Stoched: APC 524 Final Project

Generated by Doxygen 1.8.13

Contents

Chapter 1

stoched

Welcome to **stoched**, an application developed at Princeton University for modeling stochastic systems modeled by rate equations and simulating reactions from them!

Getting started information for **stoched** (including build instructions) is located in its interior doc/README.md file.

Introduction

The platform is a fast, compiled code tool with an extremely simple interface aimed towards scientists with minimal programming experience. While other tools for stochastic modeling and simulation exist, none have non-programmer-friendly interfaces and few are specialized to those systems modeled by rate equations alone. We take user-friendly modeling languages developed for Bayesian inference (BUGS/JAGS and Stan) as guides.

Stoched implements the Gillespie algorithm to perform exact simulations. Also, more scalable approximate algorithms derived from the Gillespie algorithm are useful for large systems. These algorithms have historically been used to solve problems in molecular dynamics; today, they are applied to a wide variety of stochastic modeling problems.

Platforms

- Linux
- Mac OS X
- Windows

Requirements

Serial Implementation

- None

Parallel Implementation (optional)

- `Open MPI`

Requirements for Contributors

- `Flex/Bison`
- `Google Test` (located in interior lib/ folder; must be built)

License

Chapter 2

Authors

Caleb Peckham

Caleb is a senior undergraduate student at Princeton University, studying Mechanical and Aerospace Engineering. He is currently working on his senior thesis, an application of high-speed object tracking and machine-learning, under Professor Daniel Nosenchuck. He prepared this documentation, wrote the interface between the parser and the realization, and implemented a Google Test suite.

Dylan Morris

Dylan is a second-year graduate student in the Department of Ecology and Evolutionary Biology at Princeton University. His research interests include the evolution of infectious diseases, statistical inference, and mathematical modeling.

Kevin Griffin

Kevin is a senior undergraduate student at Princeton University, studying Mechanical and Aerospaace Engineering. He is currently working on his senior thesis under Professor Michael Mueller in the area of computational fluid dynamics. He wrote the parser which uses Flex, Bison, and FunctionParser libraries.

Julienne LaChance

Julie is a first-year graduate student in the MAE Department. She obtained a Master's in Applied Mathematics and bachelors degrees in both Applied Math and Mechanical Engineering from RPI. Julie is advised by Prof. Rowley.

Chapter 3

User Guide

Installation

First go to the download page to get the latest distribution, if you have not downloaded **stoched** already.

Download

Download ZIP file at **stoched** Github

```
$ unzip stoched-master.zip
$ cd stoched-master
```

Clone Repository

```
$ git clone https://github.com/APC524/stoched
$ cd stoched
```

Getting Started

Step 1: Creating an input file

The parser uses a custom language that is designed to be accessible to non-programmers. It uses minimal syntax and allows for line comments and end of line comments, and whitespace like new lines between commands. It has been designed to reduce the likelihood of redundant and potentially incorrect information.

A simple input file:

```
SETUP_VARS "a, b"
EVENT RATE "3" "a+1" "b+0"
EVENT RATE "a/3" "a-1" "b+0"

end
```

The first line initializes the variables in the simulation with a comma separated variable list enclosed in quotes. Variable names contain the characters A-Z, a-z, and 0-9. The first character, however must be A-Z or a-z. Note that underscores and spaces are not allowed and that the variable list must not contain spaces.

The next line is an event line. An input file can contain as many event lines as needed. In an abstract sense an event consists of the likelihood of the event occuring and a definition of how it changes the system when it occurs. Practically, an event is a rate function followed by any number of equations that involve the variables in the variable list. The rate function and following event functions can contain nonlinear expressions, support the mathematical symbols + - ∗ / ( ), and can contain white space between symbols.

The last line is end. This indicates the end of the file

A more complicated input file with multiple, nonlinear events:

```
SETUP_VARS "a,b,c"
EVENT RATE "a" "a * (1 + c)" "b - 0.4" "0.56"
EVENT RATE "a*b/d" "a" "b-c" "a*b*c"

end
```

The number of variables and the number of events does not have to be the same. But the number of event functions per event must always equal the number of variables in the variable string. In this example there are three variables: a,b,c; therefore, there are always 4 equations in the EVENT. The first one is the rate function and the last three indicate how the three variables are modified. So the first event occurs at a rate equal to a, and when it occurs it sets a = a ∗ (1 + c), it sets b = b - 0.4, and it sets c = 0.56.

Finer points: Note the syntax "a+1" and "a + 1" are both acceptable. Scientific notation is not yet supported

Lines can be commented by placing a # character. The parser ignores all text on a line after a # character. This means that it can be used to comment out a whole line if it is placed at the beginning of a line, or used to add a note at the end of a line

Comment Example:

```
# This code is now well commented
SETUP_VARS "a,b"
# EVENT RATE "2" "a - 5" "b - 5"
EVENT RATE "3" "a + 1" "b + 1"     #  I've added a comment here to explain why the rate is 3
end
```

In the above example the first event will be ignored because the line begins with #. When the second line is parsed, only the text "I've added . . . " is removed. This input file is functionally equivalent to the first example input file, but it is more readable by human because it had comments.

Step 2: Running stoched

Compiling Serial Code

```
stoched-master$ cd src
stoched-master/src$ make
```

Sample Execution of Serial Code

```
stoched-master$ cd examples
stoched-master/examples$ ../src/stoched.exe chem.parser.in init_file init_file.txt
```

Compiling Parallel Code

Assumes installation of OpenMPI

```
stoched-master$ cd src
stoched-master/src$ make parallel
```

Sample Execution of Parallel Code

For usage on Adroit.

