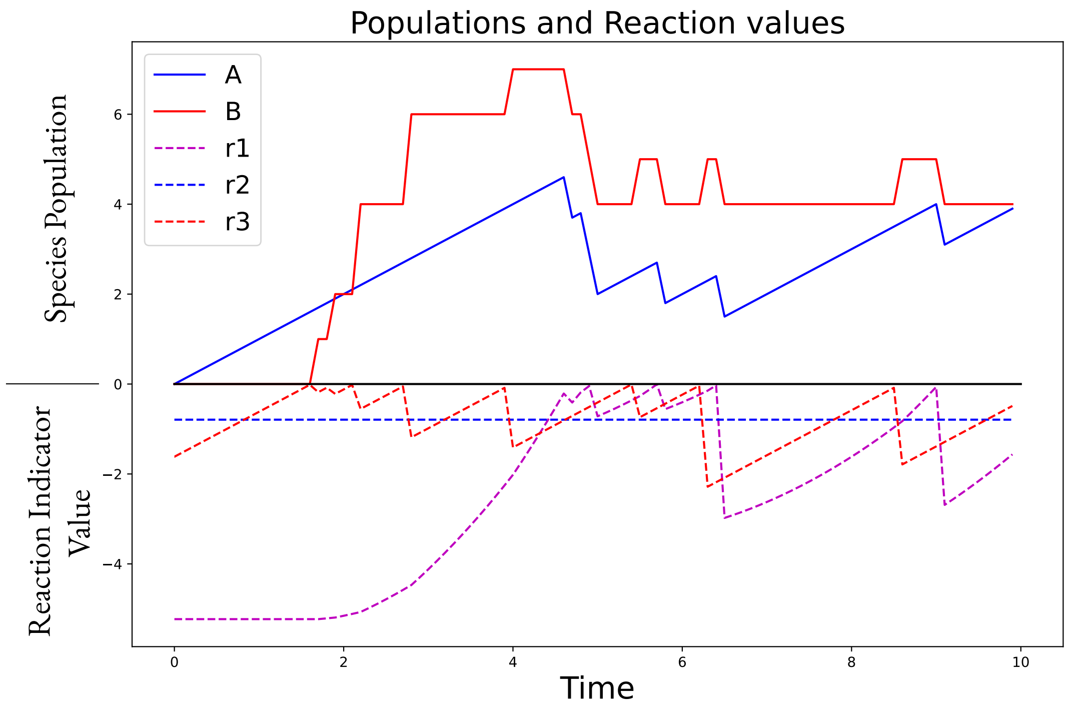

Figure 2: Visualization of the Tau-Hybrid simulation of a coupled discrete-continuous system. The model features three reactions: $R_1\colon A + B \longrightarrow \emptyset$, $R_2\colon \emptyset \longrightarrow A$, $R_3\colon \emptyset \longrightarrow B$. The values of the reaction indicator variables, r_1, r_2, and r_3 are shown along with the populations of species A, which is set to be continuous, and species B, which is set to be discrete. The population of B changes discretely when r_1 or r_3 fire, which occurs when their value crosses the $y = 0$ axis and becomes positive. We then use (11) to determine the number of times the reaction is fired and then reset the reaction indicator value to $\log(URN)$ (a negative value). The reaction R_2 is simulated as deterministic because it depends only on species marked as *continuous*. Therefore, the indicator variable r_2 is ignored and the population of A changes continuously. The population of A also changes discretely when R_1 fires. The simulation τ step size is calculated adaptively, and the state of the simulation is sampled for output every 0.1 time units. All other options use the defaults (e.g., LSODA integrator).

3.2 Automatic switching between discrete and continuous species

Gillespie described how the different simulation regimes (e.g., stochastic vs. deterministic) relate to each other, and the assumptions at each level in Gillespie (2009). Using this as a foundation, we make the observation that for many discrete stochastic systems, it is important to keep track of the exact stochastic variations when the populations are relatively low. However, when the population of a discrete species is high, and the stochastic variation from the mean field trajectory is low, we can utilize the law of large number assumption and transition the discrete species into a continuous species with minimal loss of accuracy in the solution.

The Tau-Hybrid algorithm has two different methods for detecting when a species should be switched between discrete and continuous. The first is a simple population threshold value, where above this threshold the species is considered continuous and below the threshold the species is considered discrete. The second method uses the relative noise in the species simulation trajectory to determine whether the stochastic variation about the mean is more or less than an error tolerance. For this calculation we use an approximation of the coefficient of variation $\mathrm{CV} = \sigma/\mu$, where σ and μ are the standard deviation and mean value respectively. Related auxiliary quantities $\hat{\sigma}$ and $\hat{\mu}$ are approximated as part of the τ step size selection step outlined above, and as defined in (Cao et al., 2006), thus we use

$$\hat{\mu}_i = -\sum_j^M \sum_k^N a_j(X) * \mathrm{Reactants}(k,j) + \sum_j^M \sum_k^N a_j(X) * \mathrm{Products}(k,j) \tag{13}$$

$$\hat{\sigma}_i^2 = \sum_j^M \sum_k^N a_j(X) * \mathrm{Reactants}(k,j)^2 + \sum_j^M \sum_k^N a_j(X) * \mathrm{Products}(k,j)^2 \tag{14}$$

$$\widehat{CV}_i = \sqrt{\hat{\sigma}_i^2} / (S_i + \hat{\mu}_i) \tag{15}$$

where Reactants(k,j) is the number of species k that are consumed by reaction j (i.e., 2 for a bimolecular reaction, 0 if not involved). Similarly, Products(k,j) is the number of species k that are produced by reaction j. If $\widehat{CV}_i$ for species S_i is less than the specified per-species tolerance, we mark the species *continuous*. We then examine each reaction j. If all of the reactants and products are marked as *continuous* then the reaction is marked *deterministic*, otherwise it is marked *stochastic*.

