SimuPy Documentation

Release 1.0.0

Benjamin W. L. Margolis

Dec 13, 2018

Contents

A Python framework for modeling and simulating dynamical systems.

CHAPTER 1

SimuPy

SimuPy is a framework for simulating interconnected dynamical system models and provides an open source, python-based tool that can be used in model- and system- based design and simulation workflows. Dynamical system models can be specified as an object with the interface described in the *API Documentation*. Models can also be constructed using symbolic expressions, as in

```
from sympy.physics.mechanics import dynamicsymbols
from sympy.tensor.array import Array
from simupy.systems.symbolic import DynamicalSystem

x = x1, x2, x3 = Array(dynamicsymbols('x1:4'))
u = dynamicsymbols('u')
sys = DynamicalSystem(Array([-x1+x2-x3, -x1*x2-x2+u, -x1+u]), x, u)
```

which will automatically create callable functions for the state equations, output equations, and jacobians. By default, the code generator uses a wrapper for `sympy.lambdify`. You can change it by passing the system initialization arguments `code_generator` (the function) and additional keyword arguments to the generator in a dictionary `code_generator_args`. You can change the defaults for future systems by changing the module variables

```
import simupy.systems.symbolic
simupy.systems.symbolic.DEFAULT_CODE_GENERATOR = your_code_generator_function
simupy.systems.symbolic.DEFAULT_CODE_GENERATOR_ARGS = {'extra_arg': value}
```

A number of helper classes/functions exist to simplify the construction of models. For example, a linear feedback controller can be defined as

```
from simupy.systems import LTISystem
ctrl = LTISystem([[1.73992128, 0.99212953,  -2.98819041]])
```

The gains in the example come from the infinite horizon LQR based on the system linearized about the origin. A block diagram of the system under feedback control can be constructed

```
from simupy.block_diagram import BlockDiagram
BD = BlockDiagram(sys, ctrl)
```

(continues on next page)

(continued from previous page)

```
BD.connect(sys, ctrl) # connect the current state to the feedback controller
BD.connect(ctrl, sys) # connect the controlled input to the system
```

Initial conditions for systems with non-zero dimensional state can be defined (it defaults to zeros of the appropriate dimension) and the interconnected systems can be simulated with the `BlockDiagram`'s `simulate` method,

```
sys.initial_condition = [5, -3, 1]
res = BD.simulate(10)
```

which uses `scipy.integrate.ode` as the default solver for the initial-valued problem. The results are an instance of the `SimulationResult` class, with array attributes `t`, `x`, `y`, and `e`, holding time, state, output, and event values for each integrator time step. The first axis indexes the time step. For `x`, `y`, and `e`, the second axis indexes the individual signal components, ordered first by the order each system was added to the block diagram then according to the system state and output specification. The simulation defaults to the `dopri5` solver with dense output, but a different `integrator_class` and `integrator_options` options can be used as long as it supports a subset of the `scipy.integrate.ode` API. The default values used for future simulations can be changed following the pattern for the symbolic code generator options.

A number of utilities for constructing and manipulating systems and the simulation results are also included:

- `process_vector_args` and `lambdify_with_vector_args` from `simupy.utils.symbolic` are helpers for code generation using `sympy.lambdify`
- `simupy.utils.callable_from_trajectory` is a simple wrapper for making polynomial spline interpolators using `scipy.interpolate.splprep`
- `simupy.matrices` includes tools for constructing (vector) systems using matrix expressions and re-wrapping the results into matrix form
- `simupy.systems.SystemFromCallable` is a helper for converting a function to a state-less system (typically a controller) to simulate
- `MemorylessSystem` and `LTISystem` are subclasses to more quickly create these types of systems
- `SwitchedSystem` is used to construct systems with discontinuities, defined by zero-crossings of the `event_equation_function` output.

The examples subdirectory includes a number of worked problems. The documentation and docstrings are also available for reference.

1.1 Installation

SimuPy is `pip` installable

```
$ pip install simupy
```

SimuPy has been tested locally against

- Python >= 3.6
- NumPy >= 1.11
- SciPy >= 0.18
- SymPy >= 1.0

but tests on Travis may run with newer versions. Much of the functionality works without SymPy, so installation does not require it. The examples use matplotlib to visualize the results. Testing uses pytest. The documents are built with Sphinx == 1.6.3.

