ME314 Homework 6 (Template)

*Please note that a **single** PDF file will be the only document that you turn in, which will include your answers to the problems with corresponding derivations and any code used to complete the problems. When including the code, please make sure you also include **code outputs**, and you don't need to include example code. Problems and deliverables that should be included with your submission are shown in **bold**.*

This Juputer Notebook file serves as a template for you to start homework, since we recommend to finish the homework using Jupyter Notebook. You can start with this notebook file with your local Jupyter environment, or upload it to Google Colab. You can include all the code and other deliverables in this notebook Jupyter Notebook supports $\LaTeX$ for math equations, and you can export the whole notebook as a PDF file. But this is not the only option, if you are more comfortable with other ways, feel free to do so, as long as you can submit the homework in a single PDF file.

Problem 1 (20pts)

Show that if $R(\theta_1)$ and $R(\theta_2) \in SO(n)$ then the product is also a rotation matrix; that is $R(\theta_1)R(\theta_2) \in SO(n)$.

Hint 1: You know this is true when $n = 2$ by direct calculation in class, but for $n \neq 2$ you should use the definition of $SO(n)$ to verify it for arbitrary n. Do not try to do this by analyzing individual components of the matrix.

Turn in: A scanned (or photograph from your phone or webcam) copy of your hand written solution. You can also use \LaTeX. If you use SymPy, then you just need to include a copy of code and the code outputs, with notes that explain why the code outputs could explain the results.

handwritten work attached at bottom of document

Problem 2 (20pts)

Show that if $g(x_1, y_1, \theta_1)$ and $g(x_2, y_2, \theta_2) \in SE(2)$ then the product satisfies $g(x_1, y_1, \theta_1)g(x_2, y_2, \theta_2) \in SE(2)$.

Turn in: A scanned (or photograph from your phone or webcam) copy of your hand written solution. You can also use \LaTeX. If you use SymPy, then you just need to include a copy of code and the code outputs, with notes that explain why the code outputs could explain the results.

handwritten work attached at bottom of document

Problem 3 (20pts)

Show that any homogeneous transformation in SE(2) can be separated into a rotation and a translation. What's the order of the two operations, which comes first? What's different if we flip the order in which we compose the rotation and translation?

Hint 1: For the rotation and translation operation, we first need to know what's the reference frame for these two operations.

Turn in: A scanned (or photograph from your phone or webcam) copy of your hand written solution. You can also use \LaTeX. If you use SymPy, then you just need to include a copy of code and the code outputs, with notes that explain why the code outputs could explain the results.

handwritten work attached at bottom of document

Problem 4 (20pts)

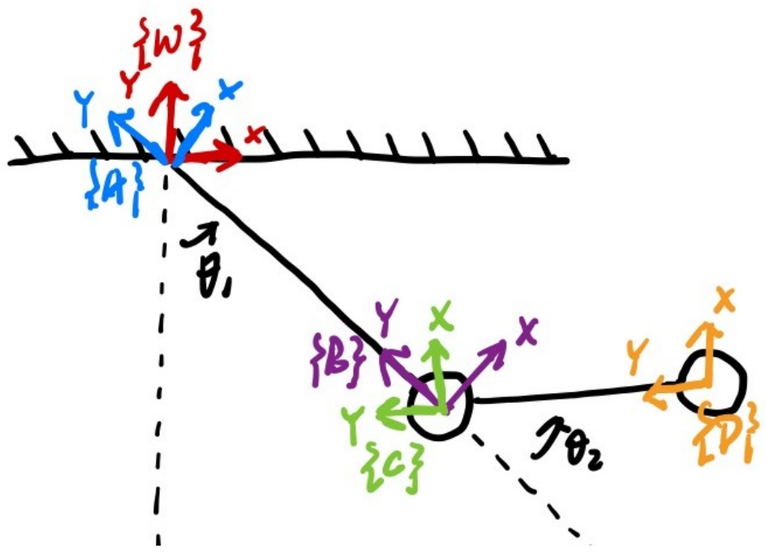
Simulate the same double-pendulum system in previous homework using only homogeneous transformation (and thus avoid using trigonometry). Simulate the system for $t \in [0, 5]$ with $dt = 0.1$. The parameters are $m_1 = m_2 = 1, R_1 = R_2 = 1, g = 9.8$ and initial conditions are $\theta_1 = \theta_2 = -\frac{\pi}{2}, \dot{\theta}_1 = \dot{\theta}_2 = 0$. Do not use functions provided in the modern robotics package for manipulating transformation matrices such as RpToTrans().

