Stochastic Adaptive Control Assignment 2

Kristoffer Erbo Kjær - s203829

April 8, 2025

Introduction

Question 1.1

Given the general form:

$$A(q^{-1})y_t = q^{-k}B(q^{-1})u_t + e_t \tag{1}$$

The system provided is decribed by the following polynomials:

$$\begin{aligned} A &= [1, -2.438, 1.942, -0.502] \\ B &= [1.446, -0.158, -1.093] \\ C &= 1 \end{aligned}$$

The input delay $k = 1$. To implement a MV0 controler an appropriate predetion step m must be chosen in the general Diophantine equation given by Eq. (2).

$$C(q^{-1}) = A(q^{-1})G_m(q^{-1}) + q^{-m}S_m(q^{-1}) \tag{2}$$

$m = k$ is chosen so that the prediction accounts for the delay in input. The equation then becomes.

$$1 = (1 - 2.438q^{-1} + 1.942q^{-2} - 0.502q^{-3})G(q^{-1}) + q^{-m}S(q^{-1}) \tag{3}$$

Solving the equation then gives:

$$\begin{aligned} G &= 1 \\ S &= [2.438, -1.942, 0.502] \end{aligned}$$

For the MV0 implementation $\mathbf{R = conv(B,G)}$ is used, and with $G = 1$ R must equal B. Using the control law for MV0 control the input formula then becomes:

$$B\left(q^{-1}\right)G_k\left(q^{-1}\right)u_t = R\left(q^{-1}\right)u_t = w_t - S_k\left(q^{-1}\right)y_t \tag{4}$$

$$u_t = \frac{1}{1.446}(\omega_t - (2.438q^{-1} - 1.942q^{-2} + 0.502q^{-3})y_t \tag{5}$$

$$- (1.446q^{-1} - 0.158q^{-2} - 1.093q^{-3})u_t) \tag{6}$$

Implimenting this yeald the result seen in Fig. 1. It is clear that the implimented controller does a good job of driving the system to the decired reference. This is done fast and with minimal noice in steady state. However, this comes with the cost of a very aggressive input signal. This is the case for both the magnitude and the quick change in direction. It is therefore likely that the performance seen here would not be realisable in reality since the input would reach its limits.

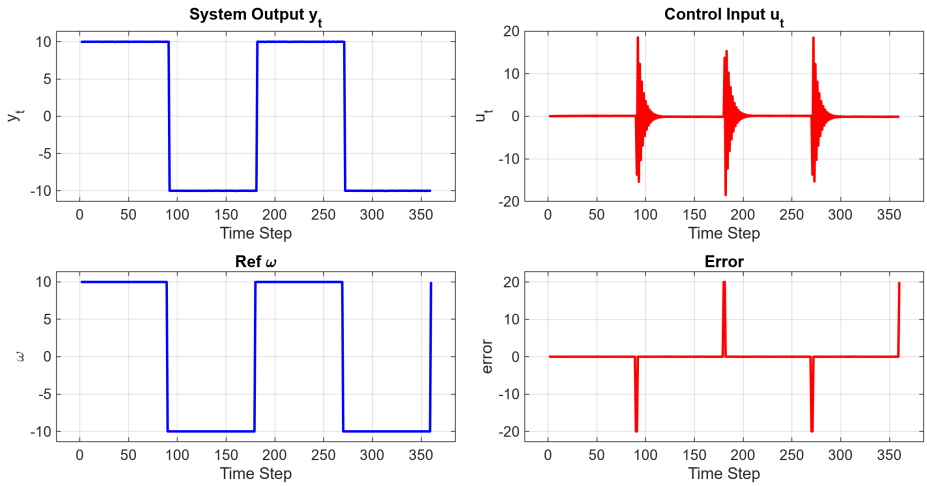

Figure 1: The ARX system with MV0 control implimented in the closed loop.

Question 1.2

The controller structure of Question 1.1 is incorporated together with the ARX recursive least squares (RLS) estimator from assignment 1. While RLS estimation is computed online, the polynomials for the controller of Question 1.1 are computed offline. To implement an adaptive controller, the calculation of the polynomials for the controller must therefore be computed for each iteration with the estimated system parameters being used for the control law.

To simulate a real-world case where a good model is known, parameters close to the exact model are used as an initial guess. The simulation result of this can be seen in Fig. 2. The performance is quite good, but since the system is initialized at 10, it is clear the wrong parameter estimates result in control inputs that drive the system away from the reference. The initial covariance matrix has diagonal values of 0.001. Increasing this can make the deviation from the reference almost unnoticeable, but it is kept as is to show the dynamics of the adaptive controller.

