ME599 Homework 4 - Akshat Dubey

Problem 1

We have a system characterized by

- Input: $V(t)$ Voltage pulse imputed on the nozzle system
- System: $\mathcal{G}$, the nozzle system
- Output: $P(t)$ pressure wave at the nozzle

The system $\mathcal{G}$ is not available to us, it is a black box. We can only give it inputs and observe the outputs. In this problem we are trying to get the system to output a specific pressure waveform, and hence we need to implement a controller that can converge to a specific input voltage waveform that produces the desired pressure waveform.

Problem 1.a

To check if the input-output operator $\mathcal{G}$ is linear, we can check if the system satisfies the superposition principle. This can be done by checking if the following equation holds true for all $V_1(t)$ and $V_2(t)$ and a_1 and a_2:

$$\mathcal{G}(a_1V_1(t) + a_2V_2(t)) = a_1\mathcal{G}(V_1(t)) + a_2\mathcal{G}(V_2(t))$$

For this, we first generate the two voltages $V_1(t)$ and $V_2(t)$ and set $a_1 = -0.23$ and $a_2 = 1.14$ (chosen randomly).

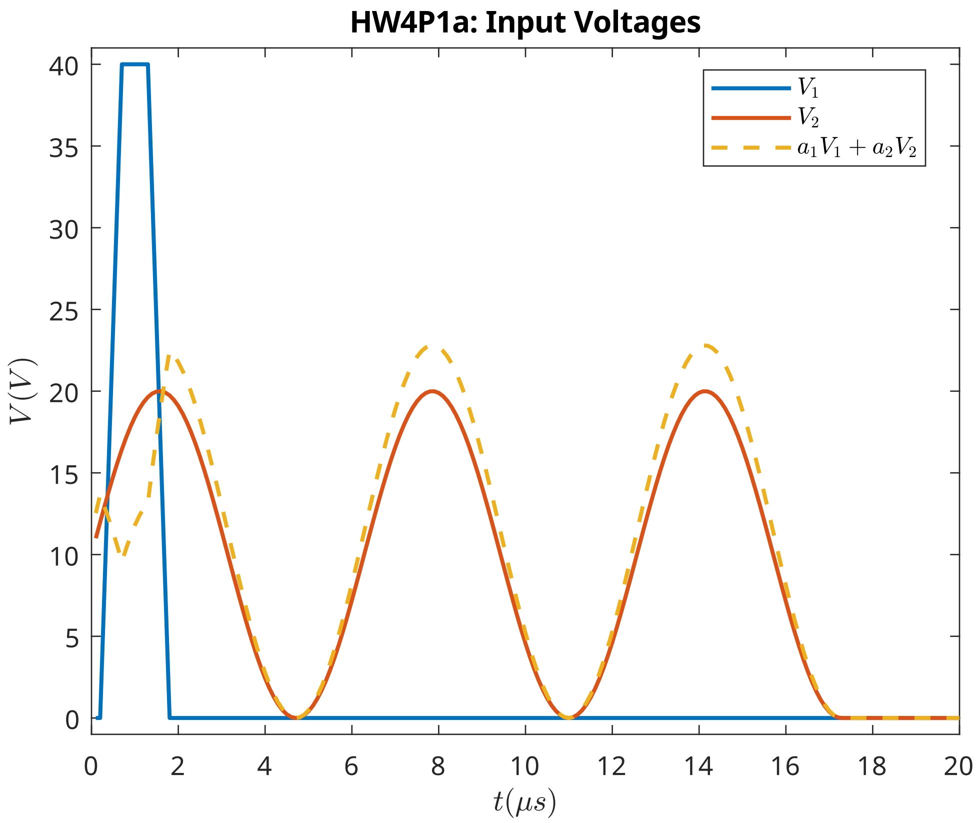


Figure 1: Input Voltage Waveforms

Then we pass them through the operator $\mathcal{G}$ and plot the output pressure waveforms. Let

$$P_1(t) = \mathcal{G}(V_1(t))$$
$$P_2(t) = \mathcal{G}(V_2(t))$$
$$P_{12}(t) = \mathcal{G}(a_1 V_1(t) + a_2 V_2(t))$$
$$P_a(t) = a_1 P_1(t) + a_2 P_2(t)$$

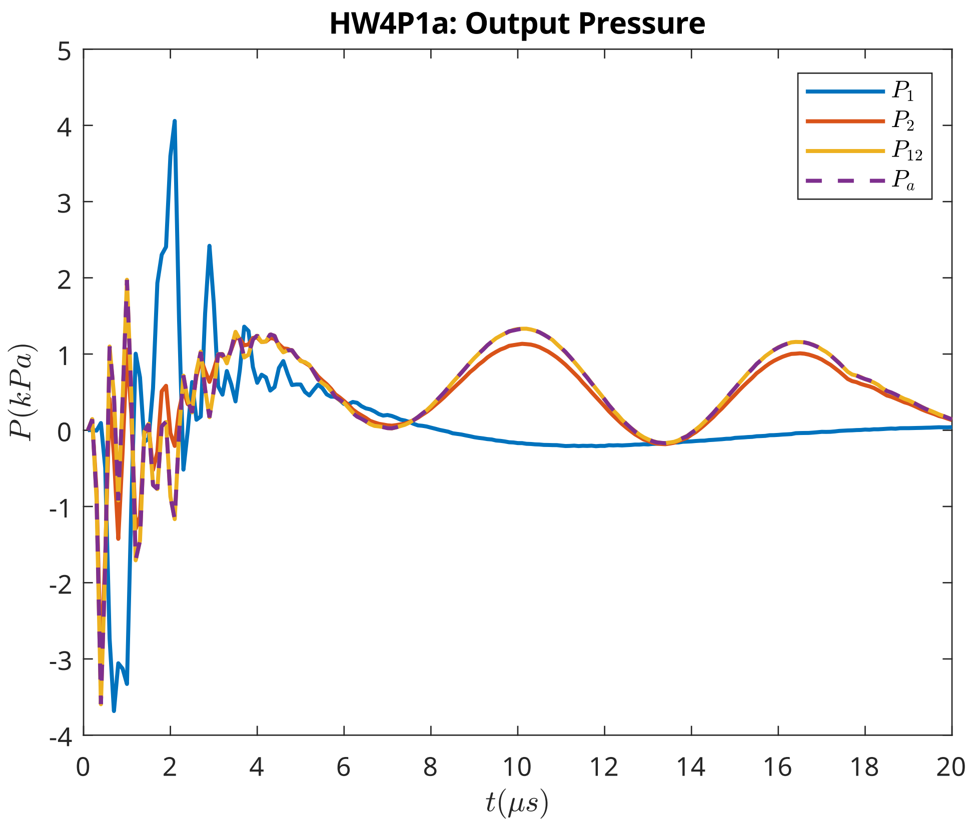


Figure 2: Output Pressure Waveforms

In this plot, we can see that the pressure waveform for $P_{12}(t)$ (the output of the system for the combined input) is the same as the pressure waveform for $P_a(t)$ (the output of the system for the individual inputs scaled by a_1 and a_2). This means that the system satisfies the superposition principle and is hence $\mathcal{G}$ is linear.

Problem 1.b

We have been given the deired pressure waveform P_{ref} as a vector and we need to find the input voltage waveform vector V^* that produces this pressure waveform. We can find this V^* by constructing a cost function of the norm error e between the desired pressure waveform and the output pressure waveform and choosing the V_* that minimizes this cost function. The cost function can be defined as:

$$J = \frac{1}{2}||e||_2^2$$

where

$$e = P_{ref} - \mathcal{G}(V)$$

Since we know from part a that the system is linear, we can write the error as

$$e = P_{ref} - GV$$

Where G is a matrix that represents the linear system $\mathcal{G}$. To minimize the cost function, we can perform gradient descent on the cost function. The gradient of the cost function can be calculated as:

$$\begin{aligned}
J &= \frac{1}{2}||e||_2^2 \\
&= \frac{1}{2}e^T e \\
\implies \frac{\partial J}{\partial e} &= e \\
e &= P_{ref} - GV \\
\implies \frac{\partial e}{\partial V} &= -G^T \\
\text{therefore } \frac{\partial J}{\partial V} &= \frac{\partial e}{\partial V}\frac{\partial J}{\partial e} \\
&= -G^T e
\end{aligned}$$

We can then use this gradient to perform gradient descent on the cost function. The update law for the gradient descent on the cost function J for an iteration $i+1$ can be written as:

$$\begin{aligned}
V_{i+1} &= V_i - \alpha\frac{\partial J}{\partial V} \\
&= V_i + \alpha G^T e
\end{aligned}$$

Snce we do not know what G is, we cannot perform its transpose directly. We need to use a matrix τ to flip the the error vector, pass it through the system and then flip the result back. This effectively gives us the gradient of the cost function by transposing the matrix G.

$$\begin{aligned}
\tau &= \begin{bmatrix} 0 & \dots & 0 & 1 \\ 0 & 0 & 1 & 0 \\ \vdots & 1 & \vdots & \vdots \\ 1 & 0 & \dots & 0 \end{bmatrix} \\
G^T e &= \tau\mathcal{G}(\tau e)
\end{aligned}$$

Finally, we can write the optimization problem as

$$\begin{aligned}
V^* &= \arg\min_V \frac{1}{2}||e||_2^2 \\
&\text{with the update law} \\
V_{i+1} &= V_i + \alpha\tau\mathcal{G}(\tau e) \\
&\text{where} \\
e &= P_{ref} - \mathcal{G}(V)
\end{aligned}$$

Setting the learning rate to 0.5, and the maximum number of iterations to be 10000, we get V^* and a $\mathcal{G}(V^*)$. The resulting input-output waveforms are shown below

The error between the resulting pressure waveform and the desired pressure waveform is very small, which means that the optimization problem has converged to a good solution. The error is also shown below

