rocketPy Documentation

Release 0.1.4

Devansh Ramgopal Agrawal

Apr 17, 2020

CONTENTS:

CHAPTER

ONE

ROCKETPY

Rocket design, simulation and analysis using Python!

- Free software: MIT license
- Documentation: https://rocketPy.readthedocs.io.

1.1 Features

- TODO

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install rocketPy, run this command in your terminal:

```
$ pip install rocketPy
```

This is the preferred method to install rocketPy, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for rocketPy can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/dev10110/rocketPy
```

Or download the tarball:

```
$ curl -OJL https://github.com/dev10110/rocketPy/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

To use rocketPy in a project:

```
import rocketPy
```

Simple usage of rocketPy is seen in the simple example file:

```

"""This example file demonstrates the construction of a simple rocket.
The corresponding .ork file is the OpenRocket implementation of the same rocket for comparison.
"""
import numpy as np
import matplotlib.pyplot as plt

import rocketPy as rp
from rocketPy import ureg


## First, create a rocket.
r = rp.Rocket(name='Simple Rocket')

## create a nose cone
nc = rp.NoseCone(name='Nose Cone', diameter=6*ureg.inch, fineness=3, material=rp.materials.PLA())
# assign to rocket
r.set_nose_cone(nc)

## create a BodyTube
bt = rp.BodyTube(name = 'Body Tube', diameter=6*ureg.inch, length=48*ureg.inch, wall_thickness=2*ureg.mm, material=rp.materials.Phenolic())
# define its location
bt.set_position(after=nc)
# assign to rocket
r.set_body_tube(bt)

# create a boat tail
boat_tail = rp.Transition(name='Boat Tail', fore_dia=6*ureg.inch, aft_dia=4*ureg.inch, length=4*ureg.inch, material=rp.materials.Phenolic())
# define its location
boat_tail.set_position(after=bt)
# assign to rocket
r.set_boat_tail(boat_tail)

## create the fins
```

(continues on next page)

(continued from previous page)

```
fins = rp.FinSet(name='Fins', n=3, span=6*ureg.inch, root_chord=12*ureg.inch, tip_
↪chord=6*ureg.inch, mid_sweep=10*ureg.degree, tube_dia=6*ureg.inch, ␣
↪thickness=2*ureg.mm, material=rp.materials.Aluminium())
# define its location
fins.set_position(end_of=bt, offset=-fins.root_chord)
#  assign to rocket
r.set_fins(fins)

# plot the entire rocket
fig = plt.figure()
ax = plt.gca()
r.plot(ax, unit=ureg.inch)
plt.draw()

# describe the rocket
r.describe(describe_components=True)

plt.show()
```

which should provide an output of

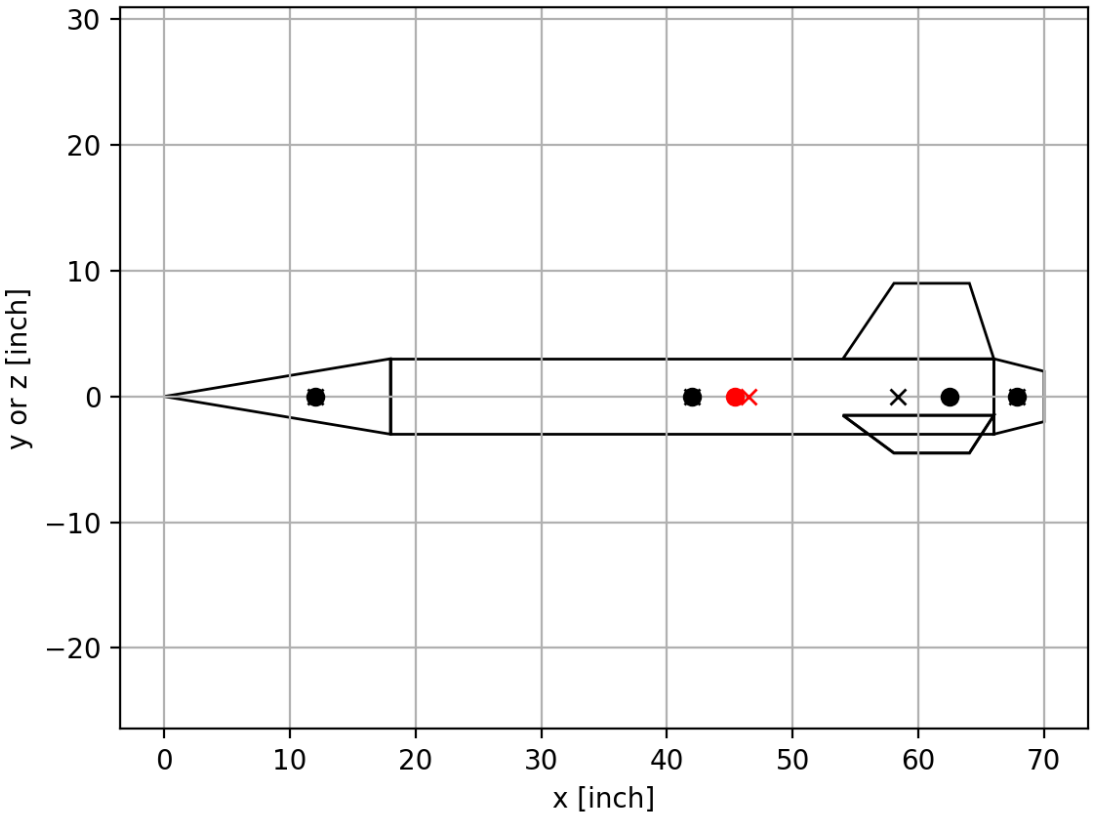


Fig. 1: Rocket schematic. x mark center of pressures, o mark center of masses. The red ones correspond to the full rocket.

```
2020-01-23 21:03:59.283 python[61418:17799736] [QL] Can't get plugin bundle info at␣
↪file:///Users/Devansh/Library/Application%20Support/Autodesk/webdeploy/production/
↪078c0152f608cb87272eeb7be2226a5f77176092/Autodesk%20Fusion%20360.app/Contents/
↪Library/QuickLook/NQLGenerator.qlgenerator
objc[61418]: Class FIFinderSyncExtensionHost is implemented in both /System/Library/
↪PrivateFrameworks/FinderKit.framework/Versions/A/FinderKit (0x7fffad21b3d8) and /
↪System/Library/PrivateFrameworks/FileProvider.framework/OverrideBundles/
↪FinderSyncCollaborationFileProviderOverride.bundle/Contents/MacOS/
↪FinderSyncCollaborationFileProviderOverride (0x119d1ff50). One of the two will be␣
↪used. Which one is undefined.
Rocket: Simple Rocket

Rocket Details
+--------------+--------------+-------------------+
|  Parameter   |    Value     |       Notes       |
+--------------+--------------+-------------------+
|  Total Mass  |  1.9803 kg   |                   |
| Total Length |  70.0000 in  |                   |
|     X_CG     |  1.152304 m  |                   |
|     X_CP     |   1.1799 m   | At default values |
|      CD      |    ERROR     | At default values |
|     CNa      | 9.0907 / rad | At default values |
+--------------+--------------+-------------------+

Component Details
+-----------+------------+------------+---------+-----------------+--------------+
| Component |    Type    |  Material  |   Mass  | Mass Fraction % |     CNa      |
+-----------+------------+------------+---------+-----------------+--------------+
| Nose Cone |  NoseCone  |    PLA     | 0.23 kg |      11.61      | 2.097 / rad  |
| Body Tube |  BodyTube  |  Phenolic  | 1.11 kg |      56.00      | 0.000 / rad  |
| Boat Tail | Transition |  Phenolic  | 0.08 kg |       3.89      | -1.165 / rad |
|    Fins   |   FinSet   | Al-6061-T6 | 0.56 kg |      28.50      | 8.159 / rad  |
+-----------+------------+------------+---------+-----------------+--------------+

Describing all components in full:

Nose Cone (type: NoseCone)
+----------------+-------------------+--------------------+
|   Parameter    |     Value (SI)    |       Value        |
+----------------+-------------------+--------------------+
|     shape      |      Conical      |                    |
|     x_ref      |      0.000 m      |      0.000 m       |
|    diameter    |      0.152 m      |      6.000 in      |
|     length     |      0.457 m      |     18.000 in      |
| wall_thickness |      0.002 m      |      2.000 mm      |
|    material    |  PLA: (Material)) |                    |
|      name      |     Nose Cone     |                    |
|      mass      |      0.230 kg     |      0.230 kg      |
|      I_xx      | 0.001 kg * m ** 2 | 1.034 in ** 2 * kg |
|      I_yy      | 0.003 kg * m ** 2 | 4.137 in ** 2 * kg |
|      I_zz      | 0.003 kg * m ** 2 | 4.137 in ** 2 * kg |
|     y_ref      |      0.000 m      |      0.000 m       |
|     z_ref      |      0.000 m      |      0.000 m       |
|     A_ref      |    0.018 m ** 2   |   28.274 in ** 2   |
+----------------+-------------------+--------------------+
Body Tube (type: BodyTube)
+----------------+-----------------------+----------------------+
```

