F_3_G_2

April 18, 2024

0.1 Formative 3

0.2 Group members:

1. Minghe SHI
2. Xiaodelong CAI
3. Laize KAI

[94]:
```
from sympy import Symbol, Function, dsolve, lambdify, simplify, Eq,␣
↪classify_ode, pi
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

0.3 The general functions

$$m\ddot{x} = mg - kx - c\dot{x} \tag{1}$$
$$x_{ss} = \frac{mg}{k} \tag{2}$$

0.4 Set default parameter values

[95]:
```
## system parameter variables
m = Symbol('m')
g = Symbol('g')
k = Symbol('k')
c = Symbol('c')
x = Symbol('x')
H = Symbol('H')
# independent variable
t = Symbol('t')
```

[96]:
```
# general differential functions
x = Function('x')(t)
xdot = x.diff(t)
xddot = xdot.diff(t)
```

0.5 General ODE using SymPy

[97]:
```
# general ODE
expr = Eq(m*xddot, m*g - k*x - c*xdot)
display(expr)
```

$$m\frac{d^2}{dt^2}x(t) = -c\frac{d}{dt}x(t) + gm - kx(t)$$

[98]:
```
# substitute parameter values
m = 85
k = 37
c = 10
H = 40
g = 9.8
```

0.6 Specific ODE with chosen parameter values

[99]:
```
# specific ODE with parameter values
expr = Eq(m*xddot, m*g - k*x - c*xdot)
display(expr)
```

$$85\frac{d^2}{dt^2}x(t) = -37x(t) - 10\frac{d}{dt}x(t) + 833.0$$

0.7 Specific solution using initial conditions

[100]:
```
x_0 = 0
xdot_0 = 0
print(f"initial conditions: x_0 = {x_0}, xdot_0 = {xdot_0}")
result = dsolve(expr, x, hint="factorable", ics={x.subs(t, 0): x_0, xdot.
  ↪subs(t, 0): xdot_0})
result.rhs
```

```
initial conditions: x_0 = 0, xdot_0 = 0
```

[100]:

$$(-2.01528273673935\sin(0.657140707942068t) - 22.5135135135135\cos(0.657140707942068t))\,e^{-0.0588235294117647t} + 22.5135135135135$$

0.8 Create data points and plot

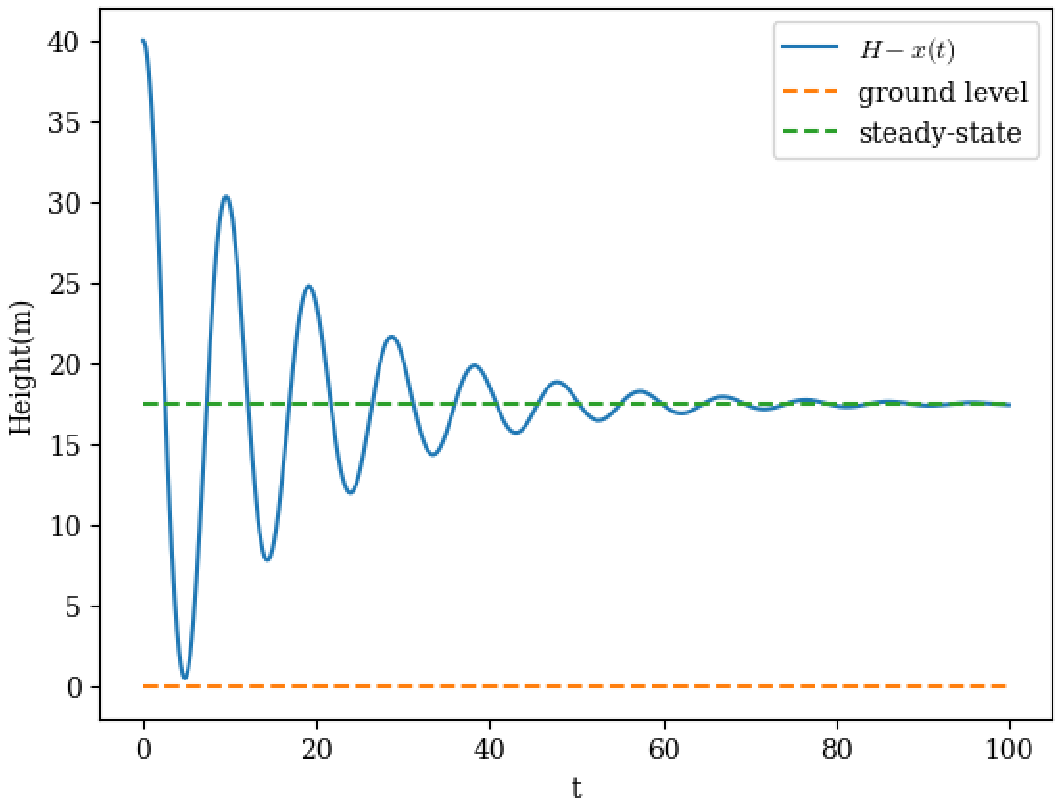
[101]:
```
# create a lambda function in order to compute data points
f = lambdify(t, result.rhs, 'numpy')
```

[102]:
```
# calculate steady-state
x_ss = H-m*g/k
```

[103]:
```
x_t = 0
```

[104]:

```
# create data points and basic plot
t = np.linspace(0, 100, 1000)
plt.plot(t, H-f(t), "-", label="$H-x(t)$")
plt.rcParams['font.family'] = 'serif'
plt.rcParams['mathtext.fontset'] = 'cm'
plt.xlabel('t')
plt.ylabel('Height(m)')
plt.plot([0, np.max(t)], [x_t, x_t], "--", label="ground level")
plt.plot([0, np.max(t)], [x_ss,x_ss],"--", label="steady-state")
plt.legend()
plt.savefig("pendulum.png")
plt.show()
```



[105]:

```
# save the data to a CSV file
df = pd.DataFrame({'t': t, 'y':H-f(t) })
display(df)
df.to_csv('data_theoretical.csv', index_label='i')
```

	t	y
0	0.0000	40.000000
1	0.1001	39.951112

```
2       0.2002  39.805423
3       0.3003  39.564698
4       0.4004  39.231094
..         …         …
995    99.5996  17.433664
996    99.6997  17.431666
997    99.7998  17.429929
998    99.8999  17.428457
999   100.0000  17.427255

[1000 rows x 2 columns]
```

1 member2

[106]:
```python
import numpy as np
import pandas as pd

def find_zero_crossings(data, time):
    zero_crossings = []
    data_shifted = np.array(data) - np.mean(data)
    sign_changes = np.sign(data_shifted[:-1]) != np.sign(data_shifted[1:])
    indices = np.where(sign_changes)[0]
    for i in indices:
        zero_crossings.append(time[i] - data_shifted[i] * (time[i+1] - time[i])␣
↪/ (data_shifted[i+1] - data_shifted[i]))
    return zero_crossings
```

[107]:
```python
def find_turning_points(data, time):
    turning_points = []
    data_diff = np.diff(data)
    extremes = np.diff(np.sign(data_diff))
    indices = np.where(extremes != 0)[0] + 1
    for i in indices:
        turning_points.append(time[i])
    return turning_points
```

[ ]:
```python

```