Compartmental Modeling Software (CMS)

Release 0.9 beta

Institute for Disease Modeling

Jan 13, 2021

CONTENTS

The Institute for Disease Modeling (IDM) develops disease modeling software that is thoroughly tested and shared with the research community to advance the understanding of disease dynamics. This software helps determine the combination of health policies and intervention strategies that can lead to disease eradication.

The primary software, EMOD, is an agent-based *stochastic* model that simulates the simultaneous interactions of agents in an effort to recreate complex phenomena. CMS is also stochastic, but uses the compartmental modeling framework to run faster, simpler simulations.

CHAPTER

ONE

INTRODUCTION TO COMPARTMENTAL MODELING

Compartmental modeling is widely used among epidemiologists to simulate disease dynamics. These models treat each disease state as a different compartment that contains a homogeneous population of individuals. Depending on the disease being simulated, the compartments can be susceptible (S), exposed (E), infectious (I), or recovered (R).

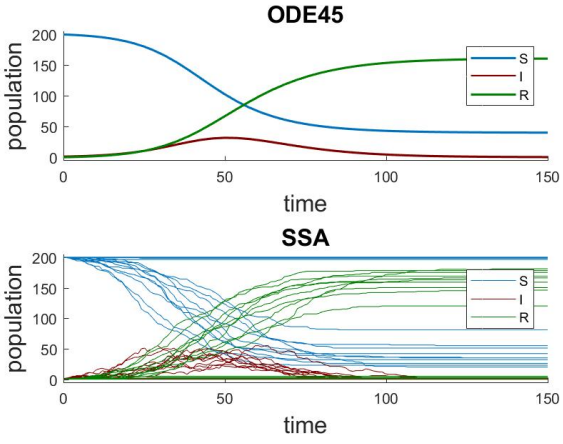
The most common compartmental models are *deterministic*; that is, given the same inputs, they will always produce the same outputs. Deterministic compartmental models use either an *ordinary differential equation (ODE)* or a *partial differential equation (PDE)*. While this type of compartmental model is very fast to simulate, they aren't suited to all situations.

There are many cases where *stochastic* modeling that uses *chemical master equation (CME)* based methods is preferred. See CME on Wikipedia for more information. Stochastic framework provides distributions associated with characteristics of a process and rigorous procedures for inference. In addition, deterministic models do not provide an accurate description of the system when the population in any of the compartments is low.

For example, consider the following simple SIR model.

$$
\begin{aligned}
S + I \xrightarrow{\beta} 2I, \quad & \beta = 0.001 \\
I \xrightarrow{\gamma} R, \quad & \gamma = 0.1 \\
\mathbf{x}_0 = [200\ 1\ 0] \quad & t_f = 150
\end{aligned}
$$

In the above system, we start with 200 susceptible individuals, 1 infectious, and 0 recovered. The deterministic results from ODE45 are compared to 20 stochastic trajectories simulated using the stochastic simulation algorithm by *Gillespie (SSA)* (SSA)[1].

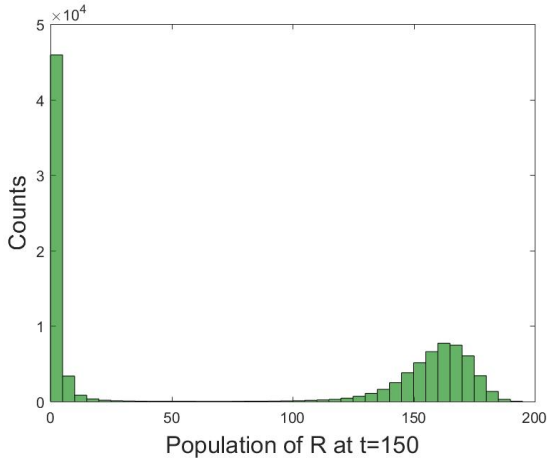


You can see that many of the SSA trajectories show no outbreak. This is because only one infectious individual is in the initial state. The probability of the single infectious individual recovering in the next time step is

[1] Gillespie, Daniel T. "Exact stochastic simulation of coupled chemical reactions." The Journal of Physical Chemistry 81.25 (1977): 2340-361.

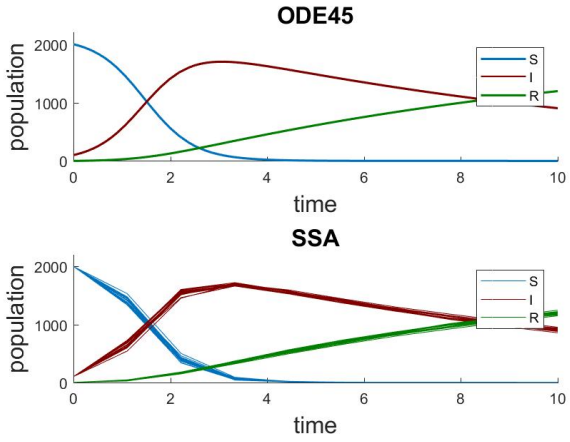
$1/3\left(\frac{1\times0.1}{200\times1\times0.001+1\times0.1}\right)$, while infecting one of 200 susceptible individual is $2/3\left(\frac{200\times1\times0.001}{200\times1\times0.001+1\times0.1}\right)$. At the same time, there are also SSA trajectories that contain earlier and larger outbreaks (population of I) compared to the trajectory of I from the deterministic simulation.

With a large number of SSA trajectories, you can obtain an accurate distribution of states in time. For example, the distribution of recovered individuals R at $t = t_f(150)$ using 10^5 SSA trajectories looks like the following:



Such distributions can be used to obtain many useful insights into the system. The first mode in the distribution (left peak) indicates that no large outbreak is observed almost half of the time by $t = 150$. The second mode indicates the type of population immunity that may be observed at $t = 150$. Looking at the same distribution in time can be used to study how the immunity changes over time.

As the size of population increases, SSA trajectories start looking more similar to the ODE result and exhibit less variability among themselves. When we change the initial population to $x_0 = [2000\ 100\ 0]$, we get the following result.



Intrinsic stochasticity may differ greatly from one model to another, depending on many factors, such as reaction rates, number of non-linear reactions, connectivity among different compartments, and population size. When a system contains compartments with a relatively large population where stochasticity still matters, we can use *Approximate methods* to speed up the simulation. Several popular *Spatial simulation methods* are also supported in CMS, along with rare event (*Doubly weighted stochastic simulation algorithm (dwSSA)* and *State-dependent doubly weighted stochastic simulation algorithm (sdwSSA)*) simulation methods.

CHAPTER

TWO

INTRODUCTION TO THE SOFTWARE

Compartmental Modeling Software (CMS) is a *stochastic compartmental model* that is simpler to configure and faster to run than an agent-based model like EMOD, while more suitable to modeling low probability events than a *deterministic* compartmental model. CMS has multiple solvers and output formats available and uses a simple text-based model specification and simulation configuration.

CMS is provided to the research community to help advance the understanding of disease dynamics. IDM does not provide the same level of support for CMS as it does for EMOD; however, if you encounter any issues while using CMS, please contact support@idmod.org and we will help if able.

2.1 CMS installation

1. Download compartments.zip from the IDM-CMS repository on GitHub.
2. Unzip compartments.zip to your desired location.

Note: Do not double-click the compartments.exe file to run simulations. CMS simulations are run at the command line by referring to the executable. See *Running a simulation* for more information.

