HW5. Simulation of Baseball Dynamics

2022313047 Boyeon,Kim

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```python
# Set plot params
plt.rcParams['figure.figsize'] = [5,5]
plt.rcParams['font.size'] = 15
plt.rcParams['font.family'] = 'Times New Roman'
plt.rcParams['axes.linewidth'] = 2
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['xtick.direction'] = 'out'
plt.rcParams['ytick.direction'] = 'out'
plt.rcParams['xtick.minor.visible'] = True
plt.rcParams['ytick.minor.visible'] = True
plt.rcParams['xtick.major.size'] = 7
plt.rcParams['ytick.major.size'] = 7
plt.rcParams['xtick.minor.size'] = 3.5
plt.rcParams['ytick.minor.size'] = 3.5
plt.rcParams['xtick.major.width'] = 1.5
plt.rcParams['ytick.major.width'] = 1.5
plt.rcParams['xtick.minor.width'] = 1.5
plt.rcParams['ytick.minor.width'] = 1.5
plt.rcParams['xtick.top'] = True
plt.rcParams['ytick.right'] = True
```

1. (Runge-Kutta Methods)

Solve the initial-value problem $x' = t + 2xt$ with $x(0) = 0$ on the interval [0,2] using the Runge-Kutta formulas.

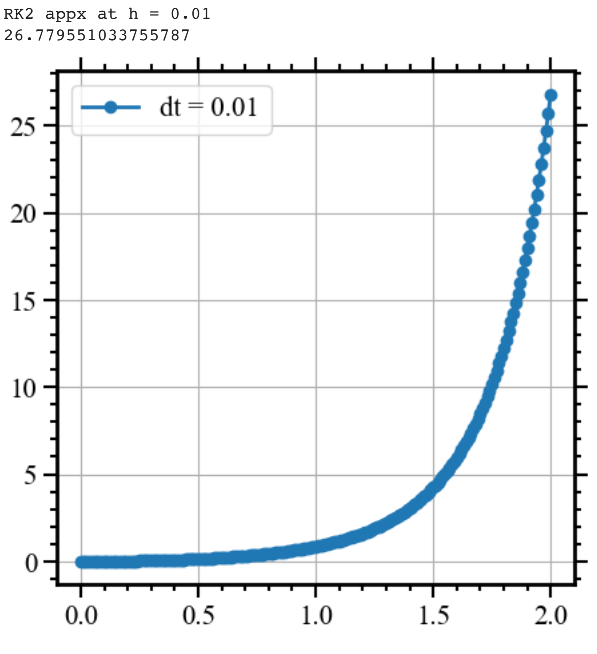
(1) Find $x(t)$ using the second-order Runge-Kutta method with $h = 0.01$.

```python
# Define the integrand
def f(t, x):
    return t + 2 * x * t
```

```python
# RK2
def RK2(f, t0, y0, tmax, dt):
    n = int((tmax - t0)/dt) + 1
    t = np.arange(t0, tmax+dt, dt)
    y = np.zeros(n)

    y[0] = y0

    #RK2
    for i in range(n-1):
        k1 = dt * f(t[i], y[i])
        k2 = dt * f(t[i] + dt/2, y[i]+k1/2)

        y[i+1] = y[i] + k2

    return t, y
```

```python
t1, y1 = RK2(f, t0 = 0, y0 = 0, tmax = 2, dt = 0.01)

print('RK2 appx at h = 0.01')
print(y1[-1])

plt.plot(t1, y1, '-o', label = 'dt = 0.01')
plt.legend()
plt.grid()
```

```
RK2 appx at h = 0.01
26.779551033755787
```

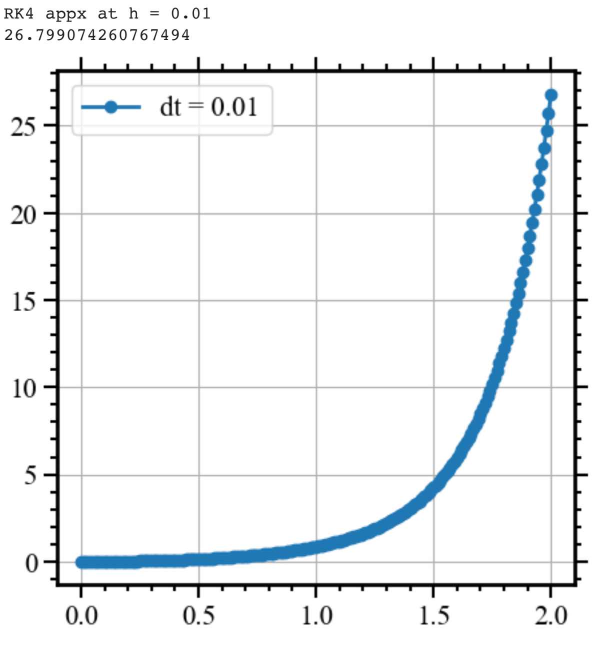


(2) Find x(t) using the fourth-order Runge-Kutta method with h = 0.01.

```python
# RK4
def RK4(f, t0, y0, tmax, dt):
    n = int((tmax - t0)/dt) + 1
    t = np.arange(t0, tmax+dt, dt)
    y = np.zeros(n)

    y[0] = y0

    #RK2
    for i in range(n-1):
        k1 = dt * f(t[i], y[i])
        k2 = dt * f(t[i] + dt/2, y[i]+k1/2)
        k3 = dt * f(t[i] + dt/2, y[i]+k2/2)
        k4 = dt * f(t[i] + dt, y[i]+k3)

        y[i+1] = y[i] + k1/6 + k2/3 + k3/3 + k4/6

    return t, y
```

```python
t2, y2 = RK4(f, t0 = 0, y0 = 0, tmax = 2, dt = 0.01)

print('RK4 appx at h = 0.01')
print(y2[-1])

plt.plot(t2, y2, '-o', label = 'dt = 0.01')
plt.legend()
plt.grid()
```

```
RK4 appx at h = 0.01
26.799074260767494
```

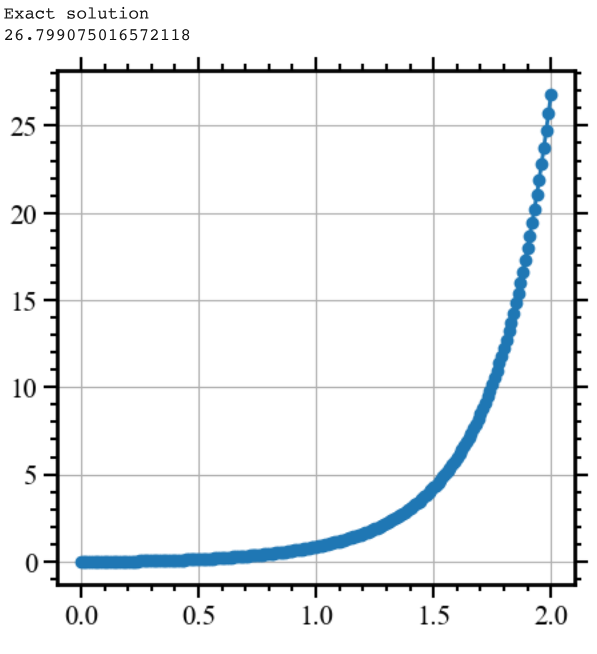


(3) Compare the solutions in (1) and (2) with the true solution: $1/2(e^{t^2} - 1)$ and discuss order of accuracy for two Runge-Kutta methods.