We also use the values from (13) and (14) in our implementation of the τ step size selection method (Cao et al., 2006)

$$\tau = \min_{i \in Reactants} \left(\frac{\max(\varepsilon_i S_i, 1)}{|\hat{\mu}_i|}, \frac{\max(\varepsilon_i S_i, 1)^2}{\hat{\sigma}_i^2} \right) \tag{16}$$

where *Reactants* is all species k where Reactants$(k,j) > 0$ for any reaction j, and ε is the error control parameter. We refer the reader to (Cao et al., 2006) for the complete derivation.

In the implementation of the τ step size selection method (Cao et al., 2006), to prevent species from going negative it is sometimes necessary to take a single SSA step (Step 9 in Algorithm 1). In our implementation, we check to see if any species are negative after all stochastic reactions have been processed for a given step. If a negative state is detected, then the system is restored to the state prior to the step and a new τ and the reaction j' to fire is determined using the next-reaction SSA methodology by calculating

$$\tau_j = \frac{-r_j(t)}{a_j(X(t), t)} \tag{17}$$

$$j' = \arg\min_j (\tau_j) \tag{18}$$

for all $j \in stoch$. Using (18) we find j', integrate the ODE system over $[t, t + \tau_{j'}]$, then fire a single j' reaction. A complete description of the methods is given in Algorithm 1.

Algorithm 1: Hybrid Tau-Leaping algorithm (hybrid of ODE and Tau-Leaping)

```
1:  Initialize state $X(t = 0)$ and each $r_j = \log(URN)$
2:  while current time < simulation end time do
3:      Evaluate propensities $a_j(X)$ at current state.
4:      Find $\hat{\mu}_i$, $\hat{\sigma}_i$ & $\widehat{CV}_i$ for each species as in Eqs. (13)–(15).
5:      Select the step size $\tau$ according to Eq. (16).
6:      Identify continuous/discrete species and deterministic/stochastic reaction sets.
7:      Integrate stochastic reaction indicator variables $r_j$, and deterministic species from $t$ to $t + \tau$ using Eqs. (8) and (10).
8:      Update state based on number of stochastic reaction firings using Eq. (12).
9:      if no species have a negative population then
10:         Store state $X(t + \tau) = X$, $t = t + \tau$
11:     else
12:         Reset state $X = X(t)$, the state at the beginning of the step.
13:         Using (17) and (18), Update state by firing a single $j'$ reaction: $X(t + \tau_{j'}) = \nu_{j'} + X(t)$, and set $t = t + \tau_{j'}$.
14:     end if
15: end while
```

4 GillesPy2 Design and Implementation

GillesPy2 is a Python 3 package that uses an *object-oriented* approach that provides an intuitive API for working with models, simulations, and results. In object oriented programming, *classes* describe the components of an *object*. GillesPy2 users create and manipulate objects via well documented *methods* (or "functions"). The primary classes in GillesPy2 are the Model, Solver, and Results classes, which are described in the subsections below.

4.1 Model class

A Model object consists of three primary components: *species*, *parameters*, and *reactions*. A GillesPy2 user can build a model by creating Species, Parameter, and Reaction objects and adding them to a Model object. This process is best described via a concrete example.

Consider the Michaelis-Menten reaction set from Eq. (1). A user can create the Species objects with the following simple Python 3 code snippet:

```
A = Species(name="A", initial_value=301)
B = Species(name="B", initial_value=120)
C = Species(name="C", initial_value=0)
D = Species(name="D", initial_value=0)
```

Similarly, one can define Parameters:

```
rate1 = Parameter(name="rate1", expression=0.0017)
rate2 = Parameter(name="rate2", expression=0.5)
rate3 = Parameter(name="rate3", expression=0.1)
```

Species and Parameters are used to build the Reaction objects:

```
r1 = Reaction(name="r1", reactants={A: 1, B: 1}, products={C: 1}, rate=rate1)
r2 = Reaction(name="r2", reactants={C: 1}, products={A: 1, B: 1}, rate=rate2)
r3 = Reaction(name="r3", reactants={C: 1}, products={B: 1, D: 1}, rate=rate3)
```

A user can then instantiate a Model object and add the Species, Parameters, and Reactions components to it:

```
michaelis_menten = Model()
michaelis_menten.add_species([A, B, C, D])
michaelis_menten.add_parameter([rate1, rate2, rate3])
michaelis_menten.add_reaction([r1, r2, r3])
```

Finally, to facilitate simulation, we define a TimeSpan that specifies the time points at which to keep data from a simulation:

```
michaelis_menten.timespan(TimeSpan(numpy.linspace(0, 100, 101)))
```

GillesPy2 models can include several additional features of the SBML Level 3 standard (Hucka et al., 2003), including rate rules, assignment rules, function definitions, and events.

4.2 Solvers

The base class for all GillesPy2 simulation algorithms is GillesPySolver, which provides an abstract "run" method that all subclasses ("Solvers") must implement. However, GillesPy2 users are not required to have knowledge of particular Solvers or algorithms in order to run a simulation. GillesPy2 intelligently uses the properties of the Model and the user's system to automatically select the appropriate Solver. Therefore, running a simulation of the Michaelis-Menten model defined above is as simple as:

```
results = michaelis_menten.run()
```

The above code will run a single stochastic simulation (or *trajectory*) using the SSA. To run an *ensemble* of several trajectories the code is:

```
results = michaelis_menten.run(number_of_trajectories=20)
```

The above code will generate a Results object containing simulation data similar to that shown in Figure 1(B).

