simulations Documentation

Release 0.8.3

Gregory McWhirter

January 14, 2014

Base framework for running and analyzing gametheory simulations

Modules:

base Handles generic set-up tasks for EventEmitter-derived classes

simulation Handles defining simulations

simulation_runner Handles running simulations

statsparser Handles analyzing simulation output

Packages:

dynamics Base implementations of various dynamics

utils Has utility modules

CHAPTER 1

base

Handles basic functionality shared by a number of other modules

Classes:

Base Handles basic __init__ and listener patterns shared among several classes

Decorators:

listener() A class decorator adding a listener without disrupting `_add_listeners()`

once() A class decorator adding a once-only listener without disrupting `_add_listeners()`

withoptions() A class decorator adding basic `OptionParser` functionality

class `simulations.base.`**Base**(**args*, ***kwdargs*)
Bases: `simulations.utils.eventemitter.EventEmitter`

The base class that handles common functionality from which other `EventEmitter` classes are derived

Keyword Parameters:

default_handlers If true or not present, adds the default handlers defined in `Base._add_default_listeners()`

_add_default_listeners()
Sets up default listeners for various events (should implement)

_add_listeners()
Set up listeners for various events (should implement)

add_listener(*event*, *listener=None*)
Adds a listener to an event

Parameters:

event the event to listen for

listener the handler for the event (should be a function / callable)

emit(*event*, **args*, ***kwargs*)
Emit an event, triggering the handlers on it with certain arguments

Parameters:

event the event to trigger

args arguments to pass to the triggered listeners

kwargs keyword arguments to pass to the triggered listeners

listeners(*event*)
Gets a COPY of the list of listeners on an event

Parameters:

event the event for which to lookup the listeners

max_errstr = '"\n WARNING: Possible EventEmitter memory leak:\n {0} listeners added for {1}. Use set_max_listener

on(**args*)
Alias for add_listener

once(*event*, *listener=None*)

Add a listener, but only execute it the first time the event occurs, then remove it

Parameters:

event the event to listen for

listener the listener function / callable

remove_all_listeners(*event*)
Clears all listeners from an event

Parameters:

event the event to clear listeners from

remove_listener(*event*, *listener=None*)
Remove a listener from an event

Parameters:

event the event from which to remove it

listener the handler to remove

set_max_listeners(*max_listeners*)
Set the maximum number of listeners for each event.

Parameters:

max_listeners the maximum number of listeners allowed (None for no limit)

simulations.base.**listener**(*event*, *handler*)
Class decorator to add listeners in a brief way

This is effectively the same as adding a call to `add_listener()` in `_add_listeners()`.

Parameters:

event the name of the event to listen for

handler the event handler

simulations.base.**once**(*event*, *handler*)
Class decorator to handler once-listeners in a brief way.

This is effectively the same as adding a call to `once()` in `_add_listeners()`.

Parameters:

event the name of the event to listen for

handler the event handler

simulations.base.**withoptions**()

A class wrapper that handles using an `OptionParser`

Adds Keyword Parameters:

option_error_handler An error handler for the `OptionParser`

option_exit_handler An exit handler for the `OptionParser`

Adds Events:

oparser set up emitted after the `OptionParser` is set up and able to add options

CHAPTER 2

simulation

Handle the basics of running parallel simulations

Classes:

Simulation Framework for a basic simulation

class simulations.simulation.**Simulation**(*data*, *iteration*, *outfile*, **args*, ***kwdargs*)

Bases: simulations.base.Base

Base class for an individual simulation

Parameters:

data The data dictionary for the simulation. Usually created by the SimulationRunner

iteration The iteration number of the simulation. Usually handled by the SimulationRunner

outfile The name of a file to which to dump output (or None, indicating stdout)

Public Methods:

run() Runs the simulation

set_output_file() Sets the output file name

Methods to Implement:

_add_listeners() Set up listeners for various simulation events

_run() Actual simulation functionality

Events:

done(this) emitted when run() is complete and results have been stored

outfile changed(this) emitted when the outfile name has been changed by set_output_file()

outfile error(this) emitted when there was an error opening the output file

run(this) emitted just before _run() is called

_add_listeners()

