# MECH 6326 - Optimal Control and Dynamic Programming
# Homework 4

Jonas Wagner
jonas.wagner@utdallas.edu

2023, April 13th

## Contents

# Problem 1 Robot navigation in an uncertain environment

**PROBLEM:**

You are designing a controller to navigate a robot through a cluttered and uncertain environment. We use a very simplified 2D grid model, where the robot starts in the bottom-left corner, and must move to the top-right corner, and can move either up or to the right at each step. Due to unpredictable robot-environment interaction, if the robot moves in the same direction as in the previous time-step, there is a higher probability of moving in the desired direction than if it tries to change direction. If the direction selected is the same as in the previous step, the robot lands in the desired state with probability 0.8 and lands in the undesired state with probability 0.2. If the direction selected is different from in the previous step, the robot lands in the intended state with probability 0.6 and lands in the unintended state with probability 0.4. The first move is treated as though there was a move in the same direction previously.
The environment is a $41 \times 41$ grid. Assume that at each time step, the robot knows its location in the grid and can use the information for feedback. The environment has a number of obstacles whose locations are defined in the file *robot_nav.mat* available on eLearning. Hitting an obstacle or the environment boundary results in a crash, and the mission is failed (i.e, you can treat the obstacle locations as absorbing states).

**SOLUTION:**

A summary of the problem is as follows:
Let the state be the robot position:

$$x_k \in \mathcal{X} = \{1, \dots, 41\}^2 \subset \mathbb{Z}^2$$

Obstacles exist within that result in a crash (along with the boundary), $\mathcal{X}_{obstacles}$, thus a safe region can be denoted as $\mathcal{X}_{safe} = \mathcal{X} \backslash \mathcal{X}_{obstacles}$. The initial state is in the SW corner: $x_0 = \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]$. Additionally, the goal state is in the NE corner: $x_{goal} = \left[\begin{smallmatrix}41\\41\end{smallmatrix}\right]$.
Let the input be the desired heading:

$$u_k \in \mathcal{U} = \{\mathrm{N}, \mathrm{E}\} = \left\{ \begin{bmatrix} 0 \\ +1 \end{bmatrix}, \begin{bmatrix} +1 \\ 0 \end{bmatrix} \right\}$$

The update equation for the position is as follows:

$$x_{k+1} = f(x_k, u_k) = x_k + w_k(u_{k-1})u_k, \quad w_k(u_k, u_{k-1}) \sim \begin{cases} \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix} & u_k = u_{k-1} \\ \begin{bmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{bmatrix} & u_k \neq u_{k-1} \end{cases}$$

You can model this as either augmenting the system with a previous time-step input to perform this simulation. Alternatively, the objective

## 1a)

**PROBLEM:**

Determine and plot the optimal policy and value function for the stage cost $\forall_{k=1,\dots,N}$

$$g_k(x) = \begin{cases} 1 & x = x_{goal} \\ -1 & x \in \mathcal{X}_{obstacles} \\ 0 & \text{otherwise} \end{cases}$$

Comment on your result.

**SOLUTION:**

See MATLAB code.
The value function and optimal policy plots are shown in Figure 1 and Figure 2 respectively.

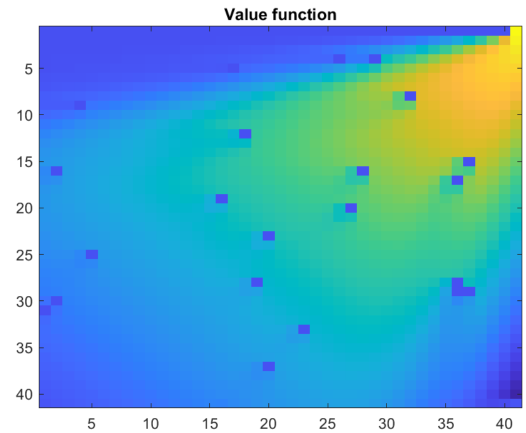


Figure 1: Value function for problem 1.

