Assignment 2 Solution

Joshua Sam

February 25, 2021

This report discusses the testing phase for `BodyT`, `CircleT`, `TriangleT` and `Scene` classes. It also discusses the results of running the same tests on the partner files. The assignment specifications are then critiqued and the requested discussion questions are answered.

1 Testing of the Original Program

The tests were done with the pytest framework which allowed easy assertions as well as use copies of each object for each test function. The program successfully passed all the test cases provided. All methods except setters, getters and the sim method of `Scene.py`, were tested against. A total of `22` tests were passed. The test cases selected include:

- Floating point numbers which were selected to test if the output was approximately equal to the expected result (Since floating point numbers cannot be compared for exact equality).
- Negative numbers which were selected to test if a `ValueError` is raised. For instance, a ValueError is raised if the mass of a `Circle`, `Triangle` or a `Body` is negative. However, this case is not tested in `test_driver.py` after reviewing the test file again.
- The number `0` was selected to test for any `ValueError` raised and any possible errors raised during calculations due to `ZeroDivisionError`. For instance, a `ValueError` is raised if the mass is 0 for any shape.

2 Results of Testing Partner's Code

All the test cases passed successfully. The test cases used to test were not consistent, i.e. a few methods were only tested with positive floating point numbers. Not all methods were tested (`sim`) against . Cases which would raise `ValueError` were not tested against. A change I would include in my implementation is to make the local functions used in `BodyT` private, thus restricting access to those functions from other classes as this is a violation of the OOP principal of `Data Encapsulation`. The particular local functions mentioned in the specification are only meant to be used in the `BodyT` class.

Doing this exercise helped me review my implementation once again with the design specifications and partner source code. This helped me find potential errors and bad practices used during implementation. This also made me realize the importance of checking my implementation before it is submited or used. Comparing the results of this exercise to that of Assignment 1, it is observed that more testcases are being used and the scope of the test cases used are much broader in this assignment.

3 Critique of Given Design Specification

I feel this design specification is very straighforward in explaining the requirements. The given specification is formal and this helps communicate the requirements clearly and prevent any ambiguity as compared to using natural language to communicate. I found that Understanding the formal notations used were a bit tedious and overwhelming as I was not sure of how to interpret it in natural language which cost me time.

The use of an interface with abstract methods gave a general idea about the behaviour of the object and how it is used in other modules. This helped me understand how each module was structured.

Using local functions to perform mathematical computuations as observed in `BodyT` and `Scene` helped understand the mathematical representation mentioned in the specification.

A drawback observed in the specification is that it doesnt specify the access modifiers for a method. This made me assume them to be public.

Overall, I found that this specification is not very beginner-friendly one, since it uses formal language to represent requirements but it is extremely helpfull to understand the semantics of a program.

4 Answers

a) In general no, It is not necesarry to write a unit testcase to test setters and getters if all they do are get values and set values. But it would be a good idea to write unit test for them if they do more than just get and set i.e. if they perform some sort of mathematical computation or if they have some complexity to them.

b) We could predefine Fx and Fy in the testing file.

c) One way we could do automated tests for plot.py is using the inbuilt matplotlib.testing package, but this package is still in the testing process and isn't officially released. Another possible way to compare two plots is to save the plots using the savefig() method and then compare them using checksum functions such as MD5 which produces a 128 bit hash values.

d) $(\exists\ xcalc, xtrue \ \cdot\ \frac{||x_{\text{calc}}-x_{\text{true}}||}{||x_{\text{true}}||} < \epsilon)$

e) No, there shouldn't be exceptions for negative cordinates. Center of mass is a point in the plane or space. If the cordinate values are positive or negative it doesnt mean that the center of mass is positive or negative in itself. The centre of mass is just an absolute value of the distance from the origin.

f) From the state invariant we can understand that the sides of a triangle is always greater than 0 and that the mass is always greater than 0 or in other words, the sides of a triangle cannot be 0 or less or the mass of a triangle cannot be 0 or less.

g) list = [i**(1/2) for i in range(5,20,2)]

h)
```
def lower_string(str):
  str_updated = ""
  for i in str:
    if not(i.isupper()):
      str_updated += i
  return str_updated
```

i) The principle of generality means to solve a more general problem than the problem at hand. Abstraction is often used to extract a general solution from a more specefic solution. In a way we can say that abstraction is an application of generality.

j) A module used by many other modules is generally better because it is more stable and less fragile.

