

PART_B_PENDULUM

December 11, 2023

```
[2]: from sympy import symbols, Matrix, pi, zeros, cos, sin

# Define symbolic variables
q1, q2, q3, q4 = symbols('q1 q2 q3 q4', real=True)
d1, d2, d3, d4 = symbols('d1 d2 d3 d4', real=True)
a1, a2 = symbols('a1 a2')
g = symbols('g')

# Initialize DH parameters
DH = [
    [a1, 0, 0, q1, 'R'],
    [a2, 0, 0, q2, 'R']
]

# Create length
# This accounts for the prismatic or revolute
# column not being included in the calculations
LENGTH = len(DH)

# Initialize transformation matrix
T = Matrix.eye(4)

# Initialize list for homogeneous transformations
Ti = [None] * LENGTH

# Define function to compute DH matrix
def compute_dh_matrix(a, alpha, d, theta, joint_type):
    if joint_type == "R":
        return Matrix([
            [cos(theta), -sin(theta)*cos(alpha), sin(theta)*sin(alpha),
             a*cos(theta)],
            [sin(theta), cos(theta)*cos(alpha), -cos(theta)*sin(alpha),
             a*sin(theta)],
            [0, sin(alpha), cos(alpha), d],
            [0, 0, 0, 1]
        ])
    else:
```

```

        return Matrix([
            [cos(alpha), -sin(alpha), 0, a*cos(alpha)],
            [sin(alpha), cos(alpha), 0, a*sin(alpha)],
            [0, 0, 1, theta],
            [0, 0, 0, 1]
        ])

# Compute homogeneous transformations
for i in range(LENGTH):
    temp = compute_dh_matrix(*DH[i])
    T = T * temp
    Ti[i] = T

# Display the resulting homogeneous transformations
for i, transform in enumerate(Ti):
    print(f'T{i + 1} =')
    display(transform)

```

T1 =

$$\begin{bmatrix} \cos(q_1) & -\sin(q_1) & 0 & a_1 \cos(q_1) \\ \sin(q_1) & \cos(q_1) & 0 & a_1 \sin(q_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T2 =

$$\begin{bmatrix} -\sin(q_1)\sin(q_2) + \cos(q_1)\cos(q_2) & -\sin(q_1)\cos(q_2) - \sin(q_2)\cos(q_1) & 0 & a_1\cos(q_1) - a_2\sin(q_1)\sin(q_2) + a_2\cos(q_1)\cos(q_2) \\ \sin(q_1)\cos(q_2) + \sin(q_2)\cos(q_1) & -\sin(q_1)\sin(q_2) + \cos(q_1)\cos(q_2) & 0 & a_1\sin(q_1) + a_2\sin(q_1)\cos(q_2) + a_2\cos(q_1)\sin(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```

[20]: # Define symbolic variables
Jw = [symbols(f'Jw{i+1}', real=True) for i in range(LENGTH)] # Angular velocity jacobian

# Angular velocity jacobian solution
Jw[0] = Matrix([[0], [0], [1]]).row_join(zeros(3, LENGTH-1))

# Calculate revolute joints
for i in range(1, LENGTH):
    jw = Matrix([[0], [0], [1]])

    for j in range(i):
        jw = jw.row_join(Ti[j][0:3, 2])

    jw = jw.row_join(zeros(3, LENGTH-1-i))

    Jw[i] = jw

```

```

# get indices of prismatic joints
pris_indices = []
for i in range(LENGTH):
    if DH[i][4] == 'P':
        pris_indices.append(i)
        print(f"Prismatic joint at index: {i}")

if len(pris_indices) > 0:
    # Calculate prismatic and update each matrix
    m = []
    prismatic_matrix = Matrix([[0], [0], [0]])
    for i, jw_matrix in enumerate(Jw):
        for j in range(len(pris_indices)):
            new_matrix = jw_matrix[:, :pris_indices[j]].
            row_join(prismatic_matrix).row_join(jw_matrix[:, pris_indices[j]+1:])
            m.append(new_matrix)

    # Update the matrix with the prismatic values
    Jw = m

# Display the resulting angular velocity jacobian
for i, jw_matrix in enumerate(Jw):
    print(f'Jw{i + 1} =')
    display(jw_matrix)
    print("-----")

```

Jw1 =

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

Jw2 =

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

```

[21]: from sympy import symbols, Matrix, diff

# Define symbolic variables
q = [symbols(f'q{x}', real=True) for x in range(1, LENGTH+1)]
c = [[symbols(f'c{x}x c{x}y c{x}z', real=True)] for x in range(1, LENGTH+1)]

# Initialize linear velocity jacobian

```

```

Jv = [None] * LENGTH

# Linear velocity jacobian solution
# Dependent on joint type
if DH[0][4] == 'R':
    P = Matrix.eye(4)
else:
    P = Matrix.zeros(4)

for i in range(LENGTH):
    c_list = list(*c[i]) # Convert from tuple to list to unpack
    P = Ti[i] * Matrix([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [*c_list,
↪1]]).T

    x = P[0, 3]
    y = P[1, 3]
    z = P[2, 3]

    for j in range(LENGTH):
        Jv[i] = Jv[i].row_join(Matrix([[x.diff(q[j])], [y.diff(q[j])], [z.
↪diff(q[j])]])) if Jv[i] else \
            Matrix([[x.diff(q[j])], [y.diff(q[j])], [z.diff(q[j])]]))

# Display the resulting linear velocity jacobian
for i, jv_matrix in enumerate(Jv):
    print(f'Jv{i + 1} =')
    display(jv_matrix)
    print("-----")

```

Jv1 =

$$\begin{bmatrix} -a_1 \sin(q_1) - c1x \sin(q_1) - c1y \cos(q_1) & 0 \\ a_1 \cos(q_1) + c1x \cos(q_1) - c1y \sin(q_1) & 0 \\ 0 & 0 \end{bmatrix}$$

Jv2 =

$$\begin{bmatrix} -a_1 \sin(q_1) - a_2 \sin(q_1) \cos(q_2) - a_2 \sin(q_2) \cos(q_1) + c2x (-\sin(q_1) \cos(q_2) - \sin(q_2) \cos(q_1)) + c2y (\sin(q_1) \sin(q_2) + \cos(q_1) \cos(q_2)) \\ a_1 \cos(q_1) - a_2 \sin(q_1) \sin(q_2) + a_2 \cos(q_1) \cos(q_2) + c2x (-\sin(q_1) \sin(q_2) + \cos(q_1) \cos(q_2)) + c2y (-\sin(q_1) \cos(q_2) - \sin(q_2) \cos(q_1)) \\ 0 \end{bmatrix}$$

```

[22]: # Define symbolic variables
m = [symbols(f'm{x}', real=True) for x in range(1, LENGTH+1)]

# Potential energy solution
P = Matrix.eye(4)

```

```
PE = 0
```

```
[23]: from sympy import symbols, Matrix, eye, Symbol, Function, symarray

def inertia_tensor(num):
    n = str(num)
    symbols_list = [f'Ixx{n}', f'Ixy{n}', f'Ixz{n}',
                    f'Iyx{n}', f'Iyy{n}', f'Iyz{n}',
                    f'Izx{n}', f'Izy{n}', f'Izz{n}']

    tensor = symarray('', len(symbols_list)).reshape(3, 3)

    for i in range(3):
        for j in range(3):
            tensor[i, j] = symbols_list[i * 3 + j]

    display(tensor)
    return tensor

# Define symbolic variables
qd = [symbols(f'qd{x}', real=True) for x in range(1, LENGTH+1)] # joint
    ↪ velocities
g = Symbol('g', real=True) # gravitational acceleration

# Inertia tensor for each link relative to the inertial frame stored in an nx1
    ↪ list
I = [inertia_tensor(i) for i in range(1, LENGTH + 1)]

array([[ 'Ixx1', 'Ixy1', 'Ixz1'],
       [ 'Iyx1', 'Iyy1', 'Iyz1'],
       [ 'Izx1', 'Izy1', 'Izz1']], dtype=object)

array([[ 'Ixx2', 'Ixy2', 'Ixz2'],
       [ 'Iyx2', 'Iyy2', 'Iyz2'],
       [ 'Izx2', 'Izy2', 'Izz2']], dtype=object)
```

```
[24]: # D = Inertia matrix solution & P = Potential Energy
D = None
PE = 0

# Calculate D and PE
for i in range(LENGTH):
    # Term one
    term_1 = (m[i] * Jv[i].T * Jv[i])

    # Term 2
    term_2 = Jw[i].T * I[i] * Jw[i]
```

```

    if i < 1:
        D = term_1 + term_2
    else:
        D = D + term_1 + term_2

    c_list = list(*c[i]) # Convert from tuple to list to unpack
    P = Ti[i] * Matrix([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [*c_list,
↪1]]).T
    PE += m[i] * g * P[2, 3]

# KE = Kinetic energy solution
q_dot_matrix = Matrix(qd)
KE = 0.5 * q_dot_matrix.T * D * q_dot_matrix

# Display the resulting kinetic energy expression
print("Kinetic Energy:")
display(KE)

```

Kinetic Energy:

$$[qd_1 \cdot (0.5qd_1 (Izz_1 + Izz_2 + m_1 (-a_1 \sin(q_1) - c1x \sin(q_1) - c1y \cos(q_1))^2 + m_1 (a_1 \cos(q_1) + c1x \cos(q_1) - c1y \sin(q_1))$$

```

[26]: from sympy import symbols, diff, zeros
import numpy as np

# Define symbolic variables
q = symbols('q:{}'.format(LENGTH), real=True)
qdd = symbols('qdd:{}'.format(LENGTH), real=True)

christoffel = []
for i in range(LENGTH):
    temp = Matrix.zeros(4,4)
    christoffel.append(temp)

# Calculate Christoffel symbols
for k in range(LENGTH):
    for i in range(LENGTH):
        for j in range(LENGTH):
            curr_matrix = christoffel[i]
            curr_matrix[j,k] = 0.5 * (diff(D[k, j], q[i]) + diff(D[k, i], q[j])
↪- diff(D[i, j], q[k]))

# Display the resulting Christoffel symbols
print("Christoffel Symbols:")
display(christoffel)

```

Christoffel Symbols:

```

[Matrix([
[
↵
↵
↵
↵
↵
↵
↵
↵
↵
↵
0, -0.5*m1*(-2*a1*sin(q1) - 2*c1x*sin(q1) - 2*c1y*cos(q1))*(a1*cos(q1) + ↵
↵c1x*cos(q1) - c1y*sin(q1)) - 0.5*m1*(-a1*sin(q1) - c1x*sin(q1) - ↵
↵c1y*cos(q1))*(-2*a1*cos(q1) - 2*c1x*cos(q1) + 2*c1y*sin(q1)) - 0. ↵
↵5*m2*(-2*a1*sin(q1) - 2*a2*sin(q1)*cos(q2) - 2*a2*sin(q2)*cos(q1) + ↵
↵2*c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + 2*c2y*(sin(q1)*sin(q2) - ↵
↵cos(q1)*cos(q2)))*(a1*cos(q1) - a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) + ↵
↵c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) - ↵
↵sin(q2)*cos(q1))) - 0.5*m2*(-a1*sin(q1) - a2*sin(q1)*cos(q2) - ↵
↵a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + ↵
↵c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(-2*a1*cos(q1) + 2*a2*sin(q1)*sin(q2) ↵
↵- 2*a2*cos(q1)*cos(q2) + 2*c2x*(sin(q1)*sin(q2) - cos(q1)*cos(q2)) + ↵
↵2*c2y*(sin(q1)*cos(q2) + sin(q2)*cos(q1))), 0, 0],
[0.5*m1*(-2*a1*sin(q1) - 2*c1x*sin(q1) - 2*c1y*cos(q1))*(a1*cos(q1) + ↵
↵c1x*cos(q1) - c1y*sin(q1)) + 0.5*m1*(-a1*sin(q1) - c1x*sin(q1) - ↵
↵c1y*cos(q1))*(-2*a1*cos(q1) - 2*c1x*cos(q1) + 2*c1y*sin(q1)) + 0. ↵
↵5*m2*(-2*a1*sin(q1) - 2*a2*sin(q1)*cos(q2) - 2*a2*sin(q2)*cos(q1) + ↵
↵2*c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + 2*c2y*(sin(q1)*sin(q2) - ↵
↵cos(q1)*cos(q2)))*(a1*cos(q1) - a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) + ↵
↵c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) - ↵
↵sin(q2)*cos(q1))) + 0.5*m2*(-a1*sin(q1) - a2*sin(q1)*cos(q2) - ↵
↵a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + ↵
↵c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(-2*a1*cos(q1) + 2*a2*sin(q1)*sin(q2) ↵
↵- 2*a2*cos(q1)*cos(q2) + 2*c2x*(sin(q1)*sin(q2) - cos(q1)*cos(q2)) + ↵
↵2*c2y*(sin(q1)*cos(q2) + sin(q2)*cos(q1))),
↵
↵
↵
↵
↵
↵
↵
↵
↵
↵
0, 0, 0],

```

```

[
↳
↳
↳
↳
↳
↳
↳
↳
↳
↳
0,
↳
↳
↳
↳
↳
↳
↳
↳
↳
↳
0, 0, 0],
[
↳
↳
↳
↳
↳
↳
↳
↳
↳
↳
↳
0,
↳
↳
↳
↳
↳
↳
↳
↳
↳
↳
0, 0, 0]],
Matrix([

```


$0, 0, 0],$

0, 0, □

```
# Define symbolic variables
```

```

qd = symbols('qd:{}'.format(LENGTH), real=True)

# Initialize a square matrix for the Coriolis matrix
C = zeros(LENGTH, LENGTH)

# Calculate the Coriolis matrix
for k in range(LENGTH):
    for j in range(LENGTH):
        temp = 0
        for i in range(LENGTH):
            temp_christoffel = christoffel[i]
            temp += temp_christoffel[j, k] * qd[i]
        C[j, k] = temp

# Display the resulting Coriolis matrix
print("Coriolis Matrix:")
print(C)
display(C.shape)

```

Coriolis Matrix:

```

Matrix([[qd1*(0.5*m1*(-2*a1*sin(q1) - 2*c1x*sin(q1) - 2*c1y*cos(q1))*(a1*cos(q1)
+ c1x*cos(q1) - c1y*sin(q1)) + 0.5*m1*(-a1*sin(q1) - c1x*sin(q1) -
c1y*cos(q1))*(-2*a1*cos(q1) - 2*c1x*cos(q1) + 2*c1y*sin(q1)) +
0.5*m2*(-2*a1*sin(q1) - 2*a2*sin(q1)*cos(q2) - 2*a2*sin(q2)*cos(q1) +
2*c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + 2*c2y*(sin(q1)*sin(q2) -
cos(q1)*cos(q2)))*(a1*cos(q1) - a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) +
c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) -
sin(q2)*cos(q1))) + 0.5*m2*(-a1*sin(q1) - a2*sin(q1)*cos(q2) -
a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) +
c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(-2*a1*cos(q1) + 2*a2*sin(q1)*sin(q2) -
2*a2*cos(q1)*cos(q2) + 2*c2x*(sin(q1)*sin(q2) - cos(q1)*cos(q2)) +
2*c2y*(sin(q1)*cos(q2) + sin(q2)*cos(q1)))), qd0*(-0.5*m1*(-2*a1*sin(q1) -
2*c1x*sin(q1) - 2*c1y*cos(q1))*(a1*cos(q1) + c1x*cos(q1) - c1y*sin(q1)) -
0.5*m1*(-a1*sin(q1) - c1x*sin(q1) - c1y*cos(q1))*(-2*a1*cos(q1) - 2*c1x*cos(q1)
+ 2*c1y*sin(q1)) - 0.5*m2*(-2*a1*sin(q1) - 2*a2*sin(q1)*cos(q2) -
2*a2*sin(q2)*cos(q1) + 2*c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) +
2*c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(a1*cos(q1) - a2*sin(q1)*sin(q2) +
a2*cos(q1)*cos(q2) + c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) +
c2y*(-sin(q1)*cos(q2) - sin(q2)*cos(q1))) - 0.5*m2*(-a1*sin(q1) -
a2*sin(q1)*cos(q2) - a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) -
sin(q2)*cos(q1)) + c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(-2*a1*cos(q1) +
2*a2*sin(q1)*sin(q2) - 2*a2*cos(q1)*cos(q2) + 2*c2x*(sin(q1)*sin(q2) -
cos(q1)*cos(q2)) + 2*c2y*(sin(q1)*cos(q2) + sin(q2)*cos(q1)))],
[qd0*(0.5*m1*(-2*a1*sin(q1) - 2*c1x*sin(q1) - 2*c1y*cos(q1))*(a1*cos(q1) +
c1x*cos(q1) - c1y*sin(q1)) + 0.5*m1*(-a1*sin(q1) - c1x*sin(q1) -
c1y*cos(q1))*(-2*a1*cos(q1) - 2*c1x*cos(q1) + 2*c1y*sin(q1)) +
0.5*m2*(-2*a1*sin(q1) - 2*a2*sin(q1)*cos(q2) - 2*a2*sin(q2)*cos(q1) +
2*c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + 2*c2y*(sin(q1)*sin(q2) -

```

$$\begin{aligned}
& \cos(q_1)\cos(q_2))) * (a_1\cos(q_1) - a_2\sin(q_1)\sin(q_2) + a_2\cos(q_1)\cos(q_2) + \\
& c_{2x}*(-\sin(q_1)\sin(q_2) + \cos(q_1)\cos(q_2)) + c_{2y}*(-\sin(q_1)\cos(q_2) - \\
& \sin(q_2)\cos(q_1))) + 0.5*m_2*(-a_1\sin(q_1) - a_2\sin(q_1)\cos(q_2) - \\
& a_2\sin(q_2)\cos(q_1) + c_{2x}*(-\sin(q_1)\cos(q_2) - \sin(q_2)\cos(q_1)) + \\
& c_{2y}*(\sin(q_1)\sin(q_2) - \cos(q_1)\cos(q_2))) * (-2*a_1\cos(q_1) + 2*a_2\sin(q_1)\sin(q_2) - \\
& 2*a_2\cos(q_1)\cos(q_2) + 2*c_{2x}*(\sin(q_1)\sin(q_2) - \cos(q_1)\cos(q_2)) + \\
& 2*c_{2y}*(\sin(q_1)\cos(q_2) + \sin(q_2)\cos(q_1))) + qd_1*(1.0*m_2*(-a_2\sin(q_1)\sin(q_2) + \\
& a_2\cos(q_1)\cos(q_2) + c_{2x}*(-\sin(q_1)\sin(q_2) + \cos(q_1)\cos(q_2)) + \\
& c_{2y}*(-\sin(q_1)\cos(q_2) - \sin(q_2)\cos(q_1))) * (-a_1\sin(q_1) - a_2\sin(q_1)\cos(q_2) - \\
& a_2\sin(q_2)\cos(q_1) + c_{2x}*(-\sin(q_1)\cos(q_2) - \sin(q_2)\cos(q_1)) + \\