Hint 1: Same as in the lecture, you will need to define the frames by yourself in order to compute the Lagrangian. An example is shown below.

Turn in: A copy of code used to simulate the system, and the plot of the trajectory of θ_1 and θ_2. Also, attach a figure showing how you define the frames.

handwritten figure of frame definitions attached at bottom of document

In [12]:
```
from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='https://github.com/MuchenSun/ME314pngs/raw/master/doubpend_frames.jpg' width=500' height='350'></table>"))
```



In [13]:

```
import sympy as sympy
from sympy import symbols, Function, Eq, simplify, solve, Matrix, sin, cos, pi

t, g, m1, m2, l1, l2 = symbols(r't, g, m1, m2, l1, l2')

th1 = Function(r'theta_1')(t)
th2 = Function(r'theta_2')(t)
q = Matrix([th1, th2])
qdot = q.diff(t)
qddot = qdot.diff(t)

p1x = l1 * sin(q[0])
p1y = l1 * -cos(q[0])
p2x = p1x + l2 * sin(q[0] + q[1])
p2y = p1y + l2 * -cos(q[0] + q[1])

p1xdot = p1x.diff(t)
p1ydot = p1y.diff(t)
p2xdot = p2x.diff(t)
p2ydot = p2y.diff(t)

#compute the quaternions
#rotation from world to frame A
gwa = Matrix([
    [cos(-pi/2), -sin(-pi/2), 0],
    [sin(-pi/2), cos(-pi/2), 0],
    [0, 0, 1]
])

#translation from A to B
gab = Matrix([
    [1, 0, l1],
    [0, 1, 0],
    [0, 0, 1]
])

#rotation from B to C
gbc = Matrix([
    [cos(th1), -sin(th1), 0],
    [sin(th1), cos(th1), 0],
    [0, 0, 1]
])

#translation from C to D
gcd = Matrix([
    [1, 0, l2],
    [0, 1, 0],
    [0, 0, 1]
])

#E frame is not shown in given figure but this is the rotation of frame D to account for theta2
gde = Matrix([
    [cos(th2), -sin(th2), 0],
    [sin(th2), cos(th2), 0],
    [0, 0, 1]
])

r1 = Matrix([0, 0, 1])
r2 = Matrix([0, 0, 1])

r1w = gwa*gbc*gab*r1
r2w = gwa*gbc*gab*gde*gcd*r2

#need velocity terms to compute kinetic energies of m1 and m2
d1w = r1w.diff(t)
d2w = r2w.diff(t)

#compute KE
```

```
ke2 = ( m2 * (dzw.T) * dzw ) / 2
ke_total = ke1 + ke2

#compute potential energies
v1 = m1 * g * r1w[1]
v2 = m2 * g * r2w[1]
v_total = v1 + v2

#compute Lagrangian
L = ke_total[0] - v_total
L = L.subs({m1:1, m2:1, l1:1, l2:1, g:9.8})
L = Matrix([L])


dldq = L.jacobian(q)
dlddq = L.jacobian(qdot)
dldqdot = dlddq.diff(t)

#compute Euler-Lagrange equations
el = (dldq - dldqdot).T
el = Eq(simplify(el), Matrix([0, 0]))
```

