Linear Controls: Inverted Pendulum

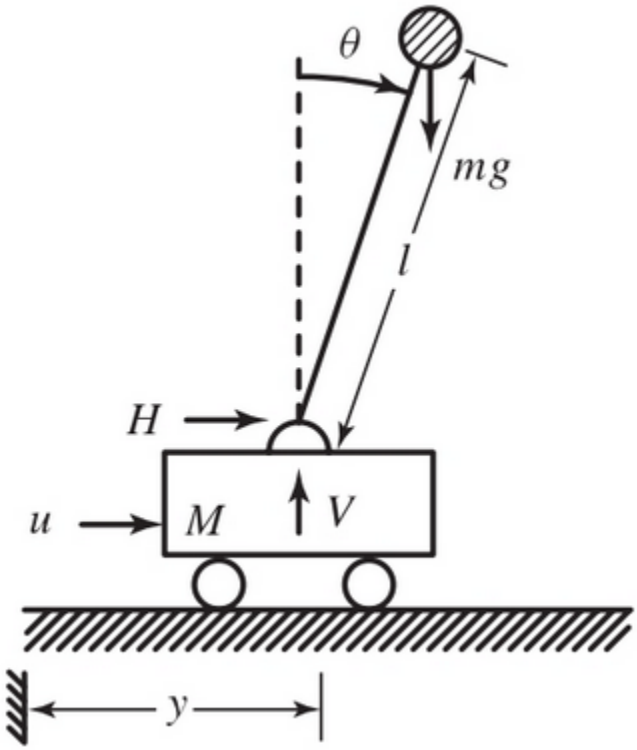
Anoushka Mathews,
Md Shahnewaz Tanvir,
Pratik Kunkolienker

April 27, 2022

Contents

1 Introduction

An inverted pendulum is a system where a mass is rigidly attached to a linearly moving platform. The platform can thus move in a 2-D plane so as to balance the attached mass at fixed angle to the plane of motion. The system is shown in Figure 1



2 Problem Statement

Experiment with Inverted Pendulum System to:

1. Establish its controllability, observability and stability.
2. Use the method of Kalman Decomposition to generate a controllable system.
3. Use Constant Gain Negative Feedback method to generate a stable system.
4. Visualize roots using Root Locus.

3 Literature Review

The course textbook [1] describes the mathematics in deriving the state space equations that represent the system. The equations for the system are given by:

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \\ \dot{x_3}(t) \\ \dot{x_4}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(M+m)g}{Ml} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{-1}{Ml} \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \mathbf{x}(t)$$

where $x\_1(t) = y(t), x\_2(t) = \dot{y}(t), x\_3(t) = \theta(t),$ and $x\_4(t) = \dot{\theta}(t)$. As the goal of the system is to maintain the attached mass in an upright postion, these equations are only valid for $\theta \to 0$. Furthermore, the output is only set to the distance, y, the cart needs to move from the starting point.

A more realistic model is given in [2]. This model accounts for friction between the cart wheels and the ground.

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \\ \dot{x_3}(t) \\ \dot{x_4}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)\mu}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-(ml\mu)}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mml^2} \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{x}(t)$$

where, I is the moment of inertia of the pendulum and μ is the coefficient of friction between the ground and the cart's wheels.

For analysis, the first model is chosen for its simplicity

4 Part I

Set the Parameters to ensure the system is controllable and observable. Prove the system is controllable and observable for the chosen parameters.

[1]:
```
# We first import all the necessary libraries
from control import *
import numpy as np
#%matplotlib
import matplotlib.pyplot as plt
from matplotlib import cm
import sympy as sym
from IPython.display import display, Markdown
from matplotlib.ticker import LinearLocator

from control.matlab import *
```

```
plt.rcParams['figure.figsize'] = [10, 10]
```

For this first part, we will pick some usual values for our parameters: m (mass of the pendulum), M (mass of the robot), l (length of the rod connecting the pendulum and the robot), g (gravity)

$$g = 9.8 \text{ m/s}^2$$
$$M = 2 \text{ kg}$$
$$m = 1 \text{ kg}$$
$$l = 0.5 \text{ m}$$

In the code below, we also set our matrices evaluated at the above values.

[2]:
```
m = 1 # mass of the pendulum.
M = 2 # mass of the cart.
g = 9.81 # gravity.
l = 0.5 # length of the pendulum.

A = np.array([[0,1,0,0],[0,0,-(m*g)/M,0],[0,0,0,1],[0,0,(M+m)*g/(M*l),0]]) #A is␣
 ↪a system matrix.
B = np.array([[0],[1/M],[0],[-1/(M*l)]]) #B is an input matrix.
C = np.array([1,0,0,0]) #C is an Output matrix.
D = 0 #D is a Transmission matrix
```

4.1 Check the Controllabilty:

In the code below, we get the controllability matrix using the `ctrb()` from the python controls library. We then check the rank of this controllability matrix. If the rank is full, the system is controllable.