(continues on next page)

(continued from previous page)

```
|   Parameter    |       Value (SI)       |        Value         |
+----------------+------------------------+----------------------+
|     A_ref      |      0.018 m ** 2      |    28.274 in ** 2    |
|     d_ref      |        0.152 m         |       6.000 in       |
|    diameter    |        0.152 m         |       6.000 in       |
|     length     |        1.219 m         |      48.000 in       |
| wall_thickness |        0.002 m         |       2.000 mm       |
|    material    | Phenolic: (Material)) |                      |
|      name      |       Body Tube        |                      |
|      mass      |        1.109 kg        |       1.109 kg       |
|      I_xx      |   0.006 kg * m ** 2    |  9.982 in ** 2 * kg  |
|      I_yy      |   0.137 kg * m ** 2    | 212.944 in ** 2 * kg |
|      I_zz      |   0.137 kg * m ** 2    | 212.944 in ** 2 * kg |
|     x_ref      |        0.457 m         |       0.457 m        |
|     y_ref      |        0.000 m         |       0.000 m        |
|     z_ref      |        0.000 m         |       0.000 m        |
+----------------+------------------------+----------------------+
Boat Tail (type: Transition)
+----------------+------------------------+--------------------+
|   Parameter    |       Value (SI)       |       Value        |
+----------------+------------------------+--------------------+
|     A_ref      |      0.018 m ** 2      |   28.274 in ** 2   |
|    fore_dia    |        0.152 m         |      6.000 in      |
|    aft_dia     |        0.102 m         |      4.000 in      |
|     d_ref      |        0.152 m         |      6.000 in      |
|     length     |        0.102 m         |      4.000 in      |
| wall_thickness |        0.002 m         |      2.000 mm      |
|    material    | Phenolic: (Material)) |                    |
|      name      |       Boat Tail        |                    |
|      mass      |        0.077 kg        |      0.077 kg      |
|      I_xx      |   0.000 kg * m ** 2    | 0.501 in ** 2 * kg |
|      I_yy      |   0.000 kg * m ** 2    | 0.101 in ** 2 * kg |
|      I_zz      |   0.000 kg * m ** 2    | 0.101 in ** 2 * kg |
|     x_ref      |        1.676 m         |      1.676 m       |
|     y_ref      |        0.000 m         |      0.000 m       |
|     z_ref      |        0.000 m         |      0.000 m       |
+----------------+------------------------+--------------------+
Fins (type: FinSet)
+----------------+-------------------------+--------------------+
|   Parameter    |        Value (SI)       |        Value       |
+----------------+-------------------------+--------------------+
|     A_ref      |       0.018 m ** 2      |   28.274 in ** 2   |
|     d_ref      |         0.152 m         |      6.000 in      |
|       n        |            3            |                    |
|      span      |         0.152 m         |      6.000 in      |
|   root_chord   |         0.305 m         |     12.000 in      |
|   tip_chord    |         0.152 m         |      6.000 in      |
|   mid_sweep    |        0.175 rad        |     10.000 deg     |
| mid_chord_span |         0.155 m         |      6.093 in      |
|    tube_dia    |         0.152 m         |      6.000 in      |
|     length     |         0.000 m         |      0.000 m       |
|   thickness    |         0.002 m         |      2.000 mm      |
|  exposed_area  |       0.035 m ** 2      |   54.000 in ** 2   |
| planform_area  |       0.058 m ** 2      |   90.000 in ** 2   |
|    material    | Al-6061-T6: (Material)) |                    |
|      name      |           Fins          |                    |
|      mass      |         0.564 kg        |      0.564 kg      |
```

(continues on next page)

(continued from previous page)

```
|      I_xx      |    0.013 kg * m ** 2    | 19.754 in ** 2 * kg |
|      I_yy      |    0.003 kg * m ** 2    |  4.284 in ** 2 * kg |
|      I_zz      |    0.003 kg * m ** 2    |  4.284 in ** 2 * kg |
|     x_ref      |         1.372 m         |       1.372 m       |
|     y_ref      |         0.000 m         |       0.000 m       |
|     z_ref      |         0.000 m         |       0.000 m       |
+----------------+-------------------------+---------------------+
```

CHAPTER

FOUR

EXAMPLE USAGE OF SIMULATION

This file demonstrates the use of rocketPy's simulation environment.

First we import some useful packages

[1]:
```
import numpy as np
import scipy as sp
import scipy.integrate as spint

import matplotlib.pyplot as plt


# and from rocket py we need simulation and solutions
from rocketPy.simulation import Simulation as Sim
from rocketPy.solution import Solution
```

We need a dynamic object to simulate, and we create this using a small class.

This rocket is a one dimensional object. We define a few useful properties at creation, but then the functions take over.

For any dynamic object you need the function `dynamics` This function takes in a current time, state, and stage number and returns the rate of change of the state

In addition to this, you can (optionally) define some staging functions. These staging functions define how the dynamic object can change between stages.

For this example, a simple rocket is modelled. It will thrust upwards, coast, and then descend under a parachute. For simplicity, we only consider the rocket as a one dimensional object. The rocket will return to the ground using dual deployment, ie both a drogue chute and a main chute, each triggered at a different time.

The drogue chute is deployed 7 seconds after apogee, and (to demonstrate the usage) jumps the position up by 1000m when it happens. This is a very powerful tool, since when staging a rocket you can imagine the mass of rocket to decrease by a step change, which would be difficult to model using other methods.

The main chute will deploy at an altitude of 2500 m.

Each of the staging functions have additional properties we need to specify.

- terminal (boolean): Should the simulation run stop when this event triggers
- direction (1 or -1): which way must the 0-crossing be for the trigger to occur
- etc

[2]:
```
class VerySimpleRocket():

    def __init__(self):
        self.m = 40
```

(continues on next page)

(continued from previous page)

```
        self.T = 4000
        self.g = 9.81
        self.y0 = np.array([0., 0.])
        self.rhoCDA1 = 0.05
        self.rhoCDA2 = 0.1

        self.stage_list = [0,1,2]
        self.staging_functions = [self.staging_deploy_drogue,self.staging_deploy_main,
↪ self.staging_landing]
        self.nominal_stages = [0,1,2] # defines this as a nominal flight

    def staging_deploy_drogue(self,t,y,stage=0):
        return y[1]
    staging_deploy_drogue.terminal = False
    staging_deploy_drogue.direction=-1
    staging_deploy_drogue.trigger_if_stage_in =[0]
    staging_deploy_drogue.possible_next_stages = [1,2]
    staging_deploy_drogue.nominal_next_stage = 1
    staging_deploy_drogue.t_offset = 7 #stages 7 seconds after the apogee is detected
    staging_deploy_drogue.modify_state = lambda self, state: self.modify_state_drogue_
↪deployed(state)

    def staging_deploy_main(self, t, y, stage=0):
        return y[0]-2500
    staging_deploy_main.terminal = False
    staging_deploy_main.direction = -1
    staging_deploy_main.trigger_if_stage_in =[0,1]
    staging_deploy_main.possible_next_stages = [ 2]
    staging_deploy_main.nominal_next_stage = 2
    staging_deploy_main.t_offset = 0
    staging_deploy_main.modify_state = None

    def staging_landing(self, t, y, stage=0):
        return y[0]
    staging_landing.terminal = True
    staging_landing.direction = -1
    staging_landing.trigger_if_stage_in =[0,1,2]
    staging_landing.possible_next_stages = []
    staging_landing.nominal_next_stage = None
    staging_landing.t_offset = 0
    staging_landing.modify_state = None

    def modify_state_drogue_deployed(self, state):

        # this function replaces the state when the corresponding branch is explored

        state[0] += 1000
        return state


    def dynamics(self, t, y, stage=0):

        if stage == 0:
            if t<4:
                return np.array([y[1], self.T/self.m - self.g])
            else:
                return np.array([y[1], -self.g])
```