1.2 Contributing

1. To discuss problems or feature requests, file an issue. For bugs, please include as much information as possible, including operating system, python version, and version of all dependencies.
2. To contribute, make a pull request. Contributions should include tests for any new features/bug fixes and follow best practices including PEP8, etc.

CHAPTER 2

Mathematical Formulation

SimuPy assumes systems have no direct feedthrough between inputs and outputs; this discpline avoids algebraic loops. You can simulate a system model that includes a feedthrough by augmenting the system. Augment the system using the input by including input components in the state and using derivatives of those signals in the control input. You can augment the system using the output by including the original output components in the state and using integrals of those signals in the system output. However, there is no requirement for the system to have a state, so

$$\begin{aligned} x'(t) &= f(t, x(t), u(t)) \\ y(t) &= h(t, x(t)) \end{aligned}$$

and

$$y(t) = h(t, u(t))$$

are both valid formulations. Here, t is the time variable, x is the system state, u is the system input, and y is the sytem output. We call f the state equation and h the output equation. SimuPy can also handle discrete-time systems with sample period Δt of the form

$$\begin{aligned} x[k+1] &= f([k], x[k], u(k)]) \\ y[k+1] &= h([k], x[k+1]) \end{aligned}$$

and

$$y[k+1] = h([k], u(k))$$

where $[k]$ indicates signal values over the half-open interval $(k\,\Delta t, (k+1)\Delta t]$ which are updated at time $t = k\,\Delta t$ for discrete-time systems and (k) indicates a zero-order hold sample of the signal at time $k\,\Delta t$ for continuous-time systems. This formulation gives the expected results for models with only discrete-time sub-systems of the same update rate Δt which can be combined into a single system of the form

$$\begin{aligned} x[k+1] &= f([k], x[k], u[k]) \\ y[k] &= h([k], x[k]) \end{aligned}$$

and makes sense in general for hybrid-time simulation.

This formulation is also consistent with common linear, time-invariant (LTI) system algebras and transformations. For example, the dynamics of the LTI system

$$\begin{aligned} x'(t) &= A\,x(t) + B\,u(t), \\ y(t) &= I\,x(t), \end{aligned}$$

with state-feedback

$$u(t) = -K\,x(t),$$

are the same as the autonomous system

$$\begin{aligned} x'(t) &= (A - B\,K)\,x(t), \\ y(t) &= I\,x(t). \end{aligned}$$

Similarly, timing transformations are consistent. The discrete-time equivalent of the continuous-time LTI system above,

$$\begin{aligned} x[k+1] &= \Phi\,x[k] + \Gamma\,u[k], \\ y[k] &= I\,x[k], \end{aligned}$$

will travel through the same state trajectory at times $k\,\Delta t$ if both are subject to the same piecewise constant inputs and the state and input matrices are related by the zero-order hold transformation

$$\begin{aligned} \Phi &= e^{A\,\Delta t}, \\ \Gamma &= \int_0^{\Delta t} e^{A\,\tau}\,d\tau B. \end{aligned}$$

The accuracy of these algebras and transformations are demonstrated in the `discrete_lti.py` example and are incorporated into the `test_block_diagram.py` tests.

CHAPTER 3

API Documentation

A system in a `BlockDiagram` needs to provide the following attributes:

- `dim_state` : the dimension of the state
- `dim_input` : the dimension of the input
- `dim_output` : the dimension of the output
- `output_equation_function` : A callable returning the system output.

If `dim_state`=0, then `output_equation_function` recieves the current time and input as arguments during integration. If `dim_state`>0 then `state_equation_function`, taking the current time, state, and input and returning the state derivative, must also be provided. In this case, `output_equation_function` recieves the current time and state as arguments during integration.

If `event_equation_function` and `update_equation_function` are provided, discontinuities at zero-crossing of `event_equation_function` are handled. The argument rules for `event_equation_function` and `update_equation_function` during integration are the same as for `output_equation_function` and `state_equation_function`, respectively. Generally, `update_equation_function` is used to change what `state_equation_function`, `output_equation_function`, and `event_equation_function` compute based on the occurance of the discontinuity. If `dim_state`>0, `update_equation_function` must return the state immediately after the discontinuity.

The base system class takes a convenience input argument, `dt`. Passing `dt`>0 will determine the sample rate that the outputs and state are computed; `dt`=0 is treated as a continuous-time system. In hybrid-time `BlockDiagram`s, the system is automatically integrated piecewise to improve accuracy.

Future versions of SimuPy may support passing jacobian functions to ode solvers if all systems in the `BlockDiagram` provide the appropriate necessary jacobian functions.

