```python
import sys
import numpy as np
import numpy.linalg as la
from scipy.linalg import expm    # function for computing the matrix exponential
from math import sqrt

class Empty:
    def __init__(self):
        pass

def get_motor_parameters(name):
    param = Empty()
    if(name=='Focchi2013'):
        param.I_m = 5.72e-5    # motor inertia
        param.b_m = 1.5e-3     # motor damping (this seemed way too large so I decreased it)
        param.L = 2.02e-3      # coil inductance
        param.R = 3.32         # coil resistance
        param.K_b = 0.19       # motor torque/speed constant
    elif(name=="Maxon148877"): # Maxon 148877 (150W,48V)
        param.V_nominal = 48.0 # nominal voltage
        param.R = 1.16        # terminal resistance [ohm]
        param.L = 0.33e-3        # terminal inductance [H]
        param.K_b = 60.3e-3      # torque constant [Nm/A]
        #K_b = 158        # speed constant [rpm/V]
        param.I_m = 134e-7        # rotor inertia [kg*m^2]
        param.i_0 = 69e-3      # no-load current [A] (Coulomb friction)
        param.tau_coulomb = param.K_b*param.i_0
        param.b_m = 1e-4       # motor damping (not found in data sheet)

    return param


class Motor:
    ''' A DC motor with the following dynamics
            V = R*i + L*di + K_b*dq
            tau = K_b*i
            tau = I_m*ddq + b_m*dq
        where:
            V = voltage
            i = current
            di = current rate of change
            dq = velocity
            ddq = acceleration
            tau = torque
            R = resistance
            L = inductance
            I_m = motor inertia
            b_m = motor viscous friction coefficient

```

```
        Defining the system state as angle, velocity, current:
            x = (q, dq, i)
        the linear system dynamics is then:
            dq  = dq
            ddq = I_m^-1 * (K_b*i - b_m*dq)
            di  = L^-1 * (V - R*i - K_b*dq)
    '''

    def __init__(self, dt, params):
        # store motor parameters in member variables
        self.dt  = dt
        self.R   = params.R
        self.L   = params.L
        self.K_b = params.K_b
        self.I_m = params.I_m
        self.b_m = params.b_m

        # set initial motor state to zero
        self.x = np.zeros(3)
        self.compute_system_matrices()

    def compute_system_matrices(self):
        # compute system matrices (A, B) in continuous time: dx = A*x + B*u
        self.A = np.array([[     0.0,              0.0,  0.0],
                           [     0.0,              0.0,  0.0],
                           [     0.0,              0.0,  0.0]])
        self.B = np.array([0.0, 0.0, 0.0]).T

        # convert to discrete time
        H = np.zeros((4,4))

        #self.Ad = ...
        #self.Bd = ...

    def set_state(self, x):
        self.x = np.copy(x)

    def simulate_voltage(self, V, method='exponential'):
        ''' Simulate assuming voltage as control input '''
        self.voltage = V
        if(method=='exponential'):
            pass
        else:
            pass
        return self.x

    def simulate(self, i):
```

self.R

$$H = \begin{array}{cc} 3\times 3 & 3\times 1 \\ \end{array} \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix}$$

0,1,2

$$e^{tH} = \begin{array}{cc} 3\times 3 & 3\times 1 \end{array} \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}$$

eH[0:3,0:3]

eH[0:3, 3]

H[0:3,0:3] = A

H[0:3, 3] = B

eH = expm(H * self.dt)

self.x = self.Ad.dot(self.x) + self.Bd.dot (V)

$x = A_d x + B_d V$

```
        ''' Simulate assuming a perfect current controller (no electrical dynamics) '''
        #self.voltage = ...

        return self.x

    def q(self):
        return self.x[0]

    def dq(self):
        return self.x[1]

    def i(self):
        return self.x[2]

    def tau(self):
        return self.K_b*self.x[2]

    def V(self):
        return self.voltage


if __name__=='__main__':
    import arc.utils.plot_utils as plut
    import matplotlib.pyplot as plt
    np.set_printoptions(precision=1, linewidth=200, suppress=True)

    dt = 1e-4       # simulation time step
    T = 1.0         # simulation time
    V_b = 25.0      # initial motor voltage
    V_a = 0.0       # linear increase in motor voltage per second
    V_w = 2.0       # frequency of sinusoidal motor voltage
    V_A = 0.0       # amplitude of isnusoidal motor voltage
    params = get_motor_parameters('Maxon148877')

    # simulate motor with linear+sinusoidal input voltage
    N = int(T/dt)   # number of time steps
    motor = Motor(dt, params)
    q = np.zeros(N+1)
    dq = np.zeros(N+1)
    current = np.zeros(N+1)
    V = np.zeros(N)
    for i in range(N):
        t = i*dt
        V[i] = V_a*t + V_b + V_A*np.sin(2*np.pi*V_w*t)
        motor.simulate_voltage(V[i])
        q[i+1] = motor.q()
        dq[i+1] = motor.dq()
        current[i+1] = motor.i()

    # plot motor angle, velocity and current
```

```python
    f, ax = plt.subplots(4,1,sharex=True)
    time = np.arange(0.0, T+dt, dt)
    time = time[:N+1]
    ax[0].plot(time, q, label ='angle')
    ax[1].plot(time, dq, label ='velocity')
    ax[2].plot(time, current, label ='current')
    ax[3].plot(time[:-1], V, label ='voltage')
    for i in range(4): ax[i].legend()
    plt.xlabel('Time [s]')
    plt.show()

    print("Final velocity", dq[-1])

# Apply a constant voltage and answer the following questions.
# How is the relationship between voltage and current? 1) Linear, 2) Approximately linear, 3) Quadratic
# How is the relationship between voltage and speed? 1) Linear, 2) Approximately linear, 3) Quadratic
# Apply a sinusoidal voltage and answer the following questions.
# What is the ratio between voltage and speed at low frequency (e.g., 1 Hz)?
# What happens to this ratio as the frequency increases (e.g., 10 Hz, 100 Hz)?

```