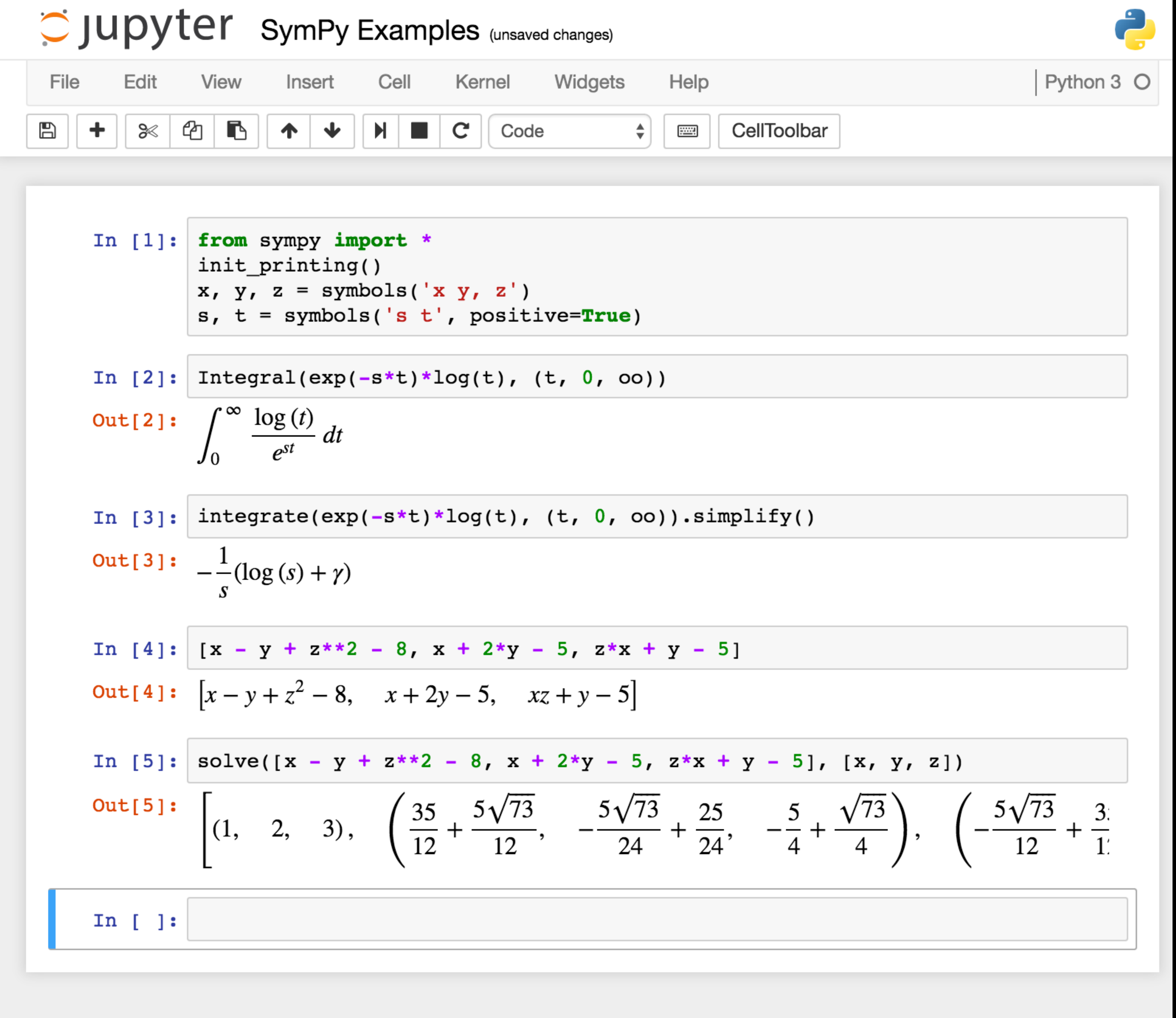
SymPy Code Generation

Aaron Meurer
Anthony Scopatz
ERGS, University of South Carolina

SciPy 2016, Austin, TX

SymPy

- Powerful computer algebra system (CAS) written in pure Python
- BSD licensed
- Usable as a library
- Just released version 1.0



In [1]:
```
from sympy import *
init_printing()
x, y, z = symbols('x y, z')
s, t = symbols('s t', positive=True)
```

In [2]:
```
Integral(exp(-s*t)*log(t), (t, 0, oo))
```

Out[2]:

$$\int_{0}^{\infty} \frac{\log{\left (t \right)}}{e^{st}}\, dt$$

In [3]:
```
integrate(exp(-s*t)*log(t), (t, 0, oo)).simplify()
```

Out[3]:

$$-\frac{1}{s}\left(\log{\left (s \right)} + \gamma\right)$$

In [4]:
```
[x - y + z**2 - 8, x + 2*y - 5, z*x + y - 5]
```

Out[4]:

$$\left[x - y + z^{2} - 8, \quad x + 2y - 5, \quad xz + y - 5\right]$$

In [5]:
```
solve([x - y + z**2 - 8, x + 2*y - 5, z*x + y - 5], [x, y, z])
```

Out[5]:

$$\left[\left (1, \quad 2, \quad 3\right), \quad \left (\frac{35}{12} + \frac{5\sqrt{73}}{12}, \quad - \frac{5\sqrt{73}}{24} + \frac{25}{24}, \quad - \frac{5}{4} + \frac{\sqrt{73}}{4}\right), \quad \left (- \frac{5\sqrt{73}}{12} + \frac{3}{1}\right.\right.$$

In [ ]:

Code Generation

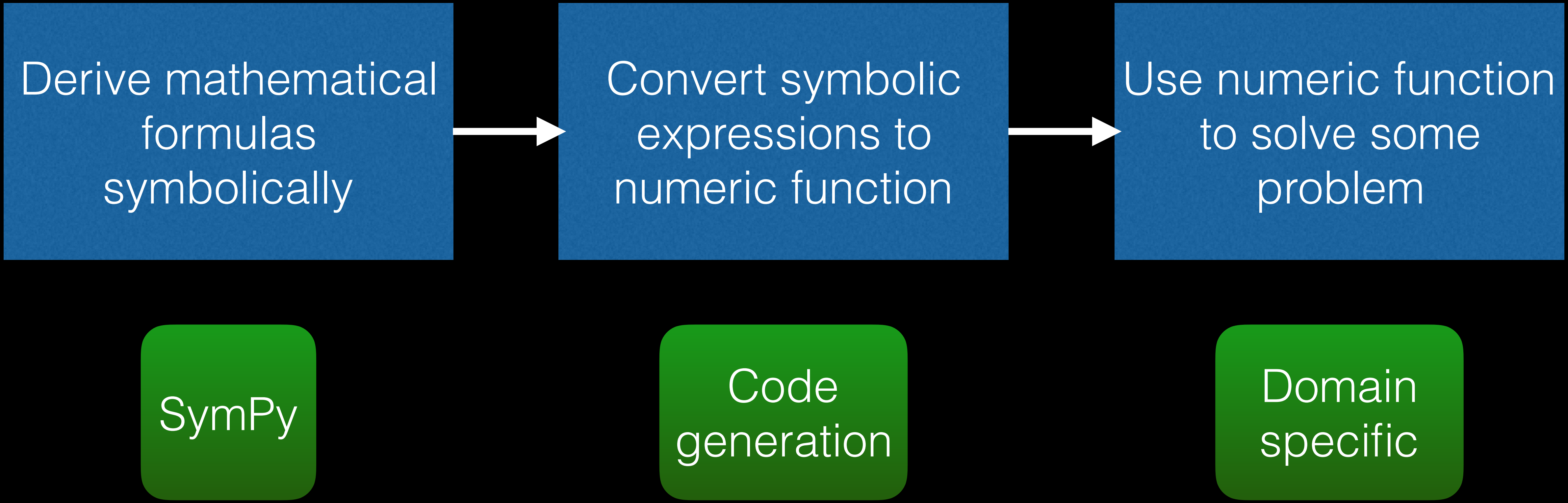
- Translate SymPy expressions to another language
- Example: translate $|\sin(\pi \cdot x)|$ to C

```
>>> ccode(abs(sin(pi*x)))
'fabs(sin(M_PI*x))'
```

Languages supported

- C
- Fortran
- MATLAB/Octave
- Python (NumPy/SciPy)
- Julia
- Mathematica
- Javascript
- LLVM
- Rust
- Theano
- Easy to extend to others

Code generation workflow



Derive mathematical formulas symbolically

Convert symbolic expressions to numeric function

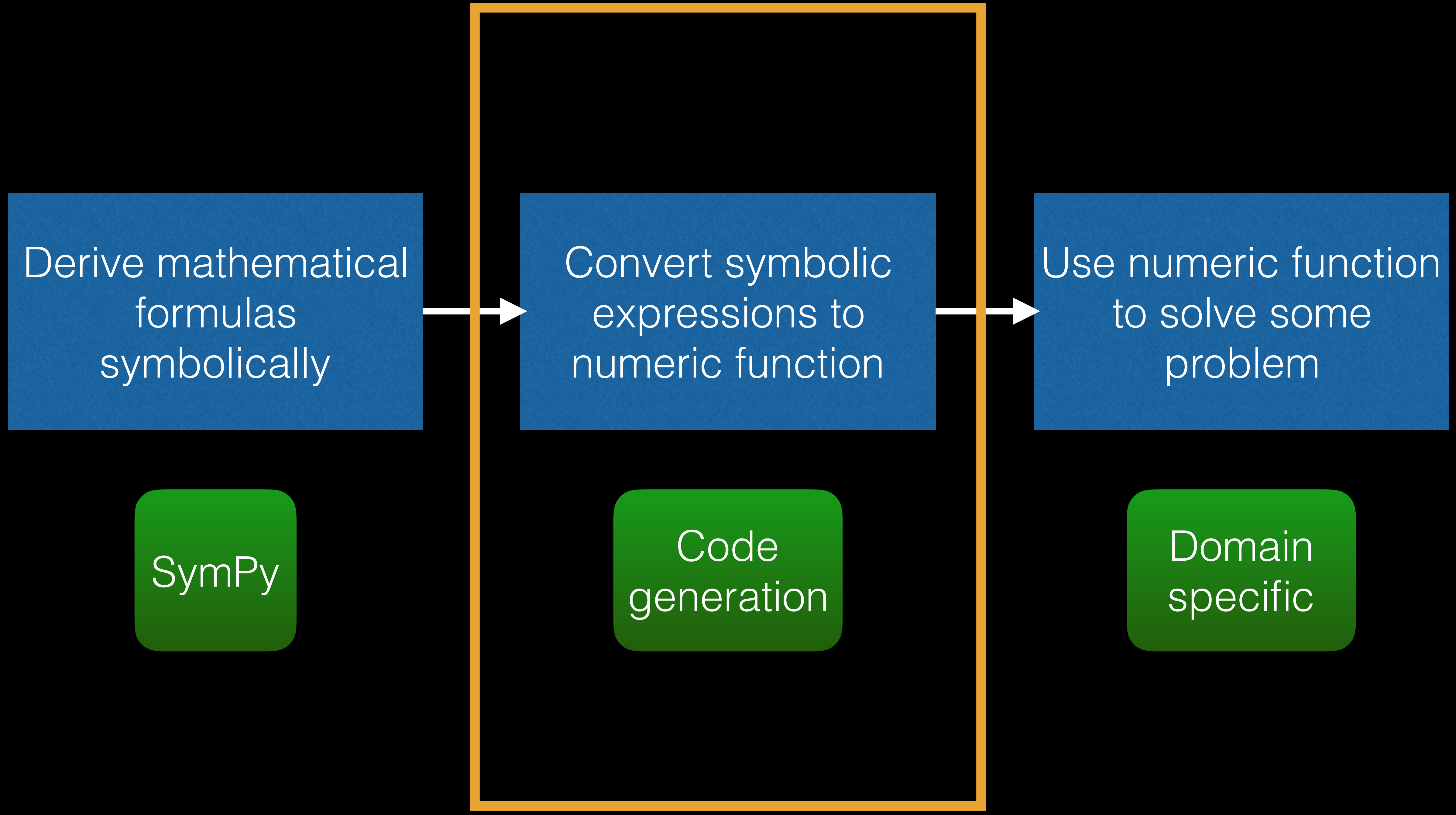
Use numeric function to solve some problem

SymPy

Code generation

Domain specific

Code generation workflow



Derive mathematical formulas symbolically

Convert symbolic expressions to numeric function

Use numeric function to solve some problem

SymPy

Code generation

Domain specific

Code generation levels of abstraction

Expression

```
'fabs(sin(M_PI*x))'
```

Code printers

Larger block of code

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

double f(double x) {

   double f_result;
   f_result = fabs(sin(M_PI*x));
   return f_result;

}
```

codegen

Python callable

```
f = ufuncify(x, abs(sin(pi*x)))
```

ufuncify
lambdify

Why do code generation?

Why do code generation?

Why do code generation?

1. SymPy can deal with mathematical expressions in a high-level way. For example, it can take symbolic derivatives.

Why do code generation?

1. SymPy can deal with mathematical expressions in a high-level way. For example, it can take symbolic derivatives.

2. Using code generation avoids mistakes that come from translating mathematics into low level code.

Why do code generation?

1. SymPy can deal with mathematical expressions in a high-level way. For example, it can take symbolic derivatives.

2. Using code generation avoids mistakes that come from translating mathematics into low level code.

3. It’s possible to deal with expressions that are otherwise too large to write by hand.

Why do code generation?

1. SymPy can deal with mathematical expressions in a high-level way. For example, it can take symbolic derivatives.

2. Using code generation avoids mistakes that come from translating mathematics into low level code.

3. It’s possible to deal with expressions that are otherwise too large to write by hand.

4. Some “mathematical” optimizations are possible, which a normal compiler would not be able to do.

Example: Iodine-131

Iodine-131

- Iodine-131 has a half-life of 8.0197 days
- Cells in the thyroid absorb Iodine
- Radioactive Iodine-131 destroys thyroid cells by short-range beta radiation

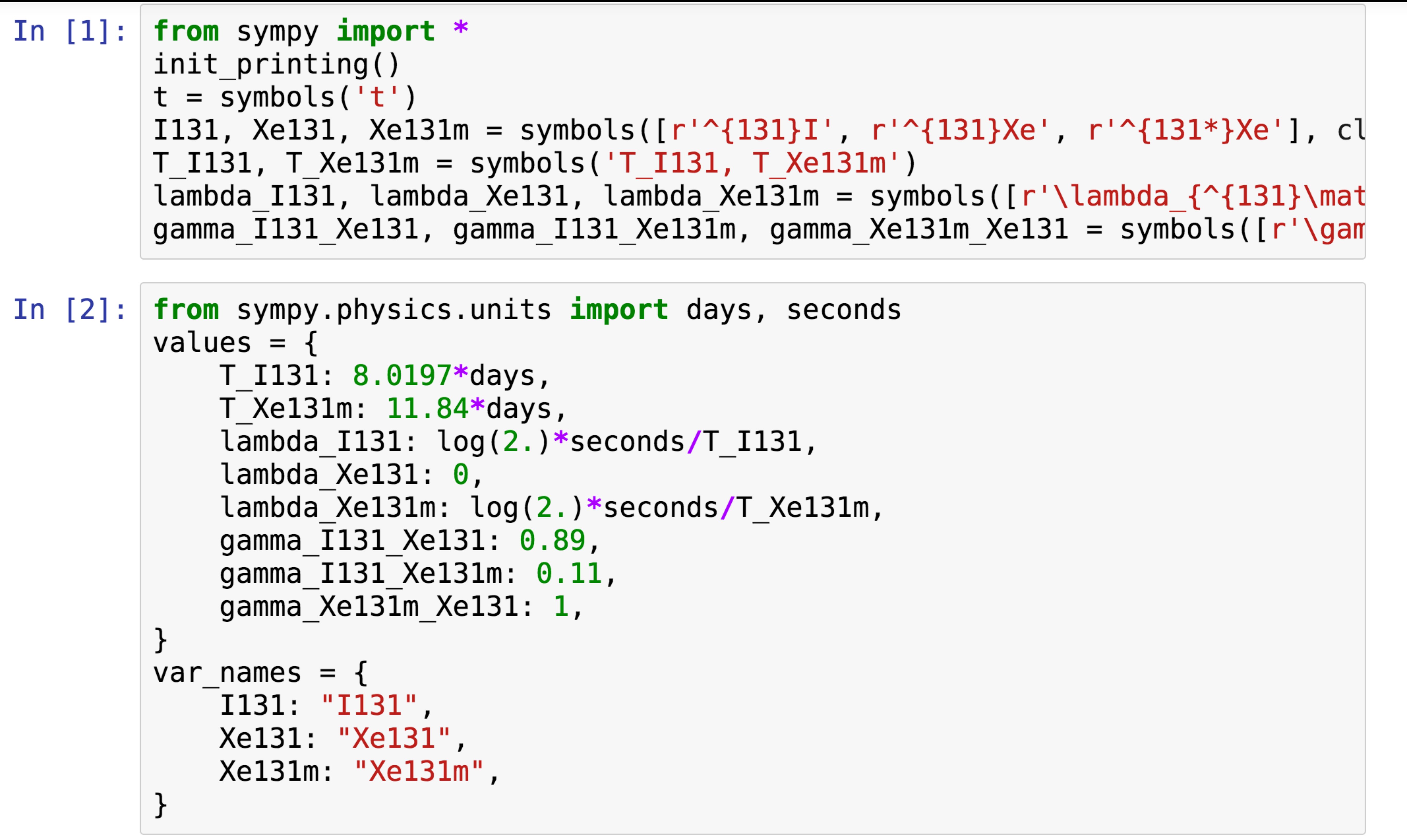
- I-131 decays with a half-life of 8.02 days

$$^{131}\text{I} \longrightarrow \beta^{-} + {}^{131*}\text{Xe}$$

$$^{131}\text{I} \longrightarrow \beta^{-} + {}^{131}\text{Xe}$$

$$^{131*}\text{Xe} \longrightarrow {}^{131}\text{Xe}$$

$$\frac{\partial x_i}{\partial t} = -\lambda_i x_i(t) + \sum_{i \neq j} \lambda_j \gamma_{j \rightarrow i} x_j(t)$$

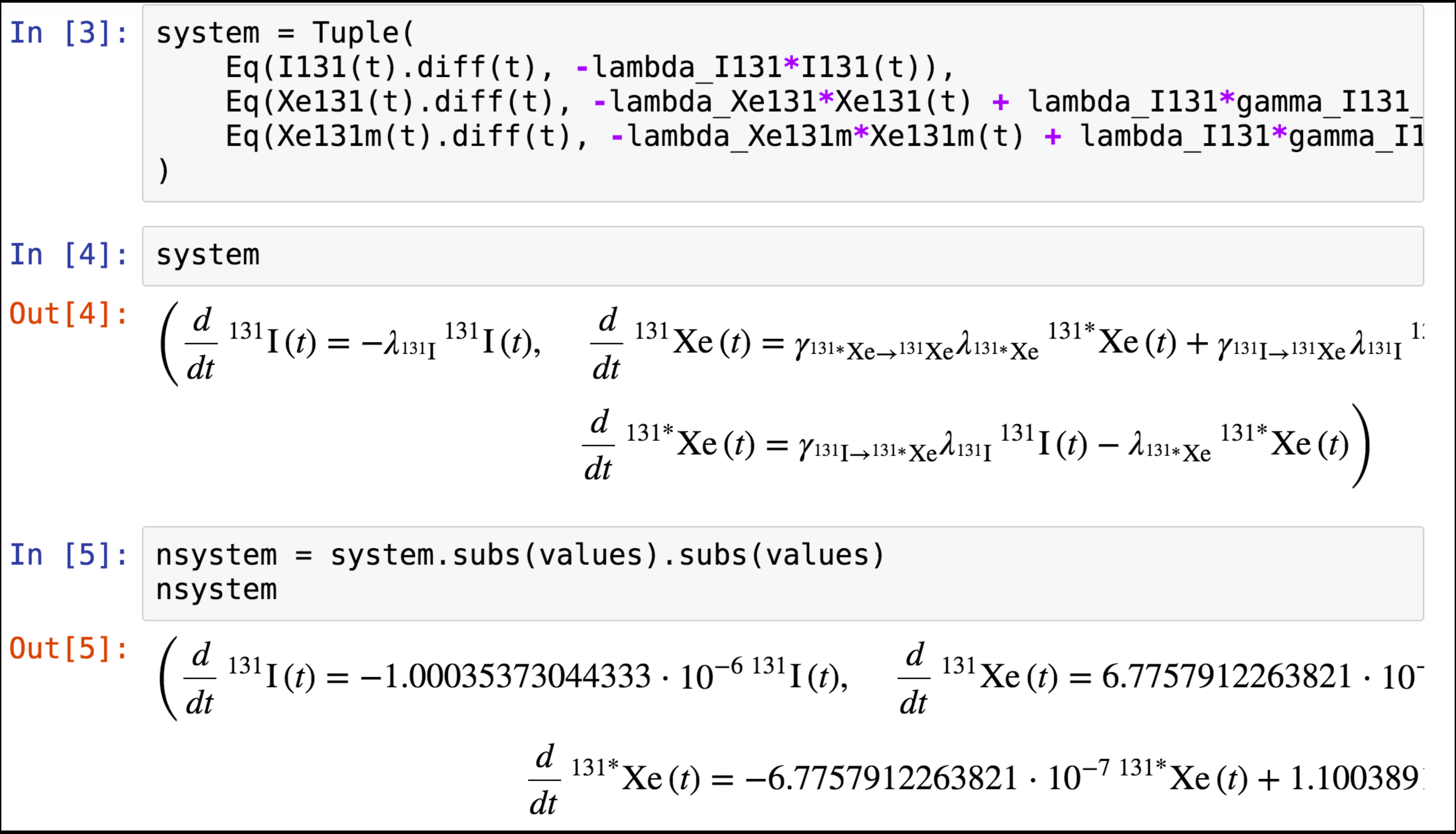


In [1]:

```
from sympy import *
init_printing()
t = symbols('t')
I131, Xe131, Xe131m = symbols([r'^{131}I', r'^{131}Xe', r'^{131*}Xe'], cl
T_I131, T_Xe131m = symbols('T_I131, T_Xe131m')
lambda_I131, lambda_Xe131, lambda_Xe131m = symbols([r'\lambda_{^{131}\mat
gamma_I131_Xe131, gamma_I131_Xe131m, gamma_Xe131m_Xe131 = symbols([r'\gam
```

In [2]:

```
from sympy.physics.units import days, seconds
values = {
    T_I131: 8.0197*days,
    T_Xe131m: 11.84*days,
    lambda_I131: log(2.)*seconds/T_I131,
    lambda_Xe131: 0,
    lambda_Xe131m: log(2.)*seconds/T_Xe131m,
    gamma_I131_Xe131: 0.89,
    gamma_I131_Xe131m: 0.11,
    gamma_Xe131m_Xe131: 1,
}
var_names = {
    I131: "I131",
    Xe131: "Xe131",
    Xe131m: "Xe131m",
}
```



In [3]:

```
system = Tuple(
    Eq(I131(t).diff(t), -lambda_I131*I131(t)),
    Eq(Xe131(t).diff(t), -lambda_Xe131*Xe131(t) + lambda_I131*gamma_I131_
    Eq(Xe131m(t).diff(t), -lambda_Xe131m*Xe131m(t) + lambda_I131*gamma_I1
)
```

In [4]:

```
system
```

Out[4]:

$$\left(\frac{d}{dt} {}^{131}\mathrm{I}(t) = -\lambda_{^{131}\mathrm{I}}\, {}^{131}\mathrm{I}(t), \quad \frac{d}{dt} {}^{131}\mathrm{Xe}(t) = \gamma_{^{131*}\mathrm{Xe}\to{}^{131}\mathrm{Xe}}\lambda_{^{131*}\mathrm{Xe}}\, {}^{131*}\mathrm{Xe}(t) + \gamma_{^{131}\mathrm{I}\to{}^{131}\mathrm{Xe}}\lambda_{^{131}\mathrm{I}}\, {}^{1}\right.$$

$$\left. \frac{d}{dt} {}^{131*}\mathrm{Xe}(t) = \gamma_{^{131}\mathrm{I}\to{}^{131*}\mathrm{Xe}}\lambda_{^{131}\mathrm{I}}\, {}^{131}\mathrm{I}(t) - \lambda_{^{131*}\mathrm{Xe}}\, {}^{131*}\mathrm{Xe}(t) \right)$$

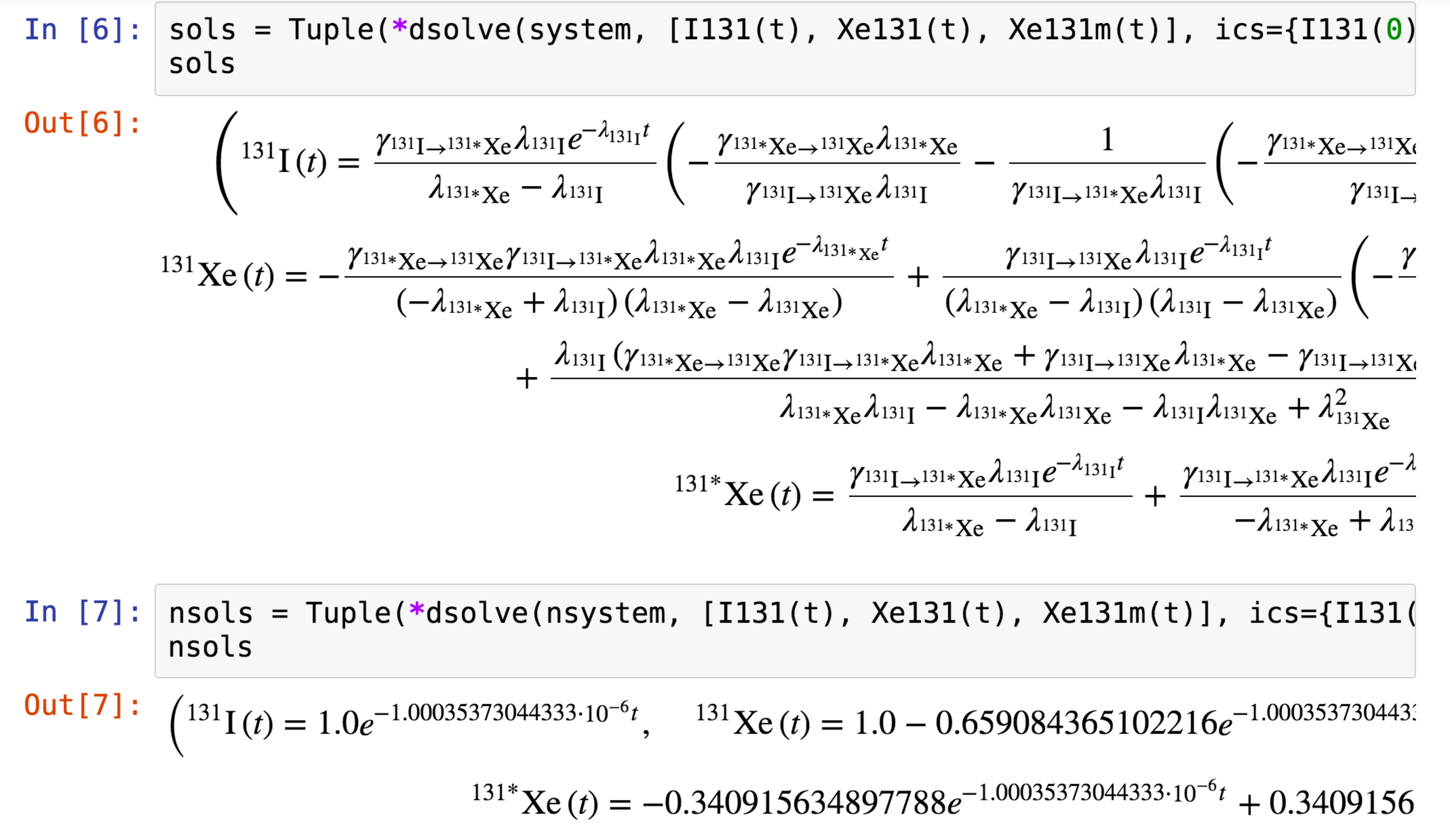
In [5]:

```
nsystem = system.subs(values).subs(values)
nsystem
```

Out[5]:

$$\left(\frac{d}{dt} {}^{131}\mathrm{I}(t) = -1.00035373044333 \cdot 10^{-6}\, {}^{131}\mathrm{I}(t), \quad \frac{d}{dt} {}^{131}\mathrm{Xe}(t) = 6.7757912263821 \cdot 10^{-}\right.$$

$$\frac{d}{dt} {}^{131*}\mathrm{Xe}(t) = -6.7757912263821 \cdot 10^{-7}\, {}^{131*}\mathrm{Xe}(t) + 1.100389$$



```
In [6]: sols = Tuple(*dsolve(system, [I131(t), Xe131(t), Xe131m(t)], ics={I131(0)
        sols
```

Out[6]:

$$({}^{131}\mathrm{I}(t) = \frac{\gamma_{131\mathrm{I}\to 131*\mathrm{Xe}}\lambda_{131\mathrm{I}}e^{-\lambda_{131\mathrm{I}}t}}{\lambda_{131*\mathrm{Xe}}-\lambda_{131\mathrm{I}}} (-\frac{\gamma_{131*\mathrm{Xe}\to 131\mathrm{Xe}}\lambda_{131*\mathrm{Xe}}}{\gamma_{131\mathrm{I}\to 131\mathrm{Xe}}\lambda_{131\mathrm{I}}} - \frac{1}{\gamma_{131\mathrm{I}\to 131*\mathrm{Xe}}\lambda_{131\mathrm{I}}} (-\frac{\gamma_{131*\mathrm{Xe}\to 131\mathrm{X}}}{\gamma_{131\mathrm{I}\to}}$$

$${}^{131}\mathrm{Xe}(t) = -\frac{\gamma_{131*\mathrm{Xe}\to 131\mathrm{Xe}}\gamma_{131\mathrm{I}\to 131*\mathrm{Xe}}\lambda_{131*\mathrm{Xe}}\lambda_{131\mathrm{I}}e^{-\lambda_{131*\mathrm{Xe}}t}}{(-\lambda_{131*\mathrm{Xe}}+\lambda_{131\mathrm{I}})(\lambda_{131*\mathrm{Xe}}-\lambda_{131\mathrm{Xe}})} + \frac{\gamma_{131\mathrm{I}\to 131\mathrm{Xe}}\lambda_{131\mathrm{I}}e^{-\lambda_{131\mathrm{I}}t}}{(\lambda_{131*\mathrm{Xe}}-\lambda_{131\mathrm{I}})(\lambda_{131\mathrm{I}}-\lambda_{131\mathrm{Xe}})} (-\gamma$$

$$+ \frac{\lambda_{131\mathrm{I}}(\gamma_{131*\mathrm{Xe}\to 131\mathrm{Xe}}\gamma_{131\mathrm{I}\to 131*\mathrm{Xe}}\lambda_{131*\mathrm{Xe}} + \gamma_{131\mathrm{I}\to 131\mathrm{Xe}}\lambda_{131*\mathrm{Xe}} - \gamma_{131\mathrm{I}\to 131\mathrm{X}}}{\lambda_{131*\mathrm{Xe}}\lambda_{131\mathrm{I}} - \lambda_{131*\mathrm{Xe}}\lambda_{131\mathrm{Xe}} - \lambda_{131\mathrm{I}}\lambda_{131\mathrm{Xe}} + \lambda_{131\mathrm{Xe}}^{2}}$$

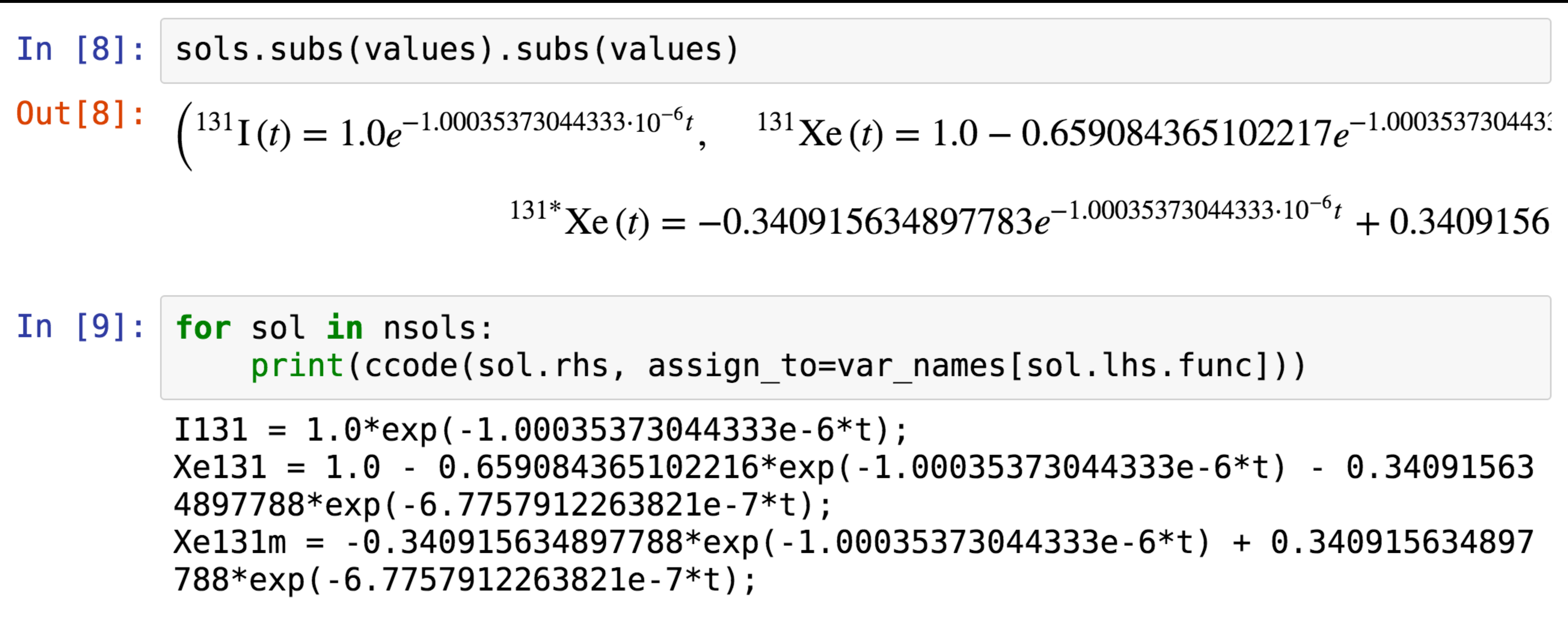
$${}^{131*}\mathrm{Xe}(t) = \frac{\gamma_{131\mathrm{I}\to 131*\mathrm{Xe}}\lambda_{131\mathrm{I}}e^{-\lambda_{131\mathrm{I}}t}}{\lambda_{131*\mathrm{Xe}}-\lambda_{131\mathrm{I}}} + \frac{\gamma_{131\mathrm{I}\to 131*\mathrm{Xe}}\lambda_{131\mathrm{I}}e^{-\lambda}}{-\lambda_{131*\mathrm{Xe}}+\lambda_{13}}$$

```
In [7]: nsols = Tuple(*dsolve(nsystem, [I131(t), Xe131(t), Xe131m(t)], ics={I131(
        nsols
```

Out[7]:

$$({}^{131}\mathrm{I}(t) = 1.0e^{-1.00035373044333\cdot 10^{-6}t}, \quad {}^{131}\mathrm{Xe}(t) = 1.0 - 0.659084365102216e^{-1.000353730443}$$

$${}^{131*}\mathrm{Xe}(t) = -0.340915634897788e^{-1.00035373044333\cdot 10^{-6}t} + 0.3409156$$



In [8]:

```
sols.subs(values).subs(values)
```

Out[8]:

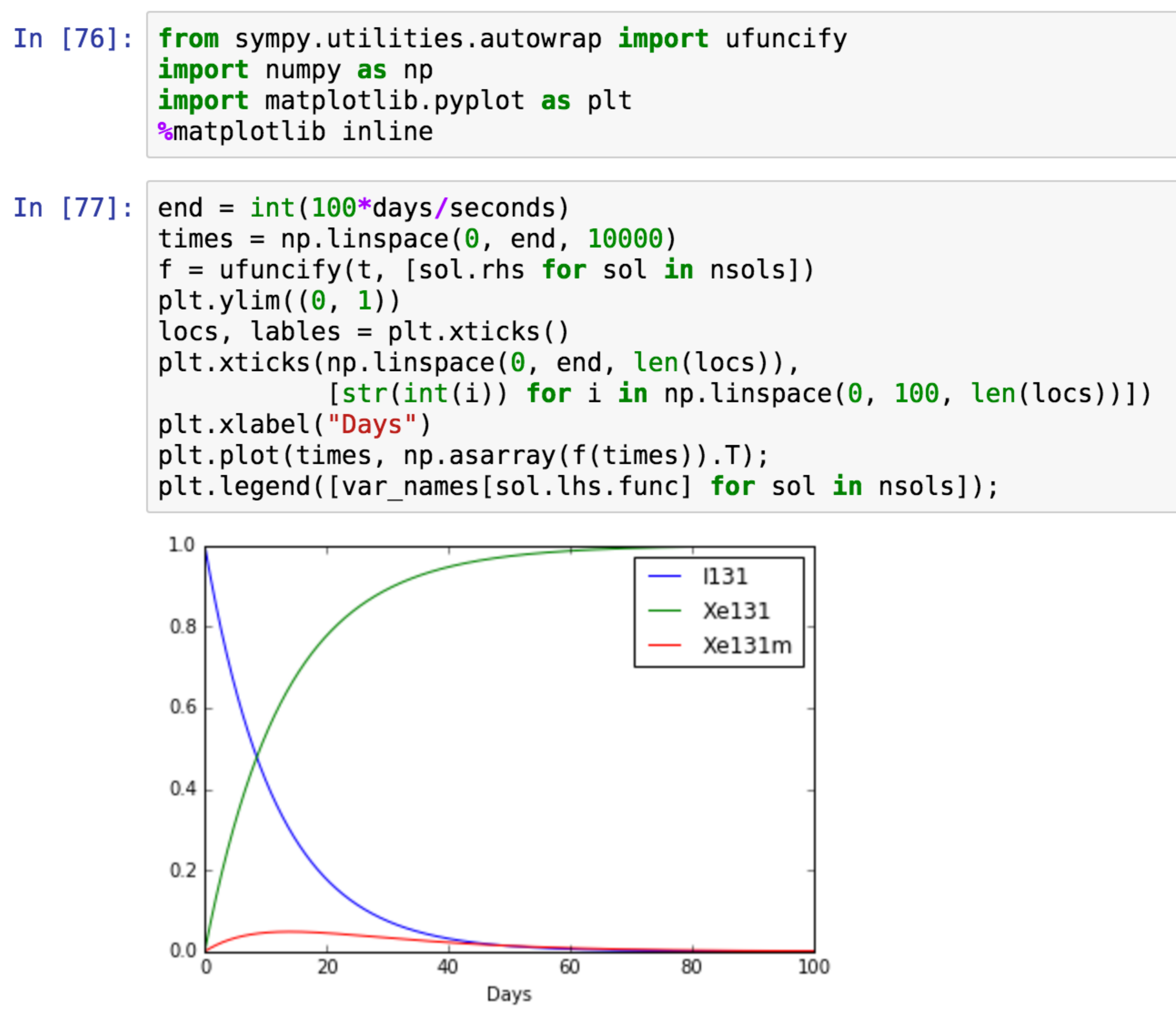
$$\left({}^{131}\mathrm{I}(t) = 1.0e^{-1.00035373044333\cdot 10^{-6}t}, \quad {}^{131}\mathrm{Xe}(t) = 1.0 - 0.659084365102217e^{-1.000353730443}\right.$$

$${}^{131*}\mathrm{Xe}(t) = -0.340915634897783e^{-1.00035373044333\cdot 10^{-6}t} + 0.3409156$$

In [9]:

```
for sol in nsols:
    print(ccode(sol.rhs, assign_to=var_names[sol.lhs.func]))
```

```
I131 = 1.0*exp(-1.00035373044333e-6*t);
Xe131 = 1.0 - 0.659084365102216*exp(-1.00035373044333e-6*t) - 0.34091563
4897788*exp(-6.7757912263821e-7*t);
Xe131m = -0.340915634897788*exp(-1.00035373044333e-6*t) + 0.340915634897
788*exp(-6.7757912263821e-7*t);
```



```
In [76]: from sympy.utilities.autowrap import ufuncify
         import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline
```

```
In [77]: end = int(100*days/seconds)
         times = np.linspace(0, end, 10000)
         f = ufuncify(t, [sol.rhs for sol in nsols])
         plt.ylim((0, 1))
         locs, lables = plt.xticks()
         plt.xticks(np.linspace(0, end, len(locs)),
                    [str(int(i)) for i in np.linspace(0, 100, len(locs))])
         plt.xlabel("Days")
         plt.plot(times, np.asarray(f(times)).T);
         plt.legend([var_names[sol.lhs.func] for sol in nsols]);
```

- This is a simple example, because the decay of ^{131}I only results in three species, ^{131}I, ^{131*}Xe, and ^{131}Xe

- A typical decay chain may result in hundreds of species

- With SymPy, we can avoid mistakes by representing the decay equations in a high level way, and deriving the low level representations

Example: n-link pendulum on cart (PyDy)

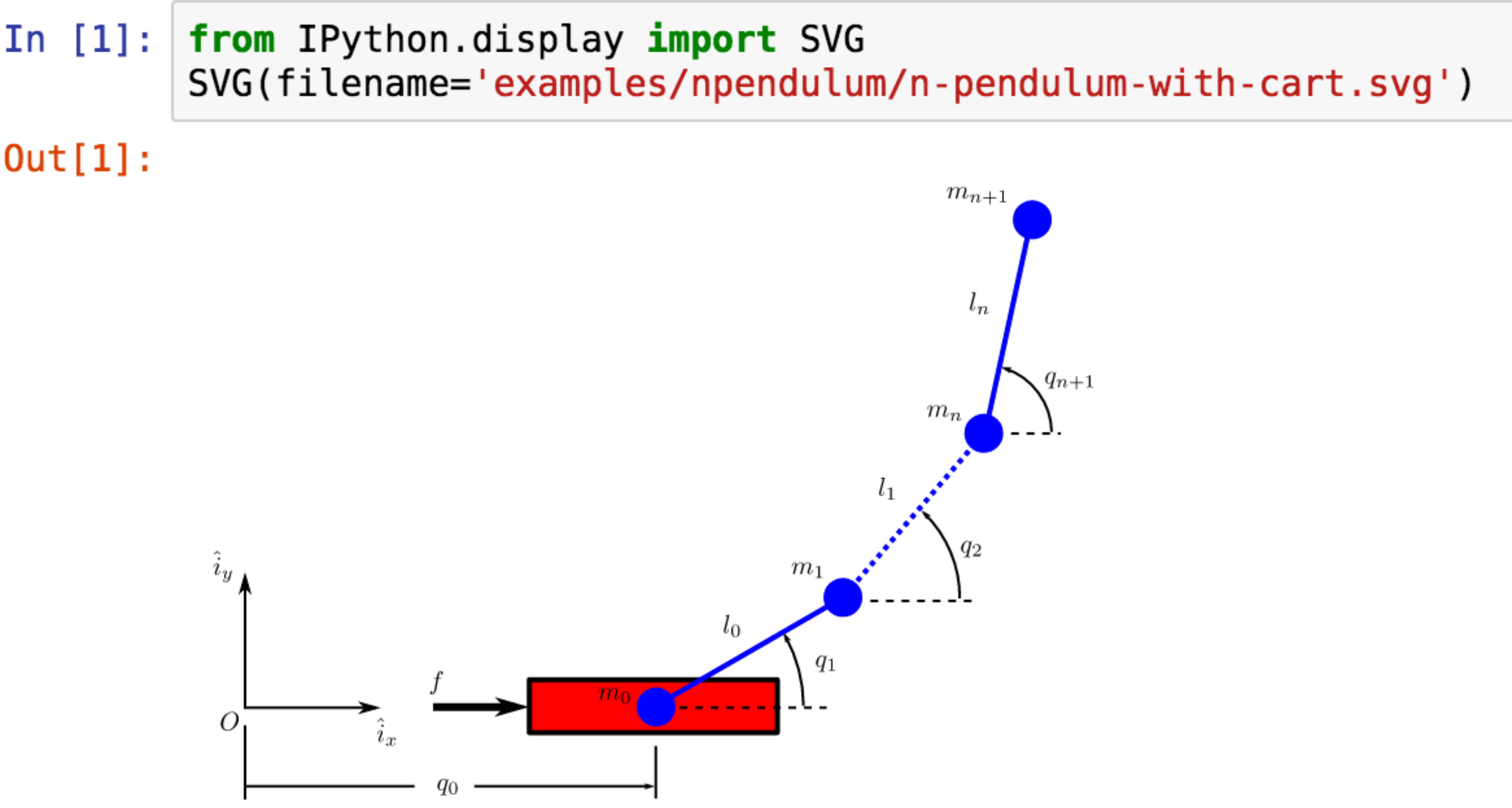
Thanks to Jason Moore

PyDy

- Multibody dynamics
- SymPy is used to derive the equations of motion for a mechanical system
- The resulting equations are a large system of nonlinear ODEs which need to be integrated (F=Ma)
- Code generation allows creating fast callbacks which can be integrated over many time steps

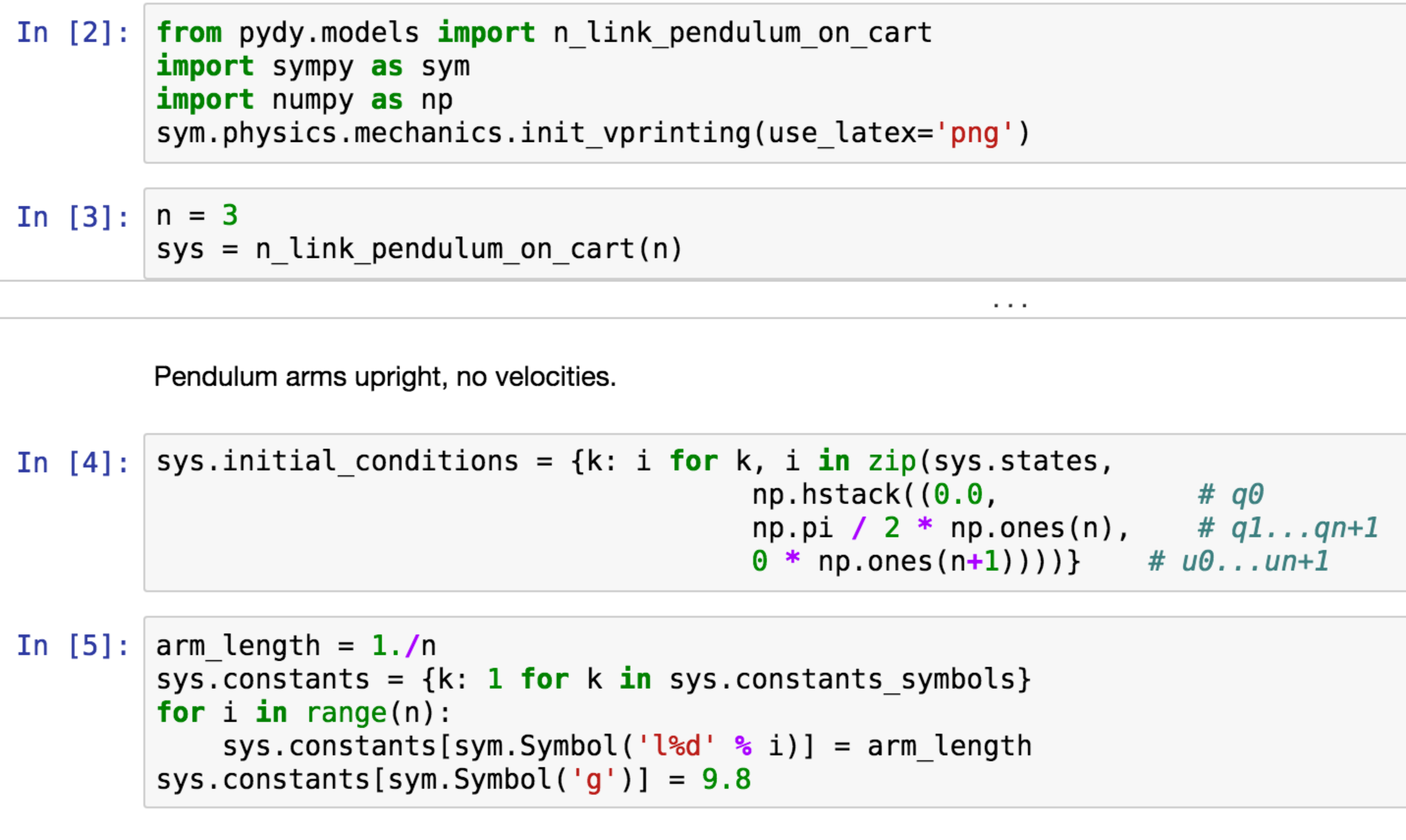
- Needs to be fast because:
  - Realtime simulation
  - Optimal control
  - Stiff systems require more time steps

n-link pendulum on a cart