& c_{2y}*(\sin(q_1)\sin(q_2) - \cos(q_1)\cos(q_2))) + 1.0*m_2*(a_2\sin(q_1)\sin(q_2) - \\
& a_2\cos(q_1)\cos(q_2) + c_{2x}*(\sin(q_1)\sin(q_2) - \cos(q_1)\cos(q_2)) + \\
& c_{2y}*(\sin(q_1)\cos(q_2) + \sin(q_2)\cos(q_1))) * (-a_1\sin(q_1) - a_2\sin(q_1)\cos(q_2) - \\
& a_2\sin(q_2)\cos(q_1) + c_{2x}*(-\sin(q_1)\cos(q_2) - \sin(q_2)\cos(q_1)) + \\
& c_{2y}*(\sin(q_1)\sin(q_2) - \cos(q_1)\cos(q_2))) + 1.0*m_2*(-a_2\sin(q_1)\cos(q_2) - \\
& a_2\sin(q_2)\cos(q_1) + c_{2x}*(-\sin(q_1)\cos(q_2) - \sin(q_2)\cos(q_1)) + \\
& c_{2y}*(\sin(q_1)\sin(q_2) - \cos(q_1)\cos(q_2))) * (-a_1\cos(q_1) + a_2\sin(q_1)\sin(q_2) - \\
& a_2\cos(q_1)\cos(q_2) + c_{2x}*(\sin(q_1)\sin(q_2) - \cos(q_1)\cos(q_2)) + \\
& c_{2y}*(\sin(q_1)\cos(q_2) + \sin(q_2)\cos(q_1))) + 1.0*m_2*(-a_2\sin(q_1)\cos(q_2) - \\
& a_2\sin(q_2)\cos(q_1) + c_{2x}*(-\sin(q_1)\cos(q_2) - \sin(q_2)\cos(q_1)) + \\
& c_{2y}*(\sin(q_1)\sin(q_2) - \cos(q_1)\cos(q_2))) * (a_1\cos(q_1) - a_2\sin(q_1)\sin(q_2) + \\
& a_2\cos(q_1)\cos(q_2) + c_{2x}*(-\sin(q_1)\sin(q_2) + \cos(q_1)\cos(q_2)) + \\
& c_{2y}*(-\sin(q_1)\cos(q_2) - \sin(q_2)\cos(q_1))), qd_1*(0.5*m_2*(-a_2\sin(q_1)\sin(q_2) + \\
& a_2\cos(q_1)\cos(q_2) + c_{2x}*(-\sin(q_1)\sin(q_2) + \cos(q_1)\cos(q_2)) + \\
& c_{2y}*(-\sin(q_1)\cos(q_2) - \sin(q_2)\cos(q_1))) * (-2*a_2\sin(q_1)\cos(q_2) - \\
& 2*a_2\sin(q_2)\cos(q_1) + 2*c_{2x}*(-\sin(q_1)\cos(q_2) - \sin(q_2)\cos(q_1)) + \\
& 2*c_{2y}*(\sin(q_1)\sin(q_2) - \cos(q_1)\cos(q_2))) + 0.5*m_2*(2*a_2\sin(q_1)\sin(q_2) - \\
& 2*a_2\cos(q_1)\cos(q_2) + 2*c_{2x}*(\sin(q_1)\sin(q_2) - \cos(q_1)\cos(q_2)) + \\
& 2*c_{2y}*(\sin(q_1)\cos(q_2) + \sin(q_2)\cos(q_1))) * (-a_2\sin(q_1)\cos(q_2) - \\
& a_2\sin(q_2)\cos(q_1) + c_{2x}*(-\sin(q_1)\cos(q_2) - \sin(q_2)\cos(q_1)) + \\
& c_{2y}*(\sin(q_1)\sin(q_2) - \cos(q_1)\cos(q_2))))]]))
\end{aligned}$$

(2, 2)