Here, our controllability matrix is a square matrix. So, we can also check its determinant. If the determinant is non-zero, the system is controllable.

From the results shown below, we can see that the matrix has a non-zero determinant.

Hence, from the observations, we can conclude that the system with the above state parameters is controllable.

[3]:
```
Co = ctrb(A,B)                          # Get the controllability matrix
rows,columns = np.shape(Co)
R1 = np.linalg.matrix_rank(Co)          # Get the rank of the controllability␣
 ↪matrix
if R1 == rows :                         # Check if matrix has full row rank
    print("Rank of the controllability matrix is full. So the system is␣
 ↪Controllable.")
    de1 = np.linalg.det(Co)
    print("The determinant of the matrix is:", round(de1, 2))
```

```
else:
    print("System is not Controllable")
```

```
Rank of the controllability matrix is full. So the system is Controllable.
The determinant of the matrix is: 96.24
```

As can be seen, the system is controllable since the controllability matrix $\mathcal{C}_o$ has full row rank. Further, the determinant of matrix $\mathcal{C}_o$ is non zero.

4.2 Check the Observability:

In the code below, we will use the "obsv()" function from the python controls library. Here, we check the rank of the observability matrix returned by the function. We then also check its determinant for non-singularity.

[4]:
```
Obs = obsv(A,C)                      # Get the observability matrix
rows,columns = np.shape(Obs)
R2=np.linalg.matrix_rank(Obs)        # Get the rank of the observability matrix
if R2 == rows:                       # Check if matrix has full row rank
    print("The rank of the observability matrix is full. So the system is␣
 ↪observable.")
    de2 = np.linalg.det(Obs)
    print("The determinant of the matrix is:", round(de2, 2))
else:
    print("System is not Observable")
```

```
The rank of the observability matrix is full. So the system is observable.
The determinant of the matrix is: 24.06
```

We can see that the system is observable as well. The observability matrix $\mathcal{O}$ is also full rank. Further, the determinant of matrix $\mathcal{O}$ is non zero.

4.2.1 So with the above parameters, the system is controllable and observable.

4.3 Symbolic Regions of Controllability and Observability

The matrix is controllable and observable for the above selected numerical values. In this section we will, try to find a more generic picture of where the system is controllable and where the system is observable.

By solving for the determinant of the controllability matrix, we can check for controllability and the same goes for the observability matrix. This is true only because our controllability matrix and observability matrix are square matrices. We recall that if the determinant of a square matrix is non zero, it can be shown that the rank of the matrix is full.

The goal in this section is to find regions where the system will not be controllable and will not be observable given the state space equations.

In the code below, we will first define the symbolic matrices. Then we will define the symbolic controllability matrix, then we will calculate its determinant symbolically. We will, then, repeat the same process for the observability matrix.

[5]:
```
# Defining our matrices symbolically.
M,m,l = sym.symbols('M m l')

A = sym.Matrix([[0,1,0,0],[0,0,-(m*g)/M,0],[0,0,0,1],[0,0,(M+m)*g/(M*l),0]]) #A␣
↪is a system matrix.
B = sym.Matrix([0,1/M,0,-1/(M*l)]) #B is an input matrix.
C = sym.Matrix([1,0,0,0])
C = C.T
```

Since we're doing this symbolically, we manually make the controllability and observability matrix and find the determinant

[6]:
```
Ctrb = B
Ctrb = Ctrb.col_insert(1,A*B)
Ctrb = Ctrb.col_insert(2,A*A*B)
Ctrb = Ctrb.col_insert(3,A*A*A*B)
Ctrb_det = Ctrb.det()
Ctrb_det
```

[6]:
$$\frac{96.2361}{M^4 l^4}$$

As we can see, the symbolic determinant of the controllability matrix can only be 0 when either $M \to \infty$, or $l \to \infty$ or both. In all other cases, the controllability matrix is non-zero.

The figure below depicts the region of controllability with respect to the mass of the robot and the length of the rod. We can see that the region is quite small.

Next, we will do the same for the observability matrix. We will create the observability matrix manually, and then find its determinant.

[7]:
```
Obs = C
Obs = Obs.row_insert(1,C*A)
Obs = Obs.row_insert(2,C*A*A)
Obs = Obs.row_insert(3,C*A*A*A)
Obs_det = Obs.det()
Obs_det
```

[7]:
$$\frac{96.2361 m^2}{M^2}$$

This shows that the system is un-observable for values of $M \to \infty$ or $m = 0$ For all other values, the system is observable. The image below depicts that this region of un-observability is actually a parabola with asymptotes coming out of the page.