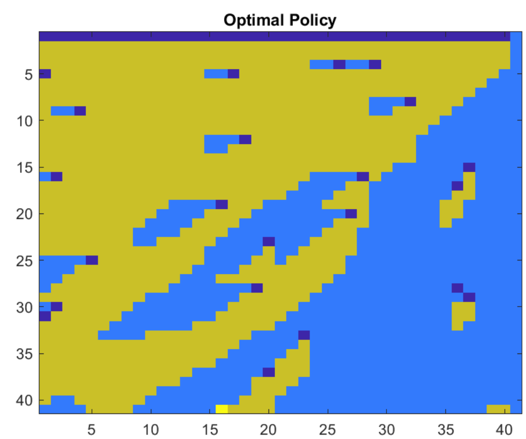


Figure 2: Optimal policy for problem 1. (yellow is East and blue is North)

The result is pretty much to be expected and makes sense logically.

## 1b)

Simulate the system to estimate (via either distribution propagation or Monte Carlo) the success rate and plot a sample trajectory of a successful mission.
The result for the controller seems to be either really terrible or the implimenation itself remains incorrect as the success rate is only around 13%. A sample of the successful trajectory is provided in Figure 3.

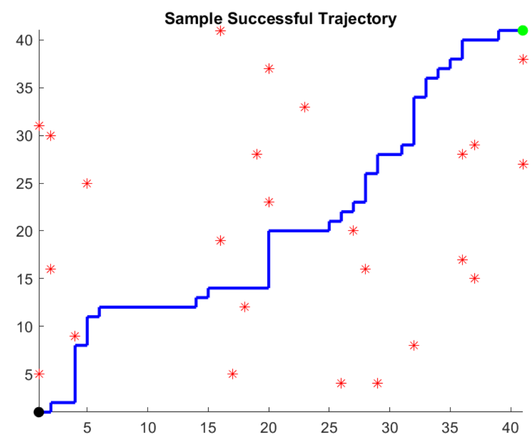


Figure 3: Sample successful trajectory for problem 1.

**Linear Quadratic Problems**

## Problem 2 Non-zero Mean Disturbances

**PROBLEM:**

Use dynamic programming to derive the optimal const functions and policies for a linear quadratic problem with non-zero mean distrubances. The dynamcis are

$$x_{k+1} = A_k x_k + B_k u_k + w_k, \quad \forall_{k=0,\dots,N-1}$$

where $\mathbf{E}[w_k] = \bar{w}_k$ and $\mathbf{E}[w_k w_k^T] = W_k$, and the cost function is the linear quadradic cost:

$$\sum_{k=0}^{N-1} (x_k^T Q_k x_k + u_k^T R_k u_k) + x_N^T Q_N x_N$$

Compute the optimal policy and optimal cost function for the problem instance with constant problem data $\forall_{k=0,\dots,N-1}$

$$A_k = \begin{bmatrix} 0.4 & -0.3 & 0 & 0.6 \\ 0.1 & -0.7 & 0.2 & 0 \\ 0.5 & 0.2 & -0.8 & 0.1 \\ 0 & 0.3 & -0.4 & 0.9 \end{bmatrix}, \quad B_k = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.3 \\ 0 & 0.1 \\ 0.2 & 0 \end{bmatrix},$$

$Q_k = \mathbf{I}$, $R_k = \mathbf{I}$, $\mathbf{E}[w_k] = \bar{w}_k = \begin{bmatrix} 0.1 & -0.1 & 0.3 & -0.3 \end{bmatrix}^T$, $\mathbf{E}[w_k w_k^T] = W_k = 0.1\mathbf{I}$, and $N = 30$.
Report the optimal policy coefficients at time 0, $k = 0$, and the optimal cost if the initial state is zero, $x_0 = \mathbf{0}$.

**SOLUTION:**

The general dynamic programing algorithm for the time horizon, $N$, the terminal cost is set as $J_N(x_N) = g_N(x_N)$ and then minimizing the cost function for each time step recursively and then determines the optimal input and continues in reverse until $k = 0$ to find the optimal policy $\pi_k^\star(x_k)$ $\forall_{k=0,\dots,N-1}$:

$$J_k(x_k) = \min_{u_k \in \mathcal{U}_k(x_k)} \mathbf{E}_w[g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))]$$

with $\pi_k^\star(x_k) = \arg\min$ of above.