```
[28]: from sympy import diff, symbols, zeros, simplify
```

```

# Calculate the gravitational terms
G = zeros(LENGTH, 1)
for k in range(LENGTH):
    G[k] = diff(PE, q[k])

qdd_matrix = Matrix([qdd]).T
qd_matrix = Matrix([qd]).T

# Calculate the left-hand side of the equations of motion
eom_lhs = D * qdd_matrix + C * qd_matrix + G
simplified_matrix = eom_lhs.applyfunc(simplify)

```

```
# Display the resulting gravitational terms and equations of motion
print("\nEquations of Motion (eom_lhs):")
display(simplified_matrix)
```

Gravitational Terms (G):

Matrix([[0], [0]])

Equations of Motion (eom_lhs):

```
Matrix([[qd0*qd1*(-0.5*m1*(-2*a1*sin(q1) - 2*c1x*sin(q1) -
2*c1y*cos(q1))*(a1*cos(q1) + c1x*cos(q1) - c1y*sin(q1)) - 0.5*m1*(-a1*sin(q1) -
c1x*sin(q1) - c1y*cos(q1))*(-2*a1*cos(q1) - 2*c1x*cos(q1) + 2*c1y*sin(q1)) -
0.5*m2*(-2*a1*sin(q1) - 2*a2*sin(q1)*cos(q2) - 2*a2*sin(q2)*cos(q1) +
2*c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + 2*c2y*(sin(q1)*sin(q2) -
cos(q1)*cos(q2)))*(a1*cos(q1) - a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) +
c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) -
sin(q2)*cos(q1))) - 0.5*m2*(-a1*sin(q1) - a2*sin(q1)*cos(q2) -
a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) +
c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(-2*a1*cos(q1) + 2*a2*sin(q1)*sin(q2) -
2*a2*cos(q1)*cos(q2) + 2*c2x*(sin(q1)*sin(q2) - cos(q1)*cos(q2)) +
2*c2y*(sin(q1)*cos(q2) + sin(q2)*cos(q1)))) + qd0*qd1*(0.5*m1*(-2*a1*sin(q1) -
2*c1x*sin(q1) - 2*c1y*cos(q1))*(a1*cos(q1) + c1x*cos(q1) - c1y*sin(q1)) +
0.5*m1*(-a1*sin(q1) - c1x*sin(q1) - c1y*cos(q1))*(-2*a1*cos(q1) - 2*c1x*cos(q1)
+ 2*c1y*sin(q1)) + 0.5*m2*(-2*a1*sin(q1) - 2*a2*sin(q1)*cos(q2) -
2*a2*sin(q2)*cos(q1) + 2*c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) +
2*c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(a1*cos(q1) - a2*sin(q1)*sin(q2) +
a2*cos(q1)*cos(q2) + c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) +
c2y*(-sin(q1)*cos(q2) - sin(q2)*cos(q1))) + 0.5*m2*(-a1*sin(q1) -
a2*sin(q1)*cos(q2) - a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) -
sin(q2)*cos(q1)) + c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(-2*a1*cos(q1) +
2*a2*sin(q1)*sin(q2) - 2*a2*cos(q1)*cos(q2) + 2*c2x*(sin(q1)*sin(q2) -
cos(q1)*cos(q2)) + 2*c2y*(sin(q1)*cos(q2) + sin(q2)*cos(q1)))) + qdd0*(Izz1 +
Izz2 + m1*(-a1*sin(q1) - c1x*sin(q1) - c1y*cos(q1))**2 + m1*(a1*cos(q1) +
c1x*cos(q1) - c1y*sin(q1))**2 + m2*(-a1*sin(q1) - a2*sin(q1)*cos(q2) -
a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) +
c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))**2 + m2*(a1*cos(q1) -
a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) + c2x*(-sin(q1)*sin(q2) +
cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)))**2) + qdd1*(Izz2 +
m2*(-a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) + c2x*(-sin(q1)*sin(q2) +
cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)))*(a1*cos(q1) -
a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) + c2x*(-sin(q1)*sin(q2) +
cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) - sin(q2)*cos(q1))) +
m2*(-a2*sin(q1)*cos(q2) - a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) -
sin(q2)*cos(q1)) + c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(-a1*sin(q1) -
a2*sin(q1)*cos(q2) - a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) -
sin(q2)*cos(q1)) + c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2))))],
[qd0*(qd0*(0.5*m1*(-2*a1*sin(q1) - 2*c1x*sin(q1) - 2*c1y*cos(q1))*(a1*cos(q1) +
c1x*cos(q1) - c1y*sin(q1)) + 0.5*m1*(-a1*sin(q1) - c1x*sin(q1) -
```