A quick overview of the of the modules:

`block_diagram` (*docstrings*) implements the `BlockDiagram` class to simulate interconnected systems.

`systems` (*docstrings*) provides a few base classes for purely numerical based systems.

`utils` (*docstrings*) provides utility functions, such as manipulating (numeric) systems and simulation results.

systems.symbolic (*docstrings*) and discontinuities (*docstrings*) provides niceties for using symbolic expressions to define systems.

array (*docstrings*) and matrices (*docstrings*) provide helper functions and classes for manipulating symbolic arrays, matrices, and their systems.

utils.symbolic (*docstrings*) provides utility symbolic functions, such as manipulating symbolic systems.

3.1 block_diagram module

class simupy.block_diagram.**BlockDiagram**(**systems*)

A block diagram of dynamical systems with their connections which can be numerically simulated.

Initialize a BlockDiagram, with an optional list of systems to start the diagram.

add_system(*system*)

Add a system to the block diagram

Parameters **system** (*dynamical system*) – System to add to BlockDiagram

computation_step(*t*, *state*, *output=None*, *selector=True*, *do_events=False*)

callable to compute system outputs and state derivatives

connect(*from_system_output*, *to_system_input*, *outputs=[]*, *inputs=[]*)

Connect systems in the block diagram.

Parameters

- **from_system_output** (*dynamical system*) – The system (already added to BlockDiagram) from which outputs will be connected. Note that the outputs of a system can be connected to multiple inputs.
- **to_system_input** (*dynamical system*) – The system (already added to BlockDiagram) to which inputs will be connected. Note that any previous input connections will be over-written.
- **outputs** (*list-like, optional*) – Selector index of the outputs to connect. If not specified or of length 0, will connect all of the outputs.
- **inputs** (*list-like, optional*) – Selector index of the inputs to connect. If not specified or of length 0, will connect all of the inputs.

create_input(*to_system_input*, *channels=[]*, *inputs=[]*)

Create or use input channels to use block diagram as a subsystem.

Parameters

- **channels** (*list-like*) – Selector index of the input channels to connect.
- **to_system_input** (*dynamical system*) – The system (already added to BlockDiagram) to which inputs will be connected. Note that any previous input connections will be over-written.
- **inputs** (*list-like, optional*) – Selector index of the inputs to connect. If not specified or of length 0, will connect all of the inputs.

dim_output

dim_state

dt

event_equation_function_implementation(*t*, *state*, *output=None*)

`initial_condition`

`output_equation_function`(*t*, *state*, *input_=None*, *update_memoryless_event=False*)

`prepare_to_integrate`()

`simulate`(*tspan*, *integrator_class=<class 'scipy.integrate._ode.ode'>*, *integrator_options={'atol': 1e-12, 'max_step': 0.0, 'name': 'dopri5', 'nsteps': 500, 'rtol': 1e-06}*, *event_finder=<function brentq>*, *event_find_options={'maxiter': 100, 'rtol': 8.881784197001252e-16, 'xtol': 2e-12}*)

Simulate the block diagram

Parameters

- **`tspan`** (*`list-like or float`*) – Argument to specify integration time-steps.

  If a single time is specified, it is treated as the final time. If two times are specified, they are treated as initial and final times. In either of these conditions, it is assumed that that every time step from a variable time-step integrator will be stored in the result.

  If more than two times are specified, these are the only times where the trajectories will be stored.

- **`integrator_class`** (*`class, optional`*) – Class of integrator to use. Defaults to `scipy.integrate.ode`. Must provide the following subset of the `scipy.integrate.ode` API:

  – `__init__(derivative_callable(time, state))`

  – `set_integrator(**kwargs)`

  – `set_initial_value(state, time)`

  – `set_solout(successful_step_callable(time, state))`

  – `integrate(time)`

  – `successful()`

  – `y`, `t` properties

- **`integrator_options`** (*`dict, optional`*) – Dictionary of keyword arguments to pass to `integrator_class.set_integrator`.

- **`event_finder`** (*`callable, optional`*) – Interval root-finder function. Defaults to `scipy.optimize.brentq`, and must take the equivalent positional arguments, `f`, `a`, and `b`, and return `x0`, where `a <= x0 <= b` and `f(x0)` is the zero.

- **`event_find_options`** (*`dict, optional`*) – Dictionary of keyword arguments to pass to `event_finder`. It must provide a key `'xtol'`, and it is expected that the exact zero lies within `x0 +/- xtol/2`, as `brentq` provides.