E Code for Shape.py

```
## @file Shape.py
#  @author  Joshua Sam Varughese, vaugj1
#  @brief  An interface for implementing masses, center of masses and inertia
#  @date  02/16/2021

from abc import ABC, abstractmethod

## @brief Shape provide an interface for different shapes
## @details The methods in the interface are abstract and need to be
# overridden by the modules that inherits this interface.


class Shape(ABC):

    @abstractmethod
    ## @brief a method to get the x cordinate of the center of mass of the shape
    # @return x cordinate of the center of mass
    def cm_x(self):
        pass

    @abstractmethod
    ## @brief a method to get the y cordinate of the center of mass of the shape
    # @return the y cordinate of the center of mass
    def cm_y(self):
        pass

    @abstractmethod
    ## @brief a method to get the mass of the shape
    # @return the mass of the shape
    def mass(self):
        pass

    @abstractmethod
    ## @brief a method to get the interia of the shape
    # @return the value of inertia of the shape
    def m_inert(self):
        pass
```

F Code for CircleT.py

```
## @file CircleT.py
#  @author Joshua Sam Varughese, varugj1
#  @brief Contains the CircleT type to represent the mass, center of mass and inertia
#  @date 02/16/21

from Shape import Shape

## @brief CircleT is used to represent the circle shape
#  @details inherits Shape


class CircleT(Shape):

    ## @brief constructor for class CircleT, represents the x-cordinate and
    #  y-coordinate of center of mass, radius and the mass of a circle
    #  @param xs the x-coordinate of centre of mass
    #  @param ys the y-coordinate of centre of mass
    #  @param rs the radius of the circular body
    #  @param ms the mass of the circle
    #  @Exception throws value error if radius and mass are negative
    def __init__(self, xs, ys, rs, ms):
        if not(rs > 0 and ms > 0):
            raise ValueError
        self.x = xs
        self.y = ys
        self.r = rs
        self.m = ms

    ## @brief a method to get the x-cordinate of the center of mass of a circle
    #  @return x-coordinate of the center of mass
    def cm_x(self):
        return self.x

    ## @brief a method to get the y-cordinate of the center of mass of a circle
    #  @return y-coordinate of the center of mass
    def cm_y(self):
        return self.y

    ## @brief a method to get the mass of a circle
    #  @return the mass of the circle
    def mass(self):
        return self.m

    ## @brief a method to get the moment of interia of a circle
    # @return the value of moment of inertia of the circle
    def m_inert(self):
        return (self.m * self.r ** 2) / 2
```

G Code for TriangleT.py

```
## @file TriangleT.py
#  @author Joshua Sam Varughese, varugj1
#  @brief Contains the TriangleT type to represent the mass, center of mass and inertia
#  @date 02/16/21

from Shape import Shape

## @brief TriangleT is used to represent the Triangle shape
#  @details inherits Shape


class TriangleT(Shape):

    ## @brief constructor for class TriangleT, represents the x-cordinate and
    #  y-coordinate of center of mass, the side and the mass of a triangle
    #  @param xs the x-coordinate of centre of mass
    #  @param ys the y-coordinate of centre of mass
    #  @param ss length of the side of the triangle
    #  @param ms the mass of the triangle
    #  @Exception throws value error if triangle side and mass are negative
    def __init__(self, xs, ys, ss, ms):
        if not (ss > 0 and ms > 0):
            raise ValueError
        self.x = xs
        self.y = ys
        self.s = ss
        self.m = ms

    ## @brief a method to get the x-cordinate of the center of mass of a triangle
    #  @return x-coordinate of the center of mass
    def cm_x(self):
        return self.x

    ## @brief a method to get the y-cordinate of the center of mass of a triangle
    #  @return y-coordinate of the center of mass
    def cm_y(self):
        return self.y

    ## @brief a method to get the mass of a triangle
    #  @return the mass of the triangle
    def mass(self):
        return self.m

    ## @brief a method to get the moment of interia of a triangle
    # @return the value of moment of inertia of the triangle
    def m_inert(self):
        return (self.m * self.s ** 2) / 12
```

