SimuPy 文档中文翻译

翻译者： cycleuser

SimuPy 是加州大学戴维斯分校的Benjamin W. L. Margolis 在 2017年8月29日发表的[1]。

论文 DOI: http://dx.doi.org/10.21105/joss.00396

GitHub Repository：https://github.com/simupy/simupy

官方文档：https://simupy.readthedocs.io

- 官方文档显示的更新日期是今年的4月7号，所以看来这个项目还是活跃的。

启动日期：2020年6月11日
完成日期：2020年6月13日

参考

```
^SimuPy: A Python framework for modeling and simulating dynamical systems ht
tps://joss.theoj.org/papers/10.21105/joss.00396
```

第一章 SimuPy 简介

2020年6月11日22:33

原文链接

SimuPy 是一个对互联动态系统模型（interconnected dynamical system models）进行模拟的框架，提供了一个基于 Python 的开源工具，可以用于基于模型和系统的设计与仿真工作流。动态系统模型（dynamical system models）可被指定为一个对象，具体信息参考 API 文档。模型也可以使用符号表达式（symbolic expressions）来创建，如下面的代码所示：

```
from sympy.physics.mechanics import dynamicsymbols #译者注：从sympy导入动态符号
from sympy.tensor.array import Array #译者注：从sympy的张量tensor导入数组 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)
```

上面这段代码就自动创建了状态方程（state equations），输出方程（output equations），以及雅可比方程（jacobians）的可调用函数（callable functions）。默认情况下代码生成器（code generator）使用的是对 `sympy.lambdify` 的封装（wrapper）。 你可以更改所用的代码生成器，只需传递系统初始化参数 `code_generator` (你选的函数) 和附加的关键词参数字典（译者注：这里的字典是 python的一种数据类型 dictionary） `code_generator_args`。 未来系统（future systems）的默认值可以通过修改模块变量（module variables）来调整。

```
import simupy.systems.symbolic
simupy.systems.symbolic.DEFAULT_CODE_GENERATOR = your_code_generator_functio
n #译者注：注意，这里的 'your_code_generator_function' 是让你传递一个函数，这句代码
是个示范，不能直接运行的，你要修改成你指定的函数才行，具体如何等后面的文档都翻看完了就知道了
simupy.systems.symbolic.DEFAULT_CODE_GENERATOR_ARGS = {'extra_arg': value}
```

一些帮助类、帮助函数，可以简化模型的创建。例如，线性反馈控制器（linear feedback controller）可以用如下代码定义：

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

上面例子中的增益值（gains）来自一个基于原点线性化系统到无穷时域线性二次型调节器（infinite horizon LDR，linear quadratic regulator）。在上面的控制器的基础上，就可以用下面的代码创建一个模块图（block diagram）：

```
from simupy.block_diagram import BlockDiagram
BD = BlockDiagram(sys, ctrl)
BD.connect(sys, ctrl) # 将当前状态连接到反馈控制器 ctrl
BD.connect(ctrl, sys) # 将控制输入连接到系统
```

非零维度状态（non-zero dimensional state）系统的初始条件可以定义（默认是适当维度的零值），然后互联系统可以用模块图的模拟方法进行模拟：

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

初值问题（initial-valued problem）的默认求解器是 `scipy.integrate.ode`。得到的结果是类 `SimulationResult` 的一个实例（instance），数组属性（array attributes）为 t, x, y, 和 e, 分别

表示每个积分器时间步长（integrator time step）的时间（time）、状态（state）、输出（output）、事件值（event values）。 第一个轴（axis）是使用时间步长作为索引。对 x, y, 和 e, 第二个轴使用独立信号分量（individual signal components）作为索引, 最初排序是按照每个系统加入模块图的顺序，然后根据系统状态和输出再进行排序。 模拟器默认使用的是带密度输出的 `dopri5` 求解器，这也是可以更换的，可以指定 `integrator_class` 积分器种类和 `integrator_options` 积分器选项，只要是兼容于 `scipy.integrate.ode` API的子集成员就可以。. 未来模拟（future simulations）所用的默认值也可以按照符号代码生成器选项（symbolic code generator options）来进行修改。

此外，本项目还包含了其他的一些用于创建和操作系统和仿真结果的工具：

- `process_vector_args` 和 `lambdify_with_vector_args` 是 `simupy.utils.symbolic` 之下使用 `sympy.lambdify` 进行代码生成的帮助器
- `simupy.utils.callable_from_trajectory` 是使用 `scipy.interpolate.splprep` 进行多项式样条插值的简易封装
- `simupy.matrices` 包含的工具可以使用矩阵表达式创建响亮系统，还可以将结果打包成矩阵形式
- `simupy.systems.SystemFromCallable` 是一个帮助器，可以将一个函数转换成一个无状态系统（state-less system），通常是一个控制器，以此来进行模拟
- `MemorylessSystem` 和 `LTISystem` 是两个子类，用于更快速创建对应类型的系统
- `SwitchedSystem` 用于创建不连续（discontinuities）系统, 不连续性通过事件方程输出（`event_equation_function`）过零点（`zero-crossings`）而定义。