(continues on next page)

(continued from previous page)

```
        elif stage == 1:
            return np.array([y[1], -0.5*self.rhoCDA1*y[1]*abs(y[1])/self.m - self.g])

        elif stage == 2:
            return np.array([y[1], -0.5*self.rhoCDA2*y[1]*abs(y[1])/self.m - self.g])

        else:
            raise ValueError
```

Instantiate the rocket and the sim

[6]:
```
r = VerySimpleRocket()
```

[7]:
```
s = Simulation(r)
```

Do a very simple sim, starting at stage 0.

[28]:
```
sol=s.solve([0,600], r.y0, 0, user_events=r.staging_functions)
```

The result object (from scipy.solve_ivp) is stored in sol.sols, as a list

[29]:
```
sol
```

[29]:
```
Solution:
[
 Stages: [0]
 ODEresults: [  message: 'A termination event occurred.'
     nfev: 98
     njev: 0
      nlu: 0
      sol: <scipy.integrate._ivp.common.OdeSolution object at 0x11bde4278>
   status: 1
  success: True
        t: array([0.00000000e+00, 1.00000000e-04, 1.10000000e-03, 1.11000000e-02,
       1.11100000e-01, 1.11110000e+00, 2.79214387e+00, 3.62727257e+00,
       4.46240128e+00, 5.31679544e+00, 1.38607371e+01, 8.10612092e+01])
 t_events: [array([41.57554744]), array([73.97050835]), array([81.06120925])]
        y: array([[ 0.00000000e+00,  4.50950000e-07,  5.45649500e-05,
         5.55615495e-03,  5.56617055e-01,  5.56717261e+01,
         3.51563658e+02,  5.93319709e+02,  8.91394822e+02,
         1.19888202e+03,  3.87988823e+03,  2.27373675e-12],
       [ 0.00000000e+00,  9.01900000e-03,  9.92090000e-02,
         1.00110900e+00,  1.00201090e+01,  1.00210109e+02,
         2.51823455e+02,  3.27143713e+02,  3.64079964e+02,
         3.55698357e+02,  2.71882290e+02, -3.87354342e+02]])
 y_events: [array([[7647.47127891,    0.        ]]), array([[2500.        , -317.
↪79456649]]), array([[ 2.27373675e-12, -3.87354342e+02]])]]
 ]
```

Now simulate the nominal trajectory

[30]:
```
nominal_sol = s.nominal_solve([0,6000], r.y0, 0)
```

You can ask for the solution at some time, for instance at

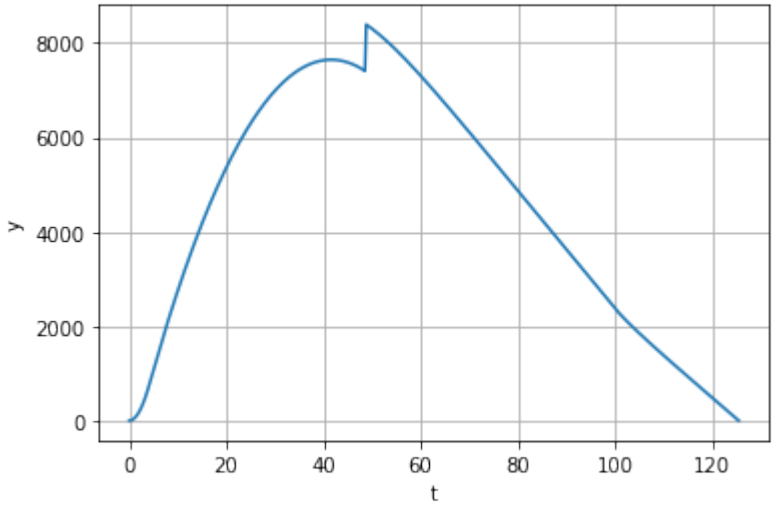
$$t = 5$$

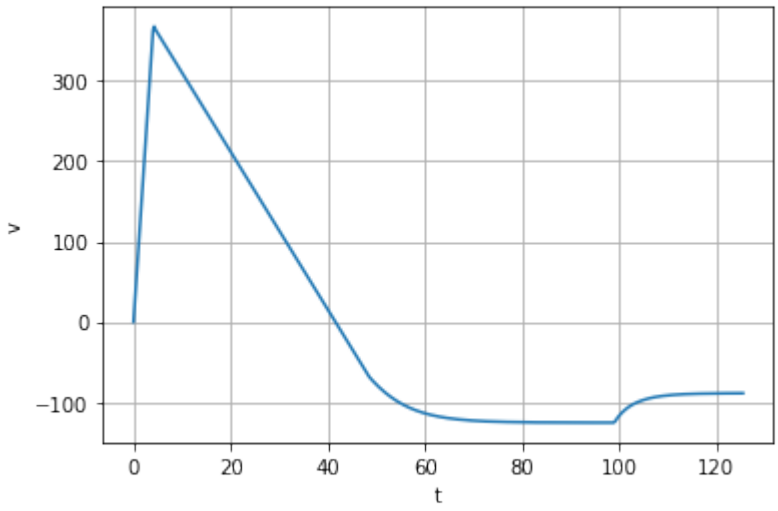
[32]:
```
nominal_sol.sol(5)
```

[32]:
```
array([1085.70613837,  358.80612043])
```

so its 1085 m up, with a speed of 358 m/s. Or you can plot it

[37]:
```
# helper function to get the bounds of the simulation
t_range = np.linspace(nominal_sol.t_min(), nominal_sol.t_max(), 500)

plt.plot(t_range, nominal_sol.sol(t_range)[0])
plt.xlabel('t')
plt.ylabel('y')
plt.grid()

plt.figure()
plt.plot(t_range, nominal_sol.sol(t_range)[1])
plt.xlabel('t')
plt.ylabel('v')
plt.grid()
```





