Assignment 2 Solution

Hriday Jham, jhamh (400227516)

February 26, 2021

This report discusses the testing phase for CircleT, TriangleT, BodyT and Scene classes implemented for Assignment 2. 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

CircleT:

1. Tested the exceptions $r > 0 \land m > 0$ for the constructor
2. Did one test on cm_x function
3. Did one test on cm_y function
4. performed one test on mass function
5. performed one test on m_inert function

TriangleT:

1. Tested the exceptions $s > 0 \land m > 0$ for the constructor
2. Did one test on cm_x function
3. Did one test on cm_y function
4. performed one test on mass function
5. performed one test on m_inert function

BodyT:

1. Tested the exception x, y, m have same lengths
2. Tested the exception m cannot have a value ≤ 0
3. Did one test on cm_x function
4. Did one test on cm_y function
5. performed one test on mass function
6. performed one test on m_inert function

Scene:

1. Implemented a basic Fx function and an Fy function where the work is being done against gravity
2. Did one test on get_shape function
3. Did one test on get_init_velo function
4. changed the shape of the Scene to test set_shape function
5. changed the initial velocities of the Scene to test set_shape function
6. Since the output of sim function is a sequence of real numbers, used a relative error formula to calculate x_{calc} and x_{true}
7. performed one test on sim function

Results of testing my code:

On testing my code against the test_driver, the code passed all 20 cases.

2 Results of Testing Partner's Code

Number of test cases: 20

Number of passed test cases: 20

On testing my partner's code against my test_driver, his code passed all tests for all classes. This shows that he kept in mind all exceptions while implementing the code and also was able to formulate the desired output for all functions.

I did however feel like the doxygen briefs for the partner's code could have been a little more in detail.

3 Critique of Given Design Specification

This assignment was better explained than the last one, as it clearly mentioned where to raise a ValueError and what exceptions to follow. I found it very easy to read and every part of the assignment was well explained.

I do however feel as if Plot.py could have been explained in a little more detail than was given in the assignment.

4 Answers

a) No I do not think getters and setters should be unit tested unlesss they have a complex code which could result in errors.

b) If we were to unit test the getters and setters for Fx and Fy, since both are functions which take a real number as a parameter and return a real number, I would run the Fx and Fy on some arbitrary real number and test them against the desired output of the function.

c) If automated tests were required for plot.py, then instead of plotting them and comparung the graphs manually, I would generate a file holding all values that have been plotted and compare them to the desired values of the true graph.

d) listcheck : seq of $\mathbb{R}$ X seq of $\mathbb{R}$ $->$ Bool

   $listcheck(x_{calc}, x_{true}) = (\wedge i : \mathbb{N} | i \in [0..|x_{calc}| - 1] : (x_{calci} - x_{truei})/x_{truei} > \epsilon)$

e) There should not be any exceptions for x and y being negative as the coordinates of centre of mass can be negative.

f) To prove $s > 0 \wedge m > 0$, we first prove $s > 0$. We know that s represents the sides of a triangle. The side of a triangle has to have some length, otherwise the triangle ceases to exist. Thus, s will always be greater than 0.

We know that m represents the mass of a triangle. We know that any object that exists in space has to have some mass. Thus, m will always be grater than 0.

g)
```python
L = [math.sqrt(x) for x in range (5, 20)]
```

h)
```python
def No_Uppercase(L):
    x = []
    for i in L:
        x.append(i)
    for i in x:
        if isupper(i):
            x.remove(i)
    return x
```

i) Generality, being the principle of writing generalised code for reusability is closely related to abstraction which is the hiding of information that is not relevant to the user. Having generalised code can easily help with abstraction as the generalised code can easily be called without having to reveal unnecessary information to the user.

j) The scenario where a module is used by many other modules will be better because in high coupling situations, it would avoid confusion and through abstraction, it would make readability of code better.