Run_mpi.slurm, located in stoched/src:

```
#!/bin/bash
# Parallel job using 4 processors:
#SBATCH -N 1
#SBATCH --ntasks-per-node=4
#SBATCH -t 0:03:00
#SBATCH --mail-type=begin
#SBATCH --mail-type=end
#SBATCH --mail-type=fail
#SBATCH --mail-user=kevinpg@princeton.edu
# Load openmpi environment
module load openmpi
# Make sure you are in the correct directory
cd ~/stoched/src/
# for nx in 128 256 512
    # do
    #    time ./heat_omp $nx 4 > heat_omp.$nx.4.out
    #    gnuplot -e "outfile='heat_omp.$nx.4.out'" surf.plt
    #    time srun ./heat_mpi $nx > heat_mpi.$nx.4.out
    #    gnuplot -e "outfile='heat_mpi.$nx.4.out'" surf.plt
# done

time srun -n 4 ./stoched_parallel.exe example.parser.in init_file init_file.txt n_realizations 100000 suppress
```

Compiling Test Code

Assumes installation of Google Test suite

```
stoched-master$ cd src
stoched-master/src$ make googletests
```

Sample Execution of Test Code

```
stoched-master$ cd src
stoched-master/src$ ./testmodel.exe
```

Step 3: Parameters

To specify additional parameters, the user may include additional command line arguments, which are listed below. For example, to run the simulation 5 times, the command would look like this:

```
stoched-master/src$ ./stoched.exe example.parser.in n_realizations 5
```

The command line arguments are as follows:

init_file: Required for specifying the initial states data for most model definitions. The exception is a model definition with two species which each start with zero population. This is the default initial state.

method: specifies which algorithm is used to perform computations. Specifying 0 will run the exact Gillespie algorithm, while 1 will run the Euler tau-leap method, and 2 will run the midpoint tau-leap method. Default is 0.

t_initial: allows user to modify the starting time of the simulation. Default is 0 .

t_final: allows user to modify the end time of the simulation. Default is 5000 .

timestep_size: allows user to fix timestep size if desired.

n_realizations: allows user to run the simulation multiple times with the same model and model conditions. Default value is 1.

max_iter: allows user to specify maximum number of iterations. Default value is 100000000.

seed: allows user to fix a seed of the random number generator (which allows for verification of consistency between multiple runs, and with external software results). The default value is 502.

out_path: Allows user to specify an alternative output file path name. The default value is stoched_output, which will write to a file named stoched_output.txt . The extension is not required when specifying pathname.

suppress_print: Option specified as either a 0 or 1. If 1, the software prints only the final value of the simulation to each output file. If 0 (default value), the software runs as usual, printing the results at each timestep. Specifying 1 results in significant speedup of the code.

Step 4: Access Documentation

For further information, visit the Doxygen-generated documentation. The associated HTML documentation can be viewed by pointing a HTML browser to the index.html file in the doc/html directory

To see a PDF version of the documentation, open the refman.pdf file located the doc/latex directory

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 7

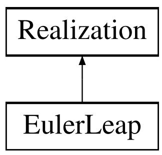
Class Documentation

7.1 EulerLeap Class Reference

Class EulerLeap implements Realization step() function using the basic tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

```
#include <eulerleap.h>
```

Inheritance diagram for EulerLeap:



Public Member Functions

- EulerLeap (Model ∗the_model, const Paramset &the_paramset, rng ∗the_rng, int n_vars, int n_events)

  Default constructor for EulerLeap.
- ∼EulerLeap ()

  Destructor for MidpointLeap.
- int step ()

  Update waiting times.
- int set_to_initial_state ()

  Sets state_array and state_time to their user-specified initial values.

Additional Inherited Members

7.1.1 Detailed Description

Class EulerLeap implements Realization step() function using the basic tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 EulerLeap()

```
EulerLeap::EulerLeap (
            Model * the_model,
            const Paramset & the_paramset,
            rng * the_rng,
            int n_vars,
            int n_events )
```

Default constructor for EulerLeap.

Parameters

the_model	is a Model object
the_paramset	is a Paramset object
the_rng	is a random number generator
n_vars	is an int specifying variable count
n_events	is an int specifying event count

Returns

nothing

7.1.2.2 ∼EulerLeap()

```
EulerLeap::∼EulerLeap ( )
```

Destructor for MidpointLeap.

Returns

nothing

7.1.3 Member Function Documentation

7.1.3.1 set_to_initial_state()

```
int EulerLeap::set_to_initial_state ( ) [virtual]
```

Sets state_array and state_time to their user-specified initial values.

Returns

int

Reimplemented from Realization.

7.1.3.2 step()

```
int EulerLeap::step ( ) [virtual]
```

Update waiting times.

Returns

int

Implements Realization.

The documentation for this class was generated from the following files:

- /Users/Caleb/APC524/stoched/src/eulerleap.h
- /Users/Caleb/APC524/stoched/src/eulerleap.cc

7.2 Event Class Reference

Class Event holds a user-specified event, namely set of functions and associated rate.

```
#include <event.h>
```

Public Member Functions

- Event ()
  Default constructor for Event.
- ∼Event ()
  Destructor of Event.
- void addFunction (string function, string variables)
  Add a function parser to the function array.
- double useFunction (int iFunction, double ∗args)
  Evaluate function stored at specified spot in the function array.
- void setRate (string function, string variables)
  Set equation for rateFunction.
- double getRate (double ∗args)
  Return rate to user based on values of the state array.
- int getSize ()
  Return size of event, namely number of functions, to user.
- double getDeltaVar (int i)
  Return how the ith variable is incremented when the ith equation is called.
- void setDeltaVar (int i, double val)
  set the amount that the ith function increments the ith variable. This is used by midpoint tau leaping

Public Attributes

- string **eventName**

Private Attributes

- FunctionParser ** functionArray_

  Array of function parsers.
- FunctionParser rateFunction

  Rate specified by an equation.
- int eq_count_

  Number of function parsers.
- double * deltaVar_

  how much each variable in the state changes when its corrsponding function is called. Only used by midpoint tau leaping to calculate approximate continuous time derivative.

7.2.1 Detailed Description

Class Event holds a user-specified event, namely set of functions and associated rate.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Event()

```
Event::Event ( )
```

Default constructor for Event.

Parameters

eq_count	is the size of the function array
functionArray	contains all user-specified FunctionParsers that govern event

Returns

nothing

7.2.2.2 ∼Event()

```
Event::∼Event ( )
```

Destructor of Event.

Returns

nothing

7.2.3 Member Function Documentation

7.2.3.1 addFunction()

```
void Event::addFunction (
            string function,
            string variables )
```

Add a function parser to the function array.

Parameters

function	is a string used to generate a FunctionParser object
variables	is a string used to generate a FunctionParser object

Returns

void

7.2.3.2 getDeltaVar()

```
double Event::getDeltaVar (
            int i )
```

Return how the ith variable is incremented when the ith equation is called.

Parameters

i	is an int specifying of which variable to find the delta. 0 is the first variable

Returns

change in value of i when its corresponding equation is called, as a double

7.2.3.3 getRate()

```
double Event::getRate (
            double * stateArray )
```

Return rate to user based on values of the state array.