After a short time the system follows the reference as in question 1.1. This makes sense since the final parameter estimates are close to the actual system compared to the initial guess.

The values of the initial guesses and final values can be seen in Eqs. (7) to (10).

$$A_{guess} = [1, -2.3625, 1.9693, -0.4453] \tag{7}$$
$$B_{guess} = [1.4937, -0.1023, -1.0489] \tag{8}$$
$$A_{est_{final}} = [1, -2.4178, 1.9158, -0.49376] \tag{9}$$
$$B_{est_{final}} = [1.4735, -0.096009, -1.0563] \tag{10}$$

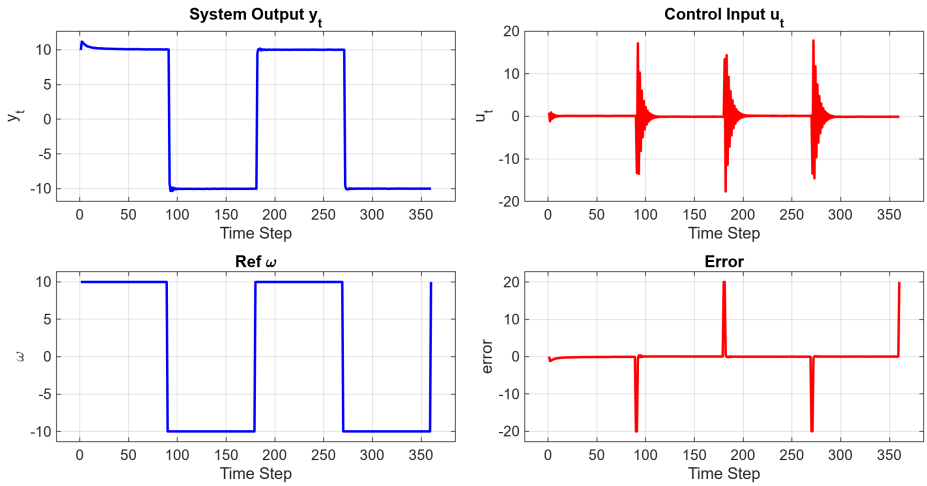

Figure 2: MV0 control with RLS estimation with initial estimates close to their actual values.

To plot the square cost functions for the system subject to any reference input function, it is necessary to account for the steady state variance and the dynamic response to the reference of the closed loop system. The trfvar2.m function supplied is used to find the steady state variance, while the dynamic response is simply calculated from the closed loop polynomials. The cumulative square cost functions of the deviation in the output, input, and error are shown in Fig. 3. With the MV0 controller implemented, it makes sense that the primary increase in the cost of the input and error is around the steps of the input square wave.

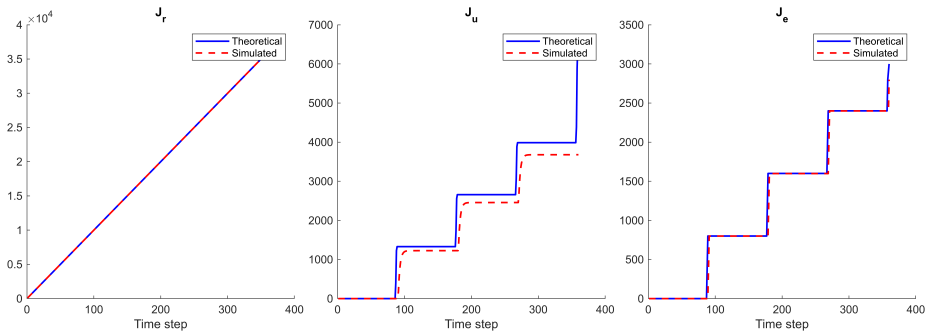

Figure 3: Cumualive cost functions of $J_r = y^2$, $J_u = u^2$ and $J_e = (y - \omega)^2$

Question 1.3

To simulate a case where the system coefficients are completely unknown, all the values are now initialized as ones, which gives a significantly worse transient response, as can be seen in Fig. 4. This is the case even though the covariance matrix is initialized with a diagonal of 10's, which is significantly higher than the more accurate initial guess. One interesting thing is that in the simulation shown in Fig. 4 the reference is followed satisfactorily before 50 simulation steps even though the final estimated parameters are far from the actual parameters. There are two main reasons that this could happen: 1) The approximated system coefficients result in an equivalent input/output response as the original system. 2) The solution to the Diophantine equation for the approximated system is equivalent to that of the original system thereby producing the same controller. In any case since the objective of implementing the adaptive controller is the closed loop response and not parameter estimation this is a satisfactory result.