2.2 Input files

You will generally use two input files to run CMS simulations: the *model file* is required and the *configuration file* is optional.

The model file contains all information that defines the mathematical model itself. For example, it defines the *species*, the transition rates, incubation time, daily recovery rate, and many other parameters specific to the disease being modeled. The model file is in *EMODL* format; the syntax and available parameters are described in *Model file syntax*.

The configuration file contains information specific to a simulation. For example, the number of realizations, the duration of each *realization*, and the solvers to use. You may have several different configuration files that are used with the same model file. If you do not specify a configuration file, the simulation runs one realization for 100 time steps. The configuration file is in *JSON (JavaScript Object Notation)* format, though it uses the ".cfg" file extension. The syntax and available parameters are described in *Configuration file syntax*.

2.3 Running a simulation

To run a CMS simulation, do the following:

1. Open a Command Prompt window and navigate to the directory that contains the configuration and model files.
2. Enter a command like the one illustrated below, substituting the full path to the executable and file names for your simulation:

```
../compartments.exe -c <config_file.cfg> -m <model_file.emodl>
```

   Where <config_file.cfg> and <model_file.emodl> are names of your configuration and model files, respectively. If you run a simulation from a different directory, you must include the full path to these files.

3. CMS will display status information, including any errors that occur, while running the simulation.
4. By default, the output file trajectories.csv will be created in the current directory. You can specify additional output files in the configuration file.

Note: You can add the path to the executable to your **PATH** environment variable so you do not need to type the full path to compartments.exe.

To set **PATH** for the current Command Prompt window only, use `set`. For example, `set PATH=%PATH%;c:\cms\`.

To permanently modify **PATH**, use `setx`. For example, `setx PATH "%PATH%;c:\cms"`. You must then open a new Command Prompt window.

CHAPTER

THREE

INPUT AND OUTPUT REFERENCE

This section describes all input and output options available in CMS. For example, the syntax used in the *model file* used to define the mathematics of the model, the random number libraries and solvers available to use in the *configuration file*, and the different output reports that can be created.

3.1 Model file syntax

The *model file* is very flexible and allows you to create spatial models, generate custom propensity functions, and more. The model file uses *EMODL* (Epidemiological Model Language) syntax defined by IDM that is similar to LISP. Lines are enclosed in parentheses. Any text that follows a semi-colon (;) is treated as a comment until the beginning of the next line.

The basic format of the model file is as follows:

```
(import (rnrs) (emodl cmslib))
(start-model "modelname")
...
(end-model)
```

3.1.1 Basic EMODL syntax

All EMODL arguments available to define the CMS model are listed below.

bool Defines a function that returns a boolean value, used with **state-event**.

Syntax `(bool name expression)`

Example `(bool exitTimeEvent (== R 85))`

func Defines a numeric function that is evaluated each time it is needed.

Syntax `(func name function)`

Exampe `func muA (/ 1 60))`

json Declares an external JSON-formatted file that can be referenced in species definitions, parameter definitions, and functions.

Syntax `json name file`

Example `(json defaults "garkiparams.json")`

locale Creates a new geographic locale.

Syntax `(locale name)`

Example `(locale site-1)`

observe Adds a variable or function to the list of observed items that are output from a simulation.

Syntax `(observe label function)`

Example `(observe carrier C)`

param Defines a named, constant value.

Syntax `(param name value)`

Example `(param reVal 0)`

reaction Defines a reaction or transition from one set of species to another.

Syntax `(reaction name input-species output-species propensity-function)`

Example `(reaction recoveryIc (Ic) () (/ (* gamma Ic) 20))`

set-locale Sets the current geographic local. New species will be associated with this locale. This is used with spatial solvers.

Syntax `(set-local name)`

Example `(set-locale site-1)`

species Defines a unique species or population of particles or agents.

Syntax `(species name [initial population])`

Example `(species Sa 2500)`

state-event Defines an event to occur given a particular system state.

Syntax `(state-event name predicate (variable-value pairs))`

Example `(state-event death-v (> I 25) ((Kv 0.02)))`

time-event Defines an event to occur at a particular time. You have the option to add recurrent events.

Syntax `(time-event name time iterations (variable-value pairs))`

Example `(time-event sia 50.0 ((Kv 0.02)))`

variable-value pairs This is not a named parameter, but rather a list of pairs of variables to set and the value to which to set them.

Syntax `((var (val)))(var (val)))`

Example `((V (* S 0.5)) (S (* S 0.5)))` sets V = S/2 and then sets S = S/2 (in other words, it transfers half the population of S to V).

3.1.2 Mathematical operators and functions

The arguments can be used with the following operators to define the mathematics of the CMS model.

Unary

negate `(- x)`

exponentiation `(exp x)` returns e^x

logarithm `(ln x)`

sine `(sin x)`

cosine `(cos x)`

absolute `(abs x)`

floor `(floor x)` returns the largest integer <= x.

ceiling `(ceil x)` returns the smallest integer >= x.

square root `(sqrt x)`

Heaviside step `(step x)` returns 1 if x >= 0 else returns 0.

empirical `(empirical "filename")` reads an empirically defined cumulative distribution function from the file specified and returns a probability from the configuration file based on a random number draw.

Binary

add `(+ x y)`

subtract `(- x y)`

multiply `(* x y)`

divide `(/ x y)`

power `(^ x y)` or `(pow x y)`

minimum `(min x y)`

maximum `(max x y)`

uniform `(uniform min max)` returns a value uniformly distributed between min and max based on a random number draw.

normal `(normal mean var)` or `(gaussian mean var)` returns a value from a normal distribution with the given mean and variance.

N-ary

add: `(+ x y z ...)` or `(sum x y z ...)` returns the sum of all the given arguments.

multiply: `(* x y z ...)` returns the product of all the given arguments.

3.1.3 Example

The following example shows an simple SEIR model specification.

```
; simplemodel

(import (rnrs) (emodl cmslib))

(start-model "seir.emodl")

(species S 990)
(species E)
(species I 10)
(species R)

(observe susceptible S)
(observe exposed     E)
(observe infectious  I)
(observe recovered   R)

(param Ki 0.0005)
(param Kl 0.2)
(param Kr (/ 1 7))
(param Kw (/ 1 135))

(reaction exposure   (S I) (E I) (* Ki S I))
(reaction infection  (E)   (I)   (* Kl E))
(reaction recovery   (I)   (R)   (* Kr I))
(reaction waning     (R)   (S)   (* Kw R))

(end-model)
```

Note that when a species is in both the input and the output, neither is technically necessary. In other words, the following reaction specifications are equivalent. You can think about this as catalysis: an S in the presence of I becomes an E with the I unaffected.

```
(reaction exposure   (S I) (E I) (* Ki S I))
(reaction exposure   (S)   (E)   (* Ki S I))
```

Custom propensity function

In traditional compartmental modeling, the rate at which a reaction occurs obeys the law of mass action. In other words, it is directly proportional to the population of the reacting compartment(s). For example, an EMODL SEIR model with waning immunity will contain reaction terms as the following:

```
(reaction exposure   (S I) (E I) (* Ki S I))
(reaction infection  (E)   (I)   (* Kl E))
(reaction recovery   (I)   (R)   (* Kr I))
(reaction waning     (R)   (S)   (* Kw R))
```

However, mass action dynamics can be too restricting in modeling a complex epidemiological system with mechanisms such as seasonal forcing, pulse vaccination, and discrete aging. CMS provides a set of *Mathematical operators and functions* to aid formulating custom propensity functions. Below is a part of the Garki model[1] that involves seasonal forcing:

[1] http://garkiproject.nd.edu/static/documents/garkiproject.pdf (Chapter 10, page 263)

```
; seasonal parameter
(func C (* 0.2 (+ 1.01 (sin (* (/ time 365) 2 pi)))))
; infection rate
(func h (* g (- 1 (exp (/ (* (- C) Y1) totalpop)))))

(reaction recoveryY3 (Y3) (X3) (/ (* Y3 h) (- (exp (/ h r2)) 1)))
```

Create a spatial model

Spatial structure is an important component in many models of disease. CMS offers an easy way to specify geographic compartments. The syntax for creating a locale is `(locale name)`, and all *Spatial simulation methods* support this feature.

