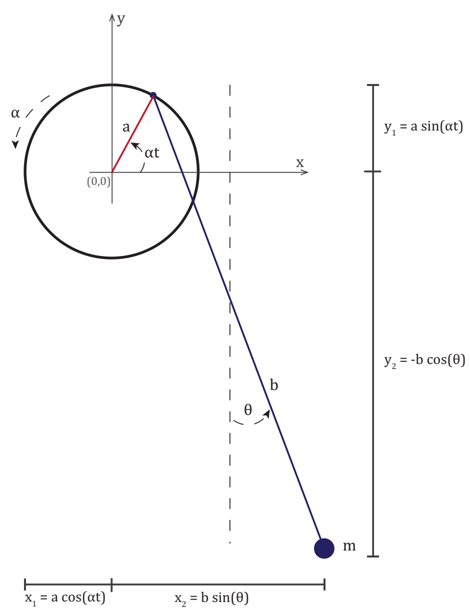
Lagrangian & Hamiltonian Dynamics
Pendulum on a Rotating Rim

BENJAMIN HOWE

bhowe@student.bridgew.edu

I. DESCRIPTION

The pendulum on a rotating rim. A simple pendulum of length b and mass m moves on a mass-less rim of radius a rotating with constant angular velocity ω. Find the equation of motion for the mass.



II. ANALYSIS

1. The rim is mass-less, so all energy the terms are only for the pendulum mass, note: $1/2 I\omega^2 = 0$

2. The pendulum, however:

   a) has kinetic energy,
   $$\mathbf{T} = 1/2 m\dot{x}^2 + 1/2 m\dot{y}^2 + 1/2 I\omega^2 \tag{1}$$

   b) and gravitational potential energy,
   $$\mathbf{U} = mg\left(y - y_0\right) \tag{2}$$

3. A change of coordinates from Cartesian to polar is the most sensible way to proceed.

$$\begin{aligned} x &= a\cos(\alpha t) + b\sin\theta & \dot{x} &= -a\alpha\sin(\alpha t) + b\sin(\theta)\dot{\theta} \\ y &= a\sin(\alpha t) - b\cos\theta & \dot{y} &= a\alpha\cos(\alpha t) + b\sin(\theta)\dot{\theta} \end{aligned} \tag{3}$$

4. From here, the plan is to get a differential equation, or two, in just θ and $\dot{\theta}$.
5. Using python code we will perform a 4th order Runge-Kutta numerical analysis and plot the pendulum's position $x(t)$ and $y(t)$.

III. The Math

$$\mathbf{T} = \frac{1}{2}m\left(a^2\alpha^2 + b^2\dot{\theta}^2 + 2ab\alpha\dot{\theta}\left[\sin\theta\cos\alpha t - \sin\alpha t\cos\theta\right]\right) \tag{4}$$

$$[\text{Use: } \sin(\alpha\pm\beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta]$$

$$\mathbf{T} = \frac{1}{2}m\left(m^2\omega^2 + b^2\dot{\theta}^2 + 2ab\alpha\dot{\theta}\sin(\theta-\alpha t)\right)$$

$$\mathbf{U} = mg\left(a\sin\alpha t - b\cos\theta\right) \tag{5}$$

$$\mathcal{L} = \mathbf{T} - \mathbf{U} \tag{6}$$

$$\mathcal{L} = \frac{1}{2}m\left(m^2\alpha^2 + b^2\dot{\theta}^2 + 2ab\alpha\dot{\theta}\sin(\theta-\alpha t)\right) - mg\left(a\sin\alpha t - b\cos\theta\right) \tag{7}$$

$$\frac{\partial\mathcal{L}}{\partial\theta} = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial\mathcal{L}}{\partial\dot{\theta}} \tag{8}$$

$$\frac{\partial\mathcal{L}}{\partial\theta} = \frac{1}{2}m\left(2ab\alpha\dot{\theta}\cos(\theta-\alpha t)\right) - mgb\sin\theta \tag{9}$$

$$\frac{\partial\mathcal{L}}{\partial\dot{\theta}} = \frac{1}{2}m\left(2b^2\dot{\theta}^2 + 2ab\alpha\sin(\theta-\alpha t)\right) \tag{10}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial\mathcal{L}}{\partial\dot{\theta}} = mb^2\ddot{\theta} + mab\alpha\cos(\theta-\alpha t)\left(\dot{\theta}-\alpha\right) \tag{11}$$

$$mb^2\ddot{\theta} + mab\alpha\cos(\theta-\alpha t)\left(\dot{\theta}-\alpha\right) = mab\alpha\dot{\theta}\cos(\theta-\alpha t) - mgb\sin\theta \tag{12}$$

$$\ddot{\theta} = \frac{a\alpha^2}{b}\cos(\theta-\alpha t) - \frac{g}{b}\sin\theta \tag{13}$$

IV. Code

As usual, it is necessary to import various libraries to perform the task at hand.

```
In [1]: import math
        import numpy as np
        from matplotlib import pyplot as plt
        import matplotlib.animation
```

Here we introduce the constant terms in our aparatus, create the empty arrays we will use and define the intial conditions.

```
In [2]: # some constant values
        a = 0.35 # radius of rim in meters
        b = 1.1 # length of pendulum in meters
        m = 0.7 # mass of bob in kg
        alpha = 2.15 # angular speed of rim in rad/sec
        g = 9.8 # acceleration due to gravity
        N = 10000 # number of iterations
        h = .001 # step size
```

```
In [3]: # create some empty arrays
        theta = np.zeros(N+1)
        omega = np.zeros(N+1)
        thetadot = np.zeros(N+1)
        omegadot = np.zeros(N+1)
        t = np.zeros(N+1)
        x = np.zeros(N+1)
        y = np.zeros(N+1)
```

```
In [4]: # initial condition
        theta[0] = np.pi/2.0
```

From the math (see part IV), we have two first order differential equations to define.

```
In [5]: # define some functions
        def thetadot(omega):
            return omega
        def omegadot(theta, t):
            return (a * alpha**2 / b) * np.cos(theta - alpha * t) - (g/b) * np.sin(theta)
```

To analyze the system, we use the 4th order Runge-Kutta method.

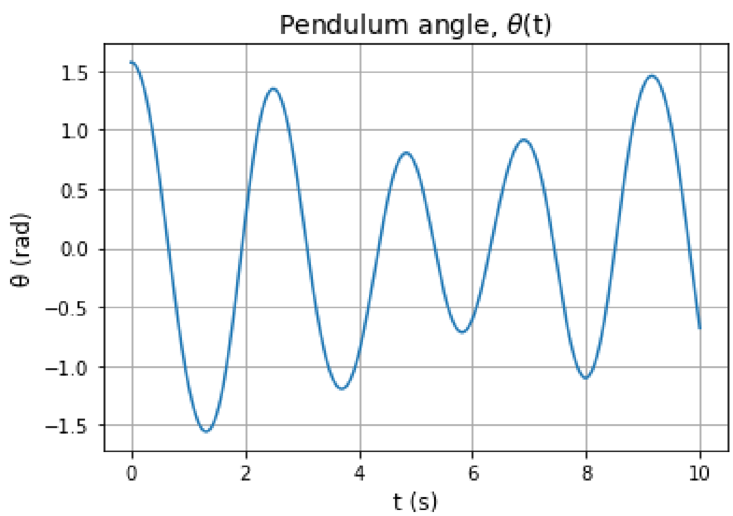
```
In [6]: for i in range(N):
            k1_theta = h * thetadot(omega[i])
            k1_omega = h * omegadot(theta[i], t[i])

            k2_theta = h * thetadot(omega[i] + k1_theta/2.0)
            k2_omega = h * omegadot(theta[i] + k1_theta/2.0, t[i] + k1_omega/2.0)
```

```
        k3_theta = h * thetadot(omega[i] + k2_theta/2.0)
        k3_omega = h * omegadot(theta[i] + k2_theta/2.0, t[i] + k2_omega/2.0)

        k4_theta = h * thetadot(omega[i] + k3_theta)
        k4_omega = h * omegadot(theta[i] + k3_omega, t[i] + k3_omega)

        theta[i+1] = theta[i] + (k1_theta + 2*k2_theta + 2*k3_theta + k4_theta)/6.0
        omega[i+1] = omega[i] + (k1_omega + 2*k2_omega + 2*k3_omega + k4_omega)/6.0
        t[i+1] = t[i] + h
```

First, we will plot the angle theta vs. time.

```
In [7]: plt.plot(t,theta)
       plt.grid(b=True, which='both', color='0.65', linestyle='-')
        plt.title('Pendulum angle, $\u03b8$(t)', fontsize=14)
        plt.xlabel('t (s)', fontsize=12)
        plt.ylabel('\u03b8 (rad)', fontsize=12)
```

```
Out[7]: Text(0, 0.5, 'theta')
```