Parameters

stateArray	is a double array specifying variable values of function

Returns

evaluated rateFunction as a double

7.2.3.4 getSize()

```
int Event::getSize ( )
```

Return size of event, namely number of functions, to user.

Returns

size of event, namely number of functions, as an int

7.2.3.5 setDeltaVar()

```
void Event::setDeltaVar (
            int i,
            double val )
```

set the amount that the ith function increments the ith variable. This is used by midpoint tau leaping

Parameters

i	is an int specifying of which variable to set. 0 is the first variable

Returns

void

7.2.3.6 setRate()

```
void Event::setRate (
            string function,
            string variables )
```

Set equation for rateFunction.

Parameters

function	is a string used for parsing rateFunction
variables	is a string used for parsing rateFunction

Returns

void

7.2.3.7 useFunction()

```
double Event::useFunction (
            int iFunction,
            double * stateArray )
```

Evaluate function stored at specified spot in the function array.

Parameters

iFunction	is an int that indexes function array
stateArray	is a double array specifying variable values of function

Returns

evaluated functionParser as a double

The documentation for this class was generated from the following files:

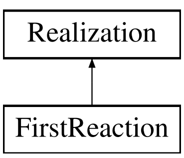
- /Users/Caleb/APC524/stoched/src/event.h
- /Users/Caleb/APC524/stoched/src/event.cc

7.3 FirstReaction Class Reference

Class FirstReaction implements Realization step() function using the exact First Reaction algorithm of Gillespie (1971)

```
#include <firstreaction.h>
```

Inheritance diagram for FirstReaction:



Public Member Functions

- FirstReaction (Model *the_model, const Paramset &the_paramset, rng *the_rng, int n_vars, int n_events)

  Default constructor for FirstReaction.
- ~FirstReaction ()

  Destructor for FirstReaction.
- int step ()

  Update waiting times.

Private Attributes

- double * waiting_times

  pause

Additional Inherited Members

7.3.1 Detailed Description

Class FirstReaction implements Realization step() function using the exact First Reaction algorithm of Gillespie (1971)

7.3.2 Constructor & Destructor Documentation

7.3.2.1 FirstReaction()

```
FirstReaction::FirstReaction (
            Model * the_model,
            const Paramset & the_paramset,
            rng * the_rng,
            int n_vars,
            int n_events )
```

Default constructor for FirstReaction.

Parameters

the_model	is a Model object
the_paramset	is a Paramset object
the_rng	is a random number generator
n_vars	is an int specifying variable count
n_events	is an int specifying event count

Returns

nothing

7.3.2.2 ∼FirstReaction()

```
FirstReaction::∼FirstReaction ( )
```

Destructor for FirstReaction.

Returns

nothing

7.3.3 Member Function Documentation

7.3.3.1 step()

```
int FirstReaction::step ( ) [virtual]
```

Update waiting times.

Returns

int

Implements Realization.

The documentation for this class was generated from the following files:

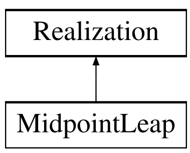
- /Users/Caleb/APC524/stoched/src/firstreaction.h
- /Users/Caleb/APC524/stoched/src/firstreaction.cc

7.4 MidpointLeap Class Reference

Class MidpointLeap implements Realization step() function using the midpoint tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

```
#include <midpointleap.h>
```

Inheritance diagram for MidpointLeap:



Public Member Functions

- MidpointLeap (Model ∗the_model, const Paramset &the_paramset, rng ∗the_rng, int n_vars, int n_events)

  Default constructor for MidpointLeap.
- ∼MidpointLeap ()

  Destructor for MidpointLeap.
- int step ()

  Update waiting times.
- int set_to_initial_state ()

  Sets state_array and state_time to their user-specified initial values.

Private Attributes

- double ∗ midpoint_array_

  midpoint_array_ is a double array holding the current midpoints for the tau-leap step

Additional Inherited Members

7.4.1 Detailed Description

Class MidpointLeap implements Realization step() function using the midpoint tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 MidpointLeap()

```
MidpointLeap::MidpointLeap (
            Model * the_model,
            const Paramset & the_paramset,
            rng * the_rng,
            int n_vars,
            int n_events )
```

Default constructor for MidpointLeap.

Parameters

the_model	is a Model object
the_paramset	is a Paramset object
the_rng	is a random number generator
n_vars	is an int specifying variable count
n_events	is an int specifying event count

Returns

nothing

intialize midpoint array

7.4.2.2 ∼MidpointLeap()

```
MidpointLeap::~MidpointLeap ( )
```

Destructor for MidpointLeap.

Returns

nothing

7.4.3 Member Function Documentation

7.4.3.1 set_to_initial_state()

```
int MidpointLeap::set_to_initial_state ( ) [virtual]
```

Sets state_array and state_time to their user-specified initial values.

Returns

int

Reimplemented from Realization.

7.4.3.2 step()

```
int MidpointLeap::step ( ) [virtual]
```

Update waiting times.

Returns

int

Implements Realization.

The documentation for this class was generated from the following files:

- /Users/Caleb/APC524/stoched/src/midpointleap.h
- /Users/Caleb/APC524/stoched/src/midpointleap.cc

7.5 Model Class Reference

Class Model, which holds user-specified models of stochastic systems from which realizations are to be simulated. A model may have variable parameters; each complete set will be stored in an object of class Paramset.

```
#include <model.h>
```

Public Member Functions

- Model ()
- ∼Model ()
- void addVars (string vars)
- void addEvent (string functionRate)
- void addEventFct (int iEvent, string function)

    Add Event function to specified Event in Model.
- void setTauLeapFalse ()

    Indicate tau leaping is inpermissible by setting the tauLeapAvail_ flag to false.
- bool checkTauLeapAvail ()

    Check if events are compatible with tau leaping.
- double useEventFct (int iEvent, int iFunction, double ∗stateArray)

    Evaluate given function in specified Event.
- string getVarsString ()

    Get a list of the variables that make up the state array.
- double getEventRate (int iEvent, double ∗stateArray)

    Evaluate rate function for a specified Event.
- int getVarsCount ()

    Return total number of variables.
- int getEventsCount ()

    Return total number of Events.
- string getIthVar (int index)

    Returns the ith variable in the variable list.
- double getContDeriv (int whichVar, double ∗stateArray)

    Returns the continuous time derivative of specified variable.
- void setDelta (int whichVar, int whichEvent, double val)

    Sets the delta of a specified variable in the specified event. This delta is needed to compute the continuous time derivative.
- void updateState (int iEvent, double ∗stateArray)

    Update state array by evaluating all functions of a given Event.
- void updateRates (double ∗stateArray, double ∗rateArray)

    Update rate for all Events in Model's Event list.

Private Attributes

- vector< Event ∗ > eventPtrList

    List of Events in Model.
- string vars_

    Variables associated with a model.
- bool **tauLeapAvail_**

7.5.1 Detailed Description

Class Model, which holds user-specified models of stochastic systems from which realizations are to be simulated. A model may have variable parameters; each complete set will be stored in an object of class Paramset.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Model()

```
Model::Model ( )
```

Default constructor for Model

Returns

nothing

7.5.2.2 ∼Model()

```
Model::∼Model ( )
```

Destructor of Model

Returns

nothing

7.5.3 Member Function Documentation

7.5.3.1 addEvent()

```
void Model::addEvent (
            string functionRate )
```

Add Event to Model's list of Events

Parameters

functionRate	is a string that defines an Event's rate

Returns

void

7.5.3.2 addEventFct()

```
void Model::addEventFct (
             int iEvent,
             string function )
```

Add Event function to specified Event in Model.

Parameters

iEvent	is an int that indexes Event list
function	is a string that specifies Event function

Returns

void

7.5.3.3 addVars()

```
void Model::addVars (
             string vars )
```

Add variable list to a Model

Parameters

vars	is a string used to set variables associate with a Model

Returns

void

7.5.3.4 checkTauLeapAvail()

```
bool Model::checkTauLeapAvail ( )
```

Check if events are compatible with tau leaping.

Returns

the status of if tau leaping is available as a boolean

7.5.3.5 getContDeriv()

```
double Model::getContDeriv (
            int whichVar,
            double * stateArray )
```

Returns the continuous time derivative of specified variable.

Parameters

int	specifying which variable from the state array to find derivative
double	array of the entire state array

Returns

ith continuous derivative as a double

7.5.3.6 getEventRate()

```
double Model::getEventRate (
            int iEvent,
            double * stateArray )
```

Evaluate rate function for a specified Event.

Parameters

iEvent	is an int that indexes Event list
stateArray	is a double array specifiying variable values of a function

Returns

evaluated rate function as a double

7.5.3.7 getEventsCount()

```
int Model::getEventsCount ( )
```

Return total number of Events.

Returns

int

7.5.3.8 getIthVar()

```
string Model::getIthVar (
            int index )
```

Returns the ith variable in the variable list.

Parameters

index	is an int that indexes variable list

Returns

ith variable as string

7.5.3.9 getVarsCount()

```
int Model::getVarsCount ( )
```

Return total number of variables.

Returns

int

7.5.3.10 getVarsString()

```
string Model::getVarsString ( )
```

Get a list of the variables that make up the state array.

Returns

comma separated list of variable names as a string

7.5.3.11 setDelta()

```
void Model::setDelta (
            int whichVar,
            int whichEvent,
            double val )
```

Sets the delta of a specified variable in the specified event. This delta is needed to compute the continuous time derivative.

Parameters

int	specifying to which variable from the state array the delta corresponds
int	specifying to which event the delta corresponds
double	specifying the value of delta

Returns

void

7.5.3.12 setTauLeapFalse()

```
void Model::setTauLeapFalse ( )
```

Indicate tau leaping is inpermissible by setting the tauLeapAvail_ flag to false.

Returns

void

7.5.3.13 updateRates()

```
void Model::updateRates (
            double * stateArray,
            double * rateArray )
```

Update rate for all Events in Model's Event list.

Parameters

stateArray	is a double array specifiying variable values of a function
rateArray	is a double array specifiying variable values of a rate function

Returns

void

7.5.3.14 updateState()

```
void Model::updateState (
            int iEvent,
            double * stateArray )
```

Update state array by evaluating all functions of a given Event.

Parameters

iEvent	is an int that indexes Event list
stateArray	is a double array specifiying variable values of a function

Returns

void

7.5.3.15 useEventFct()

```
double Model::useEventFct (
            int iEvent,
            int iFunction,
            double * stateArray )
```

Evaluate given function in specified Event.

Parameters

iEvent	is an int that indexes Event list
iFunction	is an int that indexes an Event's Function list
stateArray	is a double array specifiying variable values of a function

Returns

evaluated function as a double

The documentation for this class was generated from the following files:

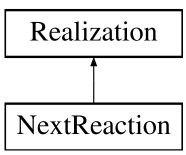
- /Users/Caleb/APC524/stoched/src/model.h
- /Users/Caleb/APC524/stoched/src/model.cc

7.6 NextReaction Class Reference

Class NextReaction implements Realization step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

```
#include <nextreaction.h>
```

Inheritance diagram for NextReaction:



Public Member Functions

- NextReaction (Model *the_model, const Paramset &the_paramset, rng *the_rng, int n_vars, int n_events)

  Default constructor for NextReaction.
- ~NextReaction ()

  Destructor for NextReaction.
- int step ()

  Update waiting times.
- int set_to_initial_state ()

  Sets state_array and state_time to their user-specified initial values.

Private Attributes

- double * waiting_times

  pause

Additional Inherited Members

7.6.1 Detailed Description

Class NextReaction implements Realization step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

7.6.2 Constructor & Destructor Documentation

7.6.2.1 NextReaction()

```
NextReaction::NextReaction (
            Model * the_model,
            const Paramset & the_paramset,
            rng * the_rng,
            int n_vars,
            int n_events )
```

Default constructor for NextReaction.

Parameters

the_model	is a Model object
the_paramset	is a Paramset object
the_rng	is a random number generator
n_vars	is an int specifying variable count
n_events	is an int specifying event count

Returns

nothing

7.6.2.2 ∼NextReaction()

```
NextReaction::∼NextReaction ( )
```

Destructor for NextReaction.

Returns

nothing

7.6.3 Member Function Documentation

7.6.3.1 set_to_initial_state()

```
int NextReaction::set_to_initial_state ( ) [virtual]
```

Sets state_array and state_time to their user-specified initial values.

Returns

int

Reimplemented from Realization.

7.6.3.2 step()

```
int NextReaction::step ( ) [virtual]
```

Update waiting times.

Returns

int

Implements Realization.