The real magic is in simulating all possible outcomes

[16]:
```
full_sol = s.full_solve([0,6000], r.y0, 0)
```

full solve gives a list of all the possible simulations

[44]:
```
full_sol
```

[44]:
```
[Solution:
 [
  Stages: [0]
  ODEresults: [  message: 'A termination event occurred.'
      nfev: 98
      njev: 0
       nlu: 0
       sol: <scipy.integrate._ivp.common.OdeSolution object at 0x11b3fefd0>
    status: 1
   success: True
         t: array([0.00000000e+00, 1.00000000e-04, 1.10000000e-03, 1.11000000e-02,
       1.11100000e-01, 1.11110000e+00, 2.79214387e+00, 3.62727257e+00,
       4.46240128e+00, 5.31679544e+00, 1.38607371e+01, 8.10612092e+01])
  t_events: [array([41.57554744]), array([73.97050835]), array([81.06120925])]
         y: array([[ 0.00000000e+00,  4.50950000e-07,  5.45649500e-05,
          5.55615495e-03,  5.56617055e-01,  5.56717261e+01,
          3.51563658e+02,  5.93319709e+02,  8.91394822e+02,
          1.19888202e+03,  3.87988823e+03,  2.27373675e-12],
        [ 0.00000000e+00,  9.01900000e-03,  9.92090000e-02,
          1.00110900e+00,  1.00201090e+01,  1.00210109e+02,
          2.51823455e+02,  3.27143713e+02,  3.64079964e+02,
          3.55698357e+02,  2.71882290e+02, -3.87354342e+02]])
  y_events: [array([[7647.47127891,    0.        ]]), array([[2500.        , -317.
↪79456649]]), array([[ 2.27373675e-12, -3.87354342e+02]])]]
  ], Solution:
 [
  Stages: [0, 1]
  ODEresults: [  message: 'A termination event occurred.'
      nfev: 98
      njev: 0
       nlu: 0
       sol: <scipy.integrate._ivp.common.OdeSolution object at 0x11b3fefd0>
    status: 1
   success: True
         t: array([0.00000000e+00, 1.00000000e-04, 1.10000000e-03, 1.11000000e-02,
       1.11100000e-01, 1.11110000e+00, 2.79214387e+00, 3.62727257e+00,
       4.46240128e+00, 5.31679544e+00, 1.38607371e+01, 8.10612092e+01])
  t_events: [array([41.57554744]), array([73.97050835]), array([81.06120925])]
         y: array([[ 0.00000000e+00,  4.50950000e-07,  5.45649500e-05,
          5.55615495e-03,  5.56617055e-01,  5.56717261e+01,
          3.51563658e+02,  5.93319709e+02,  8.91394822e+02,
          1.19888202e+03,  3.87988823e+03,  2.27373675e-12],
        [ 0.00000000e+00,  9.01900000e-03,  9.92090000e-02,
          1.00110900e+00,  1.00201090e+01,  1.00210109e+02,
          2.51823455e+02,  3.27143713e+02,  3.64079964e+02,
          3.55698357e+02,  2.71882290e+02, -3.87354342e+02]])
  y_events: [array([[7647.47127891,    0.        ]]), array([[2500.        , -317.
↪79456649]]), array([[ 2.27373675e-12, -3.87354342e+02]])],   message: 'A␣
↪termination event occurred.'
      nfev: 56
```

(continues on next page)

(continued from previous page)

```
      njev: 0
       nlu: 0
       sol: <scipy.integrate._ivp.common.OdeSolution object at 0x11b3ef128>
    status: 1
   success: True
         t: array([ 48.57554744,  48.74532045,  50.44305052,  58.80397919,
         67.16490786,  76.66698838,  88.28878273, 103.37969057,
        118.94418694])
  t_events: [array([], dtype=float64), array([98.98862675]), array([118.94418694])]
         y: array([[ 8.40712628e+03,  8.39536954e+03,  8.26755846e+03,
          7.44367144e+03,  6.46209201e+03,  5.29093782e+03,
          3.83965685e+03,  1.94984670e+03, -6.82121026e-13],
        [-6.86700000e+01, -6.98266172e+01, -8.04575605e+01,
         -1.11386543e+02, -1.21372123e+02, -1.24375610e+02,
         -1.25118873e+02, -1.25249980e+02, -1.25270546e+02]])
  y_events: [array([], dtype=float64), array([[2500.        , -125.25361785]]),␣
↪array([[-6.82121026e-13, -1.25270546e+02]])]]
  ], Solution:
[
  Stages: [0, 1, 2]
  ODEresults: [  message: 'A termination event occurred.'
      nfev: 98
      njev: 0
       nlu: 0
       sol: <scipy.integrate._ivp.common.OdeSolution object at 0x11b3fefd0>
    status: 1
   success: True
         t: array([0.00000000e+00, 1.00000000e-04, 1.10000000e-03, 1.11000000e-02,
        1.11100000e-01, 1.11110000e+00, 2.79214387e+00, 3.62727257e+00,
        4.46240128e+00, 5.31679544e+00, 1.38607371e+01, 8.10612092e+01])
  t_events: [array([41.57554744]), array([73.97050835]), array([81.06120925])]
         y: array([[ 0.00000000e+00,  4.50950000e-07,  5.45649500e-05,
          5.55615495e-03,  5.56617055e-01,  5.56717261e+01,
          3.51563658e+02,  5.93319709e+02,  8.91394822e+02,
          1.19888202e+03,  3.87988823e+03,  2.27373675e-12],
        [ 0.00000000e+00,  9.01900000e-03,  9.92090000e-02,
          1.00110900e+00,  1.00201090e+01,  1.00210109e+02,
          2.51823455e+02,  3.27143713e+02,  3.64079964e+02,
          3.55698357e+02,  2.71882290e+02, -3.87354342e+02]])
  y_events: [array([[7647.47127891,    0.        ]]), array([[2500.        , -317.
↪79456649]]), array([[ 2.27373675e-12, -3.87354342e+02]])],   message: 'A␣
↪termination event occurred.'
      nfev: 56
      njev: 0
       nlu: 0
       sol: <scipy.integrate._ivp.common.OdeSolution object at 0x11b3ef128>
    status: 1
   success: True
         t: array([ 48.57554744,  48.74532045,  50.44305052,  58.80397919,
         67.16490786,  76.66698838,  88.28878273, 103.37969057,
        118.94418694])
  t_events: [array([], dtype=float64), array([98.98862675]), array([118.94418694])]
         y: array([[ 8.40712628e+03,  8.39536954e+03,  8.26755846e+03,
          7.44367144e+03,  6.46209201e+03,  5.29093782e+03,
          3.83965685e+03,  1.94984670e+03, -6.82121026e-13],
        [-6.86700000e+01, -6.98266172e+01, -8.04575605e+01,
         -1.11386543e+02, -1.21372123e+02, -1.24375610e+02,
```

(continues on next page)

(continued from previous page)

```
        -1.25118873e+02, -1.25249980e+02, -1.25270546e+02]])
 y_events: [array([], dtype=float64), array([[2500.        , -125.25361785]]),␣
↪array([[-6.82121026e-13, -1.25270546e+02]])],   message: 'A termination event␣
↪occurred.'
     nfev: 44
     njev: 0
      nlu: 0
      sol: <scipy.integrate._ivp.common.OdeSolution object at 0x11b2e0940>
   status: 1
  success: True
        t: array([ 98.98862675,  99.16100776, 100.88481792, 105.31320676,
       110.43121881, 117.00269908, 125.26143932, 125.51024178])
 t_events: [array([], dtype=float64), array([98.98862675]), array([125.51024178])]
        y: array([[ 2.50000000e+03,  2.47855169e+03,  2.27717200e+03,
         1.82378419e+03,  1.34657707e+03,  7.55889851e+02,
         2.20680954e+01, -3.09086090e-13],
       [-1.25253618e+02, -1.23608776e+02, -1.11084655e+02,
        -9.64920142e+01, -9.10798051e+01, -8.91815892e+01,
        -8.86975963e+01, -8.86916238e+01]])
 y_events: [array([], dtype=float64), array([[2500.        , -125.25361785]]),␣
↪array([[-3.09086090e-13, -8.86916238e+01]])]]
 ], Solution:
[
 Stages: [0, 2]
 ODEresults: [  message: 'A termination event occurred.'
     nfev: 98
     njev: 0
      nlu: 0
      sol: <scipy.integrate._ivp.common.OdeSolution object at 0x11b3fefd0>
   status: 1
  success: True
        t: array([0.00000000e+00, 1.00000000e-04, 1.10000000e-03, 1.11000000e-02,
       1.11100000e-01, 1.11110000e+00, 2.79214387e+00, 3.62727257e+00,
       4.46240128e+00, 5.31679544e+00, 1.38607371e+01, 8.10612092e+01])
 t_events: [array([41.57554744]), array([73.97050835]), array([81.06120925])]
        y: array([[ 0.00000000e+00,  4.50950000e-07,  5.45649500e-05,
         5.55615495e-03,  5.56617055e-01,  5.56717261e+01,
         3.51563658e+02,  5.93319709e+02,  8.91394822e+02,
         1.19888202e+03,  3.87988823e+03,  2.27373675e-12],
       [ 0.00000000e+00,  9.01900000e-03,  9.92090000e-02,
         1.00110900e+00,  1.00201090e+01,  1.00210109e+02,
         2.51823455e+02,  3.27143713e+02,  3.64079964e+02,
         3.55698357e+02,  2.71882290e+02, -3.87354342e+02]])
 y_events: [array([[7647.47127891,    0.        ]]), array([[2500.        , -317.
↪79456649]]), array([[ 2.27373675e-12, -3.87354342e+02]])],   message: 'A␣
↪termination event occurred.'
     nfev: 74
     njev: 0
      nlu: 0
      sol: <scipy.integrate._ivp.common.OdeSolution object at 0x11b274a90>
   status: 1
  success: True
        t: array([ 48.57554744,  48.76522553,  50.66200641,  57.53957462,
        63.81072785,  71.9735484 ,  82.29088893,  96.368502  ,
       115.1045594 , 129.14786014, 143.19116087, 144.54624575])
 t_events: [array([], dtype=float64), array([116.32442853]), array([144.54624575])]
        y: array([[ 8.40712628e+03,  8.39403141e+03,  8.25628167e+03,
```