SimuPy Github 项目的 examples 子文件夹 中有更多的工作案例。这份文档以及程序源代码中的文档字符串也可以作为参考。

安装

SimuPy 可以通过 pip 来安装：

```
ip install simupy
```

SimuPy 测试兼容的依赖包包括：

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

在 Travis 上跑的持续集成测试可能用的是更新的版本。本项目中很多函数可以脱离 SymPy 工

作，所以可以不需要安装它。 SimuPy Github 项目的 examples 子文件夹 中使用的是 matplotlib 来进行结果可视化。测试部分使用的是 pytest。原版文档使用 Sphinx == 1.6.3 生成，中文翻译版本由 CycleUser 使用 Markdown 制作。

参与贡献

```
1. 如果要讨论问题或者请求添加功能，可以去[添加新的 issue](https://github.com/simupy/simupy/issues)。如果要上报 Bug，请尽可能详细添加北京信息，比如操作系统、Python版本，以及所有依赖包的版本。
2. 若要贡献代码，可以提交 pull request。贡献内容包括对新版本的测试、bug 的调试消除，以及具体验证探索包括PEP8代码规范等等。
```

第二章 数学公式

2020年6月12日08:33

原文链接

SimuPy 假设系统在输入和输出之间没有直接馈通（direct feedthrough）；这一原则就避免了代数循环（algebraic loops）。 如果你需要模拟包含馈通（feedthrough）的系统模型，需要对系统进行扩展增强来实现。这种扩展增强的实现，在输入方面，可以通过在状态中包含输入部件，并在控制输入中使用这些信号的导数来使用输入；在输出方面，可将原始输出部分包含在状态中，然后在系统输出中使用这些信号积分。 然而系统本身并不需要有一个状态函数，所以

$$x'(t) = f(t, x(t), u(t))$$
$$y(t) = h(t, x(t))$$

和

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

就都是有效的公式（valid formulations）。上面式子中的 t 是时间变量, x是系统状态（system state）, u 是系统输入（system input）, 而 y 是系统输出（sytem output）. 可以调用 f 这个状态方程（state equation）和 h 这个输出方程（output equation）。
SimuPy 还可以处理带有采样中期（sample period） $Deltat$ 的离散时间系统（discrete-time systems），形式为：

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

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

上面式子中的 $[k]$ 表示的是一个信号值（signal values），这个信号值处在$(k\,\Delta t, (k+1)\Delta t]$ 这个半开区间（half-open interval）内，在一个离散时间系统中每隔 $t = k\,\Delta t$ 就更新一次，而 (k) 表示的是在时间$k\,\Delta t$ 对连续时间系统（continuous-time systems）的信号进行的一个零阶保持采样（zero-order hold sample）。这里的公式只对由相同更新率（update rate）Δt的离散时间子系统（discrete-time sub-systems）给出预期结果，这些子系统可以组合成一个但各系统，如下所示：

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

然后就可以适用于通用的混合时间模拟（hybrid-time simulation）。

上面的公式也符合常规 `线性时不变（linear time-invariant (LTI)）` 系统代数和变换。例如，LTI 系统的动力学（dynamics）部分：

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

状态反馈（state-feedback）：

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

和自治系统（autonomous system）中的都是一样的：

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

与之相似地，时间变换（timing transformations）也是一致的（consistent），上面的连续时间（continuous-time）LTI系统的离散时间等价形式（discrete-time equivalent）为：

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

在时间 $k\,\Delta t$，上面两种形式若要经历同样的状态轨迹（state trajectory），需要服从相同的分段常数（piecewise constant）输入，并且输入矩阵通过零阶保持变换（zero-order hold transformation）相关联

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

这些代数和变换的准确性演示参考样例中的 `discrete_lti.py`，此外也被整合进了测试中

的 `test_block_diagram.py` 。

第三章 API 文档

2020年6月12日11:55

原文链接

`BlockDiagram` 中的一个系统需要提供下面的属性（attributes）：

- `dim_state` : 状态（state）的维度（dimension）
- `dim_input` : 输入（input）的维度
- `dim_output` : 输出（output）的维度
- `output_equation_function` : 一个可调用函数，返回系统输出（system output）

若 `dim_state` =0, 则 `output_equation_function` 接收当前时间和输入作为积分（integration）参数（arguments）。
若 `dim_state` >0，则 `state_equation_function` 接收当前时间（time）、状态（state）和输入（intput）, 然后返回状态导数（state derivative），这个也是必需对。这种情况下， `output_equation_function` 接收当前时间（time）和状态（state）作为积分（integration）参数（arguments）。

