```python
import numpy as np
import param as P

class ballBeamDynamics:

    def __init__(self):

        # Initial state conditions
        self.state = np.matrix([[P.z0],          # z initial position
                                [P.theta0],      # Theta initial orientation
                                [P.zdot0],       # zdot initial velocity
                                [P.thetadot0]]) # Thetadot initial velocity

    def propagateDynamics(self,u):
        # P.Ts is the time step between function calls.
        # u contains the force and/or torque input(s).

        # RK4 integration
        k1 = self.Derivatives(self.state, u)
        k2 = self.Derivatives(self.state + P.Ts/2*k1, u)
        k3 = self.Derivatives(self.state + P.Ts/2*k2, u)
        k4 = self.Derivatives(self.state + P.Ts*k3, u)
        self.state += P.Ts/6 * (k1 + 2*k2 + 2*k3 + k4)


    # Return the derivatives of the continuous states
    def Derivatives(self,state,u):

        # States and forces
        z = state.item(0)
        theta = state.item(1)
        zdot = state.item(2)
        thetadot = state.item(3)
        F = u[0]

        thetaddot = (1.0/((P.m2*P.ell**2)/3.0 + P.m1*z**2))* \
                    (-P.m2*P.g*P.ell/2.0*np.cos(theta)+P.ell*F*np.cos(theta) -
                        2.0*P.m1*z*zdot*thetadot)

        zddot = (1.0/P.m1)*(P.m1*z*thetadot**2-P.m1*P.g*np.sin(theta))

        xdot = np.matrix([[zdot],[thetadot],[zddot],[thetaddot]])

        return xdot


    # Returns the observable states
    def Outputs(self):
        # Return them in a list and not a matrix
        return self.state[0:2].T.tolist()[0]

    # Returns all current states
    def States(self):
        # Return them in a list and not a matrix
        return self.state.T.tolist()[0]
```