(continues on next page)

(continued from previous page)

```
          7.69420222e+03,  7.14890513e+03,  6.42860842e+03,
          5.51511814e+03,  4.26803222e+03,  2.60794894e+03,
          1.36397133e+03,  1.19964648e+02, -2.23110419e-12],
        [-6.86700000e+01, -6.94006878e+01, -7.54960527e+01,
         -8.55664200e+01, -8.78189043e+01, -8.84484511e+01,
         -8.85621970e+01, -8.85700103e+01, -8.85087345e+01,
         -8.85324160e+01, -8.85492817e+01, -8.85672260e+01]])
 y_events: [array([], dtype=float64), array([[2500.        ,  -88.53487616]]),␣
↪array([[-2.23110419e-12, -8.85672260e+01]])]
 ], Solution:
[
 Stages: [0, 2]
 ODEresults: [  message: 'A termination event occurred.'
     nfev: 98
     njev: 0
      nlu: 0
      sol: <scipy.integrate._ivp.common.OdeSolution object at 0x11b3fefd0>
   status: 1
  success: True
        t: array([0.00000000e+00, 1.00000000e-04, 1.10000000e-03, 1.11000000e-02,
       1.11100000e-01, 1.11110000e+00, 2.79214387e+00, 3.62727257e+00,
       4.46240128e+00, 5.31679544e+00, 1.38607371e+01, 8.10612092e+01])
 t_events: [array([41.57554744]), array([73.97050835]), array([81.06120925])]
        y: array([[ 0.00000000e+00,  4.50950000e-07,  5.45649500e-05,
         5.55615495e-03,  5.56617055e-01,  5.56717261e+01,
         3.51563658e+02,  5.93319709e+02,  8.91394822e+02,
         1.19888202e+03,  3.87988823e+03,  2.27373675e-12],
        [ 0.00000000e+00,  9.01900000e-03,  9.92090000e-02,
         1.00110900e+00,  1.00201090e+01,  1.00210109e+02,
         2.51823455e+02,  3.27143713e+02,  3.64079964e+02,
         3.55698357e+02,  2.71882290e+02, -3.87354342e+02]])
 y_events: [array([[7647.47127891,    0.        ]]), array([[2500.        , -317.
↪79456649]]), array([[ 2.27373675e-12, -3.87354342e+02]])],   message: 'A␣
↪termination event occurred.'
     nfev: 50
     njev: 0
      nlu: 0
      sol: <scipy.integrate._ivp.common.OdeSolution object at 0x11b27fb70>
   status: 1
  success: True
        t: array([73.97050835, 74.10004171, 75.39537534, 77.20624562, 79.75509271,
       83.20855596, 87.71767394, 93.41389729, 94.72605601])
 t_events: [array([], dtype=float64), array([73.97050835]), array([94.72605601])]
        y: array([[ 2.50000000e+03,  2.45977950e+03,  2.13359441e+03,
         1.80664773e+03,  1.45894891e+03,  1.07687869e+03,
         6.38337651e+02,  1.17860133e+02, -5.25801624e-13],
        [-3.17794566e+02, -3.03451782e+02, -2.12666431e+02,
         -1.56188941e+02, -1.21679916e+02, -1.02641487e+02,
         -9.35489652e+01, -8.99854107e+01, -8.96244241e+01]])
 y_events: [array([], dtype=float64), array([[2500.        , -317.79456649]]),␣
↪array([[-5.25801624e-13, -8.96244241e+01]])]
 ]]
```

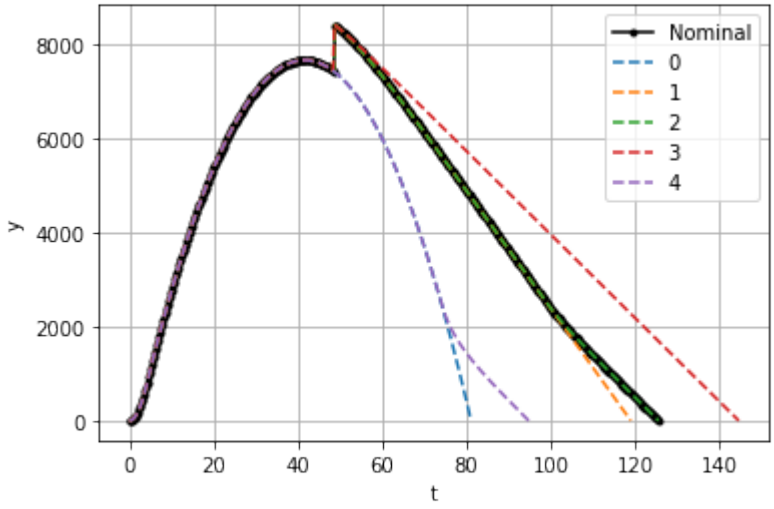
[39]:
```
# number of possible outcomes
len(full_sol)
```

[39]:
```
5
```

Plot the solutions

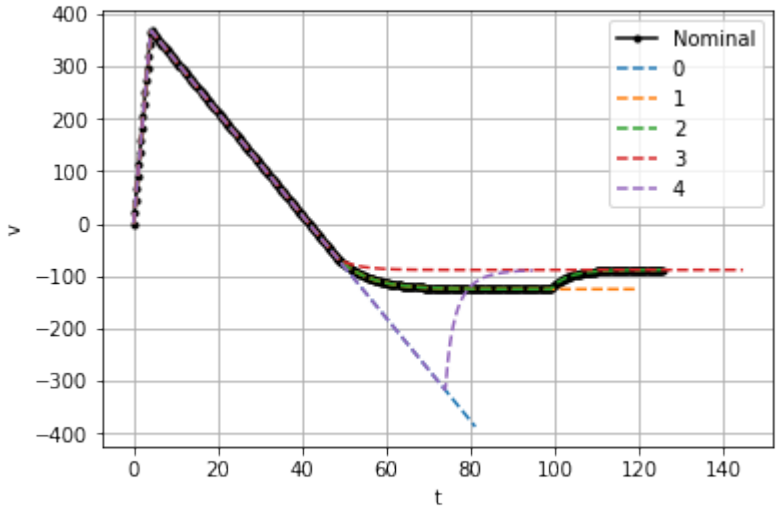
[54]:
```
t_range = np.linspace(nominal_sol.t_min(),nominal_sol.t_max(), 500)
plt.plot(t_range,nominal_sol.sol(t_range)[0], '.-k', label='Nominal')


for i, sol in enumerate(full_sol):
    t_range = np.linspace(sol.t_min(),sol.t_max(), 500)
    plt.plot(t_range,sol.sol(t_range)[0], '--',label=i)

plt.grid()
plt.xlabel('t')
plt.ylabel('y')
plt.legend()
```

[54]:
```
<matplotlib.legend.Legend at 0x11ba01748>
```



[55]:
```
t_range = np.linspace(nominal_sol.t_min(),nominal_sol.t_max(), 500)
plt.plot(t_range,nominal_sol.sol(t_range)[1], '.-k', label='Nominal')


for i, sol in enumerate(full_sol):
    t_range = np.linspace(sol.t_min(),sol.t_max(), 500)
    plt.plot(t_range,sol.sol(t_range)[1], '--',label=i)

plt.grid()
plt.xlabel('t')
plt.ylabel('v')
plt.legend()
```

