HW4

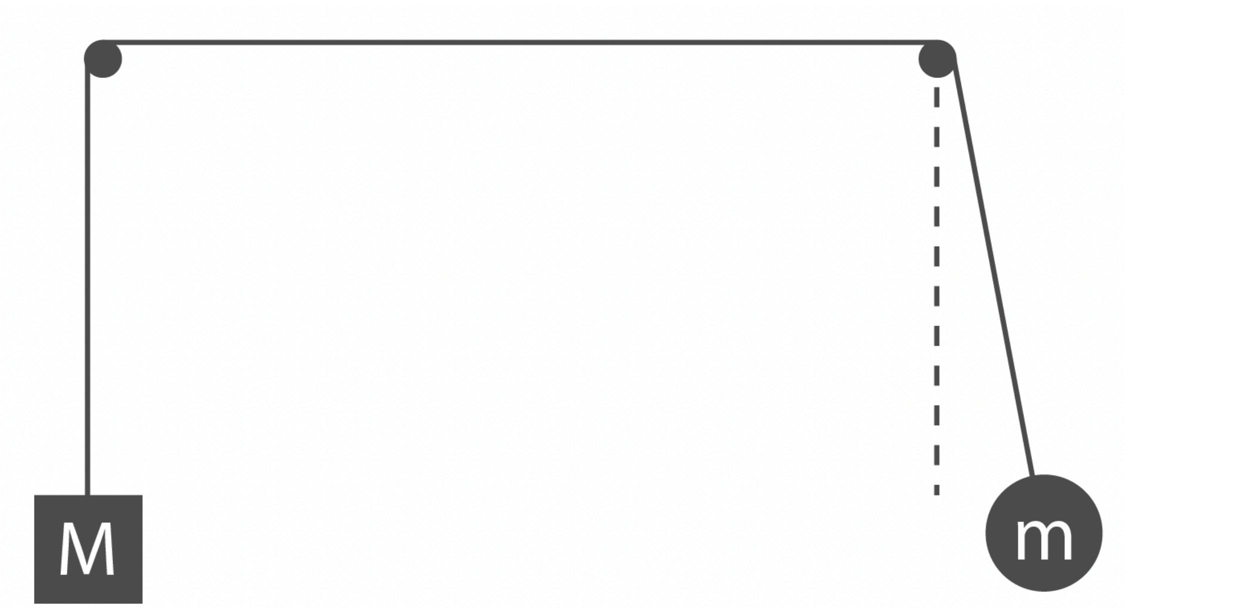
December 11, 2021

1 Swinging Atwood Machine

We study the follwoing system where there are two masses connected and

The lagrangian can be written as

[ ]:
```
from PIL import Image as Img
```

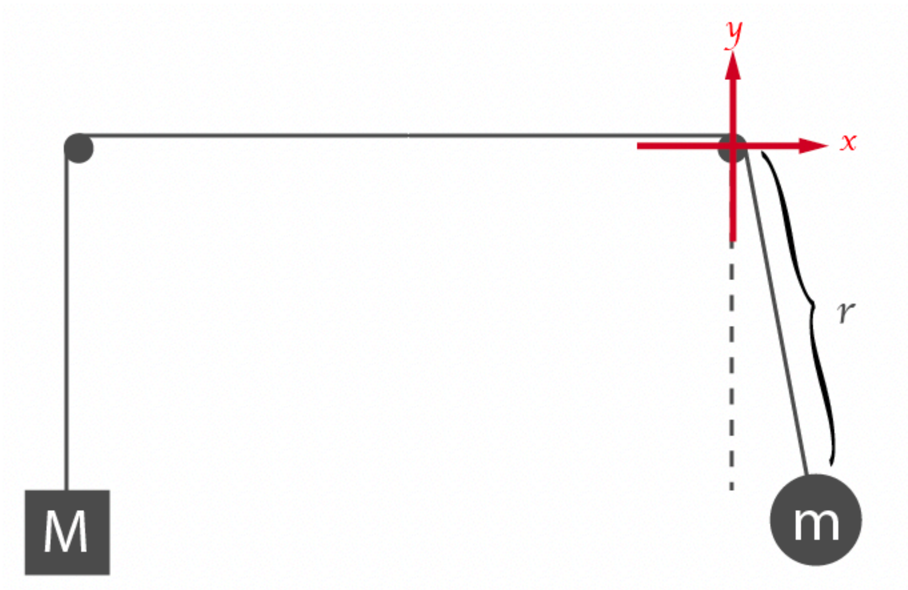


2 The Problem

We consider a system as the above picture. There are two masses m and M hangged from two supports by an inextensible string of length b. The spherical mass is allowed to swing with an angle of θ as a pendulum. Thus pendulum has is r long and it can change. The cubic mass is only allowed to move vertically.

