EEE 446/546 Summer 2024
Computational Assignment

I affirm that I worked on this assignment myself and wrote the codes with my own effort. While I received some guidance from the internet and AI, the code and algorithms are my own work.

Güneş Akbaş

Problem 1: Value Iteration/Policy Iteration and Q-Learning [50 Points]

a) Let $X = \{B, G\}$, $U = \{0, 1\}$, where X denotes whether a fading channel is in a good state ($x = G$) or a bad state ($x = B$). There exists an encoder who can either try to use the channel ($u = 1$) or not use the channel ($u = 0$).

The goal of the encoder is to send information across the channel. The encoder's per-stage cost (to be minimized) is given by:

$$c(x, u) = -1\{x = G, u = 1\} + \eta u$$

for some $\eta \in \mathbb{R}$ to be specified below. When the channel is good and the input is active, information is transmitted.

If you view this as a maximization problem, you can see that the goal is to maximize information transmission efficiency subject to a cost involving an attempt to use the channel; the model can be made more complicated but the idea is that when the channel state is good, $u = 1$ can represent a channel input which contains data to be transmitted and $u = 0$ denotes that the channel is not used. When $u = 1$ and $x = G$, the channel is utilized successfully.

For many channels with memory, the input also impacts the channel state. Suppose that the transition kernel is given by:

$$\begin{aligned}
&P(x_{t+1} = G \mid x_t = G, u_t = 1) = 0.1, && P(x_{t+1} = B \mid x_t = G, u_t = 1) = 0.9,\\
&P(x_{t+1} = G \mid x_t = G, u_t = 0) = 0.95, && P(x_{t+1} = B \mid x_t = G, u_t = 0) = 0.05,\\
&P(x_{t+1} = G \mid x_t = B, u_t = 1) = 0.75, && P(x_{t+1} = B \mid x_t = B, u_t = 1) = 0.25,\\
&P(x_{t+1} = G \mid x_t = B, u_t = 0) = 0.5, && P(x_{t+1} = B \mid x_t = B, u_t = 0) = 0.5.
\end{aligned}$$

We will consider either a discounted cost criterion for some $\beta \in (0, 1)$ (you can fix an arbitrary value)

$$\inf_{\gamma \in \Gamma_A} E_{\gamma,x}\left[\sum_{t=0}^{\infty} \beta^t c(x_t, u_t)\right]$$

or the average cost criterion

$$\inf_{\gamma \in \Gamma_A} \limsup_{T \to \infty} \frac{1}{T} E_{\gamma,x}\left[\sum_{t=0}^{T-1} c(x_t, u_t)\right].$$

a) Using Matlab or some other program, obtain a solution to the problem given above in (1) through the following:

(i) [15 Points] Value Iteration. Take some fixed $\beta \in (0,1)$ of your choice. Consider $\eta = 0.9$, $\eta = 0.7$, and $\eta = 0.01$. Interpret the optimal solution for these different values of η, in view of the application.
Here, we choose $\beta = \frac{1}{2}$ and define η for a single value: Moreover,algorithm stops for certain tolerance level:

$$\|v_n(x) - v_{n-1}(x)\|_\infty = \sup_{x \in X} |v_n(x) - v_{n-1}(x)| < \epsilon$$

β =0.5	$\eta = 0.9$	$\eta = 0.7$	$\eta = 0.01$
$V(G)$	-0.125	-0.375	-1.3008
$V(B)$	-0.041666	-0.125	-0.54604

Table 1: Values of $V(G)$ and $V(B)$ for different η values

```
% Define the global variables eta and beta
global eta beta;
eta = 0.7;
beta = 0.5;

% Define the  indicator function
function z = I(x, u)
    % Check if x is 'G' and u is 1
    if strcmp(x, 'G') && u == 1
        z = 1;
    else
        z = 0;
    end
end

% Define the function c
function t = c(x, u)
    % Access the global variable eta
    global eta;
    t = -I(x, u) + eta * u;
end

% Define the transition probabilities
function P = get_transition_probabilities()
    P = containers.Map;
    P('GG1') = 0.1;  % P(xt+1 = G | xt = G, ut = 1)
    P('GB1') = 0.9;  % P(xt+1 = B | xt = G, ut = 1)
    P('GG0') = 0.95; % P(xt+1 = G | xt = G, ut = 0)
    P('GB0') = 0.05; % P(xt+1 = B | xt = G, ut = 0)
    P('BG1') = 0.75; % P(xt+1 = G | xt = B, ut = 1)
    P('BB1') = 0.25; % P(xt+1 = B | xt = B, ut = 1)
    P('BG0') = 0.5;  % P(xt+1 = G | xt = B, ut = 0)
    P('BB0') = 0.5;  % P(xt+1 = B | xt = B, ut = 0)
end

% Define the function T which computes v_n(x)
function v_n = T(v_n_minus_1)
    % Access the global variable beta
    global beta;

    % Get the transition probabilities
    P = get_transition_probabilities();

    % Possible values for x and u
    x_values = {'G', 'B'};
    u_values = [0, 1];

    % Initialize the output
    v_n = containers.Map();

    % Compute v_n(x) for each value of x
    for i = 1:length(x_values)
```

```
        x = x_values{i};
        min_value = inf;  % Initialize with infinity

        for j = 1:length(u_values)
            u = u_values(j);

            % Compute the conditional expectation
            if strcmp(x, 'G')
                expected_value = P(['GG', num2str(u)]) * v_n_minus_1('G') + P(['GB',
                    num2str(u)]) * v_n_minus_1('B');
            else
                expected_value = P(['BG', num2str(u)]) * v_n_minus_1('G') + P(['BB',
                    num2str(u)]) * v_n_minus_1('B');
            end

            % Compute the value of c(x, u) + beta * E[v_{n-1}(x_1) | x_0 = x, u_0 = u]
            value = c(x, u) + beta * expected_value;

            % Update the minimum value over u
            if value < min_value
                min_value = value;
            end
        end

        % Store the result in v_n
        v_n(x) = min_value;
    end
end

% Initialize v_0(x) to 0 for both x = 'G' and x = 'B'
v_0 = containers.Map({'G', 'B'}, [0, 0]);

% Set tolerance level and initialize variables
tolerance = 1e-6;
difference = inf;
iteration = 0;

% Iterative computation
while difference > tolerance
    iteration = iteration + 1;
    v_n = T(v_0);

    % Calculate the supremum norm ||v_n(x) - v_n-1(x)||
    diff_G = abs(v_n('G') - v_0('G'));
    diff_B = abs(v_n('B') - v_0('B'));
    difference = max(diff_G, diff_B);

    % Display the current iteration and difference
    fprintf('Iteration %d: difference = %e\n', iteration, difference);

    % Update v_0 for the next iteration
    v_0 = v_n;
end

% Display the final results
disp(['Final v(G) = ', num2str(v_n('G'))]);
disp(['Final v(B) = ', num2str(v_n('B'))]);
```

(ii) [15 Points] Policy Iteration. With the same β as above, work again with each of the following: $\eta = 0.9$, $\eta = 0.7$, and $\eta = 0.01$. Notice that there exist 4 stationary deterministic policies first we order them then replace inside of :

$$W = (I - \beta P_\gamma)^{-1} c_\gamma$$

Then choose the policy with the minimum value over W(G) and W(B):

$\beta = 0.5$	$\eta = 0.9$	$\eta = 0.7$	$\eta = 0.01$
$W(G)$	-0.125	-0.375	-1.3008
$W(B)$	-0.041666	-0.125	-0.54604

Table 2: Values of $W(G)$ and $W(B)$ for different η values

$\beta = 0.5$	$\eta = 0.9$	$\eta = 0.7$	$\eta = 0.01$
$\gamma^*(G)$	1	1	1
$\gamma^*(B)$	0	0	1

Table 3: Values of $\gamma^*(G)$ and $\gamma^*(B)$ for different η values

```
% Define the global variables eta and beta
global eta beta;
eta = 0.7;
beta = 0.5;

% Define the function f
function z = f(x, u)
    % Check if x is 'G' and u is 1
    if strcmp(x, 'G') && u == 1
        z = 1;
    else
        z = 0;
    end
end

% Define the function c
function t = c(x, u)
    % Access the global variable eta
    global eta;
    t = -f(x, u) + eta * u;
end

% Define the transition probabilities
function P = get_transition_probabilities()
    P = containers.Map;
    P('GG1') = 0.1;  % P(xt+1 = G | xt = G, ut = 1)
    P('GB1') = 0.9;  % P(xt+1 = B | xt = G, ut = 1)
    P('GG0') = 0.95; % P(xt+1 = G | xt = G, ut = 0)
    P('GB0') = 0.05; % P(xt+1 = B | xt = G, ut = 0)
    P('BG1') = 0.75; % P(xt+1 = G | xt = B, ut = 1)
    P('BB1') = 0.25; % P(xt+1 = B | xt = B, ut = 1)
    P('BG0') = 0.5;  % P(xt+1 = G | xt = B, ut = 0)
    P('BB0') = 0.5;  % P(xt+1 = B | xt = B, ut = 0)
end

% Define the function to get the transition matrix for a policy
function P_gamma = get_transition_matrix(policy)
    P = get_transition_probabilities();

    % Initialize the transition matrix
    P_gamma = zeros(2, 2);

    % Define the policy for each state
    if policy == 0
        % Always action 0 for both states
        gamma_B = 0;
        gamma_G = 0;
    elseif policy == 1
        % Action 0 for 'B' and action 1 for 'G'
        gamma_B = 0;
        gamma_G = 1;
    elseif policy == 2
```

```
        % Action 1 for 'B' and action 0 for 'G'
        gamma_B = 1;
        gamma_G = 0;
    else
        % Always action 1 for both states
        gamma_B = 1;
        gamma_G = 1;
    end

    % Fill the transition matrix based on the policy
    P_gamma(1, 1) = P(['BB', num2str(gamma_B)]);
    P_gamma(1, 2) = P(['BG', num2str(gamma_B)]);
    P_gamma(2, 1) = P(['GB', num2str(gamma_G)]);
    P_gamma(2, 2) = P(['GG', num2str(gamma_G)]);
end

% Define the function to get the cost vector for a policy
function c_gamma = get_cost_vector(policy)
    % Initialize the cost vector
    c_gamma = zeros(2, 1);

    % Define the policy for each state
    if policy == 0
        % Always action 0 for both states
        gamma_B = 0;
        gamma_G = 0;
    elseif policy == 1
        % Action 0 for 'B' and action 1 for 'G'
        gamma_B = 0;
        gamma_G = 1;
    elseif policy == 2
        % Action 1 for 'B' and action 0 for 'G'
        gamma_B = 1;
        gamma_G = 0;
    else
        % Always action 1 for both states
        gamma_B = 1;
        gamma_G = 1;
    end

    % Fill the cost vector based on the policy
    c_gamma(1) = c('B', gamma_B);
    c_gamma(2) = c('G', gamma_G);
end

% Perform policy iteration
num_policies = 4;
best_value = inf;
best_policy = -1;

for policy = 0:num_policies-1
    % Get the transition matrix and cost vector for the current policy
    P_gamma = get_transition_matrix(policy);
    c_gamma = get_cost_vector(policy);

    % Calculate W_0 = (I - beta * P_gamma)^-1 * c_gamma
    W_0 = inv(eye(2) - beta * P_gamma)*c_gamma;

    % Calculate the value of the policy
    policy_value = sum(W_0);

    % Check if this policy has the smallest value so far
    if policy_value < best_value
        best_value = policy_value;
        best_policy = policy;
    end

    % Display the current policy and its value
```

```
    fprintf('Policy %d: W_0(B) = %f, W_0(G) = %f, Value = %f\n', policy, W_0(1), W_0(2)
        , policy_value);
end

% Display the best policy
disp(['Best policy: ', num2str(best_policy), ' with value: ', num2str(best_value)]);
```

(iii) [20 Points] Q-Learning (see (9.3) in the Lecture Notes). Try only $\eta = 0.7$. Note that a common way to pick α_t coefficients in the Q-learning algorithm is to take for every (x, u) pair:

$$\alpha_t(x, u) = \frac{1}{1 + \sum_{k=0}^{t} 1\{x_k = x, u_k = u\}}$$

Compare your solutions (obtained via different methods).

$\eta = 0.7$	Iteration = 100	Iteration = 1000	Iteration = 10000
$Q(G, 1)$	-0.33121	-0.3622	-0.37358
$Q(B, 0)$	-0.10139	-0.11386	-0.12572

Table 4: Values of $Q(G, 1)$ and $Q(B, 0)$ for different iterations and η values

Notice that it converges solution of other algorithms finding but when we start algorithm every time it gives different but close values due to nature of stochastic approximation.

```
% Define the global variables eta and beta
global eta beta;
eta = 0.7;
beta = 0.5;

% Define the function f
function z = f(x, u)
    % Check if x is 'G' and u is 1
    if strcmp(x, 'G') && u == 1
        z = 1;
    else
        z = 0;
    end
end

% Define the function c
function t = c(x, u)
    % Access the global variable eta
    global eta;
    t = -f(x, u) + eta * u;
end

% Define the transition probabilities
function P = get_transition_probabilities()
    P = containers.Map;
    P('GG1') = 0.1;  % P(xt+1 = G | xt = G, ut = 1)
    P('GB1') = 0.9;  % P(xt+1 = B | xt = G, ut = 1)
    P('GG0') = 0.95; % P(xt+1 = G | xt = G, ut = 0)
    P('GB0') = 0.05; % P(xt+1 = B | xt = G, ut = 0)
    P('BG1') = 0.75; % P(xt+1 = G | xt = B, ut = 1)
    P('BB1') = 0.25; % P(xt+1 = B | xt = B, ut = 1)
    P('BG0') = 0.5;  % P(xt+1 = G | xt = B, ut = 0)
    P('BB0') = 0.5;  % P(xt+1 = B | xt = B, ut = 0)
end

% Define the function to get the next state based on transition probabilities
function next_state = get_next_state(current_state, action)
    P = get_transition_probabilities();
    if strcmp(current_state, 'G')
```

```
        if action == 0
            if rand < P('GG0')
                next_state = 'G';
            else
                next_state = 'B';
            end
        else
            if rand < P('GG1')
                next_state = 'G';
            else
                next_state = 'B';
            end
        end
    else
        if action == 0
            if rand < P('BG0')
                next_state = 'G';
            else
                next_state = 'B';
            end
        else
            if rand < P('BG1')
                next_state = 'G';
            else
                next_state = 'B';
            end
        end
    end
end

% Initialize Q-values
Q = containers.Map({'B0', 'B1', 'G0', 'G1'}, [0, 0, 0, 0]);

% Initialize state-action count
N = containers.Map({'B0', 'B1', 'G0', 'G1'}, [0, 0, 0, 0]);

% Number of iterations
num_iterations = 100000;

% Initial state
states = {'B', 'G'};
current_state = states{randi(2)};

% Q-learning iterations
for t = 1:num_iterations
    % Select a random action (0 or 1)
    action = randi(2) - 1;

    % Get the key for the current state-action pair
    key = [current_state, num2str(action)];

    % Get the next state based on the current state and action
    next_state = get_next_state(current_state, action);

    % Get the cost for the current state-action pair
    cost = c(current_state, action);

    % Find the minimum Q-value for the next state
    if strcmp(next_state, 'G')
        min_next_Q = min(Q('G0'), Q('G1'));
    else
        min_next_Q = min(Q('B0'), Q('B1'));
    end

    % Calculate the learning rate
    N(key) = N(key) + 1;
    alpha = 1 / (1 + N(key));

```

```
    % Update the Q-value
    Q(key) = Q(key) + alpha * (cost + beta * min_next_Q - Q(key));

    % Move to the next state
    current_state = next_state;
end

% Display the final Q-values
disp('Final Q-values:');
disp(['Q(B,0) = ', num2str(Q('B0'))]);
disp(['Q(B,1) = ', num2str(Q('B1'))]);
disp(['Q(G,0) = ', num2str(Q('G0'))]);
disp(['Q(G,1) = ', num2str(Q('G1'))]);
```

Problem 2: Linear Programming for Average Cost Stochastic Control [20 Points]

For the model given in Problem 1a), consider the criterion given in (2). Apply the convex analytic method, by solving the corresponding linear program, to find the optimal policy? In Matlab, the command `linprog` can be used to solve linear programming problems. See (7.40) in the lecture notes.
In here we follow exactly same steps in 7.40(lecture notes) but to solve linear program by matlab we did small change normally one of the constraint is:

$$T * V = b$$

On the other hand, the value of b as a vector, depend on V so we adjust T and take b=0
The optimal solution V^* is:

$$V^* = \begin{bmatrix} 0 \\ 0.6429 \\ 0.3571 \\ 0 \end{bmatrix}$$

The minimum value of the objective function $c^T V$ is:

$$c^T V = -0.1071$$

```
% Define the global variable eta
global eta;
eta = 0.7;

% Define the ordering of (x, u)
% (G,0) = 1
% (B,0) = 2
% (G,1) = 3
% (B,1) = 4

% Define the cost function c(x, u)
function t = c_func(x, u)
    global eta;
    t = -f(x, u) + eta * u;
end

function z = f(x, u)
    if strcmp(x, 'G') && u == 1
        z = 1;
    else
        z = 0;
    end
end

% Define the cost vector c (4x1)
```

```
c = zeros(4, 1);
c(1) = c_func('G', 0); % c(G, 0)
c(2) = c_func('B', 0); % c(B, 0)
c(3) = c_func('G', 1); % c(G, 1)
c(4) = c_func('B', 1); % c(B, 1)

% Define the transition probabilities
P = containers.Map;
P('GG1') = 0.1;  % P(xt+1 = G | xt = G, ut = 1)
P('GB1') = 0.9;  % P(xt+1 = B | xt = G, ut = 1)
P('GG0') = 0.95; % P(xt+1 = G | xt = G, ut = 0)
P('GB0') = 0.05; % P(xt+1 = B | xt = G, ut = 0)
P('BG1') = 0.75; % P(xt+1 = G | xt = B, ut = 1)
P('BB1') = 0.25; % P(xt+1 = B | xt = B, ut = 1)
P('BG0') = 0.5;  % P(xt+1 = G | xt = B, ut = 0)
P('BB0') = 0.5;  % P(xt+1 = B | xt = B, ut = 0)

% Define the T transition matrix (2x4)
T = zeros(2, 4);
% First row corresponds to z = G
T(1, 1) = P('GG0')-1; % P(G|(G, 0))
T(1, 2) = P('BG0'); % P(G|(B, 0))
T(1, 3) = P('GG1')-1; % P(G|(G, 1))
T(1, 4) = P('BG1'); % P(G|(B, 1))
% Second row corresponds to z = B
T(2, 1) = P('GB0'); % P(B|(G, 0))
T(2, 2) = P('BB0')-1; % P(B|(B, 0))
T(2, 3) = P('GB1'); % P(B|(G, 1))
T(2, 4) = P('BB1')-1; % P(B|(B, 1))

% Define the b vector (2x1)
% b_1 = V1 + V3
% b_2 = V2 + V4
b = [0; 0];

% Objective function: minimize (c^T) * V
c = [c(1); c(2); c(3); c(4)];

% Constraints: T * V = b and V >= 0 and V1 + V2 + V3 + V4 = 1
Aeq = [T; 1 1 1 1];
beq = [b; 1];

% Lower bounds for V
lb = [0; 0; 0; 0];

% Use linprog to solve the problem
[V, fval, exitflag, output] = linprog(c, [], [], Aeq, beq, lb, []);

% Display the results
disp('Optimal solution (V):');
disp(V);
disp('Minimum value of the objective function (c^T V):');
disp(fval);
```

Problem 3: The Kalman Filter [30 Points]

Let a linear system driven by Gaussian noise be given by the following:

$$x_{t+1} = Ax_t + w_t,$$
$$y_t = Cx_t + v_t$$

Here

$$A = \begin{bmatrix} 1.2 & 1 & 0 & 0 \\ 0 & 1.2 & 1 & 0 \\ 0 & 0 & 1.2 & 1 \\ 0 & 0 & 0 & 1.5 \end{bmatrix}, C = \begin{bmatrix} 2 & 0 & 0 & 0 \end{bmatrix},$$

and the i.i.d. noise processes satisfy $w_t \sim \mathcal{N}(0, I)$, $v_t \sim \mathcal{N}(0, 1)$. The initial state is also zero-mean Gaussian and suppose that $\Sigma_{0|-1} = I$ (here, we use the notation in Chapter 6 of the Lecture notes).

a) Write down the Kalman Filter update equations (including the associated Riccati recursions for the covariance matrix updates).

$$\Sigma_{t+1|t} = A\Sigma_{t|t-1}A^T + W - (A\Sigma_{t|t-1}C^T)(C\Sigma_{t|t-1}C^T + V)^{-1}(C\Sigma_{t|t-1}A^T)$$

In here we know A,C,V and the first step.

$$m_{t+1} = Am_t + A\Sigma_{t|t-1}C^T(C\Sigma_{t|t-1}C^T + V)^{-1}(y_t - Cm_t)$$

Furthermore, we know that

$$m_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

so y_t as random vector and also:

$$\tilde{m}_{t-1} = A^{-1}m_t$$

b) By simulating the above system in Matlab (or any other program), run the Kalman Filter from $t = 0$ until $T = 500$. Plot x_t, $\tilde{m}_t$ and $x_t - \tilde{m}_t$.

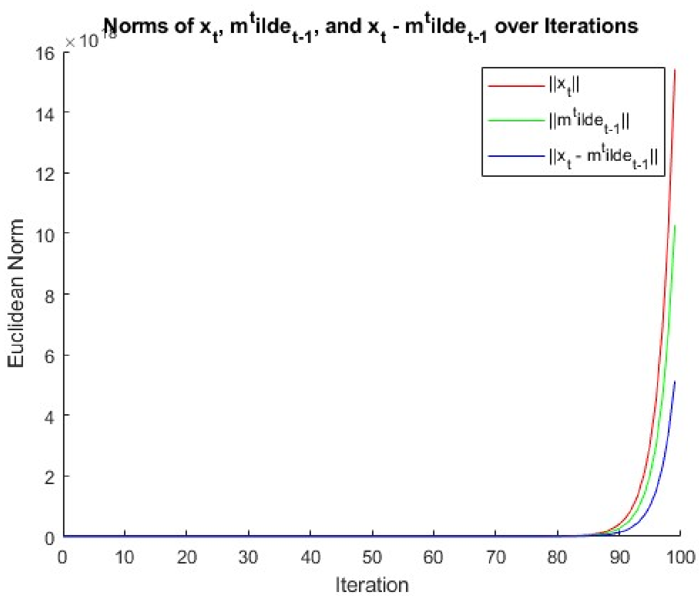

Figure 1: Plot of the norms of x_t, $m^{t\tilde{-}1}$, and $x_t - m^{t\tilde{-}1}$ over 100 iterations

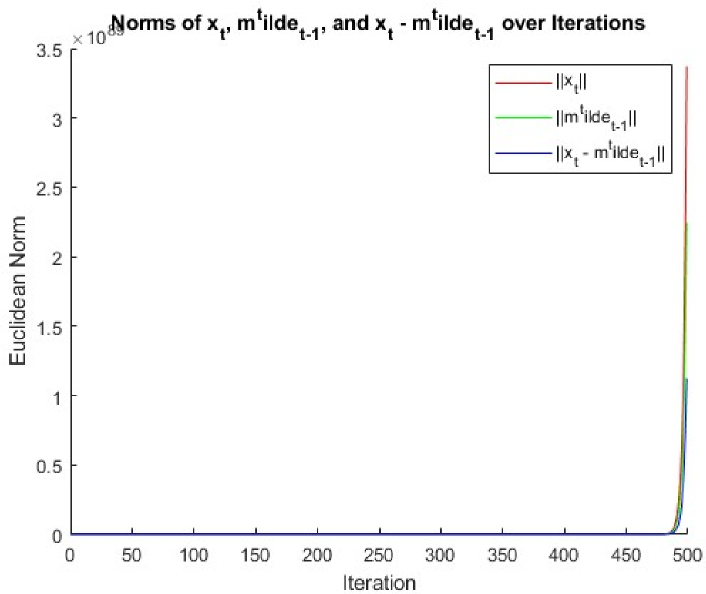

Figure 2: Plot of the norms of x_t, $m^{t\tilde{-}1}$, and $x_t - m^{t\tilde{-}1}$ over 500 iterations

```matlab
% Define matrix A
A = [1.2 1 0 0;
     0 1.2 1 0;
     0 0 1.2 1;
     0 0 0 1.5];

% Define matrix C
C = [2 0 0 0];

% Define W as a 4x4 identity matrix
W = eye(4);

% Define V as a scalar
V = 1;

% Time steps
iterations = 100;

% Initialize sigma, m, and m_tilde
sigma_t = eye(4); % sigma_0
m_t = zeros(4, iterations+1); % m_0 is a 4x1 zero vector
m_tilde_t_minus1 = zeros(4, iterations+1); % m_tilde

% Initialize x_t and y_t
x_t = zeros(4, iterations+1); % include t=0
y_t = zeros(1, iterations+1); % include t=0

% Generate initial w_t and v_t
x_t(:, 1) = randn(4, 1); % x_0 is same as w_0

% Loop for t = 0 to iterations
for i = 1:iterations
    % Generate w_t and v_t for current step
    w_t = randn(4, 1);
    v_t = randn(1, 1);

    % Calculate x_t+1
    x_t(:, i+1) = A * x_t(:, i) + w_t;

    % Calculate y_t
    y_t(i) = C * x_t(:, i) + v_t;

    % Update sigma_t+1
    sigma_t = A * sigma_t * A' + W - (A * sigma_t * C' * inv(C * sigma_t * C' + V) * C
        * sigma_t * A');

    % Update m_t+1
    m_t(:, i+1) = A * m_t(:, i) + A * sigma_t * C' * inv(C * sigma_t * C' + V) * (y_t(i
        ) - C * m_t(:, i));

    % Calculate m_tilde_t-1
    m_tilde_t_minus1(:, i) = inv(A) * m_t(:, i);
end

% Display the final values at t=20
disp('Final value of x_t :');
disp(x_t(:, iterations+1));
disp('Final value of m_tilde_t_minus1 :');
disp(m_tilde_t_minus1(:, iterations));
disp('Final value of x_t - m_tilde_t_minus1 :');
disp(x_t(:, iterations+1) - m_tilde_t_minus1(:, iterations));

% Calculate norms for plotting
norm_x_t = vecnorm(x_t(:, 1:iterations), 2, 1);
norm_m_tilde_t_minus1 = vecnorm(m_tilde_t_minus1(:, 1:iterations), 2, 1);
norm_diff = vecnorm(x_t(:, 1:iterations) - m_tilde_t_minus1(:, 1:iterations), 2, 1);

```

```
% Plot the norms
figure;
hold on;
plot(0:iterations-1, norm_x_t, 'r', 'DisplayName', '||x_t||');
plot(0:iterations-1, norm_m_tilde_t_minus1, 'g', 'DisplayName', '||m^tilde_{t-1}||');
plot(0:iterations-1, norm_diff, 'b', 'DisplayName', '||x_t - m^tilde_{t-1}||');
xlabel('Iteration');
ylabel('Euclidean Norm');
title('Norms of x_t, m^tilde_{t-1}, and x_t - m^tilde_{t-1} over Iterations');
legend;
hold off;
```

c) Do the Riccati recursions converge to a limit; is the limit unique? Explain why or why not. By Matlab (or another program) verify your answer. For uniqueness, you can take various initial conditions and study whether the recursions converge to the same limit.