5 Part II

Set the parameters to create an "uncontrollable" form and use Kalman Decomposition to obtain the controllable form.

For this part, the first step was to figure out what values of parameters would make the system uncontrollable. If the rank of the controllability matrix was forced to be made less than the full rank, the system would become uncontrollable. The following code calculates the controllability matrix of the given system.

[8]:
```
# This is for nice looking latex output
sym.init_printing()

# Defining our system matrices
m, M, l, g = sym.symbols('m M l g')

A = sym.Matrix([[0, 1, 0, 0],
                [0, 0, -m*g/M, 0],
                [0, 0, 0, 1],
                [0, 0, (M+m)*g/(M*l), 0]])

B = sym.Matrix([[0],
                [1/M],
                [0],
                [-1/(M*l)]])

C = sym.Matrix([[1, 0, 0, 0]])
```

The following section of code is a helper function that returns true if a value is a floating point number.

[9]:
```
# This function returns True if the parameter num is a float. False, otherwise.
def isfloat(num):
    try:
        float(num)
        return True
    except ValueError:
        return False
```

5.1 Controllability Matrix of the System

The following section of code is a function that takes in a symbolic matrix and returns the controllability matrix and a boolean value. The boolean value is true when the system is controllable and false otherwise.

Using this function, we can see what the controllability matrix of the given system looks like. We can see that the rank of this matrix is full, no matter what values we pick for the parameters

theoretically. However, with enough rounding it is possible to make the controllability be less than full rank. This rounding translates to the limited precision we can get using sensor measurements.

[10]:
```
# This function returns whether or not a particular matrix is controllable. The␣
↪matrices must only be symbolic matrices.
# It also returns the controllability matrix.
def controllable_sym(A, B, dim=4):
    cols = B

    for i in range(1, dim):
        col = sym.Matrix([[(A**i)*B]])
        cols = cols.col_insert(i, col)

    if(cols.rank() == dim):
        return True, cols
    else:
        return False, cols


controllable, cont_matrix = controllable_sym(A, B)

# displaying the controllability Matrix
cont_matrix
```

[10]:
$$\begin{bmatrix} 0 & \frac{1}{M} & 0 & \frac{gm}{M^2 l} \\ \frac{1}{M} & 0 & \frac{gm}{M^2 l} & 0 \\ 0 & -\frac{1}{Ml} & 0 & -\frac{g(M+m)}{M^2 l^2} \\ -\frac{1}{Ml} & 0 & -\frac{g(M+m)}{M^2 l^2} & 0 \end{bmatrix}$$

5.2 Uncontrollable System

The goal now is to make an uncontrollable system. We will make column 1 equal to column 3. (1 indexed). We can do this by following the 2 rules:

$$m >>> M$$

$$M = \frac{g * m}{l}$$

In the following code section, we will define another function that can take numerical matrices and return the controllability matrix and a boolean value describing whether the system is controllable. Then, we will define the variables according to the above rules and verify that this new system is, in fact, uncontrollable.

[11]:
```
# This function checks to see if a numerical matrix consisting of floats is␣
↪controllable.
# It returns the boolean value along with the controllability matrix.
```

```
def controllable(A, B, dim=4):
    cols = B

    for i in range(1, dim):
        col = sym.Matrix([[(A**i)*B]])
        cols = cols.col_insert(i, col)

    r, c = sym.shape(cols)

    for i in range(r*c):
        if(isfloat(cols[i])):
            cols[i] = round(cols[i], 2)

    if(cols.rank() == dim):
        return True, cols
    else:
        return False, cols
```

[12]:
```
calc_m = 10000
calc_g = 9.8
calc_l = 9900
calc_M = (calc_g*calc_m)/(calc_l)


new_A = A.evalf(subs={m:calc_m, M:calc_M, l:calc_l , g:calc_g})
new_B = B.evalf(subs={m:calc_m, M:calc_M, l:calc_l , g:calc_g})
new_C = C.evalf(subs={m:calc_m, M:calc_M, l:calc_l , g:calc_g})

new_controllable, new_cont_matrix = controllable(new_A, new_B)
```

[13]:
```
print("controllable?", new_controllable)

# display the new controllability matrix
new_cont_matrix
```

```
controllable? False
```

[13]:
$$\begin{bmatrix} 0 & 0.1 & 0 & 0.1 \\ 0.1 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

[14]:
```
print("Rank of the above Controllable Matrix:", new_cont_matrix.rank())
```

```
Rank of the above Controllable Matrix: 2
```

5.3 Kalman Decomposition

Since the rank of this new matrix is 2 (less than 4), the new system is not controllable. We can make it controllable using the method of Kalman Decomposition. Let's understand the Kalman Decomposition Process.