Longer simulations will also show better approximation of the coefficients indicating that the refrence give a limited pertubation of the system in the given timeframe.

$$A_{guess} = [1, 1, 1, 1] \tag{11}$$
$$B_{guess} = [1, 1, 1] \tag{12}$$
$$A_{est_{final}} = [1, -1.466, 0.47049, 0.024545] \tag{13}$$
$$B_{est_{final}} = [1.4467, 1.2495, 0.055831] \tag{14}$$

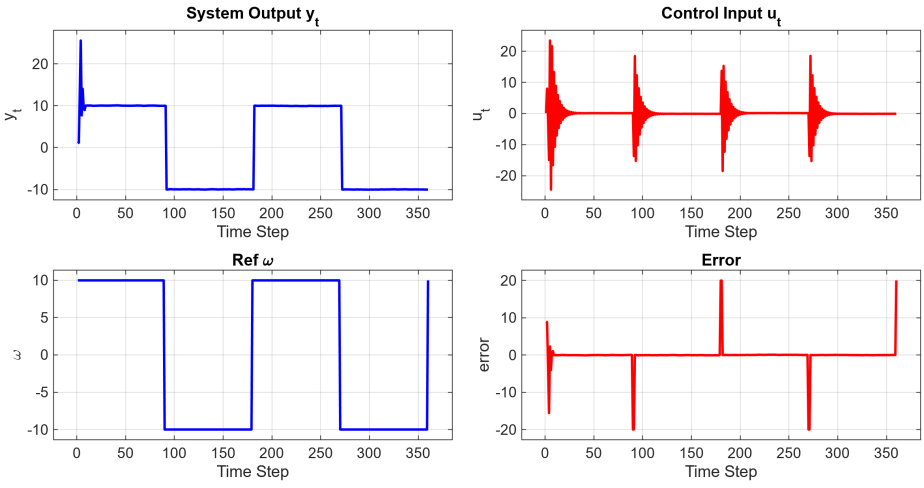

Figure 4: MV0 control with RLS estimation with initial estimates of 1 for all values.

Question 2.1

The coefficients of the C polynomial are as follows from Eq. (15).

$$C = [1, -1.441, 0.296, 0.167] \tag{15}$$

Question 2.2

The ARMAX routine is implemented as a simple iterative process, as seen in Appendix A, the result of which is shown in Fig. 5. The output is hovering around 0, which is to be expected with a PRBS input.

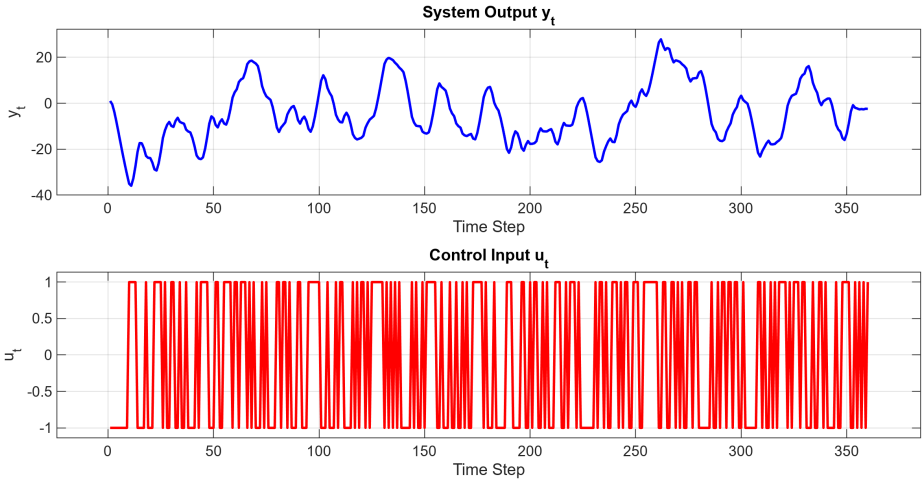

Figure 5: ARMAX simulation with PRBS input sequence.

Question 2.3

The diophantine equation is solved with the same function as in assignment 1 and shown in Appendix B. The difference from MV0 to GMV is the equation that needs to be solved. In the general case, the A polynomial is convolved with the Ay polynomial of the H_y filter and the C polynomial is convolved with the By polynomial of the H_y filter. The solution G and S is then found by the following function call of the custom Matlab function:

```
1    [G, S] = diophantine(conv(Ay, A), conv(By, C), m);
```

Where the prediction horizon $m = k = 1$. For the controller later to be implemented $A_y y = B_y = 1$ and the soultion becomes:

$$G = 1$$
$$S = [2.438, -1.942, 0.502]$$

Question 2.4

To determine the requested polynomials, the general GMV control law Eq. (16) is used.

$$[A_u BG + \alpha CB_u]\, u_t = A_u \left[C\frac{B_w}{A_w} w_t - \frac{S}{A_y} y_t \right], \quad \alpha = \frac{\rho}{b_0} \tag{16}$$

Where all polynomials are functions of q^-1. To avoid dividing by polynomials the following rearrangement is made.

$$A_w A_y\, [A_u BG + \alpha CB_u]\, u_t = A_y A_u CB_w\, w_t - A_w A_u S\, y_t, \quad \alpha = \frac{\rho}{b_0} \tag{17}$$

The ploynomials can the be itetifyed as:

$$R(q^{-1}) = A_w(q^{-1})A_y(q^{-1})\left[A_u(q^{-1})B(q^{-1})G(q^{-1}) + \alpha\, C(q^{-1})B_u(q^{-1})\right]$$
$$Q(q^{-1}) = A_y(q^{-1})A_u(q^{-1})C(q^{-1})B_w(q^{-1})$$
$$S(q^{-1}) = A_w(q^{-1})A_u(q^{-1})S(q^{-1})$$

Where the Matlab implementation for Q and S is simply convolutions. The sum in R is handled by first zero padding, summing and then convolving the remaining variables. This as well as the full Matlab routine can be seen in Appendix C.

The routing is testet with a square wave and a MV_{1a} controller. Here all filters are set to 1 except for $B_u = 1 + Q^{-1}$. This compares the current value with the previous one and is weighted with the value $\alpha = \rho/b(0)$. where $\rho = 7500$. This high value effectively means that aggressive changes in input are discouraged, ensuring a smooth change in both the input and the output temperature. As can be seen in Fig. 6 this implementation leads to an asymptotically converging towards the setpoint of the reference. The most important improvement over the MV0 controller is, however, that the input is changing at a steady pace in one direction, which in many cases would lead to less strain on the actuator. The setpoint is reached slower but since the system is already slow in nature it is an acceptable tradeoff.

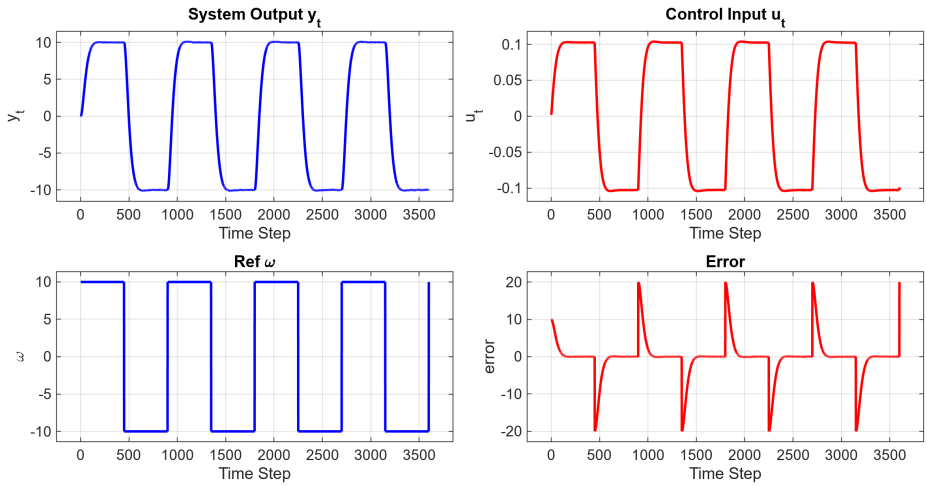

Figure 6: Enter Caption

Question 2.5

ϕ contains the current or previous values of the input, output, and the error as shown:

$$\phi = [y_{t-1}, y_{t-2}, y_{t-3}, u_{t-1}, u_{t-2}, u_{t-3}, e_{t-1}, e_{t-2}, e_{t-3}]^T \tag{18}$$

Theta contains the coefficients describing how previous values affect y_t.

$$\theta = [-2.438, 1.942, -0.502, 1.446, -0.158, -1.093, -1.441, 0.296, 0.167]^T \tag{19}$$

The system can then be described as:

$$y_t = \phi^T \theta \tag{20}$$

Question 2.6

A recursive extended least squares (RELS) estimator is implemented with Fortescue's Method. This method utilizes a time-varying forgetting factor based on the previous prediction error.

