CDS110 **October 29, 2024**

Pset 04

Problem 1

Part A

$$det(A - \lambda I) = det\left(\begin{bmatrix} -\lambda & 1 \\ 1 & -\lambda \end{bmatrix}\right) = \lambda^2 - 1 = 0$$

From above we see that the eigenvalue are $\lambda_1 = 1$ and $\lambda_2 = -1$. Because we have a positive eigenvalue $\lambda_1 = 1$, then the system is unstable.

Part B

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

$$W_r = \begin{bmatrix} B & AB \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$det(W_r) = 0 - 1 = -1$$

As shown above we have that the determinant of W_r is non-zero. For a 2 by 2 matrix (which is what W_r is), this means that it is invertible. Therefore, because W_r is invertible then the system is reachable. Therefore, we can always find an input to bring the system from an initial state x_0 to an equilibrium point x_e. Hence the closed loop system can be made asymptotically stable by choosing/designing an appropriate input u.

Part C

Substituting $u = -Kx + k_f r$, we get

$$\dot{x} = Ax + B(-Kx + k_f r) = (A - BK)x + Bk_f r$$

The closed loop system has the characteristic polynomial:

$$sI - A + BK = s\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ k_1 & k_2 \end{bmatrix} = \begin{bmatrix} s & -1 \\ k_1 - 1 & k_2 + s \end{bmatrix}$$

$$det(sI - A + BK) = det\left(\begin{bmatrix} s & -1 \\ k_1 - 1 & k_2 + s \end{bmatrix}\right) = s(k_2 + s) + (k_1 - 1) = s^2 + k_2 s + (k_1 - 1)$$

$$\lambda(s) = s^2 + k_2 s + (k_1 - 1)$$

Solving for k_1 and k_2 by setting $\lambda(s)$ to our desired polynomial:

$$s^2 + k_2 s + (k_1 - 1) = s^2 + 2\zeta_0\omega_0 s + \omega_0^2$$

The equation above gives us the following:

$$k_2 = 2\zeta_0\omega_0$$

$$k_1 - 1 = \omega_0^2 \implies k_1 = \omega_0^2 + 1$$

Thus we have $k_1 = \omega_0^2 + 1$ and $k_2 = 2\zeta_0\omega_0$

Part D

$$\dot{x} = (A - BK)x + Bk_f r$$

At the equilibrium point x_e we have:

$$(A - BK)x_e + Bk_f r = 0$$

$$x_e = -(A - BK)^{-1}Bk_f r$$

Now at equilibrium point y_e we have:

$$y_e = Cx_e = r$$

Using the equation for x_e we have:

$$Cx_e = -(A - BK)^{-1}Bk_f r = r$$

The above equation simplifies to:

$$-C(A - BK)^{-1}Bk_f = 1 \implies k_f = \frac{-1}{C(A - BK)^{-1}B}$$

Now solving for the denominator in the equation for k_f:

$$A - BK = \begin{bmatrix} 0 & 1 \\ 1 - k_1 & -k_2 \end{bmatrix}$$

$$(A - BK)^{-1} = \frac{1}{det(A - BK)} \begin{bmatrix} -k_2 & -1 \\ k_1 - 1 & 0 \end{bmatrix} = \frac{1}{k_1 - 1} \begin{bmatrix} -k_2 & -1 \\ k_1 - 1 & 0 \end{bmatrix}$$

$$C(A - BK)^{-1}B = \frac{1}{k_1 - 1} \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} -k_2 & -1 \\ k_1 - 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Substituting for k_1 and k_2, we have:

$$C(A - BK)^{-1}B = \frac{1}{\omega_0^2} \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} -2\zeta_0\omega_0 & -1 \\ \omega_0^2 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{-1}{\omega_0^2}$$

