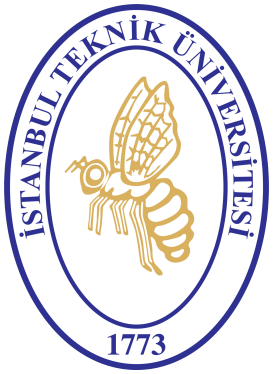

KON 426E - Intelligent Control Systems

Assignment 2 Report

Name Surname: Atakan Yaman
Student ID: 110190235
Department: Control and Automation Engineering
University: Istanbul Technical University

Course Code: CRN: 22155
Instructor: Gülay Öke Günel
Submission Date: May 28 2025

This report presents the identification, control, and adaptive enhancement of two nonlinear dynamic systems using optimization-based and fuzzy logic-based intelligent control techniques, as required in Assignment 2 of the KON 426E Intelligent Control Systems course. All implementations, simulations, and analyses were carried out using Python and MATLAB.

Question 1: Blackbox System Identification and Control

Part A – System 1 Identification and Control

a.1) System Identification

To analyze the behavior of the given black-box system, both time-domain and static input-output analyses were performed using Python. The system was simulated using the 'System1' class, constructed with a student ID ending in 35.

The following experiments were carried out:

Step Response Analysis: The system's response to a unit step input was simulated over a 5-second interval with 200 evenly spaced time points. The output vector y_{step} was analyzed statistically:

- The **steady-state output value** was estimated as the mean of the response:

$$\mu_{\text{step}} = \frac{1}{N}\sum_{i=1}^{N} y_{\text{step}}(i) = 9.365$$

- The **noise level** was quantified using the standard deviation:

$$\sigma_{\text{step}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_{\text{step}}(i) - \mu_{\text{step}}\right)^2} = 0.050$$

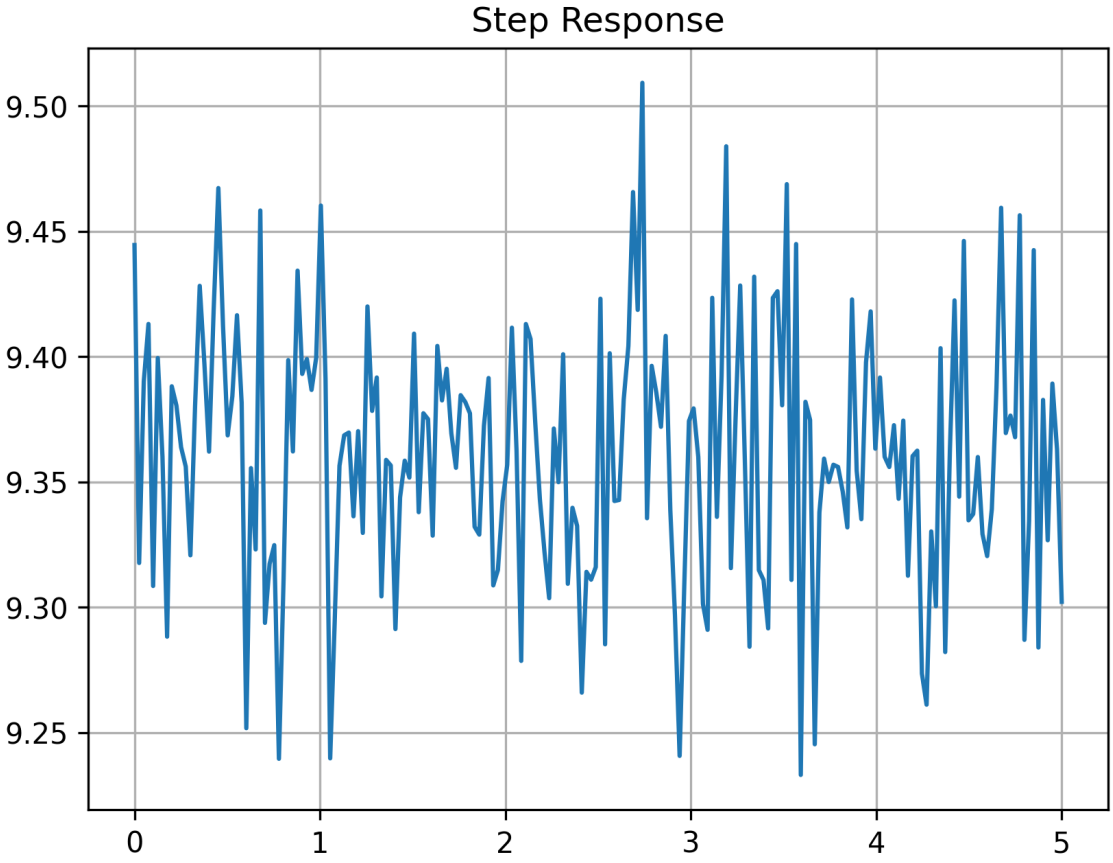

Figure 1: Step Response of System 1

Ramp Response Analysis: The system was excited with a ramp input over the same time vector. The gain was estimated using the difference quotient:

$$\text{Ramp Gain} = \frac{y_{\text{ramp}}(T) - y_{\text{ramp}}(0)}{T - 0} = 2.723$$