The general form of Fortescue's Method is shown in Eqs. (21) to (26).

$$\lambda_t = 1 - \frac{1}{N_0} \frac{\epsilon_t^2}{\sigma^2 s_t} \tag{21}$$

$$\epsilon_t = y_t - \phi_t^T \hat{\theta}_{t-1} \tag{22}$$

$$s_t = 1 + \phi_t^T P_{t-1} \phi_t \tag{23}$$

$$K_t = \frac{P_{t-1}\phi_t}{\lambda_t + s_t} \tag{24}$$

$$\hat{\theta}_t = \hat{\theta}_{t-1} + K_t\epsilon_t \tag{25}$$

$$P_t = (I - K_t\phi_t^T)P_{t-1}\frac{1}{\lambda_t} \tag{26}$$

To generalize a bit more, the varianze is estimated by Eq. (27) and the forgetting factor is the updated as shown in Eq. (28).

$$r_t = r_{t-1} + \frac{1}{t}\left(\frac{\epsilon_t^2}{s_t} - r_{t-1}\right), \quad r_0 = \epsilon_0^2 \tag{27}$$

$$\lambda_t = 1 - \frac{1}{N_0}\frac{\epsilon_t^2}{r_t s_t} \tag{28}$$

The implementation of the routine can be found in Appendix D. The main tuning parameter here is the window size, N_0, of the forgetting factor and initial values of the covariance matrix P. A window size of $N_0 = 36$ and variance matrix $P = 10 * eye(na + (nb + 1) + nc)$ show good results. With an initial guess of 1 for all parameters, the parameters can be estimated to the following values:

$$A_{est_{final}} = [1.0000, -2.4193, 1.9137, -0.4920] \tag{29}$$
$$B_{est_{final}} = [1.4463, -0.1303, -1.0718] \tag{30}$$
$$C_{est_{final}} = [1.0000, -1.0837, 0.1863, 0.1274] \tag{31}$$

The values for the A and B polynomial are very close to the actual parameters. The C coefficients are a bit further away, but considering that the actual error values are unknown and an approximation is used in the parameter estimation, this is not unreasonable.

Question 2.7

The RELS estimator and the MV1a controller are now combined into an adaptive controller which can be found in Appendix E. The ρ values of the controller are lowered to 1500 to better perturb the system while still ensuring an acceptable system response. With a good initial parameter guess close to the actual coefficients yields both a good control performance and parameter estimation. The system behavior can be seen in Fig. 7.

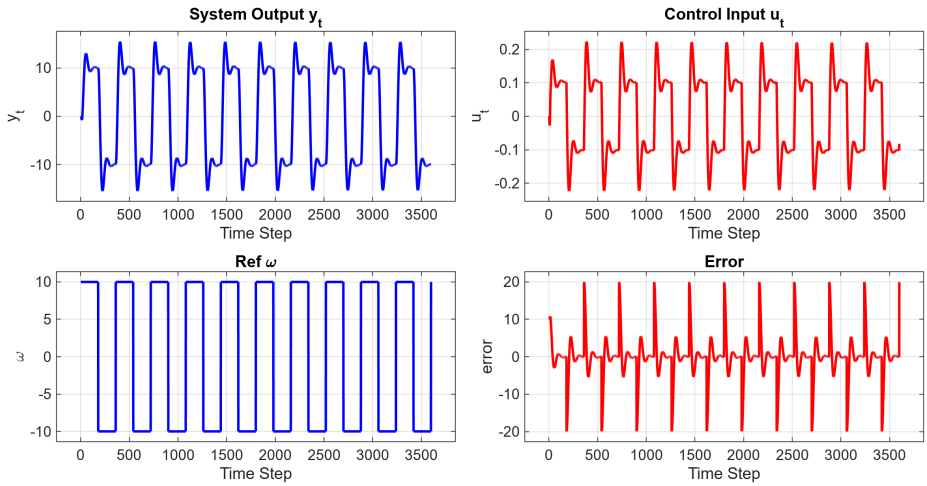

Figure 7: Closed loop implimentation of an adaptive controller with RELS estomation and MV1a control.

Since the parameters are now to be close to the initial value, a smaller initial P = 1 and a larger window size of the simulation length are used. This ensures no drastic changes to the parameters at the start and that changes are made based on all the simulated values. This results in the followin parameters:

$$A_{est_{final}} = [1.0000, -2.3894, 1.9099, -0.5151] \tag{32}$$
$$B_{est_{final}} = [1.5142, -0.0531, -0.9359] \tag{33}$$
$$C_{est_{final}} = [1.0000, -0.8076, 0.4295, -0.0843] \tag{34}$$

Here C is again the polynomial with the values furthest from the actual coefficients as again due to the fact that the actual error is unknown.

Question 2.8

The system is now initialized with estimates of ones instead. For the same simulation length and a P matrix with 1000 in the diagonal, the parameters become:

$$A_{est_{final}} = [1.0000, -1.4496, 0.5930, -0.1081] \tag{35}$$
$$B_{est_{final}} = [1.4099, 1.4522, 0.5607] \tag{36}$$
$$C_{est_{final}} = [1.0000, -0.1716, 0.0807, 0.0977] \tag{37}$$

This is significantly worse parameter estimation, but the closed-loop performance seen in Fig. 8 is similar to that of the good initialization. The estimated system must therefore either be close to the actual system's input/output behaviour or result in a similar controller. Either way, the performance of the

adaptive controller is satisfactory even with an initialization provided the initial guess of covariance is sufficiently high.

