PyDy Distribution Documentation

Release 0.2.1

PyDy Authors

February 07, 2015

Contents

This is the central page for all PyDy's Documentation.

CHAPTER 1

PyDy

PyDy, short for Python Dynamics, is a tool kit written in and accessed through the Python programming language that utilizes an array of scientific tools to study multibody dynamics. The goal is to have a modular framework and eventually a physics abstraction layer which utilizes a variety of backend that can provide the user with their desired workflow, including:

- Model specification
- Equation of motion generation
- Simulation
- Visualization
- Publication

We started by building the SymPy mechanics package which provides an API for building models and generating the symbolic equations of motion for complex multibody systems and have more recently developed two packages, pydy.codegen and pydy.viz, for simulation and visualization of the models. The remaining tools currently used in the PyDy workflow are popular scientific Python packages such as NumPy, SciPy, IPython, and matplotlib (i.e. the SciPy stack) which provide additional code for numerical analyses, simulation, and visualization.

1.1 Installation

The PyDy workflow has hard dependecies on these Python packages:

- Python >= 2.7
- setuptools

SciPy Stack

- SymPy >= 0.7.4.1
- NumPy >= 1.6.1
- SciPy >= 0.9.0
- IPython >= 0.13.0

It's best to install the SciPy Stack dependencies using the instructions provided on the SciPy website.

Once the dependencies are installed, the package can be installed from PyPi using:

```
$ easy_install pydy
```

or:

```
$ pip install pydy
```

For system wide installs you will need root permissions (perhaps prepend commands with `sudo`).

You can also grab the source and then install[1].

Using the zip download:

```
$ wget https://github.com/pydy/pydy/archive/master.zip
$ unzip pydy-master.zip
$ cd pydy-master
$ python setup.py install
```

Using Git:

```
$ git clone https://github.com/pydy/pydy.git
$ cd pydy
$ python setup.py install
```

1.1.1 Development Environment

Development Dependencies

Tests require nose:

- nose: 1.3.0

Isolated Virtual Environment Installation

The following installation assumes you have virtualenvwrapper and all the dependencies needed to build the packages:

```
$ mkvirtualenv pydy-dev
(pydy-dev)$ pip install numpy scipy cython nose theano sympy
(pydy-dev)$ pip install matplotlib # make sure to do this after numpy
(pydy-dev)$ git clone git@github.com:pydy/pydy.git
(pydy-dev)$ cd pydy
(pydy-dev)$ python setup.py develop
```

Run the tests:

```
(pydy-dev)$ nosetests
```

For the Javascript tests the Jasmine and blanket.js libraries are used. Both of these libraries are included in pydy-viz with the source. To run the Javascript tests, go to the javascript library directory:

```
$ cd pydy/viz/static/js
```

Then run a simple HTTP Server with Python (the server is required due to some cross browser issues with blanket.js):

```
$ python -m SimpleHTTPServer
```

Now visit http://localhost:8000/SpecRunner.html in a webgl compliant browser.

Run the benchmark to test the n-link pendulum problem.:

[1] Note that this is the latest development version. Specific releases can be found here: https://github.com/pydy/pydy/releases or by checking out a tag with Git.

```
(pydy-dev)$ python bin/benchmark_pydy_code_gen.py <max # of links> <# of time steps>
```

1.2 Usage

Simply import the modules and functions when in a Python interpreter:

```
>>> from sympy import symbols
>>> from sympy.physics import mechanics
>>> from pydy import codegen, viz
```

1.3 Documentation

The documentation is hosted at http://pydy-viz.readthedocs.org but you can also build them from source using the following instructions:

Requires:

- Sphinx
- numpydoc

```
pip install sphinx numpydoc
```

To build the HTML docs:

```
$ sphinx-build -b html docs/src docs/build
```

View:

```
$ firefox docs/build/index.html
```

1.4 Code Generation

This package provides code generation facilities for PyDy. For now, it generates functions that can evaluate the right hand side of the ordinary differential equations generated with sympy.physics.mechanics with three different backends: SymPy's lambdify, Theano, and Cython.

1.4.1 Optional Dependencies

To enable different code generation backends, you can install the various optional dependencies:

- Cython: >=0.15.1
- Theano: >=0.6.0

1.4.2 Usage

This is an example of a simple 1 degree of freedom system: a mass, spring, damper system under the influence of gravity and a force:

```
/ / / / / / / / /
-----------------
  |    |     |   | g
  \   | |    |   V
k /   --- c  |
  |    |     | x, v
 --------    V
 |  m   | -----
 --------
    | F
    V
```

Derive the system:

```
from sympy import symbols
import sympy.physics.mechanics as me

mass, stiffness, damping, gravity = symbols('m, k, c, g')

position, speed = me.dynamicsymbols('x v')
positiond = me.dynamicsymbols('x', 1)
force = me.dynamicsymbols('F')

ceiling = me.ReferenceFrame('N')

origin = me.Point('origin')
origin.set_vel(ceiling, 0)

center = origin.locatenew('center', position * ceiling.x)
center.set_vel(ceiling, speed * ceiling.x)

block = me.Particle('block', center, mass)

kinematic_equations = [speed - positiond]

force_magnitude = mass * gravity - stiffness * position - damping * speed + force
forces = [(center, force_magnitude * ceiling.x)]

particles = [block]

kane = me.KanesMethod(ceiling, q_ind=[position], u_ind=[speed],
                      kd_eqs=kinematic_equations)
kane.kanes_equations(forces, particles)
```

