Dhoni's Magnificience

Riddhiman Bhattacharya

1 Problem

In a moment that will forever be etched in cricketing history, the legendary M.S. Dhoni, at Wankhede Stadium, with nerves of steel and unwavering determination, unleashed his full power and mastery. The roaring crowd held their breath as the ball soared high into the night sky, defying gravity itself. With the weight of an entire nation's hopes on his shoulders, Dhoni's bat connected with the leather, sending it on a trajectory of glory. The ball collided with the stadium roof, releasing an electrifying surge of emotion, sealing India's momentous World Cup victory with an explosion of force. Now as a Physics Student, calculate With what force does the ball hit the roof of the stadium.

2 Solution

To determine the force (or possibly momentum, energy, etc.) with which the ball hits the roof, we first need to determine the point and velocity at which the ball hits the roof. We'll initially neglect any influences from the wind, but consider the effects of air friction on the ball as well as the force of gravity. Let $v(t) = (v_x(t), v_y(t))^T$ represent the velocity of the ball at time t, and let $\tan\alpha = \frac{v_y}{v_x}$. The air friction force F_R always acts against the direction of the ball's motion. We'll decompose this frictional force into its x and y components, and the forces F_x and F_y acting on the ball will be given by: $F_x = -F_R\cos\alpha$ and $F_y = -(F_R\sin\alpha + F_G)$, where F_G is the force of gravity.
Applying Newton's 2nd law, $F = ma$, we can find the x-component acceleration a_x of the ball (with $k = \frac{1}{2}c_w\rho A$ and $\kappa = \frac{k}{m}$, where c_w is the drag coefficient, ρ is the air density, A is the cross-sectional area of the ball, and m is the mass of the ball):

$$\begin{aligned} a_x &= -\frac{F_R\cos\alpha}{m} \\ \implies -ma_x &= k\|v\|^2\cos\alpha \\ \implies 0 &= \kappa(v_x^2 + v_y^2)\cos\arctan\frac{v_y}{v_x} + \dot{v}_x \end{aligned}$$

Similarly, for the y-component acceleration a_y:

$$\begin{aligned}
a_y &= -\frac{F_R \sin\alpha + F_G}{m} \\
\implies -ma_y &= k\|v\|^2 \sin\alpha + mg \\
\implies 0 &= \kappa(v_x^2 + v_y^2)\sin\arctan\frac{v_y}{v_x} + g + \dot{v}_y
\end{aligned}$$

Therefore, the system of differential equations to solve are:

$$\begin{aligned}
0 &= \kappa(v_x^2 + v_y^2)\cos\arctan\frac{v_y}{v_x} + \dot{v}_x \\
0 &= \kappa(v_x^2 + v_y^2)\sin\arctan\frac{v_y}{v_x} + \dot{v}_y + g
\end{aligned}$$

subject to the initial conditions $v_x(0), v_y(0) \in \mathbb{R}$.
Let,

$$h := \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} \implies \dot{h} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{pmatrix}$$

with $v_x = \dot{x}$ and $v_y = \dot{y}$. Then, we can rewrite the above initial value problem as:

$$\dot{h} = f(t,h) = f(t,(x,y,\dot{x},\dot{y})^T) = \begin{pmatrix} \dot{x} \\ \dot{y} \\ -\frac{\kappa(\dot{x}^2+\dot{y}^2)}{\sqrt{(\frac{\dot{y}}{\dot{x}})^2+1}} \\ -(\frac{\kappa\dot{y}(\dot{x}^2+\dot{y}^2)}{\dot{x}\sqrt{(\frac{\dot{y}}{\dot{x}})^2+1}} + g) \end{pmatrix}$$

3 Code

```
import matplotlib.pyplot as plt
import numpy as np
from math import exp, sqrt, pi
from functools import reduce, partial
from collections import deque
from itertools import accumulate

# Function to implement the Runge-Kutta method for solving ordinary
    differential equations
def runge_kutta(f, x0, y0, h, n):
    """
    Runge-Kutta method for solving ordinary differential equations (ODEs
        ).

    Parameters:
        f: Callable function representing the ODEs. It takes two
            arguments: t (current time) and h (state vector).
        x0: Initial time.
        y0: Initial state vector.
        h: Step size for the time domain.
        n: Number of steps to be taken.

    Returns:
        xs: List of time points.
        ys: List of state vectors corresponding to each time point.
    """
    def step(xsys, v):
        xs, ys = xsys
        xv = x0 + v * h
        yv = ys[0]
        k1 = f(xv, yv)
        k2 = f(xv + h/2, yv + k1 * h/2)
        k3 = f(xv + h/2, yv + k2 * h/2)
        k4 = f(xv + h, yv + h * k3)
        kv = h/6 * (k1 + 2 * (k2 + k3) + k4)
        y = yv + kv
        xs.appendleft(xv)
        ys.appendleft(y)
        return(xs, ys)
    return reduce(step, range(1, n+1), (deque([x0]), deque([y0])))

# Alternative implementation of Runge-Kutta method using yield and
    accumulate
def runge_kutta_alt(f, x0, y0, h, n):
    """
    Alternative implementation of Runge-Kutta method using generator and
        accumulate.

    Parameters:
        f: Callable function representing the ODEs. It takes two
           arguments: t (current time) and h (state vector).
        x0: Initial time.
        y0: Initial state vector.
        h: Step size for the time domain.
        n: Number of steps to be taken.

    Yields:
        Tuple (t, y) at each time step.
    """
    def step(xy, v):
        _, yv = xy
        x = x0 + v * h
        k1 = f(x, yv)
        k2 = f(x + h/2, yv + k1 * h/2)
        k3 = f(x + h/2, yv + k2 * h/2)
        k4 = f(x + h, yv + h * k3)
        kv = h/6 * (k1 + 2 * (k2 + k3) + k4)
        y = yv + kv
        return (x, y)
    yield from accumulate(range(1, n+1), step, initial=(x0, y0))

# Alternative implementation of Runge-Kutta method with an end time
def runge_kutta_alt2(f, x0, y0, end, n):
    """
    Alternative implementation of Runge-Kutta method using generator and
        accumulate with an end time.

    Parameters:
        f: Callable function representing the ODEs. It takes two
           arguments: t (current time) and h (state vector).
        x0: Initial time.
        y0: Initial state vector.
        end: End time for the simulation.
        n: Number of steps to be taken.

    Yields:
        Tuple (t, y) at each time step.
    """
    h = end / n
    yield from runge_kutta_alt(f, x0, y0, h, n)

# Function defining the system of ODEs describing a round object moving
```

```
    through a liquid subject to gravity
def f(kappa, t, h):
    """
    System of ODEs describing a round object moving through a liquid
        subject to gravity.

    Parameters:
        kappa: Air resistance coefficient.
        t: Current time.
        h: State vector representing the object's position and velocity.

    Returns:
        result: Array representing the object's velocity and
            acceleration at the current time.
    """
    g = 9.81
    vx, vy = h[2:]
    a = -kappa*(vx**2 + vy**2)
    b = np.arctan(vy/vx)
    return np.array([vx, vy, a * np.cos(b), a * np.sin(b) - g])

# Constants and parameters for the simulation

r = 0.145  # radius in meters
c_w = 0.4  # air resistance coefficient
m = 0.420  # weight of football in kilograms
A = r ** 2 * pi

rho_air = 1.2041  # density of air at 20 C  in kg/m
kappa = (c_w * rho_air * A) / (2 * m)

# Running the simulation using Runge-Kutta method
ts, hs = zip(*runge_kutta_alt2(partial(f, kappa), 0.0, np.array([0.0,
    0.0, 28.0, 18.0]), 3.0, 1000))
xs, ys, vxs, vys = np.transpose(np.array(hs))
ys[ys < 0] = 0

# Plotting the results
plt.subplot(2, 1, 1)
plt.plot(xs, ys, label="y(x)")
plt.xlabel("$x$")
plt.ylabel("$y(x)$")
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(ts, xs, label="$x(t)$")
plt.plot(ts, ys, label="$y(t)$")
plt.plot(ts, vxs, label=r"$v_x(t)$␣in␣$\frac{m}{s}$")
plt.plot(ts, vys, label=r"$v_y(t)$␣in␣$\frac{m}{s}$")
plt.xlabel("$t$␣in␣$s$")
plt.legend()

plt.show()
```

4 Plot

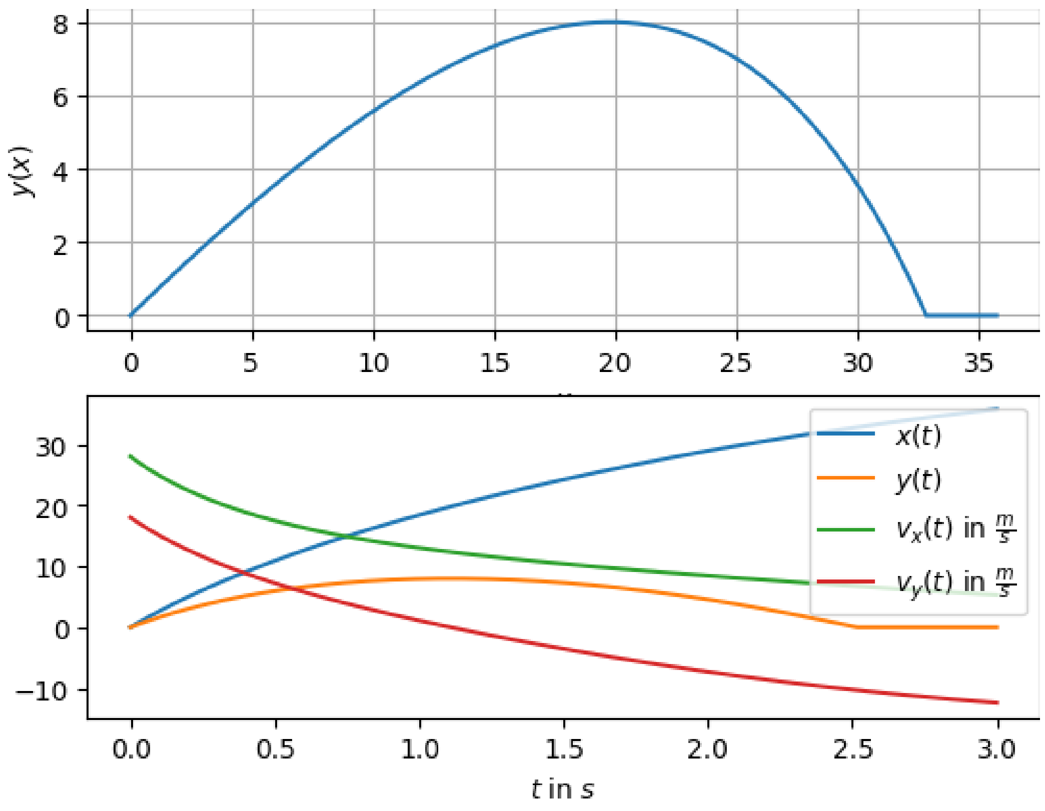

Figure 4.1: **Top Plots**: Y-axis: $y(x)$ (Vertical position of the object with respect to horizontal position x). X-axis: x (Horizontal position of the object).
Bottom Plots Blue: Y-axis: $x(t)$ (Horizontal position of the object with respect to time t). X-axis: t (Time).
Orange: Y-axis: $y(t)$ (Vertical position of the object with respect to time t). X-axis: t (Time).
Green: Y-axis: $v_x(t)$ (Horizontal velocity of the object with respect to time t) in $\frac{m}{s}$. X-axis: t (Time).
Red: Y-axis: $v_y(t)$ (Vertical velocity of the object with respect to time t) in $\frac{m}{s}$. X-axis: t (Time).