GillesPy2 includes Solver classes for the SSA, Tau-Leaping, ODE, CLE, and Tau-Hybrid algorithms. Specifically, the SSA uses the *direct method* (Gillespie, 1977), Tau-Leaping uses the adaptive step-size algorithm from (Cao et al., 2006), ODE uses Sundials (Hindmarsh et al., 2005) or SciPy.integrate's ode function (Virtanen et al., 2020), CLE uses an Euler-Maruyama method (McCauley, 2013) with dynamic step size chosen as in the Tau-Leaping method, and the Tau-Hybrid is implemented as described in Section 3. The default algorithm when a user calls the run() method will be the SSA unless the model includes Species designated as "dynamic" or "continuous" or if the model contains one or more of the following SBML features: rate rules, events, assignment rules, or function definitions, in which case the selected algorithm will be the Tau-Hybrid. The user can also manually select an algorithm, for example:

```
results = michaelis_menten.run(algorithm="Tau-Leaping")
```

Each algorithm (except the CLE) is implemented in two Solver classes, one written in Python and one written in C++. The rationale is that code written in C++, a compiled language, is faster than Python code. However, running C++ code requires a compiler. Therefore, GillesPy2, when selecting the particular Solver for a given algorithm, checks for an available C++ compiler and uses the C++ Solver if a compiler is accessible, otherwise it uses the Python implementation. A list of the available Solvers is shown in Table 1.

Table 1: List of available Solvers in GillesPy2 as of version 1.7.0. *The ODE solvers are wrappers to SciPy (Virtanen et al., 2020) (Python) and SUNDIALS (Hindmarsh et al., 2005) (C++) solvers. The ODE solvers use LSODA by default but other integrators can be chosen.

Algorithm	Implementation / Reference	Class Name	Language	Advanced Features
SSA	Direct Method / Gillespie, 1977	NumPySSASolver	Python	
SSA	Direct Method / Gillespie, 1977	SSACSolver	C++	
Tau-Leaping	Cao et al., 2006	TauLeapingSolver	Python	
Tau-Leaping	Cao et al., 2006	TauLeapingCSolver	C++	
ODE	LSODA* / Petzold, 1983	ODESolver	Python*	
ODE	LSODA* / Petzold, 1983	ODECSolver	C++*	
CLE	Euler-Maruyama / McCauley, 2013	CLESolver	Python	
Tau-Hybrid	see Section 3	TauHybridSolver	Python	Event, RateRule, AssignmentRule, FunctionDefinition
Tau-Hybrid	see Section 3	TauHybridCSolver	C++	Event, RateRule

Users can override the default by instantiating a Solver object directly to have total control over the Solver. For example, a user with a C++ compiler could manually choose to use the Python SSA implementation to run an ensemble of the Michaelis-Menten model:

```
python_ssa =  NumPySSASolver(model=michaelis_menten)
results = python_ssa.run(number_of_trajectories=20)
```

4.3 Simulation output and visualization

The GillesPy2 Solvers' simulations return a Results object. A Results object contains a Python list of Trajectory objects, where each Trajectory contains a Python dictionary of the time array specified by the model's timespan object and an array of values corresponding to those time points for each Species in the Model. The Results object is convenient for conducting analysis directly in Python.

The Results class provides several methods to facilitate common analysis tasks, including plotting and generating descriptive statistics. To plot a Results object, the syntax is:

```
results.plot()
```

If the results variable contained an SSA ensemble from the Michaelis-Menten model, the above statement will generate a plot similar to that in Figure 1(B). The default plot method uses the Python Matplotlib (Hunter, 2007) library. Users can generate a Plotly (Plotly Technologies Inc., 2015) graph with the plotplotly method. The methods average_ensemble and stddev_ensemble create a Trajectory object containing the means and standard deviations, respectively. The methods plot_mean_stdev and plotplotly_mean_stdev compute these statistics and plot them in a single call. Finally, users can export their Results by writing them to a file using the to_csv method.

5 Examples

In the following examples, we will look at the process of creating, simulating, and visualizing models with GillesPy2. All of the examples demonstrate the core benefit of GillesPy2's object-oriented implementation: that of compartmentalization. Compartmentalization makes GillesPy2 easy to use by splitting the composition of a model into separate "building blocks". Section 5.1 demonstrates the core components of a GillesPy2 model, Section 5.2 demonstrates events using an epidemic model, and the photosynthesis model in Section 5.3 demonstrates GillesPy2's hybrid solver. All of the examples in this section are available in notebooks posted here: `https://github.com/GillesPy2/GillesPy2_paper`.

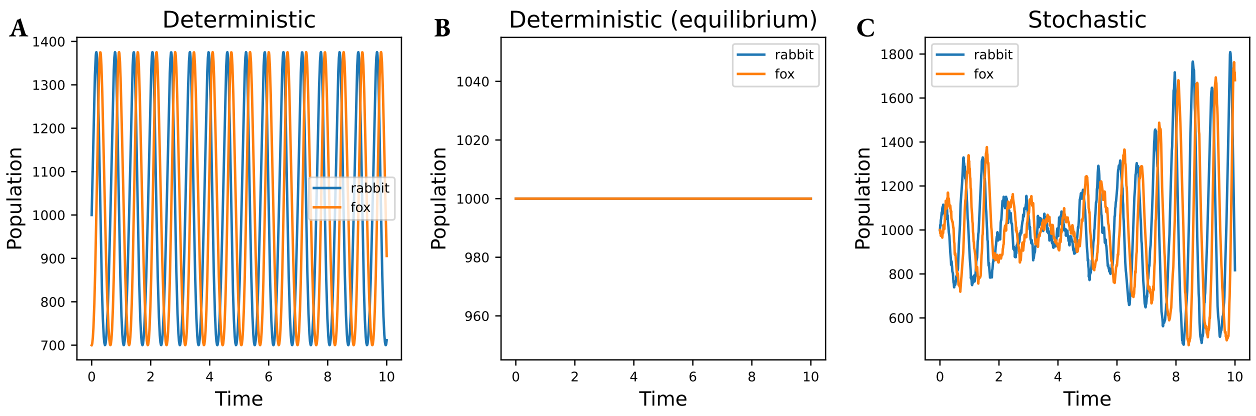

Figure 3: Simulations of the Predator-Prey model, as defined in (19) and (20). (A) deterministic (ODE) simulation from a non-equilibrium initial condition. (B) deterministic (ODE) simulation from the equilibrium initial condition. Note that the values never change. (C) stochastic simulation from the ODE equilibrium initial condition.