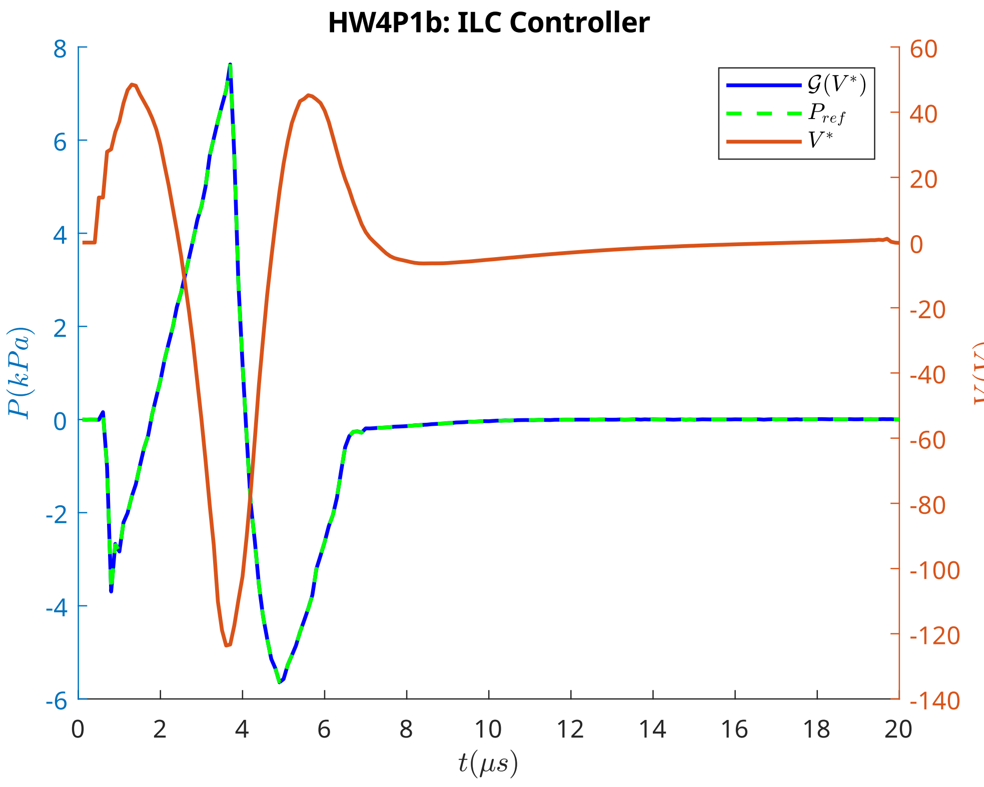


Figure 3: Optimal Input Voltage and resulting Pressure Waveform

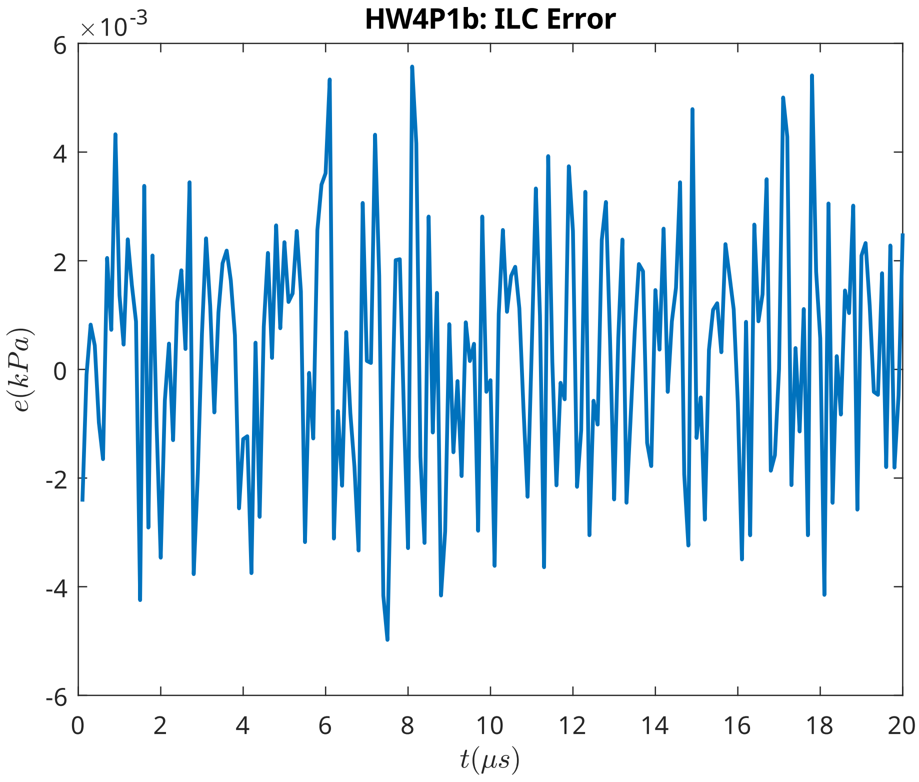


Figure 4: Error between desired and resulting pressure waveforms

Contents

ME599 HW4 Problem 1

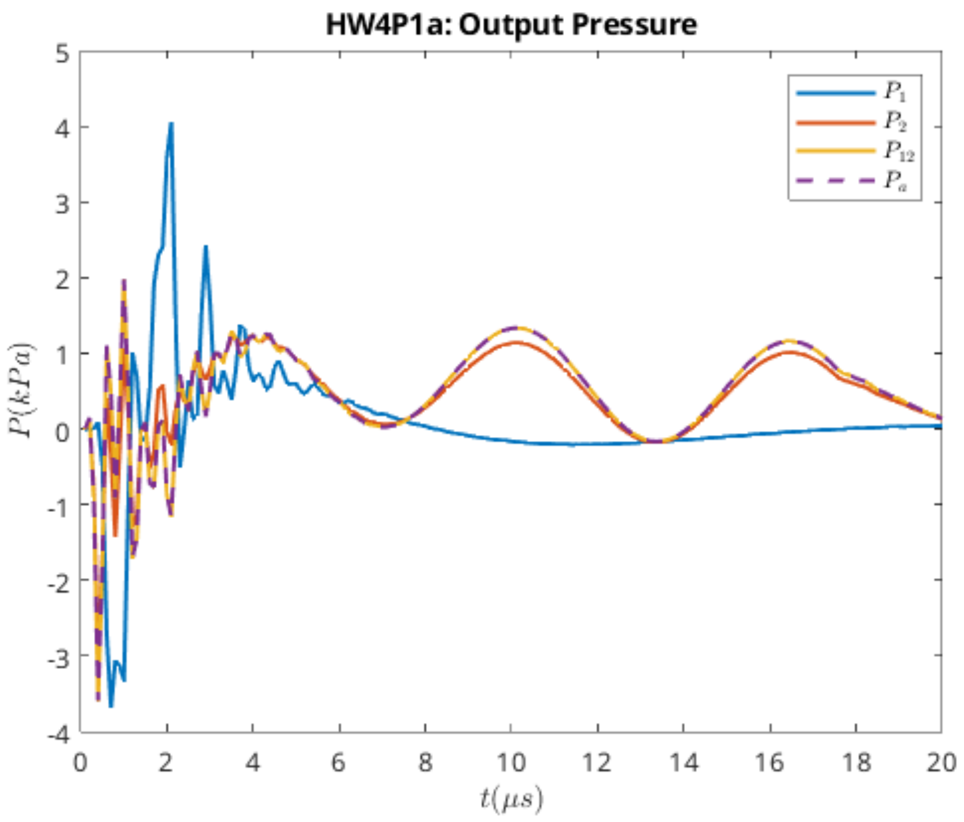
```
clc; clear; close all;
```

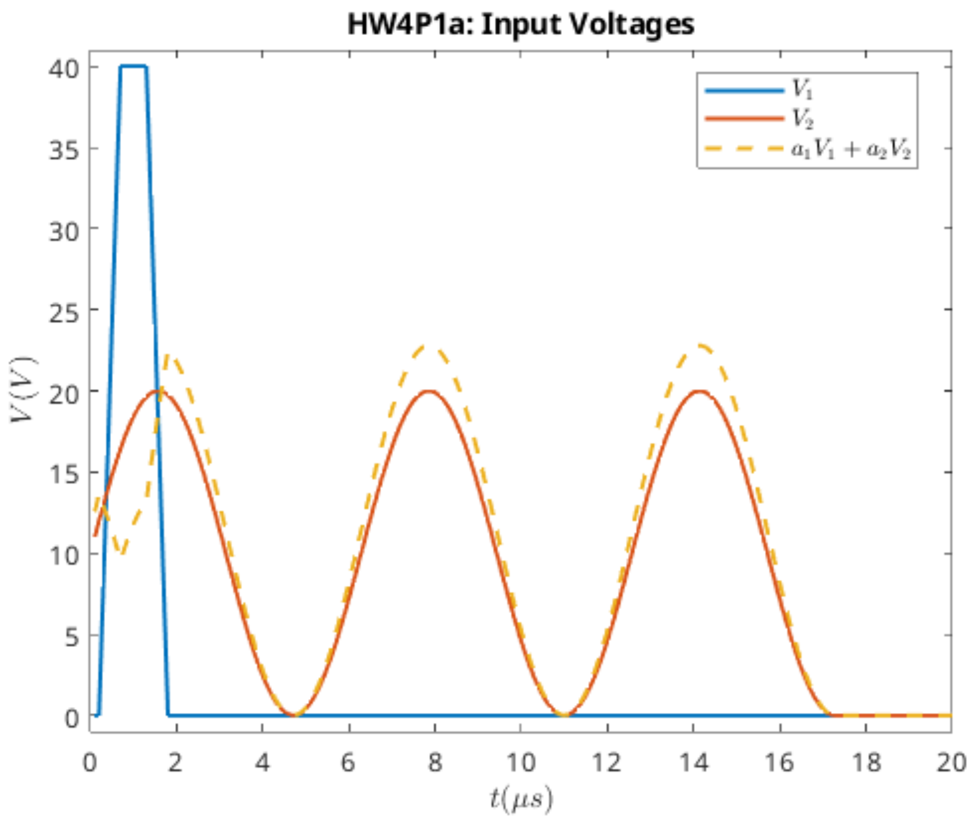
part a, proving operator G is linear

Load example voltage as V1

```
load('V_example.mat');
V1 = V_example;
N = length(V1);
tspan = 0.1*(1:N)'; % transpose so row vector

% generate some weird shit as V2
V2 = (sin(tspan)+1) * max(V1)/4; % scale to be half the height as V1
V2(173:end) = 0; % zeros after last full sine wave

a1 = -0.23;
a2 = 1.14;
V12 = a1*V1+a2*V2;

% plot input voltages
fig = figure;
plot(tspan, V1, LineWidth=1.5, DisplayName='$V_1$');
hold on;
plot(tspan, V2, LineWidth=1.5, DisplayName='$V_2$');
plot(tspan, V12, '--', LineWidth=1.5, DisplayName='$a_1V_1+a_2V_2$')
ylabel('$V(V)$', 'Interpreter', 'latex')
ylim([-1 41]);
xlabel('$t(\mu s)$', 'Interpreter', 'latex')
legend('Location', 'best', 'Interpreter', 'latex');
title("HW4P1a: Input Voltages");
saveas(fig, 'figs/hw4p1a_volts.svg');

% get outputs for all possible inputs
P1 = piezo_nozzle(V1, N);
P2 = piezo_nozzle(V2, N);
P12 = piezo_nozzle(V12, N);
P12_alt = a1*P1 + a2*P2;

% plot outputs
fig = figure;
plot(tspan, P1, LineWidth=1.5, DisplayName='$P_1$');
hold on;
plot(tspan, P2, LineWidth=1.5, DisplayName='$P_2$');
plot(tspan, P12, LineWidth=1.5, DisplayName='$P_{12}$');
plot(tspan, P12_alt, '--', LineWidth=1.5, DisplayName='$P_a$');
```

```
ylabel('$P(kPa)$', 'Interpreter', 'latex');
xlabel('$t(\mu s)$', 'Interpreter', 'latex')
legend('Location', 'best', 'Interpreter', 'latex');
title("HW4P1a: Output Pressure");
saveas(fig, 'figs/hw4p1a_press.svg');
```