若 `event_equation_function` 和 `update_equation_function` 都提供了, `event_equation_function` 在零交叉处（zero-crossing）的不连续性（discontinuities）就可以处理了。积分过程中 `event_equation_function` 和 `update_equation_function` 的参数规则（argument rules）与 `output_equation_function` 和 `state_equation_function` 各自一致。一般来说， `update_equation_function` 基于不连续性的发生情况（occurance）来改变 `state_equation_function` , `output_equation_function` , 和 `event_equation_function` 计算的结果。若 `dim_state` >0, `update_equation_function` 必须在遇到不连续处后立即返回状态（state）。

基础系统类接收一个适当的输入参数（input argument） `dt` （译者注：可以理解为采样时间间隔）。 传入 `dt` >0 将确定计算输出（output）和状态（state）的采样率（sample rate）； `dt` =0 就表示创建一个连续时间系统（continuous-time system）。在混合时间（hybrid-time）模块图 `BlockDiagram` 中，系统会被自动分段积分（ntegrated piecewise）来提高精度 .

未来版本的 SimuPy 可能会支持将雅可比函数（jacobian functions）传入给 ODE 求解器（ode solvers，ODE=ordinary differential equation，常微分方程），这需要 `BlockDiagram` 中的所有系统都提供适当必需的雅可比函数（jacobian functions）。

对这些模块（module）的快速概览：

`block_diagram` (docstrings)

是 `BlockDiagram` 类的一个实现，用于模拟互联系统（interconnected systems）。

`systems` (docstrings) 针对纯粹基于数值的系统提供了若干基类。

`utils` (docstrings)

提供了一些工具函数（utility functions），比如操作（数值）系统以及模拟结果。

`systems.symbolic` (docstrings) 和 `discontinuities` (docstrings)

提供了用于定义系统的符号表达式的细节。

`array` (docstrings) 和 `matrices` (docstrings)

提供帮助函数和一些类，用于操作符号数组（symbolic arrays）、矩阵以及他们组成的系统。

`utils.symbolic` (docstrings)

提供了功能性的符号函数，比如操作符号系统等等。

第四章 块图模块 block_diagram module

原文链接

译者注：Block Diagram直接翻译就是模块图，Module就是Python里面的模块，所以这就成了模块图模块了，看着挺别扭的，所以以后这样的标题部分就不翻译了，直接用作术语了。

class `simupy.block_diagram.` `BlockDiagram` (**systems*)[source]

若干动态系统及其之间连接的模块图是可以用数值模拟的。

初始化一个 BlockDiagram，可以赋予一系列系统构成的可选的列表, 来启动这个图。

`add_system` (*system*)[source]

添加一个系统到模块图（BlockDiagram）。

参数： **system** (*动态系统 dynamical system*) – 要添加到模块图上的系统（system）。

`computation_step`(*t*, *state*, *output=None*, *selector=True*, *do_events=False*)[source]

可调用函数，用于计算系统输出和状态导数（state derivatives）

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

连接模块图中的各个系统。

参数：

* **from_system_output** (*dynamical system*) – 已经添加到模块图中的系统，其中的输出将被连接。注意，一个系统的输出可以被链接到多处输入。
* **to_system_input** (*dynamical system*) – 已经添加到模块图中的系统，所接受的输入将被连接。注意，任何之前的输入连接（input connections）将被覆盖（over-written）。
* **outputs** (*list-like, optional*) – 连接输出的选择器索引（selector index）。如果没有指定或者长度为0，就会连接所有输出。
* **inputs** (*list-like, optional*) – 连接输入的选择器索引（selector index）。如果没有指定或者长度为0，就会连接所有输入。

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

创建或者使用输入通道（input channels），将模块图用作一个子系统（subsystem）。

参数：

* **channels** (*list-like*) – 要连接的输入通道（input channels）的选择器索引。
* **to_system_input** (*dynamical system*) – 已经添加到模块图中的系统，其中输入将被连接。注意，任何之前的输入连接（input connections）将被覆盖（over-written）。
* **inputs** (*list-like, optional*) – 连接输入的选择器索引（selector index）。如果没有指定或者长度为0，就会连接所有输入。

`dim_output`

`dim_state`

`dt`

`event_equation_function_implementation`(*t*, *state*, *output=None*)[source]

`initial_condition`

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

`prepare_to_integrate`()[source]

`simulate`(*tspan*, *integrator_class=*, *integrator_options={'atol': 1e-12*, *'max_step': 0.0*, *'name': 'dopri5'*, *'nsteps': 500*, *'rtol': 1e-06}*, *event_finder=*, *event_find_options={'maxiter': 100*, *'rtol': 8.881784197001252e-16*, *'xtol': 2e-12}*)[source]

模拟（simulate）模块图

参数：

* **tspan** (*list-like or float*) –

> 指定积分时间步长范围（integration time-steps）的参数。
>
> 如果给了一个时间，就当作是终点时间。如果给了两个时间，就分别用作起始终结的两个时间点。两种情况下，都假设可变时间步长积分器(**variable** time-step integrator)的每个时间步长将会存储到结果中。
>
> 若给的时间列表中的时间个数超过了两个，那这个列表中的这些时间就是会存储积分轨迹（trajectories）的时间。