Therefore we have:

$$k_f = \frac{-1}{\frac{-1}{\omega_0^2}} = \omega_0^2$$

Part E

From the characteristic polynomial $s^2 + 2\zeta_0\omega_0 s + \omega_0^2$, we have the eigenvalues: $s = \frac{-2\zeta_0\omega_0 \pm \sqrt{4\zeta_0^2\omega_0^2 - \omega_0^2}}{2}$. From this we see that the closed loop system, is only asymptotically stable when $\zeta_0\omega_0 > 0$ as this will yield eigenvalues negative real parts (i.e. all eigenvalues will have a negative real part). If $\zeta_0\omega_0 < 0$, then in any case for the value of $4\zeta_0^2\omega_0^2 - \omega_0^2$, there will always be at least one eigenvalue with a positive real part. If $\zeta_0\omega_0 = 0$, then this will just provide us with eigenvalues with their real part equal to 0, which would just give us a marginally stable system and not an asymptotically stable system.

Problem 2

Part A
As given in the problem we have:

$$M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + G(\theta) = \begin{bmatrix}0\\ \tau\end{bmatrix}$$

Solving for $\ddot{\theta}$

$$M(\theta)\ddot{\theta} = \begin{bmatrix}0\\ \tau\end{bmatrix} - G(\theta) - C(\theta,\dot{\theta})\dot{\theta}$$

$$\ddot{\theta} = (M(\theta))^{-1}\left(\begin{bmatrix}0\\ \tau\end{bmatrix} - G(\theta) - C(\theta,\dot{\theta})\dot{\theta}\right)$$

$$\begin{bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix} = \left(M\left(\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}\right)\right)^{-1}\left(\begin{bmatrix}0\\ \tau\end{bmatrix} - G\left(\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}\right) - C\left(\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}, \begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix}\right)\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix}\right)$$

Using the definition of x and the equation for $\ddot{\theta}$ we have:

$$x = \begin{bmatrix}\theta\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\theta_1\\ \theta_2\\ \dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} = \begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix}$$

$$\dot{x} = \begin{bmatrix}\dot{\theta}\\ \ddot{\theta}\end{bmatrix} = \begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\\ \ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix} = \begin{bmatrix}x_3\\ x_4\\ \left(M\left(\begin{bmatrix}x_1\\ x_2\end{bmatrix}\right)\right)^{-1}\left(\begin{bmatrix}0\\ \tau\end{bmatrix} - G\left(\begin{bmatrix}x_1\\ x_2\end{bmatrix}\right) - C\left(\begin{bmatrix}x_1\\ x_2\end{bmatrix}, \begin{bmatrix}x_3\\ x_4\end{bmatrix}\right)\begin{bmatrix}x_3\\ x_4\end{bmatrix}\right)\end{bmatrix}$$

$$\dot{x} = \begin{bmatrix}x_3\\ x_4\\ \left(M\left(\begin{bmatrix}x_1\\ x_2\end{bmatrix}\right)\right)^{-1}\left(-G\left(\begin{bmatrix}x_1\\ x_2\end{bmatrix}\right) - C\left(\begin{bmatrix}x_1\\ x_2\end{bmatrix}, \begin{bmatrix}x_3\\ x_4\end{bmatrix}\right)\begin{bmatrix}x_3\\ x_4\end{bmatrix}\right)\end{bmatrix} + \begin{bmatrix}0\\ 0\\ \left(M\left(\begin{bmatrix}x_1\\ x_2\end{bmatrix}\right)\right)^{-1}\left(\begin{bmatrix}0\\ 1\end{bmatrix}\right)\end{bmatrix}\tau = f(x)+g(x)\tau$$

Setting $u = \tau$ we have $\dot{x} = f(x) + g(x)u$ where $f(x)$ and $g(x)$ are defined as in the equation above. Note that the matrix $M(\theta)$ is invertible for any point (θ_1, θ_2). This is because its determinant $ab - c^2cos^2(\theta_2)$ will always be nonzero, and so the 2 by 2 matrix $M(\theta)$ will always have an inverse. To see this set $ab - c^2cos^2(\theta_2) = 0$, which yields $cos^2(\theta_2) = \frac{ab}{c^2}$. Using the equation for a,b, and c we have $a = 96$, $b = 64$, and $c = 64$, thus the equation becomes $cos^2(\theta_2) = \frac{96 \cdot 64}{64^2} = 1.50$. However, the maximum value of $cos^2(\theta_2)$ is 1.0, therefore there does not exist a value for (θ_1, θ_2) such that the determinant of $M(\theta)$ is zero. Hence $M(\theta)$ will be invertible for any θ with the given values of the parameters. Thus the system can always be written in control-affine form $\dot{x} = f(x) + g(x)u$ as shown above.

