ME599 Homework 7 - Akshat Dubey

Problem 1

We want to develop a control algorithm to control the hare population in a park at 30 each month. The hares are preyed upon by foxes, and the prey-predator dynamics can be captured by the Lotka-Volterra equations:

$$\dot{x} = \alpha x - \beta xy$$
$$\dot{y} = -\gamma y + \delta xy$$

Where

- x: hare population
- y: fox population
- α: hare growth rate
- β: hare death rate due to predation
- γ: fox death rate
- δ: fox growth rate due to predation

The timestep here is 1 month or 4 weeks, and the population data every 4 weeks is provided in `fox_hare.mat`.

Since this is a first order nonlinear system, we can use a least squares method like we did in HW5 to find the parameters because we know the form of the system. Instead of keeping the equation in its current form and having to use finnite difference to compute derivatives, we can integrate both sides between the initial time t_0 and t to get:

$$\int_{t_0}^{t} \dot{x} dt = \int_{t_0}^{t} (\alpha x - \beta xy) dt$$
$$x(t) - x(t_0) = \alpha \int_{t_0}^{t} x(t) dt - \beta \int_{t_0}^{t} x(t)y(t) dt$$
$$\int_{t_0}^{t} \dot{y} dt = \int_{t_0}^{t} (-\gamma y + \delta xy) dt$$
$$y(t) - y(t_0) = -\gamma \int_{t_0}^{t} y(t) dt + \delta \int_{t_0}^{t} x(t)y(t) dt$$

Im matrix form,

$$\begin{bmatrix} x(t) - x(t_0) \\ y(t) - y(t_0) \end{bmatrix} = \begin{bmatrix} \int_{t_0}^{t} x(t) dt & -\int_{t_0}^{t} x(t)y(t) dt & 0 & 0 \\ 0 & 0 & -\int_{t_0}^{t} y(t) dt & \int_{t_0}^{t} x(t)y(t) dt \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix}$$

We can then stack these for multiple time steps from t_0 to t_n, where t_0 is the first month where we have data. We then get a system of equations in the form of $A\theta = b$

$$\underbrace{\begin{bmatrix} \int_{t_0}^{t_0} x(t) dt & -\int_{t_0}^{t_0} x(t)y(t) dt & 0 & 0 \\ 0 & 0 & -\int_{t_0}^{t_0} y(t) dt & \int_{t_0}^{t_0} x(t)y(t) dt \\ \int_{t_0}^{t_1} x(t) dt & -\int_{t_0}^{t_1} x(t)y(t) dt & 0 & 0 \\ 0 & 0 & -\int_{t_0}^{t_1} y(t) dt & \int_{t_0}^{t_1} x(t)y(t) dt \\ \vdots & \vdots & \vdots & \vdots \\ \int_{t_0}^{t_n} x(t) dt & -\int_{t_0}^{t_n} x(t)y(t) dt & 0 & 0 \\ 0 & 0 & -\int_{t_0}^{t_n} y(t) dt & \int_{t_0}^{t_n} x(t)y(t) dt \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix}}_{\theta} = \underbrace{\begin{bmatrix} x(t_0) - x(t_0) \\ y(t_0) - y(t_0) \\ x(t_1) - x(t_0) \\ y(t_1) - y(t_0) \\ \vdots \\ x(t_n) - x(t_0) \\ y(t_n) - y(t_0) \end{bmatrix}}_{b}$$

Where the least squares solution is given by:

$$\theta^* = (A^T A)^{-1} A^T b$$

After constructing the matrices, the least squares solution found is

$$\theta^* = \begin{bmatrix} \alpha^* \\ \beta^* \\ \gamma^* \\ \delta^* \end{bmatrix} = \begin{bmatrix} 1.401247 \\ 0.093308 \\ 0.318335 \\ 0.010616 \end{bmatrix}$$

with a squared error of 52.6.

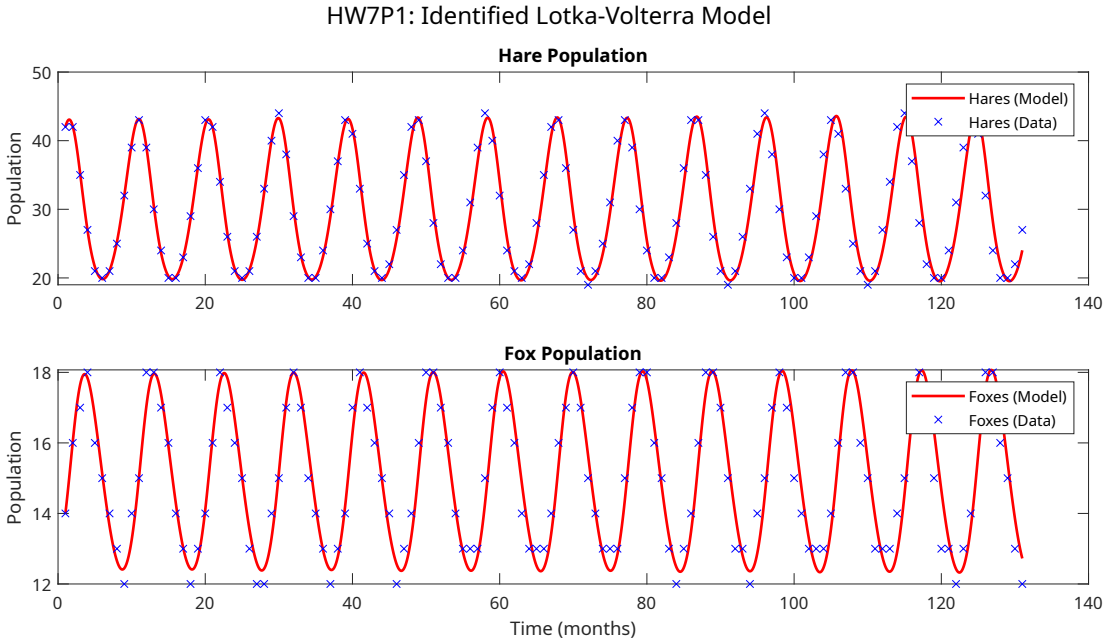


Figure 1: Comparison of model with data

We can see that the model fits the data well, and so can be used for controlling the hare population. If we want to use any advanced techniques like LQR or MPC, we need to linearize the system around the equilibrium point. The equilibrium points of the system can be found by setting $\dot{x} = 0$ and $\dot{y} = 0$:

$$\dot{x} = \alpha x - \beta xy = 0$$
$$\dot{y} = -\gamma y + \delta xy = 0$$

Bringing the negative terms to the other side

$$\alpha x = \beta xy$$
$$\gamma y = \delta xy$$

Dividing both sides by x and y respectively

$$y_e = \frac{\alpha}{\beta}$$
$$x_e = \frac{\gamma}{\delta}$$

Substituting the values of α, β, γ and δ we found earlier, we get the equilibrium points as $x_e = 29.985193$ and $y_e = 15.017423$. Since we want to keep the hare population at 30, we can use this as the equilibrium point for our linearization as it is very close to the desired population. We also know that the control input u here is the number of foxes added, which will only affect the value of y. At the equilibrium, $u_e = 0$ as well. To express this system in a

linear state space form, we can find the jacobian of the system with respect to $\boldsymbol{x}$ and u at the equilibrium point. Starting from the original nonlinear equations in state space form with the additional control input u:

$$\begin{aligned}\dot{\boldsymbol{x}} = \begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} &= f(\boldsymbol{x}, u)\\ &= \begin{bmatrix}\alpha x - \beta xy\\ -\gamma y + \delta xy + u\end{bmatrix}\end{aligned}$$

We find the jacobian $\nabla_x f$ as:

$$\begin{aligned}\nabla_x f &= \begin{bmatrix}\frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y}\\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y}\end{bmatrix}\\ \frac{\partial f_1}{\partial x} &= \alpha - \beta y\\ \frac{\partial f_1}{\partial y} &= -\beta x\\ \frac{\partial f_2}{\partial x} &= \delta y\\ \frac{\partial f_2}{\partial y} &= -\gamma + \delta x\\ \implies \nabla_x f &= \begin{bmatrix}\alpha - \beta y & -\beta x\\ \delta y & -\gamma + \delta x\end{bmatrix}\end{aligned}$$

and the jacobian $\nabla_u f$ as:

$$\begin{aligned}\nabla_u f &= \begin{bmatrix}\frac{\partial f_1}{\partial u}\\ \frac{\partial f_2}{\partial u}\end{bmatrix}\\ \frac{\partial f_1}{\partial u} &= 0\\ \frac{\partial f_2}{\partial u} &= 1\\ \implies \nabla_u f &= \begin{bmatrix}0\\ 1\end{bmatrix}\end{aligned}$$

Then we get A_c and B_c by evaluating the jacobian at the equilibrium point:

$$A_c = \nabla_x f\Big|_{(x_e, y_e)}$$

$$B_c = \nabla_u f\Big|_{(x_e, y_e)}$$

The linearized continuous time system is then given by:

$$\dot{\delta\boldsymbol{x}} = A_c \delta\boldsymbol{x} + B_c \delta u$$

Where $\delta\boldsymbol{x} = \begin{bmatrix}x - x_e\\ y - y_e\end{bmatrix}$ and $\delta u = u - u_e$, which are small deviations from the equilibrium point.