- **integrator_class** (*class, optional*) –

  积分器（integrator）的类，默认使用的是 `scipy.integrate.ode` 。必须遵循 `scipy.integrate.ode` API 提供下列参数：

  - `__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` 性质（properties）

- **integrator_options** (*dict, optional*) – 传递给积分器 `integrator_class.set_integrator` 的关键词参数字典。

- **event_finder** (*callable, optional*) – 区间求根函数（interval root-finder function）。默认使用的是 `scipy.optimize.brentq`，必须使用有效的位置参数（equivalent positional arguments），`f`，`a`，和 `b`，然后返回 `x0`，其中 `a <= x0 <= b` 而 `f(x0)` 为零值。

- **event_find_options** (*dict, optional*) – 传递给 `event_finder` 的关键词参数字典。必须提供

一个名为 `'xtol'` 的键, 还需要满足零值范围处于区间 `x0 +/- xtol/2` , 由求根函数 `brentq` 提供。

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

`systems_event_equation_functions`(*t*, *state*, *output*)[source]

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

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

一个简单的类，用于收集模拟结果轨迹（simulation result trajectories）。

`t`

类型：时间数组

`x`

类型：状态数组

`y`

类型：输出数组

`e`

类型：事件数组

`allocate_space`(*t*)[source]

`last_result`(*n=1*, *copy=False*)[source]

`max_allocation` = *128*

`new_result`(*t*, *x*, *y*, *e=None*)[source]

第五章 系统模块 systems module

原文链接

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*)[source]

基类（Base）：`object`

一个动态系统（dynamical system），其中模型系统（models systems）形式为：

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

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

上面的形式也可以用来表示离散时间系统（discrete-time systems），这时候上面的 xdot(t) 就表示了 x[k+1]。

这个系统也可以对不连续系统（discontinuous systems）进行建模（model）。不连续性（discontinuities）必须位于事件方程函数 `event_equation_function` 的零交叉处（zero-crossings），接收的参数（arguments）和输出方程函数 `output_equation_function` 相同，依赖 `dim_state`。在零交叉处，以同样的参数（arguments）调用更新方程函数 `update_equation_function`。如果 `dim_state` > 0, `update_equation_function` 的返回值就用做在不连续发生后系统状态（state）。

参数：

* **state_equation_function** (*callable, optional*) – 系统状态的导数（derivative）或者更新方程（update equation），如果 `dim_state` 为零值，就不需要这个参数了。
* **output_equation_function** (*callable, optional*) – 系统的输出方程。一个系统必须有一个输出方程函数 `output_equation_function`。若没设置，就使用完全状态输出（full state output）。
* **event_equation_function** (*callable, optional*) – 这个函数的输出决定了不连续何时发生。
* **update_equation_function** (*callable, optional*) – 当不连续发生的时候这个函数就被调用了。
* **dim_state** (*int, optional*) – 系统状态的维度，可选设置，默认是0 。
* **dim_input** (*int, optional*) – 系统输入的维度，可选设置，默认是0 。
* **dim_output** (*int, optional*) – 系统输出的维度，可选设置，默认设置为 dim_state.
* **dt** (*float, optional*) – 系统采样率（sample rate），可选设置，默认是0，意思就是连续时间系统（continuous time system）。
* **initial_condition** (*array_like of numerical values, optional*) – 数组（Array）或者矩阵（Matrix），用作系统的初始条件（initial condition）。默认是设置成和系统状态一样的维度，然后所有值默认设置为0。

`dt`

`initial_condition`

`prepare_to_integrate` ()[source]

`validate` ()[source]

class `simupy.systems.` `LTISystem` (**args*, *initial_condition=None*, *dt=0*)[source]

基类： `simupy.systems.DynamicalSystem`

一个线性时不变系统（linear, time-invariant system，缩写为 LTI）。

创建这样一个 LTI 系统使用下面的输入格式：

1. 状态矩阵（state matrix） A, 输入矩阵（input matrix） B, 输出矩阵（output matrix） C, 系统状态为：

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

2. 状态矩阵（state matrix） A, 输入矩阵（input matrix） B, 有状态（state）的系统，假设完全状态输出（full state output）：

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

3. 无状态系统的增益矩阵（gain matrix）K:4.
   ```
   y = Kx
   ```

上述矩阵都需要是形态一致的数值数组组成的。本类内提供了一些矩阵别名，比如 `A`，`B`，`C` 和 `F`，`G`，`H` 用于有状态系统，以及 `K` 作为增益矩阵（gain matrix）的别名（alias）。`data` 将这些矩阵打包成一个元组（tuple）。

`A`

`B`

`C`

`F`

`G`

`H`

`K`

`data`

`validate`()[source]