The documentation for this class was generated from the following files:

- /Users/Caleb/APC524/stoched/src/nextreaction.h
- /Users/Caleb/APC524/stoched/src/nextreaction.cc

7.7 Paramset Class Reference

Class Paramset holds a particular set of pameters for user requested simulation run(s)

```
#include <paramset.h>
```

Public Member Functions

- Paramset (int method, int n_vars, double ∗initial_values, double t_initial, double t_final, double timestep_size, int n_realizations, int max_iter, int seed, int suppress_output)

  Default constructor for Paramset.
- ∼Paramset ()

  Destructor for Paramset.

Public Attributes

- const int method

  which algorithm to use for simulation
- const int n_vars

  number of initial values/variables
- const double ∗ initial_values

  initial values for variables
- const double t_initial

  initial time for simulation
- const double t_final

  final time for simulation
- const double timestep_size

  size of timestep for approximate
- int n_realizations

  number of realizations to simulate
- int max_iter

  max number of iterations to simulate
- int seed

  seed for the random number generator
- int suppress_output

  allows user to only print final state value

7.7.1 Detailed Description

Class Paramset holds a particular set of pameters for user requested simulation run(s)

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Paramset()

```
Paramset::Paramset (
            int method,
            int n_vars,
            double * initial_values,
            double t_initial,
            double t_final,
            double timestep_size,
            int n_realizations,
            int max_iter,
            int seed,
            int suppress_output )
```

Default constructor for Paramset.

Parameters

method	is an int that specifies algorithm to use for simulation
n_vars	is an int that specifies number of variables
initial_values	is a double array that sets initial values for variables
t_initial	is a double that sets initial time for simulation
t_final	is a double that sets initial time for simulation
timestep_size	is a double representing the size of the timestep for approximate methods
n_realizations	is an int representing number of realizations to simulate
max_iter	is an int and is the maximum number of iterations to simulate

7.7.2.2 ∼Paramset()

```
Paramset::~Paramset ( )
```

Destructor for Paramset.

Returns

nothing

The documentation for this class was generated from the following files:

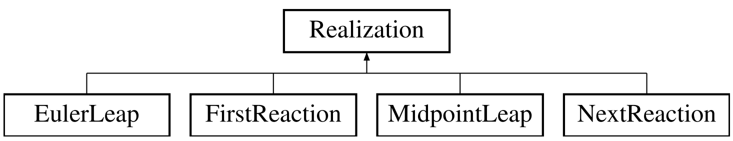
- /Users/Caleb/APC524/stoched/src/paramset.h
- /Users/Caleb/APC524/stoched/src/paramset.cc

7.8 Realization Class Reference

Class Realization holds realizations of a Model (state array, propensities, waiting times, etc.)

```
#include <realization.h>
```

Inheritance diagram for Realization:



Public Member Functions

- Realization (Model *the_model, const Paramset &the_paramset, rng *the_rng, int n_vars, int n_events)
  Default constructor for Realization.
- virtual ~Realization ()
  Destructor of Realization.
- int simulate (std::ofstream &myfile)
  Simulates the realization from t_inital to t_final.
- virtual int step ()=0
  takes one simulation step according to the chosen algorithm
- bool rates_are_zero ()
  checks whether all rates are zero
- int output_state (std::ofstream &myfile)
  Prints the current state of the simulation.
- virtual int set_to_initial_state ()
  Sets state_array and state_time to their user-specified initial values.

Public Attributes

- Model * the_model
  the_model is a Model instance
- const Paramset the_paramset
  the_paramset is a Paramset instance
- rng * the_rng
  the_rng is an random number generator
- const int n_vars
  n_vars is an int specifying number of variables
- const int n_events
  n_events is an int specifying number of events
- double * state_array
  state_array is a double array specifiying variable values of a function
- double * rates
  rates is a double array specifying variable values of a rate function
- double state_time
  state_time is a double that tracks state progress

7.8.1 Detailed Description

Class Realization holds realizations of a Model (state array, propensities, waiting times, etc.)

7.8.2 Constructor & Destructor Documentation

7.8.2.1 Realization()

```
Realization::Realization (
            Model * the_model,
            const Paramset & the_paramset,
            rng * the_rng,
            int n_vars,
            int n_events )
```

Default constructor for Realization.

Parameters

the_model	is a Model object
the_paramset	is a Paramset object
the_rng	is a random number generator
n_vars	is an int specifying variable count
n_events	is an int specifying event count

Returns

nothing

7.8.2.2 ∼Realization()

```
Realization::∼Realization ( ) [virtual]
```

Destructor of Realization.

Returns

nothing

7.8.3 Member Function Documentation

7.8.3.1 output_state()

```
int Realization::output_state (
            std::ofstream & myfile )
```

Prints the current state of the simulation.

Returns

int

7.8.3.2 rates_are_zero()

```
bool Realization::rates_are_zero ( )
```

checks whether all rates are zero

Returns

bool

7.8.3.3 set_to_initial_state()

```
int Realization::set_to_initial_state ( )  [virtual]
```

Sets state_array and state_time to their user-specified initial values.

Returns

int

Reimplemented in EulerLeap, MidpointLeap, and NextReaction.

7.8.3.4 simulate()

```
int Realization::simulate (
            std::ofstream & myfile )
```

Simulates the realization from t_inital to t_final.

Returns

int

The documentation for this class was generated from the following files:

- /Users/Caleb/APC524/stoched/src/realization.h
- /Users/Caleb/APC524/stoched/src/realization.cc

7.9 RealizationFactory Class Reference

Class RealizationFactory generates required instance of Realization (FirstReaction, NextReaction, EulerLeap, MidpointLeap) based on input.

```
#include <realization_factory.h>
```

Static Public Member Functions

- static Realization ∗ NewRealization (Model ∗the_model, const Paramset &the_paramset, rng ∗the_rng, int n_vars, int n_events)

  Create new realization.

7.9.1 Detailed Description

Class RealizationFactory generates required instance of Realization (FirstReaction, NextReaction, EulerLeap, MidpointLeap) based on input.

7.9.2 Member Function Documentation

7.9.2.1 NewRealization()

```
Realization * RealizationFactory::NewRealization (
            Model * the_model,
            const Paramset & the_paramset,
            rng * the_rng,
            int n_vars,
            int n_events ) [static]
```

Create new realization.