5.1 Predator-prey model

Lotka-Volterra models are a class of predator prey models often represented as ODEs (Lotka, 1920; Volterra, 1926). For example, the following set of ODEs can be used to described the population dynamics of a predator F and prey R (e.g., foxes and rabbits):

$$\frac{dF}{dt} = k_2FR - k_3F \qquad \frac{dR}{dt} = -k_2FR + k_1. \tag{19}$$

These ODEs can be converted into mechanistic reaction format:

$$R_1 : \emptyset \xrightarrow{k_1} R \qquad R_2 : F + R \xrightarrow{k_2} 2F \qquad R_3 : F \xrightarrow{k_3} \emptyset, \tag{20}$$

where the empty set symbol $\emptyset$ is used when there are no reactants or products. In GillesPy2, these reactions are described with the following code:

```
R1 = Reaction(reactants={}, products={R: 1}, rate=k1)
R2 = Reaction(reactants={R: 1, F: 1}, products={F: 2}, rate=k2)
R3 = Reaction(reactants={F: 1}, products={}, rate=k3)
```

For the parameter set $k_1 = 10$, $k_2 = 0.01$, $k_3 = 10$, the ODE model has an equilibrium point at $F = 1000$, $R = 1000$ (see Figure 3(B)). For non-zero populations away from the equilibrium point, the ODE system exhibits oscillations, as seen in Figure 3(A). Simulated stochastically, an initial population at the ODE equilibrium ($F = 1000$, $R = 1000$) does not remain constant, as stochasticity pushes the system away from equilibrium and the system exhibits noisy oscillations as seen in Figure 3(C).

5.2 Epidemic model with events

Next we consider a variation of the standard *SEIR* epidemic model inspired by the early Covid-19 pandemic (Jiang et al., 2021). The species S, E, and R correspond to susceptible, exposed, and recovered individuals, respectively. The total infected population is split into two categories: I_1 and I_2 corresponding to asymptomatic and symptomatic, respectively. Population C are individuals who were *silently cleared*, meaning they were infected but never became symptomatic. Finally, D is the number of deaths. The basic reaction set is as follows:

$$\begin{aligned}
&R_1 : S + I_1 \xrightarrow{k_e} I_1 + E \qquad && R_4 : I_1 \xrightarrow{k_p} I_2 \qquad && R_6 : I_2 \xrightarrow{k_r} R \\
&R_2 : S + I_2 \xrightarrow{k_e} I_1 + I_2 && R_5 : I_1 \xrightarrow{k_s} C && R_7 : I_2 \xrightarrow{k_d} D. \\
&R_3 : E \xrightarrow{k_i} I_1 && &&
\end{aligned} \tag{21}$$

In this example we consider the effect of a societal "lockdown" to reduce virus transmission, as was common in the early months of the Covid-19 pandemic. To model a lockdown, we introduce two events. The first event *triggers* at time $t = 20$ and modifies parameter k_e by a value q between 0 and 1 corresponding to the reduction in exposure due to the lockdown. The second event ends the lockdown at $t = 70$ by modifying parameter k_e by a value between q and 1, where a value of 1 would simulate the population fully returning to pre-lockdown behaviors. An Event object is composed of an EventTrigger and an EventAssignment object. For example, the first event is added to the model as follows:

```
e1trigger = EventTrigger(expression="t>=20")
e1action = EventAssignment(variable=self.listOfParameters["k_e"],
        expression="q*k_e0")
lockdown=Event(name="lockdown_start", trigger=e1trigger,
        assignments=[e1action])
self.add_event(lockdown)
```

Here "k_e0" is the nominal value of k_e. The Event's trigger will become true at $t = 20$, then the EventAssignment will be executed. The Event for ending the lockdown is defined similarly. This model was used to create part (C) in Figure 1.

5.3 Hybrid simulation of photosynthesis

To demonstrate the Tau-Hybrid solver, we have created an example model of photosynthesis based on (Whitmarsh and Covindjee, 1999). This model demonstrates the two major features of the Tau-Hybrid solver. The first feature is the ability to couple stochastic reactions to continuously changing variables. In this model, a *RateRule* is assigned to the variable "photons" to set its continuous value based on a derivative proportional to $\sin(t\pi/12)$ to mimic the daily cycle of the sun. The domain of interest is a cell of a plant, though we do not explicitly model this. Instead, we simulate the diffusion of carbon dioxide, water, and oxygen into and out of the cell using purely deterministic reactions, with the steady-state being the initial condition of those quantities. The second feature we illustrate with this model is the ability for a reaction to be simulated stochastically at low populations and automatically switch the reaction to a continuous mode when the relative noise of the reactants and products falls below the threshold level. In this case, we have specified that the variable representing the carbohydrates in the model will switch when its relative noise is less than the default threshold of 0.03, using the following code:

```
Species(name="Carbohydrate", initial_value=0, mode="dynamic")
```

Alternatively, switching behavior can also be specified by a minimum population level parameter that changes the type from *continuous* to *discrete* when the species population drops below the specified threshold.

Figure 4 shows the results of the simulation of this model. The photons (green) change continuously with time. The carbohydrates (red) increase when Photons are high, and change discretely (taking only integer values) until $t \approx 17$. Each reaction event can be seen as a step increase in this variable. After $t \approx 17$, the Carbohydrates variable is simulated deterministically, and thus has smooth changes, though the qualitative behavior remains the same. Carbon dioxide (blue) and water (orange) are consumed in the photosynthesis reaction and oxygen (purple) is produced. Before $t \approx 17$ discontinuous jumps in their values correspond to reaction events. During this same time, the deterministic diffusion in/out reaction also changes the values continuously between reaction events.

