In this notebook, you will implement the kinematic bicycle model. The model accepts velocity and steering rate inputs and steps through the bicycle kinematic equations. Once the model is implemented, you will provide a set of inputs to drive the bicycle in a figure 8 trajectory.

The bicycle kinematics are governed by the following set of equations:

$$\begin{aligned}\dot{x}_c &= v\cos(\theta+\beta)\\ \dot{y}_c &= v\sin(\theta+\beta)\\ \dot{\theta} &= \frac{v\cos\beta\tan\delta}{L}\\ \dot{\delta} &= \omega\\ \beta &= \tan^{-1}\left(\frac{l_r\tan\delta}{L}\right)\end{aligned}$$

where the inputs are the bicycle speed v and steering angle rate ω. The input can also directly be the steering angle δ rather than its rate in the simplified case. The Python model will allow us both implementations.

In order to create this model, it's a good idea to make use of Python class objects. This allows us to store the state variables as well as make functions for implementing the bicycle kinematics.

The bicycle begins with zero initial conditions, has a maximum turning rate of 1.22 rad/s, a wheelbase length of 2m, and a length of 1.2m to its center of mass from the rear axle.

From these conditions, we initialize the Python class as follows:

In [2]:
```
from notebook_grader import BicycleSolution, grade_bicycle
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

class Bicycle():
    def __init__(self):
        self.xc = 0
        self.yc = 0
        self.theta = 0
        self.delta = 0
        self.beta = 0

        self.L = 2
        self.lr = 1.2
        self.w_max = 1.22

        self.sample_time = 0.01

    def reset(self):
        self.xc = 0
        self.yc = 0
        self.theta = 0
        self.delta = 0
        self.beta = 0
```

A sample time is required for numerical integration when propagating the kinematics through time. This is set to 10 milliseconds. We also have a reset function which sets all the state variables back to 0.

With this sample time, implement the kinematic model using the function $step$ defined in the next cell. The function should take speed + angular rate as inputs and update the state variables. Don't forget about the maximum turn rate on the bicycle!

In [3]:
```
class Bicycle(Bicycle):
    def step(self, v, w):
        # ================================
        #  Implement kinematic model here
        # ================================

        #Applying this condition so that we dont exceed max limit on w

        if w > 0:
            w = min(w, self.w_max)
        else:
            w = max(w, -self.w_max)

        #sampling time
        dt = 10e-3

        #calculating change in variables

        self.beta = np.arctan((self.lr * np.tan(self.delta)) / self.L)

        theta_dot = ( v*np.tan(self.delta)*np.cos(self.beta) ) / self.L

        delta_dot = w

        xc_dot = v*np.cos(self.theta + self.beta)
        yc_dot = v*np.sin(self.theta + self.beta)

        #Updating Variables

        self.xc += xc_dot*dt
        self.yc += yc_dot*dt

        self.theta += theta_dot*dt
        self.delta += delta_dot*dt




        pass
```

With the model setup, we can now start giving bicycle inputs and producing trajectories.

Suppose we want the model to travel a circle of radius 10 m in 20 seconds. Using the relationship between the radius of curvature and the steering angle, the desired steering angle can be computed.

$$\begin{aligned}\tan\delta &= \frac{L}{r} \\ \delta &= \tan^{-1}(\frac{L}{r}) \\ &= \tan^{-1}(\frac{2}{10}) \\ &= 0.1974\end{aligned}$$

If the steering angle is directly set to 0.1974 using a simplied bicycled model, then the bicycle will travel in a circle without requiring any additional steering input.

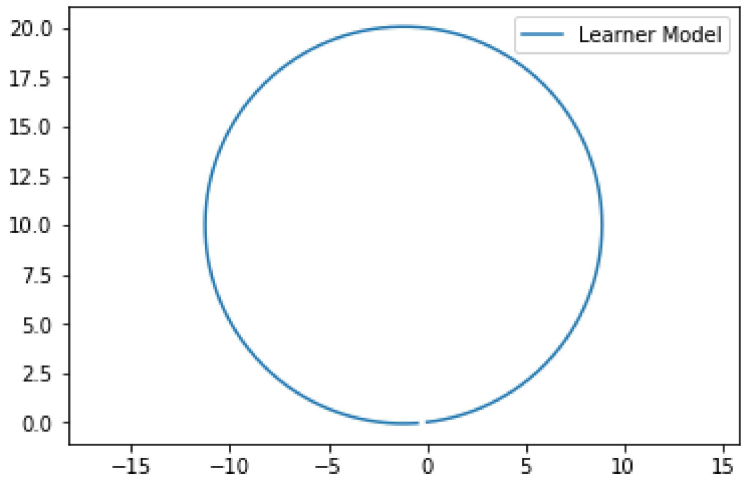
The desired speed can be computed from the circumference of the circle:

$$\begin{aligned}v &= \frac{d}{t} \\ &= \frac{2\pi 10}{20} \\ &= \pi\end{aligned}$$

We can now implement this in a loop to step through the model equations. We will also run our bicycle model solution along with your model to show you the expected trajectory. This will help you verify the correctness of your model.

In [4]:

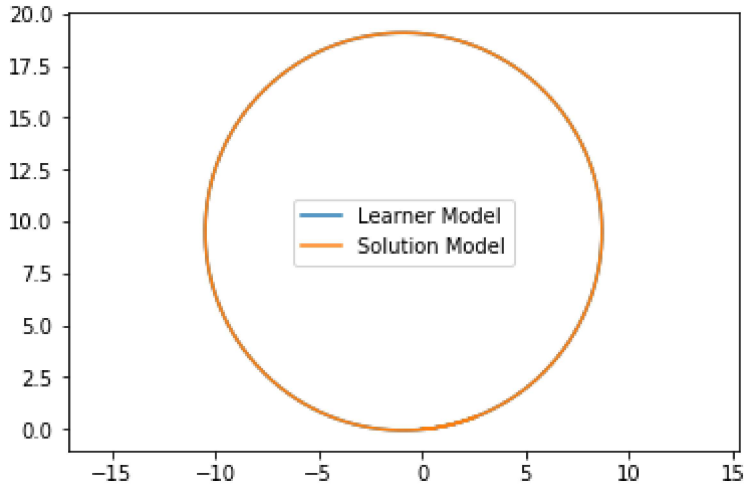
```
sample_time = 0.01
time_end = 20

model = Bicycle()
solution_model = BicycleSolution()

# set delta directly
model.delta = np.arctan(2/10)
solution_model.delta = np.arctan(2/10)

t_data = np.arange(0,time_end,sample_time)

x_data = np.zeros_like(t_data)
y_data = np.zeros_like(t_data)

x_solution = np.zeros_like(t_data)
y_solution = np.zeros_like(t_data)

#print(t_data)

for i in range(t_data.shape[0]):
    x_data[i] = model.xc
    y_data[i] = model.yc

    model.step(np.pi, 0)
    #print(model.xc)
    #print("x_data : ",x_data)

    x_solution[i] = solution_model.xc
    y_solution[i] = solution_model.yc
    solution_model.step(np.pi, 0)

    #model.beta = 0
    #solution_model.beta=0

plt.axis('equal')
plt.plot(x_data, y_data,label='Learner Model')
#plt.plot(x_solution, y_solution,label='Solution Model')
plt.legend()
plt.show()
```



The plot above shows the desired circle of 10m radius. The path is slightly offset which is caused by the sideslip effects due to β. By forcing $\beta = 0$ through uncommenting the last line in the loop, you can see that the offset disappears and the circle becomes centered at (0,10).

However, in practice the steering angle cannot be directly set and must be changed through angular rate inputs ω. The cell below corrects for this and sets angular rate inputs to generate the same circle trajectory. The speed v is still maintained at π m/s.