`state_equation_function`(*t*, *state*, *input_=None*, *output=None*)

`systems_event_equation_functions`(*t*, *state*, *output*)

`update_equation_function_implementation`(*t*, *state*, *input_=None*, *output=None*)

class `simupy.block_diagram.`**`SimulationResult`**(*dim_states*, *dim_outputs*, *tspan*, *n_sys*, *initial_size=0*)

A simple class to collect simulation result trajectories.

`t`

Type array of times

`x`

Type array of states

y

Type array of outputs

e

Type array of events

allocate_space(*t*)

last_result(*n=1*, *copy=False*)

max_allocation = 128

new_result(*t*, *x*, *y*, *e=None*)

3.2 systems module

class simupy.systems.**DynamicalSystem**(*state_equation_function=None*, *output_equation_function=None*, *event_equation_function=None*, *update_equation_function=None*, *dim_state=0*, *dim_input=0*, *dim_output=0*, *dt=0*, *initial_condition=None*)

Bases: `object`

A dynamical system which models systems of the form:

```
xdot(t) = state_equation_function(t,x,u)
y(t) = output_equation_function(t,x)
```

or:

```
y(t) = output_equation_function(t,u)
```

These could also represent discrete-time systems, in which case xdot(t) represents x[k+1].

This can also model discontinuous systems. Discontinuities must occur on zero-crossings of the `event_equation_function`, which take the same arguments as `output_equation_function`, depending on `dim_state`. At the zero-crossing, `update_equation_function` is called with the same arguments. If `dim_state` > 0, the return value of `update_equation_function` is used as the state of the system immediately after the discontinuity.

Parameters

- **state_equation_function** (*callable, optional*) – The derivative (or update equation) of the system state. Not needed if `dim_state` is zero.
- **output_equation_function** (*callable, optional*) – The output equation of the system. A system must have an `output_equation_function`. If not set, uses full state output.
- **event_equation_function** (*callable, optional*) – The function whose output determines when discontinuities occur.
- **update_equation_function** (*callable, optional*) – The function called when a discontinuity occurs.

- **dim_state** (*int, optional*) – Dimension of the system state. Optional, defaults to 0.
- **dim_input** (*int, optional*) – Dimension of the system input. Optional, defaults to 0.
- **dim_output** (*int, optional*) – Dimension of the system output. Optional, defaults to dim_state.
- **dt** (*float, optional*) – Sample rate of the system. Optional, defaults to 0 representing a continuous time system.
- **initial_condition** (*array_like of numerical values, optional*) – Array or Matrix used as the initial condition of the system. Defaults to zeros of the same dimension as the state.

dt

initial_condition

prepare_to_integrate()

validate()

class simupy.systems.**LTISystem**(**args*, *initial_condition=None*, *dt=0*)

Bases: *simupy.systems.DynamicalSystem*

A linear, time-invariant system.

Construct an LTI system with the following input formats:

1. state matrix A, input matrix B, output matrix C for systems with state:

```
dx_dt = Ax + Bu
y = Hx
```

2. state matrix A, input matrix B for systems with state, assume full state output:

```
dx_dt = Ax + Bu
y = Ix
```

3. gain matrix K for systems without state:

```
y = Kx
```

The matrices should be numeric arrays of consistent shape. The class provides `A`, `B`, `C` and `F`, `G`, `H` aliases for the matrices of systems with state, as well as a `K` alias for the gain matrix. The `data` alias provides the matrices as a tuple.

A

B

C

F

G

H

K

data

validate()

class simupy.systems.**SwitchedSystem**(*state_equations_functions=None*, *output_equations_functions=None*, *event_variable_equation_function=None*, *event_bounds=None*, *state_update_equation_function=None*, *dim_state=0*, *dim_input=0*, *dim_output=0*, *initial_condition=None*)

Bases: `simupy.systems.DynamicalSystem`

Provides a useful pattern for discontinuous systems where the state and output equations change depending on the value of a function of the state and/or input (`event_variable_equation_function`). Most of the usefulness comes from constructing the `event_equation_function` with a Bernstein basis polynomial with roots at the boundaries. This class also provides logic for outputting the correct state and output equation based on the `event_variable_equation_function` value.