This example demonstrates that the Tau-Hybrid solver allows the modeler to couple stochastic reactions with deterministic reactions and continuously time-varying inputs. It allows the modeler to specify whether a set of reactions should be stochastic, deterministic, or to switch between the two options based on specified thresholds. These features are applicable to a large class of models, as it is important to capture stochastic dynamics, however it is difficult to *a priori* determine which species and which reactions need to be simulated discretely, and which ones can be accurately simulated deterministically. GillesPy2's Tau-Hybrid solver switches between these two modes automatically, giving the modeler an accurate and efficient simulation without requiring any extra knowledge.

6 Discussion

6.1 Performance

In GillesPy2 we provide a variety of solvers to our users, so that we may address as many different computational requirements as possible. GillesPy2 includes logic to choose the correct solver for the user automatically, by inspecting the features used in the model construction and testing if the C++ solver can be used in the current computation environment. This logic is contained in the "get_best_solver()" method. This method will return a C++ implementation over the Python counterpart, if supported. It will inspect the model for any advanced features (e.g., Event, Assignment Rules, Rate Rules, or Function Definitions) and returns the Tau-Hybrid solver if so, otherwise it returns the SSA Solver. These heuristics are appropriate for many situations, however we suggest that our users specify the solver that most closely fits their computational situation to maximize productivity.

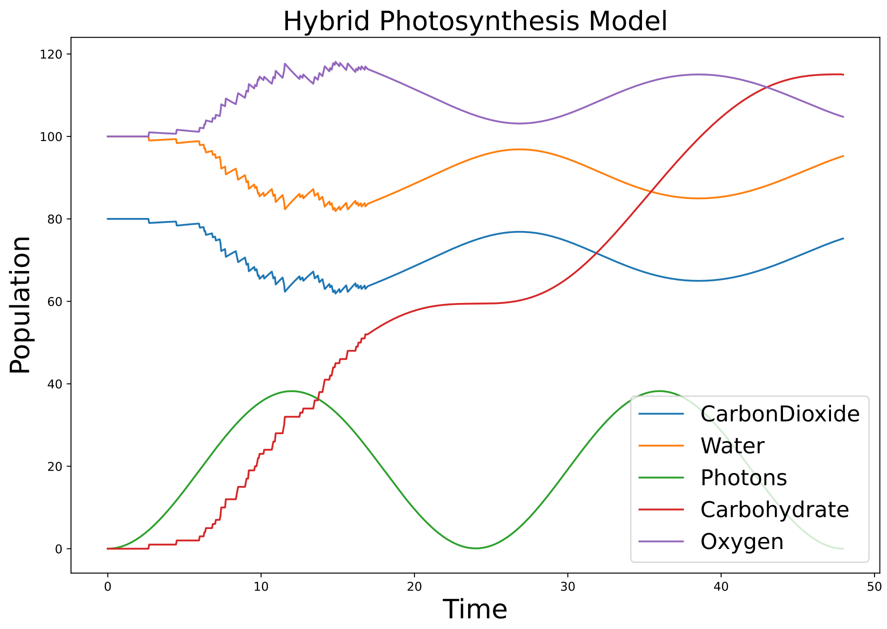

Figure 4: An example model showing hybrid ODE/SSA simulation. This model of photosynthesis, adapted from Whitmarsh and Covindjee (1999), couples an external time-varying input (sunlight, green) with a reaction network. Diffusion in and out of the volume of interest is simulated deterministically, while the reaction that creates sugars (carbohydrates, red) is stochastic when the population of carbohydrates is low (time < 17) and switches to deterministic after time ≈ 17 because the calculated noise crosses below the tolerance threshold.

The SSA solvers have been optimized for performance in simulating stochastic systems, and as such does not support as many features. The ODE solvers have similarly for deterministic systems. However, the Tau-Hybrid solvers have been developed to simulate the most complex biochemical models by simultaneously solving ODE and SSA systems, thus it must do more work than either of them alone.

Understanding all of the performance characteristics of a simulation is a complex and multifaceted endeavor. To assist our readers in understanding the performance of GillesPy2, we present three analyses. In Figure 5, we compare the performance of the SSA, Tau-Hybrid, Tau-Leaping, and ODE solvers as a function of increasing system size by increasing the steady-state population, $X_0^* = X(t = 0)$, in the birth-death model where $\emptyset \longrightarrow X$ at rate $k * X_0^*$, and $X \longrightarrow \emptyset$ at rate k (e.g., $a_1(X) = k * X_0^*$ and $a_2(X) = k * X$). Both the C++ (A) and Python (B) implementations are analyzed. In Figure 6 we show separately total time (in A), initialization and compilation time (in B), and simulation run time (in C) as a function of the number of simulated trajectories. We see that in some fast-running simulations, the compilation time dominates the total time. In Table 2 we compare a set of common biochemical models with the GillesPy2 solvers, and additionally show the time for the simulation with StochKit2 (Sanft et al., 2011), for external comparison. Some entries are marked "N/A" if the solver does not possess the capabilities to correctly simulate the model. The models compared are: Decay ($S \longrightarrow \emptyset$, $S(0) = 100$), Dimerization ($2\,S_1 \leftrightarrow S_2$, $S_1(0) = 30$), Michaelis-Menten (Michaelis and Menten, 1913), Genetic Toggle Switch (Gardner et al., 2000), Tyson 2-state Oscillator (Tyson, 1991), Vilar Oscillator (Vilar et al., 2002), a Multi-Event model (a test model with events), and an All-SBML-Features model (a test model that includes all supported SBML features: Event, Assignment Rules, Rate Rules, and Function Definitions).