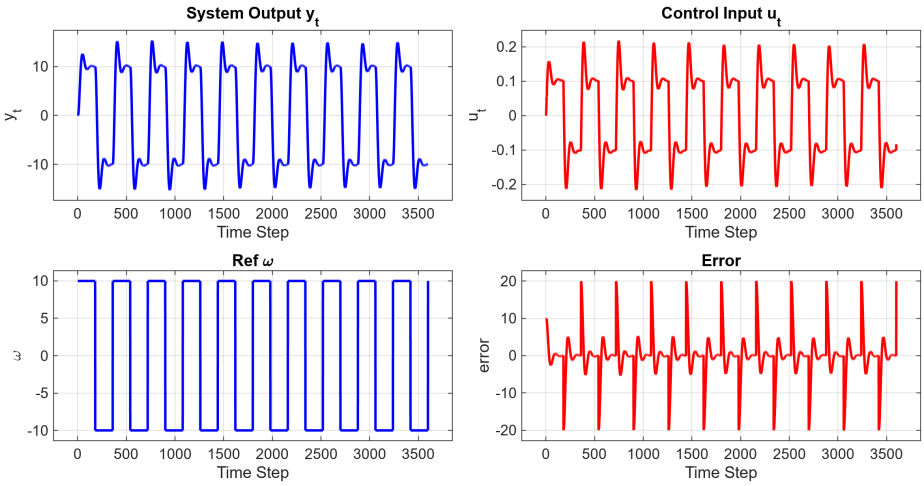

Figure 8: Closed loop implimentation of an adaptive controller with RELS estomation and MV1a control.

Increasing the simulation time 100 times does, however, improve the estimation of especially the A polynomial. This indicates that the algorithm performs as intended and that further tuning potentially could improve the estimation time. In practice, this estimation chould be done ones and the parameres could then be used to initialise the system. This migth therfore be unnessesary.

$$A_{est_{final}} = [1.0000, -2.4262, 1.9244, -0.4959] \tag{38}$$

$$B_{est_{final}} = [1.3430, 0.0738, -1.1921] \tag{39}$$

$$C_{est_{final}} = [1.0000, -1.3006, 0.1531, 0.1569] \tag{40}$$

Appendix

A Question 2.2

```
function [y, u] = ARMAX(A, B, C, k, e, N, y_init, u_init, u_in )

    e = [zeros(numel(y_init),1) ; e];

    % Get polynomial orders
    na = numel(A) - 1;
    nb = numel(B) - 1;
    nc = numel(C) - 1;
    simStart = max([na, nb, nc]) + k;

    % Initialize output and control input vectors
    y = [y_init; zeros(N,1)];
    u = [u_init; u_in];
    e = [zeros(numel(y_init),1) ; e];

    % Iterative MV1a control computation
    for t = simStart: N + max(numel(y_init),numel(u_init))
        Y_t = y(t - (1:na));
        U_t = u(t - k - (0:nb));
        E_t = e(t - (0:nc));

        % Compute y using past values (and include u in the equation)

        y(t) = - A(2:end) * Y_t + B * U_t + C * E_t;

    end

    y = y(numel(y_init)+1:end);
    u = u(numel(u_init)+1:end);
end
```

B Question 2.3

```
function [G, S] = diophantine(A,C,m)
    G = [];

    % Pad B with zeros to make S as long as A
    S = [C, zeros(1,length(A)-length(C))];

    for i = 1:m
```

```
        % Augment with first element of S
        G = [G, S(1)];

        % Update S
        S = [S(2:end) - S(1) *A(2:end), 0] ;
    end

    % Remove last element
    S = S ( 1 : end-1);
end
```

C Question 2.4

```
function [y, u] = GMV(A, B, C, k, omega, e, N,  Ay, By, Au, Bu, Aw, Bw, rho, y_init, u_init


    % Solve the Diophantine equation for G and S
    [G, S_diop] = diophantine(conv(Ay, A), conv(By, C), k);
    disp("G:")
    disp(G)

    disp("S:")
    disp(S_diop)


    % [G, S] = diophantine(A, 1, k);
    b0 = B(1);
    alpha = rho / b0;
    R1 = conv(Au,conv(B,G));
    R2 = alpha*conv(C,Bu);


    diff = numel(R1) - numel(R2);
    if diff > 0
        R2 = [R2, zeros(1,abs(diff))];
    elseif diff < 0
        R1 = [R1, zeros(1,abs(diff))];
    end

    R = conv(Aw,conv(Ay,R1 + R2));
    % R1_conv = conv(conv(Ay,R1),Aw);
    % R2_conv = conv(conv(Ay,R2),Aw);
    % R = R1_conv + R2_conv;