```
In [1]: from IPython.display import SVG
        SVG(filename='examples/npendulum/n-pendulum-with-cart.svg')

Out[1]:
```



```
In [2]: from pydy.models import n_link_pendulum_on_cart
        import sympy as sym
        import numpy as np
        sym.physics.mechanics.init_vprinting(use_latex='png')
```

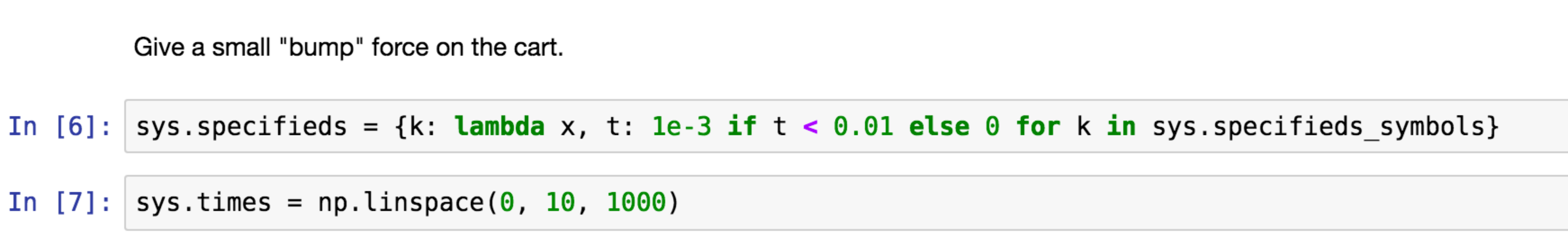
```
In [3]: n = 3
        sys = n_link_pendulum_on_cart(n)
```

...

Pendulum arms upright, no velocities.

```
In [4]: sys.initial_conditions = {k: i for k, i in zip(sys.states,
                                            np.hstack((0.0,            # q0
                                            np.pi / 2 * np.ones(n),    # q1...qn+1
                                            0 * np.ones(n+1))))}    # u0...un+1
```

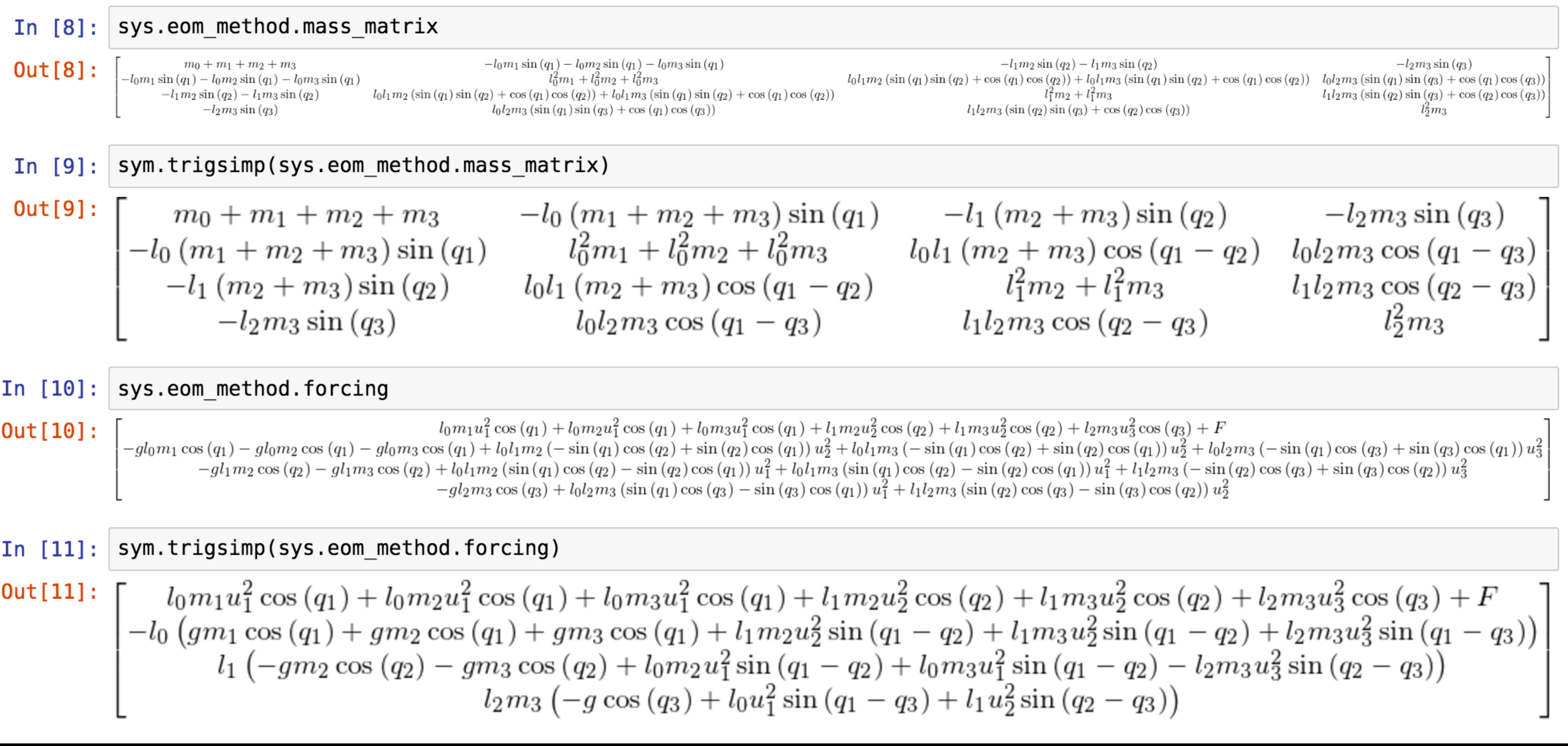
```
In [5]: arm_length = 1./n
        sys.constants = {k: 1 for k in sys.constants_symbols}
        for i in range(n):
            sys.constants[sym.Symbol('l%d' % i)] = arm_length
        sys.constants[sym.Symbol('g')] = 9.8
```



Give a small "bump" force on the cart.

```
In [6]: sys.specifieds = {k: lambda x, t: 1e-3 if t < 0.01 else 0 for k in sys.specifieds_symbols}
```

```
In [7]: sys.times = np.linspace(0, 10, 1000)
```



In [8]:
```
sys.eom_method.mass_matrix
```

Out[8]:

$$\left[\begin{matrix}m_{0} + m_{1} + m_{2} + m_{3} & - l_{0} m_{1} \sin{\left (q_{1} \right)} - l_{0} m_{2} \sin{\left (q_{1} \right)} - l_{0} m_{3} \sin{\left (q_{1} \right)} & - l_{1} m_{2} \sin{\left (q_{2} \right)} - l_{1} m_{3} \sin{\left (q_{2} \right)} & - l_{2} m_{3} \sin{\left (q_{3} \right)}\\- l_{0} m_{1} \sin{\left (q_{1} \right)} - l_{0} m_{2} \sin{\left (q_{1} \right)} - l_{0} m_{3} \sin{\left (q_{1} \right)} & l_{0}^{2} m_{1} + l_{0}^{2} m_{2} + l_{0}^{2} m_{3} & l_{0} l_{1} m_{2} \left(\sin{\left (q_{1} \right)} \sin{\left (q_{2} \right)} + \cos{\left (q_{1} \right)} \cos{\left (q_{2} \right)}\right) + l_{0} l_{1} m_{3} \left(\sin{\left (q_{1} \right)} \sin{\left (q_{2} \right)} + \cos{\left (q_{1} \right)} \cos{\left (q_{2} \right)}\right) & l_{0} l_{2} m_{3} \left(\sin{\left (q_{1} \right)} \sin{\left (q_{3} \right)} + \cos{\left (q_{1} \right)} \cos{\left (q_{3} \right)}\right)\\- l_{1} m_{2} \sin{\left (q_{2} \right)} - l_{1} m_{3} \sin{\left (q_{2} \right)} & l_{0} l_{1} m_{2} \left(\sin{\left (q_{1} \right)} \sin{\left (q_{2} \right)} + \cos{\left (q_{1} \right)} \cos{\left (q_{2} \right)}\right) + l_{0} l_{1} m_{3} \left(\sin{\left (q_{1} \right)} \sin{\left (q_{2} \right)} + \cos{\left (q_{1} \right)} \cos{\left (q_{2} \right)}\right) & l_{1}^{2} m_{2} + l_{1}^{2} m_{3} & l_{1} l_{2} m_{3} \left(\sin{\left (q_{2} \right)} \sin{\left (q_{3} \right)} + \cos{\left (q_{2} \right)} \cos{\left (q_{3} \right)}\right)\\- l_{2} m_{3} \sin{\left (q_{3} \right)} & l_{0} l_{2} m_{3} \left(\sin{\left (q_{1} \right)} \sin{\left (q_{3} \right)} + \cos{\left (q_{1} \right)} \cos{\left (q_{3} \right)}\right) & l_{1} l_{2} m_{3} \left(\sin{\left (q_{2} \right)} \sin{\left (q_{3} \right)} + \cos{\left (q_{2} \right)} \cos{\left (q_{3} \right)}\right) & l_{2}^{2} m_{3}\end{matrix}\right]$$

In [9]:
```
sym.trigsimp(sys.eom_method.mass_matrix)
```

Out[9]:

$$\left[\begin{matrix}m_{0} + m_{1} + m_{2} + m_{3} & - l_{0} \left(m_{1} + m_{2} + m_{3}\right) \sin{\left (q_{1} \right)} & - l_{1} \left(m_{2} + m_{3}\right) \sin{\left (q_{2} \right)} & - l_{2} m_{3} \sin{\left (q_{3} \right)}\\- l_{0} \left(m_{1} + m_{2} + m_{3}\right) \sin{\left (q_{1} \right)} & l_{0}^{2} m_{1} + l_{0}^{2} m_{2} + l_{0}^{2} m_{3} & l_{0} l_{1} \left(m_{2} + m_{3}\right) \cos{\left (q_{1} - q_{2} \right)} & l_{0} l_{2} m_{3} \cos{\left (q_{1} - q_{3} \right)}\\- l_{1} \left(m_{2} + m_{3}\right) \sin{\left (q_{2} \right)} & l_{0} l_{1} \left(m_{2} + m_{3}\right) \cos{\left (q_{1} - q_{2} \right)} & l_{1}^{2} m_{2} + l_{1}^{2} m_{3} & l_{1} l_{2} m_{3} \cos{\left (q_{2} - q_{3} \right)}\\- l_{2} m_{3} \sin{\left (q_{3} \right)} & l_{0} l_{2} m_{3} \cos{\left (q_{1} - q_{3} \right)} & l_{1} l_{2} m_{3} \cos{\left (q_{2} - q_{3} \right)} & l_{2}^{2} m_{3}\end{matrix}\right]$$

In [10]:
```
sys.eom_method.forcing
```

Out[10]:

$$\left[\begin{matrix}l_{0} m_{1} u_{1}^{2} \cos{\left (q_{1} \right)} + l_{0} m_{2} u_{1}^{2} \cos{\left (q_{1} \right)} + l_{0} m_{3} u_{1}^{2} \cos{\left (q_{1} \right)} + l_{1} m_{2} u_{2}^{2} \cos{\left (q_{2} \right)} + l_{1} m_{3} u_{2}^{2} \cos{\left (q_{2} \right)} + l_{2} m_{3} u_{3}^{2} \cos{\left (q_{3} \right)} + F\\- g l_{0} m_{1} \cos{\left (q_{1} \right)} - g l_{0} m_{2} \cos{\left (q_{1} \right)} - g l_{0} m_{3} \cos{\left (q_{1} \right)} + l_{0} l_{1} m_{2} \left(- \sin{\left (q_{1} \right)} \cos{\left (q_{2} \right)} + \sin{\left (q_{2} \right)} \cos{\left (q_{1} \right)}\right) u_{2}^{2} + l_{0} l_{1} m_{3} \left(- \sin{\left (q_{1} \right)} \cos{\left (q_{2} \right)} + \sin{\left (q_{2} \right)} \cos{\left (q_{1} \right)}\right) u_{2}^{2} + l_{0} l_{2} m_{3} \left(- \sin{\left (q_{1} \right)} \cos{\left (q_{3} \right)} + \sin{\left (q_{3} \right)} \cos{\left (q_{1} \right)}\right) u_{3}^{2}\\- g l_{1} m_{2} \cos{\left (q_{2} \right)} - g l_{1} m_{3} \cos{\left (q_{2} \right)} + l_{0} l_{1} m_{2} \left(\sin{\left (q_{1} \right)} \cos{\left (q_{2} \right)} - \sin{\left (q_{2} \right)} \cos{\left (q_{1} \right)}\right) u_{1}^{2} + l_{0} l_{1} m_{3} \left(\sin{\left (q_{1} \right)} \cos{\left (q_{2} \right)} - \sin{\left (q_{2} \right)} \cos{\left (q_{1} \right)}\right) u_{1}^{2} + l_{1} l_{2} m_{3} \left(- \sin{\left (q_{2} \right)} \cos{\left (q_{3} \right)} + \sin{\left (q_{3} \right)} \cos{\left (q_{2} \right)}\right) u_{3}^{2}\\- g l_{2} m_{3} \cos{\left (q_{3} \right)} + l_{0} l_{2} m_{3} \left(\sin{\left (q_{1} \right)} \cos{\left (q_{3} \right)} - \sin{\left (q_{3} \right)} \cos{\left (q_{1} \right)}\right) u_{1}^{2} + l_{1} l_{2} m_{3} \left(\sin{\left (q_{2} \right)} \cos{\left (q_{3} \right)} - \sin{\left (q_{3} \right)} \cos{\left (q_{2} \right)}\right) u_{2}^{2}\end{matrix}\right]$$

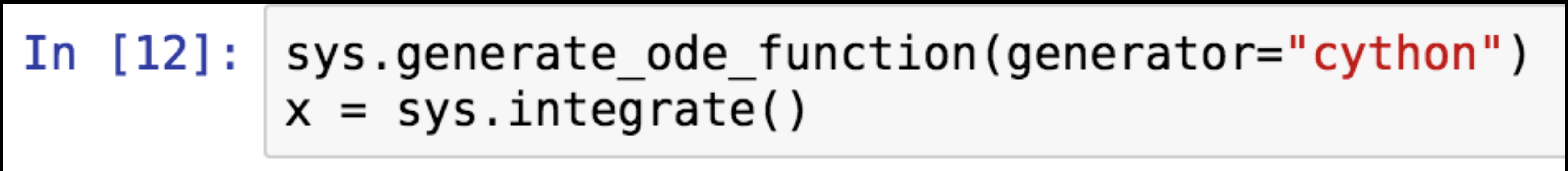
In [11]:
```
sym.trigsimp(sys.eom_method.forcing)
```

Out[11]:

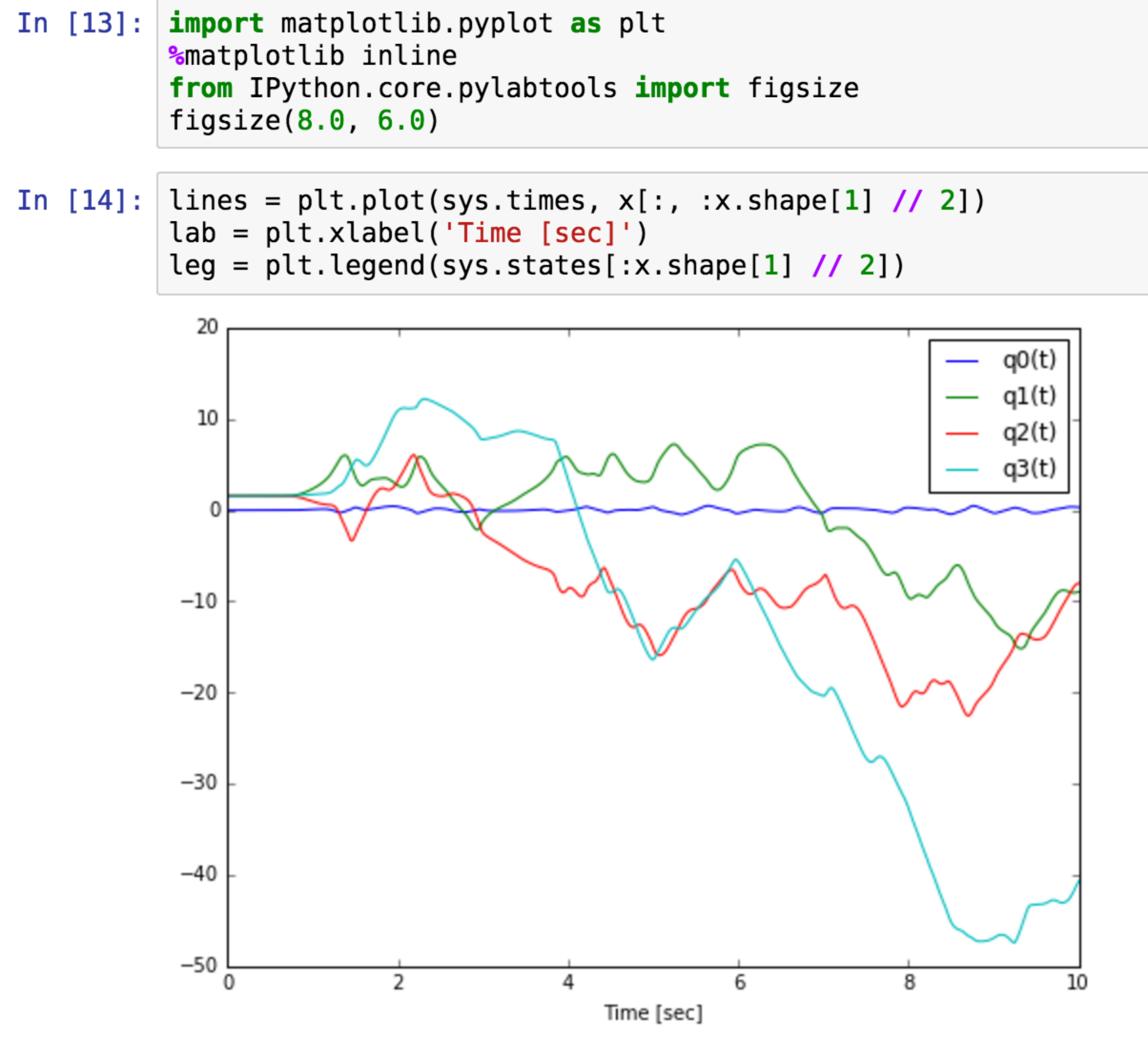
$$\left[\begin{matrix}l_{0} m_{1} u_{1}^{2} \cos{\left (q_{1} \right)} + l_{0} m_{2} u_{1}^{2} \cos{\left (q_{1} \right)} + l_{0} m_{3} u_{1}^{2} \cos{\left (q_{1} \right)} + l_{1} m_{2} u_{2}^{2} \cos{\left (q_{2} \right)} + l_{1} m_{3} u_{2}^{2} \cos{\left (q_{2} \right)} + l_{2} m_{3} u_{3}^{2} \cos{\left (q_{3} \right)} + F\\- l_{0} \left(g m_{1} \cos{\left (q_{1} \right)} + g m_{2} \cos{\left (q_{1} \right)} + g m_{3} \cos{\left (q_{1} \right)} + l_{1} m_{2} u_{2}^{2} \sin{\left (q_{1} - q_{2} \right)} + l_{1} m_{3} u_{2}^{2} \sin{\left (q_{1} - q_{2} \right)} + l_{2} m_{3} u_{3}^{2} \sin{\left (q_{1} - q_{3} \right)}\right)\\l_{1} \left(- g m_{2} \cos{\left (q_{2} \right)} - g m_{3} \cos{\left (q_{2} \right)} + l_{0} m_{2} u_{1}^{2} \sin{\left (q_{1} - q_{2} \right)} + l_{0} m_{3} u_{1}^{2} \sin{\left (q_{1} - q_{2} \right)} - l_{2} m_{3} u_{3}^{2} \sin{\left (q_{2} - q_{3} \right)}\right)\\l_{2} m_{3} \left(- g \cos{\left (q_{3} \right)} + l_{0} u_{1}^{2} \sin{\left (q_{1} - q_{3} \right)} + l_{1} u_{2}^{2} \sin{\left (q_{2} - q_{3} \right)}\right)\end{matrix}\right]$$

- `trigsimp()` can simplify the equations of motion significantly
    - In this example,
      $\sin(x) \cdot \cos(y) - \sin(y) \cdot \cos(x) \longrightarrow \sin(x - y)$
- The equations must be evaluated at each time step, so this can make a significant difference in performance

- PyDy automatically generates a fast ODE solve callback using SymPy code generation and Cython