Parameters

the_model	is a Model object
the_paramset	is a Paramset object
the_rng	is a random number generator
n_vars	is an int specifying variable count
n_events	is an int specifying event count

Returns

nothing

The documentation for this class was generated from the following files:

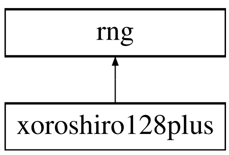
- /Users/Caleb/APC524/stoched/src/realization_factory.h
- /Users/Caleb/APC524/stoched/src/realization_factory.cc

7.10 rng Class Reference

Class rng implements random number generator, based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna (vigna@acm.org)

`#include <rng.h>`

Inheritance diagram for rng:



Public Member Functions

- virtual ~rng ()

  Destructor of rng object.
- virtual uint64_t next ()=0

  get a new random int64
- virtual double runif ()=0

  get a new random uniform(0, 1) RV
- double rexp (double lambda)

  get a new random exponential(lambda) RV
- long rpois (double mean)

  get a new random poisson(mean) RV
- virtual void jump ()=0

  quick 2^64 calls to next (for parallelism)
- double log_factorial (int k)

Private Member Functions

- long poisson_knuth (double mean)

  random poisson
- long poisson_ptrs (double mean)

  random poisson

7.10.1 Detailed Description

Class rng implements random number generator, based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna (vigna@acm.org)

7.10.2 Member Function Documentation

7.10.2.1 log_factorial()

```
double rng::log_factorial (
            int k )
```

log factorial function modified from public domain C# implementation by John D. Cook (http://www.johndcook.com/blog/csharp_log_factorial/) and PTRS algorithm by Wolfgang Hoermann (1993)

The documentation for this class was generated from the following files:

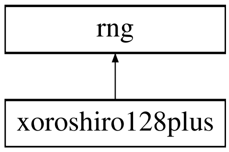
- /Users/Caleb/APC524/stoched/src/rng.h
- /Users/Caleb/APC524/stoched/src/rng.cc

7.11 xoroshiro128plus Class Reference

Class xoroshiro128plus implements a random number generator of Class rng.

```
#include <xoroshiro128plus.h>
```

Inheritance diagram for xoroshiro128plus:



Public Member Functions

- xoroshiro128plus (int seed)

    Default constructor.
- ∼xoroshiro128plus ()

    Default Destructor for Xoroshiro128plus.
- uint64_t next ()

    get random int and update state
- double runif ()

    get random uniform(0, 1) double and update state
- void jump ()

Private Member Functions

- uint64_t rotl (const uint64_t x, int k)

    simulated rotate
- uint64_t splitmix64 ()

    splitmix64 next function, for initializing generator

Private Attributes

- uint64_t **s** [2]
- uint64_t **splitmixstate**

7.11.1 Detailed Description

Class xoroshiro128plus implements a random number generator of Class rng.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 xoroshiro128plus()

```
xoroshiro128plus::xoroshiro128plus (
            int seed )
```

Default constructor.

initialize state with splitmix64 random ints from seed int (prevents similar seeds from generating correlated states)

7.11.3 Member Function Documentation

7.11.3.1 jump()

```
void xoroshiro128plus::jump ( )  [virtual]
```

This is the jump function for the generator. It is equivalent to 2^64 calls to next(); it can be used to generate 2^64 non-overlapping subsequences for parallel computations.

Implements rng.

The documentation for this class was generated from the following files:

- /Users/Caleb/APC524/stoched/src/xoroshiro128plus.h
- /Users/Caleb/APC524/stoched/src/xoroshiro128plus.cc

Chapter 8

File Documentation

8.1 /Users/Caleb/APC524/stoched/src/eulerleap.cc File Reference

Class EulerLeap implements Realization step() function using the basic tau leap approximate method of Gillepsie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

```
#include "eulerleap.h"
```

8.1.1 Detailed Description

Class EulerLeap implements Realization step() function using the basic tau leap approximate method of Gillepsie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.2 /Users/Caleb/APC524/stoched/src/eulerleap.h File Reference

Class EulerLeap implements Realization step() function using the basic tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

```
#include <stdexcept>
#include "realization.h"
```

Classes

- class EulerLeap

  Class EulerLeap implements Realization step() function using the basic tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

8.2.1 Detailed Description

Class EulerLeap implements Realization step() function using the basic tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic forward Euler method for the numerical solution of ordinary differential equations.

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.3 /Users/Caleb/APC524/stoched/src/event.cc File Reference

Class Event holds a user-specified event, namely set of functions and associated rate.

```
#include "event.h"
#include "fparser/fparser.hh"
#include <assert.h>
#include <string>
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
```

8.3.1 Detailed Description

Class Event holds a user-specified event, namely set of functions and associated rate.

Author

Caleb Peckham (peckham@princeton.edu)

Date

12/6/16

Version

1.0

8.4 /Users/Caleb/APC524/stoched/src/event.h File Reference

Class Event holds a user-specified event, namely set of functions and associated rate.

```
#include "fparser/fparser.hh"
```

Classes

- class Event

  Class Event holds a user-specified event, namely set of functions and associated rate.

8.4.1 Detailed Description

Class Event holds a user-specified event, namely set of functions and associated rate.

Author

Caleb Peckham (peckham@princeton.edu)

Date

12/6/16

Version

1.0

8.5 /Users/Caleb/APC524/stoched/src/eventtests.cc File Reference

Test Event code.

```
#include <string>
#include <stdio.h>
#include <iostream>
#include "event.h"
#include "gtest/gtest.h"
```

Functions

- **TEST_F** (EventTests, UseFunction)
- **TEST_F** (EventTests, RateFunction)
- int **main** (int argc, char **argv)

8.5.1 Detailed Description

Test Event code.