E Code for Shape.py

```
## @file Shape.py
#  @author Hriday Jham
#  @brief An interface for modules that implement Shape
#  @date 02/16/2021

from abc import ABC, abstractmethod

## @brief Shape provides an interface for shapes
#  @details The method in the interface are abstract and need to be
#  overridden by the modules that inherit this interface


class Shape(ABC):

    @abstractmethod
    ## @brief a generic method for returning the centre of mass
    #  along the x axis
    #  @return a real number indicating the centre of mass along the
    #  x axis
    def cm_x(self):
        pass

    @abstractmethod
    ## @brief a generic method for returning the centre of mass
    #  along the y axis
    #  @return a real number indicating the centre of mass along the
    #  y axis
    def cm_y(self):
        pass

    @abstractmethod
    ## @brief a generic method for returning the mass of the shape
    #  @return a real number indicating the mass of the shape
    def mass(self):
        pass

    @abstractmethod
    ## @brief a generic method for returning the moment of inertia of the shape
    #  @return a real number indicating the moment of inertia of the shape
    def m_inert(self):
        pass
```

F Code for CircleT.py

```
## @file CircleT.py
#  @author Hriday Jham
#  @brief Contains the CircleT type to represent a Circle
#  @date 02/16/2021

from Shape import Shape

## @brief CircleT is used to represent a Circle which is a Shape


class CircleT(Shape):

    ## @brief constructor for CircleT denoting a circle by its x, y, radius and mass
    #  @param takes four real numbers to denote x, y, radius and mass
    def __init__(self, x, y, r, m):
        if r > 0 and m > 0:
            self.x = x
            self.y = y
            self.r = r
            self.m = m
        else:
            raise ValueError("Invalid input for CircleT")

    ## @brief returns the centre of mass along x axis of the circle
    #  @return a real number denoting centre of mass along x axis of the circle
    def cm_x(self):
        return self.x

    ## @brief returns the centre of mass along y axis of the circle
    #  @return a real number denoting centre of mass along y axis of the circle
    def cm_y(self):
        return self.y

    ## @brief returns the mass of the circle
    #  @return a real number denoting mass of the circle
    def mass(self):
        return self.m

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

G Code for TriangleT.py

```python
## @file TriangleT.py
#  @author Hriday Jham
#  @brief Contains the TriangleT type to represent Triangles
#  @date 02/16/2021

from Shape import Shape

## @brief TriangleT is used to represent a triangle which is a Shape


class TriangleT(Shape):

    ## @brief constructor for TriangleT denoting a triangle by its x, y, side and mass
    #  @param takes four real numbers to denote x, y, side and mass
    def __init__(self, x, y, s, m):
        if s > 0 and m > 0:
            self.x = x
            self.y = y
            self.s = s
            self.m = m
        else:
            raise ValueError("Invalid input for TriangleT")

    ## @brief returns the centre of mass along x axis of the triangle
    #  @return a real number denoting centre of mass along x axis of the triangle
    def cm_x(self):
        return self.x

    ## @brief returns the centre of mass along y axis of the triangle
    #  @return a real number denoting centre of mass along y axis of the triangle
    def cm_y(self):
        return self.y

    ## @brief returns the mass of the triangle
    #  @return a real number denoting mass of the triangle
    def mass(self):
        return self.m

    ## @brief returns the moment of inertia of the triangle
    #  @return a real number denoting moment of inertia of the triangle
    def m_inert(self):
        return self.m * self.s * self.s / 12
```

H Code for BodyT.py

```
## @file BodyT.py
#  @author Hriday Jham
#  @brief Contains the BodyT type to represent the object body
#  @date 02/16/2021

from Shape import Shape
from math import pow

## @brief BodyT is used to represent the Body of an object.


class BodyT(Shape):
    ## @brief constructor for class BodyT. BodyT is a set of shapes
    #  @param three sequences of real numbers
    def __init__(self, x, y, m):
        if not (len(x) == len(y) and len(y) == len(m)):
            raise ValueError("Lengths of inputs are invalid")
        else:
            for i in m:
                if i <= 0:
                    raise ValueError("m values cannot be 0 or negative")
            self.cmx = self.__cm(x, m)
            self.cmy = self.__cm(y, m)
            self.m = sum(m)
            self.moment = self.__mmom(x, y, m)
            self.moment -= sum(m) * (pow(self.__cm(x, m), 2) + pow(self.__cm(y, m), 2))

    ## @brief used to calculate centre of mass
    #  @param two sequences of real numbers
    #  @return Real number denoting centre of mass

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

    ## @brief used to calculate moment of inertia
    #  @param three sequences of real numbers
    #  @return Real number used to calculate moment of inertia
    def __mmom(self, x, y, m):
        temp = 0
        for i in range(len(m)):
            temp += m[i] * (pow(x[i], 2) + pow(y[i], 2))
        return temp

