MAE 3403 – Solving Differential Equations

Oklahoma State University
Stillwater, Oklahoma

Introduction to State Variable Dynamic Models
and
Solving ODE's with Python

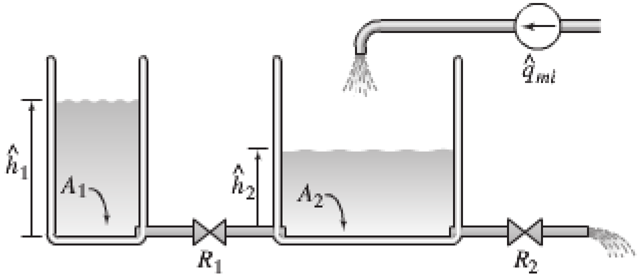
State-Variable Modeling

- Converting a coupled set of one or more ODE's into a set of Coupled First order ODE's

- Two coupled First Order ODE's

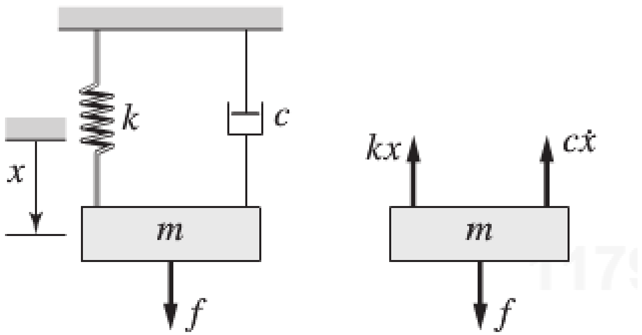
$$A_1 \frac{dh_1}{dt} = -\frac{g}{R_1}(h_1 - h_2)$$

$$\rho A_2 \frac{dh_2}{dt} = q_{mi} + \frac{\rho g}{R_1}(h_1 - h_2) - \frac{\rho g}{R_2} h_2$$



- One Second Order ODE (F = m a)

$$m\ddot{x} + c\dot{x} + kx = f$$

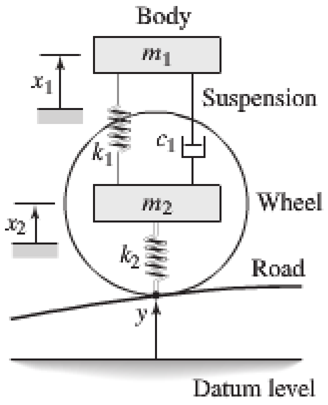


State-Variable Modeling

- Two Coupled Second Order ODE's

$$m_1\ddot{x}_1 = c_1(\dot{x}_2 - \dot{x}_1) + k_1(x_2 - x_1)$$

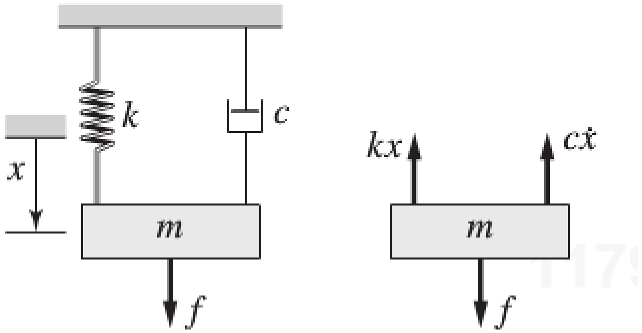
$$m_2\ddot{x}_2 = -c_1(\dot{x}_2 - \dot{x}_1) - k_1(x_2 - x_1) + k_2(y - x_2)$$



State-Variable Modeling

- Start with the single Second Order ODE

$$m\ddot{x} + c\dot{x} + kx = f$$



Determine the number of states needed - 2

Choose a new name for the ARRAY of "states" - X

Start numbering at 0?
Start numbering at 1?
The answer is computer language dependent!

New Name	Old name	Equation	Derivative (old names)
X_1	x	$X_1 = x$	$\dot{X}_1 = \dot{x}$
X_2	$\dot{x}$	$X_2 = \dot{x}$	$\dot{X}_2 = \ddot{x} = \frac{f}{m} - \frac{c}{m}\dot{x} - \frac{k}{m}x$

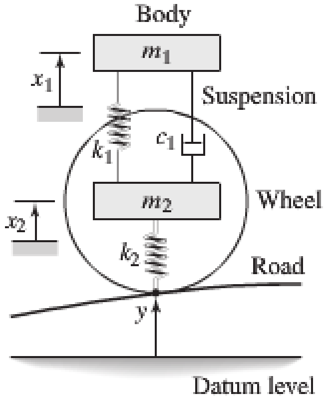
State-Variable Modeling

- Two Second Order ODE's

$$m_1\ddot{x}_1 = c_1(\dot{x}_2 - \dot{x}_1) + k_1(x_2 - x_1)$$

$$m_2\ddot{x}_2 = -c_1(\dot{x}_2 - \dot{x}_1) - k_1(x_2 - x_1) + k_2(y - x_2)$$

Number of states = 4 ….. Why?