Author

Caleb Peckham (peckham@princeton.edu)

Date

1/12/17

Version

1.0

8.6 /Users/Caleb/APC524/stoched/src/firstreaction.cc File Reference

Class FirstReaction implements Realization step() function using the exact First Reaction algorithm of Gillespie (1971)

```
#include "firstreaction.h"
```

8.6.1 Detailed Description

Class FirstReaction implements Realization step() function using the exact First Reaction algorithm of Gillespie (1971)

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.7 /Users/Caleb/APC524/stoched/src/firstreaction.h File Reference

Class FirstReaction implements Realization step() function using the exact First Reaction algorithm of Gillespie (1971)

```
#include "realization.h"
```

Classes

- class FirstReaction

    Class FirstReaction implements Realization step() function using the exact First Reaction algorithm of Gillespie (1971)

8.7.1 Detailed Description

Class FirstReaction implements Realization step() function using the exact First Reaction algorithm of Gillespie (1971)

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.8 /Users/Caleb/APC524/stoched/src/midpointleap.cc File Reference

Class MidpointLeap implements Realization step() function using the midpoint tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

```
#include "midpointleap.h"
```

8.8.1 Detailed Description

Class MidpointLeap implements Realization step() function using the midpoint tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

Author

Dylan Morris (dhmorris@princeton.edu)

Date

2016-01-16

Version

1.0

8.9 /Users/Caleb/APC524/stoched/src/midpointleap.h File Reference

Class MidpointLeap implements Realization step() function using the midpoint tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

```
#include <stdexcept>
#include "realization.h"
```

Classes

- class MidpointLeap

  Class MidpointLeap implements Realization step() function using the midpoint tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

8.9.1 Detailed Description

Class MidpointLeap implements Realization step() function using the midpoint tau leap approximate algorithm of Gillepsie (2001). The method is analogous to the deterministic midpoint (2nd-order Runge-Kutta) method for the numerical solution of ordinary differential equations.

Author

Dylan Morris (dhmorris@princeton.edu)

Date

2016-01-16

Version

1.0

8.10 /Users/Caleb/APC524/stoched/src/model.cc File Reference

Class Model, which holds user-specified models of stochastic systems from which realizations are to be simulated.

```
#include "event.h"
#include "model.h"
#include "fparser/fparser.hh"
#include <assert.h>
#include <string>
#include <stdio.h>
#include <iostream>
#include <vector>
#include <sstream>
```

8.10.1 Detailed Description

Class Model, which holds user-specified models of stochastic systems from which realizations are to be simulated.

Author

Caleb Peckham (peckham@princeton.edu)

Date

12/6/16

Version

1.0

8.11 /Users/Caleb/APC524/stoched/src/model.h File Reference

Class Model, which holds user-specified models of stochastic systems from which realizations are to be simulated.

```
#include "event.h"
#include <vector>
#include <sstream>
#include <string>
```

Classes

- class Model

  Class Model, which holds user-specified models of stochastic systems from which realizations are to be simulated. A model may have variable parameters; each complete set will be stored in an object of class Paramset.

8.11.1 Detailed Description

Class Model, which holds user-specified models of stochastic systems from which realizations are to be simulated.

Author

Caleb Peckham (peckham@princeton.edu)

Date

12/6/16

Version

1.0

8.12 /Users/Caleb/APC524/stoched/src/modeltests.cc File Reference

Test usage of Model class.

```
#include <string>
#include <stdio.h>
#include <iostream>
#include "model.h"
#include "gtest/gtest.h"
```

Functions

- **TEST_F** (ModelTests, UseEventFunction)
- **TEST_F** (ModelTests, EventRateFunction)
- **TEST_F** (ModelTests, UpdateRateFunction)
- **TEST_F** (ModelTests, UpdateStateArray)
- int **main** (int argc, char **argv)

8.12.1 Detailed Description

Test usage of Model class.

Author

Caleb Peckham (peckham@princeton.edu)

Date

12/11/16

Version

1.0

8.13 /Users/Caleb/APC524/stoched/src/nextreaction.cc File Reference

Class NextReaction implements Realization step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

```
#include "nextreaction.h"
```

8.13.1 Detailed Description

Class NextReaction implements Realization step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

Author

Dylan Morris (dhmorris@princeton.edu)

8.14 /Users/Caleb/APC524/stoched/src/nextreaction.h File Reference

Class nextreaction implements Realization step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

```
#include "realization.h"
#include <float.h>
```

Classes

- class NextReaction

  Class NextReaction implements Realization step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

8.14.1 Detailed Description

Class nextreaction implements Realization step() function using the exact Next Reaction algorithm of Gibson & Bruck (2000)

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.15 /Users/Caleb/APC524/stoched/src/paramset.cc File Reference

Class Paramset holds a particular set of pameters for user requested simulation run(s)

```
#include "paramset.h"
```

8.15.1 Detailed Description

Class Paramset holds a particular set of pameters for user requested simulation run(s)

Author

Dillon Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.16 /Users/Caleb/APC524/stoched/src/paramset.h File Reference

Class Paramset holds a particular set of pameters for user requested simulation run(s)

Classes

- class Paramset

  Class Paramset holds a particular set of pameters for user requested simulation run(s)

8.16.1 Detailed Description

Class Paramset holds a particular set of pameters for user requested simulation run(s)

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.17 /Users/Caleb/APC524/stoched/src/parsertests.cc File Reference

Example parsing code.

```
#include <string>
#include <stdio.h>
#include <iostream>
#include "model.h"
#include "event.h"
#include "gtest/gtest.h"
```

Functions

- int **parseFile** (Model &model, string inputfilename)
- **TEST_F** (ParserTests, ParserReturn)
- int **main** (int argc, char **argv)

8.17.1 Detailed Description

Example parsing code.

Author

Caleb Peckham (peckham@princeton.edu)

Date

1/12/17

Version

1.0

8.18 /Users/Caleb/APC524/stoched/src/realization.cc File Reference

Class Realization holds realizations of a Model (state array, propensities, waiting times, etc.)

```
#include "realization.h"
```

8.18.1 Detailed Description

Class Realization holds realizations of a Model (state array, propensities, waiting times, etc.)

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.19 /Users/Caleb/APC524/stoched/src/realization.h File Reference

Class Realization holds realizations of a Model (state array, propensities, waiting times, etc.)

```
#include <math.h>
#include <stdio.h>
#include <fstream>
#include <iomanip>
#include <float.h>
#include <stdexcept>
#include <iostream>
#include "model.h"
#include "paramset.h"
#include "rng.h"
```

Classes

- class Realization
  Class Realization holds realizations of a Model (state array, propensities, waiting times, etc.)

8.19.1 Detailed Description

Class Realization holds realizations of a Model (state array, propensities, waiting times, etc.)

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.20 /Users/Caleb/APC524/stoched/src/realization_factory.cc File Reference

Class RealizationFactory generates required instance of Realization (FirstReaction, NextReaction, EulerLeap) based on input.

```
#include "realization_factory.h"
```

8.20.1 Detailed Description

Class RealizationFactory generates required instance of Realization (FirstReaction, NextReaction, EulerLeap) based on input.