    ## @brief return the centre of mass along x axis
    #  @return Real number denoting centre of mass along x axis

    def cm_x(self):
        return self.cmx

    ## @brief return the centre of mass along y axis
    #  @return Real number denoting centre of mass along y axis
    def cm_y(self):
        return self.cmy

    ## @brief return the mass of the Body
    #  @return real number denoting the mass of the body
    def mass(self):
        return self.m

    ## @brief return the moment of inertia of the Body
    #  @return real number denoting the moment of inertia of the body
    def m_inert(self):
        return self.moment
```

I Code for Scene.py

```
## @file Scene.py
#  @author Hriday Jham
#  @brief Contains a class Scene which calculates the
#  required fields to simulate the physics of a scene
#  where an object moves through 2D space.
#  @date 02/16/2021
#  @details Contains functions necessary for 2D Simulation
#  when given a Shape


from scipy.integrate import odeint

## @brief Scene is used to represent the simulation of a
#  scene where an object moves through 2D space


class Scene:

    ## @brief constructor for class Scene.
    #  @param s an object of type Shape
    #  @param Fx a function which takes a real number and returns
    #  a real number
    #  @param Fy a function which takes a real number and returns
    #  a real number
    #  @param vx a real number
    #  @param vy a real number
    def __init__(self, s, Fx, Fy, vx, vy):
        self.s = s
        self.Fx = Fx
        self.Fy = Fy
        self.vx = vx
        self.vy = vy

    ## @brief returns the shape object
    #  @return a Shape object returning the shape

    def get_shape(self):
        return self.s

    ## @brief returns the functions representing the unbalanced forces
    #  @return a function representing the unbalanced forces on x axis
    #  @return a function representing the unbalanced forces on y axis

    def get_unbal_forces(self):
        return self.Fx, self.Fy

    ## @brief returns the initial velocities along the x and y axes
    #  @return a real number representing the initial velocity along the x axis
    #  @return a real number representing the initial velocity along the y axis
    def get_init_velo(self):
        return self.vx, self.vy

    ## @brief setter that sets the Shape
    #  @param s an object of type Shape
    def set_shape(self, s):
        self.s = s

    ## @brief setter that sets the unbalanced forces functions
    #  @param Fx a function that takes a real object and returns a real object
    #  @param Fy a function that takes a real object and returns a real object
    def set_unbal_forces(self, Fx, Fy):
        self.Fx = Fx
        self.Fy = Fy

    ## @brief setter that sets the initial velocities along both axes
    #  @param vx the initial velocity along x axis
    #  @param vy the initial velocity along y axis
    def set_init_velo(self, vx, vy):
        self.vx = vx
        self.vy = vy

    ## @brief function to be used as an input for calling odeint in function sim
    #  @param w a sequence of real numbers of length 4
    #  @param t a real number representing the time
    #  @return a sequence of real numbers of length 4
    def __ode(self, w, t):
```

```
    t = [w[2], w[3], self.Fx(t) / self.s.mass(), self.Fy(t) / self.s.mass()]
    return t

## @brief calculates the final values to be plotted for the simulation
#  @param tfinal a real number representing the final time
#  @param nsteps an integer representing the number of steps
#  @return a sequence of real numbers
#  @return a nested sequence of real numbers with each inner sequence of length 4
def sim(self, tfinal, nsteps):
    t = []
    for i in range(nsteps):
        t.append(i * tfinal / (nsteps - 1))

    return t, odeint(self.__ode, [self.s.cm_x(), self.s.cm_y(), self.vx, self.vy], t)
```

J Code for Plot.py

```
## @file Plot.py
#  @author Hriday Jham
#  @brief A function plot which plots the x-t, y-t and x-y
#  graphs.
#  @date 02/16/2021
#  @details plot uses mathplotlib library to plot the three graphs.

