In [ ]:
```
%matplotlib widget

import sympy as sp
import sympy.physics.mechanics as me
import sympy.plotting as splt
from typing import List
from sympy import sin, cos, pi, sqrt, acos, simplify, atan
import math
me.init_vprinting()

g, t = sp.symbols('g, t')

def homogeneous(rotation: sp.Matrix = sp.eye(3), translation: sp.Matrix = sp
    return rotation.row_join(translation).col_join(sp.Matrix([[0, 0, 0, 1]])

def dh(rotation, twist, displacement, offset):
    rotation_mat = sp.Matrix([
        [cos(rotation), -sin(rotation)*cos(twist),  sin(rotation)*sin(twist)
        [sin(rotation), cos(rotation)*cos(twist),   -cos(rotation)*sin(twist
        [0,             sin(twist),                 cos(twist)],
    ])
    translation = sp.Matrix([
        [offset*cos(rotation)],
        [offset*sin(rotation)],
        [displacement],
    ])
    return rotation_mat, translation

def rotation(homogeneous: sp.Matrix):
    return homogeneous[:3, :3]

def translation(homogeneous: sp.Matrix):
    return homogeneous[:3, 3:]

def chained_transform(transforms: List[sp.Matrix]):
    transforms_chained = [homogeneous()]
    for transform in transforms:
        transforms_chained.append(transforms_chained[-1] * transform)
    return transforms_chained

def z_vecs(transforms: List[sp.Matrix]):
    transforms_chained = chained_transform(transforms)
    z_unit_vecs = []
    for transform in transforms_chained:
        z_unit_vecs.append(rotation(transform) * sp.Matrix([0, 0, 1]))
    return z_unit_vecs

def jacobian(transforms: List[sp.Matrix], joint_types: List[sp.Matrix], base
    transforms_chained = chained_transform(transforms)
    z_unit_vecs = z_vecs(transforms)

    assert len(transforms_chained) == len(z_unit_vecs)

    jacobian = sp.zeros(6, len(transforms))
```

```
    for i, (transform, joint_type) in enumerate(zip(transforms, joint_types)
        if joint_type == 'revolute':
            jacobian[:3, i] = z_unit_vecs[i].cross(translation(transforms_ch
            jacobian[3:, i] = z_unit_vecs[i]
        elif joint_type == 'prismatic':
            jacobian[:3, i] = z_unit_vecs[i]
            jacobian[3:, i] = sp.Matrix([[0], [0], [0]])

        # angular velocity

    return jacobian

def skew(v: sp.Matrix):
    return sp.Matrix([
        [0, -v[2], v[1]],
        [v[2], 0, -v[0]],
        [-v[1], v[0], 0],
    ])

def compute_dynamics(all_joints: List[sp.Matrix], joint_types: List[str], q_
    J = jacobian(all_joints, joint_types)

    w = [sp.Matrix([0, 0, 0])]*(len(all_joints)+1) # joint linear velocities
    v = [sp.Matrix([0, 0, 0])]*(len(all_joints)+1) # joint angular velocitie
    v_c = [sp.Matrix([0, 0, 0])]*(len(all_joints)+1) # joint CoM linear velo
    T = [sp.Matrix([0])]*(len(all_joints)+1) # joint kinetic energy
    V = [sp.Matrix([0])]*(len(all_joints)+1) # joint potential energy

    chained_transforms = chained_transform(all_joints)
    # chained_translations = chained_translation(all_joints)
    # z = z_vecs(all_joints) # joint origins
    z = sp.Matrix([0, 0, 1]) # base z vector

    for i, joint, joint_type in zip(range(1, len(all_joints) + 1), all_joint
        # Compute angular velocity
        theta_dot = q_dot[i-1] if joint_type == 'revolute' else 0
        w[i] = rotation(joint).T * (w[i-1] + z*theta_dot)

        # Compute linear velocity
        d_dot = q_dot[i-1] if joint_type == 'prismatic' else 0
        r_i = (joint*sp.Matrix([0, 0, 0, 1]))[:3, :]
        v[i] = rotation(joint).T * (v[i-1] + z*d_dot) + w[i].cross(r_i)

        # Compute CoM linear velocity
        v_c[i] = v[i] + w[i].cross(r_c[i][:3, :])

        # Compute kinetic energy
        T[i] = 0.5*m[i]*v_c[i].T*v_c[i] + 0.5*w[i].T*I[i]*w[i]

        # Compute potential energy
        p_ci = (chained_transforms[i]*r_c[i])[:3, :]
        V[i] = -m[i]*sp.Matrix([0, -g, 0]).T*p_ci

    return w, v, v_c, T, V
```