Sets up listeners for various events

_run(**args*, ***kwdargs*)

Actual functionality for running the simulation (should implement)

run()

Runs the simulation. Handles opening and closing the out file object.

set_output_file(*fname*)
Sets the name of the outfile

Parameters:

fname The file name for the outfile (or None or False)

simulations.simulation.**_close_out_fd**(*this*)
Closes the Simulation.out object that simulations should print to

Parameters:

this A reference to a Simulation instance

simulations.simulation.**_open_out_fd**(*this*)
Opens the Simulation.out object that simulations should print to

Parameters:

this A reference to a Simulation instance

CHAPTER 3

simulation_runner

Handler for running simulations in a multiprocessing environment

Classes:

SimulationRunner Handles option parsing and a pp server pool for simulations

Functions:

default_pool_handler() Default handler for 'pool started' events

default_start_handler() Default handler for 'start' events

default_result_handler() Default handler for 'result' events

run_simulation() runs a simulation task

class simulations.simulation_runner.**SimulationRunner**(**args*, ***kwdargs*)

Bases: simulations.base.Base

Handles option parsing and a multiprocessing pool for simulations

Parameters:

simulation_class The class representing the Simulation to run

Keyword Parameters:

default_handlers Flag to set default event handlers for some events (default True)

option_error_handler An error handler for the OptionParser

option_exit_handler An exit handler for the OptionParser

Public Methods:

go() Kick off the batch of simulations

Methods to Implement:

_add_listeners() Set up event listeners for run events

Events:

done(this) emitted when results are totally done

go(this) emitted when the go() method is called

made output_dir(this) emitted if/when the output directory needs to be created

oparser set up(this) emitted after the OptionParser is set up and able to add options

options parsed(this) emitted after the OptionParser has parsed arguments

pool started(this, pool) emitted after the `multiprocessing.Pool` is set up

result(this, result) emitted when a result is complete

start(this) emitted just before the pool `imap_unordered()` is called

_add_default_listeners()
Sets up default listeners for various events

Events Handled:

•pool started - `default_pool_handler()`

•start - `default_start_handler()`

•result - `default_result_handler()`

_check_base_options()
Verify the values passed to the base options

Checks:

•Number of duplications is positive

•Pool size is positive, if specified

_set_base_options()
Set up the basic `OptionParser` options. Calling this is handled by the decorator `simulations.base.withoptions()`

Options:

-D, --nofiledump Do not dump individual simulation output

-F STRING, --filename=STRING Format string for file name of individual duplication output

-N NUM, --duplications=NUM Number of trials to run

-O DIR, --output=DIR Directory to which to output the results

-P NUM, --poolsize=NUM Number of simultaneous trials

-Q, --quiet Suppress all output except aggregate pickle dump

-S FILE, --statsfile=FILE File name for aggregate, pickled output

go(***kwdargs*)
Verify options and run the batch of simulations

Keyword Parameters:

option_args arguments to pass to the `OptionParser`. Defaults to sys.argv[1:].

option_values target to put the values from the `OptionParser` in (probably should not use)

`simulations.simulation_runner.`**default_pool_handler**(*this*, *pool*, *out=None*)
Default handler for the 'pool started' event

Parameters:

this a reference to a `SimulationRunner` instance

pool the `multiprocessing.Pool` that was started

out the file descriptor to print to

simulations.simulation_runner.**default_result_handler**(*this*, *result*, *out=None*)
Default handler for the 'result' event

Parameters:

this a reference to a SimulationRunner instance

result the result object from the Simulation

out the file descriptor to print to

simulations.simulation_runner.**default_start_handler**(*this*, *out=None*)
Default handler for the 'start' event

Parameters:

this a reference to a SimulationRunner instance

out the file descriptor to print to

simulations.simulation_runner.**run_simulation**(*task*)
A simple function to run a Simulation. Used with the multiprocessing pool.

Parameters:

task An instance of a Simulation to run

CHAPTER 4

statsparser

Handle the parsing and aggregation of simulation results

Classes:

StatsParser main statistics parsing class

class simulations.statsparser.**StatsParser**(**args*, ***kwdargs*)

Bases: simulations.base.Base

Base class for parsing result files.

Keyword Parameters:

option_error_handler An error handler for the OptionParser

option_exit_handler An exit handler for the OptionParser

Public Methods:

go() Kick off parsing the results

Methods to Implement:

_add_listeners() Add listeners for various parsing events

Events:

done(this, out) emitted when results are totally done

go(this) emitted when the go() method is called

oparser set up(this) emitted after the OptionParser is set up and able to add options

options parsed(this) emitted after the OptionParser has parsed arguments

result(this, out, duplication, result) emitted when a result is ready to be interpreted

result options(this, out, options) emitted when the simulation runner options are ready to be interpreted

_check_base_options()

Verify the values passed to the base options

Checks:

•Stats file exists

_go(*statsfile*, *out*)

Actually parses the data

Parameters:

statsfile a file object for the file to parse

out the output target (either a file object or sys.stdout)

_set_base_options()
Set up the basic `OptionParser` options

Options:

-F FILE, --statsfile=FILE File name of the results file

-O FILE, --outfile=FILE File to which to print data

-V, --verbose Print detailed output to stdout as things are processed

go(***kwdargs*)
Pass off the parsing of the results file after some data manipulation

Keyword Parameters:

option_args arguments to pass to the `OptionParser`. Defaults to sys.argv[1:].

option_values target of option parsing (probably should not use)

CHAPTER 5

dynamics

Framework implementations for running simulations with various dynamics

Modules:

discrete_replicator Implements a generic discrete replicator dynamics. You should probably not use this directly.

generation_machine Implements a generation-stepping machine. You should probably not use this directly.

npop_discrete_replicator Implements n-population discrete time replicator dynamics

onepop_discrete_replicator Implements 1-population discrete time replicator dynamics

5.1 discrete_replicator

5.2 generation_machine

5.3 npop_discrete_replicator

5.4 onepop_discrete_replicator

CHAPTER 6

utils

A package containing utilities for use elsewhere

Modules:

eventemitter Implements an EventEmitter model

fake_server Implements a fake pp-like server interface for running things one-at-a-time (not used)

functions Contains utility functions

optionparser Extends `optparse.OptionParser` to add error and exit handlers

6.1 eventemitter

A port of the node.js EventEmitter functionality

Classes:

EventEmitter implements the event emitter functionality

class `simulations.utils.eventemitter.`**EventEmitter**
Bases: `object`

Handles event emitting and listening

Public Methods:

add_listener() add a listener for an event

on() alias for `add_listener()`

once() adds a listener, but only executes it once, then it is removed

emit() trigger the listeners for an event

remove_listener() remove a listener from an event

remove_all_listeners() remove all listeners from an event

listeners() get a copy of the listeners on an event

set_max_listeners() set the maximum number of listeners for an event before warnings are issued (default: 10, None for no limit)

add_listener(*event*, *listener=None*)
Adds a listener to an event

Parameters:

event the event to listen for

listener the handler for the event (should be a function / callable)

emit (*event*, **args*, ***kwargs*)
Emit an event, triggering the handlers on it with certain arguments

Parameters:

event the event to trigger

args arguments to pass to the triggered listeners

kwargs keyword arguments to pass to the triggered listeners

listeners (*event*)
Gets a COPY of the list of listeners on an event

Parameters:

event the event for which to lookup the listeners

max_errstr = '"\n WARNING: Possible EventEmitter memory leak:\n {0} listeners added for {1}. Use set_max_listener

on (**args*)
Alias for add_listener

once (*event*, *listener=None*)

Add a listener, but only execute it the first time the event occurs, then remove it

Parameters:

event the event to listen for

listener the listener function / callable

remove_all_listeners (*event*)
Clears all listeners from an event

Parameters:

event the event to clear listeners from

remove_listener (*event*, *listener=None*)
Remove a listener from an event

Parameters:

event the event from which to remove it

listener the handler to remove

set_max_listeners (*max_listeners*)
Set the maximum number of listeners for each event.