import matplotlib.pyplot as plt


## @brief function plot plots the x-t, y-t and x-y graphs
#  @param w a nested sequence with the inner sequences of length 4
#  @param t a sequence of real numbers.
def plot(w, t):
    if len(w) != len(t):
        raise ValueError("Invalid Inputs for plot")
    else:
        x = []
        y = []
        for i in range(len(w)):
            x.append(w[i][0])
            y.append(w[i][1])

        fig, axs = plt.subplots(3)
        fig.suptitle('Motion Simulation')
        axs[0].plot(t, x)
        axs[0].set(ylabel="x (m)")
        axs[1].plot(t, y)
        axs[1].set(ylabel="y (m)")
        axs[2].plot(x, y)
        axs[2].set(xlabel="x (m)", ylabel="y (m)")
        plt.show()
```

K Code for test_All.py

```
## @file test_All.py
#  @author Hriday Jham
#  @brief A pytest file containing tests for all functions in
#  the classes CircleT, ShapeT, TriangleT, BodyT and Scene.
#  @date 02/16/2021
#  @details Written to test code.

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


class TestCircleT:

    def setup_method(self):
        self.test_circle = CircleT(1, 2, 3, 4)

    def teardown_method(self):
        self.test_circle = None

    def test_init(self):
        try:
            self.test_circle2 = CircleT(1, 2, 0, 2)
            raise ValueError("r is <= 0 and shouldve thrown an error")
        except ValueError:
            pass
        try:
            self.test_circle2 = CircleT(1, 2, 2, -2)
            raise ValueError("m is <= 0 and shouldve thrown an error")
        except ValueError:
            pass

    def test_cm_x(self):
        assert self.test_circle.cm_x() == 1

    def test_cm_y(self):
        assert self.test_circle.cm_y() == 2

    def test_mass(self):
        assert self.test_circle.mass() == 4

    def test_m_inert(self):
        assert self.test_circle.m_inert() == 18


class TestTriangleT:

    def setup_method(self):
        self.test_triangle = TriangleT(1, 2, 3, 4)

    def teardown_method(self):
        self.test_triangle = None

    def test_init(self):
        try:
            self.test_triangle2 = CircleT(1, 2, 0, 2)
            raise ValueError("s is <= 0 and shouldve thrown an error")
        except ValueError:
            pass
        try:
            self.test_triangle3 = CircleT(1, 2, 2, -2)
            raise ValueError("m is <= 0 and shouldve thrown an error")
        except ValueError:
            pass

    def test_cm_x(self):
        assert self.test_triangle.cm_x() == 1

    def test_cm_y(self):
        assert self.test_triangle.cm_y() == 2

    def test_mass(self):
        assert self.test_triangle.mass() == 4

    def test_m_inert(self):
```

```
        assert self.test_triangle.m_inert() == 3


class TestBodyT:

    def setup_method(self):
        self.test_body = BodyT([1, 1, 1], [2, 2, 2], [3, 3, 3])

    def teardown_method(self):
        self.test_body = None

    def test_init(self):
        try:
            self.test_body2 = BodyT([1, 2, 3], [4, 5, 6], [8, 9])
            raise ValueError("x, y, m cannot have variable lengths")
        except ValueError:
            pass
        try:
            self.test_body3 = BodyT([1, 2, 3], [4, 5, 6], [-7, 8, 9])
            raise ValueError("m having a value <= 0 should throw an error")
        except ValueError:
            pass

    def test_cm_x(self):
        assert self.test_body.cm_x() == 1

    def test_cm_y(self):
        assert self.test_body.cm_y() == 2

    def test_mass(self):
        assert self.test_body.mass() == 9

    def test_m_inert(self):
        assert self.test_body.m_inert() == 0


class TestScene:

    def setup_method(self):
        self.body = BodyT([1, 1, 1], [2, 2, 2], [1, 1, 1])
        self.test_scene = Scene(self.body, self.Fx, self.Fy, 0, 0)
        self.g = 9.81
        self.m = 1
        self.circle = CircleT(1, 2, 1, 1)

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

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

    def teardown_method(self):
        self.test_scene = None
        self.body = None
        self.m = None
        self.g = None

    def test_get_shape(self):
        assert self.test_scene.get_shape() == self.body

    def test_get_init_velo(self):
        x, y = self.test_scene.get_init_velo()
        assert x == 0 and y == 0

    def test_set_shape(self):
        self.test_scene.set_shape(self.circle)
        assert self.test_scene.get_shape() == self.circle

    def test_set_init_velo(self):
        self.test_scene.set_init_velo(0, 1)
        x, y = self.test_scene.get_init_velo()
        assert x == 0 and y == 1

    def test_sim(self):
        x, y = self.test_scene.sim(2, 2)
        a = [0.0, 2.0]
        b = [[1.0, 2., 0.0, 0.0], [4.333333335602872, -4.540000004452837, 3.333333333333334,
                                   -6.54]]
        temp1 = x
        temp2 = y
```

```
for i in range(len(x)):
    temp1[i] -= a[i]
assert abs(max(temp1)) / abs(max(a)) < 0.0001

for i in range(len(y)):
    for j in range(len(y[i])):
        temp2[i][j] -= b[i][j]

for i in range(len(y)):
    assert abs(max(temp2[i])) / abs(max(b[i])) < 0.0001
```

L Code for Partner's CircleT.py

```
## @file CircleT.py
#  @author Steven Kostiuk
#  @brief Contains the CircleT type to represent shapes that are circles
#  @date 2021-02-16
from Shape import Shape

## @brief CircleT is used to represent shapes that are circles


class CircleT(Shape):
    ## @brief Constructor for class CircleT
    #  @param xs X value of the center of mass
    #  @param ys Y value of the center of mass
    #  @param rs Radius of the circle
    #  @param ms Mass of the circle
    #  @throws ValueError Thrown if the mass or radius of circle is less than or equal to zero
    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 Getter method that gets the x value of the center of mass
    #  @return x value of the center of mass
    def cm_x(self):
        return self.x

    ## @brief Getter method that gets the y value of the center of mass
    #  @return y value of the center of mass
    def cm_y(self):
        return self.y

    ## @brief Getter method that gets the mass of the circle
    #  @return mass of the circle
    def mass(self):
        return self.m

    ## @brief Getter method that gets the moment of inertia of the circle
    #  @return moment of inertia of the circle
    def m_inert(self):
        return (self.m * (self.r ** 2)) / 2
```

M Code for Partner's TriangleT.py

```
## @file TriangleT.py
#  @author Steven Kostiuk
#  @brief Contains the TriangleT type to represent shapes that are triangles
#  @date 2021-02-16
from Shape import Shape

## @brief TriangleT is used to represent shapes that are triangles


class TriangleT(Shape):
    ## @brief Constructor for class TriangleT
    #  @param xs X value of the center of mass
    #  @param ys Y value of the center of mass
    #  @param rs Side of the triangle
    #  @param ms Mass of the triangle
    #  @throws ValueError Thrown if the mass or side of triangle is less than or equal to zero
    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 Getter method that gets the x value of the center of mass
    #  @return x value of the center of mass
    def cm_x(self):
        return self.x

    ## @brief Getter method that gets the y value of the center of mass
    #  @return y value of the center of mass
    def cm_y(self):
        return self.y

    ## @brief Getter method that gets the mass of the triangle
    #  @return mass of the triangle
    def mass(self):
        return self.m

    ## @brief Getter method that gets the moment of inertia of the triangle
    #  @return moment of inertia of the triangle
    def m_inert(self):
        return (self.m * (self.s ** 2)) / 12
```

N Code for Partner's BodyT.py

```
## @file BodyT.py
#  @author Steven Kostiuk
#  @brief Contains the BodyT type to represent physics bodies
#  @date 2021-02-16
from Shape import Shape

## @brief BodyT is used to represent a combination of shapes
#  that form a physics body


class BodyT(Shape):
    ## @brief Constructor for class TriangleT
    #  @param xs List of x values of the center of mass
    #  @param ys List of Y values of the center of mass
    #  @param ms List of masses
    #  @throws ValueError Thrown if the lengths of the 3 lists are not the same
    #  @throws ValueError Thrown if one of the masses in the list of masses
    #  is less than or equal to zero
    def __init__(self, xs, ys, ms):
        if not(len(xs) == len(ys) == len(ms)):
            raise ValueError
        for u in ms:
            if not(u > 0):
                raise ValueError
        self.cmx = self.__cm(xs, ms)
        self.cmy = self.__cm(ys, ms)
        self.m = self.__sum(ms)
        self.moment = self.__mmom(xs, ys, ms) - self.__sum(ms) * \
            (self.__cm(xs, ms) ** 2 + self.__cm(ys, ms) ** 2)

    ## @brief Getter method that gets the x value of the center of mass
    #  @return x value of the center of mass
    def cm_x(self):
        return self.cmx

    ## @brief Getter method that gets the y value of the center of mass
    #  @return y value of the center of mass
    def cm_y(self):
        return self.cmy

    ## @brief Getter method that gets the mass of the physics body
    #  @return mass of the physics body
    def mass(self):
        return self.m

    ## @brief Getter method that gets the moment of inertia of the physics body
    #  @return moment of inertia of the physics body
    def m_inert(self):
        return self.moment

    ## @brief Calculates the sum of a list
    #  @param m List of masses
    #  @return total mass of the physics body
    def __sum(self, m):
        result = 0
        for i in m:
            result = result + i
        return result

    ## @brief Calculates the center of mass of the physics body
    #  @param z List of coordinates that represent the x or y values of center of mass
    #  @param m List of masses
    #  @returns x or y coordinate of the center of mass of the physics body
    def __cm(self, z, m):
        result = 0
        for i in range(len(m)):
            result = result + z[i] * m[i]
        return result / self.__sum(m)

    ## @brief Calculates the first part of the moment of inertia of the physics body
    #  @param x List of x values of the center of mass
    #  @param y List of y values of the center of mass
    #  @param m List of masses
    #  @return first part of the moment of inertia of the physics body
    def __mmom(self, x, y, m):
        result = 0
```

```
for i in range(len(m)):
    result = result + m[i] * (x[i] ** 2 + y[i] ** 2)
return result
```

O Code for Partner's Scene.py

```
## @file Scene.py
#  @author Steven Kostiuk
#  @brief Contains the Scene type which is used to simulate a shape's movement
#  @date 2021-02-16
#  @details Simulates a shape's movement according to initial velocities
#  and unbalanced forces functions
from scipy.integrate import odeint

## @brief Scene is used to simulate a shape's movement


class Scene():
    ## @brief Constructor for class TriangleT
    #  @param s Shape
    #  @param Fx Unbalanced forces function for x coordinates
    #  @param Fy Unbalanced forces function for y coordinates
    #  @param vx Initial x velocity
    #  @param vy Initial y velocity
    def __init__(self, s, Fx, Fy, vx, vy):
        self.s = s
        self.Fx = Fx
        self.Fy = Fy
        self.vx = vx
        self.vy = vy

    ## @brief Getter method to get the shape
    #  @return shape
    def get_shape(self):
        return self.s

    ## @brief Getter method to get the unbalanced forces functions
    #  @return unbalanced forces functions
    def get_unbal_forces(self):
        return self.Fx, self.Fy

    ## @brief Getter method to get the initial velocities of the shape
    #  @return initial velocities
    def get_init_velo(self):
        return self.vx, self.vy

    ## @brief Setter method to set the shape
    #  @param s shape
    def set_shape(self, s):
        self.s = s

    ## @brief Setter method to set the unbalanced forces functions
    #  @param Fx Unbalanced forces function for x coordinates
    #  @param Fy Unbalanced forces function for y coordinates
    def set_unbal_forces(self, Fx, Fy):
        self.Fx = Fx
        self.Fy = Fy

    ## @brief Setter method to set the initial velocities of the shape
    #  @param vx Initial x velocity
    #  @param vy Initial y velocity
    def set_init_velo(self, vx, vy):
        self.vx = vx
        self.vy = vy

    ## @brief Simulation method that simulates the movement of a shape
    #  @param tfinal Final time which the simulation runs until
    #  @param nsteps Number of steps which the time will be divided into
    #  @return sequence of real numbers representing the time and a
    #  sequence containing 4 sequences of real numbers representing x and y values
    def sim(self, tfinal, nsteps):
        t = [(i * tfinal) / (nsteps - 1) for i in range(nsteps)]
        return t, odeint(self.__ode, [self.s.cm_x(), self.s.cm_y(), self.vx, self.vy], t)

    ## @brief Ordinary differential equation method
    #  @param w List of center of masses and initial velocities
    #  @param t List of time in natural ascending order
    #  @return List of initial velocities and the results of
    #  the unbalanced forces functions
    def __ode(self, w, t):
        return [w[2], w[3], self.Fx(t) / self.s.mass(), self.Fy(t) / self.s.mass()]
```