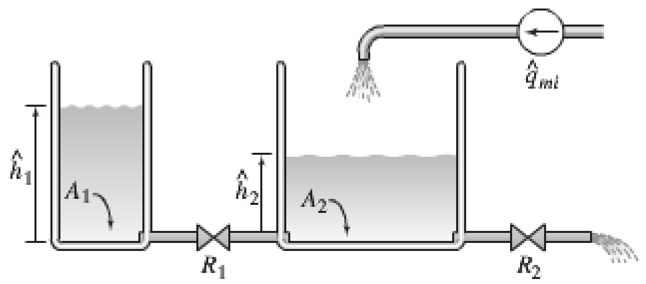
New Name	Old name	Equation	Derivative
X_1	x_1	$X_1 = x_1$	$\dot{X}_1 = \dot{x}_1$
X_2	$\dot{x}_1$	$X_2 = \dot{x}_1$	$\dot{X}_2 = \ddot{x}_1 = \dfrac{c_1(\dot{x}_2 - \dot{x}_1) + k_1(x_2 - x_1)}{m_1}$
X_3	x_2	$X_3 = x_2$	$\dot{X}_3 = \dot{x}_2$
X_4	$\dot{x}_2$	$X_4 = \dot{x}_2$	$\dot{X}_4 = \ddot{x}_2 = \dfrac{-c_1(\dot{x}_2 - \dot{x}_1) - k_1(x_2 - x_1) + k_2(y - x_2)}{m_2}$

State-Variable Modeling

- Now for the pair of First Order ODE's

$$A_1 \frac{dh_1}{dt} = -\frac{g}{R_1}(h_1 - h_2)$$

$$\rho A_2 \frac{dh_2}{dt} = q_{mi} + \frac{\rho g}{R_1}(h_1 - h_2) - \frac{\rho g}{R_2} h_2$$



Number of states needed = 2 Why?

New Name	Old name	Equation	Derivative
X_1	h_1	$X_1 = h_1$	$\dot{X}_1 = \dot{h}_1 = \dfrac{-\frac{g}{R_1}(h_1 - h_2)}{A_1}$
X_2	h_2	$X_2 = h_2$	$\dot{X}_2 = \dot{h}_2 = \dfrac{1}{\rho A_2}(q_{mi} + \frac{\rho g}{R_1}(h_1 - h_2) - \frac{\rho g}{R_2} h_2)$

Python's odeint Solver

The most difficult part of solving ODE's in Python is writing the System Definition

The System Definition

Array of Time values

Initial conditions

Optional parameters

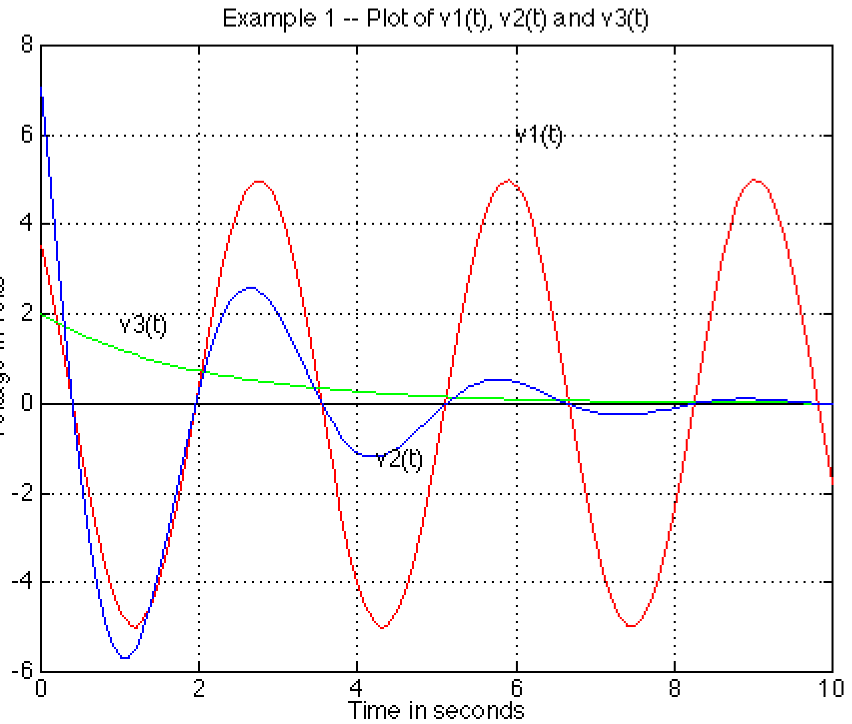
Python's odeint Function – ***a contract***

The same array of Time values

Matrix of Solution values

Plotting details – color, linestyle, scaling, etc.