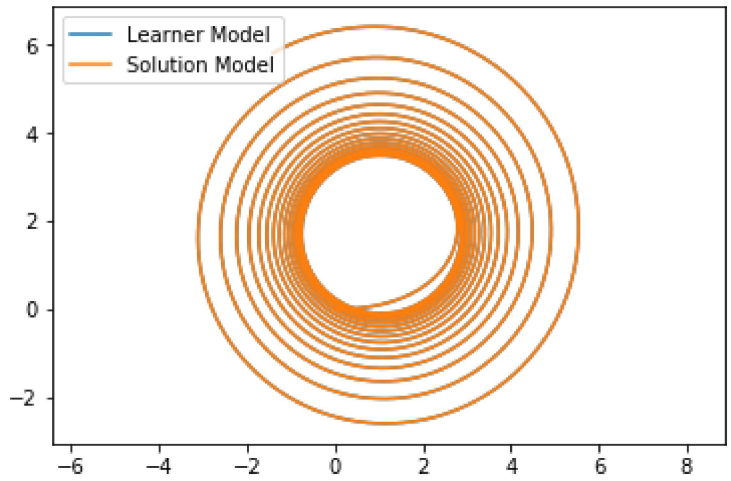
In [5]:
```
sample_time = 0.01
time_end = 20
model.reset()
solution_model.reset()

t_data = np.arange(0,time_end,sample_time)
x_data = np.zeros_like(t_data)
y_data = np.zeros_like(t_data)
x_solution = np.zeros_like(t_data)
y_solution = np.zeros_like(t_data)

for i in range(t_data.shape[0]):
    x_data[i] = model.xc
    y_data[i] = model.yc

    if model.delta < np.arctan(2/10):
        model.step(np.pi, model.w_max)
    else:
        model.step(np.pi, 0)

    x_solution[i] = solution_model.xc
    y_solution[i] = solution_model.yc

    if solution_model.delta < np.arctan(2/10):
        solution_model.step(np.pi, model.w_max)
    else:
        solution_model.step(np.pi, 0)

plt.axis('equal')
plt.plot(x_data, y_data,label='Learner Model')
plt.plot(x_solution, y_solution,label='Solution Model')
plt.legend()
plt.show()
```



Here are some other example trajectories: a square path, a spiral path, and a wave path. Uncomment each section to view.

In [6]:
```
sample_time = 0.01
time_end = 60
model.reset()
solution_model.reset()

t_data = np.arange(0,time_end,sample_time)

x_data = np.zeros_like(t_data)
y_data = np.zeros_like(t_data)

x_solution = np.zeros_like(t_data)
y_solution = np.zeros_like(t_data)

# maintain velocity at 4 m/s

v_data = np.zeros_like(t_data)
v_data[:] = 4

w_data = np.zeros_like(t_data)

# ================================
#  Square Path: set w at corners only
# ================================
w_data[670:670+100] = 0.753

w_data[670+100:670+100*2] = -0.753

w_data[2210:2210+100] = 0.753

w_data[2210+100:2210+100*2] = -0.753

w_data[3670:3670+100] = 0.753

w_data[3670+100:3670+100*2] = -0.753

w_data[5220:5220+100] = 0.753

w_data[5220+100:5220+100*2] = -0.753

# ================================
#  Spiral Path: high positive w, then small negative w
# ================================
w_data[:] = -1/100
w_data[0:100] = 1

# ================================
#  Wave Path: square wave w input
# ================================
#w_data[:] = 0
#w_data[0:100] = 1
#w_data[100:300] = -1
#w_data[300:500] = 1
#w_data[500:5700] = np.tile(w_data[100:500], 13)
#w_data[5700:] = -1

# ================================
#  Step through bicycle model
# ================================
```

```
for i in range(t_data.shape[0]):
    x_data[i] = model.xc
    y_data[i] = model.yc
    model.step(v_data[i], w_data[i])

    x_solution[i] = solution_model.xc
    y_solution[i] = solution_model.yc
    solution_model.step(v_data[i], w_data[i])

plt.axis('equal')
plt.plot(x_data, y_data,label='Learner Model')
plt.plot(x_solution, y_solution,label='Solution Model')
plt.legend()
plt.show()
```