Species must be specified with the name of the locale where it belongs in the following format: `species_name::locale_name`. The below example illustrates how locales and species are created for a spatial simulation.

```
(locale site-1)
(set-locale site-1)
(species A::1 1000)

(locale site-2)
(set-locale site-2)
(species A::2 1000)

(reaction A1->A2 (A::1) (A::2) (* D A::1))
(reaction A2->A1 (A::2) (A::1) (* D A::2))
```

3.2 Configuration file syntax

While CMS does not require a configuration file, running a simulation without one uses default settings that will not produce scientifically useful results. As a stochastic model, you must run many realizations in a CMS simulation for statistically significant results. The configuration file uses JSON syntax and a .cfg file extension.

The table below shows the basic parameters used in a minimal configuration.

Parameter	Data type	Default	Description
duration	integer	100	The number of time steps to run the realizations for; specified in unitless time relevant to the timescales of the model. The values will correspond to the units specified in the rates used in the model file.
runs	integer	1	The number of realizations to run for the simulation.
samples	integer	100	The number of samples of the various observables to take over the duration of the simulation.

Besides the parameters listed above, you will likely want to include configuration settings to control the solver that is used, the output created, and the pseudo-random number generator. By default, the *Gillespie (SSA)* solver is used and the file trajectories.csv is created in the output directory. See the other topics in this section for all available configuration parameters.

3.2.1 Example

The following example specifies the *Gillespie (SSA)* algorithm, 100 realizations, a duration of 730 time units (two years for a model with rates specified in days), and 250 samples of each realization spaced evenly over the 730 time unit duration.

```
{
    "solver"   : "SSA",
    "runs"     :   100,
    "duration" :   730,
    "samples"  :   250
}
```

Solvers

The CMS software offers 16 different solvers. They can be categorized into the following categories. The best *solver* to use will depend up on the disease and data you are simulating. Note that all solver names are case-insensitive in their corresponding .cfg file. If you do not specify a solver, *Gillespie (SSA)* will be used.

Exact methods

Assuming a homogeneously mixed system, these methods are exact in that every reaction firing and its time are computed explicitly. While exact, these methods can be computationally expensive.

Gillespie (SSA)

The Gillespie algorithm[1] , also known as the Stochastic Simulation Algorithm (SSA), is an exact *Monte Carlo method* for numerically generating time trajectories of the system state populations in accordance with the *chemical master equation (CME)*, which is the governing probability distribution of all possible states in time in homogeneously mixed population. The Gillespie solver features the Direct Method (DM) implementation, which is the most commonly used. The other variant of the algorithm, First Reaction Method (FRM), is theoretically equivalent to the DM but differs in implementation. FRM is implemented in *GillespieFirstReaction*.

While these methods are exact, both the Gillespie and the *GillespieFirstReaction* solvers are computationally expensive as every reaction and its firing time are explicitly computed.

Parameter	Data type	Description
solver	string	**Gillespie**, **GillespieDirect**, and **SSA** are all valid names to run this solver.

[1] Gillespie, Daniel T. "Exact stochastic simulation of coupled chemical reactions" The Journal of Physical Chemistry 81.25 (1977): 2340-361.

Example

```
{
    "duration": 100,
    "runs": 10000,
    "solver": "Gillespie",
    "prng_seed": 0,
    "samples": 50,
    "output": {
        "prefix": "SSA",
        "writecsv": true,
        "compress": false,
        "writeMatFile": true,
        "writeSampleTimes": false
    }
}
```

GillespieFirstReaction

The Gillespie First Reaction solver implements the First Reaction Method (FRM) version of the Gillespie algorithm[1]. The other version, the Direct Method (DM), is implemented in *Gillespie (SSA)*. While the two methods are theoretically equivalent, they differ in implementation. FRM generates a potential reaction time for every reaction and chooses the reaction corresponding to the earliest reaction time, thus *first reaction*. In DM implementation, the next reaction time is generated first. Then a reaction is chosen after considering the likelihood of each reaction firing at the computed time.

Parameter	Data type	Description
solver	string	**First**, **FirstReaction**, and **GillespieFirstReaction** are all valid names to run this solver.

Example

```
{
    "duration": 100,
    "runs": 10000,
    "solver": "First",
    "prng_seed": 10,
    "samples": 120,
    "output": {
        "prefix": "FirstReaction",
        "writecsv": true,
        "compress": false,
        "writeMatFile": false,
        "writeSampleTimes": false
    }
}
```

[1] Gillespie, Daniel T. "Exact stochastic simulation of coupled chemical reactions." The Journal of Physical Chemistry 81.25 (1977): 2340-361.

Gibson-Bruck next reaction

The Gibson-Bruck next reaction method[1] is an exact stochastic simulation algorithm that is more efficient than the standard *Gillespie (SSA)*. Specifically, this method uses only a single random number per simulation event. Also, the simulation time is proportional to the logarithm of the number of reactions. The implementation of the next reaction method in this framework is based on the work by Gibson et. al.[1]

Parameter	Data type	Description
solver	string	**Next**, **NextReaction**, and **GibsonBruck** are all valid names to run this solver.

Example

```
{
    "duration" : 365,
    "runs"     : 512,
    "solver" : "NextReaction"
}
```

Hybrid SSA

An event queue SSA hybrid is an algorithm that combines a *Gillespie (SSA)* and an *event queue*[1][2] . For the SSA, the rates associated with the chemical reactions are based on a fundamental observation that the reactions occur at an average rate. In epidemiology, this assumption may hold well for certain state transitions (for example, susceptible to infected), but not others (for example, infected to recovered).

In the latter case, the hybrid algorithm uses the SSA for the transitions that are similar to chemical reactions, whereas event queues are used for the other types of transitions such as delays. For example, the event queue could be utilized for a fixed recovery time of individuals from the infected state. The combination of these two algorithms allow for a greater range of disease models to be implemented by the compartmental modeling structure.

Parameter	Data type	Default	Description
solver	string	NA	**Hybrid** is the only valid name to run this solver.
method	string	RejectionMethod	**RejectionMethod** and **ExactMethod** are names of two methods offered by the Hybrid SSA. The **RejectionMethod** is based on the algorithm described in[1] . The **ExactMethod** is based on the algorithm described in[2] . Technically, both methods are exact; there are no approximations. However, the **ExactMethod** algorithm requires fewer random number generations.

[1] Gibson, B. and Bruck, J. "Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels." The Journal of Physical Chemistry A, 104 9 (2000): 1876-1889.