For the LQ case, the dynamic program can be defined by the algorithm with stage cost, $g_k(x_k, u_k) = x_k^T Q_k x_k + u_k^T R_k u_k$, and terminal cost, $g_N(x_N) = x_N^T Q_N x_N$. Initializing the terminal cost results in

$$J_N(x_N) = x_N^T Q_N x_N$$

and then the recursive optimization for the LTV system becomes

$$J_k(x_k) = \min_{u_k \in \mathcal{U}_k(x_k)} \mathbf{E}_w[x_k^T Q_k x_k + u_k^T R_k u_k + J_{k+1}(A_k x_k + B_k u_k + w_k)]$$

For simplicity, the quadratic form will be utilized and will be reduced to the LQ form, $x_k^T P_k x_k + r_k$, thus

$$J_N(x_k) = x_N^T Q_N x_N \implies P_N = Q_N,\ r_N = 0$$

and the recursive iteration will also aim to reduce to an update for $P_k$ and $r_k$ as follows:

$$\begin{aligned}
J_k(x_k) &= x_k^T P_k x_k + 2q_k^T x_k + r_k \\
&= \min_{u_k \in \mathcal{U}_k(x_k)} \mathbf{E}_w[x_k^T Q_k x_k + u_k^T R_k u_k + J_{k+1}(Ax_k + Bu_k + w_k, u_{k+1}, w_{k+1})]
\end{aligned}$$

since $J_{k+1} = x_{k+1}^T P_{k+1} x_{k+1} + r_{k+1}$

$$\begin{aligned}
&= \min_{u_k \in \mathcal{U}_k(x_k)} \mathbf{E}_w[x_k^T Q_k x_k + u_k^T R_k u_k + (A_k x_k + B_k u_k + w_k)^T P_{k+1}(A_k x_k + B_k u_k + w_k) \\
&\qquad + 2q_{k+1}^T(A_k x_k + B_k u_k + w_k) + r_{k+1}] \\
&= \min_{u_k \in \mathcal{U}_k(x_k)} \mathbf{E}_w[x_k^T Q_k x_k + u_k^T R_k u_k + (x_k^T A_k^T + u_k^T B_k^T + w_k^T) P_{k+1}(A_k x_k + B_k u_k + w_k) \\
&\qquad + 2q_{k+1}^T(A_k x_k + B_k u_k + w_k) + r_{k+1}] \\
&= \min_{u_k \in \mathcal{U}_k(x_k)} \mathbf{E}_w[x_k^T Q_k x_k + u_k^T R_k u_k \\
&\qquad + \big(x_k^T A_k^T P_{k+1} A_k x_k + x_k^T A_k^T P_{k+1} B_k u_k + x_k^T A_k^T P_{k+1} w_k\big) \\
&\qquad + \big(u_k^T B_k^T P_{k+1} A_k x_k + u_k^T B_k^T P_{k+1} B_k u_k + u_k^T B_k^T P_{k+1} w_k\big) \\
&\qquad + \big(w_k^T P_{k+1} A_k x_k + w_k^T P_{k+1} B_k u_k + w_k^T P_{k+1} w_k\big) \\
&\qquad + \big(2q_{k+1}^T A_k x_k + 2q_{k+1}^T B_k u_k + 2q_{k+1}^T w_k\big) + r_{k+1}]
\end{aligned}$$

Note that for scalar values the trace is equivalent and then the cyclic property of trace and linearity of expectation can be used to shift things around