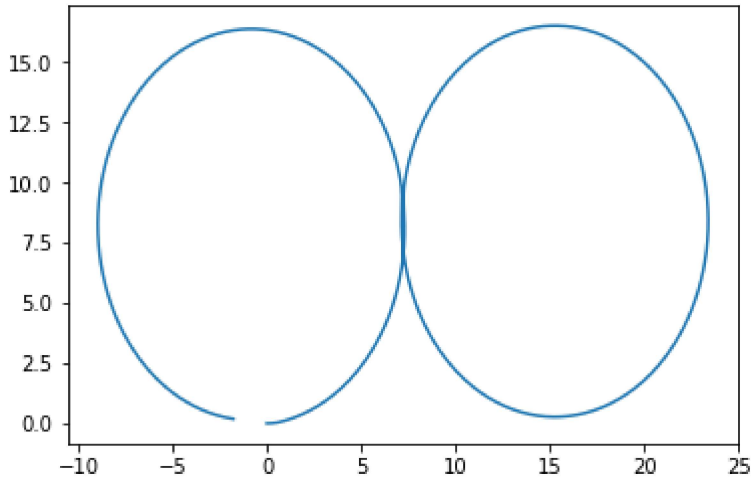
We would now like the bicycle to travel a figure eight trajectory. Both circles in the figure eight have a radius of 8m and the path should complete in 30 seconds. The path begins at the bottom of the left circle and is shown in the figure below:

Determine the speed and steering rate inputs required to produce such trajectory and implement in the cell below. Make sure to also save your inputs into the arrays v_data and w_data, these will be used to grade your solution. The cell below also plots the trajectory generated by your own model.

In [7]:

```
sample_time = 0.01
time_end = 30
radius = 8

model.reset()

t_data = np.arange(0,time_end,sample_time)

x_data = np.zeros_like(t_data)
y_data = np.zeros_like(t_data)

v_data = np.zeros_like(t_data)
w_data = np.zeros_like(t_data)

delta = 0.993 * np.arctan(model.L/radius)
vel = ( 4*np.pi*radius )/ time_end

#print(model.delta,vel)
v_data[:] = vel
#print(v_data)

for i in range(t_data.shape[0]):
#Dividing our track into 8 parts

    x_data[i] = model.xc
    y_data[i] = model.yc

    if i <= t_data.shape[0]/8:

        if delta > model.delta:
            model.step(v_data[i], model.w_max)
            w_data[i] = model.w_max
        else:
            model.step(v_data[i], 0)
            w_data[i] = 0


    elif i <= (5.07*t_data.shape[0])/8:

        if model.delta > -delta:
            model.step(v_data[i], -model.w_max)
            w_data[i] = -model.w_max

        else:
            model.step(v_data[i], 0)
            w_data[i] = 0

    else:
        if model.delta < delta:
            model.step(v_data[i], model.w_max)
            w_data[i] = model.w_max
        else:
            model.step(v_data[i], 0)
            w_data[i] = 0



#plt.axis('equal')
```

```
plt.plot(x_data, y_data)
plt.show()
```



We will now run your speed and angular rate inputs through our bicycle model solution. This is to ensure that your trajectory is correct along with your model. The cell below will display the path generated by our model along with some waypoints on a desired figure 8. Surrounding these waypoints are error tolerance circles with radius 1.5m, your solution will pass the grader if the trajectory generated stays within 80% of these circles.

In [8]:
```
grade_bicycle(t_data,v_data,w_data)
```

```
Assessment passed! Your trajectory meets 100.0% of the waypoints.
```