```python
# exact function
def exact(t):
    return 0.5 * (np.exp(t**2) - 1)
```

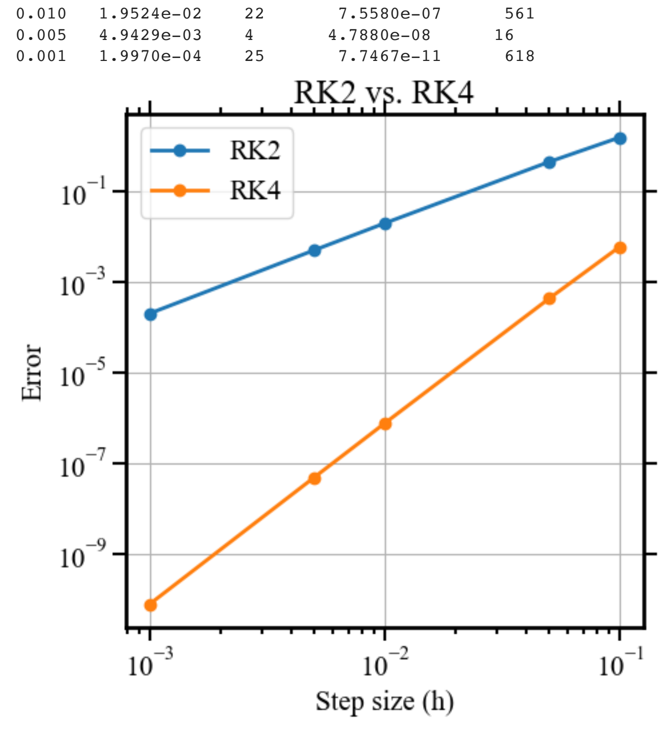
```python
# exact function values
ans = exact(t1)

print('Exact solution')
print(ans[-1])

plt.plot(t1, ans, 'o-')
plt.grid()
```

```
Exact solution
26.799075016572118
```



```python
steps = [0.1, 0.05, 0.01, 0.005, 0.001]
errs2 = []
errs4 = []
print('Error')
print('h         RK2           O(RK2)     RK4           O(RK4)')
old_err2 = np.nan
old_err4 = np.nan
for h in steps:
    t_RK2, x_RK2 = RK2(f, 0, 0, 2, h)
    t_RK4, x_RK4 = RK4(f, 0, 0, 2, h)

    err2 = np.abs(ans[-1] - x_RK2[-1])
    err4 = np.abs(ans[-1] - x_RK4[-1])

    ratio2 = old_err2/err2
    ratio4 = old_err4/err4

    errs2.append(err2)
    errs4.append(err4)

    old_err2 = err2
    old_err4 = err4

    print(f'{h:.3f}    {err2:.4e}     {ratio2:0^.0f}        {err4:.4e}       {ratio4:.0f}')

plt.plot(steps, errs2, 'o-', label='RK2')
plt.plot(steps, errs4, 'o-', label='RK4')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Step size (h)')
plt.ylabel('Error')
plt.title('RK2 vs. RK4')
plt.legend()
plt.grid()
plt.show()
```