In [14]:
```
el_soln = solve(el, qddot, dict=True)
```

In [15]:
```
# dq = 'dq'
# ddq = 'ddq'
# abc = Matrix([dq, ddq])
# display(qddot, abc)
```

output of Euler-Lagrange equations re-copied at bottom of document

In [16]:
```
print('================================================')
print()
print("Euler Lagrange equations for double pendulum system")
display(el)
print()
print()

#display the equations of motion
for sol in el_soln:
    for eqn in qddot:
        solution = Eq(simplify(eqn), sol[eqn])
        display(simplify(solution))
```

```
================================================

Euler Lagrange equations for double pendulum system
```

$$\begin{bmatrix} -9.8\sin\left(\theta_1(t)+\theta_2(t)\right) - 19.6\sin\left(\theta_1(t)\right) + 2.0\sin\left(\theta_2(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.0\sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 - 2.0\cos\left(\theta_2(t)\right)\frac{d^2}{dt^2}\theta_1(t) - 1.0\cos\left(\theta_2(t)\right)\frac{d^2}{dt^2}\theta_2(t) - 3.0\frac{d^2}{dt^2}\theta_1(t) - 1.0\frac{d^2}{dt^2}\theta_2(t) \\ -9.8\sin\left(\theta_1(t)+\theta_2(t)\right) - 1.0\sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 1.0\cos\left(\theta_2(t)\right)\frac{d^2}{dt^2}\theta_1(t) - 1.0\frac{d^2}{dt^2}\theta_1(t) - 1.0\frac{d^2}{dt^2}\theta_2(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\frac{d^2}{dt^2}\theta_1(t) = \frac{9.8\sin\left(\theta_1(t)+2\theta_2(t)\right) - 29.4\sin\left(\theta_1(t)\right) + 2.0\sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 4.0\sin\left(\theta_2(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 2.0\sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 1.0\sin\left(2\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2}{3-\cos\left(2\theta_2(t)\right)}$$

$$19.6 \sin\left(\theta_1(t) - \theta_2(t)\right) - 19.6 \sin\left(\theta_1(t) + \theta_2(t)\right) - 9.8 \sin\left(\theta_1(t) + 2\theta_2(t)\right) + 29.4 \sin\left(\theta_1(t)\right) - 6.0 \sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 4.0 \sin\left(\theta_2(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) - 2.0 \sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 - 2.0 s$$

In [ ]:

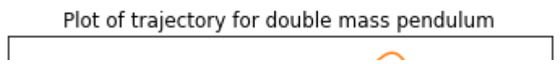
In [17]:
```
from sympy import lambdify

#solving numerically
soln_num1 = lambdify([q[0], q[1], qdot[0], qdot[1]], sol[qddot[0]])
soln_num2 = lambdify([q[0], q[1], qdot[0], qdot[1]], sol[qddot[1]])
```

In [18]:

```
#setup simulation functions
#copied over from previous homework 5

import numpy as np
import math

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
        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):
```

```
        x = np.copy(xtraj[:,i])
    return xtraj


def dyn(s):
    """
    System dynamics function (extended)

    Parameters
    ============
    s: NumPy array
        s = [theta1, theta2, theta3, theta1dot, theta2dot, theta3dot]

    Return
    ============
    sdot: NumPy array
        time derivative of input state vector,
        sdot = [theta1dot, theta2dot, theta3dot, theta1_ddot, theta2_ddot, theta3_ddot]
    """
    #simulate for 2DOF pendulum system
    return np.array([s[2], s[3], soln_num1(s[0], s[1], s[2], s[3]), soln_num2(s[0], s[1], s[2], s[3])])


#given initial condition
dt = 0.01
s0 = np.array([-math.pi/2, -math.pi/2, 0, 0])
traj = simulate(dyn, s0, [0,5], 0.01, integrate)

print('shape of trajectory array: ', traj.shape)

#begin trajectory plotting setup
import matplotlib.pyplot as plt
num = int(5/dt)
fig1 = plt.figure(0)
x_axis = np.linspace(0, 5, num)

plt.plot(x_axis, traj[0])
plt.plot(x_axis, traj[1])



#build legend for plot
legend_plt = [str(label) for label in q]
fig1.legend(legend_plt, bbox_to_anchor=(1.1, 0.5))

#axes labels and chart title
plt.title('Plot of trajectory for double mass pendulum')
plt.xlabel('Time (s)')
plt.ylabel('Angle (rads)')

plt.show()
```

```
shape of trajectory array:  (4, 500)
```



trajectory plot re-copied at bottom of document

Problem 5 (20pts)