1. We generate a matrix Q using the controllability matrix such that Q is inversible. Since the rank of our controllability matrix was 2, we will use 2 linearly independent columns of the controllability matrix as the first 2 columns of Q. We will then fill in the rest of Q such that Q is invertible. In our code, we will simply find the nullspace of the 2 vectors we get from the controllability matrix.

$$Q = \begin{bmatrix} 0 & 0.1 & 0 & 0 \\ 0.1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Find P

$$P = Q^{-1}$$

3. Find $\bar{\{A\}}, \bar{\{B\}}, \bar{\{C\}}$ with the following formulae:

$$\bar{A} = PAP^{-1}$$
$$\bar{B} = PB$$
$$\bar{C} = CP^{-1}$$

4. Finally, we pick the 2x2 top left of $\bar{\{A\}}$ (2x2 because A is a square matrix and the rank of our original controllability matrix was 2) and name it $ A_c $. We pick 2x1 of the $\bar{\{B\}}$ and name it $ B_c $. And we pick 1x2 of the $\bar{\{C\}}$ and name it $ C_c $

These new matrices that we found, $ A_c $, $ B_c $, $ C_c $, make up the new representation of the system that is controllable. Now that we've got the theory down, let's code it up and see the results.

$$\dot{x} = A_c x(t) + B_c u(t)$$
$$y = C_c x(t)$$

In the following code, we will make a function that will calculate the nullspace given some vectors in a matrix. Then, the function will append the nullspace of the matrix to the matrix. We also have another function that performs the kalman decomposition on matrix A and B and returns the controllable system matrices Ac, Bc, and Cc.

[15]:
```
# This function find the nullspace of the passed in matrix and adds the␣
 ↪nullspace to the matrix.
# This new matrix is now invertible.
def fill_square_with_identity(A):
```

```
    row, col = sym.shape(A)

    if(row == col):
        return A

    else:
        if( col < row ):
            tempA = A
            for i, c in enumerate(A.T.nullspace()):
                A = A.col_insert(tempA.rank()+i, c)

            return A

        else:
           print("something went wrong")
           return False
```

[16]:
```
# This function performs the Kalman Decomposition using the passed in matrices.
# It returns the controllable matrices, namely Ac, Bc and Cc.
def kalman_decomposition(A, B, controllability_matrix):

    rank = controllability_matrix.rank()
    # assuming that rank is atleast 1
    Q = controllability_matrix.col(0)

    for i in range(1, rank):
        Q = Q.col_insert(i, controllability_matrix.col(i))

    Q = fill_square_with_identity(Q)

    if( not(Q) ):
        return False

    P = Q.inv()
    A_bar = P*A*P.inv()
    B_bar = P*B
    C_bar = C*P.inv()

    Ac = sym.Matrix([A_bar[:rank, :rank]])
    Bc = sym.Matrix([B_bar[:rank]]).T
    Cc = sym.Matrix([C_bar[:rank]])

    if(not(controllable(Ac, Bc, rank))):
        print("Kalman Decomposition Failed.")
        return False

    return Ac, Bc, Cc
```

Now that we have our helper functions, we can use the kalman decomposition on the uncontrollable system to get a controllable system.

[17]:
```
Controllable_SS = kalman_decomposition(new_A, new_B, controllable(new_A,␣
↪new_B)[1])

if((Controllable_SS)):
    Ac = Controllable_SS[0]
    Bc = Controllable_SS[1]
    Cc = Controllable_SS[2]
else:
    print("Kalman Decomposition Failed")
```

5.4 Results:

We can see that the Kalman Decomposition function did not fail. The following matrices are the results of the Kalman Decomposition:

[18]:
```
print("Ac:")
Ac
```

Ac:

[18]:
$$\begin{bmatrix} 0 & 0 \\ 0.999755859375 & 0 \end{bmatrix}$$

[19]:
```
print("Bc:")
Bc
```

Bc:

[19]:
$$\begin{bmatrix} 1.01020408163265 \\ 0 \end{bmatrix}$$

[20]:
```
print("Cc:")
Cc
```

Cc:

[20]:
$$\begin{bmatrix} 0 & 0.1 \end{bmatrix}$$

We can now check if the newly formed system is controllable.