H Code for BodyT.py

```
## @file BodyT.py
#  @author Joshua Sam Varughese, varugj1
#  @brief Contains the BodyT type to represent the mass, center of mass and inertia
#  @date 02/16/2021

from Shape import Shape

## @brief BodyT is used to represent an abstract shape
# detials Inherits from Shape


class BodyT(Shape):

    ## @brief constructor for class BodyT, represents the x-cordinate and
    #  y-coordinate of center of mass and the mass of a body
    #  @param xs the x-coordinate of centre of mass
    #  @param ys the y-coordinate of centre of mass
    #  @param ms the mass of the triangle
    #  @Exception throws value error if length of sequence of centre of masses and the mass
    #  are different and if the sequence of mass is contains negative number
    def __init__(self, xs, ys, ms):

        if not (len(xs) == len(ys) == len(ms)) and \
                (not ([ms[i] > 0 for i in range(len(ms))])):
            raise ValueError()

        self.cmx = cm(xs, ms)
        self.cmy = cm(ys, ms)
        self.m = sum(ms)
        self.moment = mmom(xs, ys, ms) - sum(ms) * (cm(xs, ms) ** 2 + cm(ys, ms) ** 2)

    ## @brief a method to get the x-cordinate of the center of mass of a body
    #  @return x-coordinate of the center of mass
    def cm_x(self):
        return self.cmx

    ## @brief a method to get the y-cordinate of the center of mass of a body
    #  @return y-coordinate of the center of mass
    def cm_y(self):
        return self.cmy

    ## @brief a method to get the mass of a body
    #  @return the mass of the body
    def mass(self):
        return self.m

    ## @brief a method to get the moment of interia of a body
    # @return the value of moment of inertia of the body
    def m_inert(self):
        return self.moment

## @brief a method to get the total centre of mass of a given sequence
#  @param z is the sequence of centre of mass
#  @param m is the sequence of masses
#  @return the total value of center of mass


def cm(z, m):
    tot = 0
    for i in range(0, len(m)):
        tot += z[i] * m[i]
    return tot / sum(m)

## @brief a method to get the total moment of inertia of a given sequence
#  @param x is the sequence of x-coordinate of centre of mass
#  @param y is the sequence of y-coordinate of centre of mass
#  @param m is the sequence of mass
#  @return the total value of moment of inertia


def mmom(x, y, m):
    tot = 0
    for i in range(0, len(m)):
        tot += m[i] * (x[i] ** 2 + y[i] ** 2)
    return tot
```

I Code for Scene.py

```
## @file Scene.py
#  @author Joshua Sam Varughese, varugj1
#  @brief Contains the Scene type to represent type of shape, force, velocity of a shape
#  @date 02/16/21
#  @details Scene is used to set and get force, velocity and type of shape

from scipy.integrate import odeint


class Scene:

    ## @brief constructor for class Scene, represents the x-cordinate and
    #  y-coordinate of force and velocity and the type of shape
    #  @param sp the type of the shape
    #  @param Fpx the x-coordinate of the force
    #  @param Fpy the y-coordinate of the force
    #  @param vpx the x-coordinate of the velocity
    #  @param vpy the y-coordinate of the velocity
    def __init__(self, sp, Fpx, Fpy, vpx, vpy):

        self.s = sp
        self.Fx = Fpx
        self.Fy = Fpy
        self.vx = vpx
        self.vy = vpy

    ## @brief a method to get the type of shape
    #  @return the shape of the body
    def get_shape(self):
        return self.s

    ## @brief a method to get the unbalanced forces of the shape
    #  @return the unbalanced force at the x-coordinate and y-coordinate
    def get_unbal_forces(self):
        return self.Fx, self.Fy

    ## @brief a method to get the velocity of the shape
    #  @return the x-coordinate and y-coordinate of the velocity
    def get_init_velo(self):
        return self.vx, self.vy

    ## @brief a method to set the type of shape
    def set_shape(self, sp):
        self.s = sp

    ## @brief a method to set the forces
    def set_unbal_forces(self, Fpx, Fpy):
        self.Fx, self.Fy = Fpx, Fpy

    ## @brief a method to set the velocities
    def set_init_velo(self, vpx, vpy):
        self.vx, self.vy = vpx, vpy

    ## @brief a method to simulate the motion of a body
    #  @param t_final is the final time.
    #  @param the number of steps or positions.
    #  @return the time and the position and the velocity of the body under the forces
    #  specified.
    def sim(self, t_final, nsteps):
        t = [i * t_final / (nsteps - 1) for i in range(0, nsteps - 1)]
        return t, odeint(self.ode, [self.s.cm_x(), self.s.cm_y(), self.vx, self.vy], t)

    ## @brief a local function that gives the sequnce n elements with type R
    #  @param w a sequence of R
    #  @param t an argument paired to Fx and Fy
    def ode(self, w, t):
        return [w[2], w[3], (self.Fx(t)) / (self.s.mass()), (self.Fy(t)) / (self.s.mass())]
```