Part B

At equilibrium we have $\dot{x} = 0$, which means $\dot{\theta} = 0$ and $\ddot{\theta} = 0$. Using the form of the system as given in the problem we have:

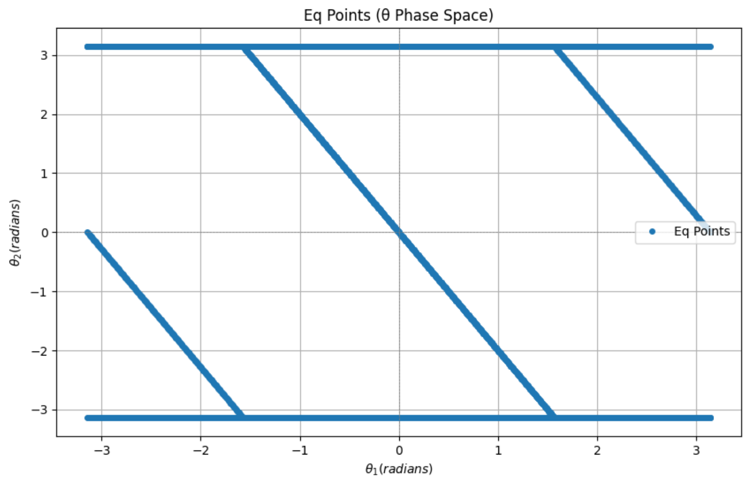
$$M(\theta)0 + C(\theta, 0)0 + G(\theta) = \begin{bmatrix} 0 \\ \tau \end{bmatrix} \implies G(\theta) = \begin{bmatrix} 0 \\ \tau \end{bmatrix}$$

Using the definition of $G(\theta)$, and setting $\tau = \tau_0$, this gives us two equations:

$$-d\ sin(\theta_1) - e\ sin(\theta_1 + \theta_2) = 0$$

$$-e\ sin(\theta_1 + \theta_2) = \tau_0$$

From the equations above we have $-d\ sin(\theta_1) = -\tau_0 \implies sin(\theta_1) = \frac{\tau_0}{d}$. Therefore the points (θ_1, θ_2) that satisfy $sin(\theta_1) = \frac{\tau_0}{d}$ and $sin(\theta_1 + \theta_2) = \frac{-\tau_0}{e}$ are equilibrium points. Below is a plot of points in the θ Phase Space, and using values of τ_0 that yield solutions to the equations above.



Part B Code

```
# Part B
import numpy as np
import matplotlib.pyplot as plt
# Given parameters
l1 = 0.5
l2 = 1
m1 = 8
m2 = 8
g = 10
d = g * m1 * l1 + g * m2 * l1
e = g * m2 * l2

# range of tau values that yield a result for theta1 and theta2
angle_vals = np.pi * np.linspace(-1, 1, 1001)
tau_vals = []

theta1_vals = []
```

```
theta2_vals = []

# Compute eq points
for angle in angle_vals:
    # Calculate theta1 and theta2
    theta1 = angle
    tau = d * np.sin(theta1)
    theta2 = np.arcsin(-tau/e) - theta1

    theta1_vals.append(theta1)
    theta2_vals.append(theta2)
    tau_vals.append(tau)

    theta1_vals.append(-theta1)
    theta2_vals.append(-theta2)
    tau_vals.append(tau)

    theta2 = np.arcsin(tau/e) - theta1 - np.pi
    if theta2 <= np.pi and theta2 >= -np.pi:
      theta1_vals.append(theta1)
      theta2_vals.append(theta2)
      tau_vals.append(tau)

      theta1_vals.append(-theta1)
      theta2_vals.append(-theta2)
      tau_vals.append(tau)

    # at this point the center of mass will be on the pivot joint
    theta1_vals.append(theta1)
    theta2_vals.append(-np.pi)
    tau_vals.append(tau)

    theta1_vals.append(-theta1)
    theta2_vals.append(np.pi)
    tau_vals.append(tau)

# Convert lists to numpy arrays
theta1_eq = np.array(theta1_vals)
theta2_eq = np.array(theta2_vals)

# Plot theta phase plane
plt.figure(figsize=(10, 6))
plt.plot(theta1_eq, theta2_eq, 'o', markersize=4, label='Eq Points')
plt.title('Eq Points (Theta Phase Space)')
plt.xlabel("theta_{1} (radians)")
plt.ylabel("theta_{2} (radians)")
plt.axhline(0, color='gray', lw=0.5, linestyle='--')
plt.axvline(0, color='gray', lw=0.5, linestyle='--')
plt.grid()
plt.legend()
plt.show()
```

