The Journal of Open Source Software

StoSpa2: A C++ software package for stochastic simulations of spatially extended systems

Bartosz J. Bartmanski[1] **and Ruth E. Baker**[1]

1 Mathematical Institute, University of Oxford, Woodstock Road, Oxford, OX2 6GG, UK

DOI: 10.21105/joss.02293

Software

- Review
- Repository
- Archive

Editor: Pierre de Buyl

Reviewers:

- @CFGrote
- @mbkumar

Submitted: 20 May 2020
Published: 17 June 2020

Summary

Mathematical modelling of complex biological phenomena allows us to understand the contributions of different processes to observed behaviours. Many of these phenomena involve the reaction and diffusion of molecules and so we use so-called reaction-diffusion models to describe them mathematically. Reaction-diffusion models are often subdivided into three types (Hellander & Petzold, 2017): macroscopic, mesoscopic and microscopic. Models that describe a system in terms of concentrations are termed macroscopic models. At the other end of the spectrum we have microscopic models that describe a system by specifying the positions (and often velocities) of each molecule. The middle ground between these two scales is covered by mesoscopic models, in which stochasticity and some individual-level details are included without directly tracking the position of every single molecule. Macroscale models ignore crucial details such as stochastic effects, while microscale models tend to be computationally intensive (Osborne, Fletcher, Pitt-Francis, Maini, & Gavaghan, 2017; Van Liedekerke, Palm, Jagiella, & Drasdo, 2015). Mesoscale models offer a good balance in that they include stochastic effects without incurring enormous computational overheads. The frameworks of the chemical master equation (CME) and its spatial extension, the reaction-diffusion master equation (RDME) (Isaacson, 2009, 2013; Van Kampen, 1992), provide mesoscopic models of reaction and diffusion. However, in the majority of these cases, models built in the CME/RDME framework are analytically intractable and so model behaviours must be explored using stochastic simulation algorithms.

`StoSpa2` is a C++ software package for stochastic simulation of models constructed using the CME and RDME frameworks. This software package allows for efficient simulations with a user friendly interface, and it includes functionality for simulations on both static and growing domains, and time-varying reaction rates.

The primary audience of `StoSpa2` are researchers who wish to model a chemical or biological system using the CME or RDME frameworks.

The software

`StoSpa2` allows for a wide range of stochastic simulations within the CME and the RDME frameworks. Within the RDME framework, the simulations are independent of the mesh type, hence, the simulations can be run on both structured and unstructured meshes. Examples of both structured and unstructured meshes can be seen in Figure 1. Furthermore, `StoSpa2` allows for simulations on a uniformly growing domain, by using the Extrande method (Voliotis, Thomas, Grima, & Bowsher, 2016).

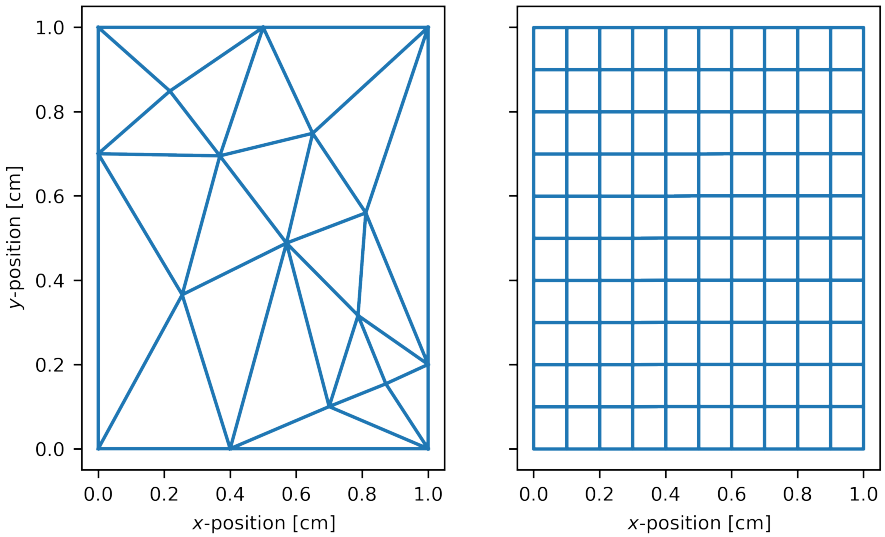

Figure 1: Examples of meshes that can be used within `StoSpa2` for RDME simulations. On the left-hand side is an example of an unstructured mesh, while on the right-hand side is an example of a structured mesh.

