2.152 Nonlinear Control Spring 2020 HW4

Rylan Schaeffer

April 14th, 2020

Problem 1

(a) Consider $\begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \end{bmatrix} \Rightarrow r = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. The total relative degree is therefore $1 + 1 = 2$.

(b) For ease, we first rewrite the system in the form given by 6.94:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}_{f(x,y,\theta)} + \underbrace{\begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix}}_{g(x,y,\theta)} \underbrace{\begin{bmatrix} v \\ \omega \end{bmatrix}}_{u} \quad \text{and} \quad \begin{bmatrix} x \\ y \end{bmatrix} = \underbrace{\begin{bmatrix} x \\ y \end{bmatrix}}_{h(x,y)}$$

We then write the system in the form given by 6.95:

$$\begin{aligned} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} &= \begin{bmatrix} L_f h_1(x) \\ L_f h_2(x) \end{bmatrix} + \begin{bmatrix} L_{g_1} h_1(x) & L_{g_2} h_1(x) \\ L_{g_1} h_2(x) & L_{g_2} h_2(x) \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \\ &= \begin{bmatrix} [1,0,0][0,0,0]^T \\ [0,1,0][0,0,0]^T \end{bmatrix} + \begin{bmatrix} [1,0,0][\cos(\theta),\sin(\theta),0]^T & [1,0,0][0,0,1]^T \\ [0,1,0][\cos(\theta),\sin(\theta),0]^T & [0,1,0][0,0,1]^T \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \underbrace{\begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \end{bmatrix}}_{E(x,y,\theta)} \begin{bmatrix} v \\ \omega \end{bmatrix} \end{aligned}$$

$E(x, y, \theta)$ is not invertible as the determinant is 0.

(c) The total relative degree is 4 because $\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \end{bmatrix} \Rightarrow \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \dot{v}\cos(\theta) - v\sin(\theta)\dot{\theta} \\ \dot{v}\sin(\theta) + v\cos(\theta)\dot{\theta} \end{bmatrix} \Rightarrow r = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$.

Like in Part b, we rewrite the system in the form given by 6.94.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \end{bmatrix} = \underbrace{\begin{bmatrix} v\cos\theta \\ v\sin\theta \\ 0 \\ 0 \end{bmatrix}}_{f(x,y,\theta,v)} + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}}_{g(x,y,\theta,v)} \begin{bmatrix} \omega \\ a \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} x \\ y \end{bmatrix} = \underbrace{\begin{bmatrix} x \\ y \end{bmatrix}}_{h(x,y,\theta,v)}$$

And then rewrite the system in the form given by 6.95:

$$\begin{aligned} \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} &= \begin{bmatrix} L_f h_1(x) \\ L_f h_2(x) \end{bmatrix} + \begin{bmatrix} L_{g_1} L_f h_1(x) & L_{g_2} L_f h_1(x) \\ L_{g_1} L_f h_2(x) & L_{g_2} L_f h_2(x) \end{bmatrix} \begin{bmatrix} \omega \\ a \end{bmatrix} \\ &= \begin{bmatrix} v\cos\theta \\ v\sin\theta \end{bmatrix} + \underbrace{\begin{bmatrix} -v\sin\theta & \cos\theta \\ v\cos\theta & \sin\theta \end{bmatrix}}_{E(x,y,\theta,v)} \begin{bmatrix} \omega \\ a \end{bmatrix} \end{aligned}$$

$E(x, y, \theta, v)$ is invertible with inverse:

$$E^{-1}(x, y, \theta v) = \begin{bmatrix} -\frac{\sin\theta}{v} & \frac{\cos\theta}{v} \\ \cos\theta & \sin\theta \end{bmatrix}$$

(d) Assuming that $E^{-1}(x, y, \theta, v)$ is well defined i.e. $v \neq 0$, then we can linearize the control inputs by choosing the control law

$$\begin{bmatrix} \omega \\ a \end{bmatrix} = E^{-1}\Big(\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} - \begin{bmatrix} v\cos\theta \\ v\sin\theta \end{bmatrix} \Big) \Rightarrow \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