Part C
Linearizing $\dot{x} = f(x) + g(x)u$ at (x_0, u_0) we have:

$$\dot{x} \approx \frac{\partial f}{\partial x}\bigg|_{(x_0,u_0)}(x - x_0) + \frac{\partial g}{\partial x}u\bigg|_{(x_0,u_0)}(x - x_0) + \frac{\partial u}{\partial u}g(x)\bigg|_{(x_0,u_0)}(u - u_0)$$

$$\dot{x} \approx \frac{\partial f}{\partial x}\bigg|_{(x_0,u_0)}(x - x_0) + \frac{\partial g}{\partial x}\bigg|_{(x_0,u_0)}u_0(x - x_0) + g(x_0)(u - u_0)$$

At the equilibrium point $x_0 = (\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0, 0, 0, 0)$ we have $u_0 = 0$. Therefore, for this problem we the linearization becomes:

$$\dot{x} \approx \frac{\partial f}{\partial x}\bigg|_{(x_0,u_0)}(x - x_0) + g(x_0)(u - u_0)$$

Using the control affine form from Part A, and the SymPy library from python we get the following lineazrized system:

$$\dot{x} = Ax + Bu$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 20 & -20 & 0 & 0 \\ -20 & 40 & 0 & 0 \end{bmatrix}$$

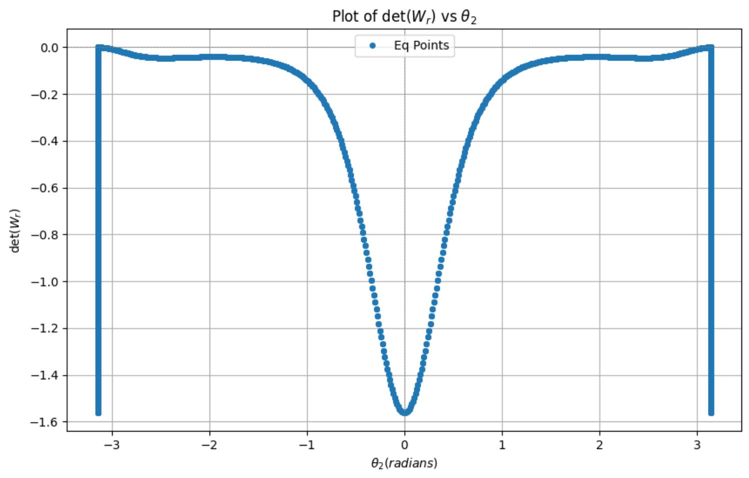
$$B = \begin{bmatrix} 0 \\ 0 \\ -0.75 \\ 1.25 \end{bmatrix}$$