6.2 Availability

GillesPy2 is open source software, available under the GNU General Public License (Version 3) (`https://github.com/GillesPy2/GillesPy2/blob/main/LICENSE`). Like many Python 3 packages, stable versions of GillesPy2 can be installed from PyPI via the Python "pip" command. It is also available on `conda-forge.org` for those using the Conda package manager. The source code and documentation can be downloaded from `http://www.github.com/gillespy2/gillespy2`. Users are encouraged to report bugs and request new features through the github repository's web interface.

GillesPy2 is part of the StochSS ecosystem (Drawert et al., 2016). The StochSS organization encompasses multiple projects, including: StochSS Live (Jiang et al., 2021), SCIOPE (Scalable inference, optimization and parameter exploration) (Singh et al., 2020), and SpatialPy. GillesPy2 is the engine for all spatially homogeneous modeling and simulation services in StochSS. Through StochSS live, a Software-as-a-Service platform, users can create, simulate, and analyze models in a web browser.

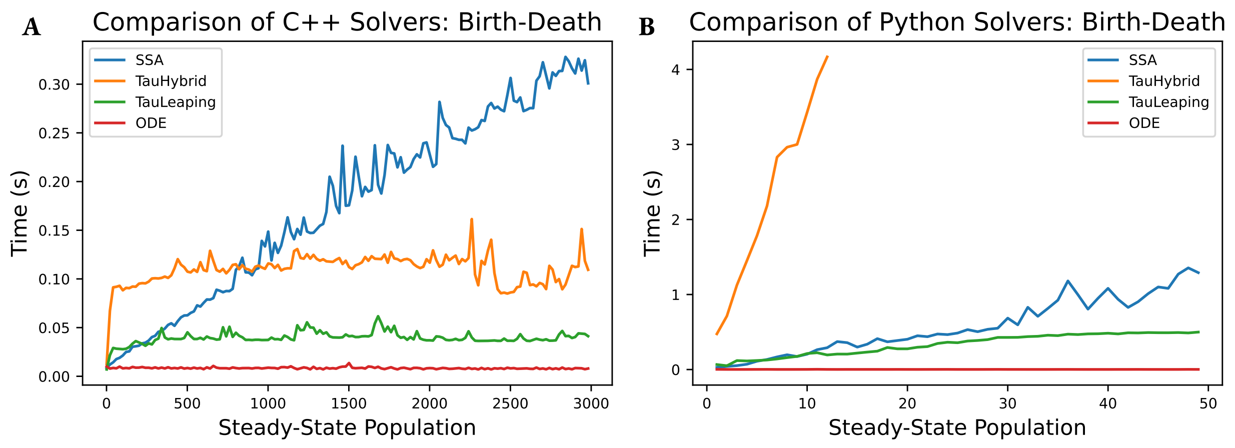

Figure 5: Comparison of solver performance for SSA, Tau-Hybrid, Tau-Leaping, and ODE solvers, for both the C++ implementations (A) and Python implementations (B). Performance is measured using a birth-death model across a range of steady-state conditions. This shows how performance of each solver changes as the system size increases. Note that the C++ solvers are significantly faster than the Python implementations, thus we compare them separately. Also note that the Tau-Leaping and Tau-Hybrid solvers are nearly flat with respect to system size, as expected since their performance scales with number of reaction channels. This is in contrast to the SSA which scales linearly with population size.

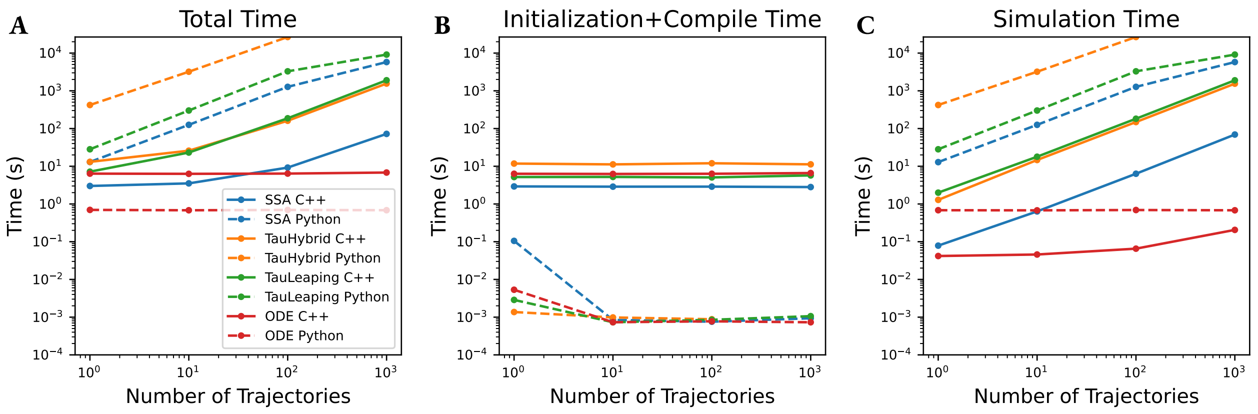

Figure 6: Performance analysis of GillesPy2 running the Vilar Oscillator (Vilar et al., 2002) model, showing (A) Total simulation time, (B) Compile time, and (C) Run time versus number of trajectories. Note that for the C++ solvers (solid lines), the compile time is a large fraction of the total time for fast-running simulations. The C++ solvers are all faster than their Python counterparts (dashed lines), however the relatively large compile time must be taken into account when considering the performance trade-offs of each solver.

Table 2: Total simulation time (in seconds) comparison for GillesPy2 Solvers and StochKit2 (Sanft et al., 2011) for popular biochemical models. Times reported are the total time (compile + simulation) for a single trajectory. See Figure 6 for an example of performance versus ensemble size.