We can then convert this continuous time linear system to a discrete time system using MATLAB's `c2d` function with a zero order hold to get the discrete time system with the matrices A_d, B_d.

$$\boldsymbol{x}_{k+1} = A_d \boldsymbol{x_k} + B_d u_k$$

Checking the eigenvalues of A_d we get:

- 0.7851 + 0.6193i
- 0.7851 - 0.6193i

Both the eigenvalues have a magnitude ≤ 1, which means the system is stable at the equilibrium point. Hence, from a given starting hare and fox population, we need to control the hare population till it reaches the equilibrium point. We can design a Linear Quadratic Regulator (LQR) using this linear model to get the population to the equilibrium. The discrete-time LQR controller has the cost function:

$$J = \sum_{k=0}^{\infty} \left(\boldsymbol{x}_k^T Q \boldsymbol{x}_k + u_k^T R u_k \right)$$

Where Q and R are the weights for the state and control input respectively. We only care about the hare population, so we can set $Q = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}$ and $R = 2$. The optimal control gain matrix K is given by the solution to the discrete-time algebraic Riccati equation (DARE), which can be solved using MATLAB's `dlqr` function. Once we have the optimal control law, we can get the optimal control input u_k to drive the system to the equilibrium point:

$$u_k = -K(\boldsymbol{x}_k - \boldsymbol{x}_e)$$

When simulating the population dynamics and the controller, we round $\boldsymbol{x}_k$ and u_k to the nearest integer at the end of each month since the animal population can only be integers at the end of each month. We then hold this value of u_k for one month while the population dynamics of the foxes and hares evolve. Simulating this system with the optimal control input, we get the following evolution of populations for various initial conditions:

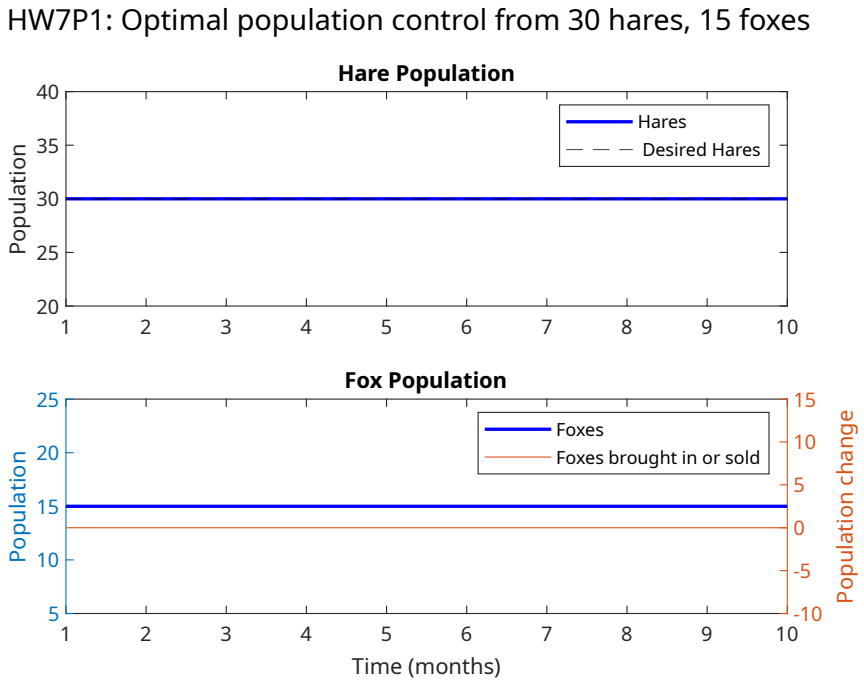


Figure 2: Optimal population control

We can see that the controller is able to keep the hare population at 30 as desired by the manager under different initial conditions and hence the controller is robust.

HW7P1: Optimal population control from 20 hares, 25 foxes

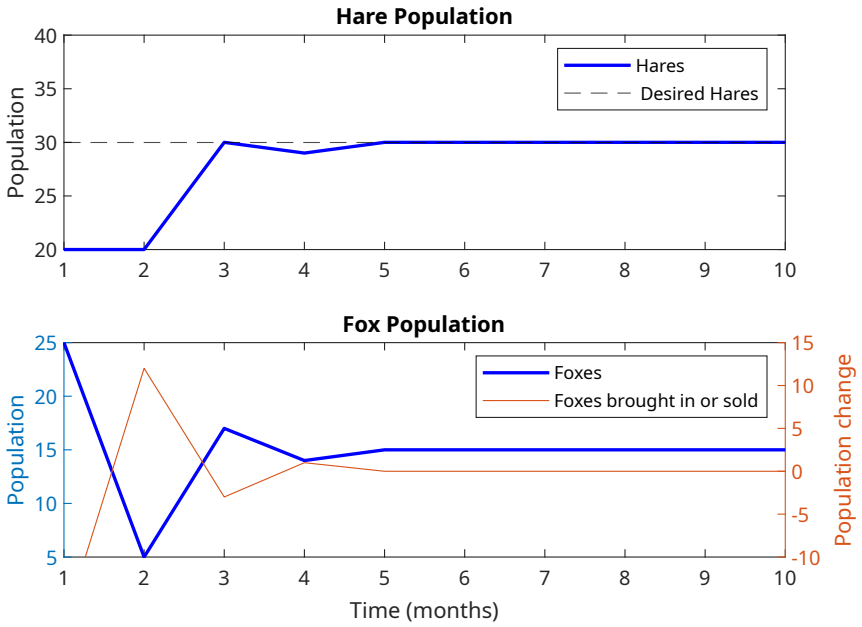


Figure 3: Optimal population control

HW7P1: Optimal population control from 35 hares, 10 foxes

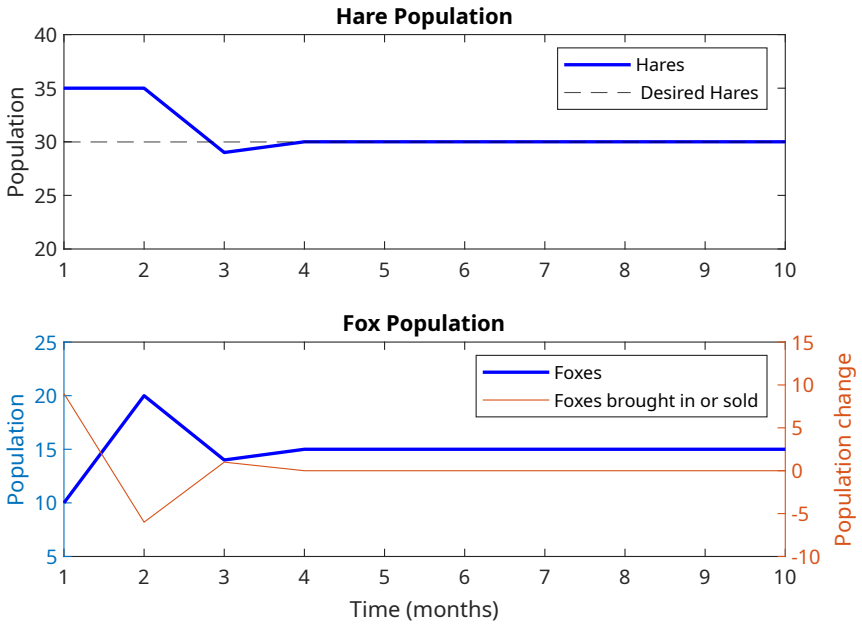


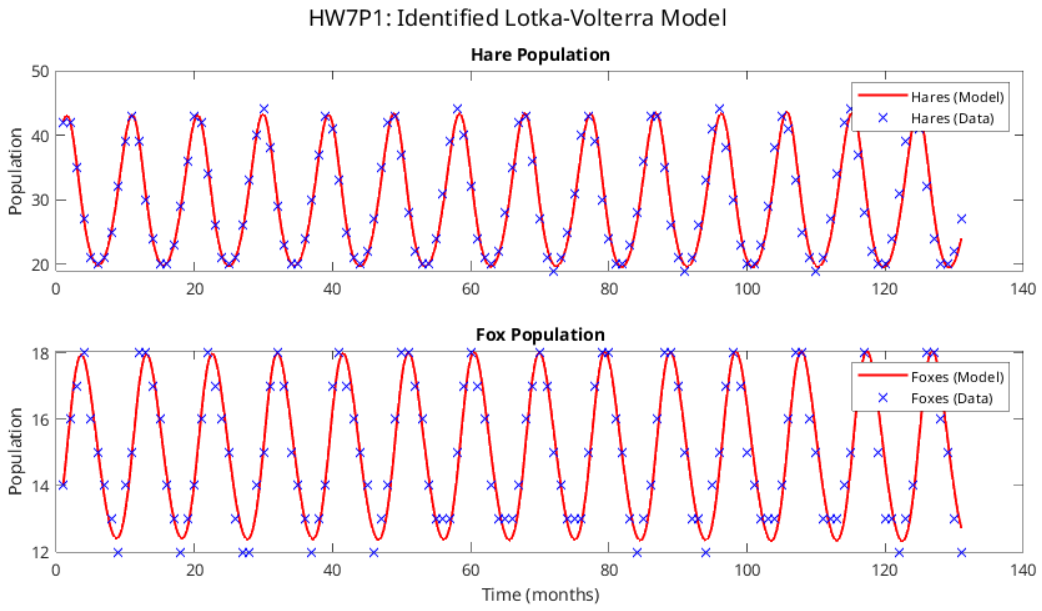
Figure 4: Optimal population control

ME599 HW7 Problem 1

```
clc; clear; close all;

load('fox_hare.mat');

x = hare; y = fox; xy = hare.*fox;
x_cum = cumtrapz(x);
y_cum = cumtrapz(y);
xy_cum = cumtrapz(xy);
n = length(x);
% construct matrices A and b
A = zeros(n*2, 4);
b = zeros(n*2, 1);
for i = 1:n
    row = 2*i-1;

    A(row, :) = [x_cum(i), -xy_cum(i), 0, 0];
    A(row+1, :) = [0, 0, -y_cum(i), xy_cum(i)];

    b(row) = x(i) - x(1);
    b(row+1) = y(i) - y(1);
end

% solution
theta = A\b;
alpha = theta(1); beta = theta(2); gamma = theta(3); delta = theta(4);
fprintf("HW7P1: least squares solution:\n - alpha: %f\n - beta: %f\n - gamma:
%f\n - delta: %f\n", alpha, beta, gamma, delta);
fprintf("HW7P1: least squares error: %f\n", norm(A*theta - b));

% simulate and plot system
tspan = 1:1:n;
[t, x_ode] = ode45(@(t, x) LotkaVolterra(t, x, 0, theta), 1:0.1:n, [x(1);
y(1)]);
fig = figure;
sgtitle('HW7P1: Identified Lotka-Volterra Model');
subplot(2, 1, 1);
plot(t, x_ode(:, 1), 'r', 'Linewidth', 1.5, 'DisplayName', 'Hares (Model)')
hold on;
scatter(tspan, x, 'xb', 'DisplayName', 'Hares (Data)');
ylabel('Population');
title('Hare Population');
legend('Location', 'northeast');

subplot(2, 1, 2);
plot(t, x_ode(:, 2), 'r', 'Linewidth', 1.5, 'DisplayName', 'Foxes (Model)')
hold on;
scatter(tspan, y, 'xb', 'DisplayName', 'Foxes (Data)');
xlabel('Time (months)');
ylabel('Population');
title('Fox Population');
```

```
legend('Location', 'northeast');
fig.Position(3:4) = [1000 500];
saveas(fig, 'figs/hw7p1_model.svg');
```

```
HW7P1: least squares solution:
 - alpha: 1.401247
 - beta: 0.093308
 - gamma: 0.318335
 - delta: 0.010616
HW7P1: least squares error: 52.602650
```



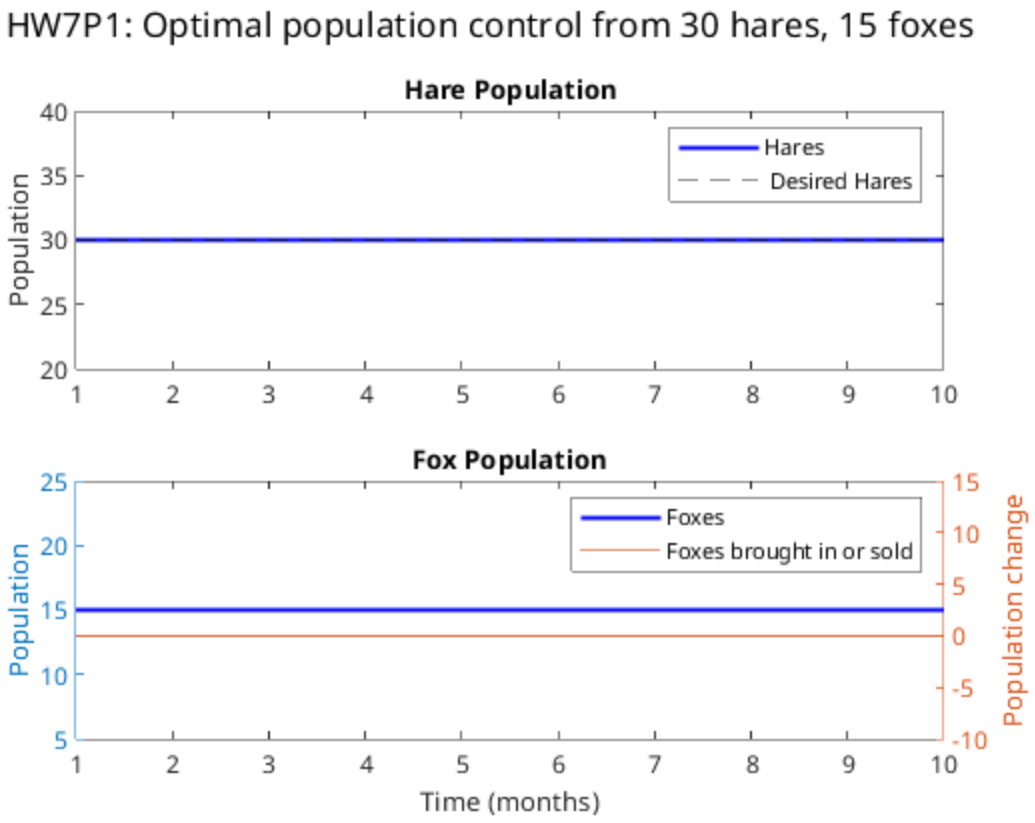
linearize around equilibrium

```
x_e = gamma/delta;
y_e = alpha/beta;
fprintf("HW7P1: Equilibrium point:\n - x_e: %f\n - y_e: %f\n", x_e, y_e);

Ac = [
    alpha-beta*y_e, -beta*x_e;
    delta*y_e, -gamma+delta*x_e
    ];
Bc = [0; 1];
[Ad, Bd, ~, ~] = c2dm(Ac, Bc, [], [], 1, 'zoh');

eigA = eig(Ad);
fprintf("HW7P1: Eigenvalues of discrete-time system:\n");
disp(eigA);
if abs(eigA(1)) <= 1 && abs(eigA(2)) <= 1
    fprintf(" - System is stable\n");
else
    fprintf(" - System is unstable\n");
end
```

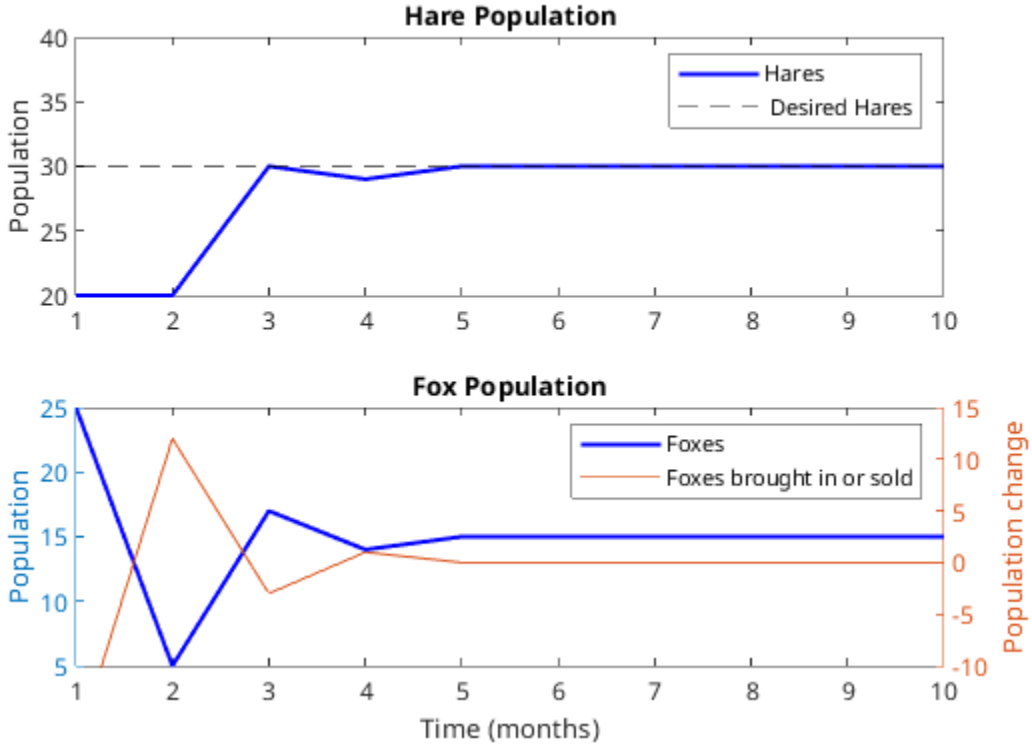
```
% setup LQR controller
Q = [10 0;
    0 1];
R = 2;
[K, ~, ~] = dlqr(Ad, Bd, Q, R);

% simulate closed-loop system
k_end = 10; % number of months to simulate
x0s = [
    30, 20, 35;
    15, 25, 10;
    ];
for i=1:size(x0s, 2)
    x0 = x0s(:, i);
    x_k = x0;
    x_history = zeros(2, k_end);
    u_history = zeros(k_end, 1);
    for k = 1:k_end
        x_history(:, k) = x_k;
        x_kdelta = round(x_k - [30; y_e]);
        u_k = round(-K*x_kdelta);
        u_history(k) = u_k;
        [~, x_ode] = ode45(@(t, x) LotkaVolterra(t, x, u_k, theta), k:k+1,
x_k);
        x_k = round(x_ode(end, :)');
    end

    % plot closed-loop system
    fig = figure;
    sgtitle("HW7P1: Optimal population control from " + x0(1) + " hares, " +
x0(2) + " foxes");
    subplot(2, 1, 1);
    plot(x_history(1, :), 'b', 'Linewidth', 1.5, 'DisplayName', 'Hares');
    hold on;
    yline(x_e(1), 'k--', 'DisplayName', ' Desired Hares');
    ylabel('Population');
    title('Hare Population');
    legend('Location', 'northeast');
    ylim([20 40]);

    subplot(2, 1, 2);
    yyaxis left;
    plot(x_history(2, :), 'b', 'Linewidth', 1.5, 'DisplayName', 'Foxes');
    ylabel('Population');
    ylim([5 25]);
    hold on;
    yyaxis right;
    plot(u_history, 'DisplayName', 'Foxes brought in or sold');
    ylabel('Population change');
    ylim([-10 15]);
    xlabel('Time (months)');
    title('Fox Population');
    legend('Location', 'northeast');
```

```
    saveas(fig, "figs/hw7p1_control_"+x0(1)+".svg");
end
function xdot = LotkaVolterra(~, x, u, theta)
alpha = theta(1);
beta = theta(2);
gamma = theta(3);
delta = theta(4);

xdot = zeros(2, 1);
xdot(1) = alpha*x(1) - beta*x(1)*x(2);
xdot(2) = -gamma*x(2) + delta*x(1)*x(2) + u;
end
```

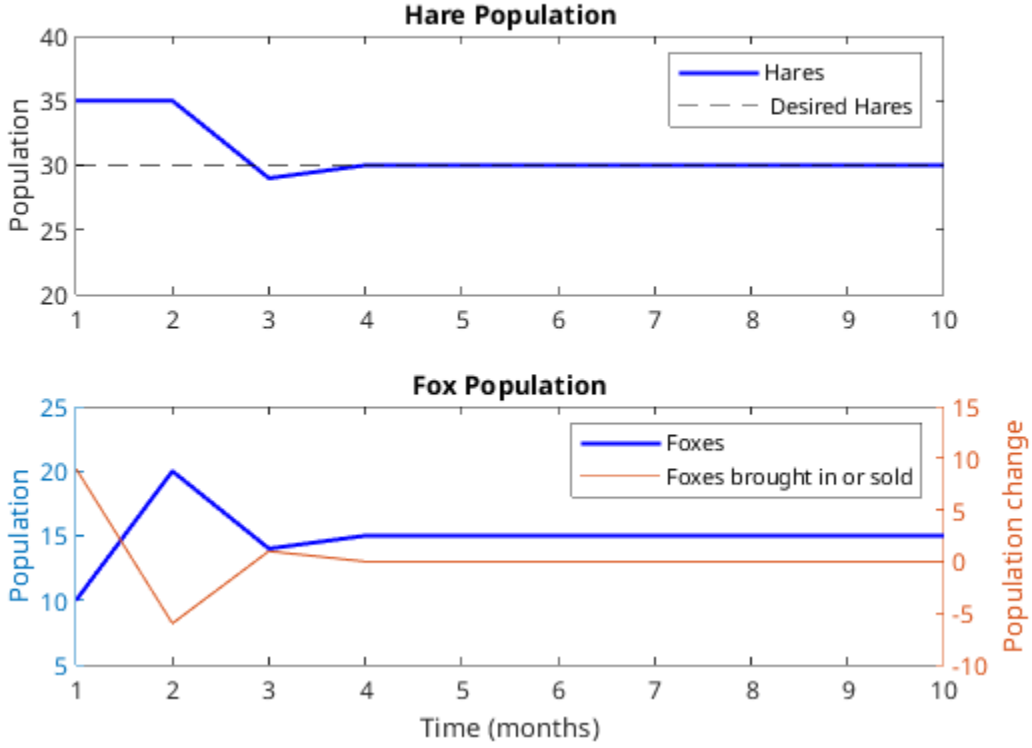
```
HW7P1: Equilibrium point:
 - x_e: 29.985193
 - y_e: 15.017423
HW7P1: Eigenvalues of discrete-time system:
   0.7851 + 0.6193i
   0.7851 - 0.6193i

 - System is stable
```



HW7P1: Optimal population control from 20 hares, 25 foxes



HW7P1: Optimal population control from 35 hares, 10 foxes



Published with MATLAB® R2023b

Problem 2

Problem 2.a

We need to identify the discrete-time system in `aerialVehSim.p`, which has been identified as a linear system in Homework 5. We have the state $\boldsymbol{x}$ and control input $\boldsymbol{u}$:

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \phi \\ \theta \\ \psi \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \in \mathbb{R}^{12}, \quad \boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \in \mathbb{R}^4 \quad n = 12, \quad m = 4$$

The dynamics are given by the function $F =$ `aerialVehSim.p` in discrete time for the given time step $Ts = 0.1$:

$$\boldsymbol{x}_{k+1} = F(\boldsymbol{x}_k, u_k, Ts)$$

We will attempt to identify the system by using DMD with control as outlined here. Given a discrete time system of the form $\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k$, we first collect data the system given an initial state $\boldsymbol{x}_0$ and random control inputs $\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, u_{N-1}$ that produce state trajectories $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$. We can then stack this data into matrices X, X' and U:

$$X = \begin{bmatrix} \vdots & \vdots & \ldots & \vdots \\ \boldsymbol{x}_0 & \boldsymbol{x}_1 & \ldots & \boldsymbol{x}_{N-1} \\ \vdots & \vdots & \ldots & \vdots \end{bmatrix} \in \mathbb{R}^{n \times N}, \quad X' = \begin{bmatrix} \vdots & \vdots & \ldots & \vdots \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & \ldots & \boldsymbol{x}_N \\ \vdots & \vdots & \ldots & \vdots \end{bmatrix} \in \mathbb{R}^{n \times N}, \quad U = \begin{bmatrix} \vdots & \vdots & \ldots & \vdots \\ \boldsymbol{u}_0 & \boldsymbol{u}_1 & \ldots & \boldsymbol{u}_{N-1} \\ \vdots & \vdots & \ldots & \vdots \end{bmatrix} \in \mathbb{R}^{m \times N}$$

Such that

$$\begin{aligned} X' &= AX + BU \\ &= \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} X \\ U \end{bmatrix} \end{aligned}$$

If $N > n$, we can solve for A and B using the least squares solution of $||X' - \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} X \\ U \end{bmatrix} ||_F$:

$$\begin{bmatrix} A & B \end{bmatrix} = X' \begin{bmatrix} X \\ U \end{bmatrix}^+$$

Where for a matrix D, D^+ is the pseudo-inverse of D

$$D^+ = D^T (DD^T)^{-1}$$

After we have the compound matrix $\begin{bmatrix} A & B \end{bmatrix}$, we can extract A and B by splitting the matrix into two parts, extracting the first n columns as A and the last m columns as B. The system was identified as follows:

```
HW7 P2: Identified A:
    1.0000         0         0    0.1000         0         0         0   -0.0490         0         0   -0.0020         0
         0    1.0000         0         0    0.1000         0    0.0490         0         0    0.0020         0         0
         0         0    1.0000         0         0    0.1000         0         0         0         0         0         0
         0         0         0    1.0000         0         0         0   -0.9810         0         0   -0.0490         0
         0         0         0         0    1.0000         0    0.9810         0         0    0.0490         0         0
         0         0         0         0         0    1.0000         0         0         0         0         0         0
         0         0         0         0         0         0    1.0000         0         0    0.1000         0         0
         0         0         0         0         0         0         0    1.0000         0         0    0.1000         0
         0         0         0         0         0         0         0         0    1.0000         0         0    0.1000
         0         0         0         0         0         0         0         0         0    1.0000         0         0
         0         0         0         0         0         0         0         0         0         0    1.0000         0
         0         0         0         0         0         0         0         0         0         0         0    1.0000

HW7 P2: Identified B:
         0         0         0         0
         0    0.0010         0         0
    0.0050         0    0.0020         0
         0         0   -0.0160    0.0020
         0    0.0160         0         0
    0.1000         0         0         0
         0    0.0500         0         0
         0         0    0.0500         0
         0         0         0    0.0500
         0    1.0000         0         0
         0         0    1.0000         0
         0         0         0    1.0000
```

Figure 1: A and B matrices of the system

Plotting the actual system against the identified system shows us that the identified system matches the actual system well, hence we can conclude that the system has been identified correctly.

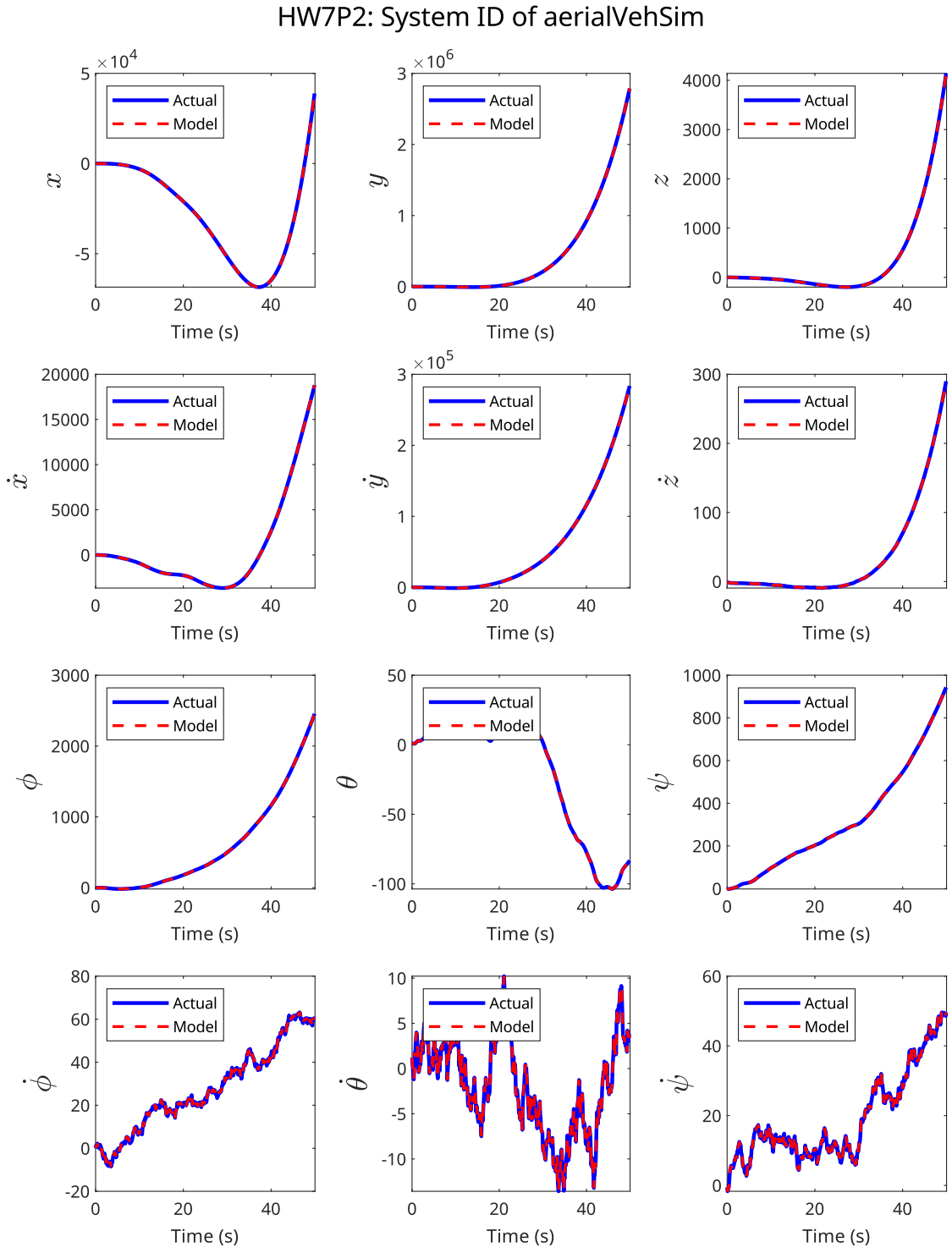


Figure 2: Comparison of actual and identified system

Problem 2.b

We are given a reference trajectory $\boldsymbol{x}_{ref}$ which is of the form:

$$\begin{bmatrix} 0.1sin\frac{t}{2} \\ 0.1cos\frac{t}{2} \\ 0.1t \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

which we need to track using an MPC controller. The MPC controller has the cost function:

$$J = \frac{1}{2}\sum_{k=j}^{j+N-1} \left((\boldsymbol{x}_{ref}(k+1) - \boldsymbol{x}(k+1))^T Q(\boldsymbol{x}_{ref}(k+1) - \boldsymbol{x}(k+1)) + (\boldsymbol{u}(k)^T R\boldsymbol{u}(k))\right)$$

Where $N = 30$, Q and R are the weights on the state and control inputs respectively.

$$Q = \begin{bmatrix} 10I_3 & 0_{3\times 9} \\ 0_{9\times 3} & 0_{9\times 9} \end{bmatrix}, \quad R = I_4$$

We can assume that the first 3 states are observable and $x_0 = \begin{bmatrix} 0 & 0.1 & 0 & 0_{1\times 9} \end{bmatrix}^T$. To express the cost function as a function of the matrices we construct

$$X_k = \begin{bmatrix} x_k \\ x_{k+1} \\ \vdots \\ x_{k+N-1} \end{bmatrix} \in \mathbb{R}^{nN\times 1} \qquad U_k = \begin{bmatrix} u_k \\ u_{k+1} \\ \vdots \\ u_{k+N-1} \end{bmatrix} \in \mathbb{R}^{mN\times 1}$$

$$\bar{\mathcal{A}} = \begin{bmatrix} A_d \\ A_d^2 \\ \vdots \\ A_d^N \end{bmatrix} \in \mathbb{R}^{nN\times n} \quad \bar{\mathcal{B}} = \begin{bmatrix} B_d & 0 & \dots & 0 \\ A_dB_d & B_d & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_d^{N-1}B_d & A_d^{N-2}B_d & \dots & B_d \end{bmatrix} \in \mathbb{R}^{nN\times mN}$$

Such that the sequence of states can be written as

$$X_{k+1} = \bar{\mathcal{A}}x_k + \bar{\mathcal{B}}U_k$$

then, let

$$X_{ref} = \begin{bmatrix} \boldsymbol{x}_{ref}(k+1) \\ \boldsymbol{x}_{ref}(k+2) \\ \vdots \\ \boldsymbol{x}_{ref}(k+N) \end{bmatrix} \in \mathbb{R}^{12N\times 1}$$

the cost matrices can be stacked into a diagonal matrices such that

$$\bar{Q} = \text{diag}(Q, Q, \dots, Q) \in \mathbb{R}^{12N\times 12N}$$
$$\bar{R} = \text{diag}(R, R, \dots, R) \in \mathbb{R}^{4N\times 4N}$$

and the cost function becomes

$$
\begin{aligned}
J &= \frac{1}{2}((X_{ref} - X_{k+1})^T \bar{Q}(X_{ref} - X_{k+1}) + U_k^T \bar{R} U_k) \\
&= \frac{1}{2}((X_{ref} - (\bar{\mathcal{A}} x_k + \bar{\mathcal{B}} U_k))^T \bar{Q}(X_{ref} - (\bar{\mathcal{A}} x_k + \bar{\mathcal{B}} U_k)) + U_k^T \bar{R} U_k) \\
&= \frac{1}{2}(-2U_k^T \bar{\mathcal{B}}^T \bar{Q}(X_{ref} - \bar{\mathcal{A}} x_k) + U_k^T (\bar{\mathcal{B}}^T \bar{Q} \bar{\mathcal{B}} + \bar{R}) U_k + (X_{ref} - \bar{\mathcal{A}} x_k)^T \bar{Q}(X_{ref} - \bar{\mathcal{A}} x_k))
\end{aligned}
$$

The solution to get the optimal control sequence U_k^* under no constraints is given by setting the derivative $\partial J / \partial U_k = 0$. Hence, we have

$$
\begin{aligned}
\frac{\partial J}{\partial U_k} &= -\bar{\mathcal{B}}^T \bar{Q}(X_{ref} - \bar{\mathcal{A}} x_k) + (\bar{\mathcal{B}}^T \bar{Q} \bar{\mathcal{B}} + \bar{R}) U_k = 0 \\
\implies \quad U_k^* &= (\bar{\mathcal{B}}^T \bar{Q} \bar{\mathcal{B}} + \bar{R})^{-1} \bar{\mathcal{B}}^T \bar{Q}(X_{ref} - \bar{\mathcal{A}} x_k)
\end{aligned}
$$

Then, at every time step, we pick the first element of the optimal control sequence U_k^* as the control input u_k to perform MPC as per the MPC control law.

For the LQR controller, we set up the cost function as follows:

$$
J = \frac{1}{2} \sum_{k=0}^{\infty} \left(\boldsymbol{x}_k Q \boldsymbol{x}_k^T + \boldsymbol{u}_k R \boldsymbol{u}_k^T \right)
$$

Then at each time step k, the optimal control input is given by $u_k^* = -K \boldsymbol{x}_k$, where K is the LQR gain matrix. K is computed by solving the discrete-time algebraic Riccati equation (DARE)

To use this for tracking, at each time step k, we compute the optimal control input as

$$
u_k^* = -K(\boldsymbol{x}_k - \boldsymbol{x}_{ref}(k))
$$

Comparing the two controller, we can see that the MPC controller is able to track the reference trajectory better than the LQR controller. This is because the LQR controller is not able to account for model mismatch and assumes that the system will always behave as per the model. The MPC controller recomputes the control input at every time step and has a preview of the future reference trajectory, and hence is able to track better.

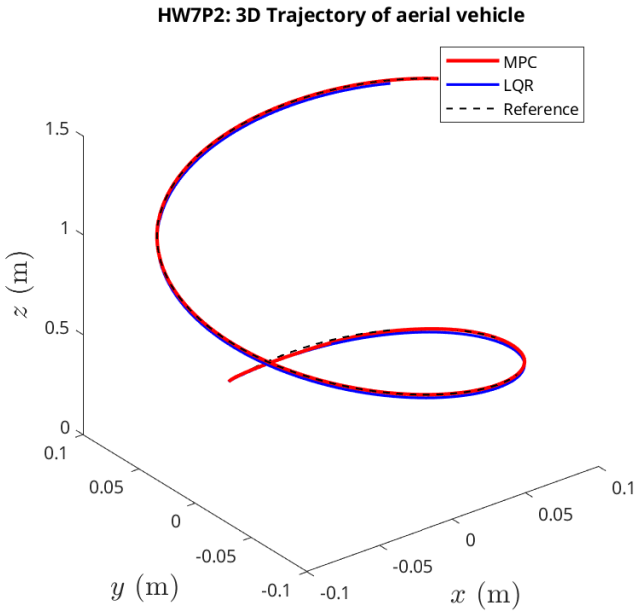


Figure 3: Comparison of MPC and LQR controller(3D)

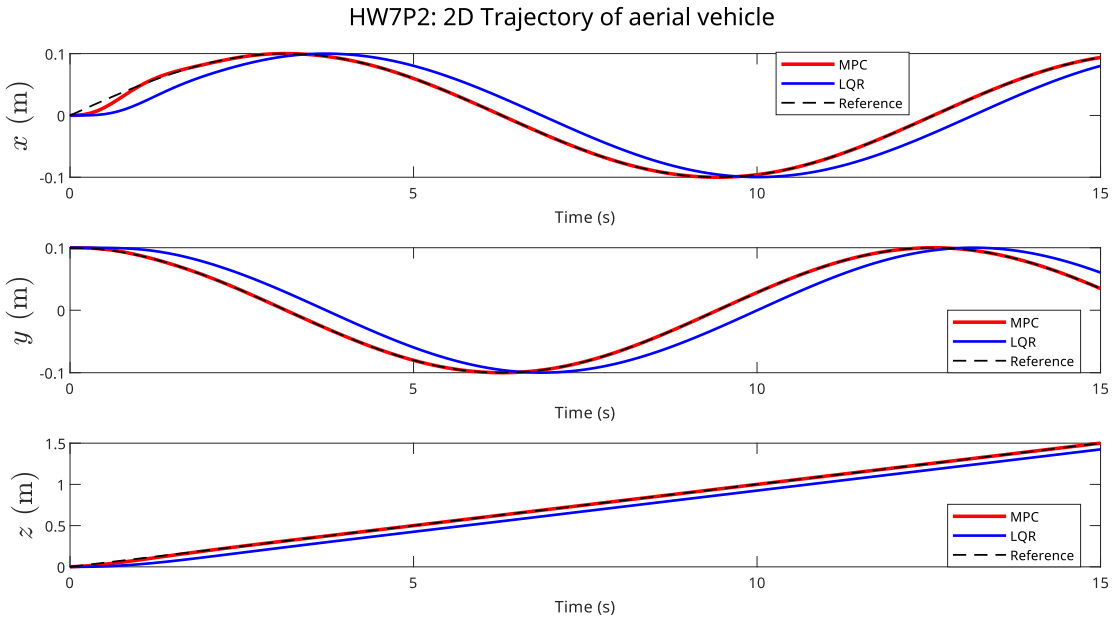


Figure 4: Comparison of MPC and LQR controller(2D)

ME599 HW7 Problem 2

```
clc; clear; close all;

% setup
n = 12; % number of states
m = 4; % number of inputs
Ts = 0.1; % sample time
rng(1); % for reproducibility
```

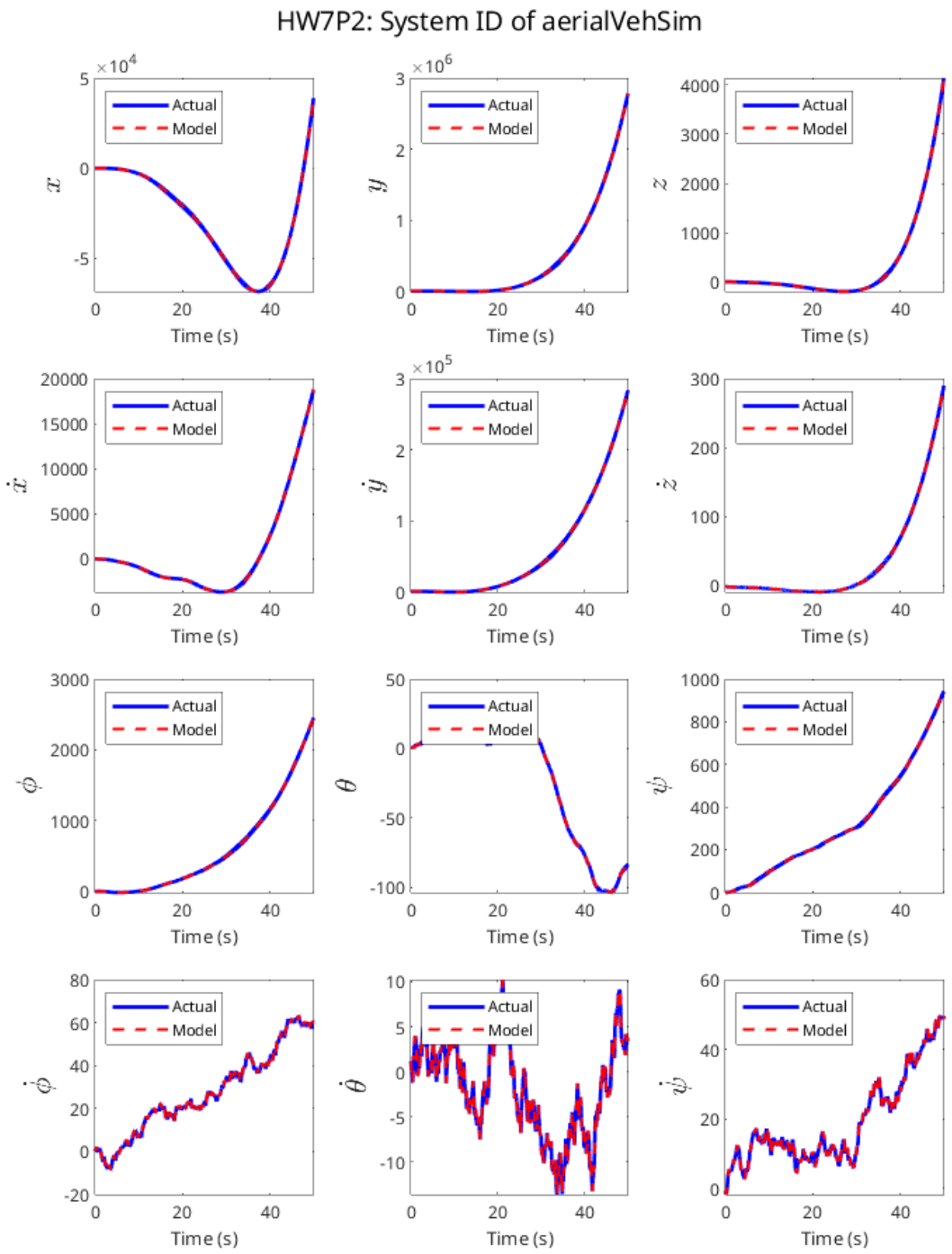
part a system ID using DMD with control

```
N = 500; % number of samples
U = randn(m, N); % input
x_0 = randn(n, 1);

X = zeros(n, N);
X_p1 = zeros(n, N);
x_k = x_0;
for i = 1:N
    X(:, i) = x_k;
    X_p1(:, i) = aerialVehSim(x_k, U(:, i), Ts);
    x_k = X_p1(:, i);
end

AB = X_p1*pinv([X;U]);
A = AB(:, 1:n);
B = AB(:, n+1:end);
format short;
fprintf('HW7 P2: Identified A:\n');
disp(round(A, 3));
fprintf('HW7 P2: Identified B:\n');
disp(round(B, 3));

% check if identified correctly
X_model = zeros(n, N);
X_model(:, 1) = x_0;
for i = 2:N
    X_model(:, i) = A*X_model(:, i-1) + B*U(:, i-1);
end

tspan = (0:N-1)*Ts;
states = ["$x$", "$y$", "$z$", "$\dot{x}$", "$\dot{y}$", "$\dot{z}$", ...
    "$\phi$", "$\theta$", "$\psi$", "$\dot{\phi}$", "$\dot{\theta}$", "$
\dot{\psi}$"];
fig = figure;
sgtitle('HW7P2: System ID of aerialVehSim');

for i = 1:n
    subplot(4, 3, i);
    plot(tspan, X(i, :), 'b', 'LineWidth', 2, 'DisplayName', 'Actual');
    hold on;
```

```
    plot(tspan, X_model(i, :), 'r--', 'LineWidth', 1.5, 'DisplayName',
'Model');
    xlabel('Time (s)');
    xlim([0 N*Ts]);
    ylabel(states(i), 'Interpreter', 'latex', 'FontSize', 16);
    legend('Location', 'northwest');
end

fig.Position(3:4) = [800 1000];
saveas(fig, 'figs/hw7p2_sysid.svg');
```

```
HW7 P2: Identified A:
  Columns 1 through 7

    1.0000         0         0    0.1000         0         0         0
         0    1.0000         0         0    0.1000         0    0.0490
         0         0    1.0000         0         0    0.1000         0
         0         0         0    1.0000         0         0         0
         0         0         0         0    1.0000         0    0.9810
         0         0         0         0         0    1.0000         0
         0         0         0         0         0         0    1.0000
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0

  Columns 8 through 12

   -0.0490         0         0   -0.0020         0
         0         0    0.0020         0         0
         0         0         0         0         0
   -0.9810         0         0   -0.0490         0
         0         0    0.0490         0         0
         0         0         0         0         0
         0         0    0.1000         0         0
    1.0000         0         0    0.1000         0
         0    1.0000         0         0    0.1000
         0         0    1.0000         0         0
         0         0         0    1.0000         0
         0         0         0         0    1.0000

HW7 P2: Identified B:
         0         0         0         0
         0    0.0010         0         0
    0.0050         0    0.0020         0
         0         0   -0.0160    0.0020
         0    0.0160         0         0
    0.1000         0         0         0
         0    0.0500         0         0
         0         0    0.0500         0
         0         0         0    0.0500
         0    1.0000         0         0
         0         0    1.0000         0
```

```
     0          0          0     1.0000
```

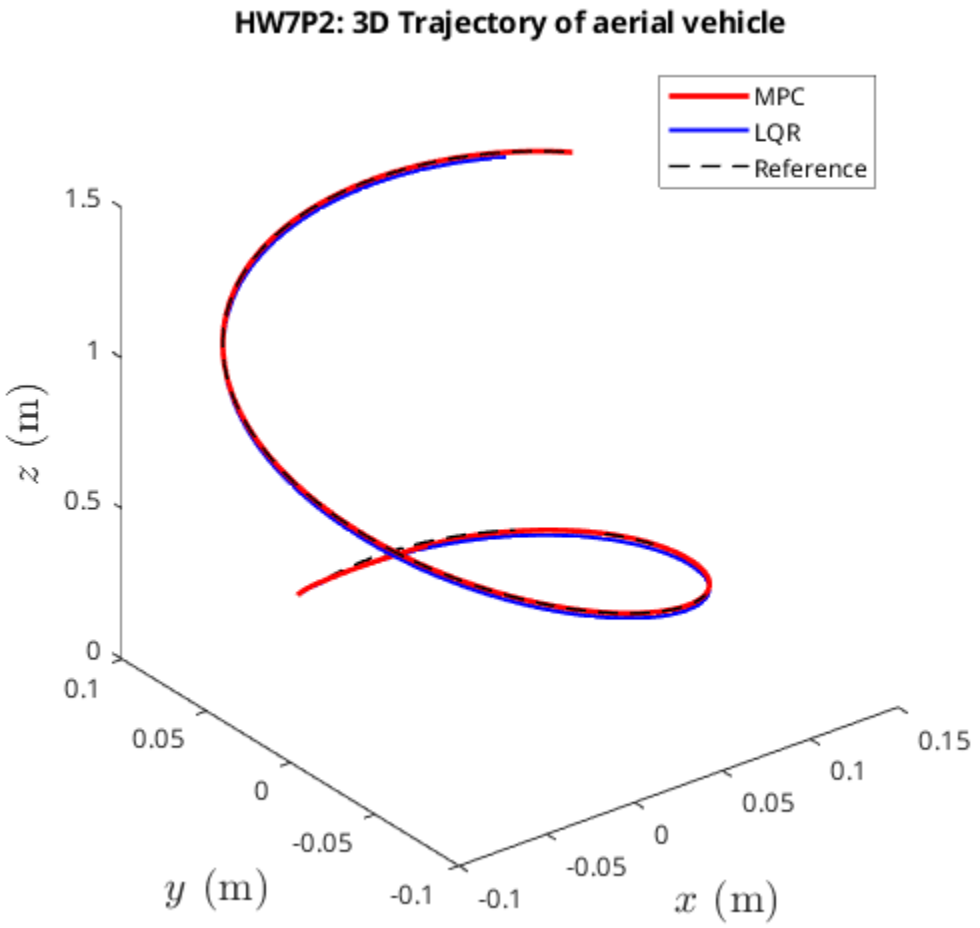


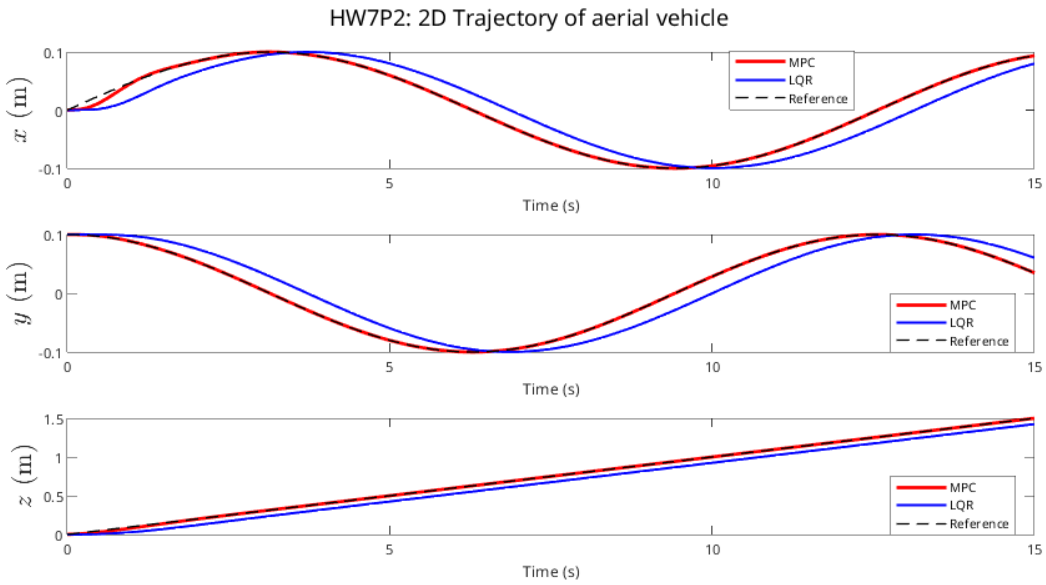
part b MPC and LQR control

```
Tend = 15; % simulation time
tsteps = 0:Ts:Tend;
Nsim = length(tsteps);
N = 30;
tsteps_ref = 0:Ts:Tend*2;
x_ref = zeros(n, length(tsteps_ref));

x_ref(1:3, :) = [
    0.1*sin(tsteps_ref/2);
    0.1*cos(tsteps_ref/2);
    0.1*tsteps_ref;
    ];

Q = [10*eye(3), zeros(3, 9);
    zeros(9, 3), zeros(9, 9)];
R = eye(m);

[coeff_MPC, A_bar] = MPCCtrl(A, B, Q, R, N);
K_lqr = dlqr(A, B, Q, R);
x_mpc_all = zeros(n, Nsim);
x_lqr_all = zeros(n, Nsim);
x0 = [0, 0.1, 0, zeros(1, 9)]';
x_mpc = x0;
x_lqr = x0;
for k=1:Nsim
    x_ref_mpc = reshape(x_ref(:, k:k+N-1), [], 1);
    u_mpc = coeff_MPC*(x_ref_mpc - A_bar*x_mpc);
    u_mpc = u_mpc(1:m);
    x_mpc = aerialVehSim(x_mpc, u_mpc, Ts);
    x_mpc_all(:, k) = x_mpc;

    u_lqr = -K_lqr*(x_lqr - x_ref(:, k));
    x_lqr = aerialVehSim(x_lqr, u_lqr, Ts);
    x_lqr_all(:, k) = x_lqr;
end

% plot
fig = figure;
plot3(x_mpc_all(1, :), x_mpc_all(2, :), x_mpc_all(3, :), 'r', 'LineWidth', 2,
'DisplayName', 'MPC');
hold on;
plot3(x_lqr_all(1, :), x_lqr_all(2, :), x_lqr_all(3, :), 'b', 'LineWidth',
1.5, 'DisplayName', 'LQR');
plot3(x_ref(1, 1:Nsim), x_ref(2, 1:Nsim), x_ref(3, 1:Nsim), 'k--',
'LineWidth', 1, 'DisplayName', 'Reference');
xlabel("$x$ (m)", 'Interpreter', 'latex', 'FontSize', 16);
ylabel("$y$ (m)", 'Interpreter', 'latex', 'FontSize', 16);
zlabel("$z$ (m)", 'Interpreter', 'latex', 'FontSize', 16);
title('HW7P2: 3D Trajectory of aerial vehicle');
legend('Location', 'best');
fig.Position(3:4) = [500 500];
```

```
saveas(fig, 'figs/hw7p2_3d.svg');

fig = figure;
sgtitle('HW7P2: 2D Trajectory of aerial vehicle');
subplot(3, 1, 1);
plot(tsteps, x_mpc_all(1, :), 'r', 'LineWidth', 2, 'DisplayName', 'MPC');
hold on;
plot(tsteps, x_lqr_all(1, :), 'b', 'LineWidth', 1.5, 'DisplayName', 'LQR');
plot(tsteps, x_ref(1, 1:Nsim), 'k--', 'LineWidth', 1, 'DisplayName',
'Reference');
xlabel('Time (s)');
ylabel("$x$ (m)", 'Interpreter', 'latex', 'FontSize', 16);
legend('Location', 'best');

subplot(3, 1, 2);
plot(tsteps, x_mpc_all(2, :), 'r', 'LineWidth', 2, 'DisplayName', 'MPC');
hold on;
plot(tsteps, x_lqr_all(2, :), 'b', 'LineWidth', 1.5, 'DisplayName', 'LQR');
plot(tsteps, x_ref(2, 1:Nsim), 'k--', 'LineWidth', 1, 'DisplayName',
'Reference');
xlabel('Time (s)');
ylabel("$y$ (m)", 'Interpreter', 'latex', 'FontSize', 16);
legend('Location', 'best');

subplot(3, 1, 3);
plot(tsteps, x_mpc_all(3, :), 'r', 'LineWidth', 2, 'DisplayName', 'MPC');
hold on;
plot(tsteps, x_lqr_all(3, :), 'b', 'LineWidth', 1.5, 'DisplayName', 'LQR');
plot(tsteps, x_ref(3, 1:Nsim), 'k--', 'LineWidth', 1, 'DisplayName',
'Reference');
xlabel('Time (s)');
ylabel("$z$ (m)", 'Interpreter', 'latex', 'FontSize', 16);
legend('Location', 'best');

fig.Position(3:4) = [1000 500];
saveas(fig, 'figs/hw7p2_2d.svg');

function [u_coeff, A_bar] = MPCCtrl(A,B,Q,R,N)
nx = size(A,2);
nu = size(B,2);

% initialize matrices
A_bar = zeros(N*nx,nx);
B_bar = zeros(N*nx,N*nu);
Q_bar = zeros(N*nx,N*nx);
R_bar = zeros(N*nu,N*nu);

% Compute first row of A_bar then use it to initialize the rest
A_bar(1:nx,:) = A;
for i = 2:N
    A_bar((i-1)*nx+1 : i*nx, :) = A_bar((i-2)*nx+1:(i-1)*nx,:)*A;
end

% Compute first column of B_bar then use it to initialize the rest
```

```
for i = 1:N
    B_bar((i-1)*nx+1 : i*nx, 1:nu) = A^(i-1)*B;
end
for i = 2:N
    zero_rows = (i-1)*nx;
    zero_cols = nu;
    B_bar(:, (i-1)*nu+1 : i*nu) = [zeros(zero_rows,zero_cols); B_bar(1:end-
zero_rows, 1:nu)];
end

% Compute Q_bar
for i = 1:N
    Q_bar((i-1)*nx+1 : i*nx, (i-1)*nx+1 : i*nx) = Q;
end

% Compute R_bar
for i = 1:N
    R_bar((i-1)*nu+1 : i*nu, (i-1)*nu+1 : i*nu) = R;
end

u_coeff = inv(B_bar'*Q_bar*B_bar + R_bar)*B_bar'*Q_bar;

end
```


HW7P2: 3D Trajectory of aerial vehicle
MPC
LQR
Reference
1.5
1
0.5
0
z (m)
0.1
0.05
0
-0.05
-0.1
y (m)
-0.1
-0.05
0
0.05
0.1
0.15
x (m)


HW7P2: 2D Trajectory of aerial vehicle
x (m)
y (m)
z (m)
MPC
LQR
Reference
Time (s)
0
5
10
15
0.1
0
-0.1
1.5
1
0.5

Published with MATLAB® R2023b