```
Error
h          RK2          O(RK2)     RK4           O(RK4)
0.100    1.5277e+00    nan          5.9207e-03      nan
0.050    4.3919e-01    3          4.2388e-04       14
0.010    1.9524e-02    22           7.5580e-07       561
0.005    4.9429e-03    4          4.7880e-08       16
0.001    1.9970e-04    25           7.7467e-11       618
```



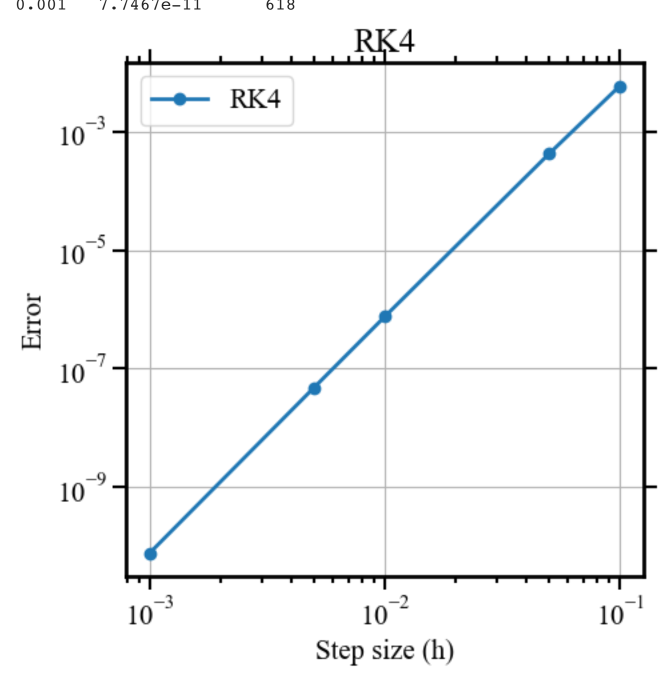
각각 2차 정확도, 4차 정확도를 보이는 것을 볼 수 있다.

(4) Discuss the effect of the step size h on the solutions by using the fourth-order Runge-Kutta method.

Hint: Compare the errors at $t = 2$ between the numerical and true solution for the different step sizes $h = 0.01, 0.05, 0.1$.

```python
steps = [0.1, 0.05, 0.01, 0.005, 0.001]
errs4 = []
print('Error')
print('h         RK4           O(RK4)')
old_err4 = np.nan
for h in steps:
    t_RK4, x_RK4 = RK4(f, 0, 0, 2, h)
    err4 = np.abs(ans[-1] - x_RK4[-1])
    ratio4 = old_err4/err4

    errs4.append(err4)
    old_err4 = err4
    print(f'{h:.3f}    {err4:.4e}       {ratio4:.0f}')

plt.plot(steps, errs4, 'o-', label='RK4')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Step size (h)')
plt.ylabel('Error')
plt.title('RK4')
plt.legend()
plt.grid()
plt.show()
```

```
Error
h          RK4          O(RK4)
0.100    5.9207e-03     nan
0.050    4.2388e-04     14
0.010    7.5580e-07     561
0.005    4.7880e-08     16
0.001    7.7467e-11     618
```



RK4의 누적 오차는 $O(h^4)$로 step size가 1/2로 줄어들면 16에 가깝게, step size가 1/5로 줄어들면 5^4인 625에 가까워 지는 것을 볼 수 있다. h의 크기가 작을 수록 계산에 대한 cost가 높아지지만 정확도 역시 4제곱만큼 더 높아진다. 즉 h의 크기가 작아질 수록 오차가 급격하게 감소한다.

2. (Baseball dynamics)

Baseball experiences a force due to gravity, a drag force due to flow resistance, and the Magnus force that causes the ball to curve as shown in Figure 1(a). For a coordinate system (x, y, z) that represents the displacement from the pitcher to catcher, the horizontal displacement, and vertical displacement from the ground, equations of baseball motion can be written as:

$$\frac{dx}{dt} = v_x$$
$$\frac{dy}{dt} = v_y$$
$$\frac{dz}{dt} = v_z$$
$$\frac{dv_x}{dt} = -F(V)Vv_x + B\omega(v_z sin\phi - v_y cos\phi)$$
$$\frac{dv_y}{dt} = -F(V)Vv_y + B\omega v_x cos\phi$$
$$\frac{dv_z}{dt} = -g - F(V)Vv_z + B\omega v_x sin\phi$$

where v_x, v_y, v_z are the velocity vector components of the baseball,$V = \sqrt{v_x^2 + v_y^2 + v_z^2}$ is the speed of the baseball. B is a dimensionless quantity that specifies the magnitude of the Magnus force, ω is the rotation rate of the baseball and π describes the direction of ω, relative to z axis. g is the gravitational acceleration ($g = 9.81m/s^2$). Note that $B = 4.1 \times 10^-4$ and $\omega = 1800rpm$. The drag force on the ball $F(V)$ is assumed as:

$$F(V) = 0.0039 + \frac{0.0058}{1+\exp[(V-35)/5]}$$

(1) Make a code for solving the above six equations using the fourth-order Runge-Kutta method. Appropriate initial conditions at $t = 0$ are $x(0) = 0, y(0) = 0, z(0) = h, v_x = v_0 cos\theta, v_y = 0, v_z = v_0 sin\theta$ where v_0 is the initial speed of the pitch, θ is the elevation angle of the pitch and h is the vertical displacement from the ground to ball release point. You need to solve the equations until the baseball reaches to the catcher $x(t) = 18.39m$. Provide a detailed procedure for making a code.

