Assignment 1: Modeling (Solution)

University of California Berkeley

ME C231A, EE C220B, Experiential Advanced Control I

Question 1. Equilibrium Point and Linearization

Part (a)

At equilibrium $\dot{\theta}_1 = \dot{\theta}_2 = \ddot{\theta}_1 = \ddot{\theta}_2 = 0$. We set $\bar{\theta}_2 = 0$ and get

$$\begin{aligned} mgl\sin\theta_1 - a^2k\cos\theta_1\sin\theta_1 + T(t) = 0 \\ a^2k\sin\theta_1 + \alpha = 0 \end{aligned}$$

After some rearrangement we get

$$\begin{aligned} \bar{\theta}_1 &= \arcsin\left(-\frac{\alpha}{a^2k}\right) \\ \bar{T} &= mgl\frac{\alpha}{a^2k} - \alpha\cos\arcsin\left(-\frac{\alpha}{a^2k}\right) \end{aligned}$$

Using $\cos\arcsin(x) = \sqrt{1-x^2}$, the equation for $\bar{T}$ can be simplified to

$$\bar{T} = \frac{\alpha}{a^2k}\left(mgl - \sqrt{a^4k^2 - \alpha^2}\right)$$

Part (b)

The new equilibrium point is $\bar{\theta}_1 = \bar{\theta}_2 = \bar{T} = 0$. We set the state and input variables as follows

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \theta_1 - \bar{\theta}_1 = \theta_1 \\ \dot{x}_1 \\ \theta_2 - \bar{\theta}_2 = \theta_2 \\ \dot{x}_2 \end{bmatrix}, \qquad u = T(t) - \bar{T} = T(t)$$

The system of four 1st order ODE $\dot{x} = f(x,u)$ can be written as

$$f(x,u) = \begin{bmatrix} x_2 \\ \frac{1}{ml^2}\left(mgl\sin x_1 + a^2\cos x_1\left(k(\sin x_3 - \sin x_1) + d(x_4 - x_2)\right) + u\right) \\ x_4 \\ \frac{1}{ml^2}\left(mgl\sin x_3 - a^2\cos x_3\left(k(\sin x_3 - \sin x_1) + d(x_4 - x_2)\right) + \alpha + \beta x_4^2\right) \end{bmatrix}$$

Taking the hint into account and linearizing with

$$A = \left.\frac{\partial f(x,u)}{\partial x}\right|_{x_{ss},u_{ss}}, \qquad B = \left.\frac{\partial f(x,u)}{\partial u}\right|_{x_{ss},u_{ss}}$$

leads to a state space description of the form $\dot{x} = Ax + Bu$ given by

$$\begin{aligned} \dot{x}(t) &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{g}{l} - \frac{a^2k}{ml^2} & -\frac{a^2d}{ml^2} & \frac{a^2k}{ml^2} & \frac{a^2d}{ml^2} \\ 0 & 0 & 0 & 1 \\ \frac{a^2k}{ml^2} & \frac{a^2d}{ml^2} & \frac{g}{l} - \frac{a^2k}{ml^2} & -\frac{a^2d}{ml^2} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \\ 0 \\ 0 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} x(t) + 0 \cdot u(t) \end{aligned}$$

In [1]:
```python
import numpy as np
from numpy import sin, cos, tan, arcsin
```

In [2]:
```python
def EquilPoint(theta2, alpha):
    a = 3
    k= 1.5
    m = 2
    g = 9.81
    l = 6
    d = 1
    beta = 0.1
    temp = (-m*g*l*sin(theta2)-alpha+a**2*k*cos(theta2)*sin(theta2))/(a**2*k*c
os(theta2))
    theta1 = arcsin(temp)
    T = -m*g*l*sin(theta1)-a**2*k*cos(theta1)*(sin(theta2)-sin(theta1))
    return  theta1, T

# For example for the fixed values of theta2 = 0.1 and alpha = 0.5
theta1_bar, T_bar = EquilPoint(theta2=0.1, alpha=0.5)
print(theta1_bar)
print(T_bar)
```

```
-0.948098425224013
88.44306254140133
```

In [3]:
```
import sympy as sym

def LinearizeModel(theta2, alpha):

    a = 3
    k = 1.5
    m = 2
    g = 9.81
    l = 6
    d = 1
    beta = 0.1
    theta1dot = 0
    theta2dot = 0

    theta1, T = EquilPoint(theta2, alpha)  # calling the above EquilPoint func
tion

    x1, x2, x3, x4, u1, u2 = sym.symbols('x1 x2 x3 x4 u1 u2')
    f = sym.Matrix([x2, (m*g*l*sym.sin(x1)+a**2*sym.cos(x1)*(k*(sym.sin(x3)-sy
m.sin(x1))+d*(x4-x2))+u1)/(m*l**2),
                x4, (m*g*l*sym.sin(x3)-a**2*sym.cos(x3)*(k*(sym.sin(x3)-sym.si
n(x1))+d*(x4-x2))+u2+beta*x4**2)/(m*l**2)])
    Asym = f.jacobian([x1, x2, x3, x4, u1, u2])
    Asub = Asym.subs([(x1, theta1), (x2, theta1dot), (x3, theta2), (x4, theta2
dot), (u1, T), (u2, alpha)])

    Bsym = f.jacobian([u1, u2])
    Bsub = Bsym.subs([(x1, theta1), (x2, theta1dot), (x3, theta2), (x4, theta2
dot), (u1, T), (u2, alpha)])

    C = np.array([1, 0, 0, 0])
    D = np.array([0, 0])

    return Asub, Bsub, C, D, theta1_bar, T_bar

# Calling the function
# for example consider the values of theta2_bar and alpha_bar to be 0.1 and 0.
5, respectively
theta2_bar = 0.1
alpha_bar = 0.5
A, B, C, D, theta1_bar, T_bar = LinearizeModel(theta2_bar, alpha_bar)
print('A = ', A)
print('B = ', B)
print('C = ', C)
print('D = ', D)
print('theta1_bar = ', theta1_bar)
print('T_bar = ', T_bar)
```

```
A =  Matrix([[0, 1, 0, 0, 0, 0], [1.02872606084634, -0.0729036009073981, 0.10
8809079849942, 0.0729036009073981, 1/72, 0], [0, 0, 0, 1, 0, 0], [0.108809079
849942, 0.124375520659753, 1.45827472846828, -0.124375520659753, 0, 1/72]])
B =  Matrix([[0, 0], [1/72, 0], [0, 0], [0, 1/72]])
C =  [1 0 0 0]
D =  [0 0]
theta1_bar =  -0.948098425224013
T_bar =  88.44306254140133
```

Question 2. Euler Discretization of a Building Heat Transfer Model

Part (a)

Starting with our original equation:

$$m_z c_z \dot{T} = q + c_p u_1 (u_2 - T)$$

Use the Euler discretization approximation:

$$\boxed{\dot{T} = \frac{T(k+1) - T(k)}{T_s}}$$

Substitute in:

$$m_z c_z \frac{T(k+1) - T(k)}{T_s} = q(k) + c_p u_1(k)(u_2(k) - T(k))$$

Reshuffle variables:

$$T(k+1) = \left[1 - \frac{c_p T_s}{m_z c_z} u_1(k)\right] T(k) + \frac{T_s}{m_z c_z}[q(k) + c_p u_1(k) u_2(k)]$$

In [4]:
```
def bldgHTM(T, u1, u2, q, mz, cz, cp):
    Tdot = (q+cp*u1*(u2-T))/(mz*cz)
    return Tdot
```

Part (b)

In [5]:
```
def eulerDiscretization(T,q,u1,u2):
    mz = 100
    cz = 20
    Ts = 0.1 # Discretization sampling time
    cp = 1000
    T_KplusOne = (1-cp*Ts/(mz*cz)*u1)*T+Ts/(mz*cz)*(q+cp*u1*u2)
    return T_KplusOne
```

Question 3. Simulation of Nonlinear Bicycle Dynamics

Part (a)

In [6]:

```python
import matplotlib.pyplot as plt

def carModel(beta, a, x, y, psi, v):

    l_r   = 1.738
    dt = 0.1

    x_dot   = v*cos(psi+beta)
    y_dot   = v*sin(psi+beta)
    psi_dot = (v/l_r)*sin(beta)
    v_dot   = a

    x_out   = x + x_dot*dt
    y_out   = y + y_dot*dt
    psi_out = psi + psi_dot*dt
    v_out   = v + v_dot*dt

    return x_out, y_out, psi_out, v_out

# Calling the function example
x, y, psi, v = carModel(0.1,2,5,2,10,0.1)
print(x)
print(y)
print(psi)
print(v)
```

```
4.992194318198308
1.9937492935110712
10.000574415515805
0.30000000000000004
```

Part (b)

In [7]:

```
from scipy.io import loadmat
Data = loadmat('sineData.mat')
a = Data['a']
beta = Data['beta']
time = Data['time']
Ts = 0.1


def sim(time, a, beta, x0):

    numSteps = len(time)

    x = x0[0]
    y = x0[1]
    psi = x0[2]
    v = x0[3]

    # Initialize trends
    xtrend = []
    xtrend.append(x)
    ytrend = []
    ytrend.append(y)
    psitrend = []
    psitrend.append(psi)
    vtrend = []
    vtrend.append(v)

    for i in range(0, numSteps-1):

        x, y, psi, v = carModel(beta[i], a[i], x, y, psi, v)
        xtrend.append(x)
        ytrend.append(y)
        psitrend.append(psi)
        vtrend.append(v)

    return np.asarray(xtrend, dtype=object), np.asarray(ytrend, dtype=object),
np.asarray(psitrend, dtype=object), np.asarray(vtrend, dtype=object)

# calling the function
x = 0.0
y = 0.0
psi = 0.0
v = 0.0
numSteps = len(time)

x0 = np.array([x , y, psi, v])
xtrend, ytrend, psitrend, vtrend = sim(time, a, beta, x0)
```

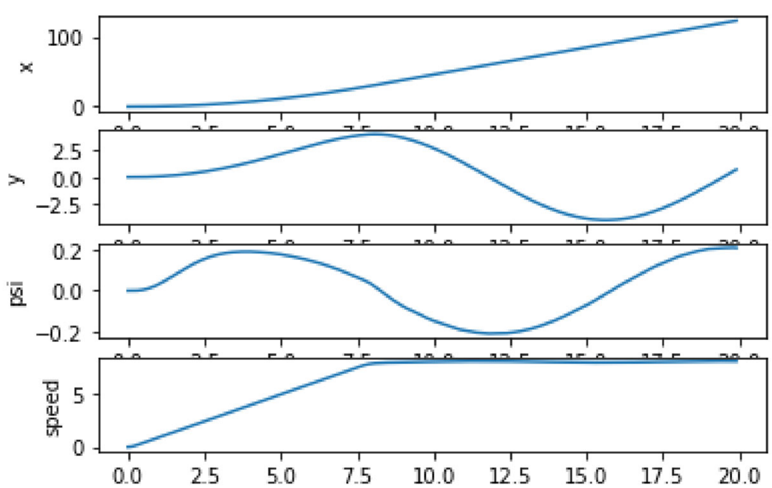
with given sequences of **time** and inputs **a** and **beta**. **time** is a vector of sampled time instants with sampling T_s starting at 0 seconds. You should use the function you write in part a. The input arguments (**time**, **a**, **beta**) should be 1001-by-1 vectors. **x0** should be a 4-by-1 vector. The output arguments should be all 1002-by-1 vectors.

Part (c)