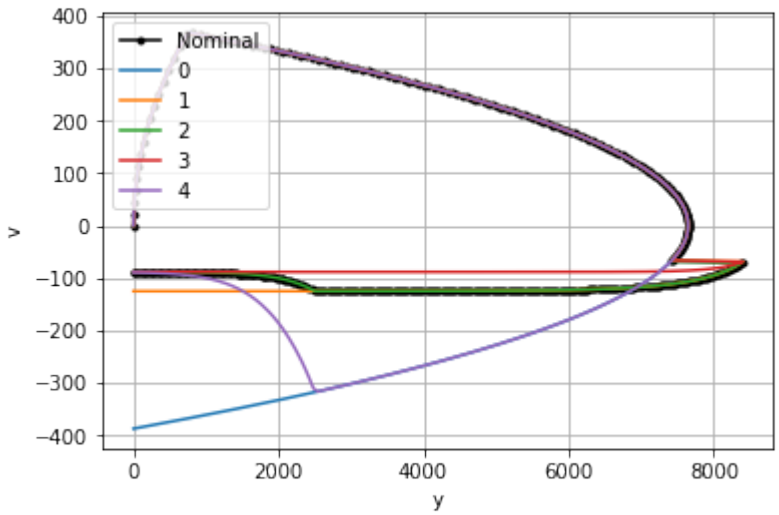
[55]:
```
<matplotlib.legend.Legend at 0x11bb699e8>
```



sometimes its easier to see it in the state space

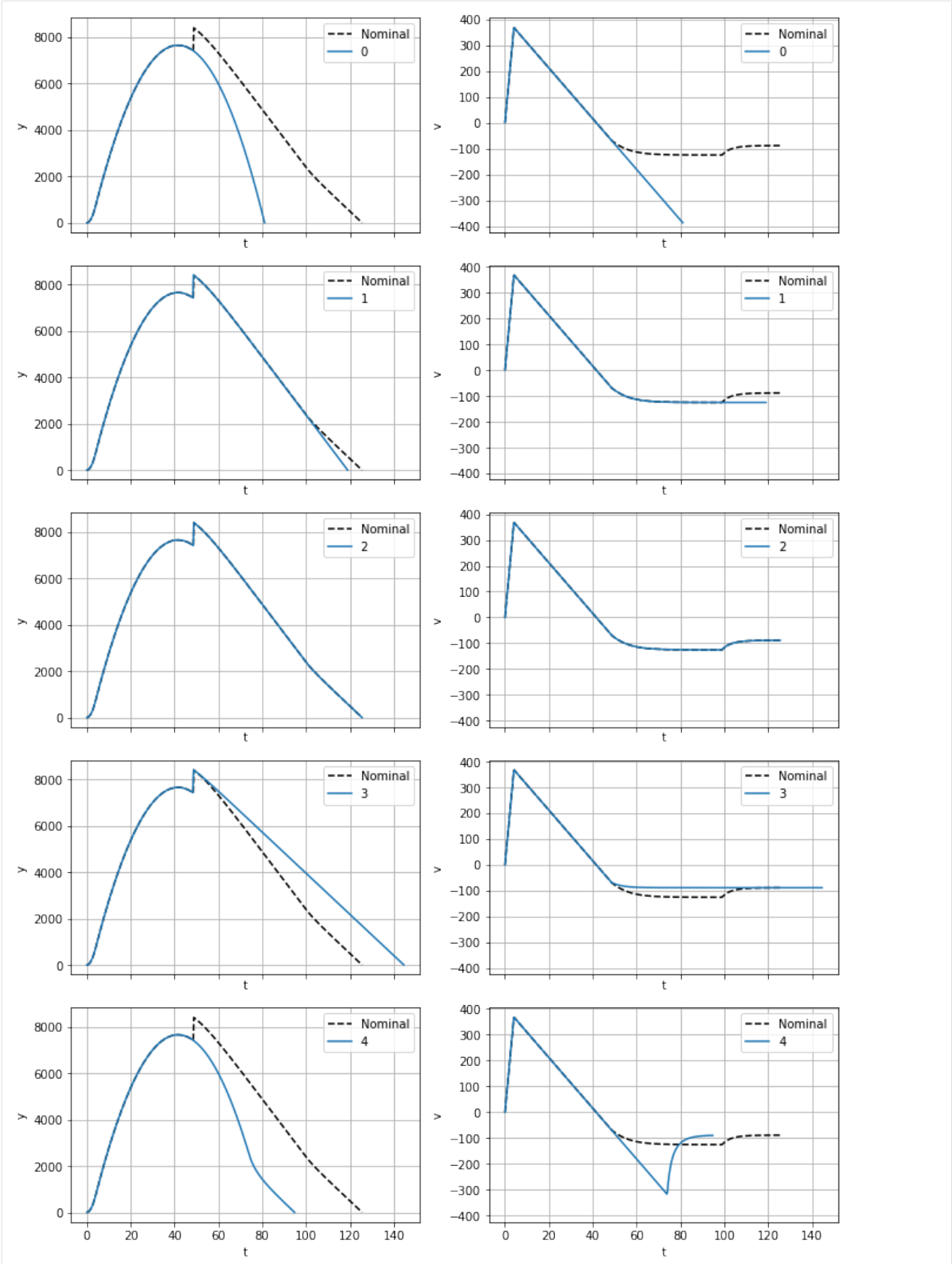
[56]:
```
t_range = np.linspace(nominal_sol.t_min(),nominal_sol.t_max(), 500)
plt.plot(nominal_sol.sol(t_range)[0],nominal_sol.sol(t_range)[1], '.-k', label=
↪'Nominal')


i=0;
for sol in full_sol:
    t_range = np.linspace(sol.t_min(),sol.t_max(), 500)
    plt.plot(sol.sol(t_range)[0],sol.sol(t_range)[1], label=i)
    i+=1
    #plt.xlim([0,50])
plt.grid()
plt.xlabel('y')
plt.ylabel('v')
plt.legend()
```

[56]:
```
<matplotlib.legend.Legend at 0x11c7688d0>
```



or as a list to see what is happening in each

[57]:
```
i=0;
fig, axes = plt.subplots(len(full_sol),2, sharex='col', sharey='col', figsize=(10,15),
↪ squeeze=False)
for sol in full_sol:

    t_range_nom = np.linspace(nominal_sol.t_min(),nominal_sol.t_max(), 500)
    axes[i][0].plot(t_range_nom,nominal_sol.sol(t_range_nom)[0], '--k', label='Nominal
↪')
    axes[i][1].plot(t_range_nom,nominal_sol.sol(t_range_nom)[1], '--k', label='Nominal
↪')

    t_range = np.linspace(sol.t_min(),sol.t_max(), 500)
    axes[i][0].plot(t_range,sol.sol(t_range)[0], label=i)
    axes[i][1].plot(t_range,sol.sol(t_range)[1], label=i)

    axes[i][0].grid(True)
    axes[i][0].set_xlabel('t')
    axes[i][0].set_ylabel('y')
    axes[i][0].legend()

    axes[i][1].grid(True)
    axes[i][1].set_xlabel('t')
    axes[i][1].set_ylabel('v')
    axes[i][1].legend()
    i+=1

plt.tight_layout()
```


Nominal
0
y
v
t
Nominal
1
y
v
t
Nominal
2
y
v
t
Nominal
3
y
v
t
Nominal
4
y
v
t

CHAPTER

FIVE

MODULE REFERENCE

5.1 rocketPy package

5.1.1 Submodules

5.1.2 rocketPy.components module

Creates the components of a rocket. Base classes and the specific useful components are created. All components inherit from `Component` From here, it splits into `InternalComponent` or `ExternalComponent` where the difference is used to help the drag model, and plotting.

class rocketPy.components.**BodyTube**(*name='Body Tube'*, *mass=None*, *inertia=None*, *diameter=None*, *length=None*, *wall_thickness=<Quantity(2, 'millimeter')>*, *material=None*)

Bases: *rocketPy.components.ExternalComponent*

CNa(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*, *K=1.1*)

estimate_inertia()