Modify the previous animation function for the double-pendulum, such that the animation could show the frames you defined in the last problem (it's similar to the "tf" in RViz, if you're familiar with ROS). All the x axes should be displayed in green and all the y axes should be displayed in red, all axes have the length of 0.3. An example can be found at https://youtu.be/2H3KvRWQqys. Do not use functions provided in the modern robotics package for manipulating transformation matrices such as RpToTrans().

Hint 1: Each axis can be considered as a line connecting the origin and the point $[0.3, 0]$ *or* $[0, 0.3]$ *in that frame. You will need to use the homogeneous transformations to transfer these two axes/points back into the world/fixed frame. Example code showing how to display one frame is provided below.*

Turn in: A copy of code used for animation and a video of the animation. The video can be uploaded separately through Canvas, and it should be in ".mp4" format. You can use screen capture or record the screen directly with your phone.

In [22]:
```
from math import cos, sin, pi
```

In [34]:

```
def animate_double_pend(theta_array,L1=1,L2=1,T=5):
    """
    Function to generate web-based animation of double-pendulum system

    Parameters:
    ================================================
    theta_array:
        trajectory of theta1 and theta2, should be a NumPy array with
        shape of (2,N)
    L1:
        length of the first pendulum
    L2:
        length of the second pendulum
    T:
        length/seconds of animation duration

    Returns: None
    """

    ################################
    # Imports required for animation.
    from plotly.offline import init_notebook_mode, iplot
    from IPython.display import display, HTML
    import plotly.graph_objects as go

    ######################
    # Browser configuration.
    def configure_plotly_browser_state():
        import IPython
        display(IPython.core.display.HTML('''
            <script src="/static/components/requirejs/require.js"></script>
            <script>
              requirejs.config({
                paths: {
                  base: '/static/base',
                  plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
                },
              });
            </script>
            '''))
    configure_plotly_browser_state()
    init_notebook_mode(connected=False)

    ################################################
    # Getting data from pendulum angle trajectories.
    xx1=L1*np.sin(theta_array[0])
    yy1=-L1*np.cos(theta_array[0])
    xx2=xx1+L2*np.sin(theta_array[0]+theta_array[1])
    yy2=yy1-L2*np.cos(theta_array[0]+theta_array[1])
    N = len(theta_array[0]) # Need this for specifying length of simulation

    ################################################
    # Define arrays containing data for frame axes
    # In each frame, the x and y axis are always fixed
    x_axis = np.array([0.3, 0.0])
    y_axis = np.array([0.0, 0.3])
    # Use homogeneous tranformation to transfer these two axes/points
    # back to the fixed frame
    frame_w_x_axis = np.zeros((2,N))
    frame_w_y_axis = np.zeros((2,N))

    frame_b_x_axis = np.zeros((2,N))
    frame_b_y_axis = np.zeros((2,N))

    frame_c_x_axis = np.zeros((2,N))
    frame_c_y_axis = np.zeros((2,N))

    frame_d_x_axis = np.zeros((2,N))
```

```
frame_e_x_axis = np.zeros((2,N))
frame_e_y_axis = np.zeros((2,N))

frame_f_x_axis = np.zeros((2,N))
frame_f_y_axis = np.zeros((2,N))


for i in range(N): # iteration through each time step
    # evaluate homogeneous transformation
    # each iteration will evaluate the 5 transformations
    t_wb = np.array([[cos(-pi/2), -sin(-pi/2), 0],
                     [sin(-pi/2),  cos(-pi/2), 0],
                     [                      0,                      0, 1]])

    t_bc = np.array([[cos(theta_array[0][i]), -sin(theta_array[0][i]), 0],
                     [sin(theta_array[0][i]),  cos(theta_array[0][i]), 0],
                     [                      0,                      0, 1]])

    t_cd = np.array([[1, 0, L1],
                     [0, 1, 0],
                     [                      0,                      0, 1]])

    t_de = np.array([[np.cos(theta_array[1][i]), -np.sin(theta_array[1][i]), 0],
                     [np.sin(theta_array[1][i]),  np.cos(theta_array[1][i]), 0],
                     [                      0,                      0, 1]])

    t_ef = np.array([[1, 0, L2],
                     [0, 1, 0],
                     [                      0,                      0, 1]])

    #do the transformations
    t_wc = t_wb.dot(t_bc)
    t_wd = t_wc.dot(t_cd)
    t_we = t_wd.dot(t_de)
    t_wf = t_we.dot(t_ef)


    # transfer the x and y axes in body frame back to fixed frame at
    # the current time step
    frame_b_x_axis[:,i] = t_wb.dot([x_axis[0], x_axis[1], 1])[0:2]
    frame_b_y_axis[:,i] = t_wb.dot([y_axis[0], y_axis[1], 1])[0:2]

    frame_c_x_axis[:,i] = t_wc.dot([x_axis[0], x_axis[1], 1])[0:2]
    frame_c_y_axis[:,i] = t_wc.dot([y_axis[0], y_axis[1], 1])[0:2]

    frame_d_x_axis[:,i] = t_wd.dot([x_axis[0], x_axis[1], 1])[0:2]
    frame_d_y_axis[:,i] = t_wd.dot([y_axis[0], y_axis[1], 1])[0:2]

    frame_e_x_axis[:,i] = t_we.dot([x_axis[0], x_axis[1], 1])[0:2]
    frame_e_y_axis[:,i] = t_we.dot([y_axis[0], y_axis[1], 1])[0:2]

    frame_f_x_axis[:,i] = t_wf.dot([x_axis[0], x_axis[1], 1])[0:2]
    frame_f_y_axis[:,i] = t_wf.dot([y_axis[0], y_axis[1], 1])[0:2]

##################################
# Using these to specify axis limits.
xm = -3 #np.min(xx1)-0.5
xM = 3 #np.max(xx1)+0.5
ym = -3 #np.min(yy1)-2.5
yM = 3 #np.max(yy1)+1.5

############################
# Defining data dictionary.
# Trajectories are here.
data=[
    # note that except for the trajectory (which you don't need this time),
    # you don't need to define entries other than "name". The items defined
    # in this list will be related to the items defined in the "frames" list
```

