Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования Московский
государственный технический университет имени Н.Э. Баумана

Лабораторная работа
«Решение баллистической задачи»
по курсу
«Моделирование»

Студент группы ИУ9-82 Белогуров А.А.

Преподаватель Домрачева А.Б.

Москва, 2018

Содержание

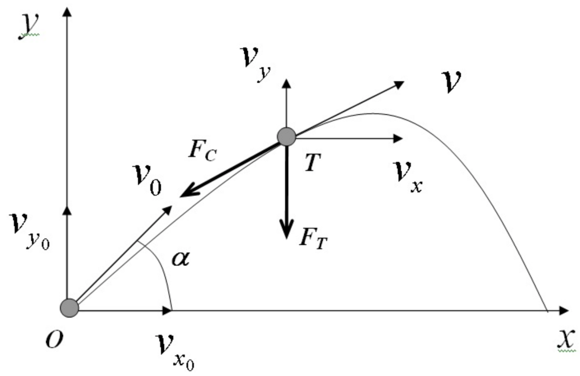
1 Постановка задачи

Смоделировать полёт баллистического снаряда методом Ньютона с учётом коэффициента сопротивления воздуха. Сравнить полученные результаты для данного метода с коэффициентом равным нулю и методом Галилея.

Даны следующие начальные данные: свинцовый шарик радиусом $r = 0.05$ м брошен под углом $\alpha = 30$ градусов с начальной скоростью $v_0 = 50$ м/с.

2 Теоретические сведения

2.1 Модель Ньютона



Главное отличие модели Ньютона от модели Галилея заключается в учитывании силы сопротивления воздуха $F_c = \beta v^2$, коэффициент которой вычисляется по формуле:

$$\beta = \frac{C\rho S}{2}, \tag{1}$$

где C - коэффициент лобового сопротивления (для многих задач баллистики $C \approx 0.15$), S – площадь поперечного сечения ($S = \pi r^2$), ρ - плотность воздуха ($\rho = 1,29$ кг/м^3).

При $\beta = 0$ модель Ньютона является частным случаем методом Галилея.

Основной идеей данного метода является решений системы дифференциальных уравнений:

$$\begin{cases} \dfrac{dv_x}{dt} = -\dfrac{\beta v_x}{m}\sqrt{v_x^2 + v_y^2} \\ \dfrac{dv_y}{dt} = -\dfrac{\beta v_y}{m}\sqrt{v_x^2 + v_y^2} - g \\ \dfrac{dx}{dt} = v_x \\ \dfrac{dy}{dt} = v_y \end{cases}, \tag{2}$$

при следующих начальных условиях:

$$\begin{cases} v_x(0) = v_0 cos(\alpha) \\ v_y(0) = v_0 sin(\alpha) \\ x(0) = 0 \\ y(0) = 0 \end{cases}. \tag{3}$$

Для решения первой системы воспользуемся методом Рунге-Кутта 4-го порядка обеспечивающий достаточно высокую точность вычислений.

2.2 Модель Галилея

Рисунок к модели Галилея будет идентичен модели Ньютона за исключением того, что не будет учтена сила сопротивления воздуха. Следовательно для вычисления конечной координаты x воспользуемся формулой:

$$x = \frac{2tan(\alpha)(cos(\alpha)v_0)^2}{g}, \tag{4}$$

где g - ускорение свободного падения.

3 Практическая реализация

Далее приведена реализация программы на языке Python 3, которая вычисляет конечную координату x для метода Галилея и методов Ньютона с учётом коэффициента сопротивления и без его учёта (Листинг 1).

```
import math

radius = 0.05
density = 11340
air_density = 1.29
v_0 = 50
g = 9.81
alpha = 30


def calc_mass(radius, density):
        """
        Calculate the mass of an object using its radius and density
        :param radius: radius of object
        :param density: density of object
        :return: mass of object
        """
        volume = 4/3 * math.pi * (radius ** 3)
        return volume * density


def calc_betta(radius, windage=0.15, air_density=1.29):
        """
        Calculate coefficient of air resistance
        :param radius: radius of object
        :param windage: windage(= 0.15)
        :param air_density: density of air
        :return: coefficient of air resistance
        """
        return windage * air_density * math.pi * (radius ** 2) / 2

```

```python

def galilee_model(speed, angle):
        """
        Calculate X-coord of Galilee model
        :param speed: start speed of object
        :param angle: start angle in degrees
        :return: x-coord
        """
        angle_rad = math.radians(angle)
        return 2 * math.tan(angle_rad) * (math.cos(angle_rad) * speed) ** 2 / g


def newton_model(speed, radius, density, angle, h=0.01, is_betta_need=True):
        """
        Calculate X-coord of Newton model using Runge-Kutta method
        :param speed: start speed of object
        :param radius: radius of object
        :param density: density of object
        :param angle: start angle in degrees
        :param h: step
        :param is_betta_need:
        :return: x-coord
        """
        mass = calc_mass(radius, density)

        if is_betta_need:
                betta = calc_betta(radius)
        else:
                betta = 0

        speed = [(calc_speed_x(speed, angle, 0), calc_speed_y(speed, angle, 0))]
        coordinates = [(0, 0)]

        while coordinates[-1][1] >= 0:
                cur_speed = speed[-1]
                cur_speed_x = cur_speed[0]
                cur_speed_y = cur_speed[1]

```

```
                k1_vx = h * calc_speed_x_der(betta, mass, cur_speed_x, cur_speed_y)
                k1_vy = h * calc_speed_y_der(betta, mass, cur_speed_x, cur_speed_y)

                k2_vx = h * calc_speed_x_der(betta, mass, cur_speed_x + k1_vx / 2, cur_speed_y + k1_vy / 2)
                k2_vy = h * calc_speed_y_der(betta, mass, cur_speed_x + k1_vx / 2, cur_speed_y + k1_vy / 2)

                k3_vx = h * calc_speed_x_der(betta, mass, cur_speed_x + k2_vx / 2, cur_speed_y + k2_vy / 2)
                k3_vy = h * calc_speed_y_der(betta, mass, cur_speed_x + k2_vx / 2, cur_speed_y + k2_vy / 2)

                k4_vx = h * calc_speed_x_der(betta, mass, cur_speed_x + k3_vx / 2, cur_speed_y + k3_vy / 2)
                k4_vy = h * calc_speed_y_der(betta, mass, cur_speed_x + k3_vx / 2, cur_speed_y + k3_vy / 2)

                cur_speed_x += (k1_vx + 2 * k2_vx + 2 * k3_vx + k4_vx) / 6
                cur_speed_y += (k1_vy + 2 * k2_vy + 2 * k3_vy + k4_vy) / 6

                speed.append((cur_speed_x, cur_speed_y))

                cur_coord = coordinates[-1]
                cur_coord_x = cur_coord[0] + h * cur_speed_x
                cur_coord_y = cur_coord[1] + h * cur_speed_y

                coordinates.append((cur_coord_x, cur_coord_y))

        return coordinates[-1][0]
```

```python
def calc_speed_x(speed, angle, t):
        """
        Calculate speed for X-axis
        :param angle: angle in degrees
        :param t: time
        :return: speed
        """
        return speed * math.cos(math.radians(angle))


def calc_speed_y(speed, angle, t):
        """
        Calculate speed for Y-axis
        :param angle: angle in degrees
        :param t: time
        :return: speed
        """
        return speed * math.sin(math.radians(angle)) - g * t


def calc_speed_x_der(betta, mass, speed_x, speed_y):
        """
        Calculate function for speed_x Runge-Kutta method
        :param betta: coefficient of air resistance
        :param mass: mass of object
        :param speed_x: speed of x-axis
        :param speed_y: speed of y-axis
        :return: function
        """
        return -betta * speed_x * math.sqrt(speed_x ** 2 + speed_y ** 2) / mass


def calc_speed_y_der(betta, mass, speed_x, speed_y):
        """
        Calculate function for speed_y Runge-Kutta method
        :param betta: coefficient of air resistance
        :param mass: mass of object
        :param speed_x: speed of x-axis
        :param speed_y: speed of y-axis
        :return: function
        """
```

```
        return -betta * speed_y * math.sqrt(speed_x ** 2 + speed_y ** 2) / mass - g


if __name__ == "__main__":
        print("Галилей\n", galilee_model(v_0, alpha))
        print("Ньютон\n", newton_model(v_0, radius, density, alpha, is_betta_need=True))
        print("Ньютон b=0\n", newton_model(v_0, radius, density, alpha, is_betta_need=False))
```

4 Результаты

Для приведённых в первом разделе начальных условий были получены следующие значения:

Модель	Координата X
$r = 0.05$ м, $\alpha = 30$, $v_0 = 50$ м/с	
Галилей	220.69964
Ньютон	216.34459
Ньютон $\beta = 0$	220.403465

Поменяем исходные данные:

Модель	Координата X
$r = 0.05$ м, $\alpha = 15$, $v_0 = 10$ м/с	
Галилей	5.09683
Ньютон	5.02115
Ньютон $\beta = 0$	5.022814
$r = 0.05$ м, $\alpha = 45$, $v_0 = 30$ м/с	
Галилей	91.74311
Ньютон	90.81862
Ньютон $\beta = 0$	91.64103
$r = 0.05$ м, $\alpha = 50$, $v_0 = 60$ м/с	
Галилей	361.39734
Ньютон	347.99881
Ньютон $\beta = 0$	361.37519

5 Вывод

В ходе выполнения лабораторной работы были изучены модели Ньютона и Галилея для решения баллистической задачи и была написана их реализация на языке Python 3.

Для решения системы дифференциальных уравнений в методе Ньютона использовался метод Рунге-Кутта 4-го порядка, который имеет четвёртый порядок точности, но стоит отметить, что так как все вычисления производились на ЭВМ, то конечный резульатат может отличаться от реального и иметь вычислительные погрешности.

Так как модель Ньютона учитывает силу сопротивление воздуха, то точка падения снаряда должна быть ближе к точке запуска, что наглядно видно в предыдущем разделе. Если же ёе не учитывать и принимать коэффициент сопротивления воздуха $\beta = 0$, то точка падения близится к точке падения модели Галилея.