Store the expressions and symbols in sequences for the code generation:

```
mass_matrix = kane.mass_matrix_full
forcing_vector = kane.forcing_full
constants = (mass, stiffness, damping, gravity)
coordinates = (position,)
speeds = (speed,)
specified = (force,)
```

Now generate the function needed for numerical evaluation of the ODEs. The generator can use various back ends: `lambdify`, `theano`, or `cython`:

```
from pydy.codegen.code import generate_ode_function

evaluate_ode = generate_ode_function(mass_matrix, forcing_vector, constants,
```

```
                                        coordinates, speeds, specified,
                                        generator='lambdify')
```

Integrate the equations of motion under the influence of a specified sinusoidal force:

```
from numpy import array, linspace, sin
from scipy.integrate import odeint

x0 = array([0.1, -1.0])
args = {'constants': array([1.0, 1.0, 0.2, 9.8]),
        'specified': lambda x, t: sin(t)}
t = linspace(0.0, 10.0, 1000)

y = odeint(evaluate_ode, x0, t, args=(args,))
```

Plot the results:

```
import matplotlib.pyplot as plt

plt.plot(t, y)
plt.legend((str(position), str(speed)))
plt.show()
```

1.5 Visualization (viz)

Visualization of multibody systems generated with PyDy.

1.6 Related Packages

- https://github.com/cdsousa/sympybotics
- https://pypi.python.org/pypi/Hamilton
- https://pypi.python.org/pypi/arboris
- https://pypi.python.org/pypi/PyODE
- https://pypi.python.org/pypi/odeViz
- https://pypi.python.org/pypi/ARS
- https://pypi.python.org/pypi/pymunk

1.7 Release Notes

1.7.1 0.2.1

- Unbundled unecessary files from tar ball.

1.7.2 0.2.0

- Merged pydy_viz, pydy_code_gen, and pydy_examples into the source tree.
- Added a method to output "static" visualizations from a Scene object.
- Dropped the matplotlib dependency and now only three.js colors are valid.
- Added joint torques to the n_pendulum model.
- Added basic examples for codegen and viz.
- Graceful fail if theano or cython are not present.
- Shapes can now use sympy symbols for geometric dimensions.

1.7.3 codegen package

codegen API

1.7.4 viz package

Introduction

The viz package in pydy is designed to facilitate browser based animations for PyDy framework.

Typically the plugin is used to generate animations for multibody systems. The systems are defined with sympy.physics.mechanics, solved numerically with the codegen package and scipy, and then visualized with this package. But the required data for the animations can theorectically be generated by other methods and passed into a Scene object.

The frontend is based on three.js, a popular interface to the WebGraphics Library (WegGL). The package provides a Python wrapper for some basic functionality for Three.js i.e Geometries, Lights, Cameras etc.

API

All the module specific docs have some test cases, which will prove helpful in understanding the usage of the particular module.

Python Modules Reference

Shapes

Shape

Cube

Cylinder

Cone

Sphere

Circle

Plane

Tetrahedron

Octahedron

Icosahedron

Torus

TorusKnot

Tube

Mesh

VisualizationFrame

Cameras

Perspective Camera

Orthographic Camera

Lights

PointLight

Scene

JavaScript functions Reference

Canvas Canvas is the base class for handling all the animation and visualization generation.

canvas/initialize.js

Constructor: This function acts as a class constructor for Canvas class It takes the JSON Object variable as the argument, which contains all the data in the JSON format. It binds onClick methods of certain Divs on the frontend with some Canvas.prototype functions.

Canvas.prototype.initialize This prototype function initializes the starting canvas, on which all the visualizations are drawn.

It adds following to the canvas:

- A Primary Camera
- Primary Trackball Controls
- A Primary Light
- Axes
- Grid
- A Div for displaying total number of frames.
- A Div for displaying the current frame animation is running on.

canvas/addObjects.js

Canvas.prototype.addControls This prototype function initializes the Primary Controls, which were defined in Canvas.prototype.initialize function.

It generates a controlsID, which contains the return value of requestAnimationFrame, and can be used to call cancelAnimationFrame, for stopping mouse controlled animation.

Canvas.prototype.resetControls This prototype function simply calls the controls reset method for the Primary Controls(canvas.prototype.primaryControls).

Canvas.prototype.addCameras This prototype function parses the JSON Object for cameras and adds them to the scene. All the cameras are stored in a Canvas.cameras object, which is an instance of THREE.Object3D();

Canvas.prototype.addLights This prototype function parses the JSON Object for lights and adds them to the scene. All the lights are stored in a Canvas.lights object, which is an instance of THREE.Object3D();

Canvas.prototype.addFrames This prototype function parses the JSON Object for frames and adds them to the scene. All the frames are stored in a Canvas.frames object, which is an instance of THREE.Object3D();

canvas/animate.js

Canvas.prototype.startAnimation This prototype function kick starts the animation. It iterates over the frames and apply transformation matrices from Simulation Matrix of that frame, iteratively. By default animation is done for a single loop, which can be changed to looped by the check button from the UI.

Canvas.prototype.pauseAnimation This prototype function pauses the animation, but retains the current animation frame.

Canvas.prototype.stopAnimation This prototype function stops the animation, and resets current animation frame to 0.

1.8 Questions

If you have any question about installation, or any general question, feel free to visit the IRC channel at irc.freenode.net, channel #pydy. In addition, our mailing list is an excellent source of community support.

If you think there's a bug or you would like to request a feature, please open an issue.

1.9 Indices and tables

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