chCTRW Manual

Tomás Aquino

December 11, 2020

Abstract

This is a brief manual on the structure of the code and how to compile and use it on a UNIX system.

1 Code structure

There are three main .cpp files: batch_delay.cpp, streamtube_gillespie.cpp, and streamtube_concentration.cpp, each in a dedicated folder inside the src folder. All the code is header-only, and it uses only the C++ standard template library (STL) and the header files provided. This is a brief introduction to the structure of the code. Examples on how to use each type of class may be found in each .cpp, and more detailed information about what features new classes must implement are found in the corresponding header files.

The central (template) class is

```
template
<typename WaitingTime, typename DelayTime,
typename... Reactions>
class Gillespie;
```

which can can be found in include/Stochastic/Gillespie/Gillespie.h. This class handles the (generalized) Gillespie algorithm for a set of a waiting time, reactions, and a delay time. All the later are implemented as classes, which are passed as template parameters.

The main .cpp files given for compilation are concerned with reaction with classical exponential waiting time. The relevant class is provided in

include/Stochastic/Gillespie/WaitingTime.h. The reactions themselves are mass-action reactions, implemented generically in include/Stochastic/Reaction.h. These are based on the Stoichiometry class, found in include/Stochastic/Stoichiometry.h, which consists of a reaction rate, a set of reactants and products and the corresponding stoichiometries, and some helper methods. Note that the Stoichiometry class can be used to set up reactions characterized by local rate laws differing from mass-action kinetics, irrespective of the delay structure. Different examples of delay times which can be used with the general setup are provided in include/Stochastic/Gillespie/DelayTime.h. The file include/Stochastic/Gillespie/Gillespie_Stoichiometric.h provides high-level functions to instantiate Gillespie solvers for this type of setup.

Streamtube dynamics rely on the main class

```
template
<typename Patch, typename Advection, typename Reactor,
typename Mass>
class StreamTubeDynamics;
```

found in include/Stochastic/Gillespie/Streamtube.h. The template parameters correspond to classes implementing the type of patches, advection, reaction dynamics, and the type corresponding to mass (either particle numbers or concentrations). An alternating reactive/conservative patch generator with distributed lengths may be found in include/Stochastic/Gillespie/Patch.h. Helper functions, as well as class choices for specific dynamics, are found in include/Stochastic/Gillespie/Models_Streamtube.h. Different model choices are encapsulated inside namespaces with the naming convention model_<advection_distribution>_<reactive_length_distribution>_<conservative_length_distribution>. For example, the namespace streamtube::model_uniform_exp_power refers to a uniform velocity across streamtubes and alternating exponentially-distributed reactive lengths and stable-distributed conservative lengths.

2 Compilation

Each executable can be compiled using the simple Makefile in the same folder. Dedicated make.sh shell scripts for automatic compilation are also provided. If the shell scripts need to be given executing permission, this can be achieved

by running

```
chmod +x make.sh
```

In the case of batch_delay, it suffices to run make.sh to compile. In the case of streamtube_gillespie and streamtube_concentration, a model type must be chosen at compile time. As explained in the previous section, different models are chosen with a using namespace declaration. Each model is compiled into a dedicated executable. For example, running

```
./make.sh uniform_exp_exp
```

in the streamtube_gillespie folder will produce the executable streamtube_gillespie_uniform_exp_exp, corresponding to the declaration using namespace model_uniform_exp_exp, which implements a uniform advection distribution across streamtubes, exponential reactive patch lengths, and exponential conservative patch lengths. Compiled executables are placed in the bin folder.

3 Execution and parameters

The code is run from a command line by running the executable followed by parameter values, separated by spaces. Alternatively, as usual, parameter values may be placed in a file, and run as

```
./exec $(< params.dat)
```

where exec is the desired executable and params.dat holds the parameters (the file extension is irrelevant).

The batch_delay example takes no external parameters. It is a ready-to-run example of a first-order decay reaction with compound Poisson-stable delay. If the wrong number of parameters is passed to the other executables, an error is produced. If they are run with no parameters, a list of the correct parameters followed by short descriptions is printed. By default, output files resulting from running the executables are placed in the output folder.