Using numpy, we get that the eigenvalues of matrix A are $\lambda_1 = -7.23606798$, $\lambda_2 = -2.76393202$, $\lambda_3 = 7.23606798$, and $\lambda_4 = 2.76393202$. Therefore the system is usntable as there is an eigenvalue with a positive real part. Using numpy, the rank of W_r (the reachability matrix) is 4 and its determinant is -1.56. In other words, W_r is full rank which means that it is invertible. Hence, the linearized system is controllable. Note: Code is attached to this problem.
Part C Code

```
# Part C
import sympy as sp

# Setup variables
# Note: x1=theta1, x2=theta2, x3=dtheta1, and x4 = dtheta4
theta1, theta2, dtheta1, dtheta2 = sp.symbols('theta1 theta2 dtheta1 dtheta2')

# Other parameter values
a = m1 * l1**2 + m2 * l1**2
b = m2 * l2**2
c = m2 * l1 * l2

# setup up the matrices
M = sp.Matrix([[a + b + 2*c*sp.cos(theta2), b + c*sp.cos(theta2)],
               [b + c*sp.cos(theta2), b]])

C = sp.Matrix([[-c*sp.sin(theta2)*dtheta2, -c*sp.sin(theta2)*(dtheta1 + dtheta2)],
               [c*sp.sin(theta2)*dtheta1, 0]])
```

```
G = sp.Matrix([[-d*sp.sin(theta1) - e*sp.sin(theta1 + theta2)],
               [-e*sp.sin(theta1 + theta2)]])

dtheta = sp.Matrix([dtheta1, dtheta2])

# We define f_x = f(x)
f_x = M.inv() * (-G - C * dtheta)
f_x = f_x.row_insert(0, sp.Matrix([[dtheta2]]))
f_x = f_x.row_insert(0, sp.Matrix([[dtheta1]]))

# Compute the Jacobian of f_x with respect to the variables we set up
x = sp.Matrix([theta1, theta2, dtheta1, dtheta2])
J_f = f_x.jacobian(x)
J_f.simplify()

# now we evaluate at (0, 0). Note at equilibrium we also have dtheta1=dtheta2=0
point = {'theta1' : 0, 'theta2': 0, 'dtheta1':0, 'dtheta2': 0}
J_eval = J_f.subs(point)
#print("J_eval = {}".format(J_eval))

# now we find g(x) at x=x_0 = (0,0,0,0)
control = sp.Matrix([0, 1])
g_x = M.inv() * control
g_x = g_x.row_insert(0, sp.Matrix([[0]]))
g_x = g_x.row_insert(0, sp.Matrix([[0]]))
g_eval = g_x.subs(point)
#print(g_eval)

# we convert the Jacobian of f and g(x0) to numpy arrays
A = np.array(J_eval).astype(np.float64)
B = np.array(g_eval).astype(np.float64)
print("A = {}".format(A))
print("B = {}".format(B))

# Compute Eigenvalues to test for stability
eigenvals, eigenvecs = np.linalg.eig(A)
print("eigenvalues of A: {}".format(eigenvals))

# compute reachability matrix W_r
#W_r = sp.Matrix.hstack(B, A*B, A**2*B, A**3*B)
W_r = np.hstack([B, A @ B, np.linalg.matrix_power(A, 2) @ B, np.linalg.matrix_power(A, 3) @
det_Wr = np.linalg.det(W_r)
print("W_r = {}".format(W_r))
print("det(W_r) = {}".format(det_Wr))
print("Rank of W_r = {}".format(np.linalg.matrix_rank(W_r)))
```

Part D
Using numpy, a plot of the rank of the reachability matrix W_r vs θ_2 is shown below. To linearized, the set of (θ_1, θ_2) along with their corresponding τ_0 were used. As we see from the plot below, the linearized system appears to always be controllable except at the ends where $\theta_2 = pi$ and $\theta_2 = -pi$. Intuitively, this makes sense. If we start at $\theta_2 = -pi$ or $\theta_2 = pi$ we essentially have l_2 overlapping l_1 such that the center of mass is the anchor joint. Without a control input, if our initial point is at $\theta_2 = \pm pi$ then we would be stuck at equilibrium since the center of mass is at the anchor joint, the rigid links would not move (i.e they will still be in the configuration as determined by the initial point). To put it simply, we would essentially be losing a degree a freedom. (Note: Code is attached to this problem for reference).