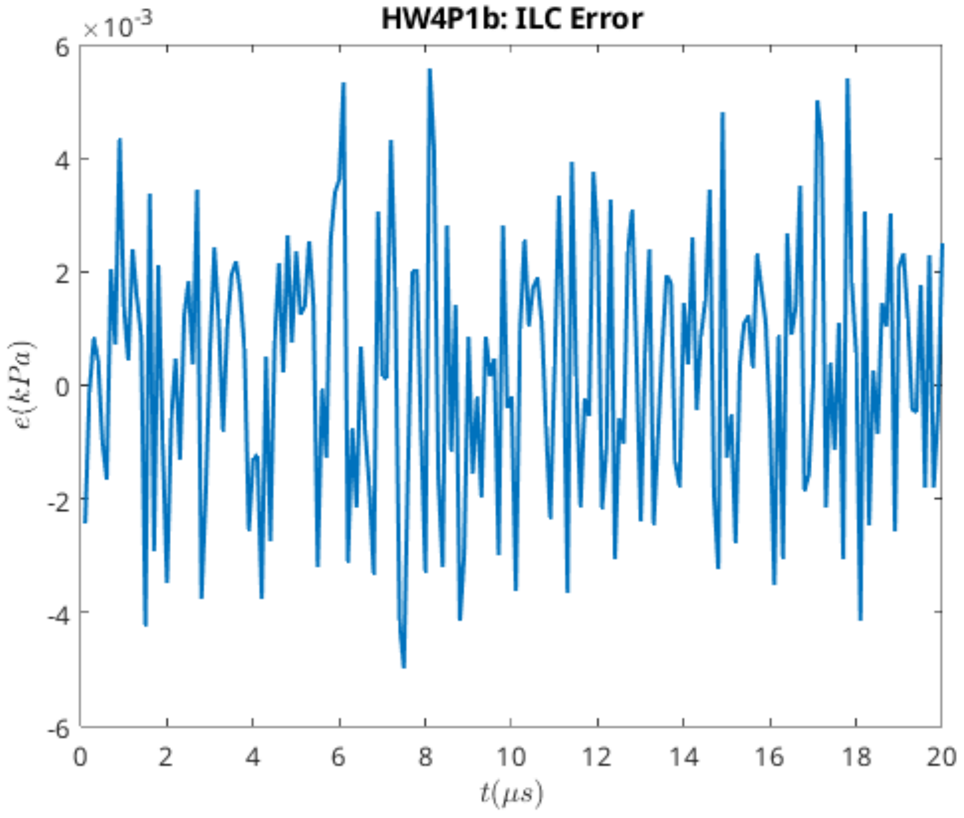


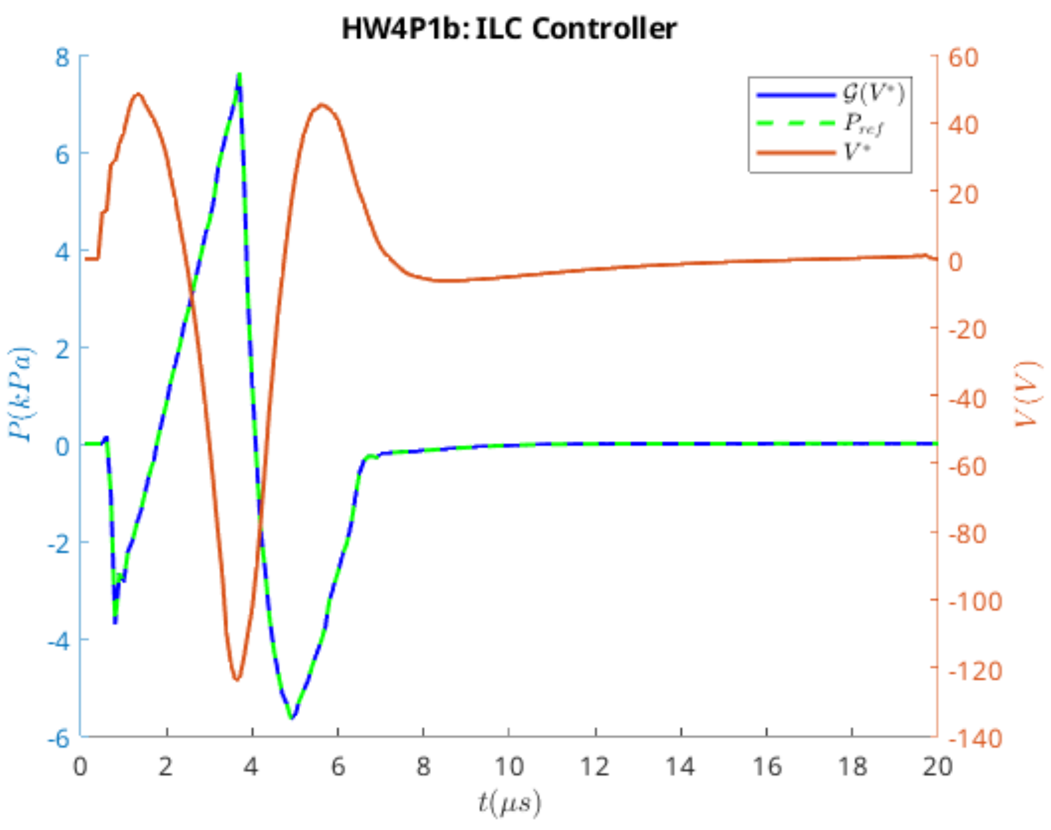
part b, designing an ILC controller

```
load('P_ref.mat');
max_iters = 10000;
learn_rate = 0.5;
tol = 1e-6;
V = zeros(N,1); % initial guess
tau = flip(eye(N)); % flipping matrix

i = 0;
while true
        % get the error
        e = P_ref - piezo_nozzle(V, N);
        % get the cost
        J = 0.5 * (e' * e);
        % get the gradient
        grad = tau * piezo_nozzle(tau*e, N);
        % update the guess
        V = V + learn_rate * grad;
        % print results
        if mod(i, 100) == 0
                fprintf("Iteration %d: J = %f, grad = %f\n", i, J, norm(grad));
        end

        % check for convergence
        if norm(grad) < 1e-6
                fprintf("Norm of gradient reached tolerance criteria");
                break;
        end
        i = i + 1;
        if i > max_iters
```

```
                fprintf("Max iterations reached\n");
                break;
        end
end

fprintf("Converged after %d iterations\n", i);

P_opt = piezo_nozzle(V, N);

% plot the result
fig = figure;
yyaxis left;
hold on;
plot(tspan, P_opt, '-b', LineWidth=1.5, DisplayName='$\mathcal{G}(V^*)$');
plot(tspan, P_ref, '--g', LineWidth=1.5, DisplayName='$P_{ref}$');
ylabel('$P(kPa)$', 'Interpreter', 'latex');
yyaxis right;
plot(tspan, V, LineWidth=1.5, DisplayName='$V^*$');
ylabel('$V(V)$', 'Interpreter', 'latex');
xlabel('$t(\mu s)$', 'Interpreter', 'latex');
legend('Location', 'best', 'Interpreter', 'latex');
title("HW4P1b: ILC Controller");
saveas(fig, 'figs/hw4p1b.svg');

% plot the error
fig = figure;
plot(tspan, P_ref - P_opt, LineWidth=1.5);
ylabel('$e(kPa)$', 'Interpreter', 'latex');
hold on;
xlabel('$t(\mu s)$', 'Interpreter', 'latex');
title("HW4P1b: ILC Error");
saveas(fig, 'figs/hw4p1b_error.svg');
```

```
Iteration 0: J = 421.094271, grad = 2.537997
Iteration 100: J = 257.675001, grad = 1.521602
Iteration 200: J = 166.009324, grad = 1.197237
Iteration 300: J = 108.349956, grad = 0.957964
Iteration 400: J = 71.221965, grad = 0.774655
Iteration 500: J = 47.010143, grad = 0.624367
Iteration 600: J = 31.153569, grad = 0.507711
Iteration 700: J = 20.694009, grad = 0.414300
Iteration 800: J = 13.775536, grad = 0.336713
Iteration 900: J = 9.208717, grad = 0.273527
Iteration 1000: J = 6.144694, grad = 0.223219
Iteration 1100: J = 4.128437, grad = 0.184077
Iteration 1200: J = 2.765726, grad = 0.150209
Iteration 1300: J = 1.861365, grad = 0.118916
Iteration 1400: J = 1.248417, grad = 0.105952
Iteration 1500: J = 0.840353, grad = 0.086207
Iteration 1600: J = 0.563285, grad = 0.072929
Iteration 1700: J = 0.378553, grad = 0.063226
Iteration 1800: J = 0.258390, grad = 0.053988
Iteration 1900: J = 0.173484, grad = 0.053506
Iteration 2000: J = 0.117645, grad = 0.045738
Iteration 2100: J = 0.079550, grad = 0.044204
Iteration 2200: J = 0.053226, grad = 0.039116
Iteration 2300: J = 0.036909, grad = 0.034975
Iteration 2400: J = 0.025516, grad = 0.036234
Iteration 2500: J = 0.016807, grad = 0.037793
Iteration 2600: J = 0.011562, grad = 0.033302
Iteration 2700: J = 0.007826, grad = 0.030264
Iteration 2800: J = 0.006315, grad = 0.036354
Iteration 2900: J = 0.004133, grad = 0.033681
Iteration 3000: J = 0.003036, grad = 0.031604
Iteration 3100: J = 0.002503, grad = 0.032590
Iteration 3200: J = 0.001689, grad = 0.033180
Iteration 3300: J = 0.001344, grad = 0.030727
Iteration 3400: J = 0.000992, grad = 0.031289
Iteration 3500: J = 0.000909, grad = 0.034958
Iteration 3600: J = 0.000710, grad = 0.034692
Iteration 3700: J = 0.000691, grad = 0.033230
Iteration 3800: J = 0.000582, grad = 0.030791
Iteration 3900: J = 0.000670, grad = 0.032250
Iteration 4000: J = 0.000684, grad = 0.032791
Iteration 4100: J = 0.000508, grad = 0.034421
Iteration 4200: J = 0.000497, grad = 0.034147
Iteration 4300: J = 0.000627, grad = 0.034319
Iteration 4400: J = 0.000619, grad = 0.033461
Iteration 4500: J = 0.000537, grad = 0.031266
Iteration 4600: J = 0.000590, grad = 0.034953
Iteration 4700: J = 0.000563, grad = 0.033056
Iteration 4800: J = 0.000481, grad = 0.031560
Iteration 4900: J = 0.000484, grad = 0.033008
Iteration 5000: J = 0.000578, grad = 0.031219
Iteration 5100: J = 0.000486, grad = 0.030892
Iteration 5200: J = 0.000537, grad = 0.030538
Iteration 5300: J = 0.000490, grad = 0.030101
Iteration 5400: J = 0.000542, grad = 0.033508
Iteration 5500: J = 0.000530, grad = 0.032244
Iteration 5600: J = 0.000609, grad = 0.034370
Iteration 5700: J = 0.000536, grad = 0.033669
Iteration 5800: J = 0.000493, grad = 0.032157
Iteration 5900: J = 0.000545, grad = 0.032662
Iteration 6000: J = 0.000457, grad = 0.030077
Iteration 6100: J = 0.000606, grad = 0.034633
Iteration 6200: J = 0.000480, grad = 0.031763
Iteration 6300: J = 0.000665, grad = 0.034920
Iteration 6400: J = 0.000441, grad = 0.032332
Iteration 6500: J = 0.000588, grad = 0.028885
Iteration 6600: J = 0.000565, grad = 0.033139
Iteration 6700: J = 0.000521, grad = 0.033176
Iteration 6800: J = 0.000545, grad = 0.036577
Iteration 6900: J = 0.000591, grad = 0.032330
Iteration 7000: J = 0.000529, grad = 0.032974
Iteration 7100: J = 0.000563, grad = 0.035933
Iteration 7200: J = 0.000519, grad = 0.030479
Iteration 7300: J = 0.000559, grad = 0.034979
Iteration 7400: J = 0.000548, grad = 0.033704
Iteration 7500: J = 0.000704, grad = 0.033466
Iteration 7600: J = 0.000656, grad = 0.033797
Iteration 7700: J = 0.000656, grad = 0.035305
Iteration 7800: J = 0.000604, grad = 0.032820
Iteration 7900: J = 0.000609, grad = 0.031912
Iteration 8000: J = 0.000472, grad = 0.033451
Iteration 8100: J = 0.000569, grad = 0.034132
Iteration 8200: J = 0.000530, grad = 0.032806
Iteration 8300: J = 0.000503, grad = 0.034921
Iteration 8400: J = 0.000529, grad = 0.032920
Iteration 8500: J = 0.000540, grad = 0.034895
Iteration 8600: J = 0.000489, grad = 0.035328
Iteration 8700: J = 0.000510, grad = 0.033210
Iteration 8800: J = 0.000543, grad = 0.029737
Iteration 8900: J = 0.000663, grad = 0.034106
Iteration 9000: J = 0.000444, grad = 0.031061
Iteration 9100: J = 0.000569, grad = 0.034289
Iteration 9200: J = 0.000608, grad = 0.034845
Iteration 9300: J = 0.000589, grad = 0.035787
Iteration 9400: J = 0.000574, grad = 0.031743
Iteration 9500: J = 0.000495, grad = 0.033192
Iteration 9600: J = 0.000494, grad = 0.034391
Iteration 9700: J = 0.000502, grad = 0.034042
Iteration 9800: J = 0.000545, grad = 0.032055
Iteration 9900: J = 0.000562, grad = 0.033864
Iteration 10000: J = 0.000604, grad = 0.035811
Max iterations reached
Converged after 10001 iterations
```





Published with MATLAB® R2023b

Problem 2

Given discrete-time system dynamics

$$x(k+1) = F(x(k), u(k))$$

and objective function

$$J = H(x(2N)) + \sum_{k=0}^{2N} G(x(k), u(k))$$

we need to find the optimal control law $u^*(k) = \phi_k(x(k))$ that minimizes the cost function J. The proposal is to break this subproblem down into two subproblems that can be solved in parallel:

$$\begin{aligned} J_1 &= \sum_{k=0}^{N-1} (x(k), u(k)) \\ \text{with the optimal control law } u^*(k) &= A_k(x(k)), k = 0, \ldots, N-1 \\ J_2 &= H(x(2N)) + \sum_{k=N}^{2N} G(x(k), u(k)) \\ \text{with the optimal control law } u^*(k) &= B_k(x(k)), k = N, \ldots, 2N-1 \end{aligned}$$

Denoting the optimal cost function value at x_k as $J^*(x_k)$, we can rewrite the cost function using Bellman's principle of optimality:

$$J^*(x(k)) = \min_{u(k)} (G(x(k), u(k)) + J^*(x(k+1)))$$

where $x(k+1) = F(x(k), u(k))$. This relationship can be used to compute the optimal control law $u^*(k)$ and the optimal cost function value $J^*(x_0)$ recursively.

For the timestep $k = N-1$, we have:

$$\begin{aligned} J^*(x(N-1)) &= \min_{u(N-1)} (G(x(N-1), u(N-1)) + J^*(x(N))) \\ &= \min_{u(N-1)} (G(x(N-1), u(N-1)) + J^*(F(x(N-1), u(N-1)))) \end{aligned}$$

We note that the cost from N to $2N$, for the sequence $x(N), \ldots, x(2N)$, we have the optimal cost function:

$$J^*(x(N)) = J^*(F(x(N-1), u(N-1))) = J_2$$

So even though we might have an expression of the the cost function J_2 if we solve recursively from $2N$, it will be in terms of $x(N-1)$ and $u(N-1)$. This means that we will not be able to compute the optimal control law $u^*(k)$ for $k = N \ldots 2N$ till we have the actual value of $x(N-1)$ and $u(N-1)$. To determine $x(N-1)$ and $u(N-1)$, we need to solve the problem from 0 to $N-1$ first, and thus it is NOT POSSIBLE to solve the two subproblems in parallel.

Problem 3

For this reinforcement learning problem, we have an agent located in a building with 5 rooms connected by doors, with the 5th room being the outdoors. We want to find the optimal policy that the agent should follw to get outside the building from any room. There is no uncertainty in the environment, which means that the same action will always lead to the same outcome

Problem 3.a

Here, we will use model-free Q-Learning where the state transition $p(x_{k+1}|x_k, u_k)$ is deterministic. This means that given an action u_k, the next state $x_{k+1} = u_k$. We have a total of 6 states $S = \{0, 1, 2, 3, 4, 5\}$ and the actions u_k are the same as the states because the action just determines the next state directly in S.

The objective is to construct a Q-table that consists of the Q-values for each state-action pair, which is similar to the value function in dynamic programming. At the end of the algorithm, the Q-table will contain the Q-value for a state-action pair, which is defined as the expected reward when taking action u_k in state x_k and following the optimal policy thereafter.

$$Q(x_k, u_k) = R(x_k, u_k) + \gamma \max_{u_{k+1}} Q(x_{k+1}, u_{k+1})$$

where $R(x_k, u_k)$ is the reward for taking action u_k in state x_k, γ is the discount factor, and $Q(x_{k+1}, u_{k+1})$ is the Q-value for the next state-action pair.

$\gamma = 0.8$ is given in the problem statement and the reward matrix R is given as follows:

State/Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	1
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	1
5	-1	0	-1	-1	0	1

The -1 values in the table mean that it is impossible for the agent to move from one state to another, and thus such actions will be ignored.

We wish to explore the possible actions the agent can take in each state, and populate the Q-table with the Q-values for each state-action pair. Once we have the Q-table, we can use it to determine the optimal policy $\pi(x_k)$ for the agent such that it maximizes the reward of going outside to room 5. The optimal policy is basically just the action that has the maximum Q-value for each state.

We initialize the Q-learning algorithm as follows

- Initialize the Q-table with zeros
- Set the discount factor $\gamma = 0.8$
- Set max iterations $t_{max} = 2000$
- Pick a random initial state x_0 from the set of states $S = \{0, 1, 2, 3, 4, 5\}$

Then we will run the Q-learning algorithm as follows:

1. Select action and get reward
    - Select an action from $\{u_k \in S | u_k \neq -1\}$ with equal probability
    - Determine next state $x_{k+1} = u_k$
    - Get the reward $r_k = R(x_k, u_k)$
    - Increment iteration $t = t + 1$
2. Update Q-table

- $$Q(x_k, u_k) = r_k + \gamma \max_{u_{k+1}} Q(x_{k+1}, u_{k+1})$$

3. Check if iterations have been exceeded
   - If $t < t_{max}$, go to step 1
   - Else, stop the algorithm

Then, the optimal policy is given by

$$\pi(x_k) = \arg\max_{u_k} Q(x_k, u_k)$$

The optimal Q-table is as follows:

State/Action	0	1	2	3	4	5
0	0	0	0	0	4	0
1	0	0	0	3.2	0	5
2	0	0	0	3.2	0	0
3	0	4	2.56	0	4	0
4	3.2	0	0	3.2	0	5
5	0	4	0	0	4	5

The optimal policy is as follows:

State	Action
0	4
1	5
2	3
3	1/4
4	5
5	5

Problem 3.b

Now, we modify the reward matrix R to change the reward for going from state 4 to 5 to be -0.9 instead of 1. The new Q-table is as follows:

State/Action	0	1	2	3	4	5
0	0	0	0	0	2.56	0
1	0	0	0	3.2	0	5
2	0	0	0	3.2	0	0
3	0	4	2.56	0	2.56	0
4	2.048	0	0	3.2	0	3.1
5	0	4	0	0	2.56	5

The optimal policy is as follows:

State	Action
0	4
1	5
2	3
3	1
4	3
5	5

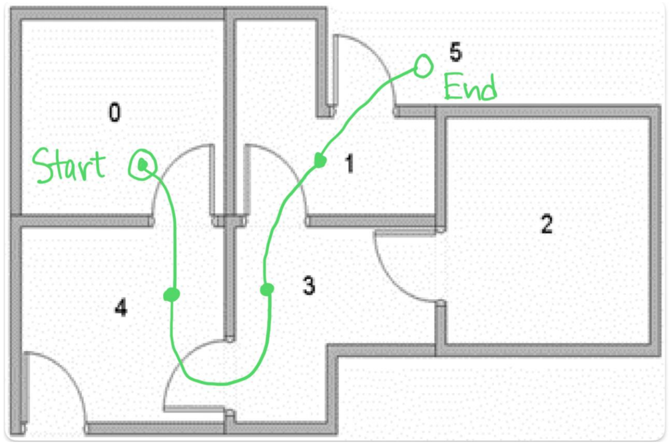


Figure 1: Q-table with reward from 4->5 changed to -0.9

We then modify the reward matrix R again to change the reward for going from state 4 to 5 to be -0.5. The new Q-table is as follows:

State/Action	0	1	2	3	4	5
0	0	0	0	0	2.8	0
1	0	0	0	3.2	0	5
2	0	0	0	3.2	0	0
3	0	4	2.56	0	2.8	0
4	2.24	0	0	3.2	0	3.5
5	0	4	0	0	2.8	5

The optimal policy is as follows:

State	Action
0	4
1	5
2	3
3	1
4	5
5	5

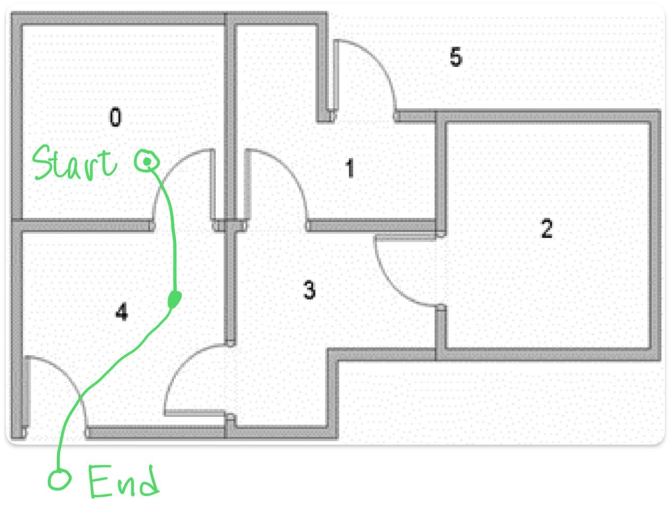


Figure 2: Q-table with reward from 4->5 changed to -0.5

Yes, the optimal path changes if we change the reward for going from state 4 to 5 to -0.5.

Contents

ME599 HW4 Problem 2

```
clc; clear; close all;

S = 1:6; % states
gamma = 0.8; % discount factor

% Rewards in the example problem
R = [ % rewards
        -1 -1 -1 -1 0 -1;
        -1 -1 -1 0 -1 100;
        -1 -1 -1 0 -1 -1;
        -1 0 0 -1 0 -1;
        0 -1 -1 0 -1 100;
        -1 0 -1 -1 0 100;
        ];

[Q, policy] = Q_learning(S, R, gamma);
disp('Optimal Q-table:');
disp(Q);
disp('Optimal policy:');
disp(policy);
```

```
Optimal Q-table:
     0     0     0     0   400     0
     0     0     0   320     0   500
     0     0     0   320     0     0
     0   400   256     0   400     0
   320     0     0   320     0   500
     0   400     0     0   400   500

Optimal policy:
     4
     5
     3
     1
     5
     5
```

part a

```
R = [ % rewards
        -1 -1 -1 -1 0 -1;
        -1 -1 -1 0 -1 1;
        -1 -1 -1 0 -1 -1;
        -1 0 0 -1 0 -1;
        0 -1 -1 0 -1 1;
        -1 0 -1 -1 0 1;
        ];

[Q_a, policy_a] = Q_learning(S, R, gamma);

disp('Optimal Q-table (part a):');
disp(Q_a);
disp('Optimal policy (part a):');
disp(policy_a);

% part b
R = [ % rewards
        -1 -1 -1 -1 0 -1;
        -1 -1 -1 0 -1 1;
        -1 -1 -1 0 -1 -1;
        -1 0 0 -1 0 -1;
        0 -1 -1 0 -1 -0.9;
        -1 0 -1 -1 0 1;
        ];

[Q_b1, policy_b1] = Q_learning(S, R, gamma);
disp('Optimal Q-table (part b1):');
disp(Q_b1);
disp('Optimal policy (part b1):');
disp(policy_b1);

% part c
R = [ % rewards
        -1 -1 -1 -1 0 -1;
        -1 -1 -1 0 -1 1;
        -1 -1 -1 0 -1 -1;
        -1 0 0 -1 0 -1;
        0 -1 -1 0 -1 -0.5;
        -1 0 -1 -1 0 1;
        ];

[Q_b2, policy_b2] = Q_learning(S, R, gamma);
disp('Optimal Q-table (part b2):');
disp(Q_b2);
disp('Optimal policy (part b2):');
disp(policy_b2);

function [Q, policy] = Q_learning(S, R, gamma)
Q = zeros(6, 6); % Q-table
tmax = 2000; % max number of iterations

% step 1: initialize
t = 1; % iteration counter
% DO NOT USE randsample(S,1) to select initial state
% https://www.mathworks.com/help/stats/randsample.html#d126e992298
x_k = datasample(S, 1); % initial state, pick randomly

while true
        % step 2: get action and reward
        % u_k = randsample(S,1); % select action randomly
        u_k = datasample(S(R(x_k, :) ~= -1), 1); % select action randomly from valid actions
        x_kp1 = u_k; % next state
        r_k = R(x_k, u_k); % reward
        t = t+1; % increment iteration counter

        % step 3: update Q-table
        Q(x_k, u_k) = r_k + gamma * max(Q(x_kp1, :));

        % step 4: check for max iterations
        if t > tmax
                break;
        else
                x_k = x_kp1; % update state
        end
end

% step 5: find optimal policy
[~, policy] = max(Q, [], 2); % find action with max Q-value for each state
policy = policy - 1; % convert to 0-indexed
end
```

```
Optimal Q-table (part a):
         0         0         0         0    4.0000         0
         0         0         0    3.2000         0    5.0000
         0         0         0    3.2000         0         0
         0    4.0000    2.5600         0    4.0000         0
    3.2000         0         0    3.2000         0    5.0000
         0    4.0000         0         0    4.0000    5.0000

Optimal policy (part a):
     4
     5
     3
     1
     5
     5

Optimal Q-table (part b1):
         0         0         0         0    2.5600         0
         0         0         0    3.2000         0    5.0000
         0         0         0    3.2000         0         0
         0    4.0000    2.5600         0    2.5600         0
    2.0480         0         0    3.2000         0    3.1000
         0    4.0000         0         0    2.5600    5.0000

Optimal policy (part b1):
     4
     5
     3
     1
     3
     5

Optimal Q-table (part b2):
         0         0         0         0    2.8000         0
         0         0         0    3.2000         0    5.0000
         0         0         0    3.2000         0         0
         0    4.0000    2.5600         0    2.8000         0
    2.2400         0         0    3.2000         0    3.5000
         0    4.0000         0         0    2.8000    5.0000

Optimal policy (part b2):
     4
     5
     3
     1
     5
     5
```

Published with MATLAB® R2023b