3 The Lagrangian

3.1 The Mass m



For the spherical mass m and using a cartesian coordinate as in the above picture, we write the Lagrangian as,

$$L_m = \frac{m}{2}(\dot{x}^2 + \dot{y}^2) - mgy$$

But we have a constraint on the coordinates as

$$\begin{cases} x = r\sin(\theta) \\ y = -r\cos(\theta) \end{cases} \qquad \Rightarrow \qquad \begin{cases} \dot{x} = \dot{r}\sin(\theta) + r\cos(\theta) \\ \dot{y} = -\dot{r}\cos(\theta) + \sin(\theta) \end{cases}$$

So

$$\dot{x}^2 + \dot{y}^2 = \dot{r}^2 + r^2\dot{\theta}^2$$

So L_m becomes

$$L_m = \frac{m}{2}(\dot{r}^2 + r^2\dot{\theta}^2) + mgr\cos(\theta)$$

3.2 The Mass M

The lgransgian for the cubic mass M with coordiante system as above is

$$L_M = \frac{M}{2}(\dot{x}^2 + \dot{y}^2) - Mgy$$

The constraint is

$$\begin{cases} x = 0 \\ y = b - r \qquad \text{we ignore the horizontal length of the string} \end{cases}$$

Hence,

$$L_M = \frac{M}{2}\left[\frac{d}{dt}(b-r)\right]^2 - Mgy = \frac{M}{2}\dot{r}^2 - Mgr$$

3.3 The Entire System

We add up the two indivisual Lgrngian to get the one of the systme, then

$$L_s = L_m + L_M = \left[\frac{m}{2}(\dot{r}^2 + r^2\dot{\theta}^2) + mgr\cos(\theta)\right] + \left[\frac{M}{2}\dot{r}^2 - Mgr\right]$$

Hence,

$$L_s = \frac{m+M}{2}\dot{r}^2 + \frac{m}{2}r^2\dot{\theta}^2 + gr(m\cos(\theta) - M)$$

3.4 The Lagrange Equation of Motions

Starting with the equtaion

$$\frac{\partial L_s}{\partial r} - \frac{d}{dt}\frac{\partial L_s}{\partial \dot{r}} = 0$$

Then,

$$\frac{\partial L_s}{\partial r} = mr\dot{\theta}^2 + g(m\cos(\theta) - M) \quad \text{and} \quad \frac{d}{dt}\frac{\partial L_s}{\partial \dot{r}} = (m+M)\ddot{r}$$

Hence,

$$(m+M)\ddot{r} - mr\dot{\theta}^2 - gm\cos(\theta) + gM = 0$$

Now for doing the same for θ

$$\frac{\partial L_s}{\partial \theta} - \frac{d}{dt}\frac{\partial L_s}{\partial \dot{\theta}} = 0$$

So we get

$$\frac{\partial L_s}{\partial \theta} = -mgr\sin(\theta) \quad \text{and} \quad \frac{d}{dt}\frac{\partial L_s}{\partial \dot{\theta}} = mr^2\ddot{\theta} + 2mr\dot{r}\dot{\theta}$$