The core of the package is written in the C++ programming language to make simulations as efficient as possible. The number of packages need to use `StoSpa2` has been intentionally kept as small as possible to make sure that the software can be used in any computing environment that can compile C++ code.

To make the software user-friendly, the application programming interface (API) has been designed with simplicity in mind. All the details about the API can be found at `StoSpa 2` documentation website (https://stospa2.readthedocs.io). Furthermore, we have included Python bindings, which allow simulations to be run from within the Python programming language. `Pybind11` (https://github.com/pybind/pybind11) is used to create `pystospa`, the Python binding of `StoSpa2`. Having a Python interface for a software package saves the user from having to recompile code themselves for every simulation, making `StoSpa2` easier to use.

The continuous integration platform TravisCI (https://travis-ci.org/) is used to make sure that any changes in the code-base do not break any functionality of `StoSpa2`. Both the installation and the functionality are tested on Linux and OSX operating systems.

Status of the field

The CME and RDME frameworks are used to model various phenomena in fields such as chemistry, epidemiology and systems biology. The goal behind making `StoSpa2` is to facilitate easy and fast simulations of systems modelled using the CME and RDME. There are some alternative software packages that can be used, for example, URDME (Drawert, Engblom, & Hellander, 2012) and MesoRD (Fange, Mahmutovic, & Elf, 2012), however these both have dependencies that can be a barrier to installation and use; URDME has proprietary software as dependencies, while MesoRD has a large number of dependencies that make it challenging to install. StochSS (Drawert et al., 2016) is another software package, and provides a docker container and a web interface. However, docker containers require special privileges to run, which not every user may have, and the web interface does not allow high

throughput execution of simulations. Hence, we developed StoSpa2, which relies upon as few dependencies as possible and is designed to be easily installed. Furthermore, our software can simulate dynamics on a uniformly growing domain using the Extrande method (Voliotis et al., 2016), while the alternatives cannot.

Installation

The code for StoSpa2 is hosted on GitHub (https://github.com/BartoszBartmanski/StoSpa2) with installation instructions contained in the README.md file. The first way of installing StoSpa2 involves downloading the source code from the GitHub repository and compiling the C++ code according to the instructions in the README.md file. However, an easier alternative is to use the Python package manager, pip, to download the Python binding of StoSpa2, pystospa, from https://pypi.org/project/pystospa/ and install it appropriately. All details of the installation, as well as more information, are contained in the documentation website (https://stospa2.readthedocs.io/en/latest/).

Examples

In the following examples, we refer as voxels to the sub-intervals of a domain of the system to be modelled, while with a mesh we refer to how the voxels are organised in space in relation to each other.

Chemical master equation example

As first example, let us consider the following chemical reaction

$$A \xrightarrow{k} \emptyset\,,$$

which occurs at some rate $k\,s^{-1}$ on a domain $\Omega = [0\,cm, 1\,cm]$. We can simulate this chemical system with the following code

```
#include "simulator.hpp"

int main() {
    //// Create voxel object. ////
    // number of molecules of species A
    std::vector<unsigned> initial_num = {100};
    // size of the domain in cm
    double domain_size = 10.0;
    // Arguments: vector of number of molecules, size of the voxel
    StoSpa2::Voxel v(initial_num, domain_size);

    //// Create reaction object. ////
    double k = 1.0;
    auto propensity = [](
        const std::vector<unsigned>& num_mols,
        const double& area)
        { return num_mols[0]; };
    std::vector<int> stoch = {-1};
    // Arguments: reaction rate, propensity func, stoichiometry vector
    StoSpa2::Reaction r(k, propensity, stoch);
```

```
    // Add a reaction to a voxel
    v.add_reaction(r);

    // Pass the voxel with the reaction(s) to the simulator object
    StoSpa2::Simulator s({v});

    // Run the simulation.
    // Arguments: path to output file, time step, number of steps
    s.run("cme_example.dat", 0.01, 500);
}
```

