ME314 Homework 6

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, .mp4 videos of animations when requested 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) for handwritten or markdown questions and include simplified code outputs (e.g. .simplify()) for python questions.
- 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.

In [1]:
```python
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```python
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):
        xtraj[:,i]=integrate(f,x,dt)
        x = np.copy(xtraj[:,i])
    return xtraj
```

```
<>:2: SyntaxWarning: invalid escape sequence '\d'
<>:31: SyntaxWarning: invalid escape sequence '\d'
<>:2: SyntaxWarning: invalid escape sequence '\d'
<>:31: SyntaxWarning: invalid escape sequence '\d'
/tmp/ipykernel_330648/3955567788.py:2: SyntaxWarning: invalid escape sequence '\d'
  """
/tmp/ipykernel_330648/3955567788.py:31: SyntaxWarning: invalid escape sequence '\d'
  """
```

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 handwritten solution. You can also use *LaTeX*. If you use SymPy, you need to include a copy of your code and the code outputs. Make sure to note why your handwritten solution / code output explains the results.

Solution 1

To show that $R(\theta_1)R(\theta_2) \in SO(n)$ we must show that the expressions below are true:

$(R(\theta_1)R(\theta_2))^T(R(\theta_1)R(\theta_2)) = I_{n\times n} \ : Eq\,1$

$det(R(\theta_1)R(\theta_2)) = 1 \ : Eq\,2$

Now that we have established this, let us take a look at the following property of transposes:

$(AB)^T = B^T A^T$

If we use this equality to show the orthagonality $Eq\,1$ we get:

$(R(\theta_1)R(\theta_2))^T = R(\theta_2)^T R(\theta_1)^T$

We can use the equality above to further simplify the left handside of $Eq\,1$:

$(R(\theta_1)R(\theta_2))^T(R(\theta_1)R(\theta_2)) = R(\theta_2)^T R(\theta_1)^T(R(\theta_1)R(\theta_2))$

The right handside of the equation can be simplified to:

$R(\theta_2)^T I R(\theta_2)) = R(\theta_2)^T R(\theta_2)) = I$

This shows that $Eq\,1$ is true.

Now let us show that the $Eq\,2$ holds.

We know that the determinant of a matrix product has this property:

$det(AB) = det(A)det(B)$

Since we know that $R(\theta_1) \in SO(3)$ and $R(\theta_2) \in SO(3)$.

We can say that $det(R(\theta_1)R(\theta_2)) = 1 \cdot 1 = 1$

This shows that $Eq\,2$ is correct.

We have proven that $Eq\,1$ and $Eq\,2$ are correct, which proves that $R(\theta_1)R(\theta_2) \in SO(n)$.

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, you need to include a copy of your code and the code outputs. Make sure to note why your handwritten soultion / code output explains the results.

Solution 2

To show that $g(x_1, y_1, \theta_1)g(x_2, y_2, \theta_2) \in SE(2)$, we must demonstrate that the expressions below are true:

$g(x_1, y_1, \theta_1)g(x_2, y_2, \theta_2) =
\begin{bmatrix}
cos(\theta_1 + \theta_2) & -sin(\theta_1 + \theta_2) & x' \
sin(\theta_1 + \theta_2) & cos(\theta_1 + \theta_2) & y' \
0 & 0 & 1
\end{bmatrix}
    Eq : 1$

$$det(g(x_1, y_1, \theta_1)g(x_2, y_2, \theta_2)) = 1 : Eq\,2$$

In $Eq\,1$

$$x' = x_1 + cos(\theta_1)x_2 - sin(\theta_1)y_2$$

$$y' = y_1 + sin(\theta_1)x_2 + cos(\theta_1)y_2$$

Now that the requirements have been defined, we can confirm that $g(x_1, y_1, \theta_1)g(x_2, y_2, \theta_2) \in SE(2)$

Let's compute the product $g(x_1, y_1, \theta_1)g(x_2, y_2, \theta_2)$

$$g(x_1, y_1, \theta_1)g(x_2, y_2, \theta_2) = \begin{bmatrix} cos(\theta_1) & -sin(\theta_1) & x_1 \\ sin(\theta_1) & cos(\theta_1) & y_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta_2) & -sin(\theta_2) & x_2 \\ sin(\theta_2) & cos(\theta_2) & y_2 \\ 0 & 0 & 1 \end{bmatrix}$$

The multiplication yields

$$= \begin{bmatrix} cos(\theta_1)cos(\theta_2) - sin(\theta_1)sin(\theta_2) & -cos(\theta_1)sin(\theta_1) - sin(\theta_1)cos(\theta_2) & cos(\theta_1)x_2 - sin(\theta_1)y_2 + \\ sin(\theta_1)cos(\theta_2) + cos(\theta_1)cos(\theta_2) & -sin(\theta_1)sin(\theta_2) + cos(\theta_1)cos(\theta_2) & sin(\theta_1)x_2 + cos(\theta_1)y_2 + \\ 0 & 0 & 1 \end{bmatrix}$$

We can use the trig identities below:

$$cos(\theta_1)cos(\theta_2) - sin(\theta_1)sin(\theta_2) = cos(\theta_1 + \theta_2)$$

$$cos(\theta_1)cos(\theta_2) + sin(\theta_1)sin(\theta_2) = sin(\theta_1 + \theta_2)$$

This will allow to rewrite the matrix:

$$= \begin{bmatrix} cos(\theta_1 + \theta_2) & -sin(\theta_1 + \theta_2) & cos(\theta_1)x_2 - sin(\theta_1)y_2 + x_1 \\ sin(\theta_1 + \theta_2) & cos(\theta_1 + \theta_2) & sin(\theta_1)x_2 + cos(\theta_1)y_2 + y_1 \\ 0 & 0 & 1 \end{bmatrix}$$

Let's show that the rotational part is in $SO(2)$.

$$\begin{bmatrix} cos(\theta_1+\theta_2) & -sin(\theta_1+\theta_2) \\ sin(\theta_1+\theta_2) & cos(\theta_1+\theta_2) \end{bmatrix}$$

The transpose of this matrix is equal to its inverse and the determinant is 1.

The determinant of the full matrix is equal to 1.

These two show that the full matrix $g(x_1, y_1, \theta_1)g(x_2, y_2, \theta_2) \in SE(2)$

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, you need to include a copy of your code and the code outputs. Make sure to note why your handwritten soultion / code output explains the results.

Solution 3

Let us define p and $\hat{p}$ (p hat)

$$p = \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$

$$\hat{p} = \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}$$

The g matrix is defined as:

$$g = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

Where R is the rotation component and t is the translation component.

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}, \quad t = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Applying the Transformation to $\hat{p}$

$$\hat{p} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}$$

Multiplying the matrices, we get:

$$\hat{p} = \begin{bmatrix} \cos(\theta)p_x - \sin(\theta)p_y + t_x \\ \sin(\theta)p_x + \cos(\theta)p_y + t_y \\ 1 \end{bmatrix}$$

This shows that the transformation g applies a rotation followed by a translation to the point p.

Separating Rotation and Translation

To separate the transformation into its rotation and translation components, we can express g as a product of a pure rotation matrix and a pure translation matrix.

Define: $g_R = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix}, \quad g_T = \begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix}$

where I is the 2×2 identity matrix.

Thus, we can express g as:

$$g = g_T g_R = \begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix}$$

Order of Operations

If we first apply translation g_T and then rotation g_R:

$$g_T g_R \hat{p} = \begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \hat{p} = \begin{bmatrix} \cos(\theta)p_x - \sin(\theta)p_y + t_x \\ \sin(\theta)p_x + \cos(\theta)p_y + t_y \\ 1 \end{bmatrix}$$

If we reverse the order and apply rotation g_R first, followed by translation g_T:

$$g_R g_T \hat{p} = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix} \hat{p} = \begin{bmatrix} R & Rt \\ 0 & 1 \end{bmatrix} \hat{p}$$

$$= \begin{bmatrix} \cos(\theta)(p_x + t_x) - \sin(\theta)(p_y + t_y) \\ \sin(\theta)(p_x + t_x) + \cos(\theta)(p_y + t_y) \\ 1 \end{bmatrix}$$

As we can see, the two operations do not commute:

- **Translation followed by rotation** $(g_T g_R)$: The translation is applied first, and the point is then rotated about the origin.
- **Rotation followed by translation** $(g_R g_T)$: The point is first rotated, and then the rotated point is translated.