Model	SSA	SSA C++	Tau-Hybrid	Tau-Hybrid C++	Tau-Leaping	Tau-Leaping C++	StochKit2 (SSA)
Decay	0.00312	3.893	0.0610	16.292	0.00605	8.015	0.0406
Dimerization	0.00338	3.829	0.125	13.464	0.0132	7.493	0.0764
Michaelis-Menten	0.0654	4.573	0.797	17.010	0.0711	7.861	0.0743
Genetic Toggle Switch	0.0876	3.085	1.457	16.22	0.0885	6.800	7.623
Tyson 2-state Oscillator	21.16	3.164	303.1	14.29	22.16	6.807	6.984
Vilar Oscillator	15.29	3.167	299.5	14.42	31.72	8.301	0.223
Multi-Event model	N/A	N/A	0.246	16.22	N/A	N/A	N/A
SBML features model	N/A	N/A	0.586	N/A	N/A	N/A	N/A

7 Conclusion

GillesPy2 is an open source Python 3 package that provides biological modelers a set of user-friendly tools to facilitate their workflows. GillesPy2's object-oriented design allows users to build, simulate, analyze, and improve their models using an intuitive set of classes and methods. The core solvers are written in C++ and Python for high performance and accessibility. The novel Tau-Hybrid implementation is compatible with several SBML model features, including events and rate rules. Additionally, GillesPy2 works well with Jupyter notebooks, facilitating open and reproducible science. In fact, all of the examples and code in this paper are available in notebooks posted at `https://github.com/GillesPy2/GillesPy2_paper`. All figures and data in this paper are based on GillesPy2 version 1.7.0.

For biological modelers who would like to use GillesPy2, a great place to start is the "Start_Here.ipynb" Jupyter notebook in the GillesPy2/examples folder. Users can start with a provided working example and then modify the code to develop their own model. The examples folder contains additional notebooks that showcase several different biochemical models and notebooks that demonstrate many GillesPy2 features. Users can also consult the extensive documentation on the web at `https://gillespy2.readthedocs.io/`. The documentation provides everything from a "Basic usage" page that walks through the basic workflow, to a complete API reference of all GillesPy2 classes and public methods. Through the user-friendly API, example notebooks, and extensive documentation, users can progress quickly from GillesPy2 beginners to experts who can utilize the powerful range of biochemical modeling, simulation, and analysis tools that are available in GillesPy2.

Acknowledgments

The authors acknowledge research funding from NIBIB Award No. 2-R01-EB014877-04A1. The content of the information does not necessarily reflect the position or the policy of the funding agency, and no official endorsement should be inferred.

We would like to thank all of the contributors to GillesPy2. A complete list can be found in the Github repository in the AUTHORS file.

References

Abel, J. H., B. Drawert, A. Hellander, and L. R. Petzold (2016). GillesPy: A Python package for stochastic model building and simulation. *IEEE Life Sciences Letters 2*(3), 35–38. 87

Ahmadian, M., S. Wang, J. Tyson, and Y. Cao (2017). Hybrid ODE/SSA model of the budding yeast cell cycle control mechanism with mutant case study. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, ACM-BCB '17, New York, NY, USA, pp. 464–473. Association for Computing Machinery. 91

Ascher, U. M. and L. R. Petzold (1998). *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, Volume 61. Siam. 90

Cao, Y., D. T. Gillespie, and L. R. Petzold (2006). Efficient step size selection for the Tau-Leaping simulation method. *The Journal of Chemical Physics 124*(4), 044109. 91, 93, 94, 95, 96

Choi, K., J. K. Medley, M. König, K. Stocking, L. Smith, S. Gu, and H. M. Sauro (2018). Tellurium: An extensible Python-based modeling environment for systems and synthetic biology. *Biosystems 171*, 74–79. 88

Drawert, B., M. Griesemer, L. R. Petzold, and C. J. Briggs (2017). Using stochastic epidemiological models to evaluate conservation strategies for endangered amphibians. *Journal of the Royal Society Interface 14*(133), 20170480. 87

Drawert, B., A. Hellander, B. Bales, D. Banerjee, G. Bellesia, B. J. Daigle Jr, G. Douglas, M. Gu, A. Gupta, S. Hellander, et al. (2016). Stochastic simulation service: Bridging the gap between the computational expert and the biologist. *PLoS Computational Biology 12*(12), e1005220. 88, 99

Drawert, B., S. Matthew, M. Powell, and B. Rumsey (2022). Saving the devils is in the details: Tasmanian devil facial tumor disease can be eliminated with interventions. *bioRxiv*. 87

Drawert, B., N. Thakore, B. Mitchell, E. Pioro, J. Ravits, and L. R. Petzold (2017). Modeling the neuroanatomic propagation of ALS in the spinal cord. In *AIP Conference Proceedings*, Volume 1863, pp. 500002. AIP Publishing LLC. 87

El Samad, H., M. Khammash, L. Petzold, and D. Gillespie (2005). Stochastic modelling of gene regulatory networks. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal 15*(15), 691–711. 87

Elowitz, M. B., A. J. Levine, E. D. Siggia, and P. S. Swain (2002). Stochastic gene expression in a single cell. *Science 297*(5584), 1183–1186. 87

Gardner, T. S., C. R. Cantor, and J. J. Collins (2000). Construction of a genetic toggle switch in Escherichia coli. *Nature 403*(6767), 339–342. 99

Gibson, M. and J. Bruck (2000). Efficient exact stochastic simulation of chemical systems with many species and many channels. *The Journal of Physical Chemistry A 104*(9), 1876–1889. 90

Gillespie, D. (2001, 07). Approximate accelerated stochastic simulation of chemically reacting systems. *Journal of Chemical Physics 115*, 1716–1733. 91

Gillespie, D. T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics 22*, 403–434. 87, 92

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry 81*(25), 2340–2361. 87, 91, 95, 96

Gillespie, D. T. (2009). Deterministic limit of stochastic chemical kinetics. *The Journal of Physical Chemistry B 113*(6), 1640–1644. 93