The first line of code in the above example makes sure that we can use the `StoSpa2` classes. In the main function we first define the `Voxel` object that will represent the domain of the system:

```
std::vector<unsigned> initial_num = {100};
double domain_size = 10.0;
StoSpa2::Voxel v(initial_num, domain_size);
```

where we place 100 molecules of species A into a domain of size $1.0\,cm$ (which in our case is a single voxel of size $1.0\,cm$).

In the next segment of the code we create the lambda function that represents the propensity function and the `Reaction` object with a rate of $k = 1.0\,s^{-1}$, the propensity function `propensity` and the stoichiometry vector which decreases the number of molecules by one any time that the decay reaction happens:

```
double k = 1.0;
auto propensity = [](
    const std::vector<unsigned>& num_mols,
    const double& area)
    { return num_mols[0]; };
std::vector<int> stoch = {-1};
StoSpa2::Reaction r(k, propensity, {-1});
```

Though, the reaction propensity function in this case would be ka with a being the number of molecules of A, whereas the above lambda function returns just the number of molecules. This interface was chosen as to not repeat a lambda function definition if similar reactions appear more than once, for example if the reaction happens in multiple voxels. The lambda functions for the reaction propensities have to take two arguments: a vector and a double. The vector represents the number of molecules and the double representing the area of a voxel. We then pass the `Reaction` object to the `Voxel`:

```
v.add_reaction(r);
```

Finally, we run the simulation, which saves the output into `example.dat` file every $0.01\,s$ for 500 steps.

```
StoSpa2::Simulator sim({v});
sim.run("cme_example.dat", 0.01, 500);
```

We can see example output of a simulation in Figure 2.

The Python binding of StoSpa2, called pystospa, allows us to run the same simulation using the Python programming language. The Python code in this case is as follows:

```
import pystospa as ss

# Create voxel object.
# Arguments: vector of number of molecules, size of the voxel
initial_num = [100]  # number of molecules of species A
domain_size = 10.0  # size of the domain in cm
v = ss.Voxel(initial_num, domain_size)

# Create reaction object.
# Arguments: reaction rate, propensity func, stoichiometry vector
k = 1.0
propensity = lambda num_mols, area : num_mols[0]
stoch = [-1]
r = ss.Reaction(k, propensity, stoch)

# Add a reaction to a voxel
v.add_reaction(r)

# Pass the voxel with the reaction(s) to the simulator object
s = ss.Simulator([v])
# Run the simulation.
# Arguments: path to output file, time step, number of steps
s.run("cme_example.dat", 0.01, 500)
```

which has a very similar interface as the C++ code, but has the benefit of not needing any compilation once pystospa is installed.

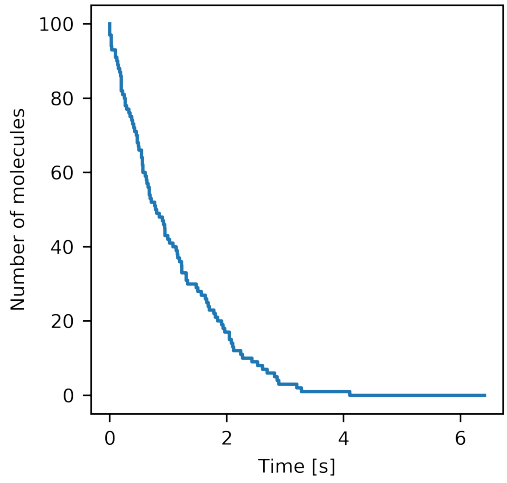

Figure 2: Example of a simulation output for a system modelled using CME framework. A single species of molecules decays at rate $k\,s^{-1}$.

Reaction-diffusion master equation example

To demonstrate how StoSpa2 can be used to run simulations within the RDME framework an example of a one-dimensional domain $[0\,cm, 10\,cm]$, discretised into 10 voxels of equal size $1\,cm$ is used. Within RDME framework, diffusion is modelled as a jump process, and so can be described as a series of reactions as shown below

$$A_1 \overset{d}{\rightleftharpoons} A_2 \overset{d}{\rightleftharpoons} A_3 \overset{d}{\rightleftharpoons} ... \overset{d}{\rightleftharpoons} A_{10}\,,$$

where A_i is the the number of molecules of A in voxel i. The propensity functions for the above diffusion reactions have the following form da_i for a molecule of A to jump from voxel i to either of the neighbouring ones. The C++ code for such a system is as follows:

```
#include "simulator.hpp"

using namespace StoSpa2;

int main() {

    // Create a vector of voxel objects.
    // Arguments: vector of number of molecules, size of the voxel
    std::vector<unsigned> initial_num = {10000};
    double voxel_size = 1.0;
    // First create nine empty voxels
    std::vector<Voxel> vs = std::vector<Voxel>(9, Voxel({0}, voxel_size));
    // Then add the non-empty voxel at the beginning
    vs.insert(vs.begin(), Voxel(initial_num, voxel_size));

    double d = 1.0;  // diffusion rate
    auto propensity = [](
        const std::vector<unsigned>& num_mols,
        const double& area)
    { return num_mols[0]; };
    std::vector<int> stoch = {-1};
    // Create and add the reaction objects
    for (unsigned i=0; i<vs.size()-1; i++) {
        // Add diffusion jump to the right from voxel i to voxel i+1
        vs[i].add_reaction(Reaction(d, propensity, stoch, i+1));
        // Add diffusion jump to the left from voxel i+1 to voxel i
        vs[i+1].add_reaction(Reaction(d, propensity, stoch, i));
    }

    // Pass the voxels with the reaction(s) to the simulator object
    Simulator s(vs);

    // Run the simulation.
    // Arguments: path to output file, time step, number of steps
    s.run("rdme_example.dat", 0.01, 500);
}
```

which is included in the `examples` directory of StoSpa2.

The code is somewhat similar to the chemical master equation example, except we have more `Voxel` objects. We also include the extra line `using namespace StoSpa2` to save us having to write `StoSpa2::` in front of every `StoSpa2` class.

We start by initialising the `Voxel` objects that make up the domain of the system. First, we set the variables that define the number of molecules in the left-most voxel and the size of every voxel. Then, we initialise a vector of nine `Voxel` objects that contain no molecules and we slot an additional `Voxel` object at the beginning of this vector with 10000 molecules:

```
std::vector<unsigned> initial_num = {10000};
double voxel_size = 1.0;
// First create nine empty voxels
std::vector<Voxel> vs = std::vector<Voxel>(9, Voxel({0}, voxel_size));
// Then add the non-empty voxel at the beginning
vs.insert(vs.begin(), Voxel(initial_num, voxel_size));
```

We add reactions to the voxels, where we assume that the voxels are ordered by their position on the x-axis. When adding the diffusion reactions, we have one additional parameter in the `Reaction` class constructors, namely `diffusion_idx`, which is the index of the neighbouring voxel in to which a molecule jumps if a diffusion reaction happens:

```
double d = 1.0;  // diffusion rate
auto propensity = [](
    const std::vector<unsigned>& num_mols,
    const double& area)
{ return num_mols[0]; };
std::vector<int> stoch = {-1};
// Create and add the reaction objects
for (unsigned i=0; i<vs.size()-1; i++) {
    // Add diffusion jump to the right from voxel i to voxel i+1
    vs[i].add_reaction(Reaction(d, propensity, stoch, i+1));
    // Add diffusion jump to the left from voxel i+1 to voxel i
    vs[i+1].add_reaction(Reaction(d, propensity, stoch, i));
}
```

And finally, as in the previous example, we run the simulation with the `Simulator` class instance by passing the vector of `Voxel` objects to it and calling the `Simulator` class run function

```
Simulator s(vs);
s.run("rdme_example.dat", 0.01, 500);
```

which takes the path to a file where to save the data, followed by the size of the time-step and the number of steps to take to finish the simulation. The state of the simulation, initially and at the final time point, is shown in Figure 3 where the molecules diffuse as expected.

