js_pydy_mass_spring_damper

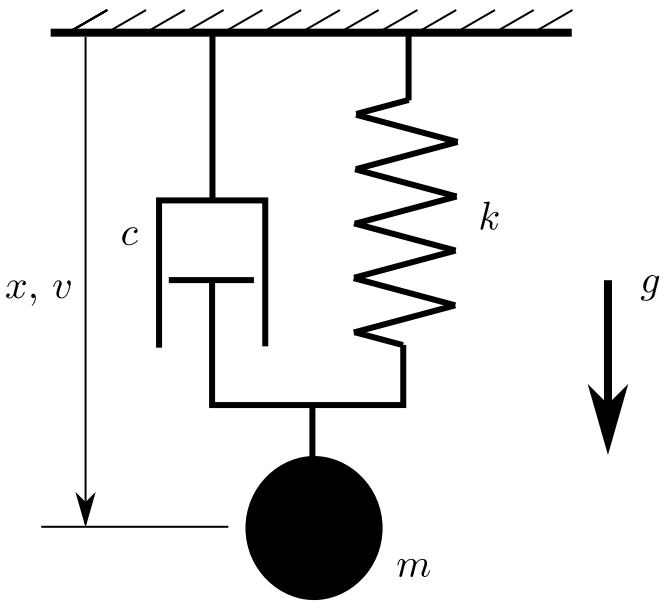
June 26, 2023

1 pydy

pydy simulates physical systems.
Example from pydy/../mass_spring_damper.
documentation examples sources
Here we will derive the equations of motion for the classic mass-spring-damper system under the influence of gravity. The following figure gives a pictorial description of the problem.

[1]:
```
from IPython.display import SVG
SVG(filename='pydy.svg')
```

[1]:



Start by loading in the core functionality of both SymPy and Mechanics.

[2]:
```
import sympy as sym
import sympy.physics.mechanics as me
```

We can make use of the pretty printing of our results by loading SymPy's printing extension, in particular we will use the vector printing which is nice for mechanics objects.

[3]:
```
from sympy.physics.vector import init_vprinting
init_vprinting(use_latex='mathjax')
```

We'll start by defining the variables we will need for this problem: - $x(t)$: distance of the particle from the ceiling - $v(t)$: speed of the particle - m: mass of the particle - c: damping coefficient of the damper - k: stiffness of the spring - g: acceleration due to gravity - t: time

[4]:
```
x, v = me.dynamicsymbols('x v')
```

[5]:
```
m, c, k, g, t = sym.symbols('m c k g t')
```

Now, we define a Newtonian reference frame that represents the ceiling which the particle is attached to, C.

[6]:
```
ceiling = me.ReferenceFrame('C')
```

We will need two points, one to represent the original position of the particle which stays fixed in the ceiling frame, O, and the second one, P which is aligned with the particle as it moves.

[7]:
```
O = me.Point('O')
P = me.Point('P')
```

The velocity of point O in the ceiling is zero.

[8]:
```
O.set_vel(ceiling, 0)
```

Point P can move downward in the y direction and its velocity is specified as v in the downward direction.

[9]:
```
P.set_pos(O, x * ceiling.x)
P.set_vel(ceiling, v * ceiling.x)
P.vel(ceiling)
```

[9]:

$$v\hat{\mathbf{c}}_\mathbf{x}$$

There are three forces acting on the particle. Those due to the acceleration of gravity, the damper, and the spring.

[10]:
```
damping = -c * P.vel(ceiling)
stiffness = -k * P.pos_from(O)
gravity = m * g * ceiling.x
forces = damping + stiffness + gravity
forces
```

[10]:

$$(-cv + gm - kx)\hat{\mathbf{c}}_\mathbf{x}$$

Now we can use Newton's second law, $0 = F - ma$, to form the equation of motion of the system.

[11]:
```
zero = me.dot(forces - m * P.acc(ceiling), ceiling.x)
zero
```

[11]:

$$-cv + gm - kx - m\dot{v}$$

We can then form the first order equations of motion by solving for $\frac{dv}{dt}$ and introducing the kinematical differential equation, $v = \frac{dx}{dt}$.

[12]:
```
dv_by_dt = sym.solve(zero, v.diff(t))[0]
dx_by_dt = v
dv_by_dt, dx_by_dt
```

[12]:

$$\left(\frac{1}{m}\left(-cv + gm - kx\right), \quad v\right)$$

Forming the equations of motion can also be done with the automated methods available in the Mechanics package: `LagrangesMethod` and `KanesMethod`. Here we will make use of Kane's method to find the same equations of motion that we found manually above. First, define a particle that represents the mass attached to the damper and spring.

[13]:
```
mass = me.Particle('mass', P, m)
```

Now we can construct a `KanesMethod` object by passing in the generalized coordinate, x, the generalized speed, v, and the kinematical differential equation which relates the two, $0 = v - \frac{dx}{dt}$.