The order of applying rotation and translation affects the final position of the point in space. In summary, a homogeneous transformation in $SE(2)$ can be separated into a rotation and a translation, but the order matters. A homogenuous tranlsation can be done in a translation and then rotation and it is true because we showed that $g_T g_R = \hat{p}$

Problem 4 (20pts)

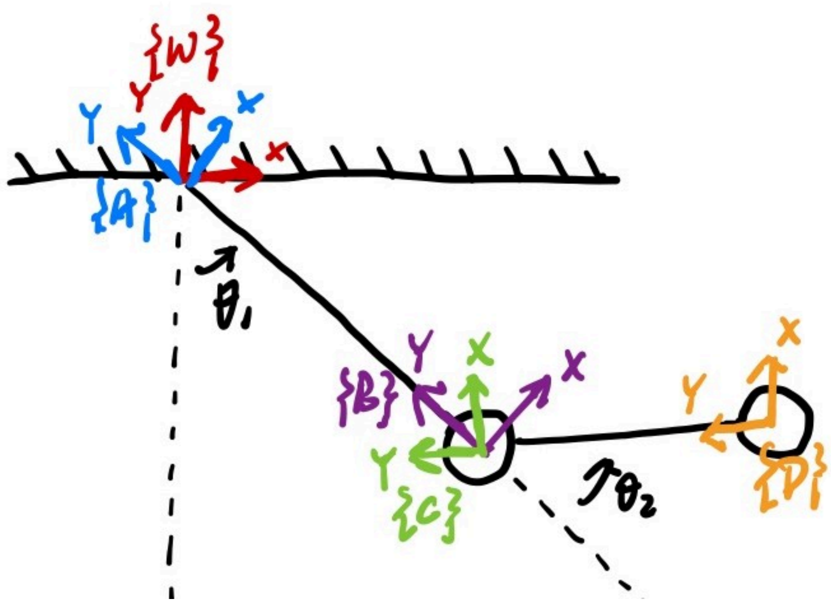
Simulate the same double-pendulum system in previous homework using only homogeneous transformation (and thus avoid using trigonometry). Simulate the system for $t \in [0, 3]$ with $dt = 0.01$.

The parameters are $m_1 = m_2 = 1, R_1 = R_2 = 1, g = 9.8$ with initial conditions $\theta_1 = \theta_2 = -\frac{\pi}{3}, \dot{\theta}_1 = \dot{\theta}_2 = 0$. Do not use functions provided in the modern robotics package for manipulating transformation matrices such as RpToTrans(), etc.

> 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: Include a copy of your code used to simulate the system, and clearly labeled plot of θ_1 and θ_2 trajectory. Also, attach a figure showing how you defined the frames.

In [3]:
```
from IPython.core.display import HTML, Markdown
display(HTML("<table><tr><td><img src='https://github.com/MuchenSun/ME314pngs/raw/master
```



In [4]:
```
g, t, m1, m2, R1, R2 = sym.symbols(r'g t m_1 m_2 R_1 R_2')
theta1 = sym.Function(r'theta_1')(t)
theta2 = sym.Function(r'theta_2')(t)

# define the configuration
q = sym.Matrix([theta1, theta2])
q_d = q.diff(t)
q_dd = q_d.diff(t)

# now define the different transformations
g_wa, g_ab, g_bc, g_cd = sym.symbols(r'g_{WA} g_{AB} g_{BC} g_{CD}')

# 1. Define g_wa (Rotation by theta1)
g_wa = sym.Matrix([
    [sym.cos(theta1), -sym.sin(theta1), 0],
    [sym.sin(theta1),  sym.cos(theta1), 0],
    [0,                0,               1]
])

# 2. Define g_ab (Translation by -R1 in y-direction)
g_ab = sym.Matrix([
    [1, 0, 0],
    [0, 1, -R1],
    [0, 0, 1]
])
```

```
# 3. Define g_bc (Rotation by theta2)
g_bc = sym.Matrix([
    [sym.cos(theta2), -sym.sin(theta2), 0],
    [sym.sin(theta2),  sym.cos(theta2), 0],
    [0,                0,               1]
])

# 4. Define g_cd (Translation by -R2 in y-direction)
g_cd = sym.Matrix([
    [1, 0, 0],
    [0, 1, -R2],
    [0, 0, 1]
])
```