Problem 2

Starting with our typical Lyapunov function candidate:

$$V = \frac{1}{2}Js^2 + \frac{1}{2}\tilde{a}P^{-1}\tilde{a}$$

If we choose our control law $u = -\eta s + Y^T\hat{a}$ and our adaptation law $\dot{\hat{a}} = -PYs$, then the time derivative of the Lyapunov function candidate becomes:

$$\dot{V} = -\eta s^2 + sY^T\tilde{a} + \dot{\tilde{a}}^T P^{-1}\tilde{a}$$

By design, we know that $sY^T\tilde{a} + \dot{\tilde{a}}^T P^{-1}\tilde{a} = 0$. I'll discuss the vector $\tilde{a}$ for brevity, but the following also applies element-wise to each of its components. In order for $\tilde{a} = \hat{a} - a \to \mathbb{R}_-$, the following must hold: $\dot{\tilde{a}} < 0$. Since P^{-1} is positive definite, this implies that $\dot{\tilde{a}}^T P^{-1}\tilde{a} > 0$. So turning off the adaptation law (i.e. setting $\tilde{a}$ to 0) removes a positive term, making $\dot{V}$ more negative.

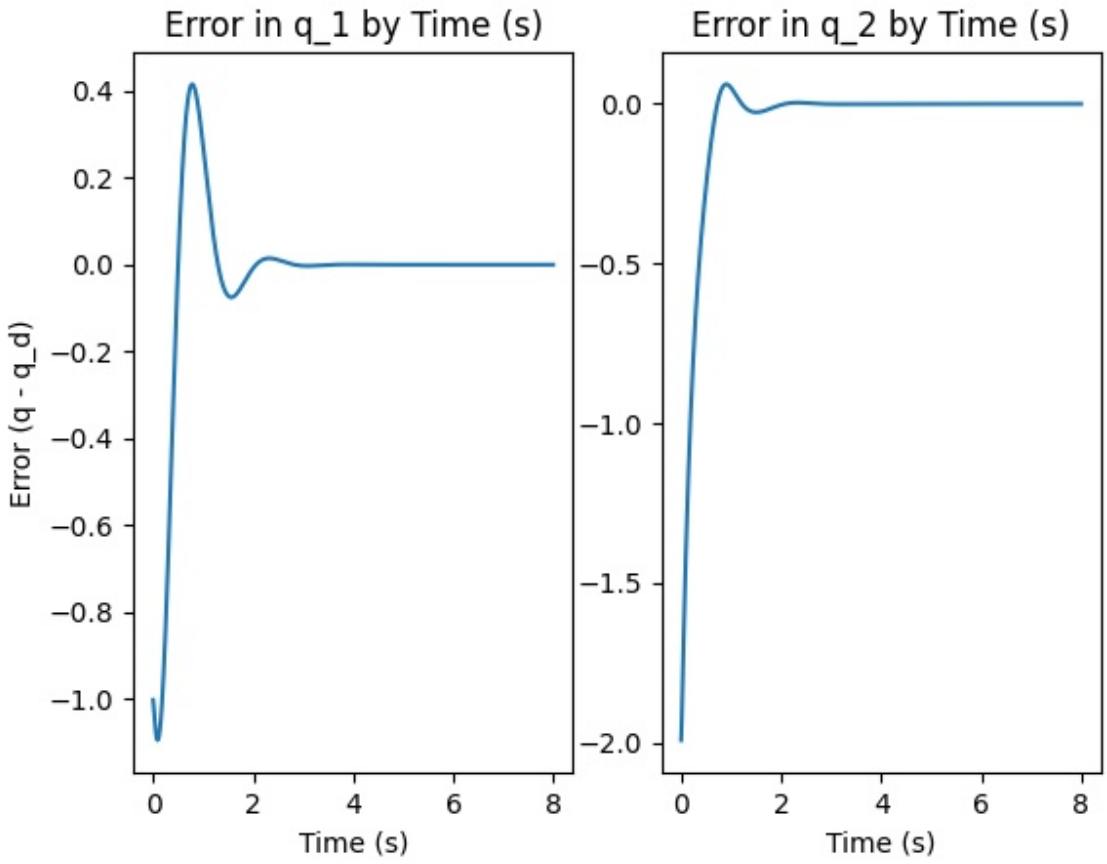
Problem 3

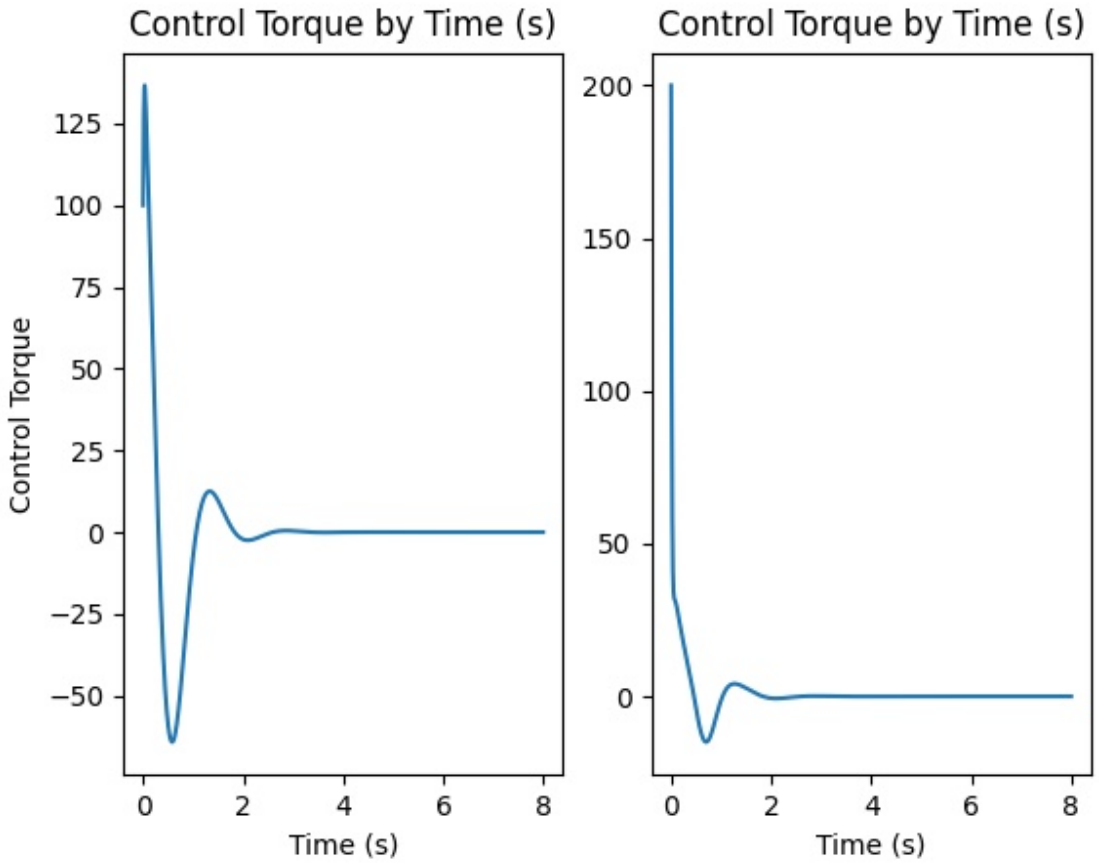
1. Code:

```
# Slotine and Li, Example 9.1
import matplotlib.pyplot as plt
import numpy as np

# problem coefficients
a_1 = 1 + 1 * (0.5 ** 2) + 0.25 + 2 * (0.6 ** 2) + 2
a_2 = 0.25 + 2 * (0.6 ** 2)
a_3 = 2 * 0.6 * np.cos(.5)
a_4 = 2 * 0.6 * np.sin(0.5)


def calc_H(q):
    H = np.array([
        [a_1 + 2 * a_3 * np.cos(q[1]) + 2 * a_4 * np.sin(q[1]),
         a_2 + a_3 * np.cos(q[1]) + a_4 * np.sin(q[1])],
        [a_2 + a_3 * np.cos(q[1]) + a_4 * np.sin(q[1]),
         a_2]
    ])
    return H


def calc_C(q, qdot):
```

```
    h = a_3 * np.sin(q[1]) - a_4 * np.cos(q[1])
    C = np.array([