J Code for Plot.py

```
## @file Plot.py
#  @author Joshua Sam Varughese
#  @brief plots a graph of the simulation
#  @date 02/16/21

import matplotlib.pyplot as matplot

## @Brief plot grpahs for the simulation
#  @param w sequence of R
#  @param t is the argument passed to Fx and fy.


def plot(w, t):

    if not (len(w) == len(t)):
        raise ValueError

    else:
        x = []
        y = []

        for i in range(len(w)):
            x.append(w[i][0])

        for i in range(len(w)):
            y.append(w[i][1])

        fig, (graph1, graph2, graph3) = matplot.subplots(3)
        fig.suptitle("Motion Simulation")
        graph1.plot(t, x)
        graph2.plot(t, y)
        graph3.plot(x, y)

        graph1.set(ylabel="x (m)")
        graph2.set(ylabel="y (m)")
        graph3.set(ylabel="y (m)")
        graph3.set(xlabel="x (m)")
        matplot.show()
```

K Code for test_driver.py

```
## @file test_driver.py
# @author
# @brief
# @date
# @details

from CircleT import CircleT
from TriangleT import TriangleT
from BodyT import BodyT
from Scene import Scene

from pytest import *

g = 9.81  # accel due to gravity (m/s^2)
m = 1  # mass (kg)


def Fx(t):
    return 5 if t < 5 else 0


def Fy(t):
    return -g * m if t < 3 else g * m


class TestCircleT:

    def setup_method(self, method):
        self.test_circle1 = CircleT(1.0, 10.0, 0.5, 1.0)

    def teardown_method(self, method):
        self.test_circle1 = None

    def test_cm_x(self):
        assert self.test_circle1.cm_x() == 1.0

    def test_cm_y(self):
        assert self.test_circle1.cm_y() == 10.0

    def test_mass(self):
        assert self.test_circle1.mass() == 1.0

    def test_m_inert(self):
        assert approx(self.test_circle1.m_inert()) == approx(0.125)


class TestTriangleT:

    def setup_method(self, method):
        self.test_triangle1 = TriangleT(1.0, -10.0, 5, 17.5)

    def teardown_method(self, method):
        self.test_triangle1 = None

    def test_cm_x(self):
        assert self.test_triangle1.cm_x() == 1.0

    def test_cm_y(self):
        assert self.test_triangle1.cm_y() == -10.0

    def test_mass(self):
        assert self.test_triangle1.mass() == 17.5

    def test_m_inert(self):
        assert approx(self.test_triangle1.m_inert()) == approx(36.4583333)


class TestBodyT:

    def setup_method(self, method):
        self.test_body1 = BodyT([1, -1, -1, 1], [1, 1, -1, -1], [10, 10, 10, 10])

    def teardown_method(self, method):
        self.test_body1 = None

    def test_cm_x(self):
        assert self.test_body1.cm_x() == 0.0

    def test_cm_y(self):
        assert self.test_body1.cm_y() == 0.0

    def test_mass(self):
        assert self.test_body1.mass() == 40

    def test_m_inert(self):
        assert approx(self.test_body1.m_inert()) == approx(80.0)


class TestScene:

    def setup_method(self, method):
        self.test_cricle = CircleT(1, 10.0, 0.5, 1)
        self.test_triangle = TriangleT(1.0, -2.0, 5.0, 10.0)
        self.test_body = BodyT([1, -1, -1, 1], [1, 1, -1, -1], [10, 10, 10, 10])
        self.test_scene1 = Scene(self.test_cricle, Fx, Fy, 0, 0)
        self.test_scene2 = Scene(self.test_triangle, Fx, Fy, 0, 0)
        self.test_scene3 = Scene(self.test_body, Fx, Fy, 0, 0)

    def teardown_method(self, method):
        self.test_scene1 = None
        self.test_scene2 = None
        self.test_scene3 = None

    def test_get_shape_circle(self):
        assert type(self.test_scene1.get_shape()) == CircleT

    def test_get_shape_triangle(self):
        assert type(self.test_scene2.get_shape()) == TriangleT

    def test_get_shape_body(self):
        assert type(self.test_scene3.get_shape()) == BodyT

    def test_get_init_velo1(self):
        assert self.test_scene1.get_init_velo() == (0, 0)

    def test_get_init_vel2(self):
        assert self.test_scene2.get_init_velo() == (0, 0)

    def test_get_init_velo_not(self):
        assert not self.test_scene1.get_init_velo() == (1, 0)

    def test_set_shape(self):
        self.test_scene1.set_shape(TriangleT)
        assert self.test_scene1.get_shape() == TriangleT

    def test_set_shape_not(self):
        self.test_scene2.set_shape(CircleT)
        assert not self.test_scene1.get_shape() == TriangleT

    def test_set_init_vel1(self):
        self.test_scene1.set_init_velo(1, 1)
        assert self.test_scene1.get_init_velo() == (1, 1)

    def test_set_init_vel_not(self):
        self.test_scene2.set_init_velo(1, 1)
        assert self.test_scene1.get_init_velo() == (0, 0)
```