[14]:
```
kane = me.KanesMethod(ceiling, q_ind=[x], u_ind=[v], kd_eqs=[v - x.diff(t)])
```

Now Kane's equations can be computed, and we can obtain F_r and F_r^*.

[15]:
```
fr, frstar = kane.kanes_equations([(P, forces)], [mass])
fr, frstar
```

[15]:

$$\left(\left[-cv + gm - kx\right], \quad \left[-m\dot{v}\right]\right)$$

The equations are also available in the form $M\frac{d}{dt}[q, u]^T = f(q, u)$ and we can extract the mass matrix, M, and the forcing functions, f.

[16]:
```
M = kane.mass_matrix_full
f = kane.forcing_full
M, f
```

[16]:

$$\left(\begin{bmatrix} 1 & 0 \\ 0 & m \end{bmatrix}, \quad \begin{bmatrix} v \\ -cv + gm - kx \end{bmatrix}\right)$$

Finally, we can form the first order differential equations of motion $\frac{d}{dt}[q, u]^T = M^{-1}f(\dot{u}, u, q)$, which is the same as previously found.

[17]:
```
M.inv() * f
```

[17]:

$$\begin{bmatrix} v \\ \frac{1}{m}\left(-cv + gm - kx\right) \end{bmatrix}$$

2 Simulating the system

Now that we have defined the mass-spring-damper system, we are going to simulate it.
PyDy's `System` is a wrapper that holds the Kanes object to integrate the equations of motion using numerical values of constants.

[18]:
```
from pydy.system import System
```

[19]:
```
sys = System(kane)
```

Now, we specify the numerical values of the constants and the initial values of states in the form of a dict.

[20]:
```
sys.constants = {m:10.0, g:9.8, c:5.0, k:10.0}
sys.initial_conditions = {x:0.0, v:0.0}
```

We must generate a time vector over which the integration will be carried out. NumPy's `linspace` is often useful for this.

[21]:
```
from numpy import linspace
sys.times = linspace(0.0, 10.0, 100)
```

The trajectory of the states over time can be found by calling the `.integrate()` method.

[22]:
```
x_trajectory = sys.integrate()
```

3 Visualizing the System

PyDy has a native module `pydy.viz` which is used to visualize a System in an interactive 3D GUI.

[23]:
```
from pydy.viz import *
```

For visualizing the system, we need to create shapes for the objects we wish to visualize, and map each of them to a `VisualizationFrame`, which holds the position and orientation of the object. First create a sphere to represent the bob and attach it to the point P and the ceiling reference frame (the sphere does not rotate with respect to the ceiling).

[24]:
```
bob = Sphere(2.0, color="red", material="metal")
bob_vframe = VisualizationFrame(ceiling, P, bob)
```

Now create a circular disc that represents the ceiling and fix it to the ceiling reference frame. The circle's default axis is aligned with its local y axis, so we need to attach it to a rotated ceiling reference frame if we want the circle's axis to align with the $\hat{c}_x$ unit vector.

[25]:
```
ceiling_circle = Circle(radius=10, color="white", material="metal")
from numpy import pi
rotated = ceiling.orientnew("C_R", 'Axis', [pi / 2, ceiling.z])
ceiling_vframe = VisualizationFrame(rotated, O, ceiling_circle)
```

Now we initialize a Scene. A Scene contains all the information required to visualize a `System` onto a canvas. It takes a ReferenceFrame and Point as arguments.

[26]:
```
scene = Scene(ceiling, O, system=sys)
```

We provide the VisualizationFrames, which we want to visualize as a list to scene.

[27]:
```
scene.visualization_frames = [bob_vframe, ceiling_vframe]
```

The default camera of Scene has the z axis of the base frame pointing out of the screen, and the y axis pointing up. We want the x axis to point downwards, so we supply a new camera that will achieve this.

[28]:
```
camera_frame = ceiling.orientnew('Camera Frame','Axis', [pi / 2, ceiling.z])
camera_point = O.locatenew('Camera Location', 100 * camera_frame.z)
primary_camera = PerspectiveCamera(camera_frame, camera_point)
scene.cameras = [primary_camera]
```

Now, we call the display method.

[29]:
```
scene.display_ipython()
```

[30]:
```
%load_ext version_information
```

[31]:
```
%version_information pydy, numpy, scipy
```

[31]:

Software	Version
Python	3.5.0 64bit [MSC v.1900 64 bit (AMD64)]
IPython	4.2.0
OS	Windows 10.0.10586
pydy	0.3.1
numpy	1.11.1rc1
scipy	0.17.1
Tue Jun 14 17:29:07 2016 Paris, Madrid (heure d?été)	

[32]:
```

```