Python's Plotting Function – ***another contract***



Python's odeint Solver – derivative function

The System Definition

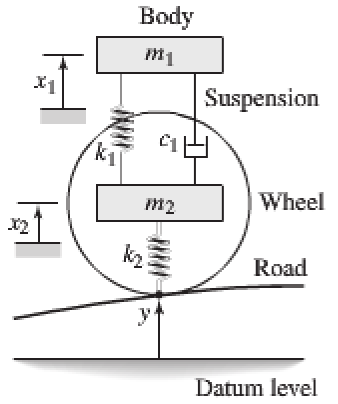
The current time in the simulation – a number

An array containing the ***values*** of the states variables of the system, at the current simulation time

A Function that you must write and give to odeint – to fulfill the contract

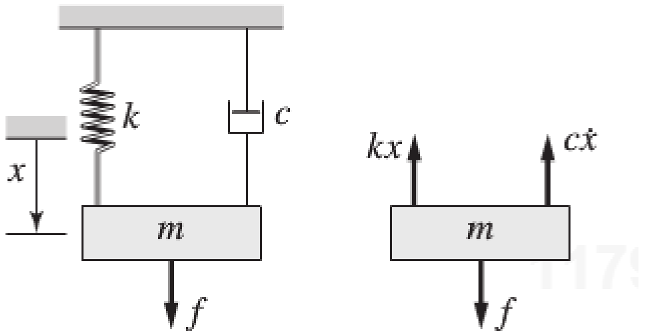
I often call this system definition function "the Derivative Function" … why?

An array containing the ***Time Derivatives*** of the state values of the system, at the current simulation time



New Name	Old name	Equation	Derivative
X_1	x_1	$X_1 = x_1$	$\dot{X}_1 = \dot{x}_1$
X_2	$\dot{x}_1$	$X_2 = \dot{x}_1$	$\dot{X}_2 = \ddot{x}_1 = \dfrac{c_1(\dot{x}_2 - \dot{x}_1) + k_1(x_2 - x_1)}{m_1}$
X_3	x_2	$X_3 = x_2$	$\dot{X}_3 = \dot{x}_2$
X_4	$\dot{x}_2$	$X_4 = \dot{x}_2$	$\dot{X}_4 = \ddot{x}_2 = \dfrac{-c_1(\dot{x}_2 - \dot{x}_1) - k_1(x_2 - x_1) + k_2(y - x_2)}{m_2}$

Writing the Derivative Function

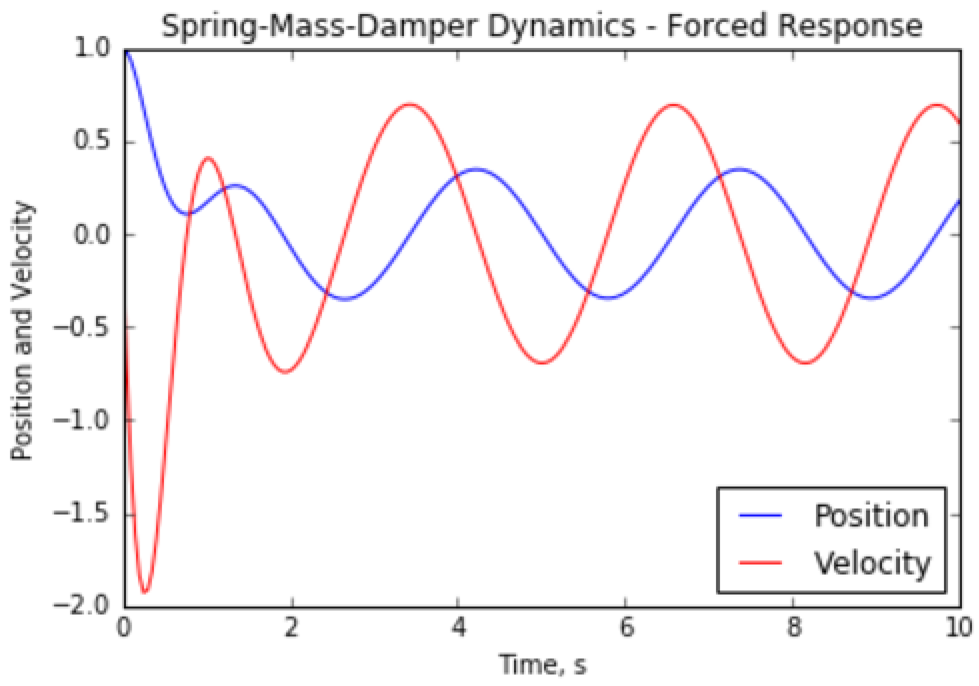


$$m\ddot{x} + c\dot{x} + kx = f$$

New Name	Old name	Equation	Derivative
X_1	x	$X_1 = x$	$\dot{X}_1 = \dot{x}$
X_2	$\dot{x}$	$X_2 = \dot{x}$	$\dot{X}_2 = \ddot{x} = \frac{f}{m} - \frac{c}{m}\dot{x} - \frac{k}{m}x$

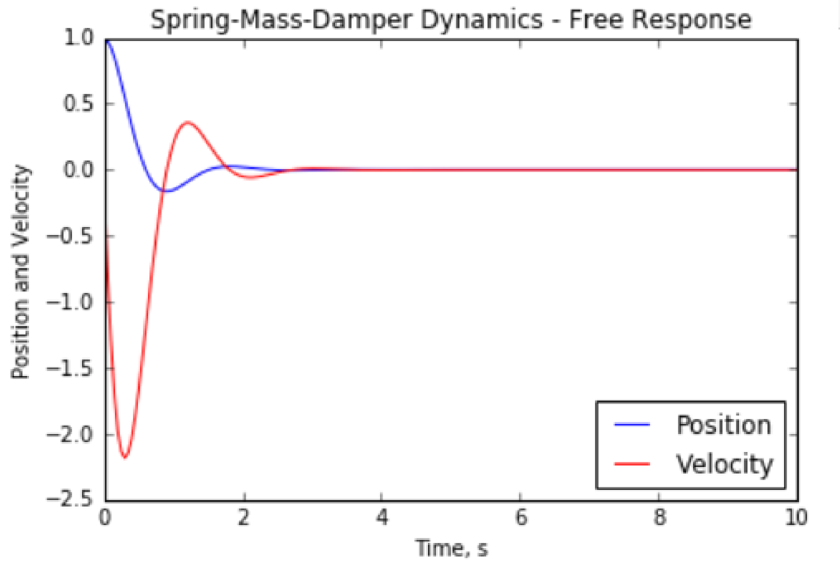
```
def ode_system(X, t, m,c,k,fmag ):
    #define any numerical parameters (constants)
    # these params were stored in a list, and must be passed in the correct order!

    #define the forcing function equation
    f=fmag*np.sin(2*t)

    x=X[0]; xdot=X[1]  # copy from the state array to nicer names

    #write the non-trivial equatin
    xddot= (1/m) * (f-c*xdot-k*x)

    return [xdot,xddot]
```

Shift to zero based arrays on this next line!



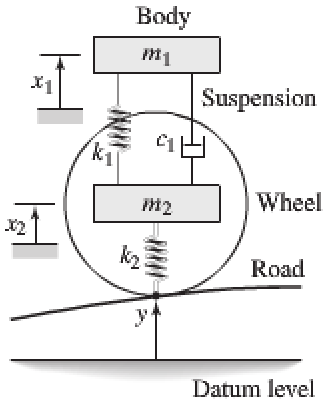

```
t = np.linspace(0, 10, 200)    #time goes from 0 to 10 seconds
ic=[1,0]

#define the model parameters
m=1  # the mass
c=4  # damping (shock absorber)
k=16 # the spring
fmag = 5 # the magnitude of the forcing function

x = odeint(ode_system, ic, t,args=(m,c,k,fmag))

plt.plot(t, x[:,0], 'b-', label = 'Position')
plt.plot(t, x[:,1], 'r-', label = 'Velocity')
plt.legend(loc = 'lower right')
plt.xlabel('Time, s')
plt.ylabel('Position and Velocity')
plt.title('Spring-Mass-Damper Dynamics - Forced')
plt.show()
```

```
def ode_system(X, t, m,c,k,fmag ):
    #define any numerical parameters (constants)
    # these params were stored in a list, and must be passed in the correct

    #define the forcing function equation
    f=fmag*np.sin(2*t)

    x=X[0]; xdot=X[1]  # copy from the state array to nicer names

    #write the non-trivial equatin
    xddot= (1/m) * (f-c*xdot-k*x)

    return [xdot,xddot]
```



Derivative Functions

```
def ode_system(X, t, carparams, roadparams ):
    #define any numerical parameters (constants)
    # these params were stored in two lists, and must be passed in the correct order!
    m1=carparams[0]; m2=carparams[1]; c1=carparams[2]; k1=carparams[3]; k2=carparams[4]

    ymag=roadparams[0]

    #define the forcing function equation
    if t < np.pi/2:
        y=ymag*np.sin(2*t)
    else:
        y=0

    x1=X[0]; x1dot=X[1]; x2=X[2]; x2dot=X[3] # copy from the state array to nicer names

    #write the non-trivial equations
    x1ddot= (1/m1) * (c1*(x2dot-x1dot)+k1*(x2-x1))
    x2ddot= (1/m2) * (-c1*(x2dot-x1dot)-k1*(x2-x1)+k2*(y-x2))

    #return the derivitaves of the inpu
    return [x1dot,x1ddot,x2dot,x2ddot]
```

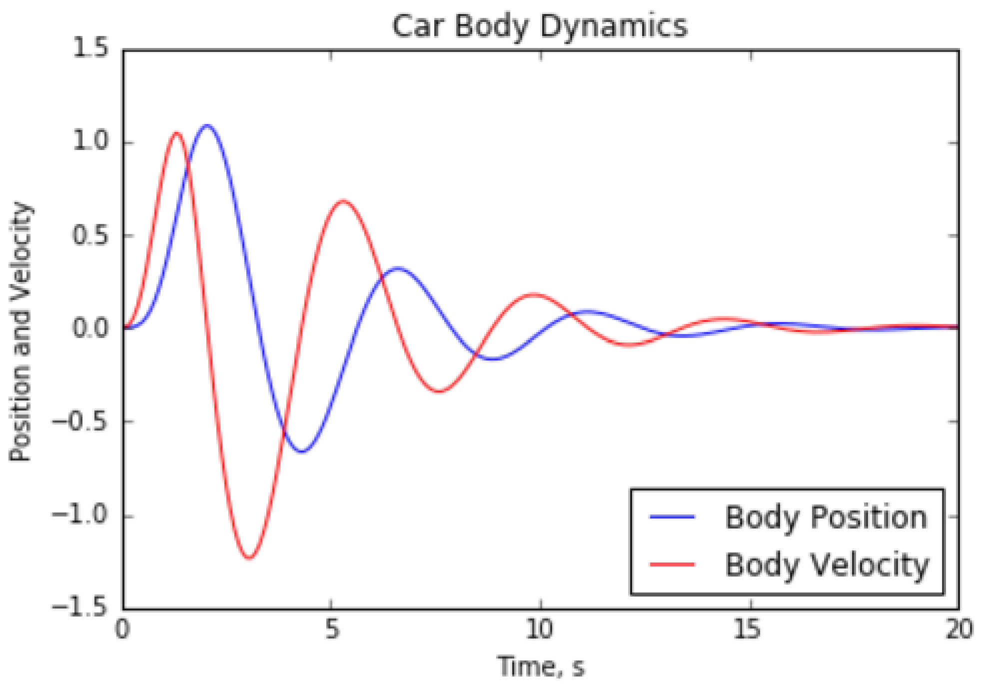


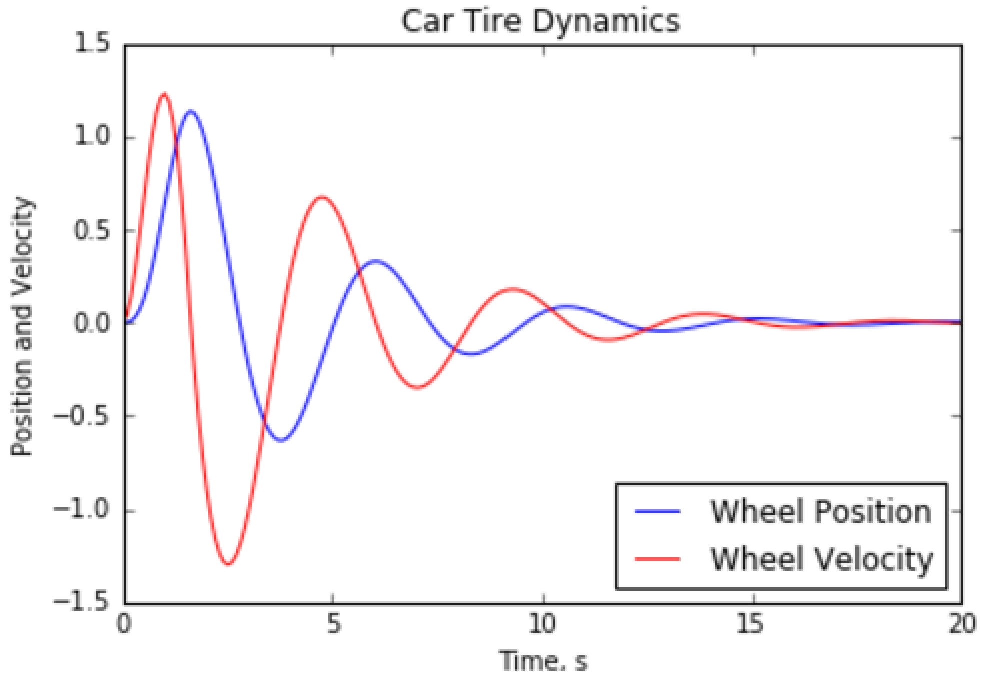
$$m_1\ddot{x}_1 = c_1(\dot{x}_2 - \dot{x}_1) + k_1(x_2 - x_1)$$

$$m_2\ddot{x}_2 = -c_1(\dot{x}_2 - \dot{x}_1) - k_1(x_2 - x_1) + k_2(y - x_2)$$

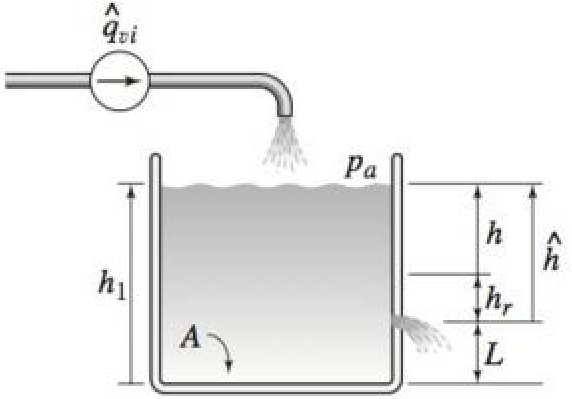
New Name	Old name	Equation	Derivative
X_1	x_1	$X_1 = x_1$	$\dot{X}_1 = \dot{x}_1$
X_2	$\dot{x}_1$	$X_2 = \dot{x}_1$	$\dot{X}_2 = \ddot{x}_1 = \dfrac{c_1(\dot{x}_2 - \dot{x}_1) + k_1(x_2 - x_1)}{m_1}$
X_3	x_2	$X_3 = x_2$	$\dot{X}_3 = \dot{x}_2$
X_4	$\dot{x}_2$	$X_4 = \dot{x}_2$	$\dot{X}_4 = \ddot{x}_2 = \dfrac{-c_1(\dot{x}_2 - \dot{x}_1) - k_1(x_2 - x_1) + k_2(y - x_2)}{m_2}$

```python
t = np.linspace(0, 20, 200)    #time goes from 0 to 10 seconds
ic=[0,0,0,0]

#define the model parameters
m1=1  # the mass
m2=1
c1=2  # damping (shock absorber)
k1=1 # the spring
k2=4
ymag = 1 # the magnitude of the forcing function
carparams=[m1,m2,c1,k1,k2]  #put the car parameters into a list
roadparams=[ymag] #put the road parameters into a list

x = odeint(ode_system, ic, t,args=(carparams,roadparams))

plt.plot(t, x[:,0], 'b-', label = 'Body Position')
plt.plot(t, x[:,1], 'r-', label = 'Body Velocity')
plt.legend(loc = 'lower right')
plt.xlabel('Time, s')
plt.ylabel('Position and Velocity')
plt.title('Car Body Dynamics')
plt.show()

plt.plot(t, x[:,2], 'b-', label = 'Wheel Position')
plt.plot(t, x[:,3], 'r-', label = 'Wheel Velocity')
plt.legend(loc = 'lower right')
plt.xlabel('Time, s')
plt.ylabel('Position and Velocity')
plt.title('Car Tire Dynamics')
plt.show()
```