In [5]:
```
# now let us get position and velocity of the two masses to compute the lagrangian
origin = sym.Matrix([0, 0, 1])
r_b = g_wa * g_ab * origin
r_d = g_wa * g_ab * g_bc * g_cd * origin

# velocity
v_b = r_b.diff(t)
v_d = r_d.diff(t)

KE = (1/2) * m1 * (v_b[0] ** 2 + v_b[1] ** 2) + (1/2) * m2 * (v_d[0] ** 2 + v_d[1] ** 2)
V = m1 * g * r_b[1] + m2 * g * r_d[1]
L = KE - V

display(Markdown("**Lagrangian is given by:**"))
Lagr = sym.symbols('L')
display(sym.Eq(Lagr, L.simplify().expand()))
```

Lagrangian is given by:

$$L = 0.5R_1^2 m_1 \left(\frac{d}{dt}\theta_1(t)\right)^2 + 0.5R_1^2 m_2 \left(\frac{d}{dt}\theta_1(t)\right)^2 + 1.0R_1R_2m_2\cos\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 1.0R_1R_2m_2\cos\left(\theta_2(t)\right)\frac{d}{dt}$$

In [6]:
```
# now let us solve for equations of motion
L = L.simplify()
dL_dq = L.diff(q)
dL_dq_dot = L.diff(q_d)
dt_dL_dq_dot = dL_dq_dot.diff(t)

EL = dL_dq - dt_dL_dq_dot

# now solve
soln = sym.solve(
    sym.Eq(
        sym.Matrix([EL[0], EL[1]]),
        sym.Matrix([0, 0])
    ),
    q_dd,
    dict=True
)


solns = []
for sol in soln:
  for config_var in q_dd:
    this_solution = sol[config_var].simplify().expand()
    solns.append(this_solution)
    display(sym.Eq(config_var, this_solution))
```

$$\frac{d^2}{dt^2}\theta_1(t) = \frac{0.5R_1m_2\sin(2\theta_2(t))\left(\frac{d}{dt}\theta_1(t)\right)^2}{R_1m_1 + R_1m_2\sin^2(\theta_2(t))} + \frac{1.0R_2m_2\sin(\theta_2(t))\left(\frac{d}{dt}\theta_1(t)\right)^2}{R_1m_1 + R_1m_2\sin^2(\theta_2(t))} + \frac{2.0R_2m_2\sin(\theta_2(t))\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t)}{R_1m_1 + R_1m_2\sin^2(\theta_2(t))}$$

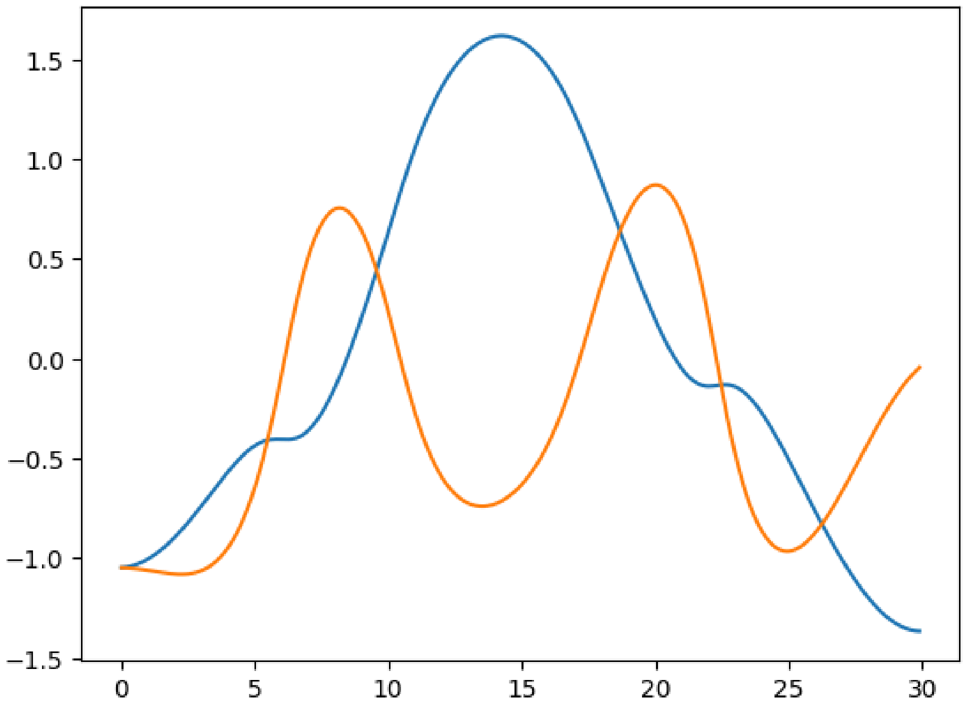
$$\frac{d^2}{dt^2}\theta_2(t) = -\frac{1.0R_1^2m_1\sin(\theta_2(t))\left(\frac{d}{dt}\theta_1(t)\right)^2}{R_1R_2m_1 + R_1R_2m_2\sin^2(\theta_2(t))} - \frac{1.0R_1^2m_2\sin(\theta_2(t))\left(\frac{d}{dt}\theta_1(t)\right)^2}{R_1R_2m_1 + R_1R_2m_2\sin^2(\theta_2(t))} - \frac{1.0R_1R_2m_2\sin(2\theta_2(t))\left(\frac{d}{dt}\theta_1(t\right.}{R_1R_2m_1 + R_1R_2m_2\sin^2(\theta_2(t}$$

$$-\frac{2.0R_2^2m_2\sin(\theta_2(t))\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t)}{R_1R_2m_1 + R_1R_2m_2\sin^2(\theta_2(t))} - \frac{1.0R_2^2m_2\sin(\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2}{R_1R_2m_1 + R_1R_2m_2\sin^2(\theta_2(t))} + \frac{1.0R_2gm_1\sin(\theta_1(t))}{R_1R_2m_1 + R_1R_2m_2\sin^2(\theta_2(t))} - \frac{0.5R}{R_1R_2}$$

In [7]:
```python
# substitutions
soln = sym.solve(EL, q_dd, dict=True)[0]
theta1_dd_sol = soln[theta1.diff(t, 2)].simplify()
theta2_dd_sol = soln[theta2.diff(t, 2)].simplify()

# Substitute numerical values
subs = {m1: 1, m2: 1, R1: 1, R2: 1, g: 9.8}

# Convert to lambda functions for simulation
f_theta1_dd = sym.lambdify([theta1, theta2, theta1.diff(t), theta2.diff(t)], theta1_dd_s
f_theta2_dd = sym.lambdify([theta1, theta2, theta1.diff(t), theta2.diff(t)], theta2_dd_s
```

In [8]:
```python
def double_pendulum_dynamics(y):
  return np.array(
    [
      y[2],
      y[3],
      f_theta1_dd(y[0],y[1],y[2],y[3]),
      f_theta2_dd(y[0],y[1],y[2],y[3])
    ]
  )

# Initial Conditions
x0 = np.array([-np.pi / 3, -np.pi / 3, 0.0, 0.0])

# Simulation Parameters
dt = 0.01
tspan = [0, 3]

# Run the simulation
traj = simulate(double_pendulum_dynamics, x0, tspan, dt, integrate)
plt.plot(np.arange(300) * 0.1, np.array([traj[0], traj[1]]).T)
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x718dfc0f7aa0>,
 <matplotlib.lines.Line2D at 0x718dfc0f7ad0>]
```



Problem 5 (20pts)

Modify the previous animation function for the double-pendulum such that the animation shows 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, with axis's length of 0.3 for all. An animation 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(), etc.

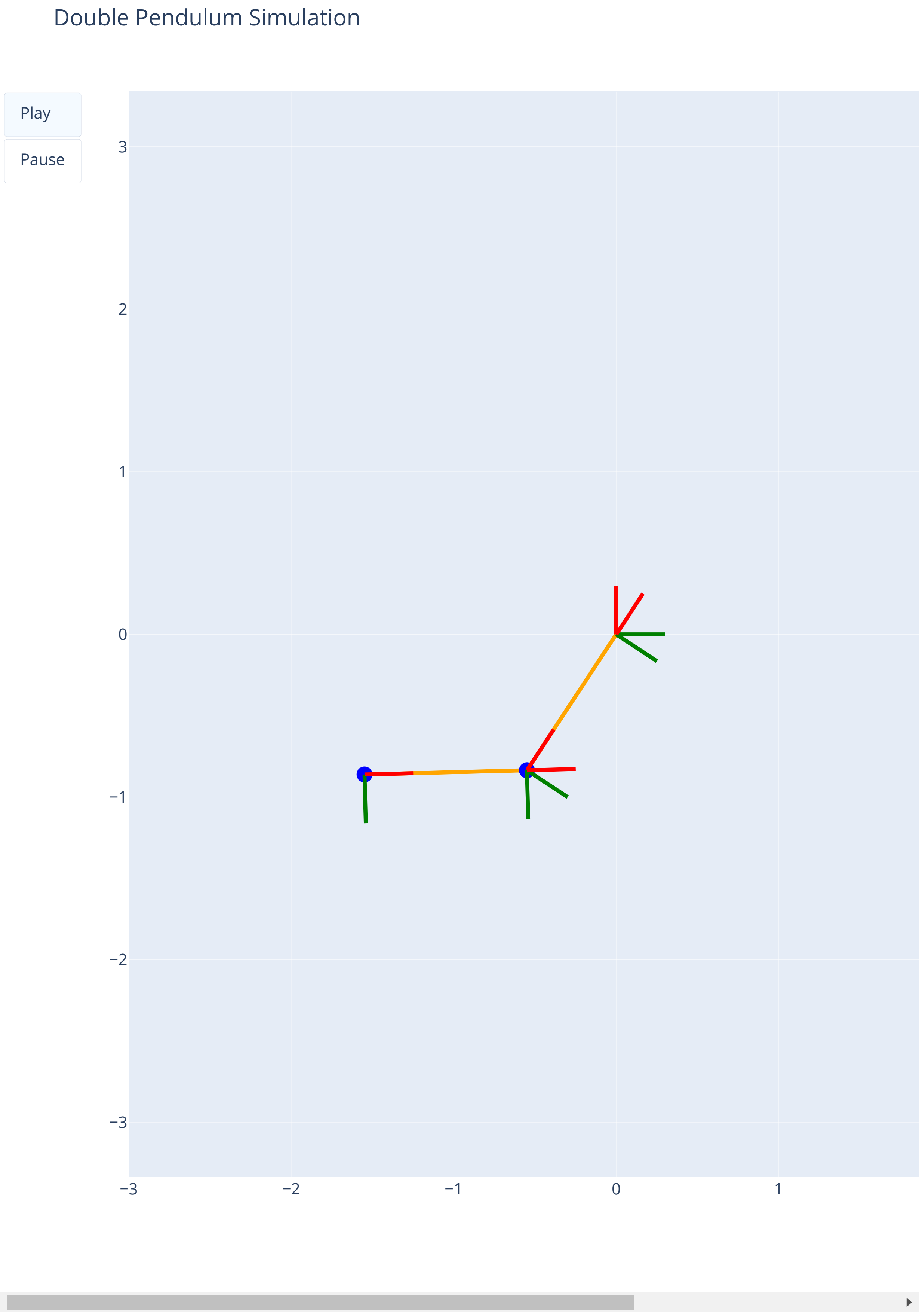
> Hint 1: Each axis can be considered as a line connecting the origin and the point $[0.3, 0]$ or $[0, 0.3]$ in that frame. You will need to use the homogeneous transformations to transfer these two axis/points back into the world/fixed frame. Example code showing how to display one frame is provided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

Turn in: Include a copy of your 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 either use screen capture or record the screen directly with your phone.

In [9]:
```
def animate_double_pend(theta_array,L1=1,L2=1,T=10):
    """
    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
    # add the origin
    origin = np.array([0.0, 0.0])
    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_a_x_axis = np.zeros((2,N))
    frame_a_y_axis = np.zeros((2,N))
    ##############
    ##############
    # Add the rest
    ##############
    ##############
    frame_a_origin = np.zeros((2, N))

    # add for b frame
    frame_b_x_axis = np.zeros((2,N))
    frame_b_y_axis = np.zeros((2,N))
    frame_b_origin = np.zeros((2,N))

    # add for c frame
    frame_c_x_axis = np.zeros((2,N))
    frame_c_y_axis = np.zeros((2,N))
    frame_c_origin = np.zeros((2,N))
    # add for d axis
    frame_d_x_axis = np.zeros((2,N))
    frame_d_y_axis = np.zeros((2,N))
    frame_d_origin = np.zeros((2,N))

    for i in range(N): # iteration through each time step
        # evaluate homogeneous transformation
        t_wa = np.array([
            [np.cos(theta_array[0][i]), -np.sin(theta_array[0][i]), 0],
            [np.sin(theta_array[0][i]),  np.cos(theta_array[0][i]), 0],
            [0, 0,  1]
        ])

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

        t_bc = 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_cd = np.array([
            [1, 0,  0],
            [0, 1,  -L2],
            [0, 0,  1]
        ])
        # transfer the x and y axes in body frame back to fixed frame at
        # the current time step
        frame_a_x_axis[:,i] = t_wa.dot([x_axis[0], x_axis[1], 1])[0:2]
        frame_a_y_axis[:,i] = t_wa.dot([y_axis[0], y_axis[1], 1])[0:2]
        # a is origin so we don't need a_origin_axis
        t_wb = t_wa @ t_ab
        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_b_origin[:,i] = t_wb.dot([origin[0], origin[1], 1])[0:2]

        t_wc = t_wb @ t_bc
        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_c_origin[:,i] = t_wc.dot([origin[0], origin[1], 1])[0:2]

        t_wd = t_wc @ t_cd
        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_d_origin[:,i] = t_wd.dot([origin[0], origin[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
        # later in the same order. Therefore, these entries can be considered as
        # labels for the components in each animation frame
        dict(name='Arm'),
        dict(name='Mass 1'),
        dict(name='Mass 2'),
        dict(name='World Frame X'),
        dict(name='World Frame Y'),
        dict(name='A Frame X Axis'),
        dict(name='A Frame Y Axis'),
        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'),
        # 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 =
                title='Double Pendulum Simulation',
                hovermode='closest',
                updatemenus= [{'type': 'buttons',
                               'buttons': [{'label': 'Play','method': 'animate',
                                            'args': [None, {'frame': {'duration': T, 're
                                           {'args': [[None], {'frame': {'duration': T, '
                                            'transition': {'duration': 0}}],'label': 'Pa
                                          ]
                              }]
               )

    ########################################
    # 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)),
                       go.Scatter(
                            x=[xx2[k]],
                            y=[yy2[k]],
                            mode="markers",
                            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 {A} frame in each animation frame
                       dict(x=[0, frame_a_x_axis[0][k]],
                            y=[0, frame_a_x_axis[1][k]],
                            mode='lines',
                            line=dict(color='green', width=3),
                            ),
                       dict(x=[0, frame_a_y_axis[0][k]],
                            y=[0, frame_a_y_axis[1][k]],
                            mode='lines',
                            line=dict(color='red', width=3),
                            ),
                       # display x and y axes of the {B} frame in each animation frame
                       dict(x=[frame_b_origin[0][k], frame_b_x_axis[0][k]],
                            y=[frame_b_origin[1][k], frame_b_x_axis[1][k]],
                            mode='lines',
                            line=dict(color='green', width=3),
                            ),
                       dict(x=[frame_b_origin[0][k], frame_b_y_axis[0][k]],
                            y=[frame_b_origin[1][k], frame_b_y_axis[1][k]],
                            mode='lines',
                            line=dict(color='red', width=3),
                            ),
                       # display x and y axes of the {C} frame in each animation frame
                       dict(x=[frame_c_origin[0][k], frame_c_x_axis[0][k]],
                            y=[frame_c_origin[1][k], frame_c_x_axis[1][k]],
                            mode='lines',
                            line=dict(color='green', width=3),
                            ),
                       dict(x=[frame_c_origin[0][k], frame_c_y_axis[0][k]],
                            y=[frame_c_origin[1][k], frame_c_y_axis[1][k]],
                            mode='lines',
                            line=dict(color='red', width=3),
                            ),
                       # display x and y axes of the {D} frame in each animation frame
                       dict(x=[frame_d_origin[0][k], frame_d_x_axis[0][k]],
                            y=[frame_d_origin[1][k], frame_d_x_axis[1][k]],
                            mode='lines',
                            line=dict(color='green', width=3),
                            ),
                       dict(x=[frame_d_origin[0][k], frame_d_y_axis[0][k]],
                            y=[frame_d_origin[1][k], frame_d_y_axis[1][k]],
                            mode='lines',
                            line=dict(color='red', width=3),
                            ),
                      ]) for k in range(N)]
```

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

In [10]:
```
animate_double_pend(traj[0:2],L1=1,L2=1,T=30)
```



```
In [ ]:
```