[1] Barrio, M., Burrage, K., Leier, A. and Tian, T. "Oscillatory Regulation of Hes1: Discrete Stochastic Delay Modelling and Simulation." (2006) PLoS Computational Biology 2 (9): e117.

[2] Cai, X. "Exact stochastic simulation of coupled chemical reactions with delays." The Journal of Chemical Physics 126 (2007): 124108.

Example

```
{
    "duration" : 365,
    "runs"   : 512,
    "solver" : "Hybrid",
    "hybrid" : {
        "method" : "RejectionMethod"
    }
}
```

Approximate methods

The following methods sacrifice accuracy for computational efficiency.

BLeaping

The BLeaping solver implements an explicit tau leaping method with a fixed step size[1]. Here, the fixed step size is assumed to be small enough such that propensity function values do not change dramatically between time steps.

Skipping tau selection and thus never reverting back to *Gillespie (SSA)* may speed up the simulation time. However, accuracy is expected to be low, and it is possible to reach negative population. When the latter phenomenon occurs, the BLeaping solver sets the corresponding population to zero and displays a warning message recommending that you reduce the time step size.

Parameter	Data type	Default	Description
solver	string	NA	**B**, **BLeap**, and **BLeaping** are all valid names to run this solver.
Tau	float	0.1	The size of the fixed time step used throughout the simulation.

Example

```
{
    "duration": 5,
    "runs": 5000,
    "solver": "B",
    "samples": 50,
    "prng_seed": 123,
    "b-leaping": {
        "Tau"  : 0.01
    }
}
```

[1] Gillespie, Daniel T. "Approximate accelerated stochastic simulation of chemically reacting systems." Journal Of Chemical Physics 115 4 (2001)

RLeaping

RLeaping[1] is a *solver* developed for speeding up the *Gillespie (SSA)*[2]. In the standard stochastic simulation algorithm (SSA), each reaction is simulated individually. In RLeaping, reactions are grouped together and executed at the same time. While leaping methods are approximate, they result in faster simulations.

RLeaping can be supplied with four parameters, but we recommend that you do not change the default values unless there is reason to do so. You can speed up the simulation time by increasing **epsilon**, but the accuracy of the method will decrease.

Parameter	Data type	Default	Description
solver	string	NA	**R** and **RLeaping** are both valid names to run this solver.
epsilon	float	0.01	Determines the error of the approximation; accepts values greater than 0 and much less than 1. A value of close to 0 is equivalent to a standard stochastic simulation and a value close to 1 is the most aggressive speedup (and largest error). We do not recommend changing this value.
theta	float	0	Controls the time step selection; accepts values between 0 and 1. A value of 0 is the most conservative and will limit the occurrence of a negative species value.
sorting interval	float	365	Sorts the reaction propensities according to this time interval; accepts positive values. To disable sorting, set the sorting interval greater than or equal to the simulation time.
verbose	bool	false	If true, extra information is printed to the command line, which can be useful for debugging or testing the solver.

Example

```
{
    "duration" : 365,
    "runs"   : 512,
    "solver" : "RLeaping",
    "r-leaping" : {
        "epsilon" : 0.01,
        "theta" : 0.0,
        "sorting interval" : 365.0,
        "verbose" : false
    }
}
```

[1] Anne Auger, Philippe Chatelain, and Petros Koumoutsakos, "R-leaping: Accelerating the stochastic simulation algorithm by reaction leaps" The Journal of Chemical Physics 125 8 (2006), 084103.

[2] Gillespie, Daniel T. "Exact stochastic simulation of coupled chemical reactions" The Journal of Physical Chemistry 81.25 (1977): 2340-361.

Tau-leaping

Tau-leaping[1] was developed by Gillespie to increase the computational speed of the SSA, which is an exact method. Instead of computing the time to every reaction, this algorithm approximates the process and attempts to leap in time, executing a large number of reactions in a period *tau*. This algorithm is computationally faster; however, the approximation removes the "exact" connection to the solution of the (master equation-based methods) for the system.

The implementation of tau-leaping in CMS is based on a work by Cao et. al[2]. This modified tau-leaping algorithm helps avoid the possibility of creating negative species counts within a compartment.

Parameter	Data type	Default	Description
solver	string	NA	**Tau** and **TauLeaping** are both valid names to run this solver.
epsilon	float	0.001	Computes the largest time step tau that is not likely to result in propensity function changes by more than **epsilon** multiplied by the sum of all the propensities. For larger values of tau, the step sizes will also be larger.
Nc	integer	2	A threshold to separate critical and noncritical reactions. A critical reaction is any reaction that is at risk for driving the count of a species below zero; all reactions become critical if **nc** is extremely large, reducing to the exact *Gillespie (SSA)*. Alternatively, if **nc** is zero, there will not be any critical reactions, reducing it to *Tau-leaping*.
Multiple	integer	10	A threshold that determines whether to execute a series of SSA steps instead of a tau-leap. If a leap value of tau is chosen (from the noncritical reaction rates) such that it is less than the **multiple** times 1/(total propensity rate), than the SSA steps are performed.
SSARuns	integer	100	The number of SSA runs that are performed when the proposed leap size from the noncritical reactions is less than **multiple** times 1/(total propensity rate).

Example

```
{
    "duration" : 365,
    "runs"   : 512,
    "solver" : "TauLeaping",
    "tau-leaping" : {
        "epsilon" : 0.001,
        "Nc" : 2,
        "Multiple" : 10.0,
        "SSARuns" : 100
    }
}
```

[1] Gillespie, D T. et al. "Approximate accelerated stochastic simulation of chemically reacting systems." The Journal of Chemical Physics 115 4 (2001): 1716.

[2] Cao, Y. et al. "Avoiding negative populations in explicit Poisson tau-leaping." The Journal of Chemical Physics 123 (2005): 054104.

Spatial simulation methods

These methods can simulate a spatial model.

Diffusive finite state projection (DFSP)

Diffusive finite state projection (DFSP)[1] is a *solver* developed for simulating diffusion processes in compartmental models. Diffusion events are modeled as particles that transition to neighboring locales. DFSP is ideally suited for systems with a small number of particles, on the order of tens or hundreds of particles per locale.

Diffusion solver errors are all first-order in time and second-order in space, and are therefore similar to using one of the leaping algorithms for the simulation of diffusion processes. However, the diffusion methods execute single reaction events per time step, as opposed to leaping algorithms that execute multiple reaction events per time step, and are therefore useful if you choose to capture detailed events.

Parameter	Data type	Default	Description
solver	string	NA	**DFSP**, **DiffusionFSP**, and **TransportFSP** are all valid names to run this solver.
epsilon	float	0.01	Determines the error of the approximation; accepts values between 0 and 1. A value of close to 0 is equivalent to a *Gillespie (SSA)* simulation and a value close to 1 is the most aggressive speedup (and largest error). We do not recommend changing this value.
verbose	bool	false	If true, extra information is printed to the command line, which can be useful for debugging or testing the solver.
uMax	integer	120	The maximum number of particles per subvolume without violating the error condition. Maximum value is 150; if your problem is expected to have more particles, consider using *TransportSSA (ISSA)* instead.