[21]:
```
# Checking the controllability of the newly generated system
controllable, controllability_matrix = controllable(Ac, Bc, 2)

print("Is the newly generated system controllable?", controllable)

controllability_matrix
```

```
Is the newly generated system controllable? True
```

[21]:
$$\begin{bmatrix} 1.01 & 0 \\ 0 & 1.01 \end{bmatrix}$$

5.5 Checking the Results

Finally, we can now check the results by comparing the transfer function generated by our original uncontrollable state space system with the transfer function of the new controllable state space system.

[22]:
```
print("uncontrollable system:", ss2tf(new_A, new_B, new_C, sym.Matrix([[0]])))
print("controllable system:", ss2tf(Ac, Bc, Cc, sym.Matrix([[0]])))
```

```
uncontrollable system:
     0.101 s^2 - 5.36e-17 s - 0.0001
----------------------------------------
s^4 - 1.001 s^2 + 3.914e-16 s - 2.441e-17

controllable system:
0.101
-----
 s^2
```

5.6 Conclusion for Part II

We can verify that the above transfer functions are the not quite exact, but almost exact. All the additive values in the uncontrollable matrix are very small, 10^{-17} or 10^{-16}.

We can confirm that we have found the controllable form out of the uncontrollable form for the system using the Kalman Decomposition.

6 Part III

Set the parameters to create an "unstable" system. Use "constant gain negative state-feedback" to make the system stable.

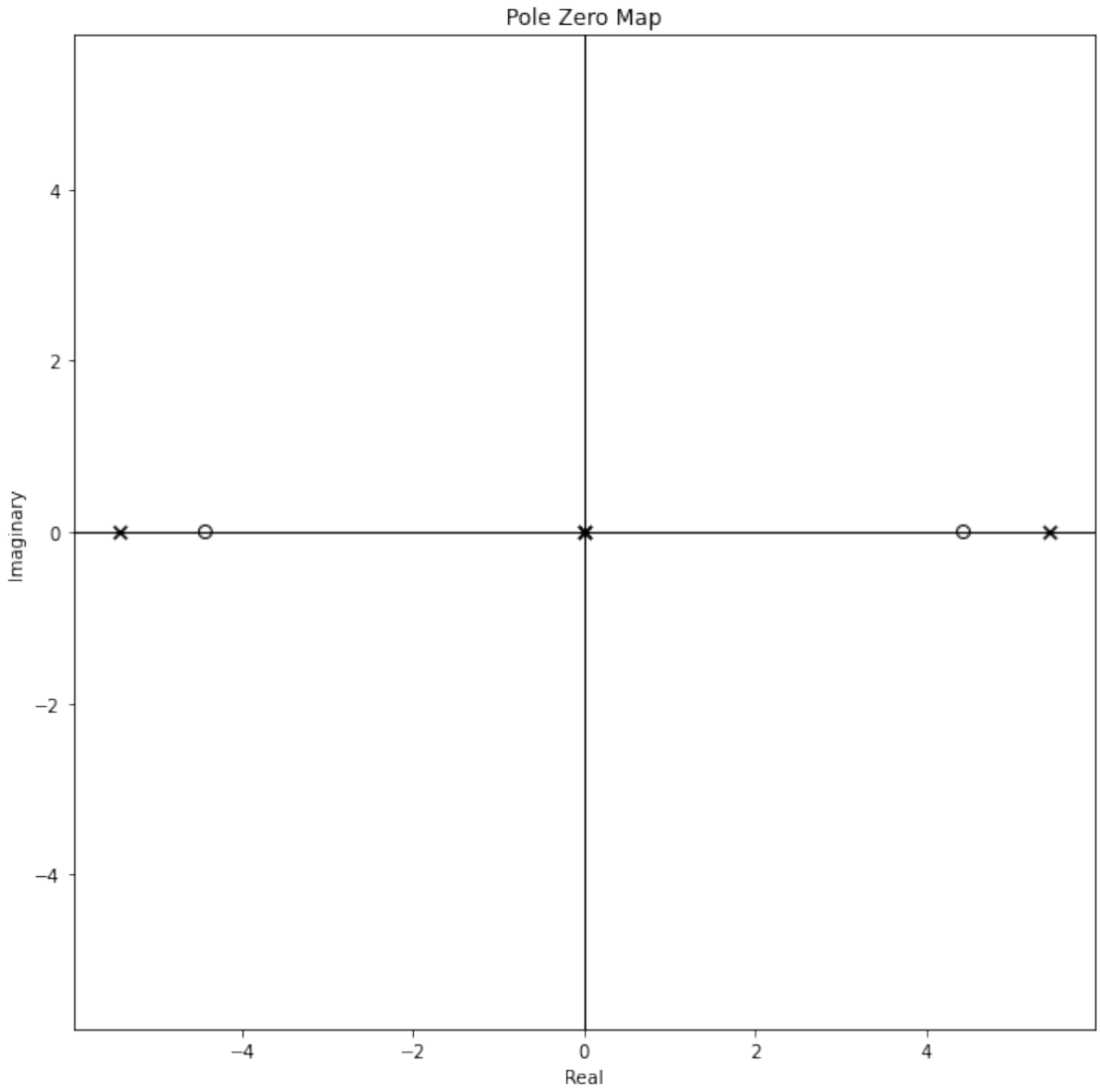
We will use the same parameters as we used in the first part to check for stability and generate the needed gains. The following code will apply values to the parameter variables and then define the matrices from the system.

