ME314_HW1_Solution_Spring22

April 4, 2022

1 ME314 Homework 1 (Solution)

1.0.1 Submission instructions

Deliverables that should be included with your submission are shown in **bold** at the end of each problem statement and the corresponding supplemental material. Your homework will be graded IFF you submit a **single** PDF and a link to a Google colab file that meet all the requirements outlined below.

- List the names of students you've collaborated with on this homework assignment.
- Include all of your code (and handwritten solutions when applicable) used to complete the problems.
- Highlight your answers (i.e. **bold** and outline the answers) and include simplified code outputs (e.g. .simplify()).
- Enable Google Colab permission for viewing
- Click Share in the upper right corner
- Under "Get Link" click "Share with..." or "Change"
- Then make sure it says "Anyone with Link" and "Editor" under the dropdown menu
- Make sure all cells are run before submitting (i.e. check the permission by running your code in a private mode)
- Please don't make changes to your file after submitting, so we can grade it!
- Submit a link to your Google Colab file that has been run (before the submission deadline) and don't edit it afterwards!

NOTE: This Juputer Notebook file serves as a template for you to start homework. Make sure you first copy this template to your own Google driver (click "File" -> "Save a copy in Drive"), and then start to edit it.

HOMEWORK SOLUTION
DO NOT DISTRIBUTE

[2]:
```
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt

print(sym.__version__)
```

```
1.7.1
```

[3]:
```
########################################################################################
# If you're using Google Colab, uncomment this section by selecting the whole␣
↪section and press
```

```
# ctrl+'/' on your and keyboard. Run it before you start programming, this will␣
↪enable the nice
# LaTeX "display()" function for you. If you're using the local Jupyter␣
↪environment, leave it alone
################################################################################
# def custom_latex_printer(exp,**options):
#     from google.colab.output._publish import javascript
#     url = "https://cdnjs.cloudflare.com/ajax/libs/mathjax/3.1.1/latest.js?
↪config=TeX-AMS_HTML"
#     javascript(url=url)
#     return sym.printing.latex(exp,**options)
# sym.init_printing(use_latex="mathjax",latex_printer=custom_latex_printer)
```

1.1 Problem 1 (15pts)

[4]:
```
from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='https://github.com/MuchenSun/ME314pngs/
↪raw/master/2mass_spring.png' width=500' height='350'></table>"))
```

```
<IPython.core.display.HTML object>
```

As shown in the image above, a block of mass m_1 is on one side connected to a wall by a massless spring with a spring constant k_1 and on another side to a block of mass m_2 by a massless spring with a spring constant k_2. Assuming that the two springs have lengths of zero when "relaxed", they are stretched with any positive displacement $\Delta x > 0$ and the magnitude of the force can be computed using Hooke's law $|F| = k\Delta x$, where k is the spring constant. Furthermore, there is no friction between the blocks and the ground.

Given masses $m_1 = 1kg$ and \$ m_2=2kg\$, spring constants $k_1 = 0.5N/m$ and $k_2 = 0.8N/m$, and positions of the blocks as x_1 and x_2, use Newton's law $F = ma$ to compute the accelerations of the blocks $a_1 = \ddot{x}_1$ and $a_2 = \ddot{x}_2$. You need to use Pythons's SymPy package to solve for symbolic solutions, as well as numerically evaluate your solutions for a_1 and a_2 as functions of x_1 and x_2 respectively. Test your numerical functions with $x_1 = 1m$ and $x_2 = 3m$ as function inputs.

Hint 1: You will have two equations based on Newton's law $F = ma$ for each block. Thus, for each block you need to write down its F in terms of x_1 and x_2 (which can be defined as symbols in SymPy).

Hint 2: You will need to use SymPy's **`solve()`** *and* **`lambdify()`** *methods in this problem as seen in Homework 0. This problem is very similar to Problem 5 in Homework 0, except that (1) you need to write down the equations yourself, and (2) you don't need to solve the equations simultaneously - you can solve them one by one for each block. Feel free to take the example code in Homework 0 as a starting point.*

Hint 3: You will need to use **`lambdify()`** *to numerically evaluate a function with multiple variables.*

Turn in: A copy of the code used to symbolically compute the Lagrangian and corresponding output of the code (i.e. the computed Lagrangian.)