Parameters:

max_listeners the maximum number of listeners allowed (None for no limit)

6.2 fake_server

Has a fake class that simulates the interface of pp.Server but is only single-threaded

Classes:

Server A fake pp.Server class

class simulations.utils.fake_server.**DestroyedServerError**
Bases: object

class simulations.utils.fake_server.**Server**(*ncpus='autodetect'*, *ppservers=()*, *secret=None*, *restart=False*, *proto=2*)
Bases: object

A fake pp.Server implementation – just single-ordered.

default_port = 60000

default_secret = 'epo20pdosl;dksldkmm'

destroy()
Sets _destroyed flag

get_active_nodes()
Returns {'fake': 1}

get_ncpus()
Returns 1

get_stats()
Returns an empty dictionary

print_stats()
Does nothing

set_ncpus(*ncpus='autodetect'*)
Does nothing

submit(*func*, *args*, *depfuncs=()*, *modules=()*, *callback=None*, *callbackargs=()*, *group='default'*, *globls=None*)
Runs a task, calling any callback provided and returning a function that returns the results. (Emulates pp.Server)

NOTE: the depfuncs, modules, globals, and group parameters are not used.

wait(*group=None*)
Does nothing

6.3 functions

Provides utility functions for the framework

Functions:

random_string() generates a random string of characters

simulations.utils.functions.**random_string**(*size=6*, *chars='ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'*)
Generates a random string of characters

Parameters:

size the length of the string (default 6)

chars the list of characters to use in the generation

6.4 optionparser

An extension of optparse.OptionParser that has custom error and exit handling

Classes:

OptionParser the extension class of optparse.OptionParser

class simulations.utils.optionparser.**OptionParser**(**args*, ***kwdargs*)
Bases: optparse.OptionParser

Overrides the error() and exit() methods to allow prevention of auto-exit

New Methods:

set_error_handler() sets an error handler instead of the default

set_exit_handler() sets an exit handler instead of the default

error(*msg*)
Declares a user-defined error has occurred.

Parameters:

msg The error message string

exit(*code=0*, *msg=None*)

Exits the parser/program (the default calls sys.exit). Often called by OptionParser.error().

Parameters:

code The exit code

msg The error message

set_error_handler(*handler*)
Sets an error handling function

Parameters:

handler A function that takes an error message and does something with it.

set_exit_handler(*handler*)
Sets an exit handling function

Parameters:

handler A function that takes an exit code and error message and does something with it.

CHAPTER 7

Examples

7.1 One Population Prisoner's Dilemma Example

The following code is an example of a simulation that runs a Prisoner's Dilemma game under one-population discrete-time replicator dynamics.

If you have downloaded the source code, the script is examples/one_population_pd.py

```
""" An example simulation for a Prisoner's Dilemma using one-population discrete time replicator dyna

"""

from simulations.simulation_runner import SimulationRunner
from simulations.dynamics.onepop_discrete_replicator import OnePopDiscreteReplicatorDynamics
from simulations.base import listener
from simulations.base import once


def firstgen(this, gennum, thisgen, lastgen):
    """ Prints 'Working...' after the first generation step is complete

    """

    print >> this.out, 'Working...'


def initialset(this, firstpop):
    """ Prints out a notice for the first population being chosen

    """

    print >> this.out, 'Initial population selected.'


def stablestate(this, genct, thisgen, lastgen, firstgen):
    """ Prints a notice when a stable state is reached

    """

    print >> this.out, 'Stable state reached!'


def forcestop(this, genct, thisgen, lastgen, firstgen):
```

```
36      """ Prints a notice when the simulation is force-stopped
37
38      """
39
40      print >> this.out, 'Force stopped!'
41
42
43  def generation(this, genct, thisgen, lastgen):
44      """ Prints a notice that a generation is done.
45
46      """
47
48      print >> this.out, 'Generation {0} complete.'.format(genct)
49
50
51  def simdone(this, result):
52      """ Prints a notice that one of the sims has finished.
53
54      """
55
56      print "Simulation {0} complete.".format(this.finished_count)
57
58
59  def alldone(this):
60      """ Prints a notice when all simulations are done.
61
62      """
63
64      print "All done."
65
66
67  @listener('generation', generation)
68  @listener('force stop', forcestop)
69  @listener('stable state', stablestate)
70  @listener('initial set', initialset)
71  @once('generation', firstgen)
72  class PrisonersDilemmaSim(OnePopDiscreteReplicatorDynamics):
73
74      _payoffs = [[3, 0], [4, 1]]
75
76      def __init__(self, *args, **kwdargs):
77          super(PrisonersDilemmaSim, self).__init__(*args, **kwdargs)
78
79          self.types = ['C', 'D']
80
81      def _interaction(self, me, profile):
82
83          if me == 0 or me == 1:
84              return self._payoffs[profile[me]][profile[1 - me]]
85          else:
86              raise ValueError("Profile index out of bounds")
87
88
89  @listener('result', simdone)
90  @listener('done', alldone)
91  class PDSimRunner(SimulationRunner):
92      pass
93
```

```
94
95  if __name__ == '__main__':
96      runner = PDSimRunner(PrisonersDilemmaSim)
97      runner.go()
```