Example

The .cfg file example below is followed by a portion of an an .emodl file to show how diffusive events are specified. **D** represents the diffusion coefficient and that the reactions specify transitions of species A to neighboring locales.

```
{
    "duration" : 1024,
    "runs"   : 512,
    "solver" : "DFSP",
    "dfsp" : {
        "umax" : 120,
        "verbose" : false
    }
}
```

```
(locale site-1)
(set-locale site-1)
(species A::1 1000)
```

(continues on next page)

[1] Brian Drawert, Michael J Lawson, Linda Petzold, Mustafa Khammash, "The diffusive finite state projection algorithm for efficient simulation of the stochastic reaction-diffusion master equation". The Journal of Chemical Physics 132 (2010): 074101

(continued from previous page)

```
(locale site-2)
(set-locale site-2)
(species A::2 1000)

(reaction A1->A2 (A::1) (A::2) (* D A::1))
(reaction A2->A1 (A::2) (A::1) (* D A::2))
```

FractionalDiffusion (FD)

FractionalDiffusion (FD)[1] is a *solver* developed for simulating heavy-tailed diffusion events in compartmental models. As opposed to standard diffusion events that model transitions to neighboring locales, FD allows transitions to any locale irrespective of how far away it is. The probability of a particle jumping to a distant locale is small but non-zero.

The method can use several parameters, but the most important for modeling physical processes are alpha and Dalpha.

Parameter	Data type	Default	Description
solver	string	NA	**FractionalDiffusion**, **Fractional**, **FD**, **Levy**, and **LevyFlight** are all valid names to run this solver.
alpha	float	1	Determines the value of the fractional derivative; accepts values greater than 0 and less than or equal to 2. A value close to 0 results in a diffusive process with very large kurtosis. A value close to 2 results in Gaussian-like diffusion. The default value results in a Cauchy distribution.
Dalpha	float	1	The diffusion coefficient; accepts values between 0 and infinity.
h	float	1	The physical distance between locales; accepts values greater than zero and less than infinity.
constant	float	0.25	The Courant–Friedrichs–Lewy (CFL) condition for parabolic partial differential equations. that is used for meeting the time step criteria; accepts values greater than 0 but much less than 1.
truncation	integer	number of locales/4	The maximum number of locales a particle can jump; accepts values greater than 1 and less than the size of the domain divided by 2. To ensure that particles remain far away from the boundary, the default value is the number of locales divided by 4. We do not recommend changing the default value. For details regarding truncation and boundary effects, see[1].
verbose	bool	false	If true, extra information is printed to the command line, which can be useful for debugging or testing the solver.

[1] Basil Bayati, "Fractional diffusion-reaction stochastic simulations". Journal of Chemical Physics 138 10 (2013): 104117.

Example

Diffusive events are not specified since the solver assumes that all species can diffusive anywhere in the domain. Therefore, the .emodl files should just include the reaction events.

```
{
    "duration" : 10,
    "runs"     : 1,
    "samples"  : 100,
    "solver"   : "FD",
    "fd" : {
        "alpha"    : 1.6,
                     "constant" : 0.025,
        "Dalpha"   : 1.0,
        "verbose"  : true
    }
}
```

TransportSSA (ISSA)

Inhomogeneous Stochastic Simulation Algorithm or ISSA[1], is a solver developed for simulating diffusion processes in compartmental models. Diffusion events are modeled as particles that transition to neighboring locales. The ISSA method is ideally suited for systems with a large number of particles, on the order of thousands of particles or more at each locale.

Diffusion solver errors are all first-order in time and second-order in space, and are therefore similar to using one of the leaping algorithms for the simulation of diffusion processes. However, the diffusion methods execute single reaction events per time step, as opposed to leaping algorithms that execute multiple reaction events per time step, and are therefore useful if you choose to capture detailed events.

Parameter	Data type	Default	Description
solver	string	NA	** ITSSA**, **TransportSSA**, **DiffusionSSA**, and **TSSA** are all valid names to run this solver.
epsilon	float	0.01	Determines the error of the approximation; accepts values greater than 0 and much less than 1. A value of close to 0 is equivalent to a *Gillespie (SSA)* simulation and a value close to 1 is the most aggressive speedup (and largest error). We do not recommend changing this value.
greenFunctionIterations	integer	100	The number of iterations used to compute the fundamental solution of the diffusion equation; accepts values between 1 and infinity.
verbose	bool	false	If true, extra information is printed to the command line, which can be useful for debugging or testing the solver.

[1] S. Lampoudi, D.T. Gillespie, & L.R. Petzold, "The multinomial simulation algorithm for discrete stochastic simulation of reaction-diffusion systems". Journal of Chemical Physics 130 9 (2009): 094104.

Example

The .cfg file example below is followed by a portion of an an .emodl file to show how diffusive events are specified. **D** represents the diffusion coefficient and that the reactions specify transitions of species A to neighboring locales.

```
{
    "duration" : 1024,
    "runs"   : 512,
    "solver" : "issa",
    "tssa" : {
        "epsilon" : 0.01,
        "greensFunctionIterations" : 100,
        "verbose" : false
    }
}
```

```
(locale site-1)
(set-locale site-1)
(species A::1 1000)

(locale site-2)
(set-locale site-2)
(species A::2 1000)

(reaction A1->A2 (A::1) (A::2) (* D A::1))
(reaction A2->A1 (A::2) (A::1) (* D A::2))
```

Rare event probability estimation

These methods are designed for efficient estimation of rare event probabilities.

Doubly weighted stochastic simulation algorithm (dwSSA)

The doubly weighted stochastic simulation algorithm (dwSSA)[1] solver is developed solely for estimating rare event probabilities and thus should not be used for recording time-course trajectories.

dwSSA requires a set of biasing parameters to reach the rare event of interest. If a set of biasing parameters is not provided in the .cfg configuration file, dwSSA will execute multilevel cross- entropy (CE) method prior to the dwSSA simulation to obtain optimal (minimum CE) biasing parameters. The solver then employs these biasing parameters in selecting the next reaction and the next time step, yielding a trajectory weight that is a product of likelihood ratios from importance sampling. For the successful trajectories that reach the rare event, these weights are used to compute an unbiased estimator of the rare event probability with a confidence interval[2].

[1] Daigle, Bernie J et al. "Automated estimation of rare event probabilities in biochemical systems." The Journal of Chemical Physics 134 4 (2011): 04410.

[2] Gillespie, Dan T et al. "Refining the weighted stochastic simulation algorithm." The Journal of Chemical Physics 130 17 (2009): 174103.

Parameter	Data type	Default	Description
solver	string	NA	**dwSSA** is the only valid name to run this solver.
reExpressionName	string	If unspecified, the solver searches the *model file* for **reExpression**.	The name of the function that defines the rare event expression in the model file.
reValName	string	If unspecified, the solver searches the *model file* for **ReVal**.	The name of the parameter that defines the rare event value in the model file.
gammas	vector of floats	Multilevel cross entropy (CE) simulations are performed to compute optimal gamma values prior to dwSSA simulations.	The positive real numbers provided in the vector are used as importance sampling parameters in selecting the next reaction and the next time step, as well as in computing the likelihood ratio of a biased trajectory. The length of the vector is equal to the total number of reactions in the model.
crossEntropyRuns	integer	100,000	The number of trajectories simulated in each level of multilevel CE simulations (not required for dwSSA simulations). Accepts values greater than or equal to 5000. If **crossEntropyRuns** * **crossEntropyThreshold** is less than **crossEntropyMinDataSize**, the value of **crossEntropyRuns** is dynamically adjusted to be the smallest integer greater than **crossEntropyMinDataSize / crossEntropyThreshold**.
crossEntropyThreshold	float	0.01	The fraction of top-performing trajectories chosen to compute an intermediate rare event in multilevel CE simulations (not required for dwSSA simulations). Accepts values between 0 and 1. **Note:** If slow convergence is detected during the multilevel CE simulations, the value is decreased to 80% of its previous value.
crossEntropyMinDataSize	integer	200	The minimum number of successful trajectories required to compute an intermediate rare event for multilevel CE simulations. Accepts values greater than or equal to 100.
outputFileName	string	File name in the form of <modelname>_dwSSA_1e<log(runs)>_log(<runs>), where the base of <log> is 10 and <modelName> is the name of the model file.	The name of the output file that includes runs, estimates for the rare event probability, 68% uncertainty, and sample variance.