HOMEWORK SOLUTION
DO NOT DISTRIBUTE

[5]:
```python
# define constants
m1 = 1
m2 = 2
k1 = 0.5
k2 = 0.8

# define positions
x1, x2 = sym.symbols(r'x_1, x_2')

# define accelerations
a1, a2 = sym.symbols(r'a_1, a_2')

# use Newton's law F=ma to define equations
eqn1 = sym.Eq( -k1*x1 + k2*(x2-x1) , m1*a1)
eqn2 = sym.Eq( -k2*(x2-x1) , m2*a2)

# solve equations
a1_soln = sym.solve(eqn1, [a1])[0]
print('Symbolic solution for a1:')
display(a1_soln)
a2_soln = sym.solve(eqn2, [a2])[0]
print('\nSymbolic solution for a2:')
display(a2_soln)

# then numerically evaluate solutions
from sympy import lambdify

a1_func = lambdify([x1, x2], a1_soln)
a2_func = lambdify([x1, x2], a2_soln)
print("\nEvaluate solutions at x1=1, x2=3: ")
print("a1 = {:.2f}".format(a1_func(1, 3)))
print("a2 = {:.2f}".format(a2_func(1, 3)))
```

```
Symbolic solution for a1:
```

$-1.3x_1 + 0.8x_2$

```
Symbolic solution for a2:
```

$0.4x_1 - 0.4x_2$

```
Evaluate solutions at x1=1, x2=3:
a1 = 1.10
a2 = -0.80
```

HOMEWORK SOLUTION
DO NOT DISTRIBUTE

1.2 Problem 2 (10pts)

For the same system in Problem 1, compute the Lagrangian of the system using Python's SymPy package with x_1, x_2 as system configuration variables.

Hint 1: For an object with mass m and velocity v, its kinetic energy is $\frac{1}{2}mv^2$.

Hint 2: For a spring stretched with displacement Δx and spring ratio k, its potential energy is $\frac{1}{2}k(\Delta x)^2$. Also, the springs have zero mass.

*Hint 3: Since x_1 and x_2 are actually functions of time t, in order to compute Euler-Lagrange equations you will need to take their derivative with respect to t. Instead of defining position and velocity as two separate symbols, you need to define position as SymPy's **Function** object and the velocity as the derivative of that function with respect to time t.*

Turn in: A copy of the code used to symbolically compute the Lagrangian.

[6]:
```
t = sym.symbols('t')
# define constants
m1 = 1
m2 = 2
k1 = 0.5
k2 = 0.8

# define position as function
x1 = sym.Function('x_1')(t)
x2 = sym.Function('x_2')(t)

# define velocity and acceleration as derivative
x1dot = x1.diff(t)
x2dot = x2.diff(t)
x1ddot = x1dot.diff(t)
x2ddot = x2dot.diff(t)

# compute kinetic energy
ke = 0.5*m1*x1dot**2 + 0.5*m2*x2dot**2
pe = 0.5*k1*x1**2 + 0.5*k2*(x1-x2)**2

# compute Lagrangian
L = ke - pe
print('Lagrangian: ')
display(L.simplify())
```

```
Lagrangian:
```

$$-0.4\left(x_1(t) - x_2(t)\right)^2 - 0.25\,x_1^2(t) + 0.5\left(\frac{d}{dt}x_1(t)\right)^2 + 1.0\left(\frac{d}{dt}x_2(t)\right)^2$$

HOMEWORK SOLUTION
DO NOT DISTRIBUTE

1.3 Problem 3 (10pts)

Based on your solution for Problem 2, compute the Euler-Lagrange equations for this system.

Hint 1: In this problem, the system has two configuration variables. Thus, when taking the derivative of the Lagrangian with respect to the system state vector, the derivative is also a vector (sometimes also called the Jacobian of the Lagrangian with respect to system states).

Turn in: A copy of the code used to symbolically compute the Euler-Lagrange equations and the code outpute (i.e. the resulting equations).

[7]:
```
t = sym.symbols('t')
# define constants
m1 = 1
m2 = 2
k1 = 0.5
k2 = 0.8

# define position as function
x1 = sym.Function('x_1')(t)
x2 = sym.Function('x_2')(t)
q = sym.Matrix([x1, x2])
# define velocity and acceleration as derivative
x1dot = x1.diff(t)
x2dot = x2.diff(t)
qdot = sym.Matrix([x1dot, x2dot])
x1ddot = x1dot.diff(t)
x2ddot = x2dot.diff(t)
qddot = sym.Matrix([x1ddot, x2ddot])

# compute kinetic energy
ke = 0.5*m1*x1dot**2 + 0.5*m2*x2dot**2
pe = 0.5*k1*x1**2 + 0.5*k2*(x1-x2)**2

# compute Lagrangian
L = ke - pe
print('Lagrangian: ')
display(L)

# take derivatives (Jacobian)
L_mat = sym.Matrix([L])
dLdqdot = L_mat.jacobian(qdot)
ddLdqdot_dt = dLdqdot.diff(t)
dLdq = L_mat.jacobian(q)

# compute Euler-Lagrange equations
el_eqns = sym.Eq(ddLdqdot_dt.T-dLdq.T, sym.Matrix([0, 0]))
print('\nEuler-Lagrange equations: ')
```

HOMEWORK SOLUTION DO NOT DISTRIBUTE

```
display(el_eqns)
```

```
Lagrangian:
```

$$-0.4\left(x_1(t) - x_2(t)\right)^2 - 0.25\,x_1^2(t) + 0.5\left(\frac{d}{dt}x_1(t)\right)^2 + 1.0\left(\frac{d}{dt}x_2(t)\right)^2$$

```
Euler-Lagrange equations:
```

$$\begin{bmatrix} 1.3\,x_1(t) - 0.8\,x_2(t) + 1.0\frac{d^2}{dt^2}x_1(t) \\ -0.8\,x_1(t) + 0.8\,x_2(t) + 2.0\frac{d^2}{dt^2}x_2(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

1.4 Problem 4 (15pts)

Based on your Euler-Lagrange equations from Problem 3, use Python's SymPy package to solve the equations for the accelerations of the two blocks $\ddot{x}_1, \ddot{x}_2$, in terms of position x_1, x_2, and velocity $\dot{x}_1, \dot{x}_2$. Compare your answer to Problem 1 to see if they match with each other and comment if this is not the case.

Hint 1: Since you need to solve a set of multiple equations symbolically in SymPy, it's recommended to wrap both sides of the equations into SymPy's Matrix object.

Turn in: A copy of the code you used to numerically and symbolically evaluate the solution.

[8]:
```
t = sym.symbols('t')
# define constants
m1 = 1
m2 = 2
k1 = 0.5
k2 = 0.8

# define position as function
x1 = sym.Function('x_1')(t)
x2 = sym.Function('x_2')(t)
q = sym.Matrix([x1, x2])
# define velocity and acceleration as derivative
x1dot = x1.diff(t)
x2dot = x2.diff(t)
qdot = sym.Matrix([x1dot, x2dot])
x1ddot = x1dot.diff(t)
x2ddot = x2dot.diff(t)
qddot = sym.Matrix([x1ddot, x2ddot])

# compute kinetic energy
ke = 0.5*m1*x1dot**2 + 0.5*m2*x2dot**2
pe = 0.5*k1*x1**2 + 0.5*k2*(x1-x2)**2
```

