CEE 6736: HW #1

Hanfeng Zhai
Mechanical Engineering, Cornell University
hz253@cornell.edu

September 6, 2022

[2]:
```
# !sudo apt install cm-super dvipng texlive-latex-extra texlive-latex-recommended
```

Please show all work (i.e. include source code, and include thoughtful, analytical discussions):

(1) Use Python to draw the "vector plot" (i.e. slopes within the velocity-time space) for the drone free fall ODE that we derived in class (Lesson 3…we sketched out the idea in there). Assume a mass, $m = 10kg$ and drag coefficient, $\gamma = 2\frac{kg}{sec}$. See the web link, under "Homework" within the course website, for guidance on how to draw the vector field plot using *Python* and *matplotlib*. Discuss the results.

The easiest way to create the Python environment for this course is to install Anaconda (see web link under "Software resources" on course website)

Solution:

According to the lecture notes, the governing equation of a pendulum follows (assuming g possesses a positive acceleration, a negative sign are added before it):

$$\frac{dv(t)}{dt} = -g - \frac{\gamma}{m}v \tag{1}$$

where

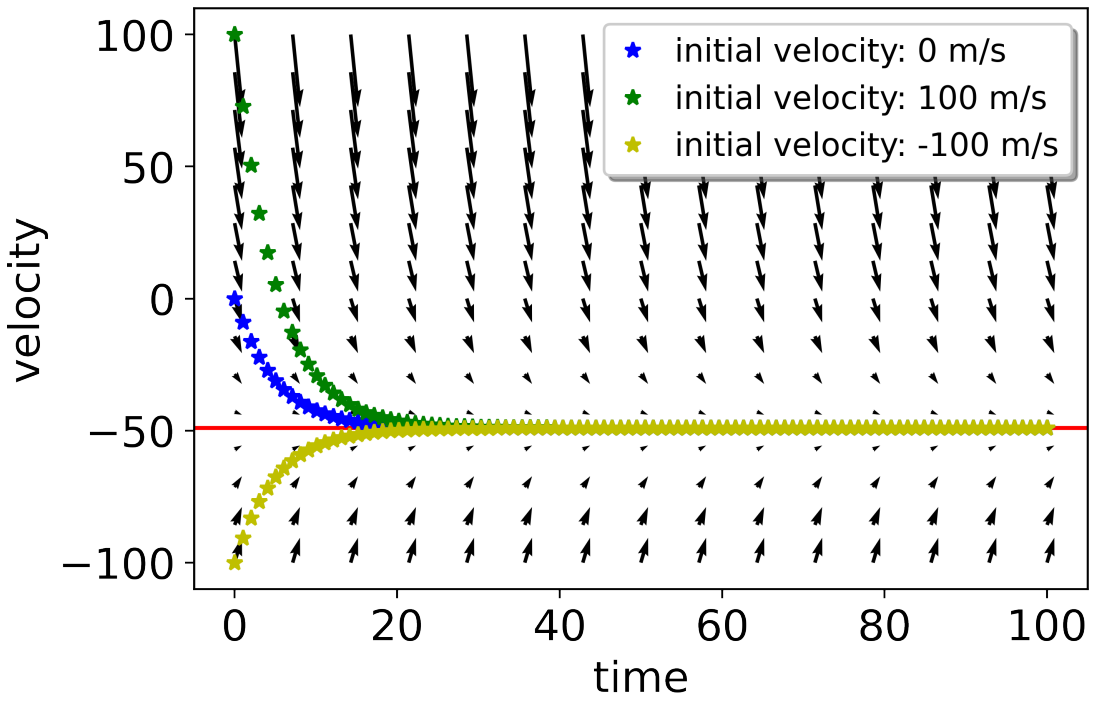
$$\begin{aligned} \gamma &\equiv \text{drag of coefficient} \\ \gamma v &\equiv \text{wind of resistance} \\ m &\equiv \text{mass of quad copter} \end{aligned} \tag{2}$$

By just observing the equation one can discern that when $\dot{v}(t) = 0$, $v_{eq} = 49\text{m/s}$, in which we denote v_{eq} as the equilibrium velocity. By plotting the vector field without solving the equation and mark the equilibrium velocity we observe that the field cnverges to this value.

Further, solving this equation using *Python*, we employ the ODE solver in `scipy`. We first define a function then apply the `odeint` for solution visualizations. Visualizing numerical solutions we observe that the solutions with different ICs also converge to the equilibrium velocity.

[8]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy.integrate import odeint
```

```
t,v = np.meshgrid(np.linspace(0,100,15),np.linspace(-100,100,15)) # generate mesh

# define parameters
gamma = 2
m = 10

# define equations
drag = (gamma/m) * v
g = 9.8
RHS = - g - drag

# define accurate form of the ODE
def drone_freefall(v, t):
  dvdt = - g - (gamma/m) * v
  return dvdt

# solving the equation(s) using odeint
t_dis = np.linspace(0,100,100)
v0_1 = 0
v0_2 = 100
v0_3 = -100
v_sol_1 = odeint(drone_freefall, v0_1, t_dis)
v_sol_2 = odeint(drone_freefall, v0_2, t_dis)
v_sol_3 = odeint(drone_freefall, v0_3, t_dis)

# plotting
plt.quiver(t,v,3,RHS)
plt.axhline(y=-49, color='r', linestyle='-')
plt.plot(t_dis, v_sol_1, 'b*', label="initial velocity: 0 m/s")
plt.plot(t_dis, v_sol_2, 'g*', label="initial velocity: 100 m/s")
plt.plot(t_dis, v_sol_3, 'y*', label="initial velocity: -100 m/s")
plt.xlabel('time')
mpl.rcParams.update({'font.size': 16})
plt.ylabel('velocity')
plt.legend(shadow=True, handlelength=1, fontsize=12)
plt.rcParams['figure.dpi'] = 500
plt.show()
plt.figure(figsize=(5, 3))
```



[8]: <Figure size 2500x1500 with 0 Axes>

<Figure size 2500x1500 with 0 Axes>

More physical intuition regarding this plot is that no matter which initial velocity the object was assigned (or given), it will always converge to a fixed velocity value. However, the converging times are different regarding different initially set velocities. One **Important Point** to note is that the horizontal-directional (i.e. time) portion of the acceleration should be a constant since there is no time term in the RHS of Equation (1). However, selecting the appropriate constant regarding time "direction" is crucial. Based on the fitting with the numerical solution of the colored dots, we think "3" is a decent value. Also, we observe that at a fixed velocity the slope remains the same, as the value can be calculated from Equation (1).