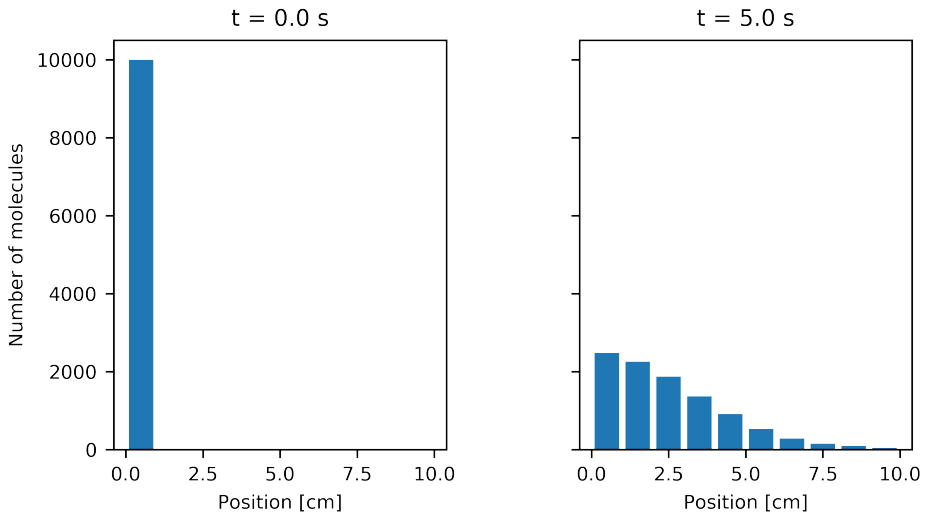

Figure 3: Example of a simulation output for a system modelled using RDME framework. Molecules of a single species jump between the voxels at rate $d = 1.0cm^2s^{-1}$, which compose the whole domain of the system.

Growing domain

StoSpa2 allows for stochastic simulations on a uniformly growing domain. The example from the previous section can be extended to a simulation on a growing domain, by defining the domain of the system using $\Omega(t) = [0, L(t)]$, where $L(t) = L_0 e^{rt}\, cm$. All the voxels growth deterministically according to $h = L_0 e^{rt}/N$ where N is the number of voxels, which doesn't change over the course of a simulation. As in the previous example, we discretise the domain Ω into 10 equally-sized voxels, with $L_0 = 10\, cm$ and $r = 0.2\, s^{-1}$, hence the code for such a simulation is as follows

```
#include "simulator.hpp"

using namespace StoSpa2;

int main() {
    // We define a lambda function that represents the domain growth function
    auto growth = [](const double& t) { return exp(0.2 * t); };

    // Create a vector of voxel objects.
    // Arguments: vector of number of molecules, size of the voxel
    std::vector<unsigned> initial_num = {10000};
    double voxel_size = 1.0;
    // First create nine empty voxels
    std::vector<Voxel> vs = std::vector<Voxel>(9, Voxel({0}, voxel_size, growth));
    // Then add the non-empty voxel at the beginning
    vs.insert(vs.begin(), Voxel(initial_num, voxel_size, growth));

    double d = 1.0;  // diffusion rate
    auto propensity = [](
        const std::vector<unsigned>& num_mols,
```

```
        const double& area)
    { return num_mols[0]; };
    std::vector<int> stoch = {-1};
    // Create and add the reaction objects
    for (unsigned i=0; i<vs.size()-1; i++) {
        // Add diffusion jump to the right from voxel i to voxel i+1
        vs[i].add_reaction(Reaction(d, propensity, stoch, i+1));
        // Add diffusion jump to the left from voxel i+1 to voxel i
        vs[i+1].add_reaction(Reaction(d, propensity, stoch, i));
    }

    // Pass the voxels with the reaction(s) to the simulator object
    Simulator s(vs);

    // Run the simulation.
    // Arguments: path to output file, time step, number of steps
    s.run("rdme_example.dat", 0.01, 500);
}
```

where there are very few differences between this case and the case in the previous section. The main difference being the addition of growth function as a lambda function

```
auto growth = [](const double& t) { return exp(0.2 * t); };
```

and initialising the `Voxel` objects with the growth lambda function as a third argument. The comparison of the simulation output between a static domain and a growing one can be seen in Figure 4, where the molecules have diffused to a space more than twice the size of the initial domain by the end of the simulation.