Parameters

- **state_equations_functions** (*array_like of callables, optional*) – The derivative (or update equation) of the system state. Not needed if `dim_state` is zero. The array indexes the event-state and should be one more than the number of event bounds. This should also be indexed to match the boundaries (i.e., the first function is used when the event variable is below the first event_bounds value). If only one callable is provided, the callable is used in each condition.
- **output_equations_functions** (*array_like of callables, optional*) – The output equation of the system. A system must have an `output_equation_function`. If not set, uses full state output. The array indexes the event-state and should be one more than the number of event bounds. This should also be indexed to match the boundaries (i.e., the first function is used when the event variable is below the first event_bounds value). If only one callable is provided, the callable is used in each condition.
- **event_variable_equation_function** (*callable*) – When the output of this function crosses the values in `event_bounds`, a discontuity event occurs.
- **event_bounds** (*array_like of floats*) – Defines the boundary points the trigger discontinuity events based on the output of `event_variable_equation_function`.
- **state_update_equation_function** (*callable, optional*) – When an event occurs, the state update equation function is called to determine the state update. If not set, uses full state output, so the state is not changed upon a zero-crossing of the event variable function.
- **dim_state** (*int, optional*) – Dimension of the system state. Optional, defaults to 0.
- **dim_input** (*int, optional*) – Dimension of the system input. Optional, defaults to 0.
- **dim_output** (*int, optional*) – Dimension of the system output. Optional, defaults to dim_state.

event_bounds

event_equation_function(**args*)

output_equation_function(**args*)

prepare_to_integrate()

state_equation_function(**args*)

update_equation_function(**args*)

validate()

simupy.systems.**SystemFromCallable**(*incallable*, *dim_input*, *dim_output*, *dt=0*)
Construct a memoryless system from a callable.

Parameters

- **incallable** (*callable*) – Function to use as the output_equation_function. Should have signature (t, u) if dim_input > 0 or (t) if dim_input = 0.
- **dim_input** (*int*) – Dimension of input.
- **dim_output** (*int*) – Dimension of output.

simupy.systems.**full_state_output**(**args*)
A drop-in `output_equation_function` for stateful systems that provide output the full state directly.

3.3 utils module

simupy.utils.**array_callable_from_vector_trajectory**(*tt*, *x*, *unraveled*, *raveled*)
Convert a trajectory into an interpolating callable that returns a 2D array. The unraveled, raveled pair map how the array is filled in. See riccati_system example.

Parameters

- **tt** (*1D array_like*) – Array of m time indices of trajectory
- **xx** (*2D array_like*) – Array of m x n vector samples at the time indices. First dimension indexes time, second dimension indexes vector components
- **unraveled** (*1D array_like*) – Array of n unique keys matching xx.
- **raveled** (*2D array_like*) – Array where the elements are the keys from unraveled. The mapping between unraveled and raveled is used to specify how the output array is filled in.

Returns **matrix_callable** – The callable interpolating the trajectory with the specified shape.

Return type callable

simupy.utils.**callable_from_trajectory**(*t*, *curves*)
Use scipy.interpolate splprep to build cubic b-spline interpolating functions over a set of curves.

Parameters

- **t** (*1D array_like*) – Array of m time indices of trajectory
- **curves** (*2D array_like*) – Array of m x n vector samples at the time indices. First dimension indexes time, second dimension indexes vector components

Returns **interpolated_callable** – Callable which interpolates the given curve/trajectories

Return type callable

simupy.utils.**discrete_callable_from_trajectory**(*t*, *curves*)
Build a callable that interpolates a discrete-time curve by returning the value of the previous time-step.

Parameters

- **t** (*1D array_like*) – Array of m time indices of trajectory

- **curves** (*2D array_like*) – Array of m x n vector samples at the time indices. First dimension indexes time, second dimension indexes vector components

Returns nearest_neighbor_callable – Callable which interpolates the given discrete-time curve/trajectories

Return type callable

3.4 symbolic systems module

class simupy.systems.symbolic.**DynamicalSystem**(*state_equation=None, state=None, input_=None, output_equation=None, constants_values={}, dt=0, initial_condition=None, code_generator=None, code_generator_args={}*)

Bases: *simupy.systems.DynamicalSystem*

DynamicalSystem constructor, used to create systems from symbolic expressions.