Generic method to estimate the mass of the component - assume inertia is 0. This method should be overridden for each component specified

estimate_mass()

Generic method to estimate the mass of the component - assume mass is 0. This method should be overridden for each component specified

plot_coords(*rotation=<Quantity(0, 'degree')>*)

xcg()

xcp(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

class rocketPy.components.**Component**(*name='Component'*, *mass=None*, *inertia=None*)

Bases: `object`

Base class for all components

Parameters

- **name** (*str*) – Name of component. Defaults to 'Component'.
- **mass** (*function or None or Quantity*) – Mass of component. Defaults to None. If `None`: calls the `self.estimate_mass()` method to assign mass. If `function`: calls the function and assigns mass If `Pint.Quantity`: assigns mass directly.
- **inertia** (*function or None or Quantity*) – Assigns inertia of the rocket, in (I_xx, I_yy, I_zz). Rest of the components are assumed to be 0. Defaults to None. If `None`:

calls the `self.estimate_inertia()` method to assign inertia. If `function`: calls the function to assign the inertia. The function must return a tuple of (I_xx, I_yy, I_zz) If `tuple of Pint.Quantity`: assigns the inertia direction. The tuple must be (I_xx, I_yy, I_zz) with each being a Pint.Quantity of the right units.

Examples Examples should be written in doctest format, and should illustrate how to use the function/class.
>>>

estimate_mass
Description of parameter *estimate_mass*.

Type type

I_xx
Description of parameter *I_xx*.

Type type

I_yy
Description of parameter *I_yy*.

Type type

I_zz
Description of parameter *I_zz*.

Type type

estimate_inertia
Description of parameter *estimate_inertia*.

Type type

x_ref
Description of parameter *x_ref*.

Type type

y_ref
Description of parameter *y_ref*.

Type type

z_ref
Description of parameter *z_ref*.

Type type

A_ref
Description of parameter *A_ref*.

Type type

name

mass

describe()

estimate_inertia()
Generic method to estimate the mass of the component - assume inertia is 0. This method should be overridden for each component specified

estimate_mass()
Generic method to estimate the mass of the component - assume mass is 0. This method should be overridden for each component specified

plot(*ax=None*, *rotation=<Quantity(0, 'degree')>*, *unit=<Unit('meter')>*)
Plots the component.

Parameters

- **ax** (*type*) – Description of parameter *ax*. Defaults to None.
- **rotation** (*type*) – Description of parameter *rotation*. Defaults to 0*ureg.degree.
- **unit** (*type*) – Description of parameter *unit*. Defaults to ureg.m.

Returns Description of returned object.

Return type type

Examples Examples should be written in doctest format, and should illustrate how to use the function/class. >>>

plot_coords()

set_position(*start_of=None*, *end_of=None*, *middle_of=None*, *after=None*, *offset=<Quantity(0, 'meter')>*)
Defines the x_ref of this component relative to other components.

Parameters

- **start_of** (Component) – If not None, aligns self with other. Defaults to None.
- **end_of** (Component) – If not None, aligns end of self with other. Defaults to None.
- **middle_of** (Component) – If not None, aligns midpoints of self and other. Defaults to None.
- **after** (Component) – If not None, aligns start of self to end of other. Defaults to None.
- **offset** (*Pint.Quantity*) – Follows rules as above, but adds offset to self.x_ref. Defaults to 0*ureg.m.

Examples Examples should be written in doctest format, and should illustrate how to use the function/class. >>>

xcg(**args*)

xcp(**args*)

ycg(**args*)

zcg(**args*)

class rocketPy.components.**Cylinder**(*name='Internal Cylinder'*, *mass=None*, *inertia=None*, *diameter=<Quantity(6, 'inch')>*, *length=<Quantity(6, 'inch')>*, *density=None*)
Bases: *rocketPy.components.InternalComponent*

estimate_inertia()
Generic method to estimate the mass of the component - assume inertia is 0. This method should be overridden for each component specified

estimate_mass()
Generic method to estimate the mass of the component - assume mass is 0. This method should be overridden for each component specified

class rocketPy.components.**ExternalComponent**(*name='External Component'*, *mass=None*, *inertia=None*, *A_ref=<Quantity(28.274333882308138, 'inch ** 2')>*)
Bases: *rocketPy.components.Component*

CN(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

CNa(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

class rocketPy.components.**FinSet**(*name='Fins'*, *mass=None*, *inertia=None*, *n=None*, *span=None*, *root_chord=None*, *tip_chord=None*, *mid_sweep=None*, *tube_dia=None*, *thickness=None*, *material=None*)
Bases: *rocketPy.components.ExternalComponent*

CNa(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

estimate_inertia()
Generic method to estimate the mass of the component - assume inertia is 0. This method should be overridden for each component specified

estimate_mass()
Generic method to estimate the mass of the component - assume mass is 0. This method should be overridden for each component specified

leading_sweep()
Return the leading edge sweep of the fins

plot_coords(*rotation=<Quantity(0, 'radian')>*)

xcg()

xcp(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

class rocketPy.components.**InternalComponent**(*name='Internal Component'*, *mass=None*, *inertia=None*)
Bases: *rocketPy.components.Component*

CN(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

class rocketPy.components.**NoseCone**(*name='Nose Cone'*, *mass=None*, *inertia=None*, *shape='Conical'*, *diameter=None*, *length=None*, *fineness=None*, *wall_thickness=<Quantity(2, 'millimeter')>*, *material=None*)
Bases: *rocketPy.components.ExternalComponent*

CNa(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

estimate_inertia()
Generic method to estimate the mass of the component - assume inertia is 0. This method should be overridden for each component specified

estimate_mass()
Method to estimate the mass of the nose cone

plot_coords(*rotation=<Quantity(0, 'degree')>*)

xcg()

xcp(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

class rocketPy.components.**Transition**(*name='Transition'*, *mass=None*, *inertia=None*, *fore_dia=None*, *aft_dia=None*, *length=None*, *wall_thickness=<Quantity(2, 'millimeter')>*, *material=None*)

Bases: *rocketPy.components.ExternalComponent*

CNa(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

estimate_inertia()

Generic method to estimate the mass of the component - assume inertia is 0. This method should be overridden for each component specified

estimate_mass()

Generic method to estimate the mass of the component - assume mass is 0. This method should be overridden for each component specified

plot_coords(*rotation=<Quantity(0, 'degree')>*)

xcg()

xcp(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

5.1.3 rocketPy.materials module

Defines all the materials and their material properties. Base class Material allows for users to define a material, while some specific commonly used materials are predefined to help speed up the design process. Users should check that the right material properties are assumed for their parts.

class rocketPy.materials.**Acrylic**(*name='Acrylic'*)

Bases: *rocketPy.materials.Material*

class rocketPy.materials.**Aluminium**(*name='Al-6061-T6'*)

Bases: *rocketPy.materials.Material*

Defines a basic aluminium.

Parameters **name** (*str*) – Description of parameter *name*. Defaults to 'Al-6061-T6'.

Examples Examples should be written in doctest format, and should illustrate how to use the function/class. >>>

density

Description of parameter *density*.

Type Pint.Quantity

tensile_modulus

Description of parameter *tensile_modulus*.

Type Pint.Quantity

tensile_strength

Description of parameter *tensile_strength*.

Type Pint.Quantity

max_temp

Description of parameter *max_temp*.

Type Pint.Quantity

class rocketPy.materials.**Material**(*name*)
Bases: object

Base class for defining material properties

Parameters **name** (*str*) – Name of material.

Examples Examples should be written in doctest format, and should illustrate how to use the function/class.
>>>

density
Material Density.