        [-h * qdot[1], -h * (qdot[0] + qdot[1])],
        [h * qdot[0], 0]
    ])
    return C


def calc_tau(q, qdot):
    tau = -np.matmul(K_p, q - q_desired) - np.matmul(K_d, qdot)
    return tau


# simulation parameters
num_dts = 2000
control_history = np.zeros(shape=(num_dts, 2))
error_history = np.zeros(shape=(num_dts, 2))

dt = 0.004
q_desired = np.array([1, 2])  # desired angles, radians
q = np.zeros(2)  # initial angles, radians
qdot = np.zeros(2)  # initial velocity
qdotdot = np.zeros(2)  # initial acceleration
K_p, K_d = 100*np.eye(2), 20*np.eye(2)  # controller positive-definite matrices

for i in range(num_dts):
    C = calc_C(q=q, qdot=qdot)
    H = calc_H(q=q)
    Hinv = np.linalg.inv(H)
    tau = calc_tau(q=q, qdot=qdot)
    qdotdot = np.matmul(Hinv, tau - np.matmul(C, qdot))
    qdot += dt * qdotdot
    q += dt * qdot
    control_history[i][:] = tau
    error_history[i][:] = q - q_desired


fig, axes = plt.subplots(nrows=1, ncols=2, sharex=True)
axes[0].set_ylabel(f'Error (q - q_d)')
for i in range(2):
    ax = axes[i]
    ax.set_title(f'Error in q_{i+1} by Time (s)')
    ax.set_xlabel('Time (s)')
    ax.plot(
        dt * np.arange(num_dts),
        error_history[:, i])
plt.savefig('3a_error.jpg')


fig, axes = plt.subplots(nrows=1, ncols=2, sharex=True)
axes[0].set_ylabel(f'Control Torque')
for i in range(2):
    ax = axes[i]
    ax.set_title(f'Control Torque by Time (s)')
```

```
        ax.set_xlabel('Time (s)')
        ax.plot(
            dt * np.arange(num_dts),
            control_history[:, i])
plt.savefig('3a_torque.jpg')
```





2.

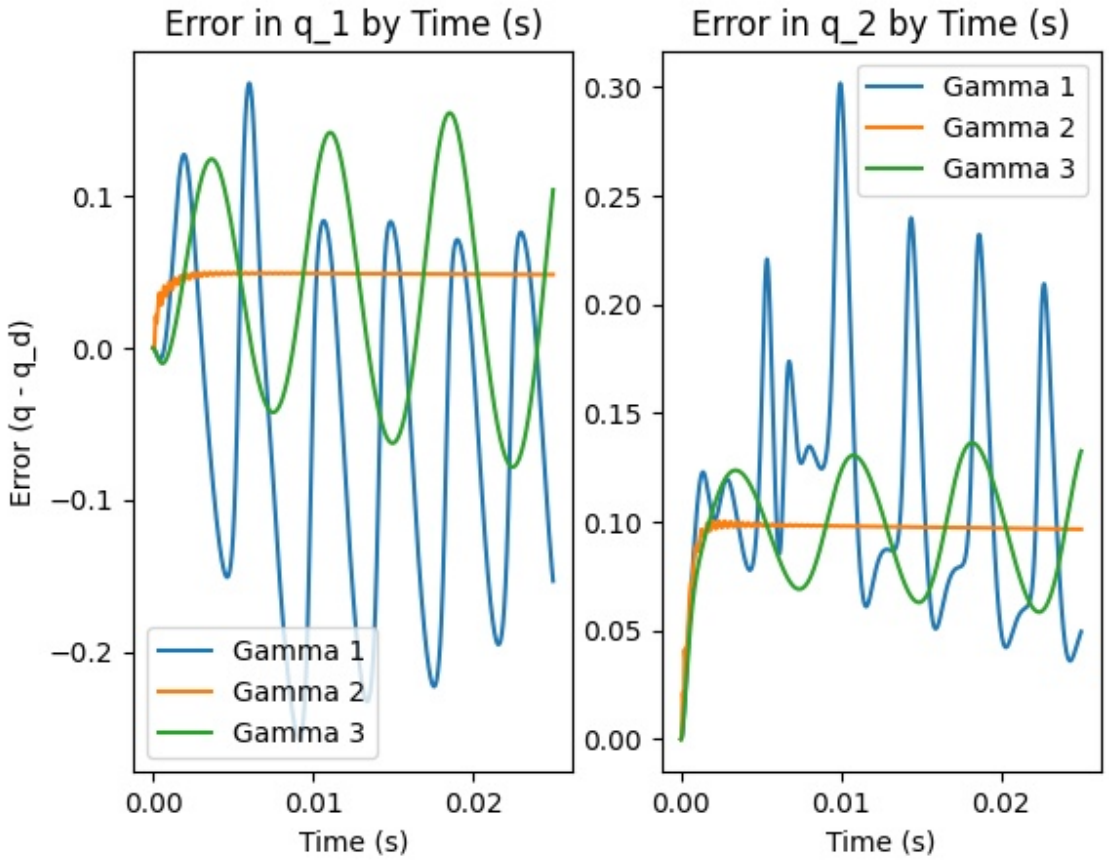
```
# Slotine and Li, Example 9.3
import matplotlib.pyplot as plt
import numpy as np

# problem coefficients
a_1 = 1 + 1 * (0.5 ** 2) + 0.25 + 2 * (0.6 ** 2) + 2
a_2 = 0.25 + 2 * (0.6 ** 2)
a_3 = 2 * 0.6 * np.cos(.5)
a_4 = 2 * 0.6 * np.sin(0.5)


def calc_H(q):
    H = np.array([
        [a_1 + 2 * a_3 * np.cos(q[1]) + 2 * a_4 * np.sin(q[1]),
         a_2 + a_3 * np.cos(q[1]) + a_4 * np.sin(q[1])],
        [a_2 + a_3 * np.cos(q[1]) + a_4 * np.sin(q[1]),
         a_2]
    ])
    return H


def calc_C(q, qdot):
```

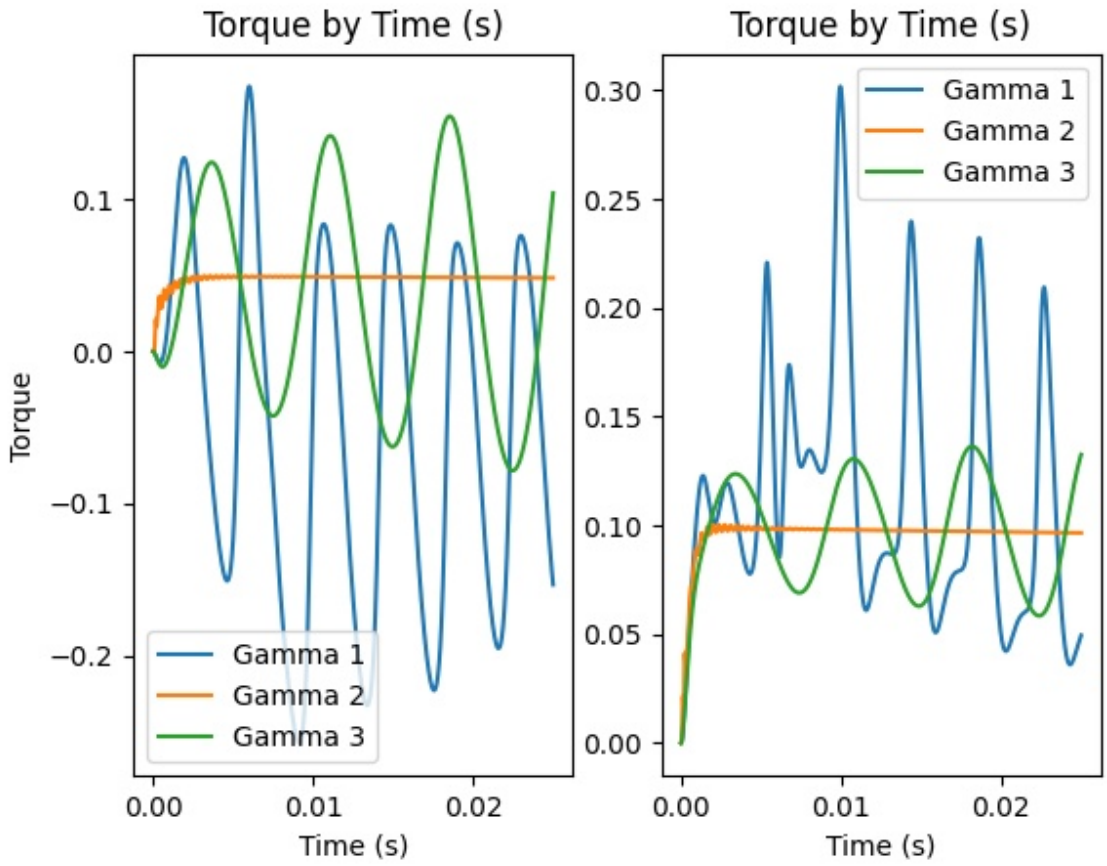
```
    h = a_3 * np.sin(q[1]) - a_4 * np.cos(q[1])
    C = np.array([
        [-h * qdot[1], -h * (qdot[0] + qdot[1])],
        [h * qdot[0], 0]
    ])
    return C


def calc_Y(q, qdot, qdot_r, qdotdot_r):
    Y_13 = (2 * qdotdot_r[0] + qdotdot_r[1])*np.cos(q[1]) - \
           (qdot[1]*qdot_r[0] + qdot[0]*qdot_r[1] + qdot[1]*qdot_r[1])*np.sin(q[1])
    Y_14 = (2 * qdotdot_r[0] + qdotdot_r[1])*np.sin(q[1]) + \
           (qdot[1]*qdot_r[0] + qdot[0]*qdot_r[1] + qdot[1]*qdot_r[1])*np.cos(q[1])
    Y = np.array([
        [qdotdot_r[0],
         qdotdot_r[1],
         Y_13,
         Y_14],
        [0,
         qdotdot_r[0] + qdotdot_r[1],
         qdotdot_r[0] * np.cos(q[1]) + qdot[0]*qdot_r[0]*np.sin(q[1]),
         qdotdot_r[1] * np.sin(q[1]) - qdot[0]*qdot_r[0]*np.cos(q[1])],
    ])
    return Y


# simulation parameters
num_dts = 25000
control_history = np.zeros(shape=(3, num_dts, 2))  # three gammas
error_history = np.zeros(shape=(3, num_dts, 2))

dt = 0.0001
K_p, K_d = 100*np.eye(2), 20*np.eye(2)  # controller positive-definite matrices
lambd = 20*np.eye(2)
gamma_1 = np.diag([0.03, 0.05, 0.1, 0.3])
gammas = [gamma_1, 200 * gamma_1, 0.1 * gamma_1]


for i, gamma in enumerate(gammas):
    q = np.zeros(2)  # initial angles, radians
    qdot = np.zeros(2)  # initial velocity
    qdotdot = np.zeros(2)  # initial acceleration
    ahat = np.zeros(4)

    for j in range(num_dts):

        # determine t
        t = dt * j / 100

        # calculate desired terms
        q_desired = np.array([1 - np.exp(-t), 2 - 2*np.exp(-t)])
        qdot_desired = np.array([np.exp(-t), 2.*np.exp(-t)])
        qdotdot_desired = -1. * qdot_desired
```

```
        # calculate tilde terms
        qtilde = q - q_desired
        qtildedot = qdot - qdotdot_desired

        # calculate r terms
        qdot_r = qdot_desired - np.matmul(lambd, qtilde)
        qdotdot_r = qdotdot_desired - np.matmul(lambd, qtildedot)

        # calculate s
        s = qdot - qdot_desired + np.matmul(lambd, qtilde)

        # calculate Y
        Y = calc_Y(q=q, qdot=qdot, qdot_r=qdot_r, qdotdot_r=qdotdot_r)

        # update adaptation law
        ahatdot = -np.matmul(gamma, np.matmul(Y.T, s))
        ahat += dt * ahatdot

        C = calc_C(q=q, qdot=qdot)
        H = calc_H(q=q)
        Hinv = np.linalg.inv(H)

        # calculate control law
        tau = np.matmul(Y, ahat) - np.matmul(K_d, s)

        # update system
        qdotdot = np.matmul(Hinv, tau - np.matmul(C, qdot))
        qdot += dt * qdotdot
        q += dt * qdot

        # record system
        control_history[i, j][:] = tau
        error_history[i, j][:] = q - q_desired


fig, axes = plt.subplots(nrows=1, ncols=2, sharex=True)
axes[0].set_ylabel(f'Error (q - q_d)')
for i in range(2):
    ax = axes[i]
    ax.set_title(f'Error in q_{i + 1} by Time (s)')
    ax.set_xlabel('Time (s)')
    for j, gamma in enumerate(gammas):
        ax.plot(
            dt * np.arange(num_dts) / 100,
            error_history[j, :, i],
            label=f'Gamma_{j+1}')
    ax.legend()
plt.savefig('3b_error.jpg')


fig, axes = plt.subplots(nrows=1, ncols=2, sharex=True)
axes[0].set_ylabel(f'Torque')
for i in range(2):
    ax = axes[i]
```

```
    ax.set_title(f'Torque by Time (s)')
    ax.set_xlabel('Time (s)')
    for j, gamma in enumerate(gammas):
        ax.plot(
            dt * np.arange(num_dts) / 100,
            error_history[j, :, i],
            label=f'Gamma {j+1}')
    ax.legend()
plt.savefig('3b_torque.jpg')
```

The error doesn't converge to 0, so something is wrong, but I don't have time to debug.




Torque by Time (s)
Torque
0.1
0.0
−0.1
−0.2
Gamma 1
Gamma 2
Gamma 3
0.00
0.01
0.02
Time (s)
Torque by Time (s)
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Gamma 1
Gamma 2
Gamma 3
0.00
0.01
0.02
Time (s)