```
In [12]: sys.generate_ode_function(generator="cython")
         x = sys.integrate()
```



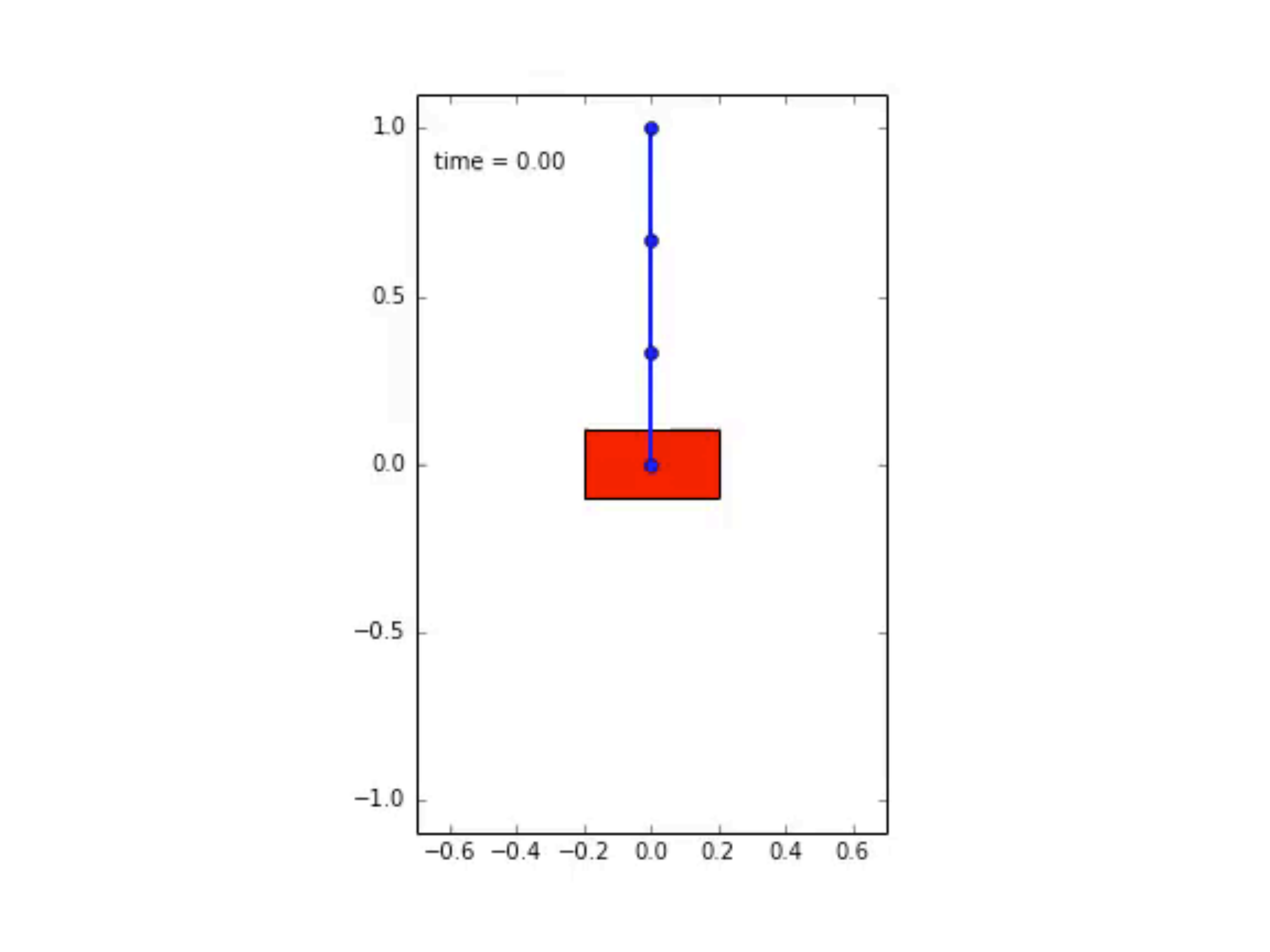
In [13]:

```
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.core.pylabtools import figsize
figsize(8.0, 6.0)
```

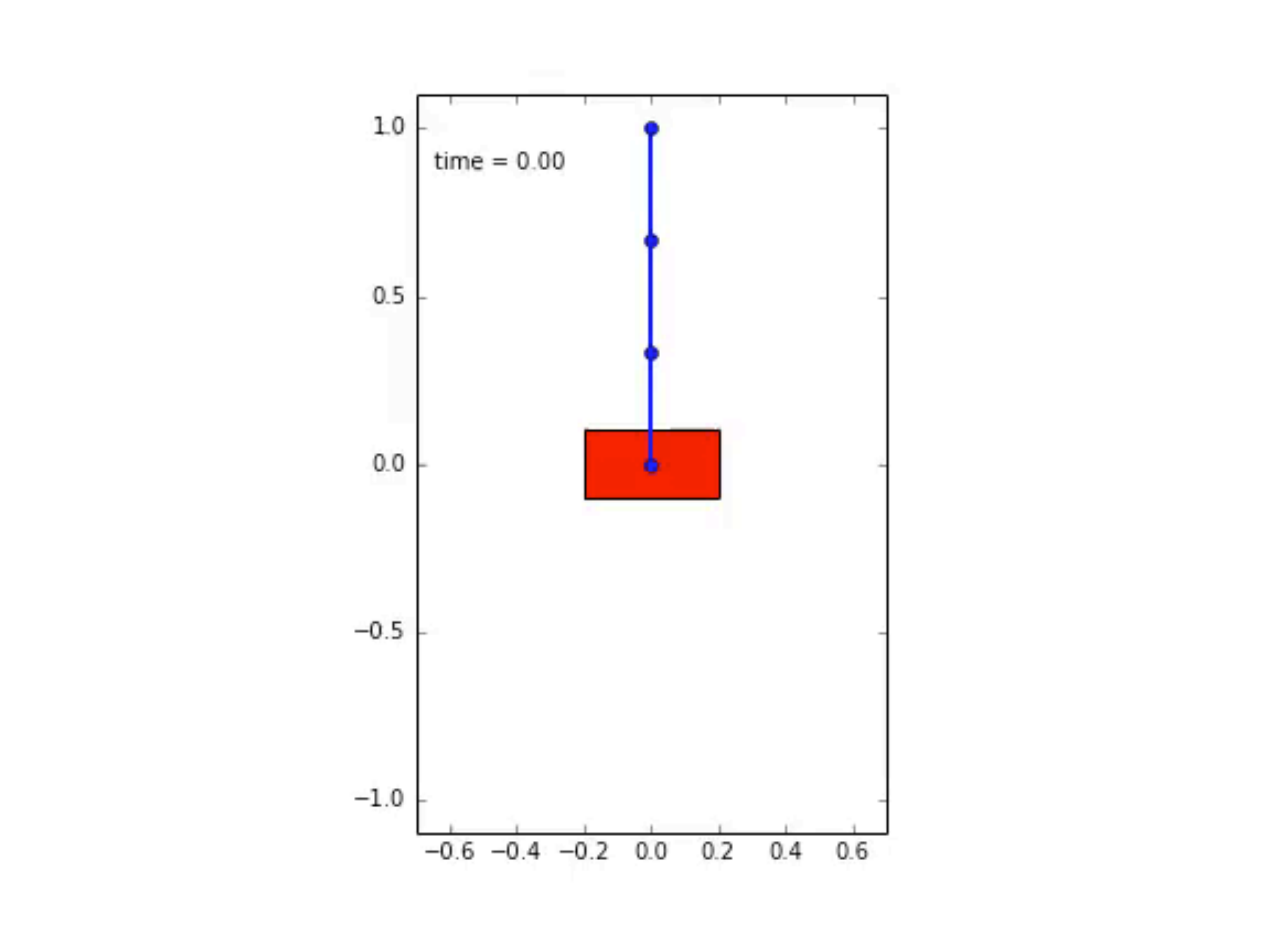
In [14]:

```
lines = plt.plot(sys.times, x[:, :x.shape[1] // 2])
lab = plt.xlabel('Time [sec]')
leg = plt.legend(sys.states[:x.shape[1] // 2])
```

Animation (code not shown)

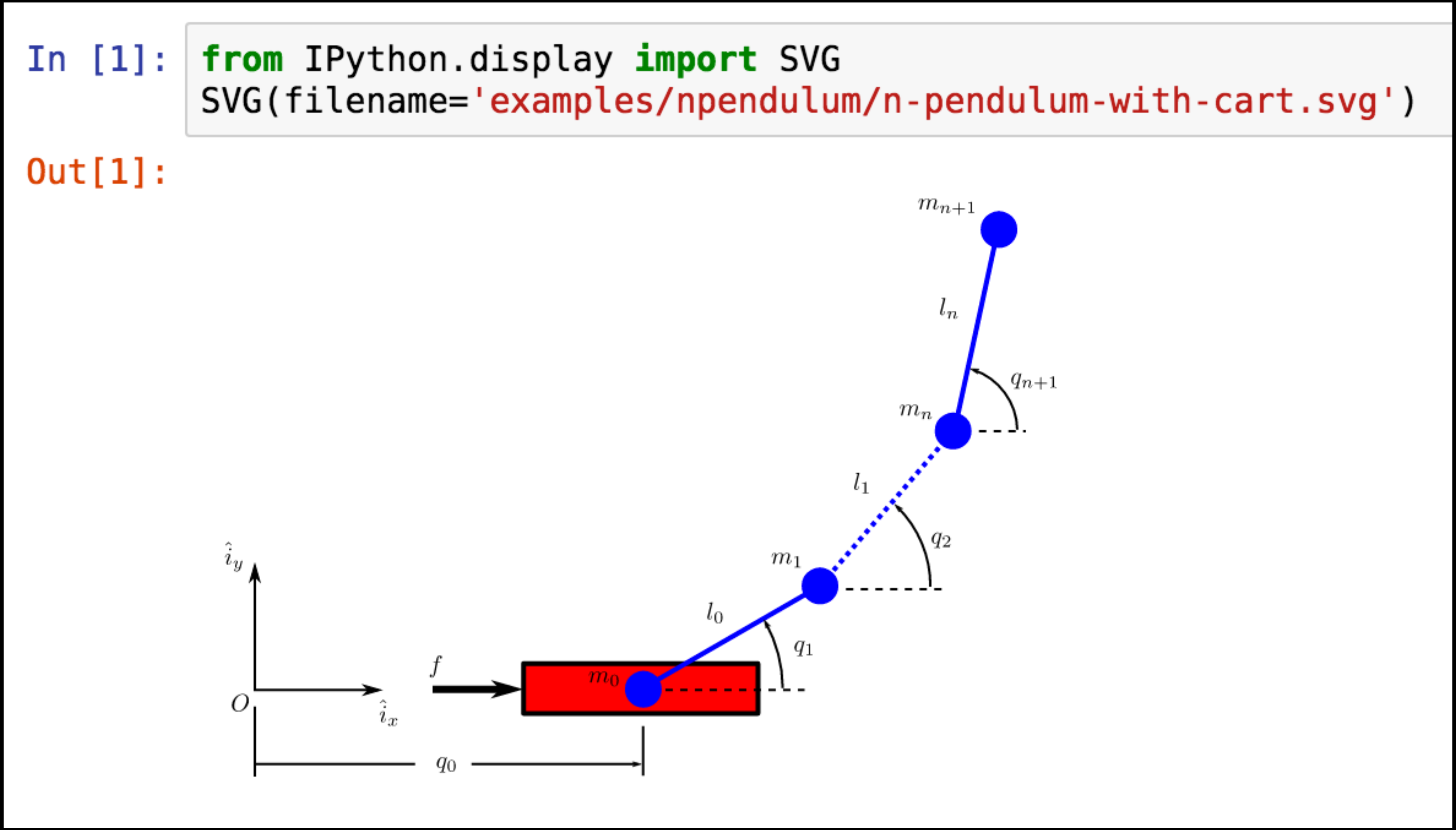


Animation (code not shown)



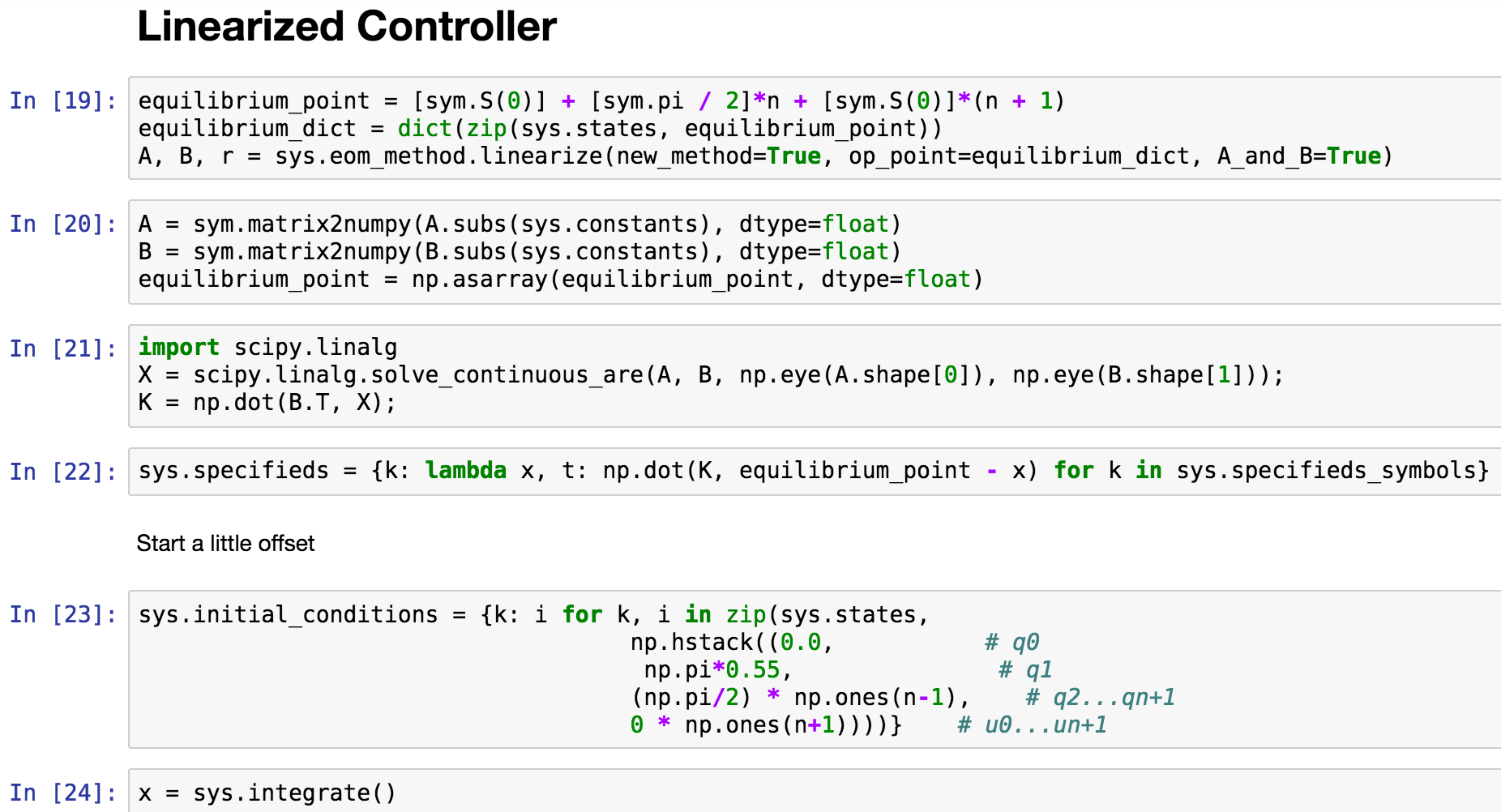
We can control the pendulum by forcing the cart

Apply the force f to keep the pendulum upright



```
In [1]: from IPython.display import SVG
        SVG(filename='examples/npendulum/n-pendulum-with-cart.svg')
```

Out[1]:



Linearized Controller

```
In [19]: equilibrium_point = [sym.S(0)] + [sym.pi / 2]*n + [sym.S(0)]*(n + 1)
         equilibrium_dict = dict(zip(sys.states, equilibrium_point))
         A, B, r = sys.eom_method.linearize(new_method=True, op_point=equilibrium_dict, A_and_B=True)
```

```
In [20]: A = sym.matrix2numpy(A.subs(sys.constants), dtype=float)
         B = sym.matrix2numpy(B.subs(sys.constants), dtype=float)
         equilibrium_point = np.asarray(equilibrium_point, dtype=float)
```

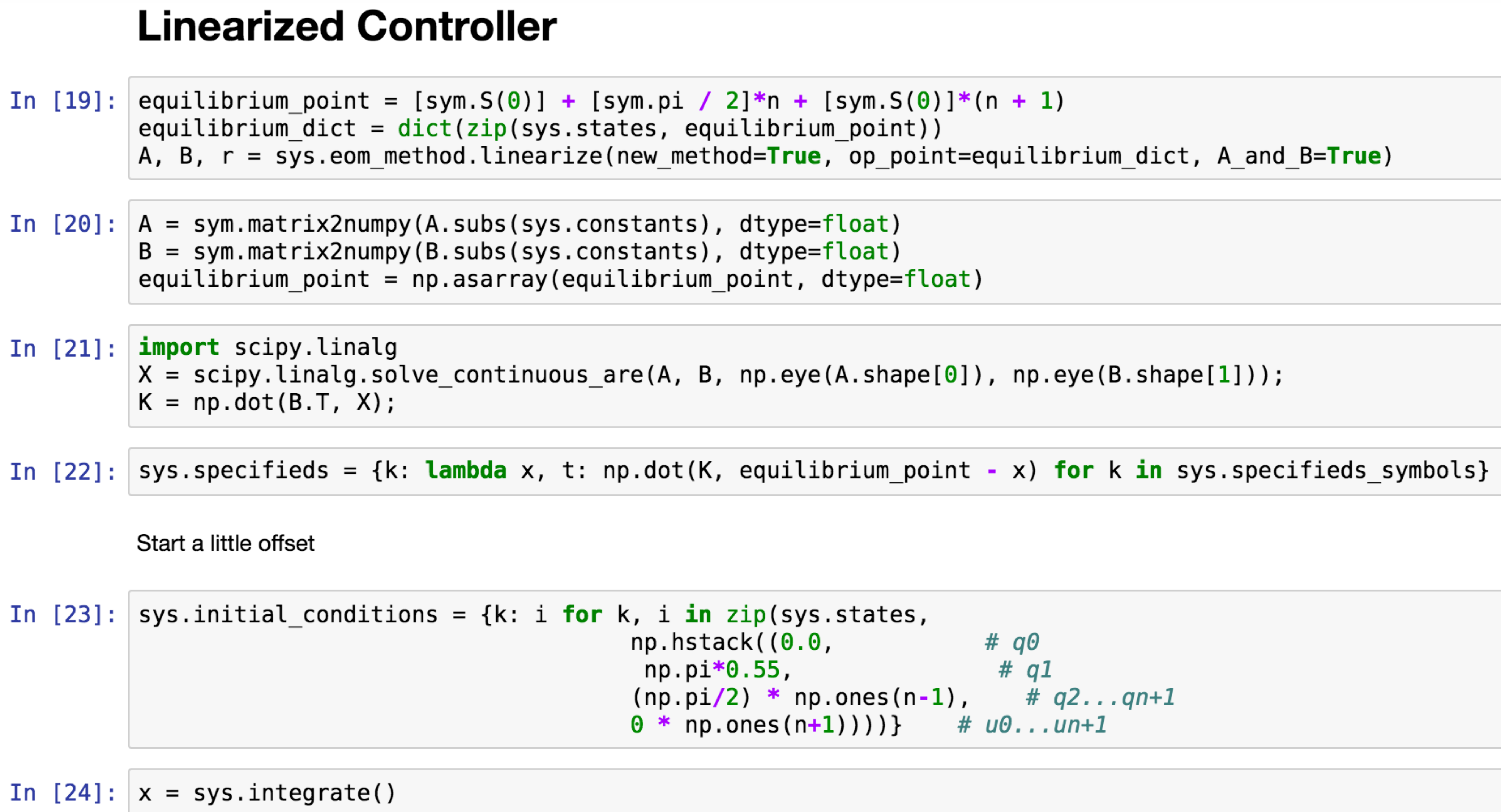
```
In [21]: import scipy.linalg
         X = scipy.linalg.solve_continuous_are(A, B, np.eye(A.shape[0]), np.eye(B.shape[1]));
         K = np.dot(B.T, X);
```

```
In [22]: sys.specifieds = {k: lambda x, t: np.dot(K, equilibrium_point - x) for k in sys.specifieds_symbols}
```

Start a little offset

```
In [23]: sys.initial_conditions = {k: i for k, i in zip(sys.states,
                                    np.hstack((0.0,           # q0
                                     np.pi*0.55,               # q1
                                    (np.pi/2) * np.ones(n-1),    # q2...qn+1
                                    0 * np.ones(n+1))))}    # u0...un+1
```

```
In [24]: x = sys.integrate()
```



Linearized Controller

In [19]:

```
equilibrium_point = [sym.S(0)] + [sym.pi / 2]*n + [sym.S(0)]*(n + 1)
equilibrium_dict = dict(zip(sys.states, equilibrium_point))
A, B, r = sys.eom_method.linearize(new_method=True, op_point=equilibrium_dict, A_and_B=True)
```

In [20]:

```
A = sym.matrix2numpy(A.subs(sys.constants), dtype=float)
B = sym.matrix2numpy(B.subs(sys.constants), dtype=float)
equilibrium_point = np.asarray(equilibrium_point, dtype=float)
```

In [21]:

```
import scipy.linalg
X = scipy.linalg.solve_continuous_are(A, B, np.eye(A.shape[0]), np.eye(B.shape[1]));
K = np.dot(B.T, X);
```

In [22]:

```
sys.specifieds = {k: lambda x, t: np.dot(K, equilibrium_point - x) for k in sys.specifieds_symbols}
```

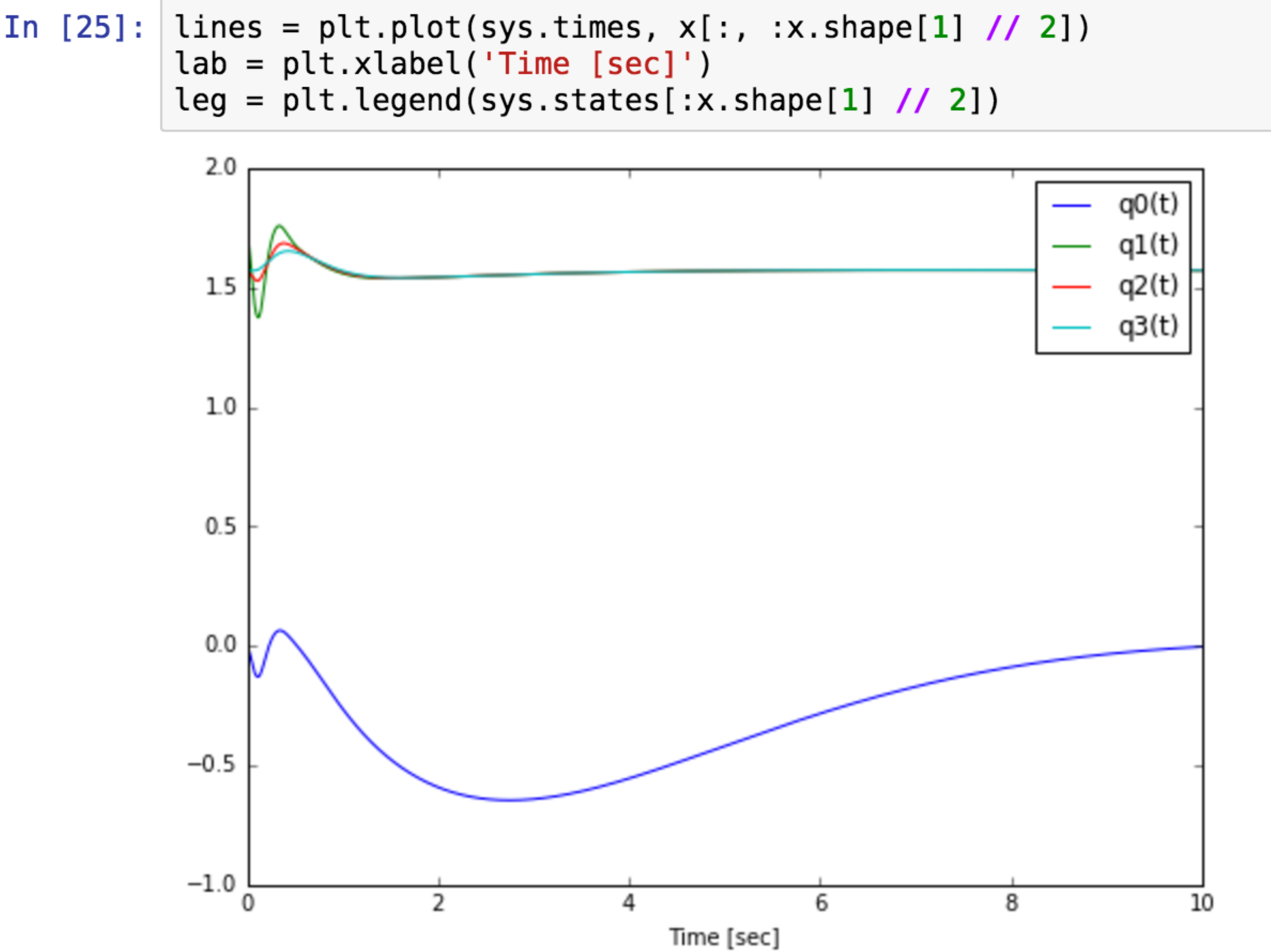
Start a little offset

In [23]:

```
sys.initial_conditions = {k: i for k, i in zip(sys.states,
                                    np.hstack((0.0,          # q0
                                     np.pi*0.55,               # q1
                                    (np.pi/2) * np.ones(n-1),    # q2...qn+1
                                    0 * np.ones(n+1))))}    # u0...un+1
```

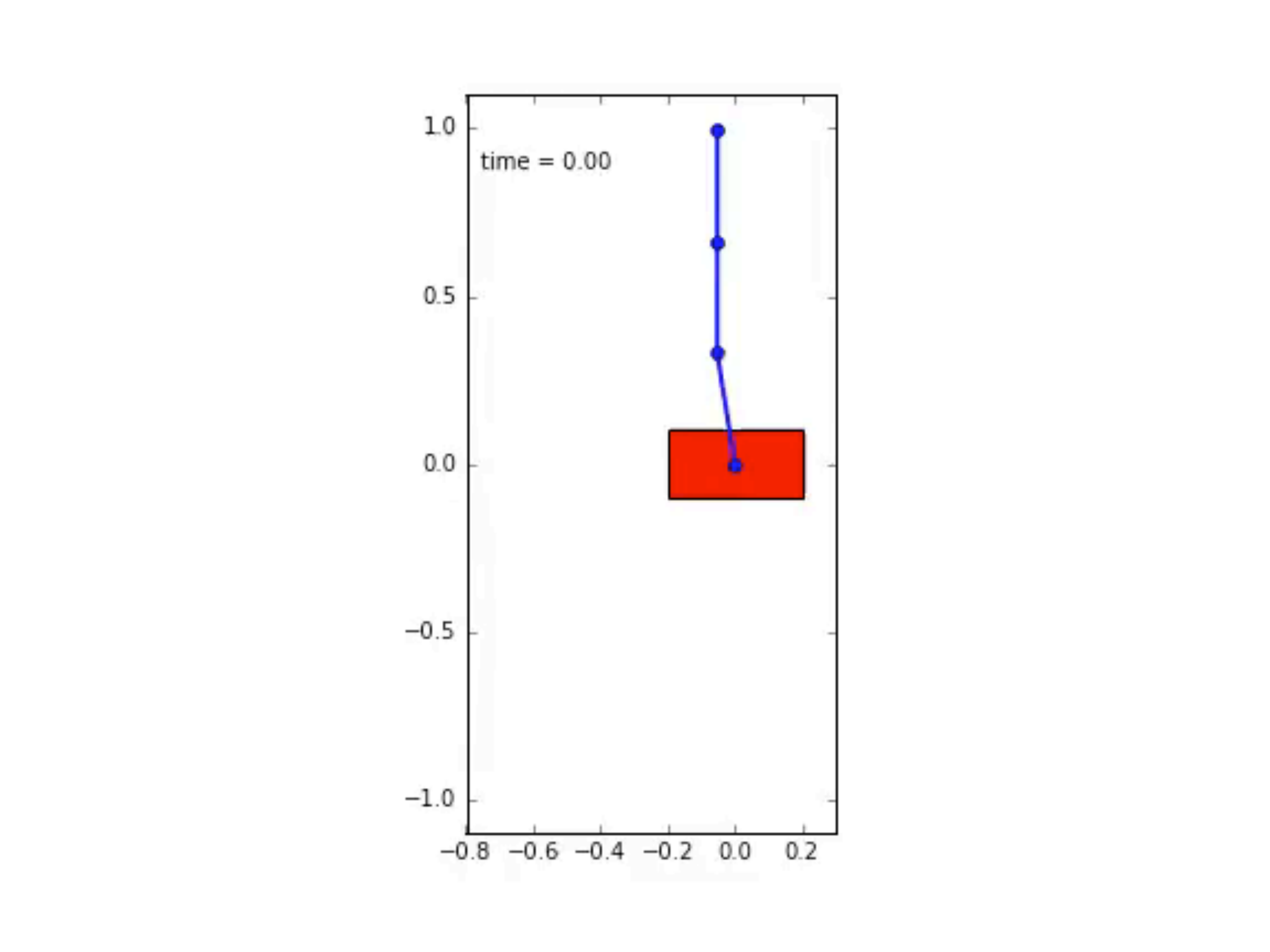
In [24]:

```
x = sys.integrate()
```

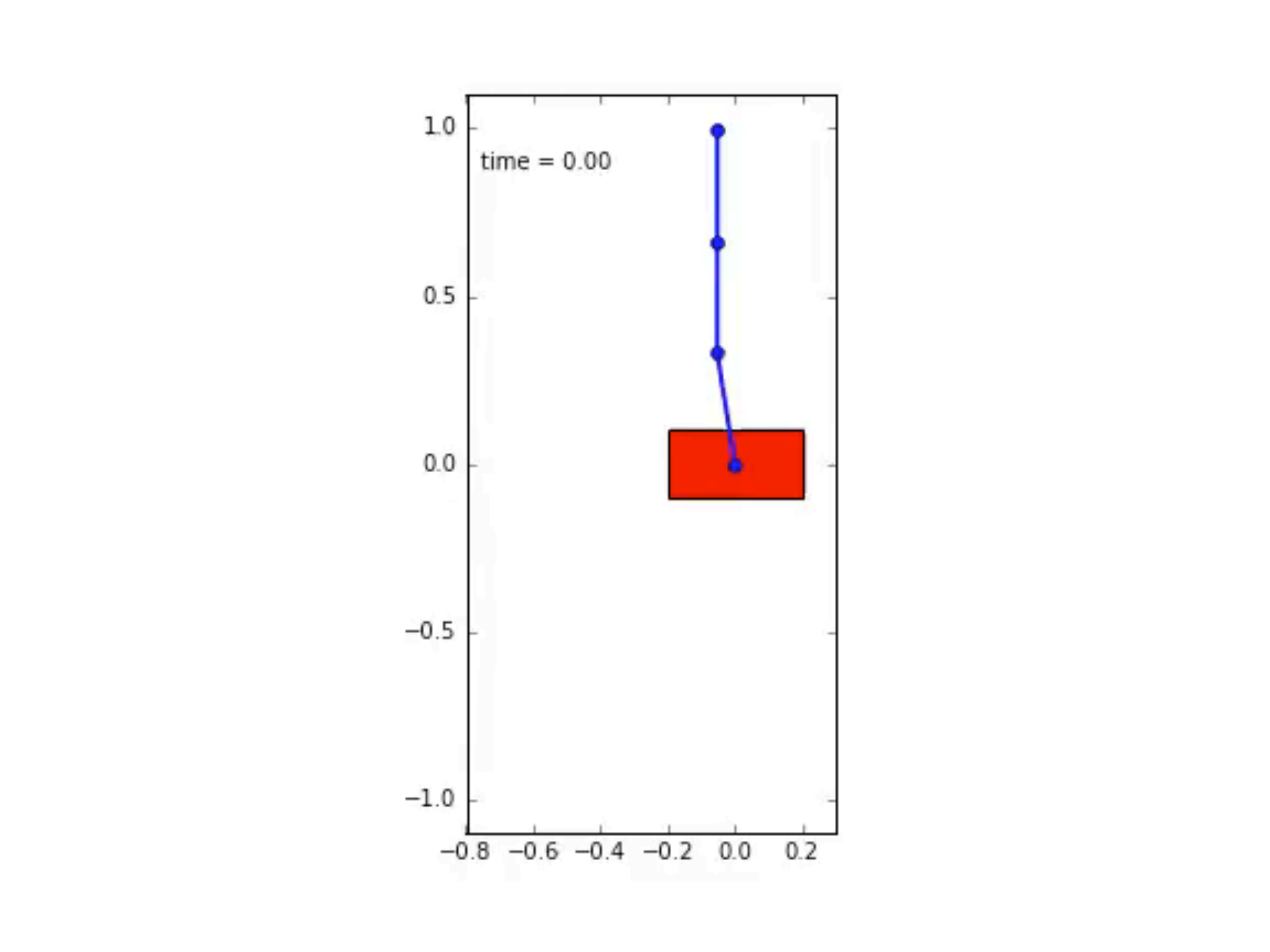


```
In [25]: lines = plt.plot(sys.times, x[:, :x.shape[1] // 2])
         lab = plt.xlabel('Time [sec]')
         leg = plt.legend(sys.states[:x.shape[1] // 2])
```

Animation



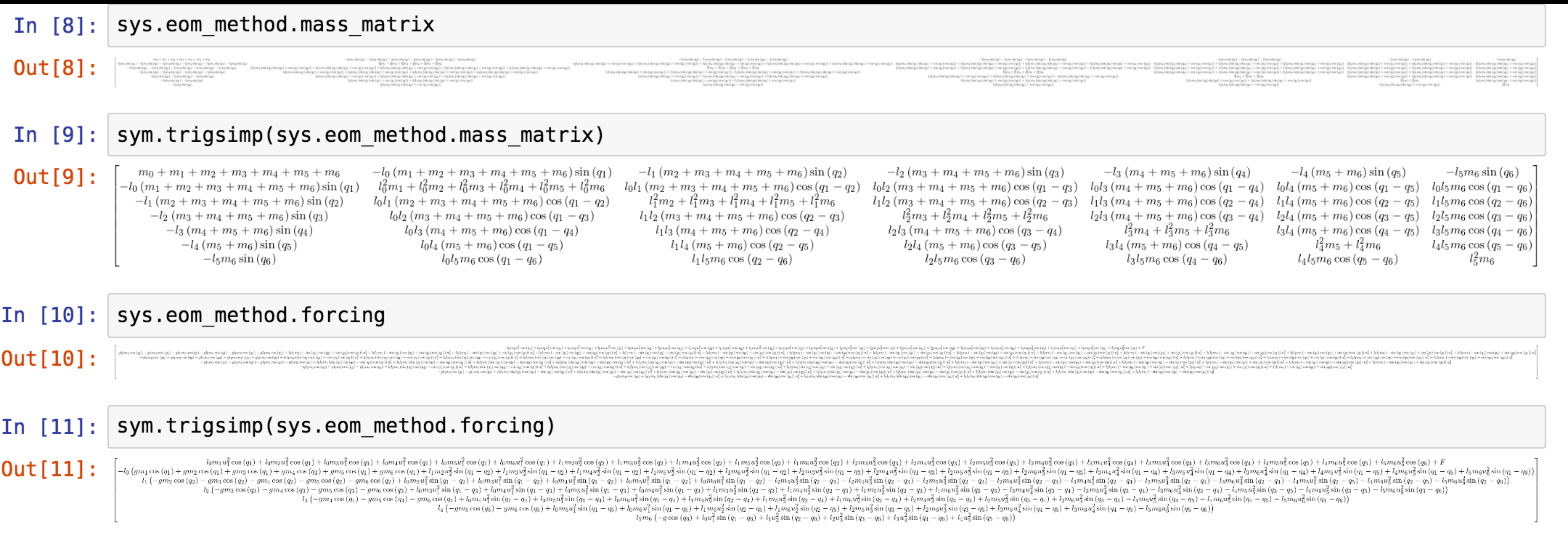
Animation



Yes this is possible

https://www.youtube.com/watch?v=cyN-CRNrb3E

We can increase the number of links (n=6)



In [8]:
```
sys.eom_method.mass_matrix
```

Out[8]: [illegible]

In [9]:
```
sym.trigsimp(sys.eom_method.mass_matrix)
```

Out[9]:

$$\left[\begin{array}{ccccccc}
m_0+m_1+m_2+m_3+m_4+m_5+m_6 & -l_0\left(m_1+m_2+m_3+m_4+m_5+m_6\right)\sin\left(q_1\right) & -l_1\left(m_2+m_3+m_4+m_5+m_6\right)\sin\left(q_2\right) & -l_2\left(m_3+m_4+m_5+m_6\right)\sin\left(q_3\right) & -l_3\left(m_4+m_5+m_6\right)\sin\left(q_4\right) & -l_4\left(m_5+m_6\right)\sin\left(q_5\right) & -l_5m_6\sin\left(q_6\right) \\
-l_0\left(m_1+m_2+m_3+m_4+m_5+m_6\right)\sin\left(q_1\right) & l_0^2m_1+l_0^2m_2+l_0^2m_3+l_0^2m_4+l_0^2m_5+l_0^2m_6 & l_0l_1\left(m_2+m_3+m_4+m_5+m_6\right)\cos\left(q_1-q_2\right) & l_0l_2\left(m_3+m_4+m_5+m_6\right)\cos\left(q_1-q_3\right) & l_0l_3\left(m_4+m_5+m_6\right)\cos\left(q_1-q_4\right) & l_0l_4\left(m_5+m_6\right)\cos\left(q_1-q_5\right) & l_0l_5m_6\cos\left(q_1-q_6\right) \\
-l_1\left(m_2+m_3+m_4+m_5+m_6\right)\sin\left(q_2\right) & l_0l_1\left(m_2+m_3+m_4+m_5+m_6\right)\cos\left(q_1-q_2\right) & l_1^2m_2+l_1^2m_3+l_1^2m_4+l_1^2m_5+l_1^2m_6 & l_1l_2\left(m_3+m_4+m_5+m_6\right)\cos\left(q_2-q_3\right) & l_1l_3\left(m_4+m_5+m_6\right)\cos\left(q_2-q_4\right) & l_1l_4\left(m_5+m_6\right)\cos\left(q_2-q_5\right) & l_1l_5m_6\cos\left(q_2-q_6\right) \\
-l_2\left(m_3+m_4+m_5+m_6\right)\sin\left(q_3\right) & l_0l_2\left(m_3+m_4+m_5+m_6\right)\cos\left(q_1-q_3\right) & l_1l_2\left(m_3+m_4+m_5+m_6\right)\cos\left(q_2-q_3\right) & l_2^2m_3+l_2^2m_4+l_2^2m_5+l_2^2m_6 & l_2l_3\left(m_4+m_5+m_6\right)\cos\left(q_3-q_4\right) & l_2l_4\left(m_5+m_6\right)\cos\left(q_3-q_5\right) & l_2l_5m_6\cos\left(q_3-q_6\right) \\
-l_3\left(m_4+m_5+m_6\right)\sin\left(q_4\right) & l_0l_3\left(m_4+m_5+m_6\right)\cos\left(q_1-q_4\right) & l_1l_3\left(m_4+m_5+m_6\right)\cos\left(q_2-q_4\right) & l_2l_3\left(m_4+m_5+m_6\right)\cos\left(q_3-q_4\right) & l_3^2m_4+l_3^2m_5+l_3^2m_6 & l_3l_4\left(m_5+m_6\right)\cos\left(q_4-q_5\right) & l_3l_5m_6\cos\left(q_4-q_6\right) \\
-l_4\left(m_5+m_6\right)\sin\left(q_5\right) & l_0l_4\left(m_5+m_6\right)\cos\left(q_1-q_5\right) & l_1l_4\left(m_5+m_6\right)\cos\left(q_2-q_5\right) & l_2l_4\left(m_5+m_6\right)\cos\left(q_3-q_5\right) & l_3l_4\left(m_5+m_6\right)\cos\left(q_4-q_5\right) & l_4^2m_5+l_4^2m_6 & l_4l_5m_6\cos\left(q_5-q_6\right) \\
-l_5m_6\sin\left(q_6\right) & l_0l_5m_6\cos\left(q_1-q_6\right) & l_1l_5m_6\cos\left(q_2-q_6\right) & l_2l_5m_6\cos\left(q_3-q_6\right) & l_3l_5m_6\cos\left(q_4-q_6\right) & l_4l_5m_6\cos\left(q_5-q_6\right) & l_5^2m_6
\end{array}\right]$$

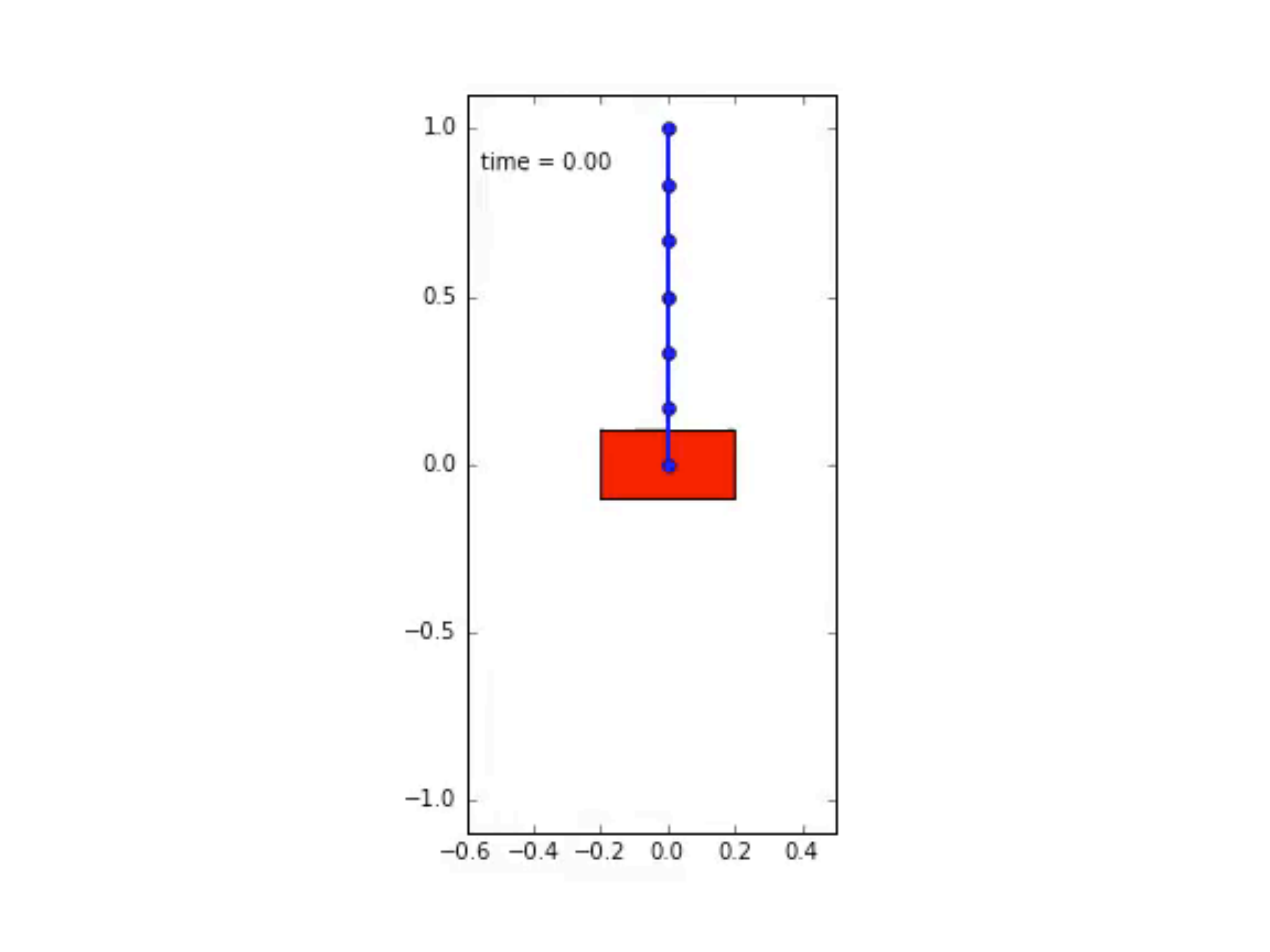
In [10]:
```
sys.eom_method.forcing
```

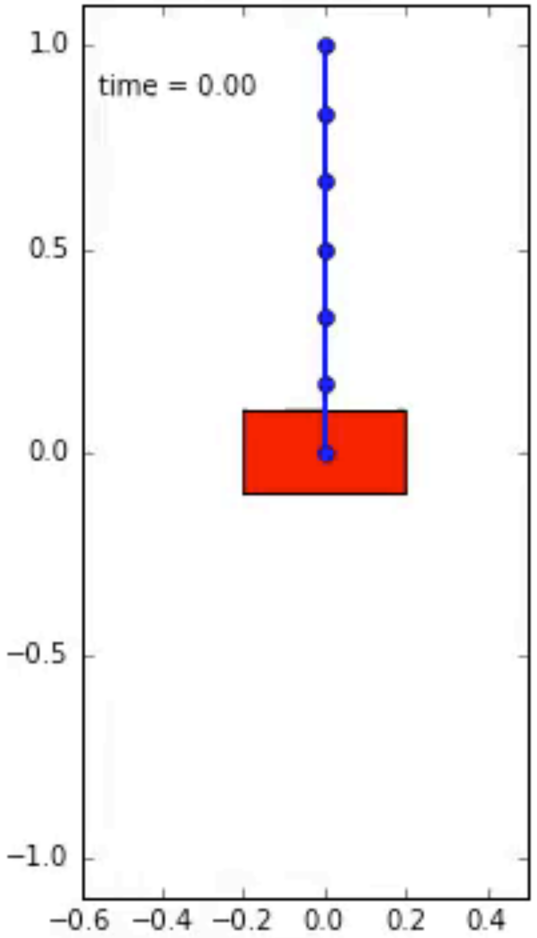
Out[10]: [illegible]

In [11]:
```
sym.trigsimp(sys.eom_method.forcing)
```

Out[11]:

$$\left[\begin{array}{c}
l_0m_1u_1^2\cos\left(q_1\right)+l_0m_2u_1^2\cos\left(q_1\right)+l_0m_3u_1^2\cos\left(q_1\right)+l_0m_4u_1^2\cos\left(q_1\right)+l_0m_5u_1^2\cos\left(q_1\right)+l_0m_6u_1^2\cos\left(q_1\right)+l_1m_2u_2^2\cos\left(q_2\right)+l_1m_3u_2^2\cos\left(q_2\right)+l_1m_4u_2^2\cos\left(q_2\right)+l_1m_5u_2^2\cos\left(q_2\right)+l_1m_6u_2^2\cos\left(q_2\right)+l_2m_3u_3^2\cos\left(q_3\right)+l_2m_4u_3^2\cos\left(q_3\right)+l_2m_5u_3^2\cos\left(q_3\right)+l_2m_6u_3^2\cos\left(q_3\right)+l_3m_4u_4^2\cos\left(q_4\right)+l_3m_5u_4^2\cos\left(q_4\right)+l_3m_6u_4^2\cos\left(q_4\right)+l_4m_5u_5^2\cos\left(q_5\right)+l_4m_6u_5^2\cos\left(q_5\right)+l_5m_6u_6^2\cos\left(q_6\right)+F \\
-l_0\left(gm_1\cos\left(q_1\right)+gm_2\cos\left(q_1\right)+gm_3\cos\left(q_1\right)+gm_4\cos\left(q_1\right)+gm_5\cos\left(q_1\right)+gm_6\cos\left(q_1\right)+l_1m_2u_2^2\sin\left(q_1-q_2\right)+l_1m_3u_2^2\sin\left(q_1-q_2\right)+l_1m_4u_2^2\sin\left(q_1-q_2\right)+l_1m_5u_2^2\sin\left(q_1-q_2\right)+l_1m_6u_2^2\sin\left(q_1-q_2\right)+l_2m_3u_3^2\sin\left(q_1-q_3\right)+l_2m_4u_3^2\sin\left(q_1-q_3\right)+l_2m_5u_3^2\sin\left(q_1-q_3\right)+l_2m_6u_3^2\sin\left(q_1-q_3\right)+l_3m_4u_4^2\sin\left(q_1-q_4\right)+l_3m_5u_4^2\sin\left(q_1-q_4\right)+l_3m_6u_4^2\sin\left(q_1-q_4\right)+l_4m_5u_5^2\sin\left(q_1-q_5\right)+l_4m_6u_5^2\sin\left(q_1-q_5\right)+l_5m_6u_6^2\sin\left(q_1-q_6\right)\right) \\
l_1\left(-gm_2\cos\left(q_2\right)-gm_3\cos\left(q_2\right)-gm_4\cos\left(q_2\right)-gm_5\cos\left(q_2\right)-gm_6\cos\left(q_2\right)+l_0m_2u_1^2\sin\left(q_1-q_2\right)+l_0m_3u_1^2\sin\left(q_1-q_2\right)+l_0m_4u_1^2\sin\left(q_1-q_2\right)+l_0m_5u_1^2\sin\left(q_1-q_2\right)+l_0m_6u_1^2\sin\left(q_1-q_2\right)-l_2m_3u_3^2\sin\left(q_2-q_3\right)-l_2m_4u_3^2\sin\left(q_2-q_3\right)-l_2m_5u_3^2\sin\left(q_2-q_3\right)-l_2m_6u_3^2\sin\left(q_2-q_3\right)-l_3m_4u_4^2\sin\left(q_2-q_4\right)-l_3m_5u_4^2\sin\left(q_2-q_4\right)-l_3m_6u_4^2\sin\left(q_2-q_4\right)-l_4m_5u_5^2\sin\left(q_2-q_5\right)-l_4m_6u_5^2\sin\left(q_2-q_5\right)-l_5m_6u_6^2\sin\left(q_2-q_6\right)\right) \\
l_2\left(-gm_3\cos\left(q_3\right)-gm_4\cos\left(q_3\right)-gm_5\cos\left(q_3\right)-gm_6\cos\left(q_3\right)+l_0m_3u_1^2\sin\left(q_1-q_3\right)+l_0m_4u_1^2\sin\left(q_1-q_3\right)+l_0m_5u_1^2\sin\left(q_1-q_3\right)+l_0m_6u_1^2\sin\left(q_1-q_3\right)+l_1m_3u_2^2\sin\left(q_2-q_3\right)+l_1m_4u_2^2\sin\left(q_2-q_3\right)+l_1m_5u_2^2\sin\left(q_2-q_3\right)+l_1m_6u_2^2\sin\left(q_2-q_3\right)-l_3m_4u_4^2\sin\left(q_3-q_4\right)-l_3m_5u_4^2\sin\left(q_3-q_4\right)-l_3m_6u_4^2\sin\left(q_3-q_4\right)-l_4m_5u_5^2\sin\left(q_3-q_5\right)-l_4m_6u_5^2\sin\left(q_3-q_5\right)-l_5m_6u_6^2\sin\left(q_3-q_6\right)\right) \\
l_3\left(-gm_4\cos\left(q_4\right)-gm_5\cos\left(q_4\right)-gm_6\cos\left(q_4\right)+l_0m_4u_1^2\sin\left(q_1-q_4\right)+l_0m_5u_1^2\sin\left(q_1-q_4\right)+l_0m_6u_1^2\sin\left(q_1-q_4\right)+l_1m_4u_2^2\sin\left(q_2-q_4\right)+l_1m_5u_2^2\sin\left(q_2-q_4\right)+l_1m_6u_2^2\sin\left(q_2-q_4\right)+l_2m_4u_3^2\sin\left(q_3-q_4\right)+l_2m_5u_3^2\sin\left(q_3-q_4\right)+l_2m_6u_3^2\sin\left(q_3-q_4\right)-l_4m_5u_5^2\sin\left(q_4-q_5\right)-l_4m_6u_5^2\sin\left(q_4-q_5\right)-l_5m_6u_6^2\sin\left(q_4-q_6\right)\right) \\
l_4\left(-gm_5\cos\left(q_5\right)-gm_6\cos\left(q_5\right)+l_0m_5u_1^2\sin\left(q_1-q_5\right)+l_0m_6u_1^2\sin\left(q_1-q_5\right)+l_1m_5u_2^2\sin\left(q_2-q_5\right)+l_1m_6u_2^2\sin\left(q_2-q_5\right)+l_2m_5u_3^2\sin\left(q_3-q_5\right)+l_2m_6u_3^2\sin\left(q_3-q_5\right)+l_3m_5u_4^2\sin\left(q_4-q_5\right)+l_3m_6u_4^2\sin\left(q_4-q_5\right)-l_5m_6u_6^2\sin\left(q_5-q_6\right)\right) \\
l_5m_6\left(-g\cos\left(q_6\right)+l_0u_1^2\sin\left(q_1-q_6\right)+l_1u_2^2\sin\left(q_2-q_6\right)+l_2u_3^2\sin\left(q_3-q_6\right)+l_3u_4^2\sin\left(q_4-q_6\right)+l_4u_5^2\sin\left(q_5-q_6\right)\right)
\end{array}\right]$$


time = 0.00
1.0
0.5
0.0
−0.5
−1.0
−0.6
−0.4
−0.2
0.0
0.2
0.4

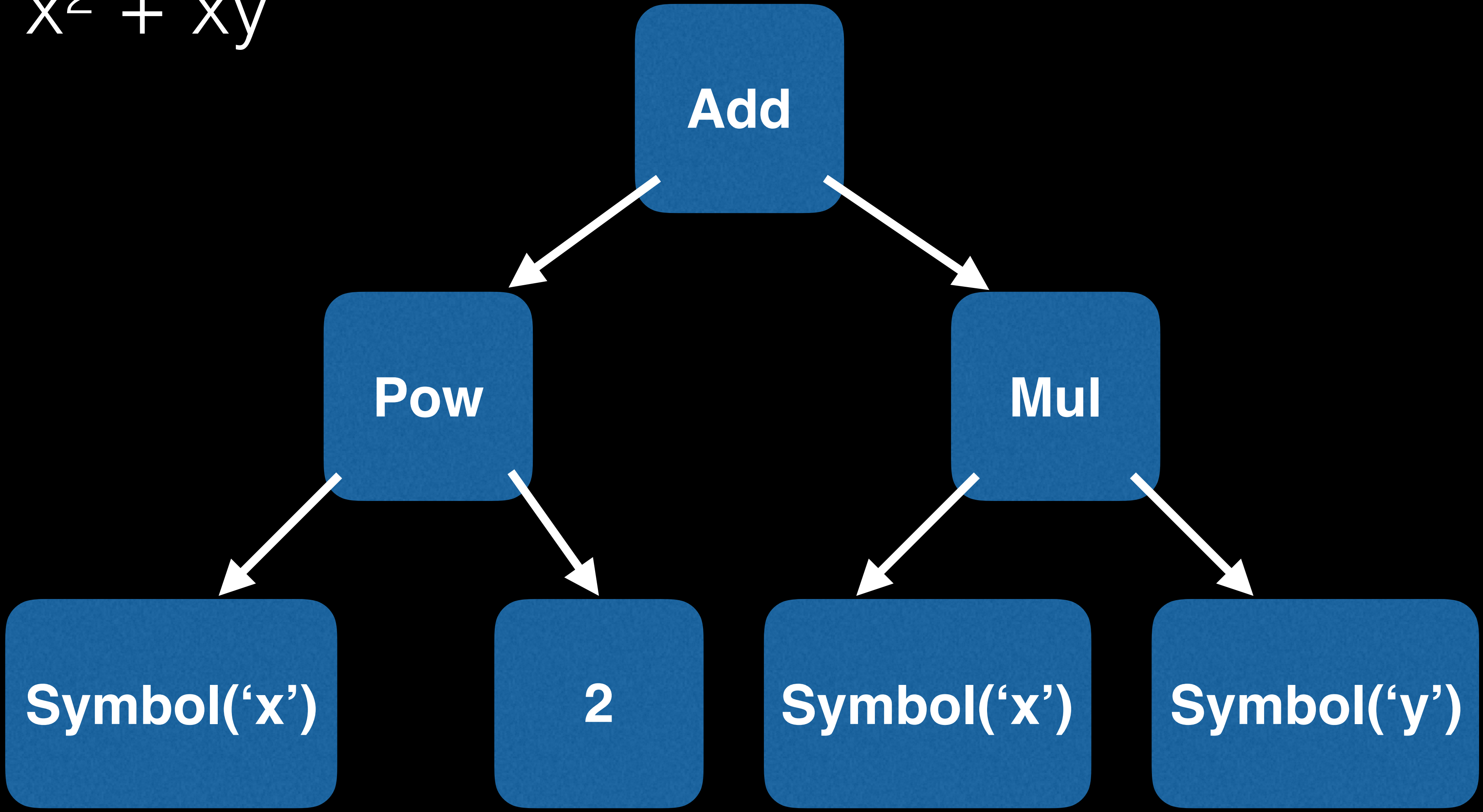


time = 0.00
1.0
0.5
0.0
−0.5
−1.0
−0.6
−0.4
−0.2
0.0
0.2
0.4

- More details on this problem are at https://github.com/pydy/pydy/blob/master/examples/npendulum/n-pendulum-control.ipynb

How does it work?

- SymPy expressions are stored as trees
- Example: $x^2 + xy$



How does it work?

- Every expression stores its children expression in `.args`

```
>>> (x**2 + x*y).args
(x**2, x*y)
>>> (x**2 + x*y).args[0].args
(x, 2)
```

How does it work?

```
class CCodePrinter(CodePrinter):
    def _print_Rational(self, expr):
        p, q = int(expr.p), int(expr.q)
        return '%d.0L/%d.0L' % (p, q)

    def _print_Exp1(self, expr):
        return "M_E"

    def _print_Pi(self, expr):
        return 'M_PI'
```

How does it work?

- Printer subclasses walk the expression tree and call methods corresponding to children (visitor pattern)

- Subclass `CodePrinter`, define methods for the expression types to code generate

- Easy to write your own code printers, or to extend existing code printers to do the things you need

Some other libraries that use SymPy code generation

- Chemreac
  - python library for solving chemical kinetics problems with possible diffusion and drift contributions
- SymPyBotics
  - Symbolic Framework for Modeling and Identification of Robot Dynamics

Takeaways

1. SymPy can deal with mathematical expressions in a high-level way. For example, it can take symbolic derivatives.

2. Using code generation avoids mistakes that come from translating mathematics into low level code.

3. It’s possible to deal with expressions that are otherwise too large to write by hand.

4. Some “mathematical” optimizations are possible, which a normal compiler would not be able to do.

- Mailing list: http://groups.google.com/group/sympy

- https://github.com/sympy/sympy

- @asmeurer, @SymPy on Twitter

- These slides are at https://github.com/asmeurer/SciPy-2016-Talk

- I’ll be at the sprints (and other SymPy developers)

Questions