Parameters

- **state_equation** (*array_like of sympy Expressions, optional*) – Vector valued expression for the derivative of the state.
- **state** (*array_like of sympy symbols, optional*) – Vector of symbols representing the components of the state, in the desired order, matching state_equation.
- **input** (*array_like of sympy symbols, optional*) – Vector of symbols representing the components of the input, in the desired order. state_equation may depend on the system input. If the system has no state, the output_equation may depend on the system input.
- **output_equation** (*array_like of sympy Expressions*) – Vector valued expression for the output of the system.
- **constants_values** (*dict*) – Dictionary of constants substitutions.
- **dt** (*float*) – Sampling rate of system. Use 0 for continuous time systems.
- **initial_condition** (*array_like of numerical values, optional*) – Array or Matrix used as the initial condition of the system. Defaults to zeros of the same dimension as the state.
- **code_generator** (*callable, optional*) – Function to be used as code generator.
- **code_generator_args** (*dict, optional*) – Dictionary of keyword args to pass to the code generator.

By default, the code generator uses a wrapper for `sympy.lambdify`. You can change it by passing the system initialization arguments `code_generator` (the function) and additional keyword arguments to the generator in a dictionary `code_generator_args`. You can change the defaults for future systems by changing the module values. See the readme or docs for an example.

copy()

equilibrium_points(*input_=None*)

input

output_equation

prepare_to_integrate()

state

state_equation

update_input_jacobian_function()

update_output_equation_function()

update_state_equation_function()

update_state_jacobian_function()

class simupy.systems.symbolic.**MemorylessSystem**(*input_=None*, *output_equation=None*, ***kwargs*)

Bases: *simupy.systems.symbolic.DynamicalSystem*

A system with no state.

With no input, can represent a signal (function of time only). For example, a stochastic signal could interpolate points and use prepare_to_integrate to re-seed the data.

DynamicalSystem constructor

Parameters

- **input** (*array_like of sympy symbols*) – Vector of symbols representing the components of the input, in the desired order. The output may depend on the system input.
- **output_equation** (*array_like of sympy Expressions*) – Vector valued expression for the output of the system.

state

3.5 discontinuities module

class simupy.discontinuities.**DiscontinuousSystem**(*state_equation=None*, *state=None*, *input_=None*, *output_equation=None*, *constants_values={}*, *dt=0*, *initial_condition=None*, *code_generator=None*, *code_generator_args={}*)

Bases: *simupy.systems.symbolic.DynamicalSystem*

A continuous-time dynamical system with a discontinuity. Must provide the following attributes in addition to those of DynamicalSystem:

`event_equation_function` - A function called at each integration time- step and stored in simulation results. Takes input and state, if stateful. A zero-crossing of this output triggers the discontinuity.

`event_equation_function` - A function that is called when the discontinuity occurs. This is generally used to change what `state_equation_function`, `output_equation_function`, and `event_equation_function` compute based on the occurance of the discontinuity. If stateful, returns the state immediately after the discontinuity.

DynamicalSystem constructor, used to create systems from symbolic expressions.

Parameters

- **state_equation** (*array_like of sympy Expressions, optional*) – Vector valued expression for the derivative of the state.
- **state** (*array_like of sympy symbols, optional*) – Vector of symbols representing the components of the state, in the desired order, matching state_equation.
- **input** (*array_like of sympy symbols, optional*) – Vector of symbols representing the components of the input, in the desired order. state_equation may depend on the system input. If the system has no state, the output_equation may depend on the system input.
- **output_equation** (*array_like of sympy Expressions*) – Vector valued expression for the output of the system.
- **constants_values** (*dict*) – Dictionary of constants substitutions.
- **dt** (*float*) – Sampling rate of system. Use 0 for continuous time systems.
- **initial_condition** (*array_like of numerical values, optional*) – Array or Matrix used as the initial condition of the system. Defaults to zeros of the same dimension as the state.
- **code_generator** (*callable, optional*) – Function to be used as code generator.
- **code_generator_args** (*dict, optional*) – Dictionary of keyword args to pass to the code generator.

By default, the code generator uses a wrapper for `sympy.lambdify`. You can change it by passing the system initialization arguments `code_generator` (the function) and additional keyword arguments to the generator in a dictionary `code_generator_args`. You can change the defaults for future systems by changing the module values. See the readme or docs for an example.

dt

event_equation_function(**args*, ***kwargs*)

update_equation_function(**args*, ***kwargs*)

class simupy.discontinuities.**MemorylessDiscontinuousSystem**(*input_=None*, *output_equation=None*, ***kwargs*)

Bases: *simupy.discontinuities.DiscontinuousSystem*, *simupy.systems.symbolic.MemorylessSystem*

DynamicalSystem constructor

Parameters

- **input** (*array_like of sympy symbols*) – Vector of symbols representing the components of the input, in the desired order. The output may depend on the system input.
- **output_equation** (*array_like of sympy Expressions*) – Vector valued expression for the output of the system.

class simupy.discontinuities.**SwitchedOutput**(*event_variable_equation*, *event_bounds_expressions*, *state_equations=None*, *output_equations=None*, *state_update_equation=None*, ***kwargs*)

Bases: *simupy.discontinuities.SwitchedSystem*, *simupy.discontinuities.MemorylessDiscontinuousSystem*

A memoryless discontinuous system to conveninetly construct switched outputs.