$$\begin{aligned}
&= \min_{u_k \in \mathcal{U}_k(x_k)} \mathbf{E}_w[x_k^T \big(Q_k + A_k^T P_{k+1} A_k\big) + u_k^T \big(R_k + B_k^T P_{k+1} B_k\big) u_k + x_k^T A_k^T P_{k+1} B_k u_k + u_k^T B_k^T P_{k+1} A_k x_k] \\
&\qquad + \mathbf{E}_w[2q_{k+1}^T A_k x_k + 2q_{k+1}^T B_k u_k + r_{k+1}] \\
&\qquad + \mathbf{E}_w[x_k^T A_k^T P_{k+1} w_k + w_k^T P_{k+1} A_k x_k] \\
&\qquad + \mathbf{E}_w[u_k^T B_k^T P_{k+1} w_k + w_k^T P_{k+1} B_k u_k] \\
&\qquad + \mathbf{E}_w[w_k^T P_{k+1} w_k + 2q_{k+1}^T w_k] \\
&= \min_{u_k \in \mathcal{U}_k(x_k)} x_k^T \big(Q_k + A_k^T P_{k+1} A_k\big) + u_k^T \big(R_k + B_k^T P_{k+1} B_k\big) u_k + x_k^T A_k^T P_{k+1} B_k u_k + u_k^T B_k^T P_{k+1} A_k x_k \\
&\qquad + 2q_{k+1}^T A_k x_k + 2q_{k+1}^T B_k u_k + r_{k+1} \\
&\qquad + 2\,\mathrm{tr}(P_{k+1} \mathbf{E}_w[x_k^T w_k]) + 2\,\mathrm{tr}(P_{k+1} B_k \mathbf{E}_2[w_k^T u_k]) + \mathrm{tr}(P_{k+1} \mathbf{E}_w[w_k^T w_k]) + 2q_{k+1}^T \mathbf{E}[w_k] \\
&= x_k^T \big(Q_k + A_k^T P_{k+1} A_k\big) + 2q_{k+1}^T A_k x_k + 2\,\mathrm{tr}(P_{k+1} \mathbf{E}_w[x_k^T w_k]) + \mathrm{tr}(P_{k+1} W_k) + 2q_{k+1}^T \bar{w} + r_{k+1} \\
&\qquad + \min_{u_k \in \mathcal{U}_k(x_k)} u_k^T \big(R_k + B_k^T P_{k+1} B_k\big) u_k + 2x_k^T A_k^T P_{k+1} B_k u_k + 2\,\mathrm{tr}(P_{k+1} B_k \mathbf{E}_w[w_k^T u_k])
\end{aligned}$$

To solve this minimization, a derivative is performed w.r.t. $u_k$:

$$\begin{aligned}
0 &= \frac{\mathrm{d}}{\mathrm{d}u}\big(u_k^T\big(R_k + B_k^T P_{k+1} B_k\big)u_k + 2x_k^T A_k^T P_{k+1} B_k u_k + 2q_{k+1}^T B_k u_k + 2\,\mathrm{tr}(P_{k+1} B_k \mathbf{E}_2[w_k^T u_k])\big) \\
&= 2\big(R_k + B_k^T P_{k+1} B_k\big)u_k + 2x_k^T A_k^T P_{k+1} B_k + 2q_{k+1}^T B_k + 2\,\mathrm{tr}(P_{k+1} B_k \mathbf{E}_w[w_k^T]) \\
&= (R_k + B_k^T P_{k+1} B_k)u_k + x_k^T A_k^T P_{k+1} B_k + q_{k+1}^T B_k + \mathrm{tr}(P_{k+1} B_k \bar{w}_k^T) \\
-(R_k + B_k^T P_{k+1} B_k)u_k &= x_k^T A_k^T P_{k+1} B_k + \mathrm{tr}(P_{k+1} B_k w_k^T) \\
u_k &= -(R_k + B_k^T P_{k+1} B_k)^{-1}\big(x_k^T A_k^T P_{k+1} B_k + q_{k+1}^T B_k + \mathrm{tr}(P_{k+1} B_k \bar{w}_k^T)\big) \\
u_k &= -(R_k + B_k^T P_{k+1} B_k)^{-1}\big(B_k^T P_{k+1} A_k\big)x_k - (R_k + B_k^T P_{k+1} B_k)^{-1}\big(q_{k+1}^T B_k + \mathrm{tr}(P_{k+1} B_k \bar{w}_k^T)\big)
\end{aligned}$$

Resulting an an optimal control law:

$$u_k^\star = K_k x_k + l_k$$

with

$$K_k = -(R_k + B_k^T P_{k+1} B_k)^{-1}\big(B_k^T P_{k+1} A_k\big)$$

and

$$l_k = -(R_k + B_k^T P_{k+1} B_k)^{-1}\big(q_{k+1}^T B_k + \mathrm{tr}(P_{k+1} B_k \bar{w}_k^T)\big)$$

Next, the cost function results is calculated recursively by

$$\begin{aligned}
J_k^\star &= x_k^T Q_k x_k + 2q_k^T + r_k \\
&= x_k^T\big(Q_k + A_k^T P_{k+1} A_k\big) + 2q_{k+1}^T A_k x_k \\
&\quad + 2\,\mathrm{tr}(P_{k+1}\mathbf{E}_w[x_k^T w_k]) + \mathrm{tr}(P_{k+1} W_k) + 2q_{k+1}^T \bar{w} + r_{k+1} \\
&\quad + (-(R_k + B_k^T P_{k+1} B_k)^{-1}\big(x_k^T A_k^T P_{k+1} B_k + q_{k+1}^T B_k + \mathrm{tr}(P_{k+1} B_k \bar{w}_k^T)\big))^T \cdot \\
&\qquad \cdot \big(R_k + B_k^T P_{k+1} B_k\big)(-(R_k + B_k^T P_{k+1} B_k)^{-1}\big(x_k^T A_k^T P_{k+1} B_k + q_{k+1}^T B_k + \mathrm{tr}(P_{k+1} B_k \bar{w}_k^T)\big)) \\
&\quad + 2x_k^T A_k^T P_{k+1} B_k(-(R_k + B_k^T P_{k+1} B_k)^{-1}\big(x_k^T A_k^T P_{k+1} B_k + q_{k+1}^T B_k + \mathrm{tr}(P_{k+1} B_k \bar{w}_k^T)\big)) \\
&\quad + 2\,\mathrm{tr}(P_{k+1} B_k \mathbf{E}_w[w_k^T(-(R_k + B_k^T P_{k+1} B_k)^{-1}\big(x_k^T A_k^T P_{k+1} B_k + q_{k+1}^T B_k + \mathrm{tr}(P_{k+1} B_k \bar{w}_k^T)\big))])
\end{aligned}$$

which when evaluated results in (after making an assumption of zero correlation between $x_k$ and $w_k$)

$$\begin{aligned}
J_{k+1}(x) = x^T&[(Q + A^T P_{k+1} A) - (A^T P_{k+1}^T B^T)(R + B^T P_{k+1} B)^{-1}]x \\
&+ 2\bar{w}^T[(P_{k+1} A) - (R + B^T P_{k+1} B)^{-1}(B P_{k+1} A)]x \\
&+ \bar{w}^T[(P_{k+1}^T B)(R + B^T P_{k+1} B) - (B^T P_{k+1})]\bar{w} \\
&+ \mathbf{tr}(P_{k+1} W_t) + r_{k+1}
\end{aligned}$$