Gillespie, D. T., A. Hellander, and L. R. Petzold (2013). Perspective: Stochastic algorithms for chemical kinetics. *The Journal of Chemical Physics 138*(17), 05B201_1. 87, 90, 91

Harris, L. A., J. S. Hogg, J.-J. Tapia, J. A. Sekar, S. Gupta, I. Korsunsky, A. Arora, D. Barua, R. P. Sheehan, and J. R. Faeder (2016). BioNetGen 2.2: Advances in rule-based modeling. *Bioinformatics 32*(21), 3366–3368. 88

Helms, T., P. Wilsdorf, and A. M. Uhrmacher (2018). Hybrid simulation of dynamic reaction networks in multi-level models. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pp. 133–144. 91

Hindmarsh, A. C., P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward (2005). SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS) 31*(3), 363–396. 90, 95, 96

Hoops, S., S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer (2006). COPASI—a COmplex PAthway SImulator. *Bioinformatics 22*(24), 3067–3074. 88

Hucka, M., A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, et al. (2003). The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics 19*(4), 524–531. 90, 95

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering 9*(3), 90–95. 96

Jiang, R., B. Jacob, M. Geiger, S. Matthew, B. Rumsey, P. Singh, F. Wrede, T.-M. Yi, B. Drawert, A. Hellander, and L. Petzold (2021). Epidemiological modeling in StochSS Live! *Bioinformatics*. 87, 97, 99

Kurtz, T. G. (1972). The relationship between stochastic and deterministic models for chemical reactions. *The Journal of Chemical Physics 57*(7), 2976–2978. 91

Landeros, A., T. Stutz, K. L. Keys, A. Alekseyenko, J. S. Sinsheimer, K. Lange, and M. E. Sehl (2018). BioSimulator.jl: Stochastic simulation in Julia. *Computer Methods and Programs in Biomedicine 167*, 23–35. 88

Lopez, C. F. and S. P. Garbett (2014). Pysb: A modeling framework to explore biochemical signaling processes and cell-decisions. *Biophysical Journal 106*(2), 643a. 88

Lotka, A. J. (1920). Analytical note on certain rhythmic relations in organic systems. *Proceedings of the National Academy of Sciences 6*(7), 410–415. 97

Maarleveld, T. R., B. G. Olivier, and F. J. Bruggeman (2013). StochPy: A comprehensive, user-friendly tool for simulating stochastic biological processes. *PloS ONE 8*(11), e79345. 88

Mauch, S. and M. Stalzer (2011). Efficient formulations for exact stochastic simulation of chemical systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics 8*(1), 27–35. 90

McCauley, J. (2013). *Stochastic Calculus and Differential Equations for Physics and Finance*. Cambridge University Press. 95, 96

Michaelis, L. and M. L. Menten (1913). Die Kinetik der Invertinwirkung. *Biochemische Zeitschrift 49*, 333–369. 88, 89, 99

Munsky, B. and M. Khammash (2006). The finite state projection algorithm for the solution of the chemical master equation. *The Journal of Chemical Physics 124*(4), 044104. 90

Pahle, J. (2009). Biochemical simulations: Stochastic, approximate stochastic and hybrid approaches. *Briefings in Bioinformatics 10*(1), 53–64. 91

Petzold, L. (1983). Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM Journal on Scientific and Statistical Computing 4*(1), 136–148. 90, 96

Pineda-Krch, M. (2008). GillespieSSA: Implementing the Gillespie stochastic simulation algorithm in R. *Journal of Statistical Software 25*, 1–18. 88

Plotly Technologies Inc. (2015). Collaborative data science. 96

Ramaswamy, R., N. González-Segredo, and I. F. Sbalzarini (2009). A new class of highly efficient exact stochastic simulation algorithms for chemical reaction networks. *The Journal of Chemical Physics 130*(24), 244104. 90

Salis, H. and Y. Kaznessis (2005). Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. *The Journal of Chemical Physics 122*(5), 054103. 92

Sanft, K. R. and H. G. Othmer (2015). Constant-complexity stochastic simulation algorithm with optimal binning. *The Journal of Chemical Physics 143*(7), 08B609_1. 90

Sanft, K. R., S. Wu, M. Roh, J. Fu, R. K. Lim, and L. R. Petzold (2011). StochKit2: Software for discrete stochastic simulation of biochemical systems with events. *Bioinformatics 27*(17), 2457–2458. 87, 99, 100

Singh, P., F. Wrede, and A. Hellander (2020, 07). Scalable machine learning-assisted model exploration and inference using Sciope. *Bioinformatics*. 99

Slepoy, A., A. P. Thompson, and S. J. Plimpton (2008). A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks. *The Journal of Chemical Physics 128*(20). 90

Somogyi, E. T., J.-M. Bouteiller, J. A. Glazier, M. König, J. K. Medley, M. H. Swat, and H. M. Sauro (2015). libRoadRunner: A high performance SBML simulation and analysis library. *Bioinformatics 31*(20), 3315–3321. 88

Tyson, J. J. (1991). Modeling the cell division cycle: cdc2 and cyclin interactions. *Proceedings of the National Academy of Sciences 88*(16), 7328–7332. 99

Van Rossum, G. and F. L. Drake (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace. 87

Vilar, J. M. G., H. Y. Kueh, N. Barkai, and S. Leibler (2002). Mechanisms of noise-resistance in genetic oscillators. *Proceedings of the National Academy of Sciences 99*(9), 5988–5992. 99, 100

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods 17*(3), 261–272. 90, 95, 96

Volterra, V. (1926). Fluctuations in the abundance of a species considered mathematically. *Nature 118*(2972), 558–560. 97

Whitmarsh, J. and Covindjee (1999). The photosynthetic process. In G. Singhal, G. Renger, S. Sopory, K.-D. Irrgang, and Govindjee (Eds.), *Concepts in Photobiology: Photosynthesis and Photomorphogenesis*. Narosa Publishing House, New Dehli, India. 98, 99