Example

```
{
    "duration": 10,
    "runs": 1000000,
    "solver": "dwSSA",
    "dwSSA": {
        "gamma": [1.53, 0.45],
        "reValName": "reVal",
        "reExpressionName": "reExpression",
        "crossEntropyRuns": 100000,
        "crossEntropyThreshold": 0.01,
        "crossEntropyMinDataSize": 300,
        "outputFileNate": "SIR_dwSSA_1e6.txt"
    }
}
```

State-dependent doubly weighted stochastic simulation algorithm (sdwSSA)

Similar to the *Doubly weighted stochastic simulation algorithm (dwSSA)* solver, the state-dependent doubly weighted stochastic simulation algorithm (sdwSSA)[1] is developed exclusively for estimating rare event probabilities and should not be used for recording time-course trajectories.

sdwSSA employs state-dependent importance sampling using a set of parameters for each reaction in the model. If the biasing parameters are not provided in a separate JSON file, sdwSSA will execute multilevel cross-entropy (CE) method prior to the sdwSSA simulation to obtain optimal (minimum CE) biasing parameters. The overall flow of the algorithm is the same as the *Doubly weighted stochastic simulation algorithm (dwSSA)* solver. After sdwSSA simulations finish, an estimate of the rare event with a confidence interval are returned as output[2].

[1] Roh, Min K. et al. "State-dependent doubly weighted stochastic simulation algorithm for automatic characterization of stochastic biochemical rare events." The Journal of Chemical Physics 135 (2011): 234108.

[2] Gillespie, Dan T et al. "Refining the weighted stochastic simulation algorithm." The Journal of Chemical Physics 130 17 (2009): 174103.

Parameter	Data type	Default	Description
solver	string	NA	**sdwSSA** is the only valid name to run this solver.
reExpressionName	string	If unspecified, the solver searches the *model file* for **reExpression**.	The name of the function that defines the rare event expression in the model file.
reValName	string	If unspecified, the solver searches the model file for **ReVal**.	The name of the parameter that defines the rare event value in the model file.
gammaSize	integer	15	The initial length of importance sampling parameters *per reaction*; accepts positive values.
binCount	integer	20	The minimum number of data required to maintain a single bin; bins are otherwise merged until each bin contains at least the value in **binCount**. Used only for multilevel CE simulations; accepts positive values greater than or equal to 10.
biasingParametersFileName	string	File name in the form of <modelName>_biasingParameters.json.	The name of the JSON file containing importance sampling (IS) parameters.
crossEntropyRuns	integer	100,000	The number of trajectories simulated in each level of multilevel CE simulations (not required for dwSSA simulations). Accepts values greater than or equal to 5000. If **crossEntropyRuns** * **crossEntropyThreshold** is less than **crossEntropyMinDataSize**, the value of **crossEntropyRuns** is dynamically adjusted to be the smallest integer greater than **crossEntropyMinDataSize / crossEntropyThreshold**.
crossEntropyThreshold	float	0.01	The fraction of top-performing trajectories chosen to compute an intermediate rare event in multilevel CE simulations (not required for dwSSA simulations). Accepts values between 0 and 1. **Note:** If slow convergence is detected during the multilevel CE simulations, the value is decreased to 80% of its previous value.
crossEntropyMinDataSize	integer	200	The minimum number of successful trajectories required to compute an intermediate rare event for multilevel CE simulations. Accepts values greater than or equal to 100.
outputFileName	string	File name in the form of <modelname>_sdwSSA_log<log(runs)>, where the base of <log> is 10 and <modelName> is the name of the model file.	The name of the output file that includes runs, estimates for the rare event probability, 68% uncertainty, and sam-

Example

```
{
    "duration": 70,
    "runs": 100000,
    "solver": "sdwSSA",
    "sdwSSA": {
        "reExpressionName": "reExpression",
        "reValName": "SIR-RE-val",
        "crossEntropyThreshold"  : 0.01,
        "crossEntropyMinDataSize": 300,
        "crossEntropyRuns": 100000,
        "gammaSize": 20,
        "binCount": 15,
        "biasingParametersFileNate": "SIRS_biasingParameters_custom.json",
        "outputFileNate": "SIR_sdwSSA_1e5.txt"
    }
}
```

Exploratory methods

The CMS framework is designed to enable efficient prototyping of new methods. A new solver can be easily implemented by extending the base solver. Below we list four prototype methods included in the CMS. We note that solvers in this category are included as an example of ongoing method development and are not currently supported by the developers.

ExitTimes

ExitTimes[1] is a *solver* developed for computing the time it takes for a specified event to occur. For example, you may want to compute the time it takes for the infectious population to reach zero. This method will usually be faster than *Gillespie (SSA)*. The method attempts to group the propensities into sets, approximate their values, and sample the final time from multiple gamma distributions. See[1] for a detailed derivation.

Parameter	Data type	Default	Description
solver	string	NA	**ExitTimes**, **ET**, and **ExitTime** are all valid names to run this solver.
epsilon	float	0.2	Determines the error of the approximation; accepts values between 0 and 1. A value of close to 0 is equivalent to a *Gillespie (SSA)* simulation and a value close to 1 is the most aggressive speedup (and largest error). We do not recommend changing this value.
verbose	bool	false	If true, extra information is printed to the command line, which can be useful for debugging or testing the solver.

[1] Basil Bayati, "A Method to Calculate the Exit Time in Stochastic Simulations"

Example

The .cfg file example below is followed by a portion of an an .emodl file to show how exit time events are specified.

```
{
    "duration" : 30,
    "runs"     : 20000,
    "solver"   : "ET",
    "et" : {
              "verbose" : false,
              "epsilon" : 0.2
    }
}
```

```
; sir model with exit condition

(import (rnrs) (emodl cmslib))

(start-model "sir-exit")

(locale site-a)
(set-locale site-a)

(species S 95)
(species I 5)
(species R 0)

(observe Susceptible S)
(observe Infected I)
(observe Recovered R)


(param beta 0.015)
(param gamma 1.0)

(reaction S->I (S) (I) (* beta S I))
(reaction I->R (I) (R) (* gamma I))

(bool exitTimeEvent (== R 85) )


(end-model)
```

Midpoint tau-leaping

The midpoint tau-leaping algorithm is a modification of *Tau-leaping*[1]. Instead of using the current state of the system to evaluate the propensity functions, an estimated midpoint state is constructed. Then, this midpoint state is used to evaluate the propensity functions from the current time **t**. This modification has a direct analogy in the simulation of *deterministic* systems.

[1] Gillespie, D. T. et al. "Approximate accelerated stochastic simulation of chemically reacting systems." The Journal of Chemical Physics 115 4 (2001): 1716.

Parameter	Data type	Default	Description
solver	string	NA	**Tau** and **TauLeaping** are both valid names to run this solver.
epsilon	float	0.01	Computes the largest time step tau that is not likely to result in propensity function changes by more than **epsilon** multiplied by the sum of all the propensities. For larger values of tau, the step sizes will also be larger.
nc	integer	2	A threshold to separate critical and noncritical reactions. A critical reaction is any reaction that is at risk for driving the count of a species below zero; all reactions become critical if **nc** is extremely large, reducing to the exact *Gillespie (SSA)*. Alternatively, if **nc** is zero, there will not be any critical reactions, reducing it to *Tau-leaping*.
multiple	integer	10	A threshold to decide on whether to execute a series of SSA steps instead of a tau-leap. If a leap value of tau is chosen (from the noncritical reaction rates) such that it is less than the **multiple** times 1/(total propensity rate), than the SSA steps are performed.
SSAruns	integer	100	The number of SSA runs that are performed when the proposed leap size from the noncritical reactions is less than **multiple** times 1/(total propensity rate).

Example

```
{
    "duration" : 365,
    "runs"   : 1000,
    "solver" : "MidPoint",
    "midpoint" : {
        "epsilon" : 0.001,
        "nc" : 2,
        "multiple" : 10.0,
        "SSAruns" : 100
    }
}
```

RLeapingFast

RLeapingFast[1][2] is a solver developed for speeding up the *Gillespie (SSA)*[3]. In the standard SSA, each reaction is simulated individually. The difference between RLeaping[1] and RLeapingFast[2] is the computation of how many reactions to leap over, which is likely to be quicker with RLeapingFast[2]. This method uses the time step computation developed for tau-leaping[2] and recasts it for use in RLeapingFast. While leaping methods are approximate, they result in faster simulations.

RLeaping can be supplied with four parameters, but we recommend that you do not change the default values unless there is reason to do so. You can speed up the simulation time by increasing **epsilon**, but the accuracy of the method

[1] Anne Auger, Philippe Chatelain, and Petros Koumoutsakos, "R-leaping: Accelerating the stochastic simulation algorithm by reaction leaps" The Journal of Chemical Physics 125 8 (2006): 084103.

[2] Yang Cao, Daniel T. Gillespie, and Linda R. Petzold "Efficient step size selection for the tau-leaping simulation method" Journal of Chemical Physics 124 4 (2006): 044109

[3] Gillespie, Daniel T. "Exact stochastic simulation of coupled chemical reactions" The Journal of Physical Chemistry 81.25 (1977): 2340-361.

will decrease.

Parameter	Data type	Default	Description
solver	string	NA	**RF**, **RFast**, and **RLeaping** are all valid names to run this solver.
epsilon	float	0.01	Determines the error of the approximation; accepts values greater than 0 and much less than 1. A value of close to 0 is equivalent to a *Gillespie (SSA)* simulation and a value close to 1 is the most aggressive speedup (and largest error). We do not recommend changing this value.
theta	float	0	Controls the time step selection; accepts values between 0 and 1. A value of 0 is the most conservative and will limit the occurrence of a negative species value.
sorting interval	float	365	Sorts the reaction propensities according to this time interval; accepts positive values. To disable sorting, set the sorting interval greater than or equal to the simulation time.
verbose	bool	false	If true, extra information is printed to the command line, which can be useful for debugging or testing the solver.

Example

```
{
    "duration" : 365,
    "runs"     : 512,
    "solver" : "RLeapingFast",
    "r-leaping" : {
        "epsilon" : 0.01,
        "sorting interval" : 365.0,
        "verbose" : false
    }
}
```

OptimalTransportSSA (OTSSA)

The OptimalTransport diffusion (OTSSA) solver attempts to automatically choose between the *TransportSSA (ISSA)*[1] and *Diffusive finite state projection (DFSP)*[2] solvers. Diffusion events are modeled as particles that transition to neighboring locales. The ISSA method is ideally suited for systems with a large number of particles, on the order of thousands of particles or more at each locale. On the other hand, DFSP is ideally suited for systems with a small number of particles, on the order of tens or hundreds of particles. OTSSA will attempt to dynamically choose which of the above solvers to use per time step. This solver is recommended if you do not know whether the system contains a small or large number of particles, or if you know that the system will evolve with both large and small populations.

Diffusion solver errors are all first-order in time and second-order in space, and are therefore similar to using one of the leaping algorithms for the simulation of diffusion processes. However, the diffusion methods execute single reaction events per time step, as opposed to leaping algorithms that execute multiple reaction events per time step, and are therefore useful if you choose to capture detailed events.

[1] S. Lampoudi, D.T. Gillespie, & L.R. Petzold, "The multinomial simulation algorithm for discrete stochastic simulation of reaction-diffusion systems". Journal of Chemical Physics 130 9 (2009): 094104.

[2] Brian Drawert, Michael J Lawson, Linda Petzold, Mustafa Khammash, "The diffusive finite state projection algorithm for efficient simulation of the stochastic reaction-diffusion master equation". The Journal of Chemical Physics 132 (2010): 074101

Parameter	Data type	Default	Description
solver	string	NA	**OTSSA**, **OptimalTransportSSA**, and **DFSPPrime** are all valid names to run this solver.
epsilon	float	0.01	Determines the error of the approximation; accepts values greater than 0 and much less than 1. A value of close to 0 is equivalent to a *Gillespie (SSA)* simulation and a value close to 1 is the most aggressive speedup (and largest error). We do not recommend changing this value.
greenFunctionIterations	integer	100	The number of iterations used to compute the fundamental solution of the diffusion equation; accepts values between 1 and infinity.
transportSSAThreshold	integer	uMax * graphDimension, where graphDimension is the dimension of the diffusion problem (calculated internally using the number of neighbors)	The threshold for choosing when to run TransportSSA vs. DFSP. We do not recommend changing the default value.
verbose	bool	false	If true, extra information is printed to the command line, which can be useful for debugging or testing the solver.

Example

The .cfg file example below is followed by a portion of an an .emodl file to show how diffusive events are specified. **D** represents the diffusion coefficient and that the reactions specify transitions of species A to neighboring locales.

```
{
    "duration" : 1024,
    "runs"   : 512,
    "solver" : "otssa",
    "dfsp" : {
        "umax" : 120,
        "verbose" : false
    },
    "otssa" : {
        "verbose" : false
    },
    "tssa" : {
        "epsilon" : 0.01,
        "greensFunctionIterations" : 100,
        "verbose" : false
    }
}
```

```
(locale site-1)
(set-locale site-1)
(species A::1 1000)
```

(continues on next page)

(continued from previous page)

```
(locale site-2)
(set-locale site-2)
(species A::2 1000)

(reaction A1->A2 (A::1) (A::2) (* D A::1))
(reaction A2->A1 (A::2) (A::1) (* D A::2))
```

Output

By default, trajectories.csv will be created in the output directory, with the realization index appended to each observable name in the file. If you prefer, you have the option of creating JSON or MATLAB files that contain the output of a simulation.