L Code for Partner's CircleT.py

```
## @file CircleT.py
#  @author John Popovici, popovj3
#  @brief Contains the CircleT type to represent a circle shape
#  @date 2021-02-12

from Shape import Shape

## @brief CircleT is a class that implements an ADT for the
#  mathematical concept of a circle


class CircleT(Shape):

    ## @brief constructor method for CircleT
    #  @param x_s a real number representing the center of mass on x axis
    #  @param y_s a real number representing the center of mass on y axis
    #  @param r_s a real number representing the radius
    #  @param m_s a real number representing the mass
    #  @throws Exception ValueError if r_s or m_s not greater than 0
    def __init__(self, x_s, y_s, r_s, m_s):
        if r_s <= 0 or m_s <= 0:
            raise ValueError
        self.x = x_s
        self.y = y_s
        self.r = r_s
        self.m = m_s

    ## @brief get the center of mass on x axis
    #  @return a real number representing the center of mass on x axis
    def cm_x(self):
        return self.x

    ## @brief get the center of mass on y axis
    #  @return a real number representing the center of mass on y axis
    def cm_y(self):
        return self.y

    ## @brief get the mass
    #  @return a real number representing the mass
    def mass(self):
        return self.m

    ## @brief get the moment of inertia
    #  @return a real number representing the moment of inertia
    def m_inert(self):
        return self.m * (self.r ** 2) / 2.0
```

M Code for Partner's TriangleT.py

```
## @file TriangleT.py
#  @author John Popovici, popovj3
#  @brief Contains the TriangleT type to represent a triangle shape
#  @date 2021-02-12

from Shape import Shape

## @brief TriangleT is a class that implements an ADT for the
#  mathematical concept of an equilateral triangle


class TriangleT(Shape):

    ## @brief constructor method for TriangleT
    #  @param x_s a real number representing the center of mass on x axis
    #  @param y_s a real number representing the center of mass on y axis
    #  @param s_s a real number representing the side lengths
    #  @param m_s a real number representing the mass
    #  @throws Exception ValueError if s_s or m_s not greater than 0
    def __init__(self, x_s, y_s, s_s, m_s):
        if s_s <= 0 or m_s <= 0:
            raise ValueError
        self.x = x_s
        self.y = y_s
        self.s = s_s
        self.m = m_s

    ## @brief get the center of mass on x axis
    #  @return a real number representing the center of mass on x axis
    def cm_x(self):
        return self.x

    ## @brief get the center of mass on y axis
    #  @return a real number representing the center of mass on y axis
    def cm_y(self):
        return self.y

    ## @brief get the mass
    #  @return a real number representing the mass
    def mass(self):
        return self.m

    ## @brief get the moment of inertia
    #  @return a real number representing the moment of inertia
    def m_inert(self):
        return self.m * (self.s ** 2) / 12.0
```