Type Pint.Quantity

name
name

Type str

describe()

class rocketPy.materials.**PLA**(*name='PLA'*)
Bases: *rocketPy.materials.Material*

class rocketPy.materials.**Phenolic**(*name='Phenolic'*)
Bases: *rocketPy.materials.Material*

class rocketPy.materials.**Plywood**(*name='Plywood'*)
Bases: *rocketPy.materials.Material*

class rocketPy.materials.**Polycarbonate**(*name='Polycarbonate'*)
Bases: *rocketPy.materials.Material*

5.1.4 rocketPy.rocket module

Module to describe the rocket

class rocketPy.rocket.**Rocket**(*name='Rocket'*)
Bases: object

CA(*alpha=<Quantity(0, 'radian')>, Re=1000000.0, Mach=0.3*)
Compute the axial drag force from the normal force and the axial force

CD(*alpha=<Quantity(0, 'radian')>, Re=1000000.0, Mach=0.3*)
Calculate the drag force at some angle of attack, including compressibility

CD0(*Re=1000000.0*)
Calcualte the zero angle-of-attack incompressible drag of the rocket. Generally uses DATCOM method (as specified by Box [1]) Reynolds number refers to the reynolds number by the length of the rocket.

CD0_b(*Re=1000000.0*)
Calcualte the zero-angle of attack drag due to base drag

CD0_f(*Re=1000000.0*)
Calcualte the zero-angle of attack drag due to the fins, including the effect of the interference

CD0_fb(*Re=1000000.0*)
Calcualte the zero-angle of attack drag due to forebody of the rocket

CD_body_alpha(*alpha=<Quantity(0, 'radian')>*)

CD_fin_alpha(*alpha*)

CN(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

CNa(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

Cf(*Re=1000000.0*)
Return the viscous friction coefficient at a Reynolds number

add(*component*)

describe(*describe_components=False*)

inertia_matrix(*mass=None*, *with_inverse=False*)

inertia_xx()

inertia_yy()

inertia_zz()

length()

mass()

plot(*ax=None*, *unit=<Unit('meter')>*, *rotation=<Quantity(0, 'degree')>*, *plot_component_cp=True*, *plot_component_cg=True*, *alpha=<Quantity(0, 'degree')>*, *Re=1000000.0*, *Mach=0.3*)

plot_cg(*ax=None*, *unit=<Unit('meter')>*)

plot_cp(*ax=None*, *unit=<Unit('meter')>*, *alpha=<Quantity(0, 'degree')>*, *Re=1000000.0*, *Mach=0.3*)

set_boat_tail(*component*)
Set the boat tail component

set_body_tube(*component*)
Set the body tube of the rocket

set_fins(*component*)
Set the fin set of the rocket

set_nose_cone(*component*)
Set the nose cone of the rocket

static_margin(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*, *mass=None*)

xcg(*mass=None*)

xcp(*alpha=<Quantity(0, 'radian')>*, *Re=1000000.0*, *Mach=0.3*)

5.1.5 rocketPy.util module

Utility functions for rocketPy

rocketPy.util.**angle_between**(*va*, *vb*)
Return the angle between two column vectors using the dot product

rocketPy.util.**mach_correction**(*Ma=0.0*, *method='default'*)
Performs the Prandtl-Glauert compressibility correction, extended for supersonic region.

Parameters

- **Ma** (*dimensionless float*) – Mach Number. Defaults to 0.0.
- **method** (*str*) – Choose the correction method (‘default’ or ‘Cambridge’). Defaults to ‘default’.

Returns Mach correction multiplier (float)

Return type dimensionless float

Examples Examples should be written in doctest format, and should illustrate how to use the function/class. >>> mach_correction(0.5) 1.1547005383792517 >> mach_correction(1.5) 0.8944271909999159

rocketPy.util.**si**(*v*)

Utility function to convert a Pint Quantity into a float in SI units.

Parameters **v** (*Pint.Quantity or list/numpy.Array of Pint.Quantity*) – quantity or list of quantities to convert.

Returns magnitudes in SI units.

Return type float or list of floats

Examples Examples should be written in doctest format, and should illustrate how to use the function/class. >>> si(5.0*ureg.meter) 5.0 >> si(6.0*ureg.inch) 0.1524

rocketPy.util.**unit_vector**(*v*)

Return a unit vector in the direction of v.

5.1.6 rocketPy.solution module

class rocketPy.solution.**Solution**(*sol*, *stage*)

Bases: object

DOF()

Returns the number of degrees of freedom in the state vector

Returns length of state vector

Return type int

__add__(*other*)

Overloading the + operator to allow solutions to be 'added' together, essentially chaining them.

Parameters **other** (Solution) –

Returns Solution object with both objects inside

Return type *Solution*

__init__(*sol*, *stage*)

Wrapper for np.ODEresult to store extra information. Most importantly, allows multiple ODEresults to be chained together, with potential for storing extra information in them, and making it easy to access properties from each.

Parameters

- **sol** (*ODEresult or List of ODEresult*) – The ODEresult objects to be stored
- **stage** (*int or list of int*) – Corresponding stage counters

Returns Solution object

Return type *Solution*

__repr__()

Display string

sol(*time*, *error='raise'*)

Returns the state at requested times. Assumes that scipy.integrate was called with dense_output=True and thus uses scipy's interpolation method

Parameters

- **time** (*float or np.array*) – Float time to request the solution for, or list/array of times that the state is requested for.
- **error** (*'raise' or numeric*) – if the requested time is outside the interpolations capabilities, if error='raise', will raise a warning, else will return error in the state. Maintains the shape of the state vector.

Returns state vector. If time is float, returns simple array of the right length. if time is a list of length n and state has length m, returns a shape of size (n x m).

Return type np.array

t_max()

Maximim time of all the solutions in this object. Computes a simple maximum, gaps are not accounted for.

Returns maximum time

Return type float

t_min()

Minimum time of all the solutions in this object. Computes a simple minimum, gaps are not accounted for.

Returns minimum time

Return type float

5.1.7 rocketPy.simulation module

class rocketPy.simulation.**Simulation**(*object*)

Bases: object

full_solve(*t_span*, *y0*, *starting_stage*, ***kwargs*)

Solves every single possible outcome.

#todo (low): add probability weighting

Parameters

- **t_span** (*[float, float]*) – Timespan for the solve
- **y0** (*np.array*) – Initial state vector
- **starting_stage** (*int*) – Initial stage to start from

Returns list of possible outcomes

Return type list of rocketPy.Solution

nominal_solve(*t_span*, *y0*, *starting_stage*, ***kwargs*)

Solves for a nominal flight

Parameters

- **t_span** (*[float, float]*) – Timespan to simulate over
- **y0** (*np.array*) – Initial state
- **starting_stage** (*int*) – which stage to start in

Returns Solution object

Return type rocketPy.Solution

solve(*t_span*, *y0*, *stage*, *user_events=[]*, ***kwargs*)

Thin wrapper for scipy.solve_ivp to handle stages and events more intuitively. Arguments generally follow scipy.solve_ivp. The dynamics are picked up from object.dynamics, and dense_output=True to allow for solutions to be queried later Solve method uses scipy default, but can be specified using kwargs

Parameters

- **t_span** (*[float, float]*) – Simulation start and stop time.
- **y0** (*np.array*) – Initial state vector
- **stage** (*float*) – stage number to use for the solve

- **user_events** (*list of functions*) – each function is an event function. Defaults to [].

Returns solution object with the stage stored
Return type rocketPy.Solution

5.1.8 Module contents

Top-level package for rocketPy.

CHAPTER

SIX

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/dev10110/rocketPy/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

6.1.4 Write Documentation

rocketPy could always use more documentation, whether as part of the official rocketPy docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dev10110/rocketPy/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *rocketPy* for local development.

1. Fork the *rocketPy* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/rocketPy.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv rocketPy
$ cd rocketPy/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 rocketPy tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.org/dev10110/rocketPy/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ pytest tests.test_rocketPy
```

6.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SEVEN

CREDITS

7.1 Development Lead

- Devansh Ramgopal Agrawal <devanshinspace@gmail.com>

7.2 Contributors

None yet. Why not be the first?

CHAPTER

EIGHT

HISTORY

8.1 0.1.0 (2020-01-23)

- First release on PyPI.

8.2 0.1.3 (2020-01-23)

- Updates, with the rocket components and an example defined.

8.3 0.1.4 (2020-04-15)

- Added the simulation class!

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

E

F

I

L

M

N

P

R

S

T

U

X

Y

Z