Parameter	Data type	Default	Description
channeltitles	bool	false	Specifies whether or not to populate the **ChannelTitles** array in JSON output files. If set to true, the entries of the **ChannelTitles** pair with the channel data in the **ChannelData** array. Each entry in **ChannelTitles** consists of an observable name followed by the realization number in curly braces, for example `susceptible{0}`.
compress	bool	false	Specifies whether or not CSV and JSON output files should be compressed using gzip or MATLAB .MAT files with internal compression. This can be useful for large data files.
newmatformat	bool	false	Specifies whether to use the original MATLAB schema or the new MATLAB schema. The original schema included four elements in the .MAT file: **version** A string. **sampletimes** A matrix of sample times, with one row and columns equal to the number of samples. **observables** A matrix of the names of observable quantities, with one column and rows equal to the number of observables. **data** A matrix with rows equal to the number of observables times the number of realizations and columns equal to the number of samples. The new schema has the same **version** and **sampletimes** elements, but combines the observables and data elements into **observable1** through **observableN** (each entry is a matrix with rows equal to the umber of realizations and columns equal to the number of samples).
prefix	string	trajectories	Specifies the main name of the output files to be written, minus the file extension.
writecsv	bool	true	Specifies whether or not to write realization data in CSV format.
writejson	bool	false	Specifies whether or not to write realization data in JSON format. See any example of this type of output format *Example*.
writematfile	bool	false	Specifies whether or not to write realization data in MATLAB .mat format.
writerealizationindex	bool	true	Specifies whether or not to add a suffix with the realization index to each observable name in a CSV file.

Example

The example configuration file below shows one way you can configure CMS to create JSON output files.

```
{
    "runs": 1,
    "duration": 365,
    "samples": 100,
    "solver": "SSA",
    "output": {
```

(continues on next page)

(continued from previous page)

```
        "prefix": "test_trajectories",
        "writecsv": false,
        "writejson": true,
        "compress": false,
        "channeltitles": true,
        "writematfile": false,
        "newmatformat": false
    }
}
```

The test_trajectories.json file below shows the type of output you can expect from the example configuration file above.

```
{
    "FrameworkVersion": "1.0.176.23",
    "BuildDescription": "clorton (CMS/Main/framework)",
    "Runs": 1,
    "Samples": 100,
    "ObservableNames": [
        "susceptible",
        "exposed",
        "infectious",
        "recovered"
    ],
    "ChannelTitles": [
        "susceptible{0}",
        "exposed{0}",
        "infectious{0}",
        "recovered{0}"
    ],
    "SampleTimes": [
        0.0,
        1.010101,
        2.020202,
        "etc.",
        98.9899,
        100.0
    ],
    "ChannelData": [
        [
            990.0,
            979.0,
            974.0,
            973.0,
            "etc.",
            281.0,
            286.0
        ],
        [
            0.0,
            11.0,
            15.0,
            14.0,
            "etc.",
            5.0,
            3.0
        ],
        [
```

(continues on next page)

(continued from previous page)

```
            10.0,
            9.0,
            8.0,
            6.0,
            "etc.",
            6.0,
            7.0
        ],
        [
            0.0,
            1.0,
            3.0,
            7.0,
            "etc.",
            708.0,
            704.0
        ]
    ],
    "ObservablesCount": 4
}
```

Random number library

CMS supports four different pseudo-random number generators (PRNG) that can be specified in the configuration file.

Parameter	Data type	Default	Description
RNG	enum	AESCOUNTER. If the hardware does not support this, falls back to PSEUDO-DES.	The pseudo-random number generator to use in this simulation. Supported values are: **VANILLA** Based on the .NET Random class. **RANDLIB** Based on the pseudo-random number generator that uses a combination of multiplicative linear congruential generators proposed by Pierre L'Ecuyer[1] and used in numerous scientific computing libraries. **PSEUDODES** Based on the entropy generating step of the Data Encryption Standard published by NIST. The algorithm is **psdes** described in Numerical Recipes in C. **AESCOUNTER** Similar in concept to **PSEUDODES** and is based on the entropy generating step of the Advanced Encryption Standard published by NIST in 2001. The implementation uses AES Counter Mode to generate a stream a pseudo-random numbers from the given seed values.
rng_seed	integer	0	The value that seeds the generator and is used with **prng_index** to determine its initial state.
rng_index	integer	0	The value that indexes the generator. This can be used to identify different runs of an experiment or to seed different instantiations of the compartmental modeling software across multiple processors.

[1] Efficient and Portable Combined Random Number Generators, Communications of the ACM, June 1988

Example

```
{
    "solver": "SSA",
    "duration": 1000,
    "runs": 3,
    "RNG": { "type": "RANDLIB" },
    "rng_seed" : 2017,
    "rng_index" : 42
}
```

CHAPTER

FOUR

GLOSSARY

chemical master equation (CME) The equation that describes a homogeneously mixed population that can be modeled as a probabilistic combination of states at any given time. Switching between states is determined by a transition rate matrix.

compartmental model In epidemiology, a type of mathematical model in which each disease state is treated as a separate compartment and the individuals within each compartment are assumed to be equivalent.

configuration file The optional JSON syntax file that specifies the duration of each realization, how many realizations to calculate, and other characteristics of the simulation. If this file is not included, the model runs one realization of Gillespie (SSA) for 100 time units. It often uses a .cfg extension.

cross-entropy method A general Monte Carlo approach that is useful for simulations where estimation of very small probabilities is important. It is an iterative method in which a random data sample is generated according to a specified mechanism and then the parameters of the mechanism are updated based on the data to produce a better data sample in the next iteration.

deterministic Characterized by the output being fully determined by the parameter values and the initial conditions. Given the same inputs, a deterministic model will always produce the same output.

EMODL The file format used to specify the model file that defines the species and mathematics of the model. It stands for Epidemiological Model Language and uses syntax similar to LISP.

event queue A data structure that holds events prior to being processed by a receiving program or system.

Gillespie stochastic simulation algorithm (SSA) An exact numerical simulation procedure developed by Dan Gillespie in 1977 to describe homogeneous chemical systems. SSA is a type of continuous-time, discrete-state, Markov chain Monte Carlo (MCMC) method.

JSON (JavaScript Object Notation) A human-readable, open standard, text-based file format for data interchange. It is typically used to represent simple data structures and associative arrays, and is language-independent. For more information, see https://www.json.org.

model file The required .emodl file that defines the model: the different species, the locale, the mathematics that determine transitions, etc. You will often have one model file and many different configuration files.

Monte Carlo method A class of algorithms using repeated random sampling to obtain numerical results. Monte Carlo simulations create probability distributions for possible outcomes, which provides a more realistic way of describing uncertainty.

ordinary differential equation (ODE) A differential equation containing one or more functions of one independent variable and its derivatives.

partial differential equation (PDE) A differential equation containing unknown multivariable functions and their partial derivatives.

propensity function A function that describes the probability of a reaction occurring during the next infinitesimal time interval given the current state.

realization A single pass of a model through a solver with a given (implicit or explicit) random number stream seed. Most models, due to their stochastic nature, should be run multiple times to generate many realizations in order to characterize the distribution of model states.

solver A particular algorithm for advancing the state of a model through simulation time. Variations in CMS solvers are similar to the various methods for numerically solving ordinary differential equations.

species Borrowed from CME-style models in which a species represents an element or molecule, a species in CMS generally represents a unique state in a disease model, such as susceptible, infectious, or recovered.

stochastic Characterized by having a random probability distribution that may be analyzed statistically but not predicted precisely.

INDEX