SwitchedSystem constructor, used to create switched systems from symbolic expressions. The parameters below are in addition to parameters from the `systems.symbolic.DynamicalSystems` constructor.

Parameters

- **`event_variable_equation`** (*`sympy Expression`*) – Expression representing the event_equation_function
- **`event_bounds_expressions`** (*`list-like of sympy Expressions or floats`*) – Ordered list-like values which define the boundaries of events (relative to event_variable_equation).
- **`state_equations`** (*`array_like of sympy Expressions, optional`*) – The state equations of the system. The first dimension indexes the event-state and should be one more than the number of event bounds. This should also be indexed to match the boundaries (i.e., the first expression is used when the event_variable_equation is below the first event_bounds value). The second dimension is dim_state of the system. If only 1-D, uses single equation for every condition.
- **`output_equations`** (*`array_like of sympy Expressions, optional`*) – The output equations of the system. The first dimension indexes the event-state and should be one more than the number of event bounds. This should also be indexed to match the boundaries (i.e., the first expression is used when the event_variable_equation is below the first event_bounds value). The second dimension is dim_output of the system. If only 1-D, uses single equation for every condition.
- **`state_update_equation`** (*`sympy Expression`*) – Expression representing the state_update_equation_function

class `simupy.discontinuities.`**`SwitchedSystem`**(*event_variable_equation*, *event_bounds_expressions*, *state_equations=None*, *output_equations=None*, *state_update_equation=None*, ***kwargs*)

Bases: `simupy.systems.SwitchedSystem`, `simupy.discontinuities.DiscontinuousSystem`

SwitchedSystem constructor, used to create switched systems from symbolic expressions. The parameters below are in addition to parameters from the `systems.symbolic.DynamicalSystems` constructor.

Parameters

- **`event_variable_equation`** (*`sympy Expression`*) – Expression representing the event_equation_function
- **`event_bounds_expressions`** (*`list-like of sympy Expressions or floats`*) – Ordered list-like values which define the boundaries of events (relative to event_variable_equation).
- **`state_equations`** (*`array_like of sympy Expressions, optional`*) – The state equations of the system. The first dimension indexes the event-state and should be one more than the number of event bounds. This should also be indexed to match the boundaries (i.e., the first expression is used when the event_variable_equation is below the first event_bounds value). The second dimension is dim_state of the system. If only 1-D, uses single equation for every condition.
- **`output_equations`** (*`array_like of sympy Expressions, optional`*) – The output equations of the system. The first dimension indexes the event-state and should be one more than the number of event bounds. This should also be indexed to match the boundaries (i.e., the first expression is used when the event_variable_equation is below the

first event_bounds value). The second dimension is dim_output of the system. If only 1-D, uses single equation for every condition.

- **state_update_equation** (*sympy Expression*) – Expression representing the state_update_equation_function

event_bounds_expressions

event_variable_equation

output_equations

prepare_to_integrate()

state_equations

state_update_equation

validate(*from_self=False*)

3.6 array module

class simupy.array.**SymAxisConcatenatorMixin**

Bases: object

A mix-in to convert numpy AxisConcatenator classes to use with sympy N-D arrays.

static concatenate(**args*, ***kwargs*)

makemat

alias of sympy.matrices.immutable.ImmutableDenseMatrix

class simupy.array.**SymCClass**

Bases: *simupy.array.SymAxisConcatenatorMixin*, numpy.lib.index_tricks.CClass

class simupy.array.**SymRClass**

Bases: *simupy.array.SymAxisConcatenatorMixin*, numpy.lib.index_tricks.RClass

simupy.array.**empty_array**()

Construct an empty array, which is often needed as a place-holder

3.7 matrices module

simupy.matrices.**block_matrix**(*blocks*)

Construct a matrix where the elements are specified by the block structure by joining the blocks appropriately.

Parameters blocks (*two level deep iterable of sympy Matrix objects*) – The block specification of the matrices used to construct the block matrix.

Returns matrix – A matrix whose elements are the elements of the blocks with the specified block structure.

Return type sympy Matrix

simupy.matrices.**construct_explicit_matrix**(*name*, *n*, *m*, *symmetric=False*, *diagonal=0*, *dynamic=False*, ***kwass*)

construct a matrix of symbolic elements

Parameters

- **name** (*string*) – Base name for variables; each variable is name_ij, which admitedly only works clearly for n,m < 10
- **n** (*int*) – Number of rows
- **m** (*int*) – Number of columns
- **symmetric** (*bool, optional*) – Use to enforce a symmetric matrix (repeat symbols above/below diagonal)
- **diagonal** (*bool, optional*) – Zeros out off diagonals. Takes precedence over symmetry.
- **dynamic** (*bool, optional*) – Whether to use sympy.physics.mechanics dynamicsymbol. If False, use sp.symbols
- **kwargs** (*dict*) – remaining kwargs passed to symbol function

Returns **matrix** – The Matrix containing explicit symbolic elements

Return type sympy Matrix

`simupy.matrices.`**`matrix_subs`**(**subs*)

Generate an object that can be passed into sp.subs from matrices, replacing each element in from_matrix with the corresponding element from to_matrix

There are three ways to use this function, depending on the input: 1. A single matrix-level subsitution - from_matrix, to_matrix 2. A list or tuple of (from_matrix, to_matrix) 2-tuples 3. A dictionary of {from_matrix: to_matrix} key-value pairs

`simupy.matrices.`**`system_from_matrix_DE`**(*mat_DE*, *mat_var*, *mat_input=None*, *constants={}*)

Construct a symbolic DynamicalSystem using matrices. See riccati_system example.

Parameters

- **mat_DE** (*sympy Matrix*) – The matrix derivative expression (right hand side)
- **mat_var** (*sympy Matrix*) – The matrix state
- **mat_input** (*list-like of input expressions, optional*) – A list-like of input expressions in the matrix differential equation
- **constants** (*dict, optional*) – Dictionary of constants substitutions.

Returns **sys** – A DynamicalSystem which can be used to numerically solve the matrix differential equation.

Return type *DynamicalSystem*

3.8 symbolic utils module

`simupy.utils.symbolic.`**`augment_input`**(*system*, *input_=[]*, *update_outputs=True*)

Augment input, useful to construct control-affine systems.

Parameters

- **system** (`DynamicalSystem`) – The sytsem to augment the input of
- **input** (*array_like of symbols, optional*) – The input to augment. Use to augment only a subset of input components.
- **update_outputs** (*boolean*) – If true and the system provides full state output, will also add the augmented inputs to the output.

`simupy.utils.symbolic.`**`grad`**(*f*, *basis*, *for_numerical=True*)

Compute the symbolic gradient of a vector-valued function with respect to a basis.

Parameters

- **`f`** (*`1D array_like of sympy Expressions`*) – The vector-valued function to compute the gradient of.
- **`basis`** (*`1D array_like of sympy symbols`*) – The basis symbols to compute the gradient with respect to.
- **`for_numerical`** (*`bool, optional`*) – A placeholder for the option of numerically computing the gradient.

Returns **grad** – The symbolic gradient.

Return type 2D array_like of sympy Expressions

`simupy.utils.symbolic.`**`lambdify_with_vector_args`**(*args*, *expr*, *modules=({'ImmutableMatrix': <class 'numpy.matrixlib.defmatrix.matrix'>}, 'numpy', {'Mod': <ufunc 'remainder'>})*)

A wrapper around sympy's lambdify where process_vector_args is used so generated callable can take arguments as either vector or individual components

Parameters

- **`args`** (*`list-like of sympy symbols`*) – Input arguments to the expression to call
- **`expr`** (*`sympy expression`*) – Expression to turn into a callable for numeric evaluation
- **`modules`** (*`list`*) – See lambdify documentation; passed directly as modules keyword.

`simupy.utils.symbolic.`**`process_vector_args`**(*args*)

A helper function to process vector arguments so callables can take vectors or individual components. Essentially unravels the arguments.

`class` `simupy.utils.symbolic.`**`sinc`**

Bases: `sympy.core.function.AppliedUndef`

`default_assumptions = {}`

`class` `simupy.utils.symbolic.`**`step`**

Bases: `sympy.core.function.AppliedUndef`

`default_assumptions = {}`

Python Module Index

Index