Solving Methods

September 27, 2023

[1]:
```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import control as ct  # If not installed --> pip install control

plt.style.use('../maroon_ipynb.mplstyle')
```

Contents

1 Problem

Solve $25\ddot{x}_0 + 5\dot{x}_0 + 150x_0 = 100e^{-5t}$ using several different ways. The initial conditions are zero.

1.1 Analytical Solution

[2]:
```python
t = sp.Symbol('t')
x0 = sp.Function('x_0')(t)
eq = sp.Eq(25*x0.diff(t, 2) + 5*x0.diff() + 150*x0, 100*sp.exp(-5*t))
eq
```

[2]:

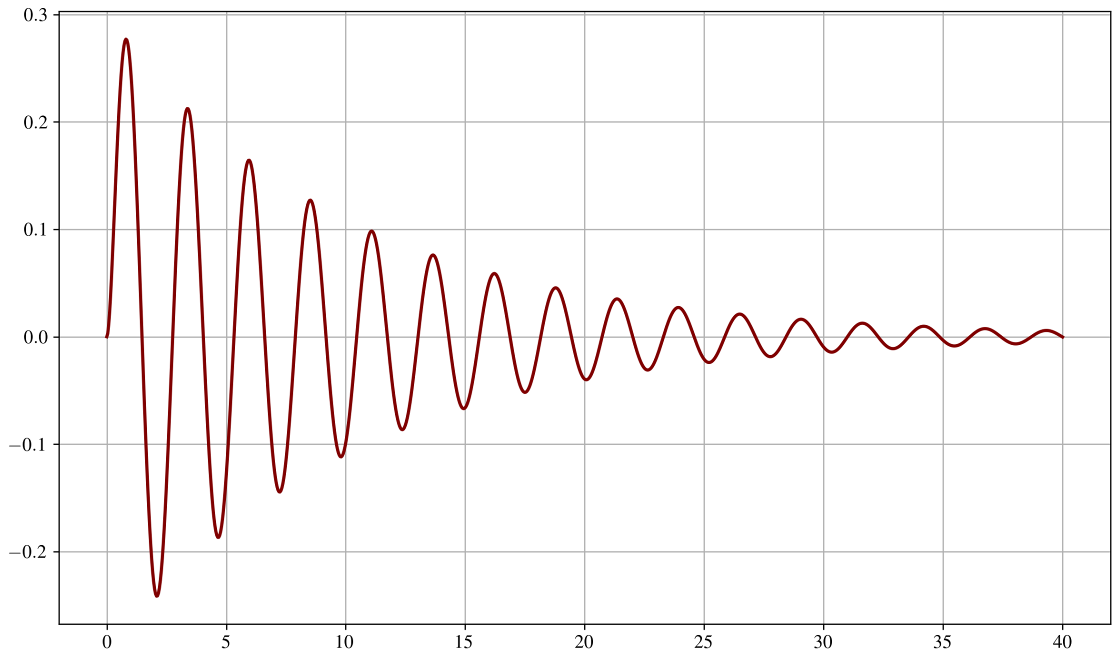
$$150x_0(t) + 5\frac{d}{dt}x_0(t) + 25\frac{d^2}{dt^2}x_0(t) = 100e^{-5t}$$

[3]:
```python
sol = sp.dsolve(eq, ics={
    x0.subs(t, 0): 0,
    x0.diff().subs(t, 0): 0
})
sol
```

[3]:

$$x_0(t) = \left(\frac{98\sqrt{599}\sin\left(\frac{\sqrt{599}t}{10}\right)}{8985} - \frac{2\cos\left(\frac{\sqrt{599}t}{10}\right)}{15}\right)e^{-\frac{t}{10}} + \frac{2e^{-5t}}{15}$$

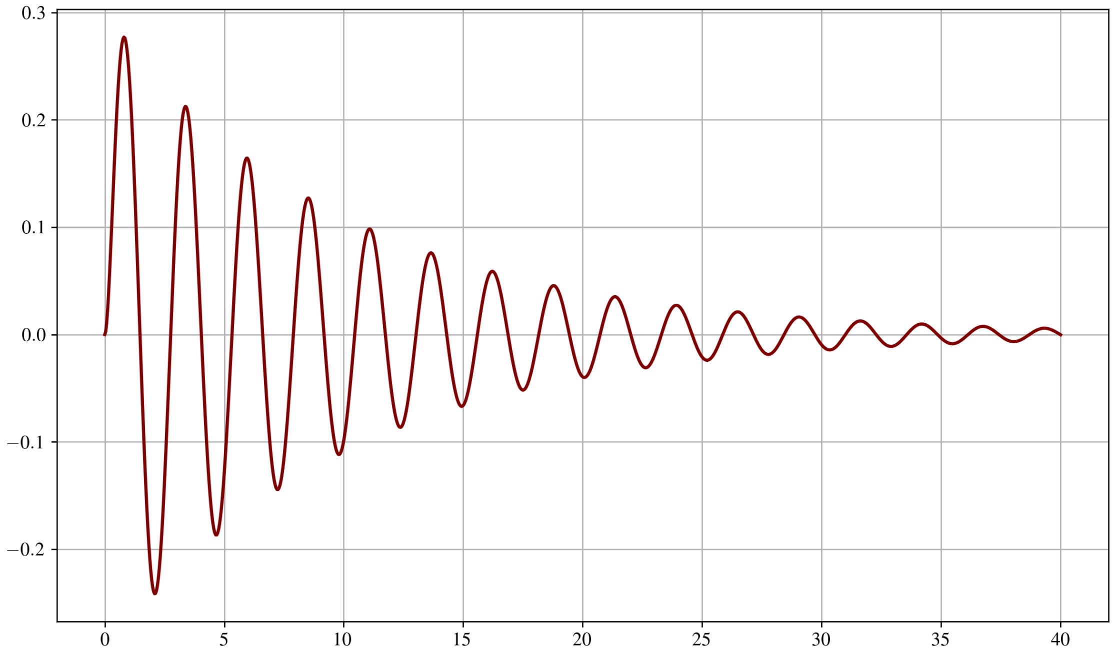
[4]:
```python
x0_lamb = sp.lambdify(t, sol.rhs, modules='numpy')
x1_lamb = sp.lambdify(t, sol.rhs.diff(), modules='numpy')
time_array = np.linspace(0, 40, 1000)
plt.plot(time_array, x0_lamb(time_array))
plt.show()
```



1.2 State Variable Solution

[5]:
```
def state_vars(x, t_):
    return [
        x[1],
        1/25*(100*np.exp(-5*t_) - 150*x[0] - 5*x[1])
    ]

sol = odeint(state_vars, (0, 0), time_array)
x_0 = sol[:, 0]
plt.plot(time_array, x_0)
plt.show()
```



1.3 Transfer Function Solution

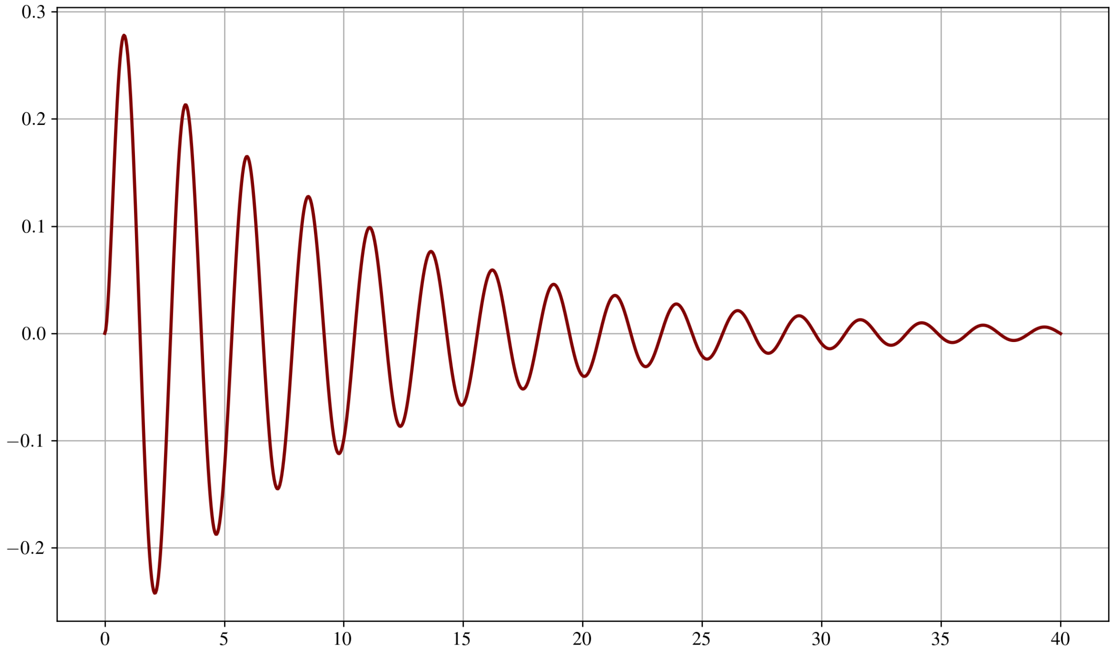
[6]:
```
sys1 = ct.tf(1, [25, 5, 150])
sys1
```

[6]:

$$\frac{1}{25s^2 + 5s + 150}$$

[7]:
```
_, x_0 = ct.forced_response(sys1, T=time_array, U=100*np.exp(-5*time_array))  #␣
↪returns a tuple containing the time array and the response
```

```
plt.plot(time_array, x_0)
plt.show()
```



1.4 State Space Model

If your model has initial conditions, and you would like to use the control package, use the state space model over the transfer function model.

[8]:
```
A = [
    [0, 1],
    [-6, -0.2]
]

B = [
    [0],
    [1/25]
]

C = [
    [1, 0],
    [0, 1]
]

D = [
    [0],
```

```
    [0]
]

sys1 = ct.ss(A, B, C, D)
sys1
```

[8]:

$$\left(\begin{array}{cc|c} 0 & 1 & 0 \\ -6 & -0.2 & 0.04 \\ \hline 1 & 0 & 0 \\ 0 & 1 & 0 \end{array}\right)$$

[9]:

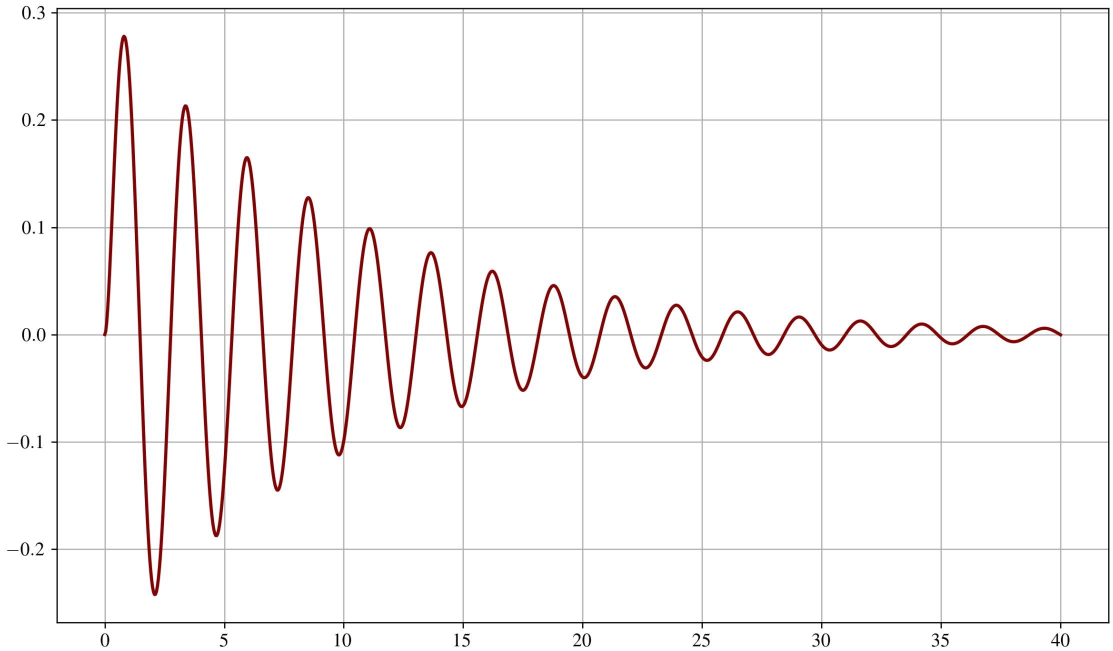
```
_, sol = ct.forced_response(sys1, T=time_array, U=100*np.exp(-5*time_array))  #␣
↪returns a tuple containing the time array and the response
plt.plot(time_array, sol[0])

# Velocity
# vel_ax = plt.twinx()
# vel_ax.plot(time_array, sol[1], color='black', ls='--')
# vel_ax.plot(time_array, x1_lamb(time_array))
# vel_ax.grid(False)

# For initial conditions if x(0) = 0.1 and x_dot(0) = 0
# _, x_free = ct.initial_response(sys1, T=time_array, X0=(0.1, 0))
# plt.plot(time_array, sol[0] + x_free[0])

# or you can do initial conditions all at once rather than doing forced␣
↪response + free response
# _, sol = ct.forced_response(sys1, T=time_array, U=100*np.exp(-5*time_array),␣
↪X0=(0.1, 0))
# plt.plot(time_array, sol[0])

plt.show()
```


0.3
0.2
0.1
0.0
−0.1
−0.2
0
5
10
15
20
25
30
35
40