N   Code for Partner's BodyT.py

```
## @file Body.py
#  @author John Popovici, popovj3
#  @brief Contains the BodyT type to represent a body shape
#  @date 2021-02-12

from Shape import Shape

## @brief BodyT is a class that implements an ADT for
#  a body that is of a shape


class BodyT(Shape):

    ## @brief constructor method for BodyT
    #  @param x_s a list of real numbers representing center of mass on x axis
    #  @param y_s a list of real numbers representing center of mass on y axis
    #  @param m_s a list of real numbers representing mass
    #  @throws Exception ValueError if parameter list lengths are not equal
    #  or any members of m_s are not greater than 0 or the calculated moment
    #  is less than 0
    def __init__(self, x_s, y_s, m_s):
        if len(x_s) != len(y_s) or len(y_s) != len(m_s):
            raise ValueError
        for mu in m_s:
            if mu <= 0:
                raise ValueError
        self.cmx = self.__cm__(x_s, m_s)
        self.cmy = self.__cm__(y_s, m_s)
        self.m = self.__sum__(m_s)
        self.moment = self.__mmom__(x_s, y_s, m_s) - self.__sum__(m_s) * \
            (self.__cm__(x_s, m_s) ** 2 + self.__cm__(y_s, m_s) ** 2)
        if self.moment < 0:
            raise ValueError

    ## @brief get the center of mass on x axis
    #  @return a real number representing the center of mass on x axis
    def cm_x(self):
        return self.cmx

    ## @brief get the center of mass on y axis
    #  @return a real number representing the center of mass on y axis
    def cm_y(self):
        return self.cmy

    ## @brief get the mass
    #  @return a real number representing the mass
    def mass(self):
        return self.m

    ## @brief get the moment of inertia
    #  @return a real number representing the moment of inertia
    def m_inert(self):
        return self.moment

    @staticmethod
    def __sum__(m_s):
        out = 0
        for mu in m_s:
            out += mu
        return out

    @staticmethod
    def __cm__(z, m):
        top = 0
        bot = 0
        for i in range(len(m)):
            top += z[i] * m[i]
            bot += m[i]
        return top / bot

    @staticmethod
    def __mmom__(x, y, m):
        out = 0
        for i in range(len(m)):
            out += m[i] * (x[i] ** 2 + y[i] ** 2)
        return out
```

O Code for Partner's Scene.py

```python
## @file Scene.py
#  @author John Popovici, popovj3
#  @brief Contains the Scene to run simulation on a shape
#  @date 2021-02-12

# from Shape import Shape
# unused import statement since working in terms of references to objects
from scipy.integrate import odeint

## @brief Scene is a class that runs a simulation for a shape
#  starting with initial velocities and unbalanced forces


class Scene:

    ## @brief constructor method for Scene
    #  @param s_prime a shape object
    #  @param F_x_prime an unbalanced force function in x dir
    #  @param F_y_prime an unbalanced force function in y dir
    #  @param v_x_prime a real number representing initial velocity in x dir
    #  @param v_y_prime a real number representing initial velocity in y dir
    def __init__(self, s_prime, F_x_prime, F_y_prime, v_x_prime, v_y_prime):
        self.s = s_prime
        self.F_x = F_x_prime
        self.F_y = F_y_prime
        self.v_x = v_x_prime
        self.v_y = v_y_prime

    ## @brief get the shape object
    #  @return a shape object
    def get_shape(self):
        return self.s

    ## @brief get the unbalanced force functions
    #  @return the unbalanced force functions
    def get_unbal_forces(self):
        return self.F_x, self.F_y

    ## @brief get the initial velocities
    #  @return the initial velocities
    def get_init_velo(self):
        return self.v_x, self.v_y

    ## @brief set the shape object
    #  @param s_prime a shape object
    def set_shape(self, s_prime):
        self.s = s_prime

    ## @brief set the unbalanced force functions
    #  @param F_x_prime an unbalanced force function in x dir
    #  @param F_y_prime an unbalanced force function in y dir
    def set_unbal_forces(self, F_x_prime, F_y_prime):
        self.F_x = F_x_prime
        self.F_y = F_y_prime

    ## @brief set the initial velocities
    #  @param v_x_prime a real number representing initial velocity in x dir
    #  @param v_y_prime a real number representing initial velocity in y dir
    def set_init_velo(self, v_x_prime, v_y_prime):
        self.v_x = v_x_prime
        self.v_y = v_y_prime

    ## @brief run the simulation on the shape object
    #  @param t_final a real number representing the final time
    #  @param nsteps an integer representing the number of steps
    #  @return lists representing values after simulation executed
    def sim(self, t_final, nsteps):
        t = []
        for i in range(nsteps):
            t.append(i * t_final / (nsteps - 1))
        return t, odeint(self.__ode__, [self.s.cm_x(), self.s.cm_y(),
                                        self.v_x, self.v_y], t)

    def __ode__(self, w, t):
        return [w[2], w[3], self.F_x(t) / self.s.mass(),
                self.F_y(t) / self.s.mass()]
```