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*)[source]

基类： `simupy.systems.DynamicalSystem`

提供了一个用于不连续系统（discontinuous systems）的有用模式（pattern），其中状态和输出方程的变更依赖于一个值，这个值是关于状态和/或输入的函数，事件变量方程函数（`event_variable_equation_function`）。主要用处在于利用边界有根的伯恩斯坦基多项式（Bernstein basis polynomial with roots at the boundaries）创建事件方程函数`event_equation_function`。 这个类还提供了基础洛基，用来基于`event_variable_equation_function` 值输出正确的状态和输出方程。

参数：

- **state_equations_functions** (*array_like of callables, optional*) – 系统状态的导数或者更新方程。若 `dim_state` 为零值，则不需要设置这个参数来。 事件状态（event-state）的数组索引（array indexes） 需要比事件边界（event bounds）多一个。这个索引还需要和边界匹配（也就是说，当事件变量（event variable）低于第一个事件边界（event bounds）值，就使用来第一个函数）。如果只提供了一个可调用函数（callable），那它就会在每个条件下使用。
- **output_equations_functions** (*array_like of callables, optional*) – 系统的输出方程。一个系统必须有一个输出方程函数 `output_equation_function`。如果没有设置，就使用完全状态输出。事件状态（event-state）的数组索引（array indexes） 需要比事件边界（event bounds）多一个。这个索引还需要和边界匹配（也就是说，当事件变量（event variable）低于第一个事件边界（event bounds）值，就使用来第一个函数）。如果只提供了一个可调用函数（callable），那它就会在每个条件下使用。
- **event_variable_equation_function** (*callable*) – 当函数的输出与事件边界`event_bounds`的值交叉，就发生了以此不连续事件（discontuity event）。
- **event_bounds** (*array_like of floats*) – 根据`event_variable_equation_function` 的输出定义触发不连续事件的边界点。
- **state_update_equation_function** (*callable, optional*) – 当一个事件发生，状态更新方程函

数被调用，来决定状态更新。如果没有设置，就使用完全状态输出，所以这时候当事件变量函数遇到零交叉处的时候也不改变状态了。

- **dim_state** (*int, optional*) – 系统状态维度，可选参数，默认为0。
- **dim_input** (*int, optional*) – 系统输入维度，可选参数，默认为0。
- **dim_output** (*int, optional*) – 系统输出维度，可选参数，默认设置为 dim_state。

`event_bounds`

`event_equation_function`(**args*)[source]

`output_equation_function`(**args*)[source]

`prepare_to_integrate`()[source]

`state_equation_function`(**args*)[source]

`update_equation_function`(**args*)[source]

`validate`()[source]

`simupy.systems.` `SystemFromCallable`(*incallable*, *dim_input*, *dim_output*, *dt=0*)[source]

Construct a memoryless system from a callable.

参数：

* **incallable** (*callable*) – 用作输出方程函数 output_equation_function，(t, u) if dim_input > 0 或者 (t) if dim_input = 0.
* **dim_input** (*int*) – 输入维度.
* **dim_output** (*int*) –输出维度.

`simupy.systems.` `full_state_output`(**args*)[source]

针对有状态系统的一个随时可用的输出方程函数`output_equation_function`，直接提供了完全状态输出。

第六章 工具模块 utils module

原文链接

`simupy.utils.` `array_callable_from_vector_trajectory`(*tt*, *x*, *unraveled*, *raveled*)[source]

转换轨迹（trajectory）为返回一个二维数组的可调用的插值。解开的、散开的对映射了数组的填充方式（The unraveled, raveled pair map how the array is filled in.）。具体参考样例 riccati_system。

参数：

* **tt** (*1D array_like*) – 一个轨迹的 m 时间索引数组
* **xx** (*2D array_like*) – 时间索引中 m x n 向量样本的索引。第一个维度的索引是时间（time），第二个维度的索引是向量元素（vector components）。
* **unraveled** (*1D array_like*) – 与上面的 xx 匹配的 n 个独立键值（n unique keys）组成的数组。
* **raveled** (*2D array_like*) – 解开数据（unraveled）键值（keys）的元素（elements）组成的数组（array）。从解开到未解开之间的映射用来确定输出数组的填充方式。

返回值：

matrix_callable – 用指定形状插值轨迹的可调用函数（callable）。

返回类型：
callable 可调用函数

`simupy.utils.` `callable_from_trajectory` (*t*, *curves*)[source]

使用 scipy.interpolate.make_interp_spline 来对一系列的曲线（curves）构建立方B样条插值（cubic b-spline interpolating）。

参数：

* **t** (*1D array_like*) – 一个轨迹的 m 时间索引数组
* **curves** (*2D array_like*) – 时间索引中 m x n 向量样本的索引。第一个维度的索引是时间（time），第二个维度的索引是向量元素（vector components）。