7.2 Two Population Hawk-Dove Example

The following code is an example of a simulation that runs a Hawk-Dove game under two-population discrete-time replicator dynamics.

If you have downloaded the source code, the script is examples/two_population_hd.py

```
1   """ An example simulation for a Hawk-Dove game using two-population discrete time replicator dynamics
2
3   """
4
5   from simulations.simulation_runner import SimulationRunner
6   from simulations.dynamics.npop_discrete_replicator import NPopDiscreteReplicatorDynamics
7   from simulations.base import listener
8   from simulations.base import once
9
10
11  def firstgen(this, gennum, thisgen, lastgen):
12      """ Prints 'Working...' after the first generation step is complete
13
14      """
15
16      print >> this.out, 'Working...'
17
18
19  def initialset(this, firstpop):
20      """ Prints out a notice for the first population being chosen
21
22      """
23
24      print >> this.out, 'Initial population selected.'
25
26
27  def stablestate(this, genct, thisgen, lastgen, firstgen):
28      """ Prints a notice when a stable state is reached
29
30      """
31
32      print >> this.out, 'Stable state reached!'
33
34
35  def forcestop(this, genct, thisgen, lastgen, firstgen):
36      """ Prints a notice when the simulation is force-stopped
37
38      """
39
40      print >> this.out, 'Force stopped!'
41
42
43  def generation(this, genct, thisgen, lastgen):
44      """ Prints a notice that a generation is done.
```

```
45
46      """
47
48      print >> this.out, 'Generation {0} complete.'.format(genct)
49
50
51  def simdone(this, result):
52      """ Prints a notice that one of the sims has finished.
53
54      """
55
56      print "Simulation {0} complete.".format(this.finished_count)
57
58
59  def alldone(this):
60      """ Prints a notice when all simulations are done.
61
62      """
63
64      print "All done."
65
66
67  @listener('generation', generation)
68  @listener('force stop', forcestop)
69  @listener('stable state', stablestate)
70  @listener('initial set', initialset)
71  @once('generation', firstgen)
72  class HawkDoveSim(NPopDiscreteReplicatorDynamics):
73
74      _payoffs = [[0, 4], [1, 2]]
75
76      def __init__(self, *args, **kwdargs):
77          super(HawkDoveSim, self).__init__(*args, **kwdargs)
78
79      def _default_types(self):
80          return [['H', 'D'], ['H', 'D']]
81
82      def _interaction(self, me, profile):
83
84          if me == 0 or me == 1:
85              return self._payoffs[profile[me]][profile[1 - me]]
86          else:
87              raise ValueError("Profile index out of bounds")
88
89
90  @listener('result', simdone)
91  @listener('done', alldone)
92  class HDSimRunner(SimulationRunner):
93      pass
94
95  if __name__ == '__main__':
96      runner = HDSimRunner(HawkDoveSim)
97      runner.go()
```

CHAPTER 8

Indices and tables

- *genindex*
- *modindex*
- *search*

S