Part D Code

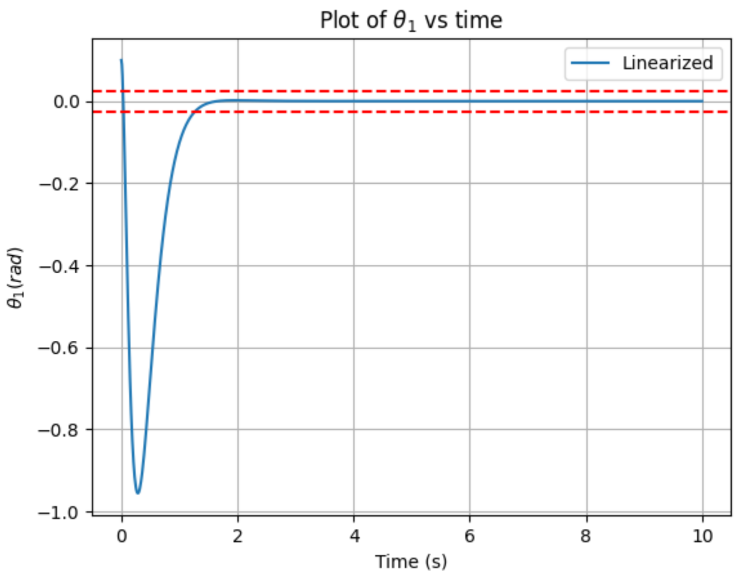
```
# Find Jacobian of g
theta1, theta2, dtheta1, dtheta2 = sp.symbols('theta1 theta2 dtheta1 dtheta2')
g_x = M.inv() * control
g_x = g_x.row_insert(0, sp.Matrix([[0]]))
g_x = g_x.row_insert(0, sp.Matrix([[0]]))

x = sp.Matrix([theta1, theta2, dtheta1, dtheta2])
J_g = g_x.jacobian(x)
J_g.simplify()


det_vals = []
rank_vals = []
for i in range(len(tau_vals)):
  #tau = tau_vals[i]
  theta1 = theta1_vals[i]
  theta2 = theta2_vals[i]
  tau = d * np.sin(theta1)

  point = {'theta1' : theta1, 'theta2': theta2, 'dtheta1':0, 'dtheta2': 0}
  Jf_eval = J_f.subs(point)
  Jg_eval = J_g.subs(point)
  g_eval = g_x.subs(point)
```

```
    A = np.array(Jf_eval).astype(np.float64) + tau * np.array(Jg_eval).astype(np.float64)
    B = np.array(g_eval).astype(np.float64)

    W_r = np.hstack([B, A @ B, np.linalg.matrix_power(A, 2) @ B, np.linalg.matrix_power(A, 3
    det_Wr = np.linalg.det(W_r)

    det_vals.append(det_Wr)
    rank_vals.append(np.linalg.matrix_rank(W_r))

print(det_vals)
print(rank_vals)

rank_vals_np = np.array(rank_vals)
plt.figure(figsize=(10, 6))
plt.plot(theta2_eq, rank_vals_np, 'o', markersize=4, label='Eq Points')
plt.title('Plot of the rank of W_r vs theta_{2}')
plt.xlabel("theta_{2} (radians)")
plt.ylabel("rank($W_r$)")
plt.axhline(0, color='gray', lw=0.5, linestyle='--')
plt.axvline(0, color='gray', lw=0.5, linestyle='--')
plt.grid()
plt.legend()
plt.show()

det_vals_np = np.array(det_vals)
plt.figure(figsize=(10, 6))
plt.plot(theta2_eq, det_vals_np, 'o', markersize=4, label='Eq Points')
plt.title('Plot of det(W_r) vs theta_{2}')
plt.xlabel("theta_{2} (radians)")
plt.ylabel("det($W_r$)")
plt.axhline(0, color='gray', lw=0.5, linestyle='--')
plt.axvline(0, color='gray', lw=0.5, linestyle='--')
plt.grid()
plt.legend()
plt.show()
```

Part E

Using the initial point $(0.1, 0)$ we will linearize about $(0, 0)$. From Part C we already know the corresponding linearization about $(0, 0)$. Using the matrices for A and B of the linearized system $\dot{x} = Ax + Bu$ from part C, we perform pole placement with SciPy (python). As you can see from the plot, the 5% settling time for θ_1 is less than 2 seconds and $|\theta_1| < 1\ rad$ before settling. The code is attached to the problem. From the code we get the following vector for K

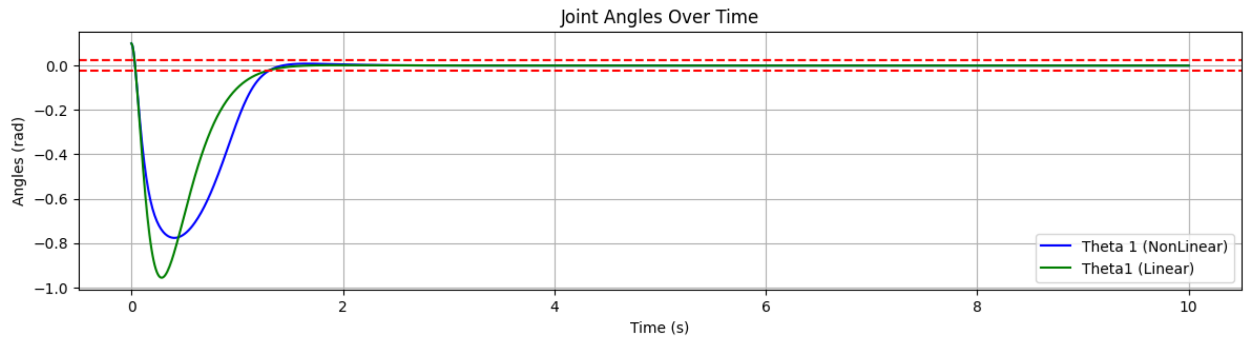
$$K = \begin{bmatrix} -2993.6 & -1558.32 & -1050.2 & -608.44 \end{bmatrix}$$



Part E Code

```
# Part E
# for setting time,want to be 95% away from our intial condition
# greater the magnitude in Re(lambda), the less the settling time is
#
from scipy.signal import place_poles

# we get A and B As we did in part C

point = {'theta1' : 0, 'theta2': 0, 'dtheta1':0, 'dtheta2': 0}
J_eval = J_f.subs(point)

control = sp.Matrix([0, 1])
g_x = M.inv() * control
g_x = g_x.row_insert(0, sp.Matrix([[0]]))
g_x = g_x.row_insert(0, sp.Matrix([[0]]))
g_eval = g_x.subs(point)

A = np.array(J_eval).astype(np.float64)
B = np.array(g_eval).astype(np.float64)
print("A = {}".format(A))
print("B = {}".format(B))

# pole placement
desired_poles = [-14.1, -3, -4, -6]
result = place_poles(A, B, desired_poles)
K = result.gain_matrix

print("Feedback Gain K:", K)
from scipy.integrate import odeint

def linearized_dynamics(x, t):
    dxdt = np.dot(A - np.dot(B, K), x)  # Replace with your control input logic
    return dxdt

xi = [0.1, 0, 0, 0]
t = np.linspace(0, 10, 1000)

solution = odeint(linearized_dynamics, xi, t)
theta1 = solution[:, 0]
theta2 = solution[:, 1]

# Plot results
plt.figure()
plt.plot(t, theta1, label='Linearized')
#plt.plot(t, theta2, label='theta2 (rad)')
plt.axhline(0.025, color='red', linestyle='--')
plt.axhline(-0.025, color='red', linestyle='--')
plt.title('Plot of theta_1 vs time')
plt.xlabel('Time (s)')
plt.ylabel('theta_1 (rad)')
plt.legend()
plt.grid()
plt.show()

# Final value after simulation
y_final_theta1 = theta1[-1]
y_final_theta2 = theta2[-1]
print("Final theta1:", y_final_theta1)
print("Final theta2:", y_final_theta2)
```

Part F
Yes, as shown below, the system still meets the specifications.