```python
# constant parameters
g = 9.81
B = 4.1e-4
# 1rpm = 2pi/60*rad
omega = 1800 * 2 * np.pi / 60
```

```python
# Define functions
def F(V):
    return 0.0039 + 0.0058 / (1 + np.exp((V - 35) / 5))

# Equations of motion
def dvx_dt(vx, vy, vz, phi):
    V = np.sqrt(vx**2 + vy**2 + vz**2)
    return -F(V) * V * vx + B * omega * (vz * np.sin(phi) - vy * np.cos(phi))

def dvy_dt(vx, vy, vz, phi):
    V = np.sqrt(vx**2 + vy**2 + vz**2)
    return -F(V) * V * vy + B * omega * vx * np.cos(phi)

def dvz_dt(vx, vy, vz, phi):
    V = np.sqrt(vx**2 + vy**2 + vz**2)
    return -g - F(V) * V * vz + B * omega * vx * np.sin(phi)
```

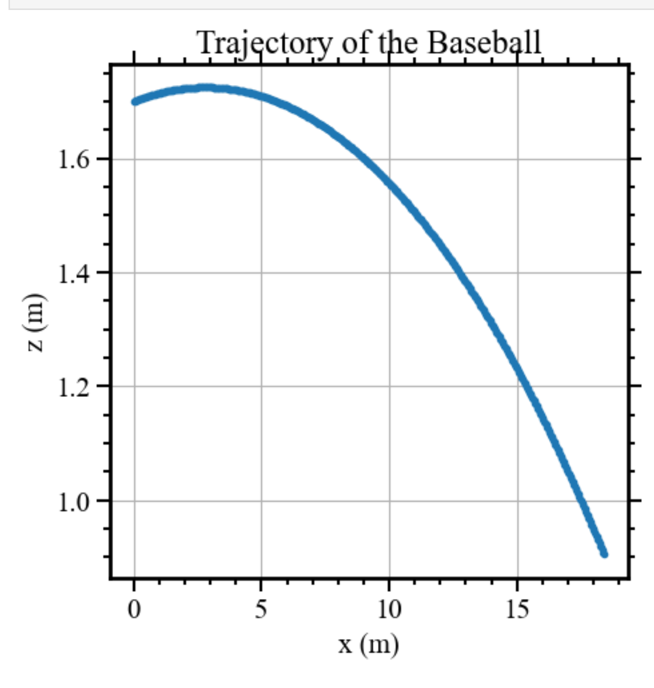
```python
# Fourth-order Runge-Kutta method
def base_RK4(x, y, z, vx, vy, vz, phi, dt):
    k1_x = vx
    k1_y = vy
    k1_z = vz
    k1_vx = dvx_dt(vx, vy, vz, phi)
    k1_vy = dvy_dt(vx, vy, vz, phi)
    k1_vz = dvz_dt(vx, vy, vz, phi)

    k2_x = vx + k1_vx * dt / 2
    k2_y = vy + k1_vy * dt / 2
    k2_z = vz + k1_vz * dt / 2
    k2_vx = dvx_dt(vx + k1_vx * dt / 2, vy + k1_vy * dt / 2, vz + k1_vz * dt / 2, phi)
    k2_vy = dvy_dt(vx + k1_vx * dt / 2, vy + k1_vy * dt / 2, vz + k1_vz * dt / 2, phi)
    k2_vz = dvz_dt(vx + k1_vx * dt / 2, vy + k1_vy * dt / 2, vz + k1_vz * dt / 2, phi)

    k3_x = vx + k2_vx * dt / 2
    k3_y = vy + k2_vy * dt / 2
    k3_z = vz + k2_vz * dt / 2
    k3_vx = dvx_dt(vx + k2_vx * dt / 2, vy + k2_vy * dt / 2, vz + k2_vz * dt / 2, phi)
    k3_vy = dvy_dt(vx + k2_vx * dt / 2, vy + k2_vy * dt / 2, vz + k2_vz * dt / 2, phi)
    k3_vz = dvz_dt(vx + k2_vx * dt / 2, vy + k2_vy * dt / 2, vz + k2_vz * dt / 2, phi)

    k4_x = vx + k3_vx * dt
    k4_y = vy + k3_vy * dt
    k4_z = vz + k3_vz * dt
    k4_vx = dvx_dt(vx + k3_vx * dt, vy + k3_vy * dt, vz + k3_vz * dt, phi)
    k4_vy = dvy_dt(vx + k3_vx * dt, vy + k3_vy * dt, vz + k3_vz * dt, phi)
    k4_vz = dvz_dt(vx + k3_vx * dt, vy + k3_vy * dt, vz + k3_vz * dt, phi)

    x_new = x + (k1_x + 2 * k2_x + 2 * k3_x + k4_x) * dt / 6
    y_new = y + (k1_y + 2 * k2_y + 2 * k3_y + k4_y) * dt / 6
    z_new = z + (k1_z + 2 * k2_z + 2 * k3_z + k4_z) * dt / 6
    vx_new = vx + (k1_vx + 2 * k2_vx + 2 * k3_vx + k4_vx) * dt / 6
    vy_new = vy + (k1_vy + 2 * k2_vy + 2 * k3_vy + k4_vy) * dt / 6
    vz_new = vz + (k1_vz + 2 * k2_vz + 2 * k3_vz + k4_vz) * dt / 6

    return x_new, y_new, z_new, vx_new, vy_new, vz_new
```

```python
# Main loop
def traj(x0, y0, z0, v0, theta, phi, dt):
    """ x0 : initial x
        y0 : initial y
        z0 : initial z
        v0 : intiial velocity
        theta : the elevation angle of the pitch
        phi : Rotation direction """

    x, y, z = x0, y0, z0
    vx, vy, vz = v0 * np.cos(theta), 0, v0 * np.sin(theta)
    x_values, y_values, z_values = [x], [y], [z]

    # x(t) = 18.39m에 도달할 때 까지 loop
    while x < 18.39:
        x, y, z, vx, vy, vz = base_RK4(x, y, z, vx, vy, vz, phi, dt)
        # z(t) 위치는 0 미만은 없으므로!
        if z <= 0:
            z = 0
        x_values.append(x)
        y_values.append(y)
        z_values.append(z)

    return x_values, y_values, z_values
```

Check the code

```python
# Initial conditions from 2(2)
v0 = 40
# 1rad = 1*pi/180
theta = 1 * np.pi / 180
h = 1.7
x0, y0, z0 = 0, 0, h
phi = 0
dt = 0.001

x_values, y_values, z_values = traj(x0, y0, z0, v0, theta, phi,dt)
```

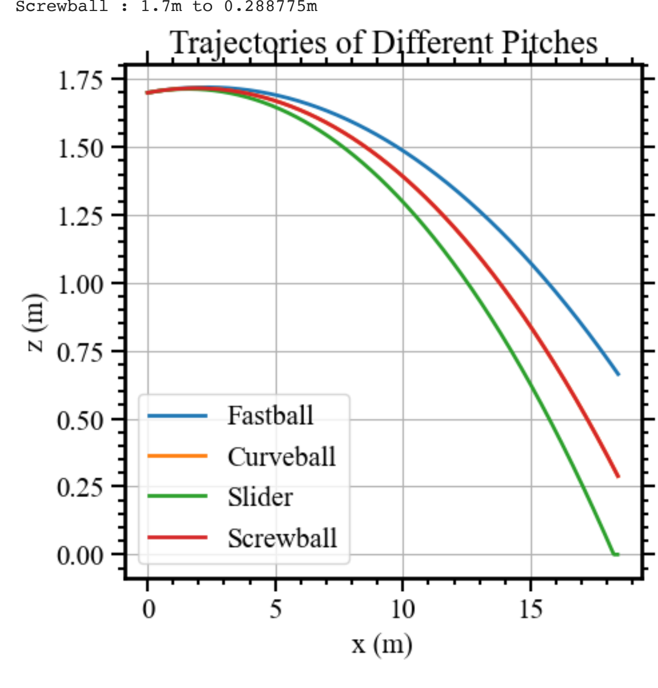
```python
# Plot
plt.plot(x_values, z_values, '.-')
plt.xlabel('x (m)')
plt.ylabel('z (m)')
plt.title('Trajectory of the Baseball')
plt.grid()
plt.show()
```



(2) Solve the equations for four pitches shown in Figure 1(b) and plot trajectories of the baseball for each pitches. Typical elevation angle is $\theta = 1^\circ$, the initial speeds of the fastball and others are $v_0 = 40m/s$ and $v_0 = 30m/s$, respectively. Rotation directions (ϕ) are $225^\circ, 45^\circ, 0^\circ$ and 135° for the fastball, curveball, slider and screwball, respectively. Note that $h = 1.7m$.

```python
# Initial conditions
h = 1.7
# 1도 = 1*pi/180
theta = 1 * np.pi / 180

# Fastball, Curveball, Slider, and Screwball initial conditions
pitches = {
    "Fastball": {"v0": 40, "phi": 225 * np.pi / 180},
    "Curveball": {"v0": 30, "phi": 45 * np.pi / 180},
    "Slider": {"v0": 30, "phi": 0 * np.pi / 180},
    "Screwball": {"v0": 30, "phi": 135 * np.pi / 180},
}

# 각 방법에 대한 trajectory loop
print('Final positions')
x_valuess = []
y_valuess = []
z_valuess = []
for pitch_name, pitch in pitches.items():
    x_values, y_values, z_values = traj(x0, y0, z0, pitch["v0"], theta, pitch["phi"], dt)

    # for each plot
    x_valuess.append(x_values)
    y_valuess.append(y_values)
    z_valuess.append(z_values)

    # Final position and Plot
    print(f'{pitch_name} : {z_values[0]:}m to {z_values[-1]:.6f}m')
    plt.plot(x_values, z_values, label=pitch_name)

plt.xlabel('x (m)')
plt.ylabel('z (m)')
plt.title('Trajectories of Different Pitches')
plt.legend()
plt.grid()
plt.show()
```

```
Final positions
Fastball : 1.7m to 0.664163m
Curveball : 1.7m to 0.288775m
Slider : 1.7m to 0.000000m
Screwball : 1.7m to 0.288775m
```




```python
fig= plt.figure()
ax=fig.add_subplot(111, projection='3d')
ax.plot(x_valuess[0],y_valuess[0],z_valuess[0], label = 'Fastball')
ax.plot(x_valuess[1],y_valuess[1],z_valuess[1], label = 'Curveball')
ax.plot(x_valuess[2],y_valuess[2],z_valuess[2], label = 'Slider')
ax.plot(x_valuess[3],y_valuess[3],z_valuess[3], label = 'Screwball')
ax.view_init(20,60)
ax.set_xlabel('x(m)')
ax.set_ylabel('y(m)')
ax.set_zlabel('z(m)')
ax.legend(loc = 'right')
plt.title('Trajectories of Different Pitches')
plt.show()
```