返回值：

interpolated_callable – 用于对给定的曲线（curve）、策略（trajectories）进行插值的可调用函数（callable）

返回类型：
callable 可调用函数

`simupy.utils.` `discrete_callable_from_trajectory` (*t*, *curves*)[source]

构建一个可调用函数，基于前一个时间步长（time-step）的返回值来对一个离散时间曲线

（discrete-time curve）进行插值。

参数：

* **t** (*1D array_like*) – 一个轨迹的 m 时间索引数组
* **curves** (*2D array_like*) – 时间索引中 m x n 向量样本的索引。第一个维度的索引是时间（time），第二个维度的索引是向量元素（vector components）。

返回值：

nearest_neighbor_callable – 一个可调用函数，用于对给定的离散时间曲线、策略（discrete-time curve/trajectories）进行插值

返回类型：
callable 可调用函数

第七章 符号系统模块 symbolic systems module

2020年6月13日21:00

原文链接

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={}*)[source]

基类： `simupy.systems.DynamicalSystem`

动态系统构造器（dynamicalSystem constructor），用来从符号表达式（symbolic expressions）创建系统。

参数：

* **state_equation** (*array_like of sympy Expressions, optional*) – 状态导数的向量值表达式（vector valued expression）。
* **state** (*array_like of sympy symbols, optional*) – 表征状态（state）成分（components）的符号（symbols）组成的向量，按所需顺序，匹配状态方程（state_equation）。
* **input** (*array_like of sympy symbols, optional*) – 表征输入（input）成分的符号组成的向量，按

照所需顺序。状态方程（state_equation）可能依赖于系统输入（system input）。如果系统没有状态，则输出方程（output_equation）依赖于系统输入（system input）。
* **output_equation** (*array_like of sympy Expressions*) – 系统输出的向量值表达式。
* **constants_values** (*dict*) – 常量替换的字典（dictionary of constants substitutions）。
* **dt** (*float*) – 系统采样率，设置为0意为连续系统。
* **initial_condition** (*array_like of numerical values, optional*) – 用作系统初始条件的数组或者矩阵。默认是和状态相同维度，全设为零值。
* **code_generator** (*callable, optional*) – 用作代码生成器的函数。
* **code_generator_args** (*dict, optional*) – 传递给代码生成器的关键词参数的字典。

默认情况下，代码生成器使用的是对 `sympy.lambdify` 的一个封装。你可以修改所用的代码生成器，只需传递系统初始化参数 `code_generator` 和传递给代码生成器的附加的关键词参数组成的字典 `code_generator_args`。未来的系统中只要调整模块值就可以修改默认的代码生成器。

`copy`()[source]

`equilibrium_points`(*input_=None*)[source]

`input`

`output_equation`

`prepare_to_integrate`()[source]

`state`

`state_equation`

`update_input_jacobian_function`()[source]

`update_output_equation_function`()[source]

`update_state_equation_function`()[source]

`update_state_jacobian_function`()[source]

class `simupy.systems.symbolic.MemorylessSystem`(*input_=None*, *output_equation=None*, ***kwargs*)[source]

基类：`simupy.systems.symbolic.DynamicalSystem`

无状态系统。

没有输入，就可以表征一个信号（只是时间的函数）。例如，随机信号（stochastic signal）可以插值点，然后使用 prepare_to_integrate 来重新散播（re-seed）数据。

动态系统构造器

参数：

- **input** (*array_like of sympy symbols*) – 表征输入成分的符号向量，按照所需顺序。输出可能要依赖与系统输入。
- **output_equation** (*array_like of sympy Expressions*) – 系统输出的向量值表达式。

`state`

第八章 discontinuities module

2020年6月13日21:28

原文链接

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={}*)[source]

基类： `simupy.systems.symbolic.DynamicalSystem`

一个存在一处不连续点（discontinuity）的连续时间动态系统（continuous-time dynamical system）。除了动态系统（DynamicalSystem）需要的初始化参数之外，还需要提供下面的属性（attributes）：

`event_equation_function` - 每个积分时间步长（integration time-step）都会调用的一个函数，存储在模拟结果中。如果是有状态系统，就接收输入和状态为参数。若输出中出现零交叉点，就触发不连续性，表示是不连续处。

`event_equation_function` - 当不连续发生时调用的函数。一般来说是用来改变 `state_equation_function`，`output_equation_function`，以及 `event_equation_function` 计算的值，基于不连续性的具体情形。如果有状态，在不连续被发现后会立即返回状态。

