Double Pendulum

Chaotic characteristics of a double pendulum and the impact on numerical solutions.

The equations of a double pendulum were calculated using two program with two different differentiation function to feed to the scipy ode integrator with of type 'vode', otherwise the two programs were the same.

<u>Program 1, annotated with s1 had the following derivative</u>

```
def get_derivatives_double_pendulum(t, state):
    ''' definition of ordinary differential equation for a
        double pendulum
    '''
    theta1, theta1_dot, theta2, theta2_dot = state

    _num1 = -g * (2 * mass_bob1 + mass_bob2) * np.sin(theta1)
    _num2 = -mass_bob2 * g * np.sin(theta1-2*theta2)
    _num3 = -2*np.sin(theta1-theta2) * mass_bob2
    _num4 = theta2_dot * theta2_dot * length_r2 + \
            theta1_dot * theta1_dot * length_r1 * np.cos(theta1-theta2)
    _den = length_r1 * (2 * mass_bob1 + mass_bob2 - \
           mass_bob2 * np.cos(2*theta1-2*theta2))
    theta1_doubledot = (_num1 + _num2 + _num3 * _num4) / _den

    _num1 = 2 * np.sin(theta1-theta2)
    _num2 = (theta1_dot * theta1_dot * length_r1 * (mass_bob1 + mass_bob2))
    _num3 = g * (mass_bob1 + mass_bob2) * np.cos(theta1)
    _num4 = (theta2_dot * theta2_dot * length_r2 * \
            mass_bob2 * np.cos(theta1-theta2))
    _den = length_r2 * (2 * mass_bob1 + mass_bob2 - \
           mass_bob2 * np.cos(2*theta1-2*theta2))
    theta2_doubledot = (_num1 * (_num2 + _num3 + _num4)) / _den

    state_differentiated = np.zeros(4)
    state_differentiated[0] = theta1_dot
    state_differentiated[1] = theta1_doubledot
    state_differentiated[2] = theta2_dot
    state_differentiated[3] = theta2_doubledot

    return state_differentiated
```

<u>Program 2, annotated with s2 had the following derivative</u>

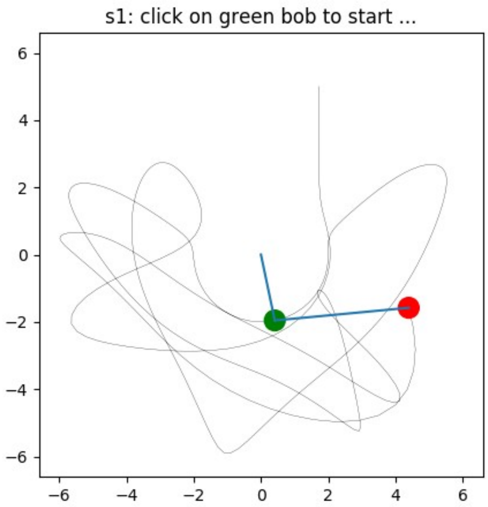
```
def get_derivatives_double_pendulum(t, state):
    ''' definition of ordinary differential equation for a
        double pendulum
    '''
    t1, w1, t2, w2 = state
    dt = t1 - t2
    _sin_dt = sin(dt)
    _den1 = (m1+m2*_sin_dt*_sin_dt)

    _num1 = m2*l1*w1*w1*sin(2*dt)
    _num2 = 2*m2*l2*w2*w2*_sin_dt
    _num3 = 2*g*m2*cos(t2)*_sin_dt+2*g*m1*sin(t1)
    _num4 = 0
    w1_dot = (_num1+_num2+_num3+_num4)/ (-2*l1*_den1)

    _num1 = m2*l2*w2*w2*sin(2*dt)
    _num2 = 2*(m1+m2)*l1*w1*w1*_sin_dt
    _num3 = 2*g*(m1+m2)*cos(t1)*_sin_dt
    _num4 = 0
```

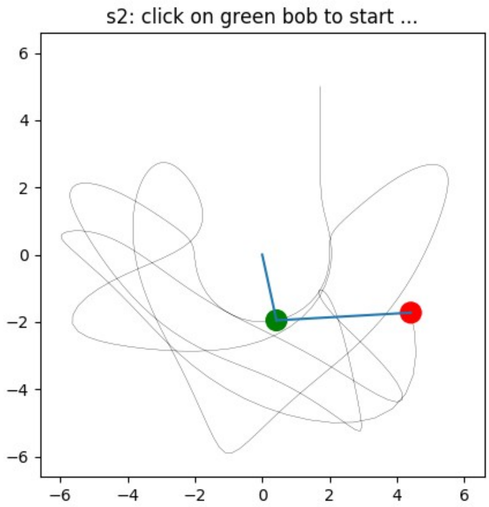
```
    w2_dot = (_num1+_num2+_num3+_num4)/ (2*l2*_den1)

    state_differentiated = np.zeros(4)
    state_differentiated[0] = w1
    state_differentiated[1] = w1_dot
    state_differentiated[2] = w2
    state_differentiated[3] = w2_dot

    return state_differentiated
```