HOMEWORK SOLUTION
DO NOT DISTRIBUTE

```
# compute Lagrangian
L = ke - pe
print('Lagrangian: ')
display(L)

# take derivatives (Jacobian)
L_mat = sym.Matrix([L])
dLdqdot = L_mat.jacobian(qdot)
ddLdqdot_dt = dLdqdot.diff(t)
dLdq = L_mat.jacobian(q)

# compute Euler-Lagrange equations
el_eqns = sym.Eq(ddLdqdot_dt.T-dLdq.T, sym.Matrix([0, 0]))
print('\nEuler-Lagrange equations: ')
display(el_eqns)

# solve the Euler-Lagrange equations for qddot
print('\nSolutions:')
soln = sym.solve(el_eqns, qddot)
for xddot in qddot:
    display(sym.Eq(xddot, soln[xddot]))
```

```
Lagrangian:
```

$$-0.4\left(x_1(t) - x_2(t)\right)^2 - 0.25\,x_1^2(t) + 0.5\left(\frac{d}{dt}x_1(t)\right)^2 + 1.0\left(\frac{d}{dt}x_2(t)\right)^2$$

```
Euler-Lagrange equations:
```

$$\begin{bmatrix} 1.3\,x_1(t) - 0.8\,x_2(t) + 1.0\frac{d^2}{dt^2}x_1(t) \\ -0.8\,x_1(t) + 0.8\,x_2(t) + 2.0\frac{d^2}{dt^2}x_2(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

```
Solutions:
```

$$\frac{d^2}{dt^2}x_1(t) = -1.3\,x_1(t) + 0.8\,x_2(t)$$

$$\frac{d^2}{dt^2}x_2(t) = 0.4\,x_1(t) - 0.4\,x_2(t)$$

1.5 Problem 5 (10pts)

You are given the unforced inverted cart-pendulum as shown below. The pendulum is in gravity and can swing freely (the cart will not interfere with pendulum's motion). The cart can move freely in the horizontal direction. Take x and θ as the system configuration variables and compute the Lagrangian of the system using Python's SymPy package. Provide symoblic solution of the Lagrangian with m and M as symbols instead of given constants.

HOMEWORK SOLUTION DO NOT DISTRIBUTE

Hint 1: Assume that the positive direction for the rotation of the pendulum $\dot{\theta}$ is clockwise. Likewise, the positive direction for the translation of the cart is to the right.

Hint 2: You will need to define m and M as SymPy's symbols.

Hint 3: Note that the pendulum is attached to the cart! In order to compute the kinetic energy of the pendulum, the velocity of the cart needs to be considered. One suggestion is to compute the velocity of the pendulum in a different coordinate system (other than Cartesian with x and y coordinates).

Turn in: A copy of the code used to symbolically compute Lagrangian and the code output (i.e. the computed expression of the Lagrangian).

[9]:
```
from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='https://raw.githubusercontent.com/
↪MuchenSun/ME314pngs/master/dyninvpend.png' width=500' height='350'></
↪table>"))
```

```
<IPython.core.display.HTML object>
```

[10]:
```
# define constants (as symbols)
t, m, M, R, g = sym.symbols(r't, m, M, R, g')

# define states and time derivatives
x = sym.Function(r'x')(t)
th = sym.Function(r'\theta')(t)
q = sym.Matrix([x, th])

xdot = x.diff(t)
thdot= th.diff(t)
qdot = q.diff(t)

xddot = xdot.diff(t)
thddot = thdot.diff(t)
qddot = qdot.diff(t)

# compute pendulum's velocity in x and y direction
# this is under the "world frame", from where we measure
# the position of the cart as x(t)
pen_xdot = xdot + R*thdot*sym.cos(th)
pen_ydot = R*thdot*sym.sin(th)

# compute Lagrangian
ke = 0.5*M*xdot**2 + 0.5*m*(pen_xdot**2 + pen_ydot**2)
pe = m*g*R*sym.cos(th)
L = ke - pe

print('Lagrangian: ')
display(L)
```

HOMEWORK SOLUTION
DO NOT DISTRIBUTE

Lagrangian:

$$0.5M\left(\frac{d}{dt}x(t)\right)^2 - Rgm\cos(\theta(t)) + 0.5m\left(R^2\sin^2(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 + \left(R\cos(\theta(t))\frac{d}{dt}\theta(t) + \frac{d}{dt}x(t)\right)^2\right)$$

1.6 Problem 6 (15pts)

Based on your solution in Problem 5, compute the Euler-Lagrange equations for this inverted cart-pendulun system using Python's SymPy package.

Turn in: A copy of the code used to symbolically compute Euler-Lagrange equations and the code output (i.e. the computed expression of the Lagrangian).

[11]:
```
# define constants (as symbols)
t, m, M, R, g = sym.symbols(r't, m, M, R, g')

# define states and time derivatives
x = sym.Function(r'x')(t)
th = sym.Function(r'\theta')(t)
q = sym.Matrix([x, th])

xdot = x.diff(t)
thdot= th.diff(t)
qdot = q.diff(t)

xddot = xdot.diff(t)
thddot = thdot.diff(t)
qddot = qdot.diff(t)

# compute pendulum's velocity in x and y direction
# this is under the "world frame", from where we measure
# the position of the cart as x(t)
pen_xdot = xdot + R*thdot*sym.cos(th)
pen_ydot = R*thdot*sym.sin(th)

# compute Lagrangian
ke = 0.5*M*xdot**2 + 0.5*m*(pen_xdot**2 + pen_ydot**2)
pe = m*g*R*sym.cos(th)
L = ke - pe

print('Lagrangian: ')
display(L)

# compute the derivatives we needed
L = sym.Matrix([L])
dLdq = L.jacobian(q).T
dLdqdot = L.jacobian(qdot).T
```

HOMEWORK SOLUTION
DO NOT DISTRIBUTE

```
ddLdqdot_dt = dLdqdot.diff(t)

# define Euler-Lagrange equations
el_eqns = sym.Eq(ddLdqdot_dt-dLdq, sym.Matrix([0, 0]))
print('\nEuler-Lagrange equations: ')
display(el_eqns.simplify())
```

```
Lagrangian:
```

$$0.5M\left(\frac{d}{dt}x(t)\right)^2-Rgm\cos\left(\theta(t)\right)+0.5m\left(R^2\sin^2\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2+\left(R\cos\left(\theta(t)\right)\frac{d}{dt}\theta(t)+\frac{d}{dt}x(t)\right)^2\right)$$

```
Euler-Lagrange equations:
```

$$\begin{bmatrix}0\\0\end{bmatrix}=\begin{bmatrix}1.0M\frac{d^2}{dt^2}x(t)+m\left(-R\sin\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2+R\cos\left(\theta(t)\right)\frac{d^2}{dt^2}\theta(t)+\frac{d^2}{dt^2}x(t)\right)\\ Rm\left(R\frac{d^2}{dt^2}\theta(t)-g\sin\left(\theta(t)\right)+\cos\left(\theta(t)\right)\frac{d^2}{dt^2}x(t)\right)\end{bmatrix}$$

1.7 Problem 7 (15pts)

Find symbolic expressions of $\ddot{x}$ and $\ddot{\theta}$ from the Euler-Lagrange equations in Problem 6 using SymPy's **solve()** method (NOTE: the expressions should be in terms of x, θ, $\dot{x}$ and $\dot{\theta}$ only). Convert these results to numerical functions using SymPy's **lambdify()** method by substituting M, m, R, g symbols with $M = 2, m = 1, R = 1, g = 9.8$ values. Test your numerical functions of $\ddot{x}$ and $\ddot{\theta}$ by evaluating them given $x = 0, \theta = 0.1, \dot{x} = 0, \dot{\theta} = 0$ as function inputs.

Turn in: A copy of the code used to symbolically solve and numerically evaluate the solutions of Euler-Lagrange equations (i.e. $\ddot{x}$ and $\ddot{\theta}$). Include the code output, consisting of symbolic expression of $\ddot{x}$ and $\ddot{\theta}$, as well as your test results for the numerical evaluations.