```

```
    Q = conv(conv(conv(Au,Ay),Bw),C);
    S = conv(Au,conv(Aw,S_diop));


    % Ensure input signals have length N
    omega = [y_init; omega];
    e = [zeros(numel(y_init),1) ; e];

    % Get polynomial orders
    na = numel(A) - 1;
    nb = numel(B) - 1;
    nc = numel(C) - 1;
    nr = numel(R) - 1;
    ng = numel(G) - 1;
    ns = numel(S_diop) - 1;
    nw_in = numel(Q) - 1;
    nS_in = numel(S) - 1;
    simStart = max([na, nb, nc, nr, ns, ng, ns,nw_in,nS_in]) + k;

    % Initialize output and control input vectors
    y = [y_init; zeros(N,1)];
    u = [u_init; zeros(N,1)];

    % Iterative MV1a control computation
    for t = simStart: N + max(numel(y_init),numel(u_init))
        Y_t = y(t - (1:na));
        U_t = u(t - k - (0:nb));
        E_t = e(t - (0:nc));

        % Compute y using past values (and include u in the equation)
        y(t) = - A(2:end) * Y_t + B * U_t + C * E_t;

        Y_s = y(t - (0:nS_in));
        U_r = u(t - (1:nr));
        W_t = omega(t - (0:nw_in));
        % W_t = omega(t);

        % Compute control input u using MV1a law % H_u(q) = 1 - q^-1
        u(t) = 1/(R(1)) * (Q * W_t - S * Y_s - R(2:end) * U_r);

        % Store output
    end
    y = y(numel(y_init)+1:end);
    u = u(numel(u_init)+1:end);
end
```

D Question 2.6

```
function [y, theta_rls] = armax_parameter_estimation(A, B, C, u, e, N, y_init, theta_in


% Get polynomial orders
na = numel(A) - 1;
nb = numel(B) - 1;
nc = numel(C) - 1;

simStart = max([na, nb + k, nc]) + 1;

% Initialize output vector
y = [y_init; zeros(N, 1)];

 % Ensure input signals have length N
u = [zeros(numel(y_init) + k, 1); u];
e = [zeros(numel(y_init), 1); e];
E_t_est = zeros(numel(e),1);

% Initialize Recursive Least Squares (RLS) parameters
num_params = na + (nb + 1) + nc;
theta_rls = zeros(num_params, N); % Store estimated parameters over time
theta_hat = theta_init; % Initial parameter estimates
P = P_init; % Initial covariance (high uncertainty)
r = e(1)^2 / 6; % Initial variance estimate

% Iterative parameter estimation
for t = simStart:N + numel(y_init)

    % Construct regressor vector
    Y_t = y(t - (1:na));
    U_t = u(t - k - (0:nb));
    E_t = e(t - (0:nc));

    % Output computation from current parameters (simulation model)
    y(t) = - A(2:end) * Y_t + B * U_t + C * E_t;

    Phi_t = [-Y_t', U_t', E_t_est(t - (1:nc))']';  % Regressor vector

    % Prediction error
    e_t = y(t) - Phi_t' * theta_hat;

    E_t_est(t) = e_t;

```

```
44
45
46          % Update forgetting factor based on prediction error
47          s_t = 1 + Phi_t' * P * Phi_t;
48          r = r + (1 / t) * ((e_t^2 / s_t) - r);
49          lambda = 1 - (1 / N0) * (e_t^2 / (r * s_t));
50
51          % Kalman gain
52          K_t = P * Phi_t / (lambda + s_t);
53
54          % Parameter update
55          theta_hat = theta_hat + K_t * e_t;
56
57          % Covariance update
58          P = (P - K_t * Phi_t' * P) / lambda;
59
60          % Store estimates over time
61          theta_rls(:, t) = theta_hat;
62
63
64
65
66
67
68      end
69
70      % Remove initial conditions from output
71      y = y(numel(y_init) + 1:end);
72      theta_rls = theta_rls(:, numel(y_init) + 1:end);
73  end
74
75
```