[23]:
```
m = 1      # mass of the pendulum.
M = 2      # mass of the cart.
g = 9.81   # gravity.
l = 0.5    # length of the pendulum.
```

[24]:
```
A = np.array([[0,1,0,0],[0,0,-(m*g)/M,0],[0,0,0,1],[0,0,(M+m)*g/(M*l),0]])
B = np.array([[0],[1/M],[0],[-1/(M*l)]])
C = np.array([[1,0,0,0],[0,0,1,0]])
D = 0
```

6.1 Eigen values of the system

[25]:
```
sys_y = ss(A,B,C[0:1],D)
r = pzmap(sys_y,xlim=[-10,10],ylim=[-20,20])
```



As can be seen, there are poles on the left half plane. Thus the system is unstable.

6.2 Pole Placement using Matlab

Let us arbritrarily pick 4 poles such that the system can be made stable for some forward gain. Let the chosen poles be at (-5Âś0.5j) and (-6Âś1j). These poles are fast enough to get us a settling time of < 0.5s

[26]:
```
poles = [-5+0.5j,-5-0.5j,-6-1j,-6+1j]
k = place(A,B,poles)
print(k)
```

```
[[ -95.23445464  -68.60346585 -259.29722732  -56.30173293]]
```

6.3 Pole Placement using Ackermann's formula

Ackermann's formula can be used to easily place poles. The formula is given by:

$$k = [0\ 0\ 0\ ...\ 1]\mathcal{C}_o^{-1}\Delta_d(A)$$

Where, Δ_d is the characteristic equation obtained from the desired eigen values

[27]:
```
# Calculate same gains using Ackermann's formula
det_new = np.identity(4)
for root in poles:
    det_new = np.matmul(det_new,(A-root*np.identity(4)))

k_ack = np.matmul(np.matmul(np.array([0,0,0,1]),np.linalg.inv(Co)),det_new)
k_ack = np.real(k_ack.reshape((1,4)))
print(k_ack)
```

```
[[ -95.23445464  -68.60346585 -259.29722732  -56.30173293]]
```

7 Part IV

We can plot the step responses of the the system's outputs, the tilt angle, θ and the cart's displacement, y.

7.1 Step response

The state space equations for the new system are given by:

$$\dot{x} = (A - Bk)x + Br$$

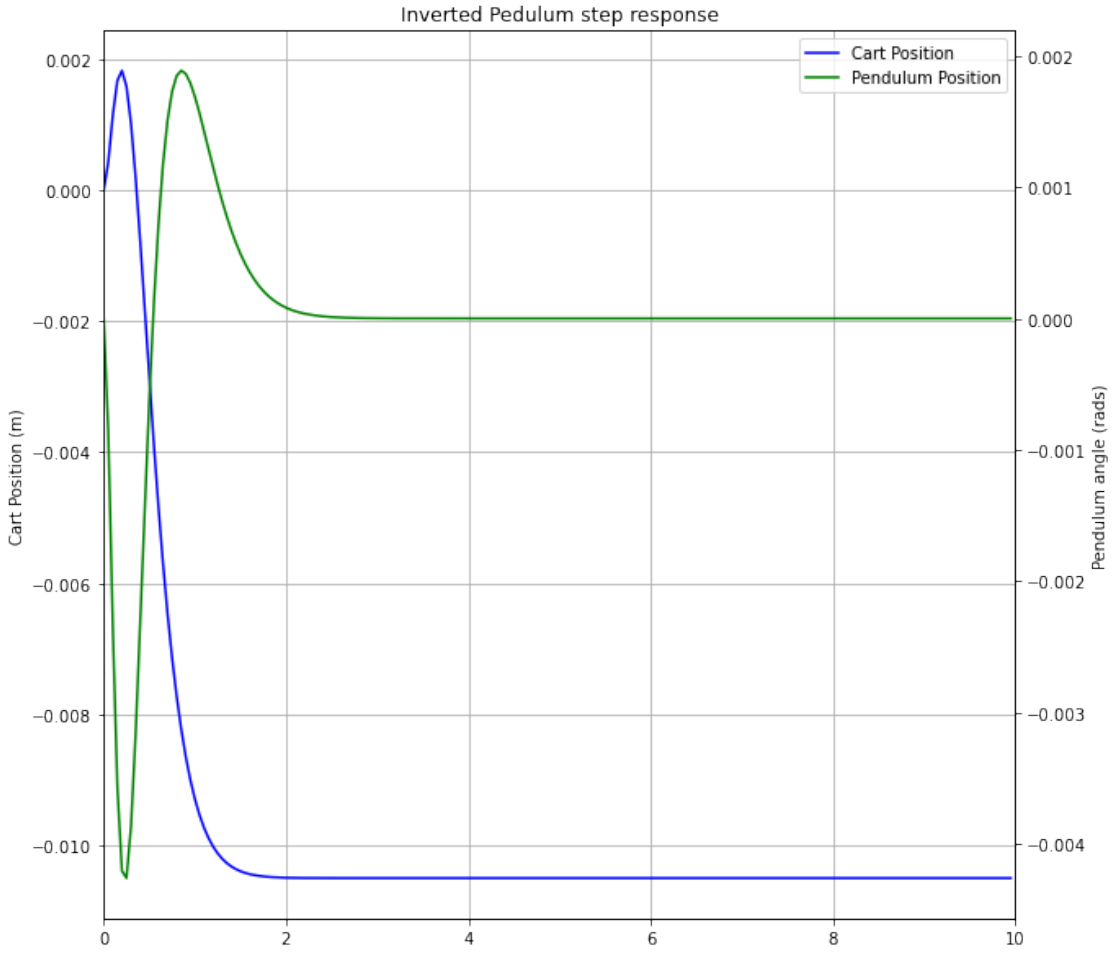
$$y = Cx + Du$$

[28]:
```

```

```
def plot_step():
    T = np.arange(0,10,0.05)     # Time range against which the step response is␣
 ↪shown
    fig, ax1 = plt.subplots()
    ax2 = ax1.twinx()            # Get plot axes

    y,T=step(sysfb_y,T)          # Generate step response of the displacement
    p = ax1.plot(T,y,'b-')

    y,T = step(sysfb_angle,T)    # Generate step response of the tilt angle
    p = ax2.plot(T,y,'g-')

    # Make the plot look pretty
    e = ax1.grid()
    e = plt.xlim([0,10])
    e = ax1.set_ylabel('Cart Position (m)')
    e = ax2.set_ylabel('Pendulum angle (rads)')
    e = plt.xlabel('Time (s)')
    e = plt.title('Inverted Pedulum step response')
    e = fig.legend(['Cart Position','Pendulum Position'],loc="upper right",␣
 ↪bbox_to_anchor=(1,1), bbox_transform=ax1.transAxes)

    plt.show()
```

[29]:
```
Afb = (A-np.matmul(B,k_ack))  # Define the new A matrix

sysfb_y = ss(Afb,B,C[0:1],D)  # Setup the new statespace equation with 1 column␣
 ↪of the C matrix
sysfb_angle = ss(Afb,B,C[1:],D) # Setup the new statespace equation with the 2nd␣
 ↪column of the C matrix
```