```

c1y*cos(q1))*(-2*a1*cos(q1) - 2*c1x*cos(q1) + 2*c1y*sin(q1)) +
0.5*m2*(-2*a1*sin(q1) - 2*a2*sin(q1)*cos(q2) - 2*a2*sin(q2)*cos(q1) +
2*c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + 2*c2y*(sin(q1)*sin(q2) -
cos(q1)*cos(q2)))*(a1*cos(q1) - a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) +
c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) -
sin(q2)*cos(q1))) + 0.5*m2*(-a1*sin(q1) - a2*sin(q1)*cos(q2) -
a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) +
c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(-2*a1*cos(q1) + 2*a2*sin(q1)*sin(q2) -
2*a2*cos(q1)*cos(q2) + 2*c2x*(sin(q1)*sin(q2) - cos(q1)*cos(q2)) +
2*c2y*(sin(q1)*cos(q2) + sin(q2)*cos(q1)))) + qd1*(1.0*m2*(-a2*sin(q1)*sin(q2) +
a2*cos(q1)*cos(q2) + c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) +
c2y*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)))*(-a1*sin(q1) - a2*sin(q1)*cos(q2) -
a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) +
c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2))) + 1.0*m2*(a2*sin(q1)*sin(q2) -
a2*cos(q1)*cos(q2) + c2x*(sin(q1)*sin(q2) - cos(q1)*cos(q2)) +
c2y*(sin(q1)*cos(q2) + sin(q2)*cos(q1)))*(-a1*sin(q1) - a2*sin(q1)*cos(q2) -
a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) +
c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2))) + 1.0*m2*(-a2*sin(q1)*cos(q2) -
a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) +
c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(-a1*cos(q1) + a2*sin(q1)*sin(q2) -
a2*cos(q1)*cos(q2) + c2x*(sin(q1)*sin(q2) - cos(q1)*cos(q2)) +
c2y*(sin(q1)*cos(q2) + sin(q2)*cos(q1))) + 1.0*m2*(-a2*sin(q1)*cos(q2) -
a2*sin(q2)*cos(q1) + c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) +
c2y*(sin(q1)*sin(q2) - cos(q1)*cos(q2)))*(a1*cos(q1) - a2*sin(q1)*sin(q2) +
a2*cos(q1)*cos(q2) + c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) +
c2y*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)))) +
qd1**2*(0.5*m2*(-a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) + c2x*(-sin(q1)*sin(q2)
+ cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) -
sin(q2)*cos(q1)))*(-2*a2*sin(q1)*cos(q2) - 2*a2*sin(q2)*cos(q1) +
2*c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + 2*c2y*(sin(q1)*sin(q2) -
cos(q1)*cos(q2))) + 0.5*m2*(2*a2*sin(q1)*sin(q2) - 2*a2*cos(q1)*cos(q2) +
2*c2x*(sin(q1)*sin(q2) - cos(q1)*cos(q2)) + 2*c2y*(sin(q1)*cos(q2) +
sin(q2)*cos(q1)))*(-a2*sin(q1)*cos(q2) - a2*sin(q2)*cos(q1) +
c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + c2y*(sin(q1)*sin(q2) -
cos(q1)*cos(q2)))) + qdd0*(Izz2 + m2*(-a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) +
c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) -
sin(q2)*cos(q1)))*(a1*cos(q1) - a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) +
c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) -
sin(q2)*cos(q1))) + m2*(-a2*sin(q1)*cos(q2) - a2*sin(q2)*cos(q1) +
c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + c2y*(sin(q1)*sin(q2) -
cos(q1)*cos(q2)))*(-a1*sin(q1) - a2*sin(q1)*cos(q2) - a2*sin(q2)*cos(q1) +
c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + c2y*(sin(q1)*sin(q2) -
cos(q1)*cos(q2)))) + qdd1*(Izz2 + m2*(-a2*sin(q1)*sin(q2) + a2*cos(q1)*cos(q2) +
c2x*(-sin(q1)*sin(q2) + cos(q1)*cos(q2)) + c2y*(-sin(q1)*cos(q2) -
sin(q2)*cos(q1)))**2 + m2*(-a2*sin(q1)*cos(q2) - a2*sin(q2)*cos(q1) +
c2x*(-sin(q1)*cos(q2) - sin(q2)*cos(q1)) + c2y*(sin(q1)*sin(q2) -
cos(q1)*cos(q2))))**2]]]

```