Hence,

$$mr^2\ddot{\theta} + 2mr\dot{r}\dot{\theta} + mgr\sin(\theta) = 0$$

To add up, we have these two EOMs,

$$(m+M)\ddot{r} - mr\dot{\theta}^2 - gm\cos(\theta) + gM = 0$$

$$mr^2\ddot{\theta} + 2mr\dot{r}\dot{\theta} + mgr\sin(\theta) = 0$$

Let us divide those two by m, and define the mass ration $\mu = M/m$, so get the follwoing equations

$$(1+\mu)\ddot{r} - r\dot{\theta}^2 - g\cos(\theta) + g\mu = 0$$

$$r^2\ddot{\theta} + 2r\dot{r}\dot{\theta} + gr\sin(\theta) = 0$$

Finaly we isolate $\ddot{r}$, and $\ddot{\theta}$, like so

$$\ddot{r} = \frac{r\dot{\theta}^2 + g\cos(\theta) - g\mu}{1+\mu}, \qquad \ddot{\theta} = -\frac{2r\dot{\theta} + g\sin\theta}{r}$$

We will use python to interate those equations for different initial values

4 Writing The program

[ ]:
```python
import scipy as sc
import sympy as sp
from sympy import Function, Derivative, Eq
from sympy.abc import t, g, mu, r
```

4.1 Define the ODE of the radial coordinate θ

[ ]:
```python
# define symbols r and theta as functions of t
r, theta = sp.symbols(r'r \theta', cls=Function)
# define the ODE of the radial coordinate
radial = Eq( (1+mu) * r(t).diff(t, t) - r(t) * theta(t).diff(t)**2 - g * sp.
 ↪cos(theta(t)) + g * mu, 0 )
```

```
radial
```

[ ]:

$$g\mu - g\cos(\theta(t)) + (\mu+1)\frac{d^2}{dt^2}r(t) - r(t)\left(\frac{d}{dt}\theta(t)\right)^2 = 0$$

[ ]:
```python
from scipy.optimize import fsolve

# find the solution
radial_sol = sp.dsolve(radial.subs({mu: 3}), r(t))

radial_sol
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
/var/folders/yx/2byr_xhj00ldw_7gxt43m5kc0000gn/T/ipykernel_57962/3191131889.py␣
↪in <module>
      2
      3 # find the solution
----> 4 radial_sol = sp.dsolve(radial.subs({mu: 3}), r(t))
      5
      6 radial_sol

~/projects/python/env/lib/python3.10/site-packages/sympy/solvers/ode/ode.py in␣
↪dsolve(eq, func, hint, simplify, ics, xi, eta, x0, n, **kwargs)
    602
    603         # See the docstring of _desolve for more details.
--> 604         hints = _desolve(eq, func=func,
    605             hint=hint, simplify=True, xi=xi, eta=eta, type='ode',␣
↪ics=ics,
    606             x0=x0, n=n, **kwargs)

~/projects/python/env/lib/python3.10/site-packages/sympy/solvers/deutils.py in␣
↪_desolve(eq, func, hint, ics, simplify, prep, **kwargs)
    239                 str(eq) + " is not a solvable differential equation in ␣
↪+ str(func))
    240         else:
--> 241             raise NotImplementedError(dummy + "solve" + ": Cannot solve␣
↪" + str(eq))
    242     if hint == 'default':
    243         return _desolve(eq, func, ics=ics, hint=hints['default'],␣
↪simplify=simplify,

NotImplementedError: solve: Cannot solve -g*cos(\theta(t)) + 3*g -␣
↪r(t)*Derivative(\theta(t), t)**2 + 4*Derivative(r(t), (t, 2))
```

[ ]:
```
def radial(r, mu, g):
    return
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/var/folders/yx/2byr_xhj00ldw_7gxt43m5kc0000gn/T/ipykernel_57962/143045527.py i␣
↪<module>
----> 1 sp.constnts.g

AttributeError: module 'sympy' has no attribute 'constnts'
```

[ ]:
```
from numpy import *
import matplotlib.pyplot as plt
import os

seterr(all='raise')       #If we encounter overflows, trigger an error (that'll␣
↪be resolved by 'try-catch')

def RKalgorithm():
    global flag
    flag         = False
    variables[0] = array([r0, rdot0, -theta0, -thetadot0])  #Thetas are␣
↪negative to agree with the diagram in the paper that goes along with it

    print(indices)
    for i in range(indices - 1):
        try:
            if(((theta0 == 0 and thetadot0 == 0) or theta0 == pi) and i == 0):
                print('Warning: Unstable or stable equilibrium chosen as␣
↪initial value. Results may be unintended.')
            if(variables[i][0] <= (threshold * r0)):
                print('The trajectory came within the threshold for identifying␣
↪a singularity (%.4f%% of r0). The program has finished early (%.2f s) to␣
↪avoid infinities.' % ((threshold * r0 * 100), (i * step)))
                break

            k1 = step * RKaccel(variables[i], times[i])
                k2 = step * RKaccel(variables[i] + k1 / 2, times[i] + step/2)
                k3 = step * RKaccel(variables[i] + k2 / 2, times[i] + step/2)
                k4 = step * RKaccel(variables[i] + k3, times[i] + step)

                variables[i + 1] = variables[i] + k1/6 + k2/3 + k3/3 + k4/6

               except FloatingPointError:    #This isn't actually an error, but␣
↪we've told the system to associate OVERFLOWS with errors.
                flag = True
```

```
                    print('A Runtime Warning was triggered, indicating␣
↪infinities as r -> 0. Increase the singularity threshold.')
                    print('As a result, plotting procedures have been abandoned␣
↪to avoid an erroneous display.')
                    break

def RKaccel(variables, times):
    radius    = variables[0]
    radiusdot = variables[1]
    theta     = variables[2]
    thetadot  = variables[3]

    radiusdotdot = ((radius / (1 + mu)) * ((thetadot) ** 2)) + (((g *␣
↪cos(theta)) - (g * mu)) / (1 + mu))
    thetadotdot  = - ((g * sin(theta)) / radius) - (2 * ((radiusdot) *␣
↪(thetadot)) / radius)

    return array([radiusdot, radiusdotdot, thetadot, thetadotdot])

#Initialize algorithmic variables.
step      = 0.002
maxtime   = 50
threshold = 0.01   #singularity theshold! read the paper.

indices = int(maxtime / step)
times   = linspace(0, (indices - 1) * step, indices)

#Initialize the initial physical variables.
r0         = 1
rdot0      = 0
theta0     = -pi / 2
thetadot0  = 0
g          = 9.8
variables = zeros([indices, 4], dtype=float)

def simulate(ratio):
    global mu
    mu = ratio / 1000

    #Runge-Kutta algorithm variables
    RKalgorithm()


    #Begin plotting
    fig = plt.figure()
    ax = plt.subplot(111, projection='polar')
    ax.set_theta_zero_location("S")
```

```
    ax.plot(variables[:,2], variables[:,0], color='b', linewidth=1)
    plt.title('$\mu = %.3f$, $r_0 = %.3f$, $\\theta_0 = %.3f^\degree$' % (mu,␣
↪r0, -theta0 * 180 / pi), y = 1.06)
    plt.axis('off')
    ax.grid(True)
    if(flag == False):
        print('Calculation was successful for mu = %.3f.' % mu)
        savePath = os.getcwd() + r'\images\mu' + str(int(ratio)) + '.png'
        plt.savefig(savePath)
    plt.close(fig)

# import csv
#
# with open('data.csv', 'w') as writeFile:
#     writer = csv.writer(writeFile)
#     for var in variables:
#         writer.writerow(var)
#
# writeFile.close()

from numpy import *

start = 4500
end = 6500
mus = linspace(start, end, end-start+1)

for mu in mus:
    simulate(mu)
```

```
<function indices at 0x108b3c430>
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/yx/2byr_xhj00ldw_7gxt43m5kc0000gn/T/ipykernel_57962/536333101.py i␣
↪<module>
     99
    100 for mu in mus:
--> 101     simulate(mu)

/var/folders/yx/2byr_xhj00ldw_7gxt43m5kc0000gn/T/ipykernel_57962/536333101.py i␣
↪simulate(ratio)
     66
     67     #Runge-Kutta algorithm variables
---> 68     RKalgorithm()
     69
     70
```

```
/var/folders/yx/2byr_xhj00ldw_7gxt43m5kc0000gn/T/ipykernel_57962/536333101.py i
↪RKalgorithm()
     29
     30     print(indices)
---> 31     for i in range(indices - 1):
     32         try:
     33             if(((theta0 == 0 and thetadot0 == 0) or theta0 == pi) and i
↪== 0):

TypeError: unsupported operand type(s) for -: 'function' and 'int'
```