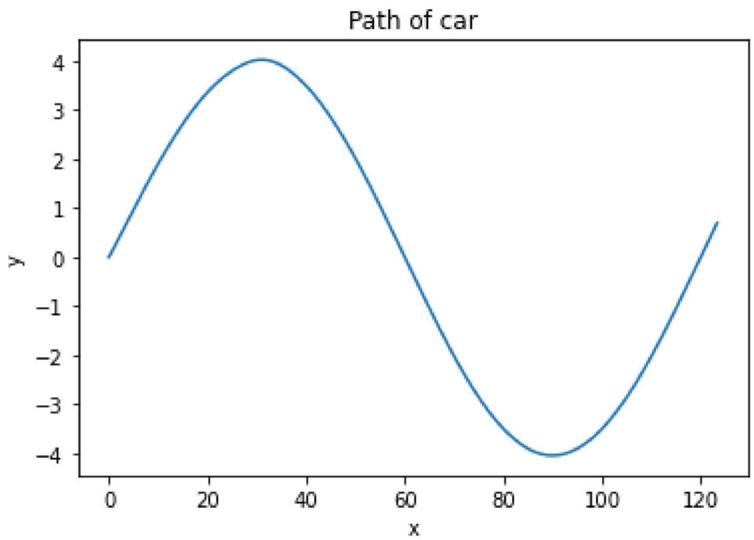
Write a script to test your code by using the file **sineData.mat**. This file contains three vectors: **time**, and the corresponding inputs at each time instant: **a** (acceleration) and **beta** (β). Make a plot of the states versus time and y versus x. Create a simple animation that follows the trajectory. Use a bicycle or car shape that makes sense. Make sure the bike is pointing in the correct direction at every point.

In [8]:

```
dt = time[1]-time[0]

# Shorten simulation time
numSteps = 200
time_red = time[0:numSteps]
xtrend_red = xtrend[0:numSteps]
ytrend_red = ytrend[0:numSteps]
psitrend_red = psitrend[0:numSteps]
vtrend_red = vtrend[0:numSteps]


# Plot the results

plt.subplot(4,1,1)
plt.plot(time_red, xtrend_red)
plt.ylabel('x')
plt.subplot(4,1,2)
plt.plot(time_red, ytrend_red)
plt.ylabel('y')
plt.subplot(4,1,3)
plt.plot(time_red, psitrend_red)
plt.ylabel('psi')
plt.subplot(4,1,4)
plt.plot(time_red, vtrend_red)
plt.ylabel('speed')
plt.show()
```



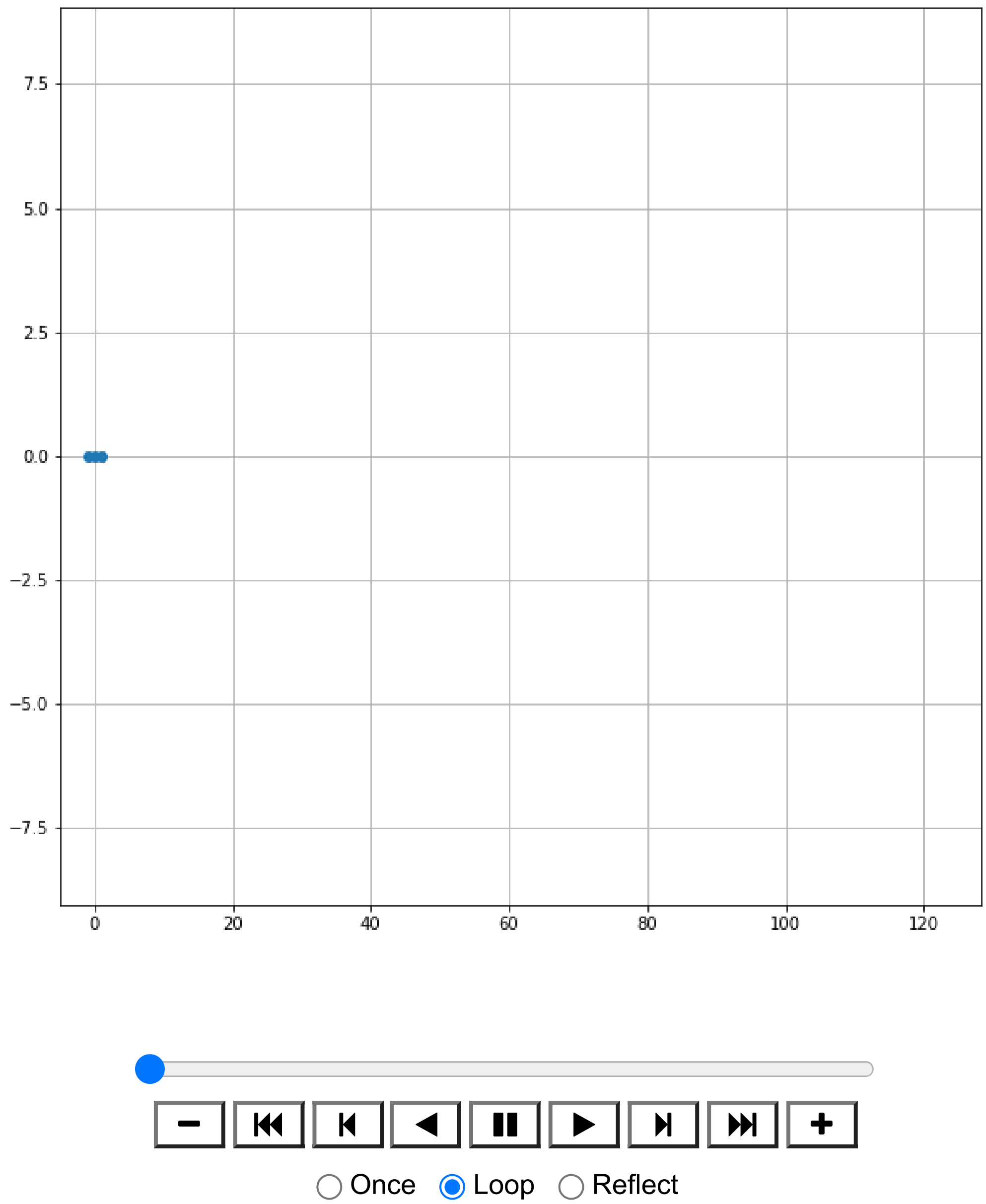
In [9]:
```
# Plot the path
plt.plot(xtrend_red,ytrend_red)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Path of car')
plt.show()
```

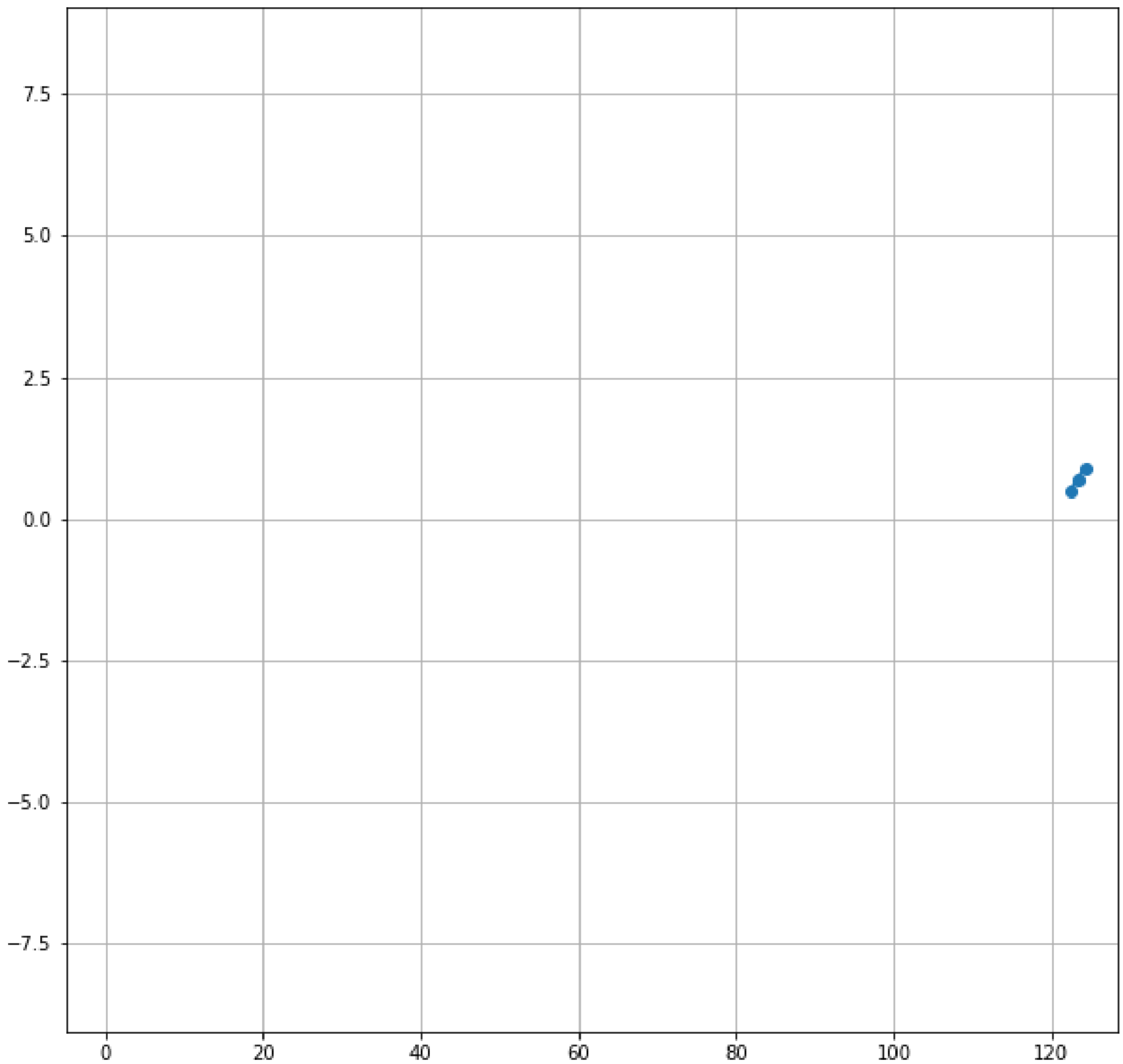


In [10]:

```
# Plot animation
from matplotlib import animation, rc
from IPython.display import HTML
w = 1
h = 2

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, autoscale_on=False, xlim=(np.min(xtrend_red)-5, np.m
ax(xtrend_red)[0]+5), ylim=(np.min(ytrend_red)[0]-5, np.max(ytrend_red)[0]+5))
ax.grid()

line, = ax.plot([], [], 'o-', lw=2)

def init():
    line.set_data([], [])
    return line,

def animate(i):
    x = xtrend_red[i]
    y = ytrend_red[i]
    psi = psitrend_red[i]
    bikex = [x+cos(psi), x, x-cos(psi)]
    bikey = [y+sin(psi), y, y-sin(psi)]
    line.set_data(bikex, bikey)
    return line,

ani = animation.FuncAnimation(fig, animate, range(1, numSteps),
                              interval=0.1*1000, blit=True, init_func=init)
rc('animation', html='jshtml')
ani
```

Out[10]:





Question 4. Analysis of LTI Discrete-Time Systems

1. Let $\alpha = 0$. Is the system stable? - No, one of the eigenvalues is outside the unit circle ($\lambda_{1,2,3,4} = \frac{1}{3}, \frac{1}{3}, -\frac{1}{2}, -\frac{5}{4}$)
2. Now let $\alpha = \frac{1}{2}$. Is the system stable? No, again there is an eigenvalue outside the unit circle ($\lambda_{1,2,3,4} = \frac{1}{3}, \frac{1}{3}, -\frac{3}{2}, -\frac{1}{4}$)

In [11]:
```
import scipy.linalg
def isSystemStable(alpha):

    A = np.array([[1/3, 0, 0, 0],
        [0, -1/2, alpha, 0],
        [0, 1/2, -5/4, 0],
        [-1/2, 0, 0, 1/3]])

    eigVals, eigVecs = scipy.linalg.eig(A)
    TF = np.all(abs(eigVals)<1)

    return TF, eigVals
```

In [12]:
```
alpha = 1
TF = isSystemStable(alpha)
TF
```

Out[12]:
```
(False,
 array([-1.67539053+0.j, -0.07460947+0.j,  0.33333333+0.j,  0.33333333+0.j]))
```

In [13]:
```
alpha = -0.5
TF = isSystemStable(alpha)
TF
```

Out[13]:
```
(True,
 array([-0.875     +0.33071891j, -0.875     -0.33071891j,
         0.33333333+0.j        ,  0.33333333+0.j        ]))
```