First Order Nonlinear

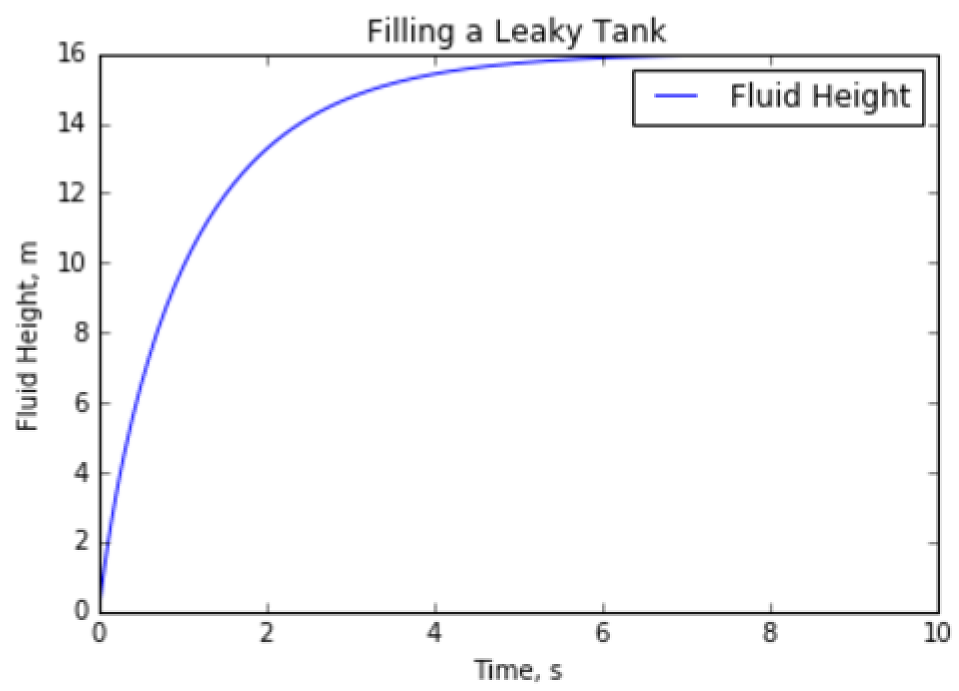


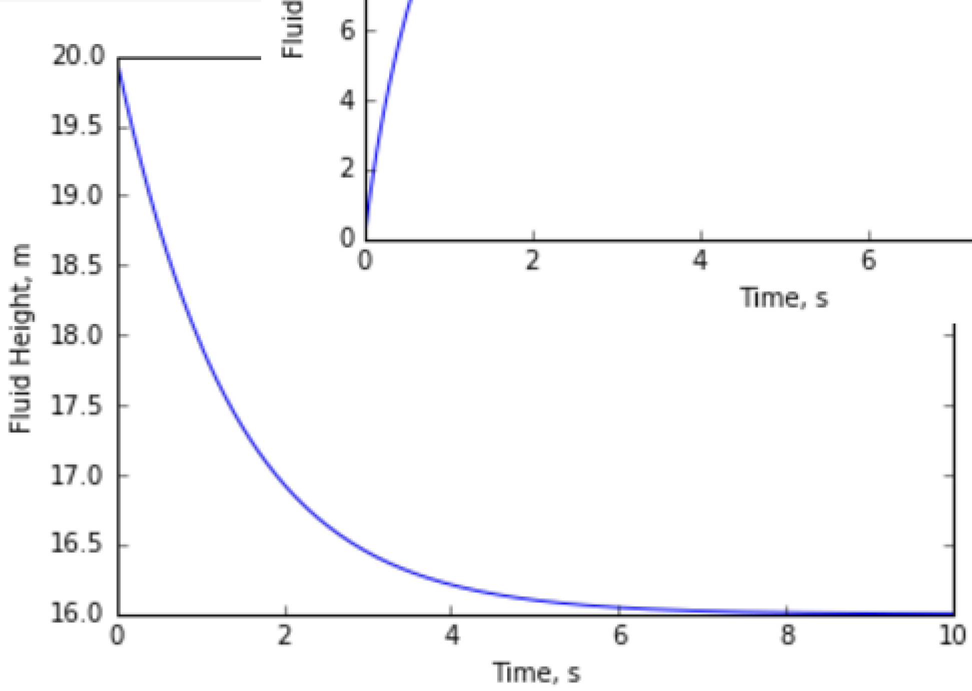
$$A\frac{d\hat{h}}{dt} = \hat{q}_{vi} - C_d A_o \sqrt{2g\hat{h}}$$

```
def ode_system(x, t, ):
        #define any numerical parameters (constants)
    qvi=24  # input flow rate
    cdA2g=6 # cd * Area* sqrt(2g)
    A=1     # Tank cross-sectional area

    h=x[0]
    hdot=(1/A)*(qvi-cdA2g*np.sqrt(h))

    return [hdot]
```

```python
t = np.linspace(0, 10, 200)     #time goes from 0 to 10 seconds
h0 = np.zeros(1)                # and array to hold the initial conditions
h0[0]=0

h = odeint(ode_system, h0, t)

plt.plot(t, h, 'b-', label = 'Fluid Height')
plt.legend(loc = 'upper right')
plt.xlabel('Time, s')
plt.ylabel('Fluid Height, m')
plt.title('Filling a Leaky Tank')
plt.show()
```

```python
def ode_system(x, t, ):
       #define any numerical parameters (cc
    qvi=24  # input flow rate
    cdA2g=6 # cd * Area* sqrt(2g)
    A=1     # Tank cross-sectional area

    h=x[0]
    hdot=(1/A)*(qvi-cdA2g*np.sqrt(h))

    return [hdot]
```