E Question 2.7 - 2.8

```
1  function [y, u, theta_rls,P_rls] = MVa1_RLS(A, B, C, k, omega, e, N, theta_init, P_init, Ay
2
3      % Solve the Diophantine equation for G and S
4      [G, S] = diophantine(conv(Ay, A), conv(By, C), k);
5
6      b0 = B(1);
7      alpha = rho / b0;
8      R1 = conv(Au, conv(B, G));
9      R2 = alpha * conv(C, Bu);
```

```
10
11      diff = numel(R1) - numel(R2);
12      if diff > 0
13          R2 = [R2, zeros(1, abs(diff))];
14      elseif diff < 0
15          R1 = [R1, zeros(1, abs(diff))];
16      end
17
18      R = conv(Aw, conv(Ay, R1 + R2));
19      W_in = conv(conv(conv(Au, Ay), Bw), C);
20      Y_in = conv(Au, conv(Aw, S));
21
22      % Ensure input signals have length N
23      omega = [y_init; omega];
24      e = [zeros(numel(y_init), 1); e];
25      E_t_est = zeros(numel(e),1);
26
27      % Get polynomial orders
28      na = numel(A) - 1;
29      nb = numel(B) - 1;
30      nc = numel(C) - 1;
31      nr = numel(R) - 1;
32      ng = numel(G) - 1;
33      ns = numel(S) - 1;
34      nw_in = numel(W_in) - 1;
35      ny_in = numel(Y_in) - 1;
36      simStart = max([na, nb, nc, nr, ns, ng, ns, nw_in, ny_in]) + k;
37
38      % Initialize output and control input vectors
39      y = [y_init; zeros(N, 1)];
40      u = [u_init; zeros(N, 1)];
41
42      % Initialize Recursive Least Squares (RLS) parameters
43      num_params = na + (nb + 1) + nc;
44      theta_rls = zeros(num_params, N); % Store estimated parameters over time
45      theta_hat = theta_init; % Initial parameter estimates
46      P = P_init; % Initial covariance (high uncertainty)
47      P_rls = zeros(numel(diag(P)), N);
48
49      r = e(1)^2 / 6; % Initial variance estimate
50
51      % Iterative MV1a control computation
52      for t = simStart:N + max(numel(y_init), numel(u_init))
53
54          Y_t = y(t - (1:na));
55          U_t = u(t - k - (0:nb));
```

```
        E_t = e(t - (0:nc));

        % Compute y using past values (and include u in the equation)
        y(t) = -A(2:end) * Y_t + B * U_t + C * E_t;

        % Construct regressor vector
        Phi_t = [-Y_t', U_t', E_t_est(t - (1:nc))'];  % Regressor vector

        % Prediction error
        e_t = y(t) - Phi_t' * theta_hat;

        E_t_est(t) = e_t;

        % Update forgetting factor based on prediction error
        s_t = 1 + Phi_t' * P * Phi_t;
        r = r + (1 / t) * ((e_t^2 / s_t) - r);
        lambda = 1 - (1 / N0) * (e_t^2 / (r * s_t));

        % Kalman gain
        K_t = P * Phi_t / (lambda + s_t);

        % Parameter update
        theta_hat = theta_hat + K_t * e_t;

        % Covariance update
        P = (P - K_t * Phi_t' * P) / lambda;

        % if not(mod(t,T))
        %     P = P_init;
        % end

        % Store estimates over time
        theta_rls(:, t) = theta_hat;
        P_rls(:,t) = diag(P);

        A_est = [1, theta_hat(1:na)'];
        B_est = theta_hat(na + 1:na + 1 + nb)';
        C_est = [1, theta_hat(na + 1 + nb + 1:end)'];

        % Solve the Diophantine equation for G and S
        [G, S] = diophantine(conv(Ay, A_est), conv(By, C_est), k);

        b0 = B_est(1);
        alpha = rho / b0;
        R1 = conv(Au, conv(B_est, G));
        R2 = alpha * conv(C_est, Bu);
```

```
102
103         diff = numel(R1) - numel(R2);
104         if diff > 0
105             R2 = [R2, zeros(1, abs(diff))];
106         elseif diff < 0
107             R1 = [R1, zeros(1, abs(diff))];
108         end
109         R = conv(Aw, conv(Ay, R1 + R2));
110         W_in = conv(conv(conv(Au, Ay), Bw), C_est);
111         Y_in = conv(Au, conv(Aw, S));
112
113         Y_s = y(t - (0:ny_in));
114         U_r = u(t - (1:nr));
115         W_t = omega(t - (0:nw_in));
116
117         % Compute control input u using MV1a law % H_u(q) = 1 - q^-1
118         u(t) = 1 / R(1) * (W_in * W_t - Y_in * Y_s - R(2:end) * U_r);
119     end
120
121     y = y(numel(y_init) + 1:end);
122     u = u(numel(u_init) + 1:end);
123 end
124
125
126
```

F Main

The main script is proviten in the .zip folder with the assignment and is named: "assignment_1.mlx".