The physical properties and initial state at t=0 were:

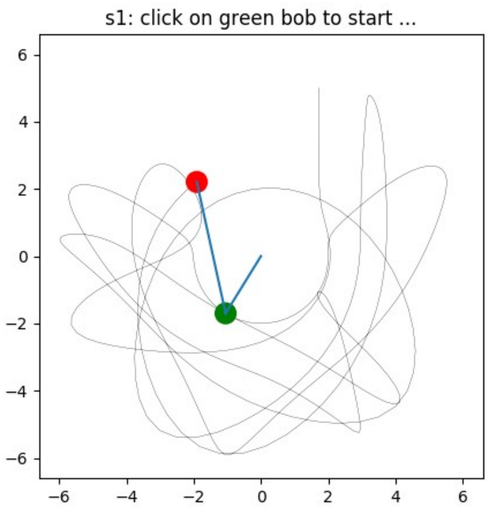
```
# physical properties
g = 9.8
length_r1 = 2.0  # meter
length_r2 = 4.0  # meter
mass_bob1 = 10.0 # kg
mass_bob2 = 5.0  # kg

# initial state
theta1_initial = + 120 / 180 * np.pi
theta2_initial = + 180 / 180 * np.pi
theta1_dot_initial = 0  # no initial angular velocity
theta2_dot_initial = 0  # no initial angular velocity
```

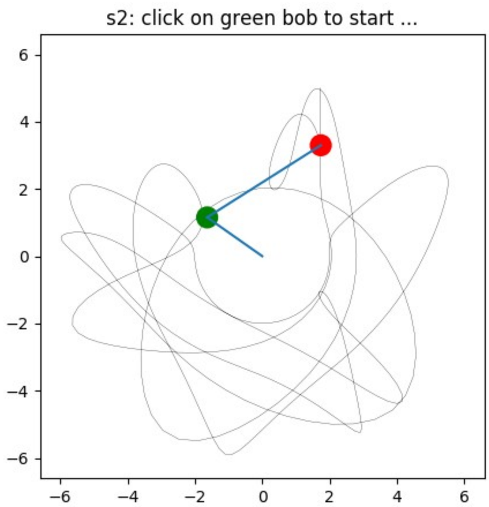
Purely using different equations for the same solution already change the outcome of the numerical results. Up to 15 seconds the two programs show the same result but thereafter they start to deviate as the the plots show for 20 seconds and 25 seconds.



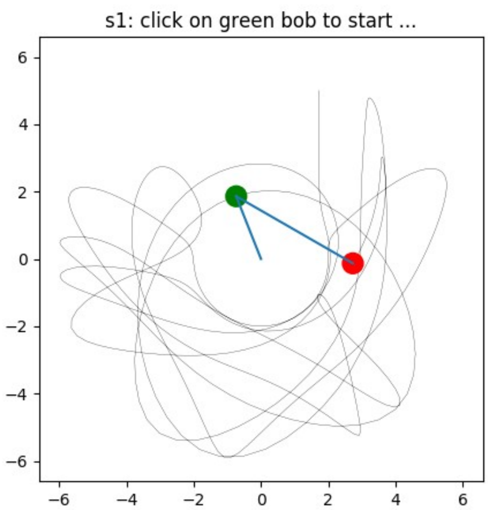


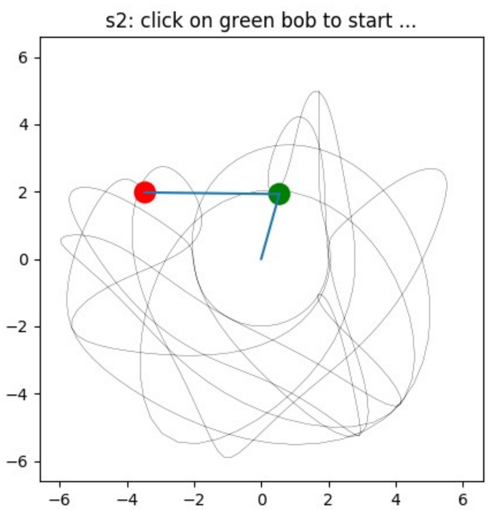
15 seconds





20 seconds





25 seconds

So although each program may give a fairly accurate representation of what the the pendulum is doing at any instance, the actual positions and angular velocities cannot be compared with each other when time increases.