[12]:
```
# define constants (as symbols)
t, m, M, R, g = sym.symbols(r't, m, M, R, g')

# define states and time derivatives
x = sym.Function(r'x')(t)
th = sym.Function(r'\theta')(t)
q = sym.Matrix([x, th])

xdot = x.diff(t)
thdot= th.diff(t)
qdot = q.diff(t)

xddot = xdot.diff(t)
thddot = thdot.diff(t)
qddot = qdot.diff(t)
```

HOMEWORK SOLUTION
DO NOT DISTRIBUTE

```
# compute pendulum's velocity in x and y direction
# this is under the "world frame", from where we measure
# the position of the cart as x(t)
pen_xdot = xdot + R*thdot*sym.cos(th)
pen_ydot = R*thdot*sym.sin(th)

# compute Lagrangian
ke = 0.5*M*xdot**2 + 0.5*m*(pen_xdot**2 + pen_ydot**2)
pe = m*g*R*sym.cos(th)
L = ke - pe

print('Lagrangian: ')
display(L.simplify())

# compute the derivatives we needed
L = sym.Matrix([L])
dLdq = L.jacobian(q).T
dLdqdot = L.jacobian(qdot).T
ddLdqdot_dt = dLdqdot.diff(t)

# define Euler-Lagrange equations
el_eqns = sym.Eq(ddLdqdot_dt-dLdq, sym.Matrix([0, 0]))
print('\nEuler-Lagrange equations: ')
display(el_eqns.simplify())

# solve Euler-Lagrange equations
el_solns = sym.solve(el_eqns, [qddot[0], qddot[1]])
print('\nSymbolic solution for xddot:')
display(el_solns[xddot].simplify())
print('\nSymbolic solution for thddot:')
display(el_solns[thddot].simplify())

# substitute constants into symbolic solutions
subs_dict = {m:1, M:2, g:9.8, R:1}
xddot_sol = el_solns[xddot].subs(subs_dict)
thddot_sol = el_solns[thddot].subs(subs_dict)

# evaluate solutions as numerical functions
xddot_func = sym.lambdify([x, th, xdot, thdot], xddot_sol)
thddot_func = sym.lambdify([x, th, xdot, thdot], thddot_sol)

# test numerical solutions
s0 = [0, 0.1, 0, 0]
print('\nxddot(0) = {:.2f}'.format(xddot_func(*s0))) # or xddot_func(s0[0],␣
 ↪s0[1], s0[2], s0[3])
print('thddot(0) = {:.2f}'.format(thddot_func(*s0)))
```

HOMEWORK SOLUTION
DO NOT DISTRIBUTE

```
Lagrangian:
```

$$0.5M\left(\frac{d}{dt}x(t)\right)^2 - Rgm\cos\left(\theta(t)\right) + 0.5m\left(R^2\left(\frac{d}{dt}\theta(t)\right)^2 + 2R\cos\left(\theta(t)\right)\frac{d}{dt}\theta(t)\frac{d}{dt}x(t) + \left(\frac{d}{dt}x(t)\right)^2\right)$$

```
Euler-Lagrange equations:
```

$$\begin{bmatrix}0\\0\end{bmatrix} = \begin{bmatrix}1.0M\frac{d^2}{dt^2}x(t) + m\left(-R\sin\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2 + R\cos\left(\theta(t)\right)\frac{d^2}{dt^2}\theta(t) + \frac{d^2}{dt^2}x(t)\right)\\ Rm\left(R\frac{d^2}{dt^2}\theta(t) - g\sin\left(\theta(t)\right) + \cos\left(\theta(t)\right)\frac{d^2}{dt^2}x(t)\right)\end{bmatrix}$$

```
Symbolic solution for xddot:
```

$$\frac{m\left(R\left(\frac{d}{dt}\theta(t)\right)^2 - g\cos\left(\theta(t)\right)\right)\sin\left(\theta(t)\right)}{M + m\sin^2\left(\theta(t)\right)}$$

```
Symbolic solution for thddot:
```

$$\frac{\left(Mg - Rm\cos\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2 + gm\right)\sin\left(\theta(t)\right)}{R\left(M + m\sin^2\left(\theta(t)\right)\right)}$$

```
xddot(0) = -0.48
thddot(0) = 1.46
```

1.8 Problem 8 (10pts)

Based on your symbolic and numerical solutions for $\ddot{x}(t)$ and $\ddot{\theta}(t)$, which are now functions of $x(t), \theta(t), \dot{x}(t)$ and $\dot{\theta}(t)$, simulate the system for $t \in [0, 10]$, with initial condition as $x(0) = 0, \theta(0) = 0.1, \dot{x}(0) = 0, \dot{\theta}(0) = 0$, using the numerical integration and simulation functions provided below. Plot the trajectories of $x(t)$ and $\theta(t)$ versus time.

Hint 1: The numerical simulation function here can only simulate systems with first-order dynamics, which means the dynamics function feeded to numerical integration and simulation functions needs to return the first-order time derivative of the input state. This might be confusing because our solutions $\ddot{x}(t)$ and $\ddot{\theta}(t)$ are second-order time derivative. The trick here is to extend the system state from $[x(t), \theta(t)]$ to $[x(t), \theta(t), \dot{x}(t), \dot{\theta}(t)]$, thus the time derivative of the state vector becomes $[\dot{x}(t), \dot{\theta}(t), \ddot{x}(t), \ddot{\theta}(t)]$. Now, when you write down the system dynamics function, the third and forth elements of input vector $\dot{x}(t)$ and $\dot{\theta}(t)$ can be put into the output vector directly, and we already know the rest two elements of the output vector, from our previous solution of Euler-Lagrange equations. More information can be found in Lecture Note 1 (Backgroud) - Section 1.4 Ordinary Differential Equations.

*Hint 2: You will need to include the numerical evaluations for $\ddot{x}(t)$ and $\ddot{\theta}(t)$ in system dynamics function, you can either use your previous **lambdify()** results, or hand code the equations from previous symbolic solutions. Here using **lambdify()** is recommended, since in later homeworks you will have much more complicated equations.*

HOMEWORK SOLUTION
DO NOT DISTRIBUTE

Turn in: A copy of the code used to simulate the system, with the plot of the simulated trajectories.

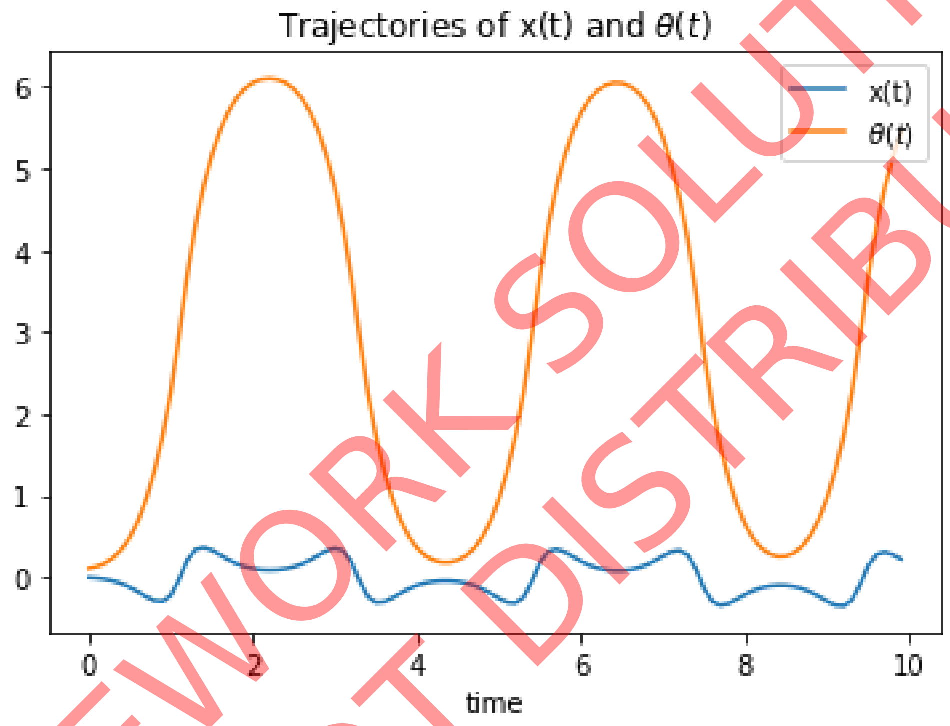
[13]:
```
# define functions to be used for trajectory plotting
def integrate(f, xt, dt):
    """
    This function takes in an initial condition x(t) and a timestep dt,
    as well as a dynamical system f(x) that outputs a vector of the
    same dimension as x(t). It outputs a vector x(t+dt) at the future
    time step.

    Parameters
    ============
    dyn: Python function
        derivate of the system at a given step x(t),
        it can considered as \dot{x}(t) = func(x(t))
    xt: NumPy array
        current step x(t)
    dt:
        step size for integration

    Return
    ============
    new_xt:
        value of x(t+dt) integrated from x(t)
    """
    k1 = dt * f(xt)
    k2 = dt * f(xt+k1/2.)
    k3 = dt * f(xt+k2/2.)
    k4 = dt * f(xt+k3)
    new_xt = xt + (1/6.) * (k1+2.0*k2+2.0*k3+k4)
    return new_xt

def simulate(f, x0, tspan, dt, integrate):
    """
    This function takes in an initial condition x0, a timestep dt,
    a time span tspan consisting of a list [min_time, max_time],
    as well as a dynamical system f(x) that outputs a vector of the
    same dimension as x0. It outputs a full trajectory simulated
    over the time span of dimensions (xvec_size, time_vec_size).

    Parameters
    ============
    f: Python function
        derivate of the system at a given step x(t),
        it can considered as \dot{x}(t) = func(x(t))
    x0: NumPy array
```

HOMEWORK SOLUTION DO NOT DISTRIBUTE

```
        initial conditions
    tspan: Python list
        tspan = [min_time, max_time], it defines the start and end
        time of simulation
    dt:
        time step for numerical integration
    integrate: Python function
        numerical integration method used in this simulation

    Return
    ============
    x_traj:
        simulated trajectory of x(t) from t=0 to tf
    """
    N = int((max(tspan)-min(tspan))/dt)
    x = np.copy(x0)
    tvec = np.linspace(min(tspan),max(tspan),N)
    xtraj = np.zeros((len(x0),N))
    for i in range(N):
        xtraj[:,i]=integrate(f,x,dt)
        x = np.copy(xtraj[:,i])
    return xtraj

# define extended dynamics for the cart pendulum system
def cart_pendulum_dyn(s):
    """
    System dynamics function (extended)

    Parameters
    ============
    s: NumPy array
        s = [x, th, xdot, thdot] is the extended system
        state vector, includng the position and
        the velocity of the particle

    Return
    ============
    sdot: NumPy array
        time derivative of input state vector,
        sdot = [xdot, thdot, xddot, thddot]

    """
    return np.array([s[2], s[3], xddot_func(s[0], s[1], s[2], s[3]),␣
 ↪thddot_func(s[0], s[1], s[2], s[3])])

# initial conditions
x0 = np.array([0, 0.1, 0, 0])
```

HOMEWORK SOLUTION
DO NOT DISTRIBUTE

```
# simulate the trajectory
traj = simulate(cart_pendulum_dyn, x0, [0, 10], 0.1, integrate)

# plot x(t) and th(t)
plt.plot(np.arange(100)*0.1, traj[0:2].T)
plt.title(r'Trajectories of x(t) and $\theta(t)$')
plt.xlabel("time")
plt.legend(('x(t)', r'$\theta(t)$'), loc='upper right')
plt.show()
```



HOMEWORK SOLUTION
DO NOT DISTRIBUTE