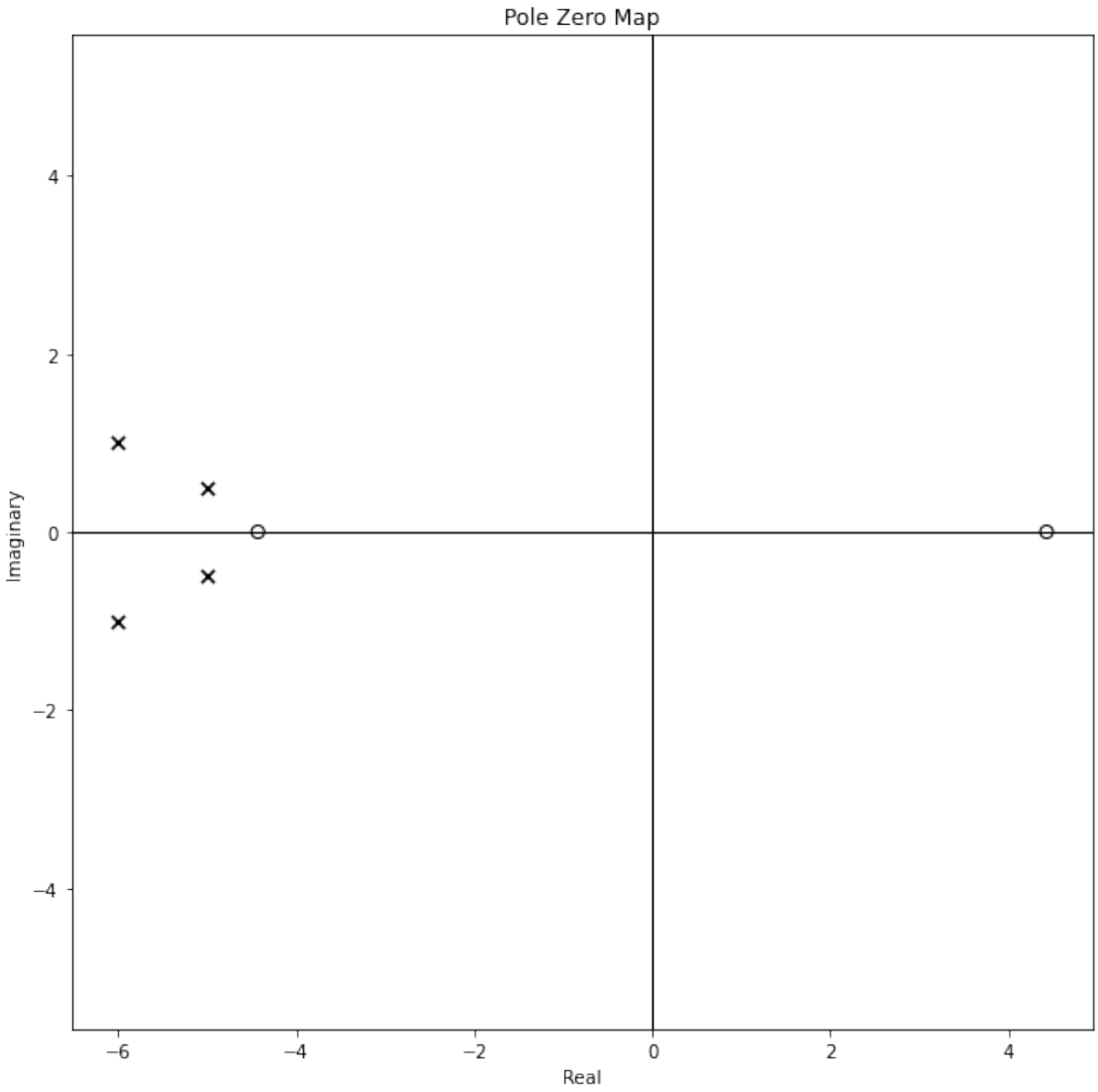
[30]:
```
plot_step()
```



7.2 Pole-Zero map of the desired system

We can have a look at the pole zero map again to make sure that the poles ended up where we wanted them. Now the system is stable since all the eigen values lie in the left half plane

[32]:
```
fig = plt.figure()
axz  = plt.axes()
r,k = pzmap(sysfb_y,xlim=[-10,5],ylim=[-10,10],ax=ax)
```



8 Conclusions

The inverted pendulum has been studied extensively in the field of controls. The system is always controllable as long as the values for the parameters are chosen sensibly. For example the mass of the pendulum can't be greater than the mass of the cart itself or physical constants can't be assumed to have values other than are agreed to be accurate by the scientific community.

Observability of the the system depends on the output equation. If output is chosen to be either the angle of the pendulum from vertical, the rate of change of the angle or the velocity of the cart, the system becomes unobservable. On the other hand if the output is chosen to be the distance the cart has moved, the system is then controllable.

To make the system unobservable with the given state space equation. We can choose the parameters

of mass of the cart or pendulum or the length of the pendulum to be absurdly large. This makes the determinant of the square controllability matrix tend towards 0. This in turn makes the the Controllability and Observability matrix not have full rank.

The inverted pendulum system is inherently unstable. The mass at the end of the pendulum will not stay upright on its own accord. This is supposrted by the right half plane pole shown in Fig 2. Thus to balance the mass vertically a feedback system must be used. Feedback gains can be chosen to place the poles on the left half plane so as to achive the requred dynamics. To calculate the feedback gains, the `place` command can be used in Matlab or the Ackermann's formula can also be used. The resulting system is now stable. But in order to use these gais we would need to

9 References

[1] C.-T. Chen, "Mathematical descriptions of Systems," in Linear System Theory and design, New York, NY: Oxford university press, 2014, pp. 33–35.

[2] "Inverted pendulum: System modeling," Control Tutorials for MATLAB and Simulink - Inverted Pendulum: System Modeling. [Online]. Available: https://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulumÂğion=SystemModeling. [Accessed: 24-Apr-2022].