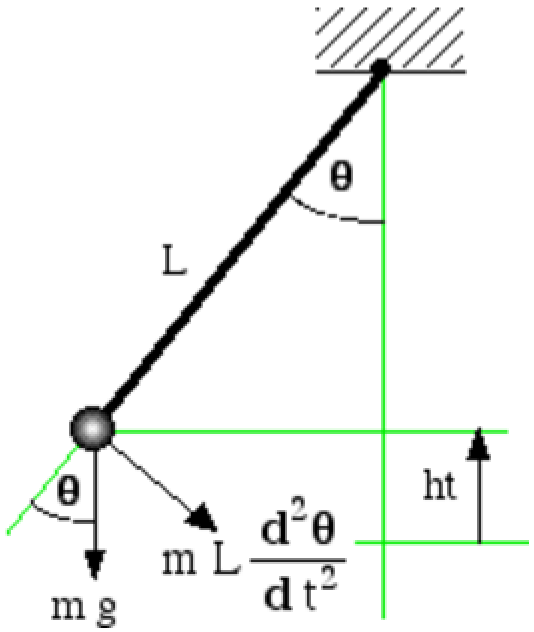
Simple Pendulum Model – Nonlinear

Try this one yourself!

From a force balance we obtain:

$$m\,g\,\sin(\theta) + m\,L\,\frac{d^2\theta}{dt^2} = 0$$

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\sin(\theta)$$



New Name	Old name	Equation	Derivative
X_1			
X_2			

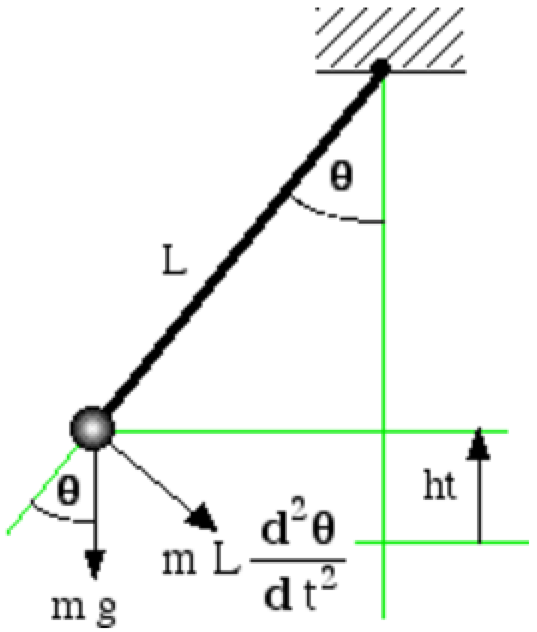
Simple Pendulum Model – Nonlinear

Try this one yourself!

From a force balance we obtain:

$$m\,g\,\sin(\theta) + m\,L\,\frac{d^2\theta}{dt^2} = 0$$

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\sin(\theta)$$



New Name	Old name	Equation	Derivative
X_1	θ	$X_1 = \theta$	$\dot{X}_1 = \dot{\theta}$
X_2	$\dot{\theta}$	$X_2 = \dot{\theta}$	$\dot{X}_2 = \ddot{\theta} = -\frac{g}{L}\sin(\theta)$

Simple Pendulum Model – Nonlinear

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as np

def ode_system(X, t, gc, L ):

    theta=X[0]; thetadot=X[1]   # copy from the state array to nicer names

    #write the non-trivial equation
    thetaddot= -gc/L * np.sin(theta)

    return [thetadot,thetaddot]

def main():
    t = np.linspace(0, 10, 200)    #time goes from 0 to 10 seconds
    ic=[1,0]

    #define the model parameters
    L = 2   # the mass
    gc = 9.81
    x = odeint(ode_system, ic, t,args=(gc,L))

    plt.plot(t, x[:,0], 'b-', label = 'Angular Position')
    plt.plot(t, x[:,1], 'r-', label = 'Angular Velocity')
    plt.legend(loc = 'lower right')
    plt.xlabel('Time, s')
    plt.ylabel('Angular Position and Velocity')
    plt.title('Simple Pendulum')
    plt.show()

main()
```

From a force balance we obtain:

$$m\,g\,\sin(\theta) + m\,L\,\frac{d^2\theta}{dt^2} = 0$$

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\sin(\theta)$$