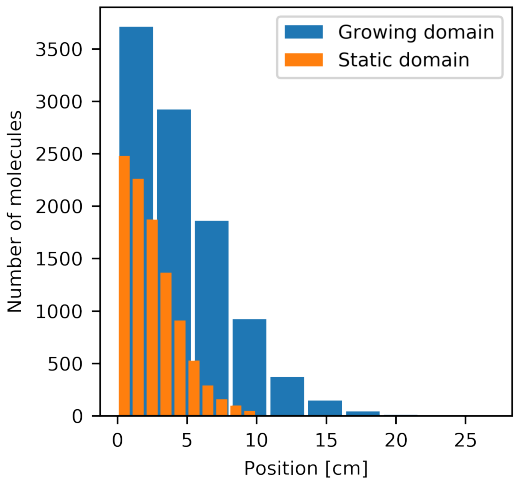

Figure 4: Comparison of a simulation on growing domain with a static one.

Acknowledgements

B.J.B. was supported by the Engineering and Physical Sciences Research Council [grant number EP/G03706X/1]; R.E.B. is a Royal Society Wolfson Research Merit Award holder, would like to thank the Leverhulme Trust for a Research Fellowship.

References

Drawert, B., Engblom, S., & Hellander, A. (2012). URDME: A modular framework for stochastic simulation of reaction-transport processes in complex geometries. *BMC systems biology*, *6*(1), 76. doi:10.1186/1752-0509-6-76

Drawert, B., Hellander, A., Bales, B., Banerjee, D., Bellesia, G., Daigle Jr, B. J., Douglas, G., et al. (2016). Stochastic simulation service: Bridging the gap between the computational expert and the biologist. *PLoS computational biology*, *12*(12). doi:10.1371/journal.pcbi.1005220

Fange, D., Mahmutovic, A., & Elf, J. (2012). MesoRD 1.0: Stochastic reaction-diffusion simulations in the microscopic limit. *Bioinformatics*, *28*(23), 3155–3157. doi:10.1093/bioinformatics/bts584

Hellander, S., & Petzold, L. (2017). Reaction rates for reaction-diffusion kinetics on unstructured meshes. *The Journal of Chemical Physics*, *146*(6), 064101. doi:10.1063/1.4975167

Isaacson, S. A. (2009). The reaction-diffusion master equation as an asymptotic approximation of diffusion to a small target. *SIAM Journal on Applied Mathematics*, *70*(1), 77–111. doi:10.1137/070705039

Isaacson, S. A. (2013). A convergent reaction-diffusion master equation. *The Journal of Chemical Physics*, *139*(5), 054101. doi:10.1063/1.4816377

Osborne, J. M., Fletcher, A. G., Pitt-Francis, J. M., Maini, P. K., & Gavaghan, D. J. (2017). Comparing individual-based approaches to modelling the self-organization of multicellular tissues. *PLOS Computational Biology*, *13*(2), 1–34. doi:10.1371/journal.pcbi.1005387

Van Kampen, N. G. (1992). *Stochastic processes in physics and chemistry* (Vol. 1). Elsevier.

Van Liedekerke, P., Palm, M. M., Jagiella, N., & Drasdo, D. (2015). Simulating tissue mechanics with agent-based models: Concepts, perspectives and some novel results. *Computational particle mechanics*, *2*(4), 401–444. doi:10.1007/s40571-015-0082-3

Voliotis, M., Thomas, P., Grima, R., & Bowsher, C. G. (2016). Stochastic simulation of biomolecular networks in dynamic environments. *PLoS Computational Biology*, *12*(6). doi:10.1371/journal.pcbi.1004923