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.21 /Users/Caleb/APC524/stoched/src/realization_factory.h File Reference

Class RealizationFactory generates required instance of Realization (FirstReaction, NextReaction, EulerLeap) based on input.

```
#include "realization.h"
#include "nextreaction.h"
#include "firstreaction.h"
#include "eulerleap.h"
#include "midpointleap.h"
```

Classes

- class RealizationFactory

  Class RealizationFactory generates required instance of Realization (FirstReaction, NextReaction, EulerLeap, MidpointLeap) based on input.

8.21.1 Detailed Description

Class RealizationFactory generates required instance of Realization (FirstReaction, NextReaction, EulerLeap) based on input.

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.22 /Users/Caleb/APC524/stoched/src/rng.cc File Reference

Based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna (vigna@acm.org)

```
#include "rng.h"
#include <math.h>
```

8.22.1 Detailed Description

Based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna (vigna@acm.org)

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.22.2 poisson_ptrs(double mean) implements the PTRS

algorithm of Wolfgang Hoermann (1993) and is based upon the function rk_poisson_ptrs from the mtrand module of numpy, available here: github.com/numpy/numpy/blob/master/numpy/random/mtrand/distributions.c

That function appears with the following copyright and license:

8.23 /Users/Caleb/APC524/stoched/src/rng.h File Reference

Based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna (vigna@acm.org)

```
#include <stdint.h>
#include <math.h>
#include <stdexcept>
```

Classes

- class rng

  Class rng implements random number generator, based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna (vigna@acm.org)

Functions

- double to_double (uint64_t x)

  convert uint64_t to double in (0, 1)

8.23.1 Detailed Description

Based upon public domain xorshift implementations by David Blackman and Sebastiano Vigna (vigna@acm.org)

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.24 /Users/Caleb/APC524/stoched/src/simulate.cc File Reference

Example definition for ending simulation loop.

```
#include <stdio.h>
#include "model.h"
#include "paramset.h"
#include "realization.h"
```

Functions

- int main (int argc, char *argv[ ])

  Example definition for ending simulation loop.

8.24.1 Detailed Description

Example definition for ending simulation loop.

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.24.2 Function Documentation

8.24.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Example definition for ending simulation loop.

Returns

int

8.25 /Users/Caleb/APC524/stoched/src/tauleapavail.cc File Reference

Called by parser.y to determine whether to use tau leap or not.

```
#include <cstdio>
#include <iostream>
#include "model.h"
```

Functions

- bool tauLeapAvail (Model &cModel, string varListStr, string functionStr, int eqnCnt, int eventCnt)

  Is tau leap available?

8.25.1 Detailed Description

Called by parser.y to determine whether to use tau leap or not.

Author

Kevin Griffin (kpgriffin@princeton.edu)

8.25.2 Function Documentation

8.25.2.1 tauLeapAvail()

```
bool tauLeapAvail (
            Model & cModel,
            string varListStr,
            string functionStr,
            int eqnCnt,
            int eventCnt )
```

Is tau leap available?

Parameters

cmodel	is a Model object
varList	str is a string of variables
functionStr	is a string used to generate a Function Parser
eqnCnt	is an int specifying number of equations
eventCnt	is an int specifying number of events

Returns

nothing

8.26 /Users/Caleb/APC524/stoched/src/testevent.cc File Reference

Example usage of Event class.

```
#include <iostream>
#include <string>
#include <stdio.h>
#include "event.h"
```

Functions

- int main ()

  Example usage of Event class.

8.26.1 Detailed Description

Example usage of Event class.

Author

Caleb Peckham (peckham@princeton.edu)

Date

12/6/16

Version

1.0

8.26.2 Function Documentation

8.26.2.1 main()

```
int main ( )
```

Example usage of Event class.

Returns

int

8.27 /Users/Caleb/APC524/stoched/src/testmodel.cc File Reference

Example usage of Model class.

```
#include <iostream>
#include <string>
#include <stdio.h>
#include "event.h"
#include "model.h"
```

Functions

- int main ()

  Example usage of Model class.

8.27.1 Detailed Description

Example usage of Model class.

Author

Caleb Peckham (peckham@princeton.edu)

Date

12/6/16

Version

1.0

8.27.2 Function Documentation

8.27.2.1 main()

```
int main ( )
```

Example usage of Model class.

Returns

int

8.28 /Users/Caleb/APC524/stoched/src/testparser.cc File Reference

Example parsing code.

```
#include <stdio.h>
#include <iostream>
#include "model.h"
#include "event.h"
```

Functions

- int parseFile (Model &model, string inputfilename)

  Declare the parser method written by flex and bison.
- int main (int argc, char *argv[ ])

  Example parsing code.

8.28.1 Detailed Description

Example parsing code.

Author

Kevin Griffin (kevinpg@princeton.edu)

Date

12/6/16

Version

1.0

8.28.2 Function Documentation

8.28.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Example parsing code.

Returns

int

8.29 /Users/Caleb/APC524/stoched/src/testsimulate.cc File Reference

Example simulation code.

```
#include <iostream>
#include <string>
#include <stdio.h>
#include <fstream>
#include <iomanip>
#include <chrono>
#include "event.h"
#include "model.h"
#include "paramset.h"
#include "realization.h"
#include "xoroshiro128plus.h"
#include "nextreaction.h"
#include "firstreaction.h"
```

Functions

- int **main** ()

8.29.1 Detailed Description

Example simulation code.

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.30 /Users/Caleb/APC524/stoched/src/xoroshiro128plus.cc File Reference

Class xoroshorio128plus implements a random number generator of Class rng.

```
#include "xoroshiro128plus.h"
#include "rng.h"
#include "math.h"
#include "string.h"
```

Macros

- #define **INT64_C**(c) (int64_t) c
- #define **UINT64_C**(c) (uint64_t) c

Functions

- double to_double (uint64_t x)

  convert uint64_t to double in (0, 1)

8.30.1 Detailed Description

Class xoroshorio128plus implements a random number generator of Class rng.

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

8.31 /Users/Caleb/APC524/stoched/src/xoroshiro128plus.h File Reference

Class xoroshorio128plus implements a random number generator of Class rng.

```
#include <stdint.h>
#include "rng.h"
```

Classes

- class xoroshiro128plus

    Class xoroshiro128plus implements a random number generator of Class rng.

8.31.1 Detailed Description

Class xoroshorio128plus implements a random number generator of Class rng.

Author

Dylan Morris (dhmorris@princeton.edu)

Date

12/6/16

Version

1.0

Index