Part F Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint


# Functions to compute Matrices given the value for theta and theta_dor
def M(theta):
    theta1, theta2 = theta
    return np.array([[a + b + 2 * c * np.cos(theta2), b + c * np.cos(theta2)],
                     [b + c * np.cos(theta2), b]])
def C(theta, theta_dot):
    theta1, theta2 = theta
    dtheta1, dtheta2 = theta_dot
    return np.array([[-c * np.sin(theta2) * dtheta2, -c * np.sin(theta2) * (dtheta1 + dthe
                     [c * np.sin(theta2) * dtheta1, 0]])
def G(theta):
    theta1, theta2 = theta
    return np.array([[-d * np.sin(theta1) - e * np.sin(theta1 + theta2)],
                     [-e * np.sin(theta1 + theta2)]])

def nonlinear_dynamics(x, t):
    theta1, theta2, dtheta1, dtheta2 = x
    theta = np.array([theta1, theta2])
    theta_dot = np.array([dtheta1, dtheta2])

    u = -K @ np.array([theta1, theta2, dtheta1, dtheta2])
    u_vec = np.array([[0], [u[0]]])

    M_inv = np.linalg.inv(M(theta))
    C_eval = C(theta, theta_dot)
    G_eval = G(theta)

    theta_dot = np.array([[dtheta1], [dtheta2]])
    theta_ddot = M_inv @ (u_vec - C_eval @ theta_dot - G_eval)

    return [dtheta1, dtheta2, theta_ddot[0][0], theta_ddot[1][0]]


# intial condition
x0 = [0.1, 0, 0, 0]
t = np.linspace(0, 10, 1000)
solution = odeint(nonlinear_dynamics, x0, t)

# Extract angles and angular velocities
theta1 = solution[:, 0]
theta2 = solution[:, 1]
dot_theta1 = solution[:, 2]
dot_theta2 = solution[:, 3]

# Plot the results
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
plt.plot(t, theta1, label='Theta 1 (NonLinear)', color='blue')
plt.plot(t, linear_theta1, label='Theta1 (Linear)', color='green')
plt.axhline(0.025, color='red', linestyle='--')
plt.axhline(-0.025, color='red', linestyle='--')
plt.title('Joint Angles Over Time')
plt.xlabel('Time (s)')
plt.ylabel('Angles (rad)')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()
```

Part G

a) If the actuator has a maximum torque, limiting the control authority, we can increase the settling time to make the control less aggressive in fixing error. Using pole placement, this means increasing the real part of you desired eigenvalues (where the real part of the desired eigenvalues are negative).
b) If the control system is implemented digitally (recomputed at discrete intervals) then there may be some overshooting. One way to correct this is adding the gain k_r to the input response such that the input control becomes $u = -Kx + k_r \cdot r$
c) With an unknown torque bias, the steady state error will increase. When dealing with uncertainties that are less precise (in this case an unknown torque bias), adaptive control may prove to be useful. In this case, an integral controller may be suitable to add when dealing with an unknown torque bias.

Problem 3

Part A

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

$$W_r = \begin{bmatrix} B & AB \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$det(W_r) = 0$$

As shown above we have that the determinant of W_r is zero. For a 2 by 2 matrix (which is what W_r is), this means that the matrix is not invertible. Therefore, because W_r is not invertible then the system is not reachable.

Part B

$$sI - A + BK = s\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} k_1 & k_2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} s + k_1 & k_2 - 1 \\ 0 & s \end{bmatrix}$$

$$det(sI - A + BK) = s(s + k_1)$$

When we set the determinant to zero, we see that the eigenvalues must be $\lambda_1 = 0$ and $\lambda_2 = -k_1$. Hence the eigenvalues cannot be set to arbitrary values as one of them must always be zero and the other in terms of k_1.