In [ ]:
```
L_1, L_2, L_c1, L_c2, r_c1, r_c2, m_1, m_2, I_c1, I_c2 = sp.symbols('L_1, L_
theta_1, theta_2 = me.dynamicsymbols('theta_1, theta_2')
q = sp.Matrix([theta_1, theta_2])
q_dot = q.diff(t)
m = [0, m_1, m_2]
I = [0, I_c1, I_c2]
r_c = [0, sp.Matrix([L_c1, 0, 0, 1]), sp.Matrix([L_c2, 0, 0, 1])]
joint_1 = homogeneous(*dh(theta_1, 0, 0, L_1))
joint_2 = homogeneous(*dh(theta_2, 0, 0, L_2))
all_joints = [joint_1, joint_2]
joint_types = ['revolute', 'revolute']

w, v, v_c, T, V = compute_dynamics([joint_1, joint_2], joint_types, q_dot, m
T_sum = sp.zeros(1)
V_sum = sp.zeros(1)
for t_i in T:
    T_sum += t_i
T = simplify(T_sum)
for v_i in V:
    V_sum += v_i
V = simplify(V_sum)
J = jacobian(all_joints, joint_types)[:2, :]
J
```

Out[ ]:

$$\begin{bmatrix} -L_1 \sin(\theta_1) - L_2 \sin(\theta_1)\cos(\theta_2) - L_2 \sin(\theta_2)\cos(\theta_1) & -L_2 \sin(\theta_1)\cos(\theta_2) - L_2 \text{s} \\ L_1 \cos(\theta_1) - L_2 \sin(\theta_1)\sin(\theta_2) + L_2 \cos(\theta_1)\cos(\theta_2) & -L_2 \sin(\theta_1)\sin(\theta_2) + L_2 \text{c} \end{bmatrix}$$

In [ ]:
```
# Solve for joint velocities
p_dot = sp.Matrix([0.5, 0])
subs = {
    theta_1: math.radians(30),
    theta_2: math.radians(-90),
    L_1: 1,
    L_2: 1,
    m_1: 1,
    m_2: 1
}
J.inv().evalf(subs=subs) * p_dot
```

Out[ ]:

$$\begin{bmatrix} -0.25 \\ 0.683012701892219 \end{bmatrix}$$

In [ ]:
```
L = T - V
display(simplify(L))
f_x, f_y, f_z, g_x, g_y, g_z = sp.symbols('f_x, f_y, f_z, g_x, g_y, g_z')
subs = {
    theta_1: math.radians(30),
    theta_2: math.radians(-90),
    L_1: 1,
    L_2: 1,
    m_1: 1,
    m_2: 1,
    f_x: 0,
    f_y: -50,
```

```
    # f_z: 0,
    # g_x: 0,
    # g_y: 0,
    # g_z: 0,
    theta_1.diff(t): -0.25,
    theta_2.diff(t): 0.683012701892219,
    I_c1: sp.Identity(3),
    I_c2: sp.Identity(3),
    L_c1: 0.5,
    L_c2: 0.5,
    g: 9.81
}
F_ext = sp.Matrix([f_x, f_y])
temp = L.diff(q_dot).diff(t) - L.diff(q)
temp = sp.Matrix([temp[0][0][0][0], temp[1][0][0][0]])
simplify((temp - J.T*F_ext).evalf(subs=subs))
```

$$\left[\begin{array}{r} 0.5I_{c1}\dot{\theta}_1^2 + 0.5I_{c2}\left(\dot{\theta}_1 + \dot{\theta}_2\right)^2 + 0.5L_1^2 m_1 \sin^2(\theta_1)\dot{\theta}_1^2 - g\left(m_1\left(L_1 + L_{c1}\right)\sin(\theta_1) + r \right. \\ + 0.5m_2\left(L_1 \sin(\theta_1 - \theta_2)\dot{\theta}_1 + L_2\sin(\theta_2)\dot{\theta}_1 + L_2\sin(\theta_2)\dot{\theta}_2\right)^2 + 0.5m \end{array}\right.$$

Out[ ]: $\begin{bmatrix} 97.1114846863766 + 0 \\ 32.5991265877365 + 0 \end{bmatrix}$

In [ ]: