ME599 Homework 6 - Akshat Dubey

Problem 1

Given the general transfer function:

$$G(s) = \frac{b_0 + b_1 s + b_2 s^2 + \ldots + b_m s^m}{a_0 + a_1 s + a_2 s^2 + \ldots + a_n s^n}$$
$$\text{multiply by } \frac{a_n}{a_n} \text{ and abuse notation such that}$$
$$b_0 = \frac{b_0}{a_n} \ldots, \quad a_0 = \frac{a_0}{a_n} \ldots$$
$$G(s) = \frac{b_0 + b_1 s + b_2 s^2 + \ldots + b_m s^m}{a_0 + a_1 s + a_2 s^2 + \ldots a_{n-1} s^{n-1} + s^n}$$

we need to come up with the least squares formulation for the coefficients b_i and a_i from the frequency response $G(j\omega)$ collected from an experiment.

Problem 1.a

Let $G(j\omega) = G_R + jG_I$ be the complex frequency response, with G_R and G_I being the real and imaginary parts respectively. By setting $s = j\omega$, and using the complex frequency response, we can rewrite the equation given in the problem as

$$(a_0 + a_1 j\omega + a_2 (j\omega)^2 + \ldots a_{n-1}(j\omega)^{n-1} + (j\omega)^n)(G_R + jG_I) = (b_0 + b_1 j\omega + b_2 (j\omega)^2 + \ldots + b_m (j\omega)^m)$$

Noting that

$$(j\omega)^p = \begin{cases} (-1)^{p/2}\omega^p & \text{if } p \text{ is even} \\ j(-1)^{(p-1)/2}\omega^p & \text{if } p \text{ is odd} \end{cases}$$

we can rewrite the above equation as

$$\begin{aligned}(a_0 + a_1 j\omega + a_2(-1)^{2/2}\omega^2 + \ldots + a_{n-1}(j\omega)^{n-1} + (j\omega)^n)(G_R + jG_I) &= b_0 + b_1 j\omega + b_2(-1)^{2/2}\omega^2 + \ldots + b_m (j\omega)^m \\ (a_0 + a_1 j\omega - a_2\omega^2 + \ldots + a_{n-1}(j\omega)^{n-1} + (j\omega)^n)(G_R + jG_I) &= b_0 + b_1 j\omega - b_2\omega^2 + \ldots + b_m (j\omega)^m \\ a_0 G_R + a_0 jG_I + a_1 j\omega G_R - a_1\omega G_I - a_2\omega^2 G_R - a_2\omega^2 jG_I + \ldots & \\ +a_{n-1}(j\omega)^{n-1}G_R + a_{n-1}(j\omega)^{n-1}jG_I + (j\omega)^n G_R + (j\omega)^n G_I &= b_0 + b_1 j\omega - b_2\omega^2 + \ldots + b_m (j\omega)^m\end{aligned}$$

assuming that n and m are both even, we can separate the real and imaginary parts. The real part can be expressed as

$$\begin{aligned}a_0 G_R - a_1\omega G_I - a_2\omega^2 G_R + \ldots + a_{n-1}(j\omega)^{n-1}jG_I + (j\omega)^n G_R &= b_0 - b_2\omega^2 + b_4\omega^4 + \ldots + b_m (j\omega)^m \\ -a_0 G_R + a_1\omega G_I + a_2\omega^2 G_R + \ldots - a_{n-1}(j\omega)^{n-1}jG_I + b_0 - b_2\omega^2 + b_4\omega^4 + \ldots + b_m(j\omega)^m &= (j\omega)^n G_R\end{aligned}$$

Similarly, the imaginary part can be expressed as

$$\begin{aligned}a_0 G_I + a_1\omega G_R - a_2\omega^2 G_I + \ldots + a_{n-1}(j\omega)^{n-1}G_R + (j\omega)^n G_I &= b_1\omega - b_3\omega^3 + \ldots + b_{m-1}(j\omega)^{m-1} \\ -a_0 G_I - a_1\omega G_R + a_2\omega^2 G_I + \ldots - a_{n-1}(j\omega)^{n-1}G_R + b_1\omega - b_3\omega^3 + \ldots + b_{m-1}(j\omega)^{m-1} &= (j\omega)^n G_I\end{aligned}$$

These equations can be expressed in matrix form as

$$\begin{bmatrix} -G_R & \omega G_I & \dots & -(j\omega)^{n-1}jG_I & 1 & 0 & -\omega^2 & \dots & 0 & (j\omega)^m \\ -G_I & -\omega G_R & \dots & -(j\omega)^{n-1}G_R & 0 & \omega & 0 & \dots & (j\omega)^{m-1} & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \\ b_0 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} (j\omega)^n G_R \\ (j\omega)^n G_I \end{bmatrix}$$

In the case that n is odd, in the coefficient matrix the coefficient of a_{n-1}, $(j\omega)^{n-1}G_R$ would be in the real part and $(j\omega)^{n-1}jG_I$ would be in the imaginary part. Similarly for the vector in the RHS, $(j\omega)^n G_I$ would be the real part and $(j\omega)^n G_R$ would be the imaginary part. The real part would change to

$$\begin{aligned} a_0G_R - a_1\omega G_I - a_2\omega^2 G_R + \dots + a_{n-1}(j\omega)^{n-1}G_R + (j\omega)^n G_I &= b_0 - b_2\omega^2 + b_4\omega^4 + \dots + b_m(j\omega)^m \\ -a_0G_R + a_1\omega G_I + a_2\omega^2 G_R + \dots - a_{n-1}(j\omega)^{n-1}G_R + b_0 - b_2\omega^2 + b_4\omega^4 + \dots + b_m(j\omega)^m &= (j\omega)^n G_I \end{aligned}$$

Similarly, the imaginary part can be expressed as

$$\begin{aligned} a_0G_I + a_1\omega G_R - a_2\omega^2 G_I + \dots + a_{n-1}(j\omega)^{n-1}jG_I + (j\omega)^n G_R &= b_1\omega - b_3\omega^3 + \dots + b_{m-1}(j\omega)^{m-1} \\ -a_0G_I - a_1\omega G_R + a_2\omega^2 G_I + \dots - a_{n-1}(j\omega)^{n-1}jG_I + b_1\omega - b_3\omega^3 + \dots + b_{m-1}(j\omega)^{m-1} &= (j\omega)^n G_R \end{aligned}$$

In the case that m is odd, in the coefficient matrix the coefficient of b_{m-1}, $(j\omega)^{m-1}$ would end up being in the real part and the coefficient of b_m, $(j\omega)^m$ would end up being in the imaginary part. The real part can be expressed as

$$\begin{aligned} a_0G_R - a_1\omega G_I - a_2\omega^2 G_R + \dots + a_{n-1}(j\omega)^{n-1}jG_I + (j\omega)^n G_R &= b_0 - b_2\omega^2 + b_4\omega^4 + \dots + b_{m-1}(j\omega)^{m-1} \\ -a_0G_R + a_1\omega G_I + a_2\omega^2 G_R + \dots - a_{n-1}(j\omega)^{n-1}jG_I + b_0 - b_2\omega^2 + b_4\omega^4 + \dots + b_{m-1}(j\omega)^{m-1} &= (j\omega)^n G_R \end{aligned}$$

Similarly, the imaginary part can be expressed as

$$\begin{aligned} a_0G_I + a_1\omega G_R - a_2\omega^2 G_I + \dots + a_{n-1}(j\omega)^{n-1}G_R + (j\omega)^n G_I &= b_1\omega - b_3\omega^3 + \dots + b_m(j\omega)^m \\ -a_0G_I - a_1\omega G_R + a_2\omega^2 G_I + \dots - a_{n-1}(j\omega)^{n-1}G_R + b_1\omega - b_3\omega^3 + \dots + b_m(j\omega)^m &= (j\omega)^n G_I \end{aligned}$$

In case both m and n are odd, we can follow the same rules as defined above to change the matrices and the vector in the RHS.

Hence, we can use the above equations to construct matrices that equate the real and imaginary parts to each other and solve them to get the least squares solution.

Problem 1.b

Assuming m and n are even, we can stack the equations from the previous part for multiple frequencies(ω_1 to ω_k), where $G(j\omega_k) = G_R(\omega_k) + jG_I(\omega_k)$ is given and express the problem as

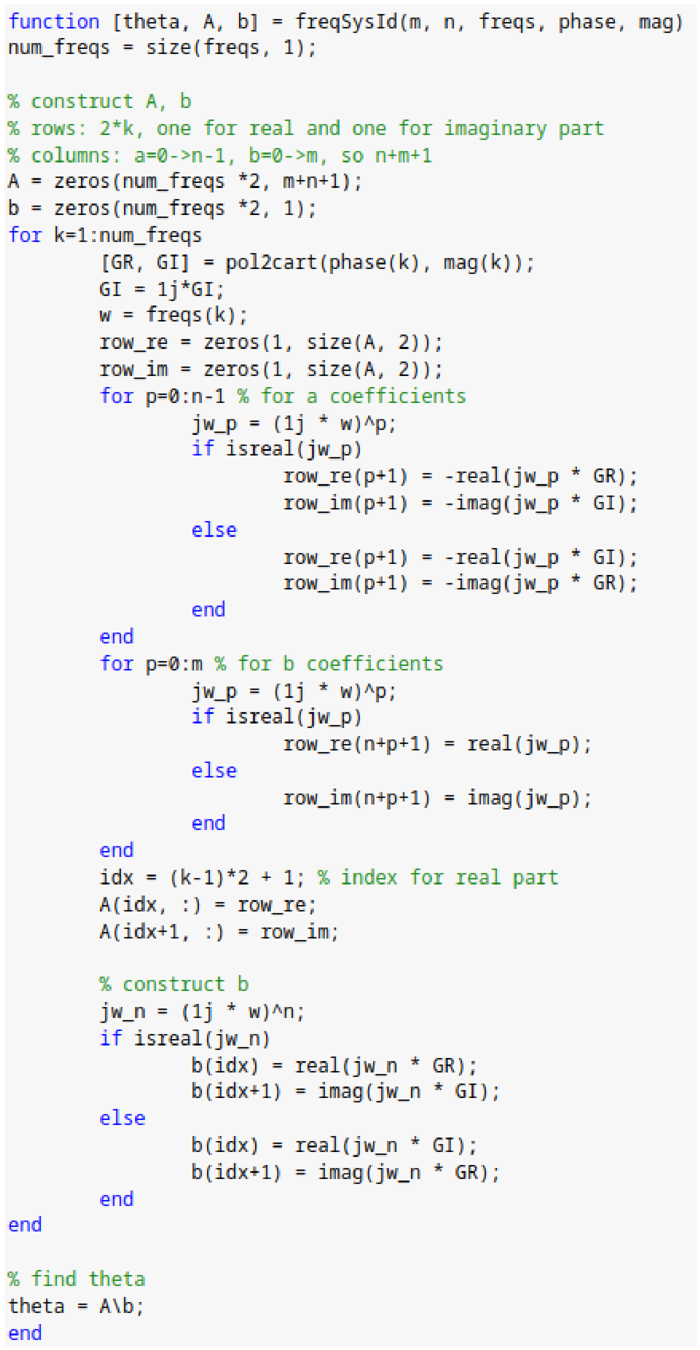
$$\underbrace{\begin{bmatrix} -G_R(\omega_1) & \omega_1 G_I(\omega_1) & \dots & -(j\omega_1)^{n-1}jG_I(\omega_1) & 1 & 0 & -\omega_1^2 & \dots & 0 & (j\omega_1)^m \\ -G_I(\omega_1) & -\omega_1 G_R(\omega_1) & \dots & -(j\omega_1)^{n-1}G_R(\omega_1) & 0 & \omega_1 & 0 & \dots & (j\omega_1)^{m-1} & 0 \\ -G_R(\omega_2) & \omega_2 G_I(\omega_2) & \dots & -(j\omega_2)^{n-1}jG_I(\omega_2) & 1 & 0 & -\omega_2^2 & \dots & 0 & (j\omega_2)^m \\ -G_I(\omega_2) & -\omega_2 G_R(\omega_2) & \dots & -(j\omega_2)^{n-1}G_R(\omega_2) & 0 & \omega_2 & 0 & \dots & (j\omega_2)^{m-1} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -G_R(\omega_k) & \omega_k G_I(\omega_k) & \dots & -(j\omega_k)^{n-1}jG_I(\omega_k) & 1 & 0 & -\omega_k^2 & \dots & 0 & (j\omega_k)^m \\ -G_I(\omega_k) & -\omega_k G_R(\omega_k) & \dots & -(j\omega_k)^{n-1}G_R(\omega_k) & 0 & \omega_k & 0 & \dots & (j\omega_k)^{m-1} & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \\ b_0 \\ \vdots \\ b_m \end{bmatrix}}_{\theta} = \underbrace{\begin{bmatrix} (j\omega_1)^n G_R(\omega_1) \\ (j\omega_1)^n G_I(\omega_1) \\ (j\omega_2)^n G_R(\omega_2) \\ (j\omega_2)^n G_I(\omega_2) \\ \vdots \\ (j\omega_k)^n G_R(\omega_k) \\ (j\omega_k)^n G_I(\omega_k) \end{bmatrix}}_{b}$$

If either m or n is odd, we can follow the same rules as defined at the end of part (a) to change the matrices A and b. We solve for the coeffients a_i and b_i in θ using the least squares method, where

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

Problem 1.c

The MATLAB scrpt is included as an appendix, and constructs the matrices A and b as described above, then solves for θ using the least squares method.



```matlab
function [theta, A, b] = freqSysId(m, n, freqs, phase, mag)
num_freqs = size(freqs, 1);

% construct A, b
% rows: 2*k, one for real and one for imaginary part
% columns: a=0->n-1, b=0->m, so n+m+1
A = zeros(num_freqs *2, m+n+1);
b = zeros(num_freqs *2, 1);
for k=1:num_freqs
        [GR, GI] = pol2cart(phase(k), mag(k));
        GI = 1j*GI;
        w = freqs(k);
        row_re = zeros(1, size(A, 2));
        row_im = zeros(1, size(A, 2));
        for p=0:n-1 % for a coefficients
                jw_p = (1j * w)^p;
                if isreal(jw_p)
                        row_re(p+1) = -real(jw_p * GR);
                        row_im(p+1) = -imag(jw_p * GI);
                else
                        row_re(p+1) = -real(jw_p * GI);
                        row_im(p+1) = -imag(jw_p * GR);
                end
        end
        for p=0:m % for b coefficients
                jw_p = (1j * w)^p;
                if isreal(jw_p)
                        row_re(n+p+1) = real(jw_p);
                else
                        row_im(n+p+1) = imag(jw_p);
                end
        end
        idx = (k-1)*2 + 1; % index for real part
        A(idx, :) = row_re;
        A(idx+1, :) = row_im;

        % construct b
        jw_n = (1j * w)^n;
        if isreal(jw_n)
                b(idx) = real(jw_n * GR);
                b(idx+1) = imag(jw_n * GI);
        else
                b(idx) = real(jw_n * GI);
                b(idx+1) = imag(jw_n * GR);
        end
end

% find theta
theta = A\b;
end
```

Figure 1: Code

Problem 1.d

In the file `freq_resp_data.mat`, we are given the frequency response data for 77 different frequencies, so $k = 77$. We given the phase offset and amplitude of the output of the system, so we can use that to construct $G(j\omega)$, where

$$G_R = \text{amplitude} \cdot \cos(\text{phase offset})$$
$$G_I = \text{amplitude} \cdot \sin(\text{phase offset})$$

We can construct the A and b matrices as described above, but in a programmatic way that allows us to input odd values of m and n and solve for θ to find the model of the system.

We can then loop over multiple values of m and n, making sure we only try values that make the transfer function proper $m \leq n$ to find the values that provide the best fit to the data, which are values that minimize

$$||A\theta - b||_2^2$$

If we vary m and n between 1 to 6, we find that there is a good fit for multiple m and n values(in bold):

	n=1	n=2	n=3	n=4	n=5	n=6
m=1	9.3089e+000	15.2356e+000	993.8861e+003	583.3088e+006	214.0905e+009	49.7797e+012
m=2	Inf	14.9863e+000	**237.7279e-012**	583.3088e+006	214.0905e+009	49.7797e+012
m=3	Inf	Inf	**179.9161e-012**	138.0471e-009	214.0905e+009	49.7797e+012
m=4	Inf	Inf	Inf	122.6861e-009	125.4958e-006	49.7797e+012
m=5	Inf	Inf	Inf	Inf	255.8723e+000	4.9544e-003
m=6	Inf	Inf	Inf	Inf	Inf	118.4789e+003

We select $m = 2$ and $n = 3$, since it is the smallest order that gives us a good fit. This gives us the transfer function

$$G(s) = \frac{s^2 + 10s + 10}{s^3 + s^2 + 0.2s + 40}$$

And the bode plot of the approximated system along with the data is

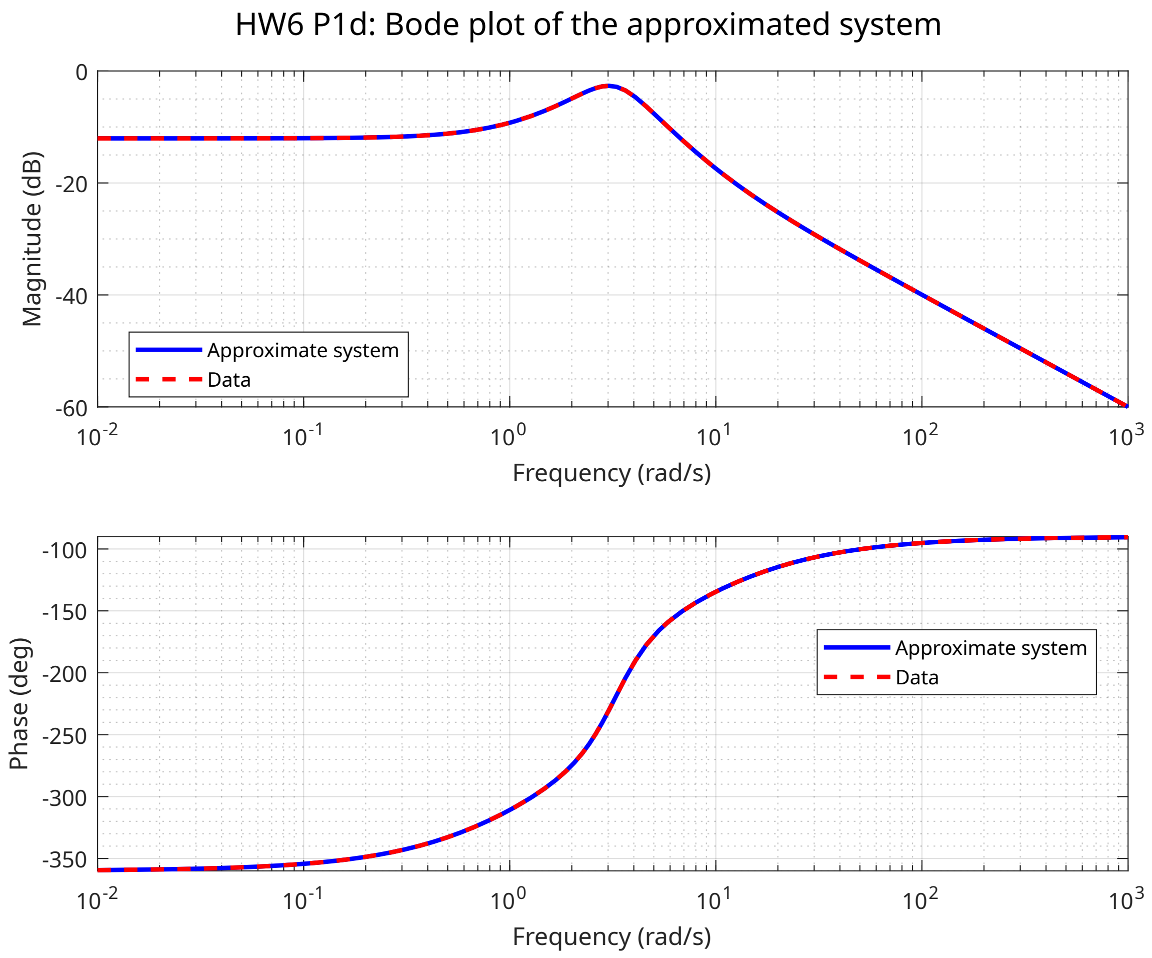


Figure 2: HW6 P1d Bode plot of approximate system

The approximated system matches our data quite well as seen in the plot, and hence we can conclude that our system identification is correct.

ME599 HW6 Problem 1

```
clc; clear; close all;

load('freq_resp_data.mat');
```

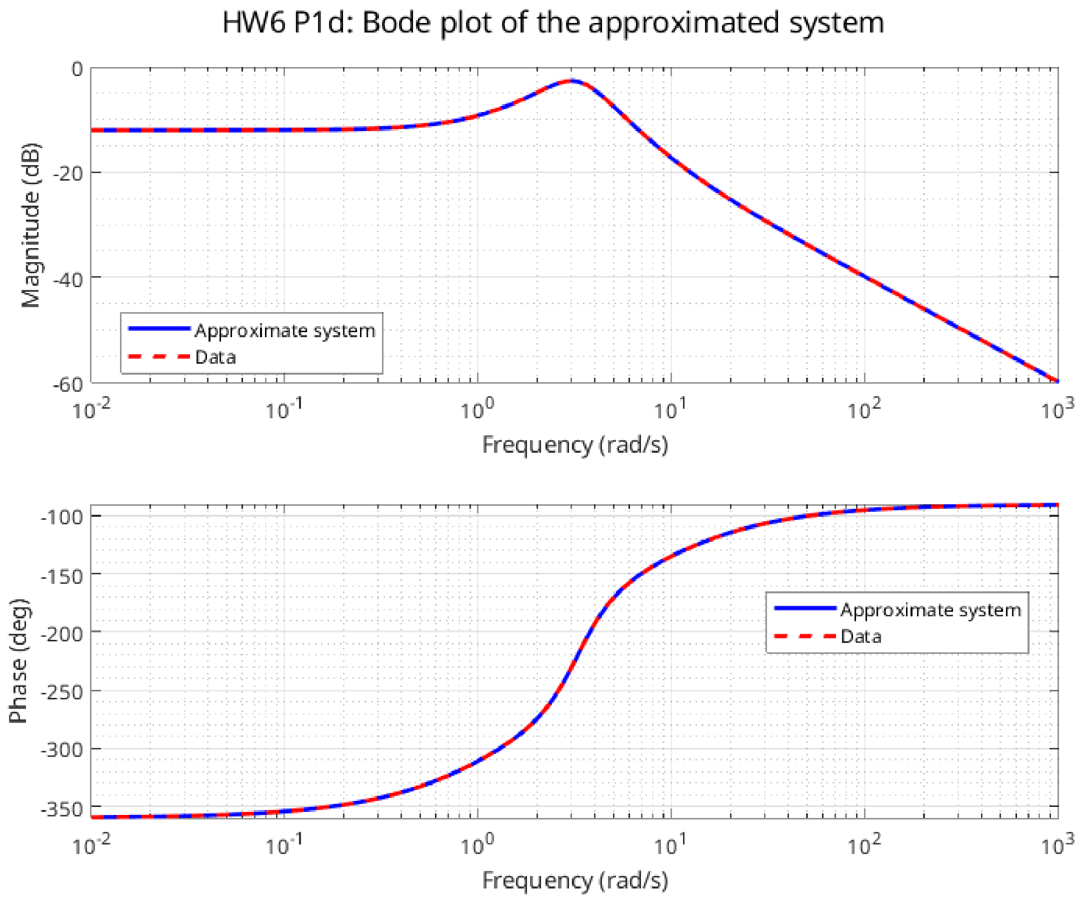
loop through m and n and check least squares error

```
max_order = 6;
errs = ones(max_order, max_order)*Inf;
for n=1:max_order
    for m=1:n
        [theta, A, b] = freqSysId(m, n, W, PHASE_RAD_M, MAG_M);
        errs(m, n) = norm(A*theta - b);
    end
end

fprintf("%s\n", formattedDisplayText(errs,"NumericFormat","shortEng"));

m = 2; n = 3; % best from experiment
[theta, A, b] = freqSysId(m, n, W, PHASE_RAD_M, MAG_M);
fprintf('HW6 P1d: Approximate system transfer function:\n');
model = getModelFromTheta(m, theta)

% plot
% bode_opts = bodeoptions;
% bode_opts.Title.String = 'HW6 P1d: Bode plot of the approximated system';
% bode_opts.Title.FontSize = 14;
[mags, phases, freqs] = bode(model, W);
mags = squeeze(mags); phases = squeeze(phases);
% NOTE: saving doesnt work, go to the plot and save it manually

fig = figure;
sgtitle('HW6 P1d: Bode plot of the approximated system');

subplot(2, 1, 1)
semilogx(freqs, db(mags), 'b', 'LineWidth', 2, 'DisplayName', 'Approximate
system')
hold on;
semilogx(W, db(MAG_M), 'r--', 'LineWidth', 2, 'DisplayName', 'Data')
xlabel('Frequency (rad/s)');
ylabel('Magnitude (dB)');
ylim([-60 0]);
legend('Location', 'best');
grid on; grid minor;

subplot(2, 1, 2)
semilogx(freqs, phases, 'b', 'LineWidth', 2, 'DisplayName', 'Approximate
system')
```

```
hold on;
semilogx(W, rad2deg(PHASE_RAD_M), 'r--', 'LineWidth', 2, 'DisplayName',
'Data')
xlabel('Frequency (rad/s)');
ylabel('Phase (deg)');
ylim([-360 -90]);
legend('Location', 'best');
grid on; grid minor;

fig.Position(3:4) = [800 600];
saveas(fig, 'figs/hw6p1d_bode.svg');

function [theta, A, b] = freqSysId(m, n, freqs, phase, mag)
num_freqs = size(freqs, 1);

% construct A, b
% rows: 2*k, one for real and one for imaginary part
% columns: a=0->n-1, b=0->m, so n+m+1
A = zeros(num_freqs *2, m+n+1);
b = zeros(num_freqs *2, 1);
for k=1:num_freqs
    [GR, GI] = pol2cart(phase(k), mag(k));
    GI = 1j*GI;
    w = freqs(k);
    row_re = zeros(1, size(A, 2));
    row_im = zeros(1, size(A, 2));
    for p=0:n-1 % for a coefficients
        jw_p = (1j * w)^p;
        if isreal(jw_p)
            row_re(p+1) = -real(jw_p * GR);
            row_im(p+1) = -imag(jw_p * GI);
        else
            row_re(p+1) = -real(jw_p * GI);
            row_im(p+1) = -imag(jw_p * GR);
        end
    end
    for p=0:m % for b coefficients
        jw_p = (1j * w)^p;
        if isreal(jw_p)
            row_re(n+p+1) = real(jw_p);
        else
            row_im(n+p+1) = imag(jw_p);
        end
    end
    idx = (k-1)*2 + 1; % index for real part
    A(idx, :) = row_re;
    A(idx+1, :) = row_im;

    % construct b
    jw_n = (1j * w)^n;
    if isreal(jw_n)
        b(idx) = real(jw_n * GR);
        b(idx+1) = imag(jw_n * GI);
    else
```

```
        b(idx) = real(jw_n * GI);
        b(idx+1) = imag(jw_n * GR);
    end
end

% find theta
theta = A\b;
end

function model = getModelFromTheta(m, theta)
% tf needs higher orders first, so enumerate backwards
theta = flip(theta);
numerator = theta(1:m+1)'; % for b's
denominator = [1, theta(m+2:end)']; % for a's, add 1 as highest order
coefficient
model = tf(numerator, denominator);
end
```

```
Warning: Rank deficient, rank = 6, tol =  3.560671e-08.
Warning: Rank deficient, rank = 7, tol =  3.440996e-05.
Warning: Rank deficient, rank = 8, tol =  3.423102e-02.
Warning: Rank deficient, rank = 7, tol =  3.441143e-05.
Warning: Rank deficient, rank = 7, tol =  3.441143e-05.
Warning: Rank deficient, rank = 8, tol =  3.423102e-02.
Warning: Rank deficient, rank = 6, tol =  3.420129e+01.
Warning: Rank deficient, rank = 8, tol =  3.423239e-02.
Warning: Rank deficient, rank = 8, tol =  3.423239e-02.
Warning: Rank deficient, rank = 8, tol =  3.423239e-02.
Warning: Rank deficient, rank = 6, tol =  3.420129e+01.
Warning: Rank deficient, rank = 5, tol =  3.419605e+04.
  Columns 1 through 4

     9.3089e+000    15.2356e+000   993.8861e+003   583.3088e+006
             Inf    14.9863e+000   237.7279e-012   583.3088e+006
             Inf             Inf   179.9161e-012   138.0471e-009
             Inf             Inf             Inf   122.6861e-009
             Inf             Inf             Inf             Inf
             Inf             Inf             Inf             Inf

  Columns 5 through 6

   214.0905e+009    49.7797e+012
   214.0905e+009    49.7797e+012
   214.0905e+009    49.7797e+012
   125.4958e-006    49.7797e+012
   255.8723e+000     4.9544e-003
             Inf   118.4789e+003

HW6 P1d: Approximate system transfer function:

model =

     s^2 + 10 s + 10
  ----------------------
  s^3 + s^2 + 0.2 s + 40

Continuous-time transfer function.
```



Published with MATLAB® R2023b

Problem 2

For this problem, we will try to identify the transfer function for the model provided in Homework 4 in the `piezoNozzle.p` file. The model is a linear system as determined in Hwk 4 with a timestep of 0.1 μs.

Problem 2.a

We can describe this model by a difference equation. A difference equation is similar to a differential equation, but in discrete time, and it relates the current output of the system to past outputs and inputs. A difference equation can be expressed in the form:

$$y(k) + a_1 y(k-1) + \ldots + a_n y(k-n) = b_1 u(k) + \ldots + b_m u(k-m)$$

The order of the system is determined by m and n, which are an indication of how far back in time the control inputs and outputs are relevant.

Since the model is linear, we can use ARX(AutoRegressive with Extra input) model to identify the system. To do that, we first send a random sequence of inputs into the system and generate outputs. Because we need to compare our results with `P_ref.mat`, we have to use a sample time of 0.1s as was used when generating the reference. I generated 500 random samples of $u(k)$ and used this sequence to get 500 samples of $y(k)$.

Now, for each generated $y(k)$, we want to find the coefficients a_i, b_i that when multiplied by the previous u s and y s will get us an answer close to the $y(k)$. Based on what order we choose, we will have $n+m$ coefficients and equations. So that we have $n+m$ independent equations, we form equations between time instances $k = N_1$ and $k = N_2$ where $N_2 - N_1 > max(m,n)$ and $N_1 > max(m,n)$.

We can formulte this as a least squares problem:

$$\underbrace{\begin{bmatrix} y(N_1) \\ \vdots \\ y(N_2) \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} -y(N_1-1) & \ldots & -y(N_1-n) & u(N_1) & \ldots & u(N_1-m) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ -y(N_2-1) & \ldots & -y(N_2-n) & u(N_2) & \ldots & u(N_2-m) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_1 \\ \vdots \\ a_n \\ b_1 \\ \vdots \\ b_m \end{bmatrix}}_{\theta}$$

and the optimal solution is given by:

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

This least squares problem was solved with m and n in the range of 5 to 10 as suggested in the problem to get the coefficients. To determine the best order, we select the m and n that minimize the following cost function:

$$||b - A\theta^*||_2$$

Once we have the best order and its coefficients, the difference equation can be converted to a discrete time transfer function using the Z-transform

$$y(k) + a_1 y(k-1) + \ldots + a_n y(k-n) = b_1 u(k) + \ldots + b_m u(k-m)$$

Taking Z-transform,

$$Y(z) + a_1 z^{-1} Y(z) + \ldots + a_n z^{-n} Y(z) = b_1 U(z) + \ldots + b_m z^{-m} U(z)$$

$$Y(z)(1 + a_1 z^{-1} + \ldots + a_n z^{-n}) = U(z)(b_1 + b_2 z^{-1} + \ldots + b_m z^{-m})$$

$$\frac{Y(z)}{U(z)} = \frac{b_1 + b_2 z^{-1} + \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + \ldots + a_n z^{-n}}$$

Converting to polynomial form, with the higest order being n

$$\frac{Y(z)}{U(z)} = \frac{b_1 z^n + b_2 z^{n-1} + \ldots + b_m z^{n-m}}{z^n + a_1 z^{n-1} + \ldots + a_n}$$

where $b_i = 0$ for $i > m$ since $m \leq n$ for a proper transfer function

(basically all coefficients of orders of the numerator that are smaller than m are zero)

This way, we can construct a transfer function in MATLAB, with numerator as $[b_1, b_2, \ldots, b_m, 0 \times (n-m+1)]$ and denominator as $[1, a_1, \ldots, a_n]$. The best m and n are determined to be 10 for both as can be seen in the table below while making sure that the transfer function is proper ($m \leq n$).

	$n = 5$	$n = 6$	$n = 7$	$n = 8$	$n = 9$	$n = 10$
$n = 5$	272.2843e+000	214.8993e+000	206.1317e+000	190.9935e+000	135.1521e+000	95.8259e+000
$n = 6$	Inf	169.2370e+000	167.9149e+000	127.7042e+000	92.6606e+000	84.5737e+000
$n = 7$	Inf	Inf	39.6386e+000	27.9621e+000	22.2111e+000	19.6407e+000
$n = 8$	Inf	Inf	Inf	2.0156e+000	2.0135e+000	2.0064e+000
$n = 9$	Inf	Inf	Inf	Inf	544.9197e-003	543.4382e-003
$n = 10$	Inf	Inf	Inf	Inf	Inf	**218.9867e-003**

And the transfer function is as follows:

$$G(z) = \frac{5.516e-07z^{10} + 0.01165z^9 - 0.1152z^8 + 0.04481z^7 + 0.4392z^6 - 0.662z^5 + 0.1977z^4 + 0.355z^3 - 0.4052z^2 + 0.1341z}{z^{10} - 2.561z^9 + 2.711z^8 - 1.341z^7 - 0.5648z^6 + 1.817z^5 - 1.777z^4 + 0.851z^3 - 0.1909z^2 + 0.05781z - 1.865e-5}$$

When using MATLAB's system ID toolbox, we need to set $n_a = n$, $n_b = m$ and $n_k = 0$ since we have no dead time in the system and $y(k)$ is affected by $u(k)$ directly. For MATLAB's toolbox, we can make our decision based on the MSE, $m = n = 10$ as shown in bold in the table below.

	$n = 5$	$n = 6$	$n = 7$	$n = 8$	$n = 9$	$n = 10$
$n = 5$	148.2774e+000	92.3634e+000	84.9805e+000	72.9571e+000	36.5322e+000	18.3652e+000
$n = 6$	Inf	57.2823e+000	56.3908e+000	32.6167e+000	17.1720e+000	14.3054e+000
$n = 7$	Inf	Inf	3.1424e+000	1.5638e+000	986.6686e-003	771.5156e-003
$n = 8$	Inf	Inf	Inf	8.1254e-003	8.1082e-003	8.0512e-003
$n = 9$	Inf	Inf	Inf	Inf	593.8750e-006	590.6502e-006
$n = 10$	Inf	Inf	Inf	Inf	Inf	**95.9104e-006**

This gives us the transfer function

$$G(z) = \frac{5.516e-07z^{10} + 0.01165z^9 - 0.1152z^8 + 0.04481z^7 + 0.4392z^6 - 0.662z^5 + 0.1977z^4 + 0.355z^3 - 0.4052z^2 + 0.1341z}{z^{10} - 2.561z^9 + 2.711z^8 - 1.341z^7 - 0.5648z^6 + 1.817z^5 - 1.777z^4 + 0.851z^3 - 0.1909z^2 + 0.05781z - 1.865e-5}$$

Which is the same as the one we got from the doing ARX manually. Hence, we can assume that the manual ARX method is correct.

In the following plots, P is the pressure wave generated by `V_example.mat` using the actual system dynamics, P_{manual} is the pressure wave generated by the system identified by manual ARX method, and P_{matlab} is the pressure wave generated by the system identified by MATLAB's system ID toolbox.

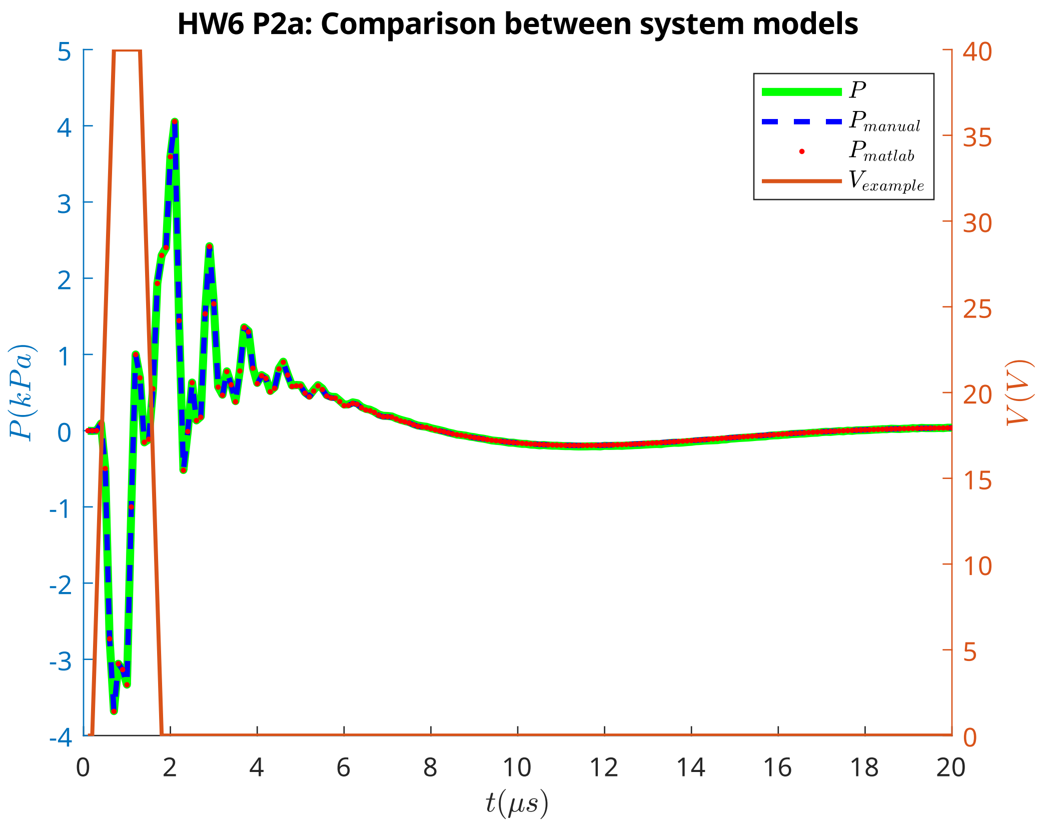


Figure 1: Comparison betwen system models

We can see that both the manual and MATLAB ARX methods give us a very good approximation of the system, and both models have similar norm of errors:

- Manual ARX($||P - P_{manual}||_2$): 0.134917
- MATLAB ARX($||P - P_{matlab}||_2$): 0.134917

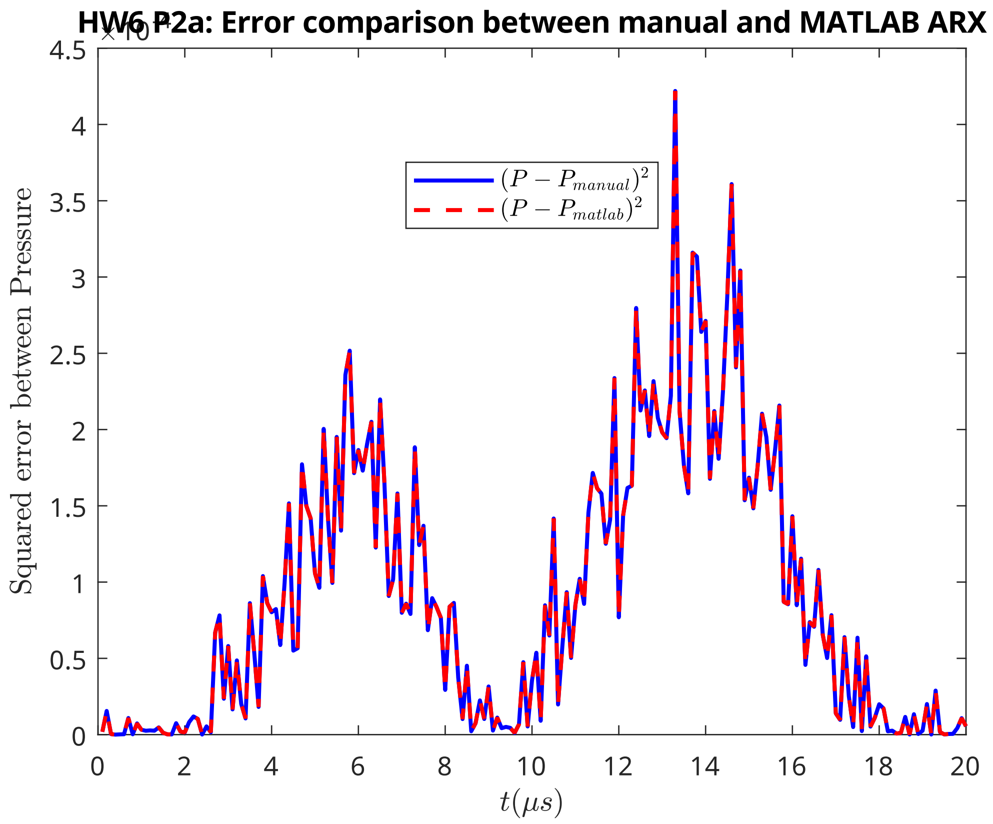


Figure 2: Error comparison between manual and MATLAB ARX

Problem 2.b

For this problem, we want to design a controller such that the system produces the desred pressure wave `P_ref.mat`. We start by constructing a state space representation and then making an impulse response matrix to solve the least squares solution to compute the best control input $V^*(t)$.

The state space representation of the system using can be found by using MATLAB's `ss` function. We can then use this state space representation to compute the impulse response matrix.

$$\underbrace{\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ \vdots \\ y(N-1) \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ CB & 0 & \dots & 0 & 0 \\ CAB & CB & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ CA^{N-2}B & CA^{N-3}B & \dots & CB & 0 \end{bmatrix}}_{\mathcal{G}} \underbrace{\begin{bmatrix} u(0) \\ u(1) \\ u(2) \\ \vdots \\ u(N-1) \end{bmatrix}}_{u} + \begin{bmatrix} Cx(0) \\ CAx(0) \\ CA^2x(0) \\ \vdots \\ CA^{N-1}x(0) \end{bmatrix}$$

Assuming $x(0) = 0$, we can ignore the second term and thus $y = \mathcal{G}u$. We can then use the pseudoinverse of $\mathcal{G}$ to compute the best control input $V^*(t)$ that will give us the desired pressure wave.

$$V^*(t) = \mathcal{G}^+ P_{ref}$$

Where $\mathcal{G}^+$ is the pseudoinverse of $\mathcal{G}$.

The $V^*(t)$ is then used to compute the pressure wave $P(t)$ using the system dynamics in `piezoNozzle.p`. We can see that the pressure wave generated by the system using $V^*(t)$ is very close to the desired pressure wave, so the controller is working well.

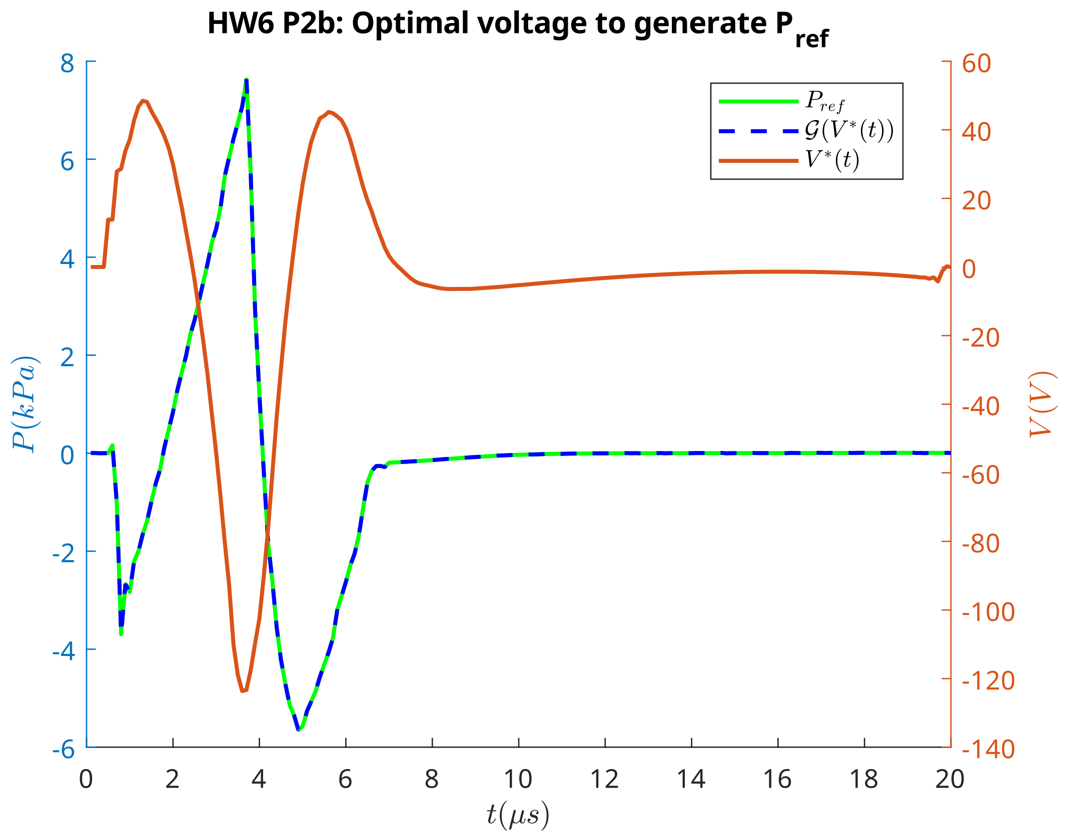


Figure 3: Comparison between desired and generated pressure wave

ME599 HW6 Problem 2

```
clc; clear; close all;

load('P_ref.mat');
load('V_example.mat');
```

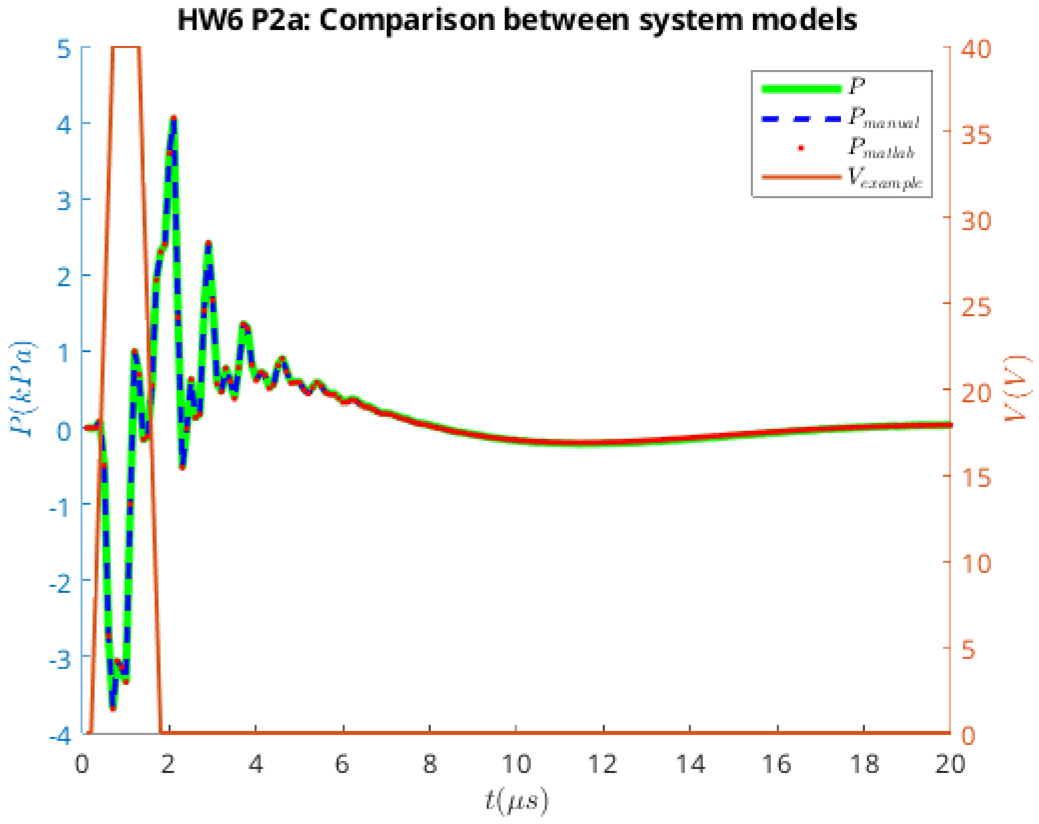
part a, identify the system using ARX

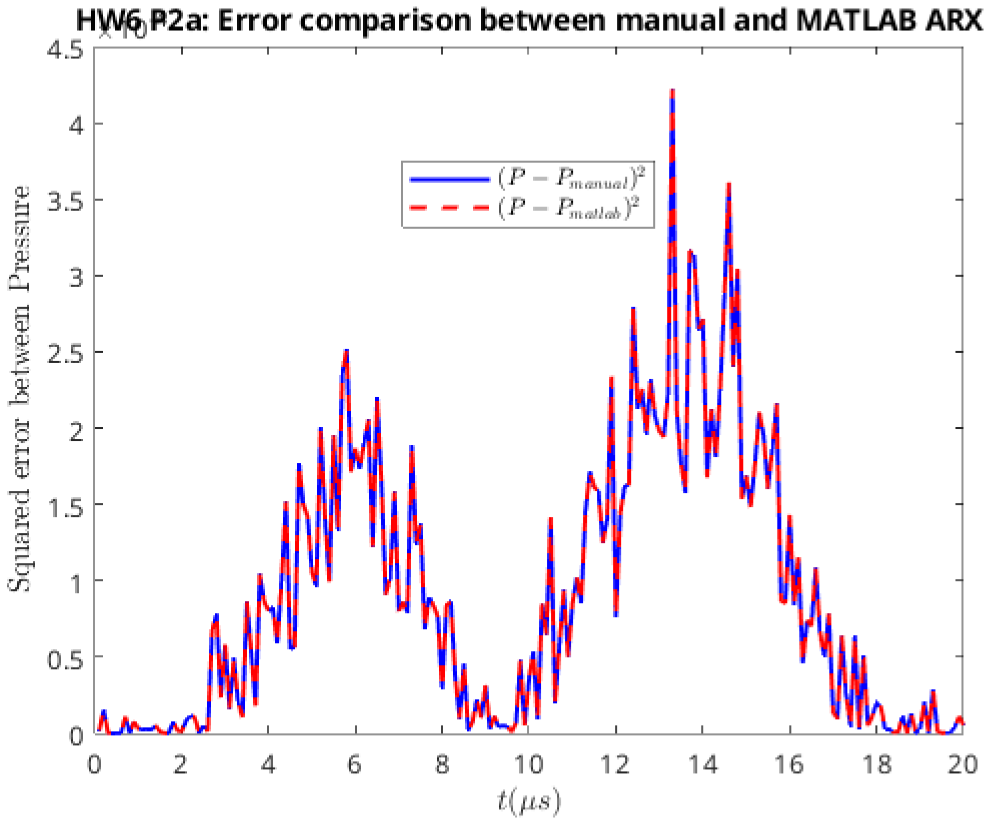
```
ts = 0.1; % time step in microseconds
N_train = 500; % simulation duration

% generate random input
rng(1);
u_train = rand(N_train, 1) * 500; % unif rand up to 500V
y_train = piezo_nozzle(u_train,N_train); % simulate pressure evolution
mat_sysid_train = iddata(y_train, u_train, ts);

test_orders = 5:10;
num_orders = length(test_orders);
errs_man = ones(num_orders, num_orders)*Inf; % for manual system ID
errs_mat = ones(num_orders, num_orders)*Inf; % for matlab's toolbox system ID
for n_idx=1:num_orders
    for m_idx=1:n_idx
        [theta, A, b] = timeSysId(test_orders(m_idx), test_orders(n_idx),
u_train, y_train);
        errs_man(m_idx, n_idx) = norm(A*theta - b);
        model_mat = arx(mat_sysid_train, [test_orders(n_idx),
test_orders(m_idx), 0]);
        errs_mat(m_idx, n_idx) = model_mat.Report.Fit.MSE;
    end
end
fprintf("Manual %s\n",
formattedDisplayText(errs_man,"NumericFormat","shortEng"));
fprintf("Matlab toolbox %s\n",
formattedDisplayText(errs_mat,"NumericFormat","shortEng"));

m=10; n=10; % best from experiment
[theta, ~, ~] = timeSysId(m, n, u_train, y_train);
fprintf('HW6 P2a: Transfer function (Manual ARX):\n');
model_man = getModelFromTheta(m, n, theta, ts)
fprintf('HW6 P2a: Transfer function (MATLAB ARX):\n');
% arx will give in the z^-1 form, so get the num and den first
[num_mat, den_mat]= tfdata(tf(arx(mat_sysid_train, [n, m, 0])));
% flip the coefficients to get the polynomial form
model_mat = tf(fliplr(num_mat), fliplr(den_mat), ts)


% plot
N_ex = length(V_example);
tspan = ts*(1:N_ex);
P = piezo_nozzle(V_example, N_ex);
```

```
P_man = lsim(model_man, V_example, tspan);
P_mat = lsim(model_mat, V_example, tspan);

fig = figure;
yyaxis left;
hold on;
plot(tspan, P, 'g', 'LineWidth', 3, 'DisplayName', '$P$');
plot(tspan, P_man, '--b', 'LineWidth', 2, 'DisplayName', '$P_{manual}$');
plot(tspan, P_mat, '.r', 'LineWidth', 1.5, 'DisplayName', '$P_{matlab}$');
ylabel("$P(kPa)$", 'Interpreter', 'latex');
yyaxis right;
plot(tspan, V_example, 'LineWidth', 1.5, 'DisplayName', '$V_{example}$');
ylabel("$V(V)$", 'Interpreter', 'latex');
xlabel("$t(\mu s)$", 'Interpreter', 'latex');
legend('Location', 'best', 'Interpreter', 'latex');
title("HW6 P2a: Comparison between system models");
saveas(fig, 'figs/hw6p2a_vex.svg');

fig = figure;
plot(tspan, (P-P_man).^2, 'b', 'LineWidth', 1.5, 'DisplayName', '$(P-
P_{manual})^2$');
hold on;
plot(tspan, (P-P_mat).^2, '--r', 'LineWidth', 1.5, 'DisplayName', '$(P-
P_{matlab})^2$');
ylabel("Squared error between Pressure", 'Interpreter', 'latex');
xlabel("$t(\mu s)$", 'Interpreter', 'latex');
legend('Location', 'best', 'Interpreter', 'latex');
title("HW6 P2a: Error comparison between manual and MATLAB ARX");
saveas(fig, 'figs/hw6p2a_err.svg');

fprintf('HW6 P2a: Norm of errors from expected pressure:\n - Manual: %f \n -
MATLAB: %f\n', norm(P-P_man), norm(P-P_mat));
```

```
Manual   Columns 1 through 4

   272.2843e+000   214.8993e+000   206.1317e+000   190.9935e+000
             Inf   169.2370e+000   167.9149e+000   127.7042e+000
             Inf             Inf    39.6386e+000    27.9621e+000
             Inf             Inf             Inf     2.0156e+000
             Inf             Inf             Inf             Inf
             Inf             Inf             Inf             Inf

  Columns 5 through 6

   135.1521e+000    95.8259e+000
    92.6606e+000    84.5737e+000
    22.2111e+000    19.6407e+000
     2.0135e+000     2.0064e+000
   544.9197e-003   543.4382e-003
             Inf   218.9867e-003

Matlab toolbox   Columns 1 through 4

   148.2774e+000    92.3634e+000    84.9805e+000    72.9571e+000
```

```
            Inf    57.2823e+000    56.3908e+000    32.6167e+000
            Inf             Inf     3.1424e+000     1.5638e+000
            Inf             Inf             Inf     8.1254e-003
            Inf             Inf             Inf             Inf
            Inf             Inf             Inf             Inf

  Columns 5 through 6

   36.5322e+000    18.3652e+000
   17.1720e+000    14.3054e+000
  986.6686e-003   771.5156e-003
    8.1082e-003     8.0512e-003
  593.8750e-006   590.6502e-006
            Inf    95.9104e-006

HW6 P2a: Transfer function (Manual ARX):

model_man =
 

  5.516e-07 z^10 + 0.01165 z^9 - 0.1152 z^8 + 0.04481 z^7 + 0.4392 z^6
                                                                         
              - 0.662 z^5 + 0.1977 z^4 + 0.355 z^3 - 0.4052 z^2 + 0.1341 z
                                                                         
  -----------------------------------------------------------------------
                                                                         
  z^10 - 2.561 z^9 + 2.711 z^8 - 1.341 z^7 - 0.5648 z^6 + 1.817 z^5
                                                                         
              - 1.777 z^4 + 0.851 z^3 - 0.1909 z^2 + 0.05781 z - 1.865e-05
 

Sample time: 0.1 seconds
Discrete-time transfer function.
HW6 P2a: Transfer function (MATLAB ARX):

model_mat =
 

  5.516e-07 z^10 + 0.01165 z^9 - 0.1152 z^8 + 0.04481 z^7 + 0.4392 z^6
                                                                         
              - 0.662 z^5 + 0.1977 z^4 + 0.355 z^3 - 0.4052 z^2 + 0.1341 z
                                                                         
  -----------------------------------------------------------------------
                                                                         
  z^10 - 2.561 z^9 + 2.711 z^8 - 1.341 z^7 - 0.5648 z^6 + 1.817 z^5
                                                                         
              - 1.777 z^4 + 0.851 z^3 - 0.1909 z^2 + 0.05781 z - 1.865e-05
 

Sample time: 0.1 seconds
Discrete-time transfer function.
HW6 P2a: Norm of errors from expected pressure:
 - Manual: 0.134917
 - MATLAB: 0.134917
```

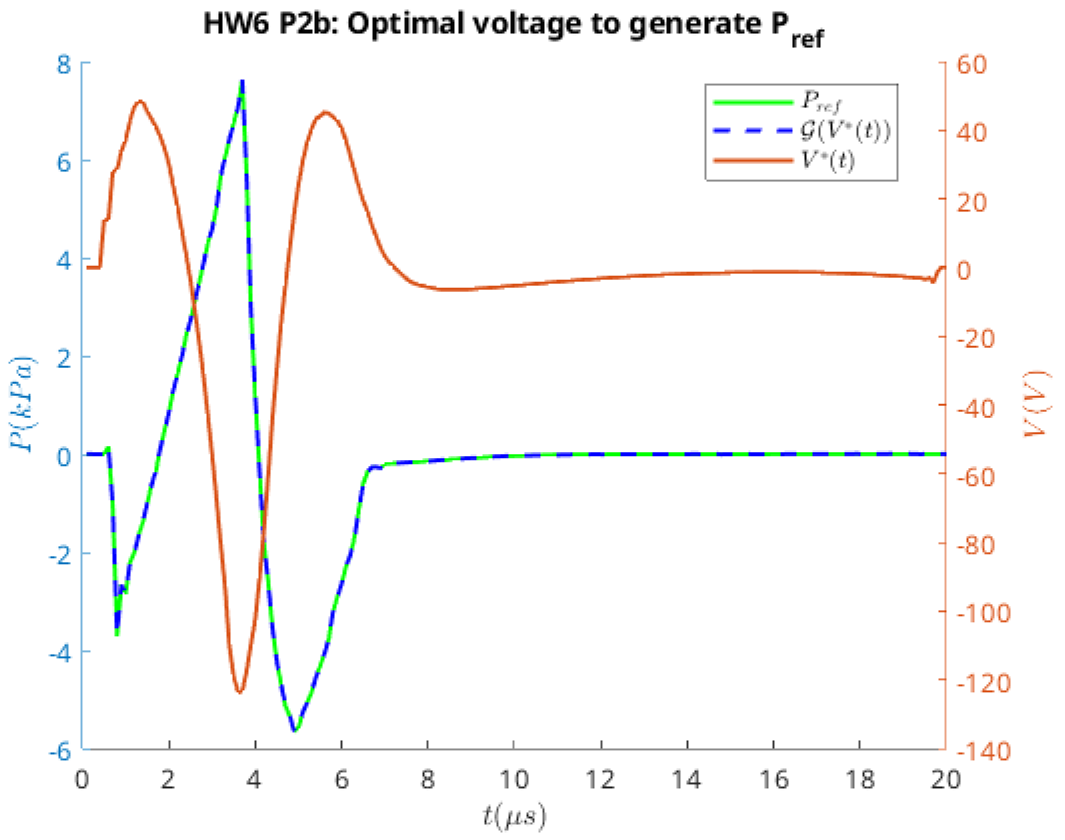

HW6 P2a: Comparison between system models
P
P_{manual}
P_{matlab}
$V_{example}$
$P(kPa)$
$V(V)$
$t(\mu s)$


HW6 P2a: Error comparison between manual and MATLAB ARX
$\times 10^{-7}$
$(P - P_{manual})^2$
$(P - P_{matlab})^2$
Squared error between Pressure
$t(\mu s)$

part b, drive the system to P_ref

```
[num_mat, den_mat]= tfdata(tf(arx(mat_sysid_train, [12, 12, 0]))); %
otherwise voltage is weird
model_mat = tf(fliplr(num_mat), fliplr(den_mat), ts);

% construct inpulse response matrix
ss_model = ss(model_mat);
A = ss_model.A;
B = ss_model.B;
C = ss_model.C;
N_ref = length(P_ref);

% compute last row of G first
last_row = zeros(1, N_ref);
for i=1:N_ref-1
    last_row(i) = C*A^(N_ref-i-1)*B;
end
last_row(end) = 0;

G = zeros(N_ref, N_ref);
for i=1:N_ref
    G(i, 1:i) = last_row(N_ref-i+1:end);
end

% find optimal v
V_opt = pinv(G)*P_ref;
GV_opt = piezo_nozzle(V_opt, N_ref);

% plot
tspan = ts*(1:N_ref);
fig = figure;
yyaxis left;
hold on;
plot(tspan, P_ref, 'g', 'LineWidth', 1.5, 'DisplayName', '$P_{ref}$');
plot(tspan, GV_opt, '--b', 'LineWidth', 1.5, 'DisplayName', '$\mathcal{G}
(V^*(t))$');
ylabel("$P(kPa)$", 'Interpreter', 'latex');
yyaxis right;
plot(tspan, V_opt, 'LineWidth', 1.5, 'DisplayName', '$V^*(t)$');
ylabel("$V(V)$", 'Interpreter', 'latex');
xlabel("$t(\mu s)$", 'Interpreter', 'latex');
legend('Location', 'best', 'Interpreter', 'latex');
title("HW6 P2b: Optimal voltage to generate P_{ref}");
saveas(fig, 'figs/hw6p2b.svg');


function [theta, A, b] = timeSysId(m, n, u, y)
N1 = max(m,n)+1;
N2 = length(u);

% construct b
b = y(N1:N2);
```

```
% construct A
A = zeros(N2-N1+1, m+n);
% row by row
for r=1:size(A,1)
    idx_y = N1 - (1:n) + r-1;
    idx_u = N1 - (1:m) + r;
    A(r,:) = [-y(idx_y)', u(idx_u)'];
end
theta = A\b; % solve for theta
end

function model = getModelFromTheta(m, n, theta, ts)
% look at explanation in the notes
numerator = [theta(n+1:end)', zeros(1, n-m+1)]; % for b's
denominator = [1, theta(1:n)']; % for a's

model = tf(numerator, denominator, ts);
end
```



Published with MATLAB® R2023b