动态系统构造器，用来从符号表达式处创造系统。

参数：

* **state_equation** (*array_like of sympy Expressions, optional*) – 状态导数的向量值表达式。
* **state** (*array_like of sympy symbols, optional*) – 表征状态成分的符号组成的向量，按照所需顺序，匹配状态方程（state_equation）。
* **input** (*array_like of sympy symbols, optional*) – 表征输入成分的符号组成的向量，按照所需顺序。状态方程可能依赖于系统输入。如果系统没有状态，则输出方程可能依赖于系统输入。
* **output_equation** (*array_like of sympy Expressions*) – 系统输出的向量值表达式。
* **constants_values** (*dict*) – 常量替换的字典（dictionary of constants substitutions）。
* **dt** (*float*) – 系统采样率，如果设置为0则表示是时间连续系统。
* **initial_condition** (*array_like of numerical values, optional*) – 用作系统初始条件的数组或者矩阵。默认与状态维度相同，全部默认为零值。
* **code_generator** (*callable, optional*) – 用作代码生成器的函数。
* **code_generator_args** (*dict, optional*) – 传递给代码生成器的关键词参数组成的字典。

默认情况下，代码生成器使用的是对 `sympy.lambdify` 的一个封装。你可以修改所用的代码生成器，只需传递系统初始化参数 `code_generator` 和传递给代码生成器的附加的关键词参数组成的字典 `code_generator_args`。未来的系统中只要调整模块值就可以修改默认的代码生成器。

`dt`

`event_equation_function`(**args*, ***kwargs*)[source]

`update_equation_function`(**args*, ***kwargs*)[source]

class `simupy.discontinuities.` `MemorylessDiscontinuousSystem`(*input_=None*, *output_equation=None*, ***kwargs*)[source]

基类: `simupy.discontinuities.DiscontinuousSystem`, `simupy.systems.symbolic.MemorylessSystem`

动态系统构造器

参数:
* **input** (*array_like of sympy symbols*) – 表征输入成分的符号组成的向量，按照所需顺序。输出可能依赖于系统输入。
* **output_equation** (*array_like of sympy Expressions*) – 系统输出的向量值表达式。

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

基类: `simupy.discontinuities.SwitchedSystem`, `simupy.discontinuities.MemorylessD`

`iscontinuousSystem`

一个无记忆间断系统（memoryless discontinuous system），方便构建可切换的输出（switched outputs）。

切换系统构造器（SwitchedSystem constructor），用来从符号表达式创建切换系统（switched systems）。所需要的参数除了 `systems.symbolic.DynamicalSystems` 构造器所用的之外，还需要额外添加下面的参数.

参数：

- **event_variable_equation** (*sympy Expression*) – 表示事件方程函数（event_equation_function）的表达式。
- **event_bounds_expressions** (*list-like of sympy Expressions or floats*) – 有序列表状值（ordered list-like values），定义了事件边界，相对于事件变量表达式（event_variable_equation)。
- **state_equations** (*array_like of sympy Expressions, optional*) – 系统的状态方程。第一个维度索引的是事件状态（event-state），需要比事件边界（event bounds）的数目多一。这个索引还需要和边界匹配（也就是说，当事件变量（event variable）低于第一个事件边界（event bounds）值，就使用来第一个表达式）。第二个维度是系统的 dim_state。如果只有一维，那这个方程就会在每个条件下使用。
- **output_equations** (*array_like of sympy Expressions, optional*) – 系统的输出方程。第一个维度索引的是事件状态（event-state），需要比事件边界（event bounds）的数目多一。这个索引还需要和边界匹配（也就是说，当事件变量（event variable）低于第一个事件边界（event bounds）值，就使用来第一个表达式）。第二个维度是系统的 dim_output。如果只有一维，那这个方程就会在每个条件下使用。
- **state_update_equation** (*sympy Expression*) – 表示状态更新方程函数（state_update_equation_function）的表达式

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

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

切换系统构造器（SwitchedSystem constructor），用来从符号表达式创建切换系统（switched systems）。所需要的参数除了 `systems.symbolic.DynamicalSystems` 构造器所用的之外，还需要额外添加下面的参数.

参数：

- **event_variable_equation** (*sympy Expression*) – 表示事件方程函数（event_equation_function）的表达式
- **event_bounds_expressions** (*list-like of sympy Expressions or floats*) – 有序列表状值（ordered list-like values），定义了事件边界，相对于事件变量表达式（event_variable_equation)。
- **state_equations** (*array_like of sympy Expressions, optional*) – 系统的状态方程。第一个维度索引的是事件状态（event-state），需要比事件边界（event bounds）的数目多一。这个索引还需要和边界匹配（也就是说，当事件变量（event variable）低于第一个事件边界（event bounds）值，就使用来第一个表达式）。第二个维度是系统的 dim_state。如果只有一维，那这个方程就会在每个条件下使用。
- **output_equations** (*array_like of sympy Expressions, optional*) – 系统的输出方程。第一个维度索引的是事件状态（event-state），需要比事件边界（event bounds）的数目多一。这个索引还需要和边界匹配（也就是说，当事件变量（event variable）低于第一个事件边界（event bounds）值，就使用来第一个表达式）。第二个维度是系统的 dim_output。如果只有一维，那这个方程就会在每个条件下使用。