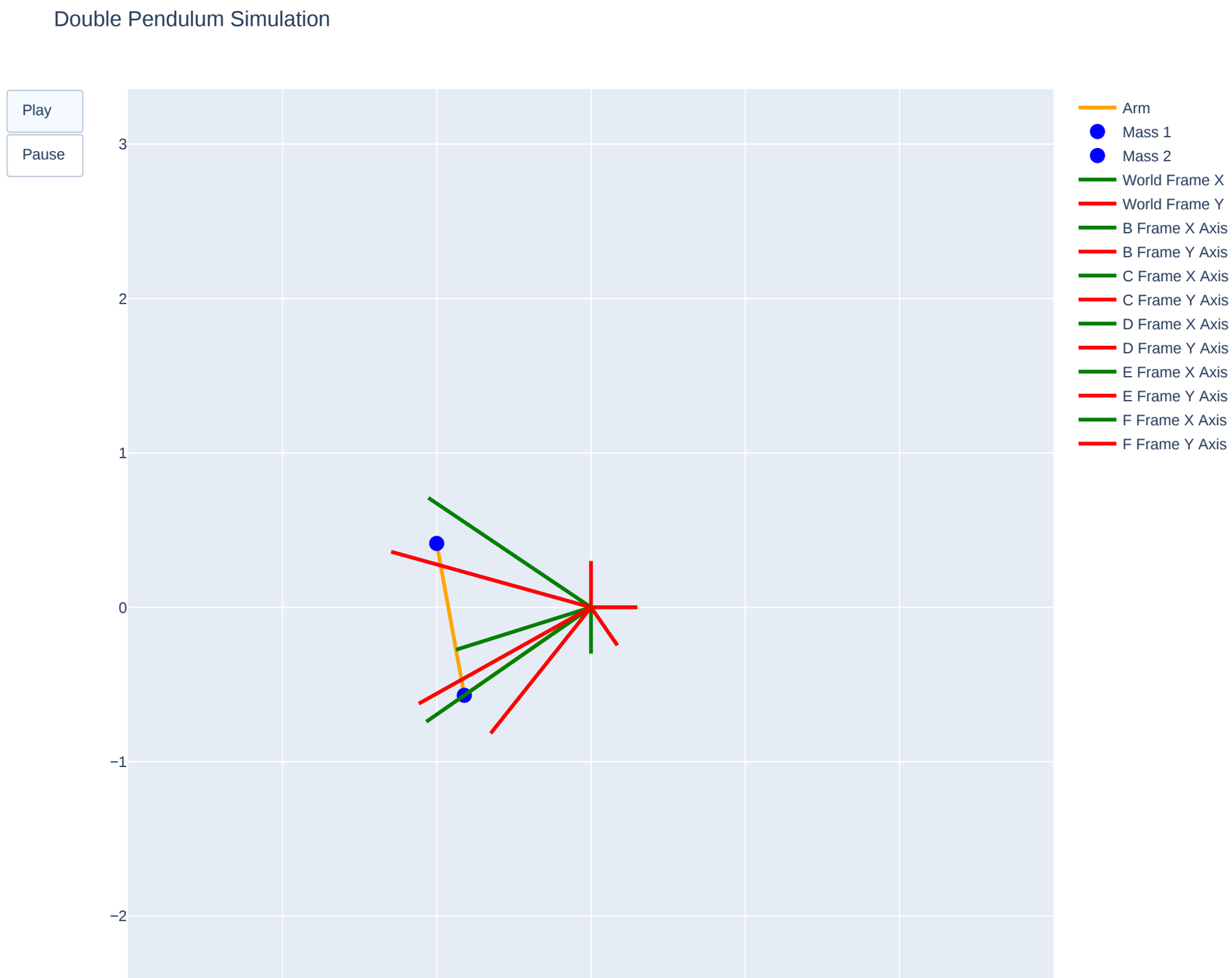
```
        dict(name='Arm'),
        dict(name='Mass 1'),
        dict(name='Mass 2'),
        dict(name='World Frame X'),
        dict(name='World Frame Y'),


        dict(name='B Frame X Axis'),
        dict(name='B Frame Y Axis'),
        dict(name='C Frame X Axis'),
        dict(name='C Frame Y Axis'),
        dict(name='D Frame X Axis'),
        dict(name='D Frame Y Axis'),
        dict(name='E Frame X Axis'),
        dict(name='E Frame Y Axis'),
        dict(name='F Frame X Axis'),
        dict(name='F Frame Y Axis'),

        # You don't need to show trajectory this time,
        # but if you want to show the whole trajectory in the animation (like what
        # you did in previous homeworks), you will need to define entries other than
        # "name", such as "x", "y". and "mode".

        # dict(x=xx1, y=yy1,
        #      mode='markers', name='Pendulum 1 Traj',
        #      marker=dict(color="fuchsia", size=2)
        #     ),
        # dict(x=xx2, y=yy2,
        #      mode='markers', name='Pendulum 2 Traj',
        #      marker=dict(color="purple", size=2)
        #     ),
        ]

    ################################
    # Preparing simulation layout.
    # Title and axis ranges are here.
    layout=dict(autosize=False, width=1000, height=1000,
                xaxis=dict(range=[xm, xM], autorange=False, zeroline=False,dtick=1),
                yaxis=dict(range=[ym, yM], autorange=False, zeroline=False,scaleanchor = "x",dtick=1),
                title='Double Pendulum Simulation',
                hovermode='closest',
                updatemenus= [{'type': 'buttons',
                               'buttons': [{'label': 'Play','method': 'animate',
                                            'args': [None, {'frame': {'duration': T, 'redraw': False}}]},
                                           {'args': [[None], {'frame': {'duration': T, 'redraw': False}, 'mode': 'immediate',
                                            'transition': {'duration': 0}}],'label': 'Pause','method': 'animate'}
                                          ]
                            }]
               )

    ######################################
    # Defining the frames of the simulation.
    # This is what draws the lines from
    # joint to joint of the pendulum.
    frames=[dict(data=[# first three objects correspond to the arms and two masses,
                       # same order as in the "data" variable defined above (thus
                       # they will be labeled in the same order)
                       dict(x=[0,xx1[k],xx2[k]],
                            y=[0,yy1[k],yy2[k]],
                            mode='lines',
                            line=dict(color='orange', width=3),
                            ),
                       go.Scatter(
                            x=[xx1[k]],
                            y=[yy1[k]],
                            mode="markers",
                            marker=dict(color="blue", size=12)),
```

```
                marker=dict(color="blue", size=12)),
            # display x and y axes of the fixed frame in each animation frame
            dict(x=[0,x_axis[0]],
                y=[0,x_axis[1]],
                mode='lines',
                line=dict(color='green', width=3),
                ),
            dict(x=[0,y_axis[0]],
                y=[0,y_axis[1]],
                mode='lines',
                line=dict(color='red', width=3),
                ),
            # display x and y axes of the {B} frame in each animation frame
            dict(x=[0, frame_b_x_axis[0][k]],
                y=[0, frame_b_x_axis[1][k]],
                mode='lines',
                line=dict(color='green', width=3),
                ),
            dict(x=[0, frame_b_y_axis[0][k]],
                y=[0, frame_b_y_axis[1][k]],
                mode='lines',
                line=dict(color='red', width=3),
                ),

            # display x and y axes of the {A} frame in each animation frame
            dict(x=[0, frame_c_x_axis[0][k]],
                y=[0, frame_c_x_axis[1][k]],
                mode='lines',
                line=dict(color='green', width=3),
                ),
            dict(x=[0, frame_c_y_axis[0][k]],
                y=[0, frame_c_y_axis[1][k]],
                mode='lines',
                line=dict(color='red', width=3),
                ),

            # display x and y axes of the {A} frame in each animation frame
            dict(x=[0, frame_d_x_axis[0][k]],
                y=[0, frame_d_x_axis[1][k]],
                mode='lines',
                line=dict(color='green', width=3),
                ),
            dict(x=[0, frame_d_y_axis[0][k]],
                y=[0, frame_d_y_axis[1][k]],
                mode='lines',
                line=dict(color='red', width=3),
                ),

            # display x and y axes of the {A} frame in each animation frame
            dict(x=[0, frame_e_x_axis[0][k]],
                y=[0, frame_e_x_axis[1][k]],
                mode='lines',
                line=dict(color='green', width=3),
                ),
            dict(x=[0, frame_e_y_axis[0][k]],
                y=[0, frame_e_y_axis[1][k]],
                mode='lines',
                line=dict(color='red', width=3),
                ),

            # display x and y axes of the {A} frame in each animation frame
            dict(x=[0, frame_f_x_axis[0][k]],
                y=[0, frame_f_x_axis[1][k]],
                mode='lines',
                line=dict(color='green', width=3),
```

```
                        ),

                    ]) for k in range(N)]

    ######################################
    # Putting it all together and plotting.
    figure1=dict(data=data, layout=layout, frames=frames)
    iplot(figure1)
```

In [35]:

```
animate_double_pend(traj[0:2],L1=1,L2=1,T=5)
```



In [ ]: