Computational Physics Weekly Assessments

Week 4: ODE part 2

Weekly assessment task & hints

This week you will write a program to study the trajectory of a spherical iron cannonball in the presence of gravity and drag forces, and determine the launch angle for maximum range to within ±5°. **You write your own code using euler's method, then compare results from this with scipy.integrate**

1. Create a module, named '`cp_4.py`'

2. **In the global scope off the module create the following variables:**

```
r        - radius of the cannonball in meters          0.15
rho_iron - density of iron in kg m^-3               7874.00
g        - acceleration due to gravity in ms^-2         9.81
kappa    - drag coefficient of a sphere                 0.47
rho_air  - density of air in kg m^-3                    1.23
t1       - end time for our ODE integration in s       25.00
v0       - launch speed in m s^-1                     125.00
n_panels - the number of panels to use               400
```

Then calculate the following variables from them:

```
area     - cross sectional area of the cannon ball in m^2
mass     - mass of the cannonball in kg
```

3. Create a function `f((x, y, vx, vy),time)` that implements the differential equation for the cannonball:

- The function should accept two values, the first is a numpy array representing the cannonball's state (x and y position, x and y components of velocity) and the second is time – note that time doesn't factor in to our DEQ, but we must accept it to be compatible with scipy.integrate.odesolve. The function should return a four element numpy array of (dx/dt, dy/dt, dvx/dt and dvy/dt)
- You should implement the forces due to gravity and atmospheric drag as *mg* and *kappa.rho.area.v*^{2}.

4. Create a function `solve_euler(state, t1, n_panel)` that solves the above DEQ with the method:

- state is a numpy array representing the initial conditions (x0, y0, vx0, vy0).
- When working with a numpy array of quantities instead of a single variable you can still use the same commands, e.g. `state = state + f(state, t0)*dt`.
- Again, this function should record all state values (x, y etc.) at each timepoint for later display. As with last week we will allocate a numpy array for this, although this week it needs to be a 2D array – see the example code.

5. Create a timebase for use by the odeint function.

- These are arbitrarily spaced but increasing points in time to which odeint will evaluate your DEQ.
- Use `timebase = numpy.arange(0, t1, t1/n_panels)` so that odeint uses the same timestep as your Euler method

6. Invoke both `solve_euler` and `scipy.integrate.odeint` to solve your differential equation for a range of initial conditions corresponding to different launch angles from an initial position of (0,0)

- Initialise a list for the range of each trajectory, `proj_range = []`
- See the pendulum example from the lecture notes
- Generate a range of initial launch angles to explore, called `thetas`
    - Simulate at every 5° between horizontal (0°) and vertical (90°)
    - `thetas = range(5,90,5)`
- Use a for loop to loop over each angle and generate the initial conditions – you will need to use the launch angle to resolve the initial speed (`v0`) into its vector form (`vx, vy`)
- For each trajectory:
    - **Trim the trajectories -** the trajectories generated by your solver and odeint don't stop when the cannonball hits the ground, as defined by y=0. Indeed, the cannonball will continue to fall until the allocated time is exhausted. Use the trim trajectory function (supplied, see example code below) to trim the trajectories so that only the part before collision with the ground remains.
    - **Plot the trimmed trajectory -** pyplot.subplot(211) – see last weeks assessment for subplots. Plot the odeint trajectories as grey lines. Plot the trajectories from your Euler solver as dashed blue lines.
    - **Calculate the range of the projectile**. Only do this for the odeint method. This is the distance between the launch point and the point where it hits the ground – i.e. the first and last points in the trimmed trajectory. Append the range to the '`proj_range`' list.

7. Plot range vs launch angle. `Use pyplot.subplot(212) for this.`

8. Answer the following questions, using your simulation:

ANSWER1 = What is the angle from the horizontal for maximum range under these conditions? (i.e. horizontal is 0°)

ANSWER2 = How does this angle change with increasing air density?

General comments:

- Place two variables at the start of the code,
  - USER=”your name”
  - USER_ID = “your CIS login”
- Test your final code before submission to ensure that it works
- Your solution should not be excessively long.
- A partial example is shown below

```
from __future__ import division
import numpy
import scipy.integrate
import matplotlib.pyplot as pyplot

# Define all your constants here
# r, area, rho_air, rho_iron, g, kappa, v0, t1, n_panels
# Calculate mass and area (of cannonball of radius r, of iron) here

def f((x, y, vx, vy), t):
    # Calculate the forces  on our cannonball
    Fx_grav = ??  # Gravity, x-component
    Fy_grav = ??  # Gravity, y-component
    Fx_drag = ??? # Fluid resistance, x-component
    Fy_drag = ??? # Fluid resistance, y-component
    d_x =  ???    # dx/dt
    d_y =  ???    # dy/dt
    d_vx = ???    # dvx/dt (acceleration)
    d_vy = ???    # dvy/dt
    return numpy.array((d_x, d_y, d_vx, d_vy))

def solve_euler(x, t1, n_panels):
    # Allocate somewhere to store (x,y,vx,vy) at all timepoints
    history = numpy.zeros((n_panels, len(x)))
    ???

pyplot.figure()
timebase = numpy.arange(0, t1, t1/n_panels)

def trim_trajectory(values):
    # Process a trajectory to terminate it when it goes below y=0
    for i in range(len(values)-1):
        x0, y0, vx0, vy0 = values[i]
        x1, y1, vx1, vy1 = values[i+1]
        if y0 < 0: return values[:i]
    return values

proj_range = []          # Range corresponding to each angle
thetas = range(5, 90, 5) # Angles to explore, in degrees
pyplot.subplot(211)      # full width, half height, on top

for theta in thetas:
    vx, vy = ???, ???
    initial_conditions = (0, 0, vx, vy)

    values_scipy = scipy.integrate.odeint(f, initial_conditions, timebase)
    values_euler = solve_euler(initial_conditions, t1, n_panels)
    values_scipy = trim_trajectory(values_scipy)
    values_euler = trim_trajectory(values_euler)
    # Calculate the range
    x_first, y_first, vx_first, vy_first = values_euler[0]
    x_final, y_final, vx_final, vy_final = values_euler[-1]
    rnge = ??? # Note the munged name, as 'range' is a Python keyword
    proj_range.append(rnge)
    # Plot the odeint trajectory - grey line
    # Plot the Euler trajectory - blue dashed line

pyplot.subplot(212)
# Plot range vs theta
pyplot.show()
```

Ln: 58 Col: 20