- **state_update_equation** (*sympy Expression*) – 表示状态更新方程函数（state_update_equation_function）的表达式。

`event_bounds_expressions`

`event_variable_equation`

`output_equations`

`prepare_to_integrate` ()[source]

`state_equations`

`state_update_equation`

`validate` (*from_self=False*)[source]

第九章 数组模块 array module

2020年6月13日22:20

原文链接

class `simupy.array.` `SymAxisConcatenatorMixin` [source]

基类： `object`

一个 mix-in 类，转化 numpy 的 AxisConcatenator 成使用 sympy 的 N 维数组（N-D arrays）。

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

`makemat`
是 `sympy.matrices.immutable.ImmutableDenseMatrix` 的别名。

class `simupy.array.` `SymCClass` [source]
基类： `simupy.array.SymAxisConcatenatorMixin`, `numpy.lib.index_tricks.CClass`

class `simupy.array.` `SymRClass` [source]
基类： `simupy.array.SymAxisConcatenatorMixin`, `numpy.lib.index_tricks.RClass`

`simupy.array.` `empty_array` ()[source]
构造一个空数组，一般是用来做占位符（place-holder）。

第十章 矩阵模块 matrices module

2020年6月13日22:40

原文链接

`simupy.matrices.` `block_matrix` (*blocks*)[source]

通过适当地块连接构造一个库安装结构，然后用其元素创建一个矩阵。

参数： **blocks** (*two level deep iterable of sympy Matrix objects*) – 指定的用于创建矩阵的块。
返回： **matrix** – 一个矩阵，矩阵元素就是上面的块。
返回类型： sympy Matrix

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

创建一个符号元素的矩阵

参数：

- **name** (*string*) – 变量基类的名字；每个变量都是 name_ij, 两个索引分别中 n,m 范围内有效

`simupy.matrices.` `matrix_subs` (**subs*)[source]

从矩阵生成对象，使之能传递给 sp.subs ， 将源头矩阵（from_matrix）中的每个元素替换成目标矩阵（to_matrix）中对应的元素。

根据输入不同，使用这个函数有三种方法，如下所示：
1. 一个单独的矩阵减法（matrix-level subsitution） - from_matrix, to_matrix
2. 一个(from_matrix, to_matrix) 组成的列表或者元组- 元组
3. 一个字典，键值形式为{from_matrix: to_matrix} - 键值对

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

使用矩阵创建一个符号动态系统（symbolic DynamicalSystem）。具体样例参考 `riccati_system`.

参数：

* **mat_DE** (*sympy Matrix*) – 矩阵导数表达式（右手侧）he matrix derivative expression (right hand side)
* **mat_var** (*sympy Matrix*) – 矩阵状态
* **mat_input** (*list-like of input expressions, optional*) – 列表状的矩阵微分方程中的输入表达式
* **constants** (*dict, optional*) – 常量替换字典。

返回：

sys – 一个动态系统（DynamicalSystem），可以用于数值求解矩阵微分方程。

返回类型：
DynamicalSystem

第十一章 符号工具模块 symbolic utils module

2020年6月13日22:46

原文链接

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

参数输入，用于创建控制仿射系统（control-affine systems）。

参数：

- **system** (*DynamicalSystem*) – 输入参数的系统
- **input** (*array_like of symbols, optional*) – 对参数对输入。仅用于扩充输入成分对子集。
- **update_outputs** (*boolean*) – 如果为真，则系统提供完全状态输出，将有参数的输入也添加到输出中。

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

计算向量值函数（vector-valued function）对应基（basis）的符号梯度（symbolic gradient）。

参数：

* **f** (*1D array_like of sympy Expressions*) – 要计算符号梯度对向量值函数。
* **basis** (*1D array_like of sympy symbols*) – 用于计算符号梯度对对应基（basis）。
* **for_numerical** (*bool, optional*) – 占位符，用于计算梯度所需对选项。

返回：

grad – 符号梯度。

返回类型： 二维数组形态对 sympy 表达式。

`simupy.utils.symbolic.` `lambdify_with_vector_args`(*args*, *expr*, *modules=({'ImmutableMatrix': *, 'atan2': }, 'numpy', *{'Mod': *, 'atan2': })*)[source]

对 sympy 的 lambdify 进行的一个封装，其中 process_vector_args 用于生成可调用函数，接收参数作为向量或者独立成分（individual components）。

参数：

* **args** (*list-like of sympy symbols*) – 表达式调用的输入参数
* **expr** (*sympy expression*) – 要转化为可调用的数值计算函数（numeric evaluation）的表达式。
* **modules** (*list*) – 具体参考 sympy 的 lambdify 的文档，直接传递过去作为模块关键词（modules keyword）。

`simupy.utils.symbolic.` `process_vector_args`(*args*)[source]

一个帮助程序，用于处理向量参数，使得可调用函数接收向量活着独立成分。最根本上拆解参数（essentially unravels the arguments）。