where $P = (Q + A^T P_{k+1} A) - (A^T P_{k+1}^T B^T)(R + B^T P_{k+1} B)^{-1}$, $Q = (P_{k+1} A) - (R + B^T P_{k+1} B)^{-1}(B P_{k+1} A)$, and $R = (P_{k+1}^T B)(R + B^T P_{k+1} B) - (B^T P_{k+1}) + \mathbf{tr}(P_{k+1} W_t) + r_{k+1}$.
As for a final result, results (calculated in matlab) for the total cost at $k = 0$, with $N = 30$ is $J_0 = 6.0635 \times 10^3$.

```
%% MECH 6326 - Homework 4
% Author: Jonas Wagner
% Date: 2023-04-14
% Much credit to collaboraters:
% Devshan, Leon, Alyssa
% (+ Chat GPT/Bing Chatbot)

%% Problem 1
% part a
clc; clear; close all

a = load('robot_nav.mat');
[i_ind, j_ind, ~] = find(a.X); % Unpack the structure data
map = -1* a.X;
grid_size = size(map,1);

map(1,end) = 1; % End point
map(1,1:end-1) = -1;
map(2:end,end) = -1;

% Scenarios:
u_mat = [
    0.8 0.2; % North North
    0.6 0.4; % North East
    0.2 0.8; % East North
    0.4 0.6]; % East East

N = 82;

J = zeros(grid_size,grid_size,N+1); % Value function
J(:,:,N+1) = map;

pi_Star = zeros(grid_size,grid_size,N); % Optimal policy


for k=N:-1:1 % backwards time recursion
    for i=1:grid_size
        for j=1:grid_size
            if map(i,j) ~= 0            % Hit something
                if (i < grid_size) && (j < grid_size)   % Hit obstacle
                    J(i,j,k) = -1;
```

```
                    pi_Star(i,j,k) = 0;
                elseif i == grid_size           % Hit top boundary
                    Ji = zeros(2,1);
                    for u = 3:4
                        Ji(u) = map(i,j) + u_mat(u,:)*[J(max(1,i-1),j,↙
k+1) J(i,min(j+1,grid_size),k+1)]';
                    end
                    [J(i,j,k), pi_Star(i,j,k)] = max(Ji);
                elseif j == grid_size           % Hit right boundary
                    Ji = zeros(2,1);
                    for u = 1:2
                        Ji(u) = map(i,j) + u_mat(u,:)*[J(max(1,i-1),j,↙
k+1) J(i,min(j+1,grid_size),k+1)]';
                    end
                    [J(i,j,k), pi_Star(i,j,k)] = max(Ji);
                end
            else                            % Did not hit anything
                Ji = zeros(4,1);
                for u = 1:4
                    Ji(u) = map(i,j) + u_mat(u,:)*[J(max(1,i-1),j,k+1) J↙
(i,min(j+1,grid_size),k+1)]';
                end
                [J(i,j,k), pi_Star(i,j,k)] = max(Ji);
            end

        end
    end
end

figure;
imagesc(J(:,:,1));
title("Value function");
saveas(gcf,'figs/pblm1_valFunc.png')

figure;
imagesc(pi_Star(:,:,1));
title("Optimal Policy");
saveas(gcf,'figs/pblm1_optPolicy.png')

% part b
% initialize variables
```

```
N = 1000; % number of Monte Carlo simulations
success_count = 0; % count successful simulations
success_trajectories = {}; % store successful trajectories
obstacles = [i_ind j_ind]; % obstacles matrix

% Monte Carlo simulation
for i = 1:N
    % initialize a new simulation
    x = 1; % start state
    y = 1;
    trajectory = [x y];

    % simulate until reaching the goal or crashing
    while ~(x == 41 && y == 41) && ~ismember([x y], obstacles, 'rows')
        % choose an action based on the optimal policy
        action = pi_Star(42-x, y);
        if action == 1 % move right
            if rand() < 0.8 % move in the intended direction
                y = min(41, y+1);
            else % move in the unintended direction
                x = min(41, x+1);
            end
        else % move up
            if rand() < 0.6 % move in the intended direction
                x = min(41, x+1);
            else % move in the unintended direction
                y = min(41, y+1);
            end
        end

        % add the new state to the trajectory
        trajectory = [trajectory; x y];
    end

    % check if the simulation was successful
    if x == 41 && y == 41
        success_count = success_count + 1;
        success_trajectories{end+1} = trajectory;
    end
end
```

```
% estimate the success rate
success_rate = (success_count / N)*100;
fprintf("Success rate: %.2f%%", success_rate);
fprintf("\n");

% plot a successful trajectory
figure;
hold on;
for i = 1:size(success_trajectories{1}, 1)-1
    x1 = success_trajectories{1}(i, 1);
    y1 = success_trajectories{1}(i, 2);
    x2 = success_trajectories{1}(i+1, 1);
    y2 = success_trajectories{1}(i+1, 2);
    plot([y1 y2], [x1 x2],'b', 'LineWidth', 2);
end
scatter(1, 1, 50, 'ko', 'filled');
scatter(41, 41, 50, 'go', 'filled');
scatter(j_ind, i_ind, 50, 'r*');
xlim([1 41]);
ylim([1 41]);
title("Sample Successful Trajectory");
saveas(gcf,'figs/pblm1_sampleTraj.png')


%% Problem 2
clear; close all;% clc

% Problem Data
A = [.4 -.3 0 .6;
     .1 .7 .2 0;
     .5 .2 -.8 .1;
     0 .3 -.4 9];
B = [.1 .1;
     .1 .3;
     0 .1;
     .2 0];
Q = eye(4);
R = eye(2);
w = [.1; -.1; .3; -.3];
W = .1*eye(4);
N = 30;
```

```
% Calculation of Cost
P(:,:,31) = Q;
for t = N:-1:1
    P(:,:,t) = Q + A'*P(:,:,t+1)*A - A'*P(:,:,t+1)*B*inv(R + B'*P(:,:,
t+1)*B)*B'*P(:,:,t+1)*A;
end

x0 = [0;0;0;0];
constants = 0;
for i = 1:30
    constants = constants + trace(P(:,:,t+1)*W);
end

initial_cost = x0'*P(:,:,1)*x0 + constants
initial_coefficients = -inv(R + B'*P(:,:,2)*B)*(B'*P(:,:,2)*A*x0 + B'*P
(:,:,2)*w)
```