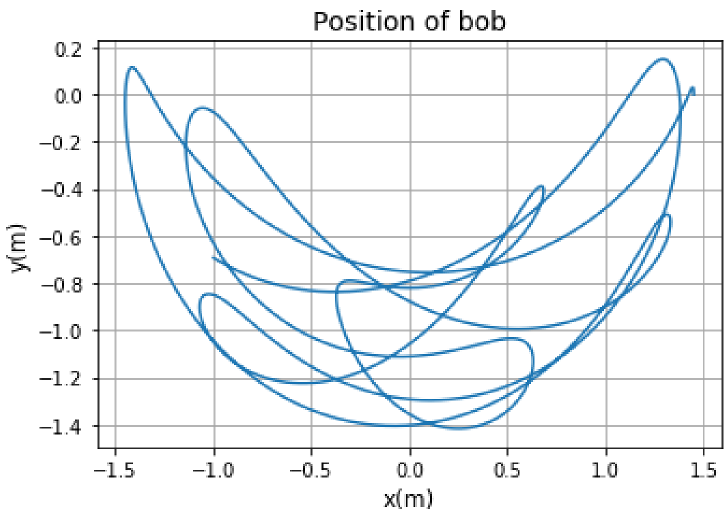
It is useful to switch back to Cartesian coordinates to visualize the motion of the aparatus.

```
In [8]: # compute some x, y coordinates
        def x_vec(theta, t):
            return a * np.cos(alpha * t) + b * np.sin(theta)
        def y_vec(theta, t):
            return a * np.sin(alpha * t) - b * np.cos(theta)
```

```
In [9]: for i in range(N+1):
            x[i] = x_vec(theta[i], t[i])
            y[i] = y_vec(theta[i], t[i])
```

```
In [10]: plt.plot(x,y)
         plt.grid(b=True, which='both', color='0.65', linestyle='-')
         plt.title('Position of bob', fontsize=14)
         plt.xlabel('x(m)', fontsize=12)
         plt.ylabel('y(m)', fontsize=12)
```

```
Out[10]: Text(0, 0.5, 'y')
```



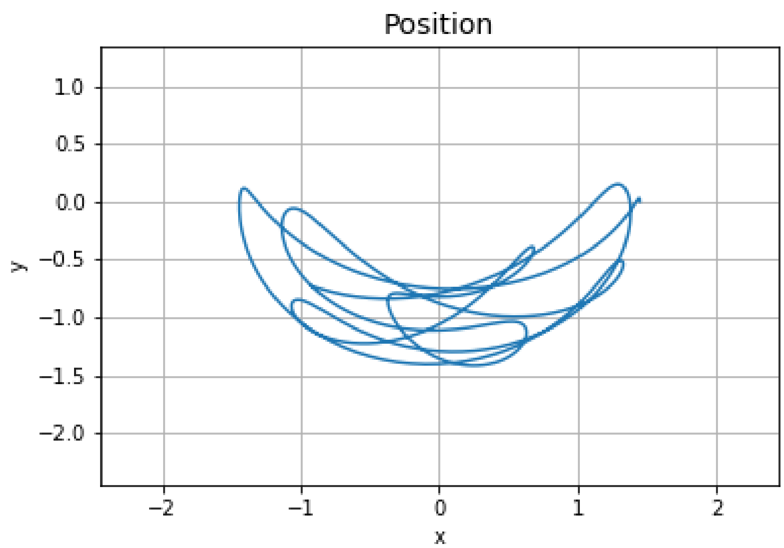
To enhance the presentation we now animate the motion of the pendulum bob.

```
In [11]: # trim the data down for the animation
         # 10000 points is too much to process
         t_new = t[0::20]
         x_new = x[0::20]
         y_new = y[0::20]
```

```
In [12]: matplotlib.rcParams['animation.embed_limit'] = 2**128
         # even with fewer values it is still necessary to increase the program's memory

         fig, ax = plt.subplots()
         ax.axis([-(a+b+1), (a+b+1), -(a+b+1), (a+1)])
         plt.grid()
         plt.title('Position', fontsize=14)
         plt.xlabel('x')
         plt.ylabel('y')
         l, = ax.plot([],[])

         def animate(i):
             l.set_data(x_new[:i], y_new[:i])

         ani = matplotlib.animation.FuncAnimation(fig, animate, frames=len(t_new))
```

```python
from IPython.display import HTML
HTML(ani.to_jshtml())
```

Out[12]: <IPython.core.display.HTML object>



V. References

1. Credit for animation code:
   https://stackoverflow.com/questions/43445103/inline-animations-in-jupyter.
2. Download from my GitHub site:
   https://github.com/benhowe75/Physics-439.git
3. Inspiration for the problem:
   http://www.astro.uwo.ca/
   http://www.astro.uwo.ca/~houde/courses/PDF%20files/physics350/Lagrange.pdf