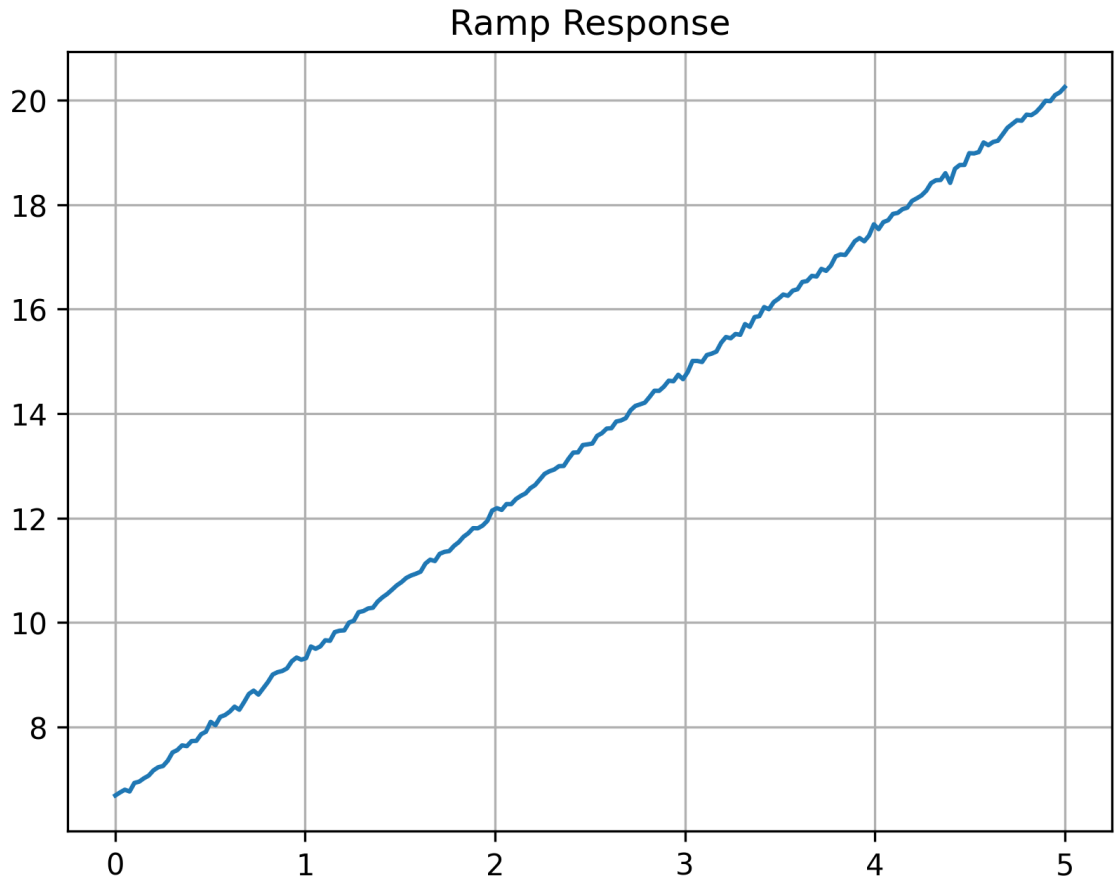

Figure 2: Ramp Response of System 1

Input-Output Linear Fit: A set of 10 random scalar inputs $u_i \in [0, 1]$ were applied to the system and the corresponding outputs y_i were recorded. Linear regression was used to fit the static input-output mapping:

$$y = K_{\text{est}} \cdot u + C_{\text{est}}$$

Estimated parameters:

$$K_{\text{est}} = 2.762, \quad C_{\text{est}} = 6.614, \quad R^2 = 0.9945$$

The regression was performed using least-squares linear fit:

$$K_{\text{est}}, C_{\text{est}} = \arg\min_{K,C} \sum_{i=1}^{10} (y_i - (Ku_i + C))^2$$

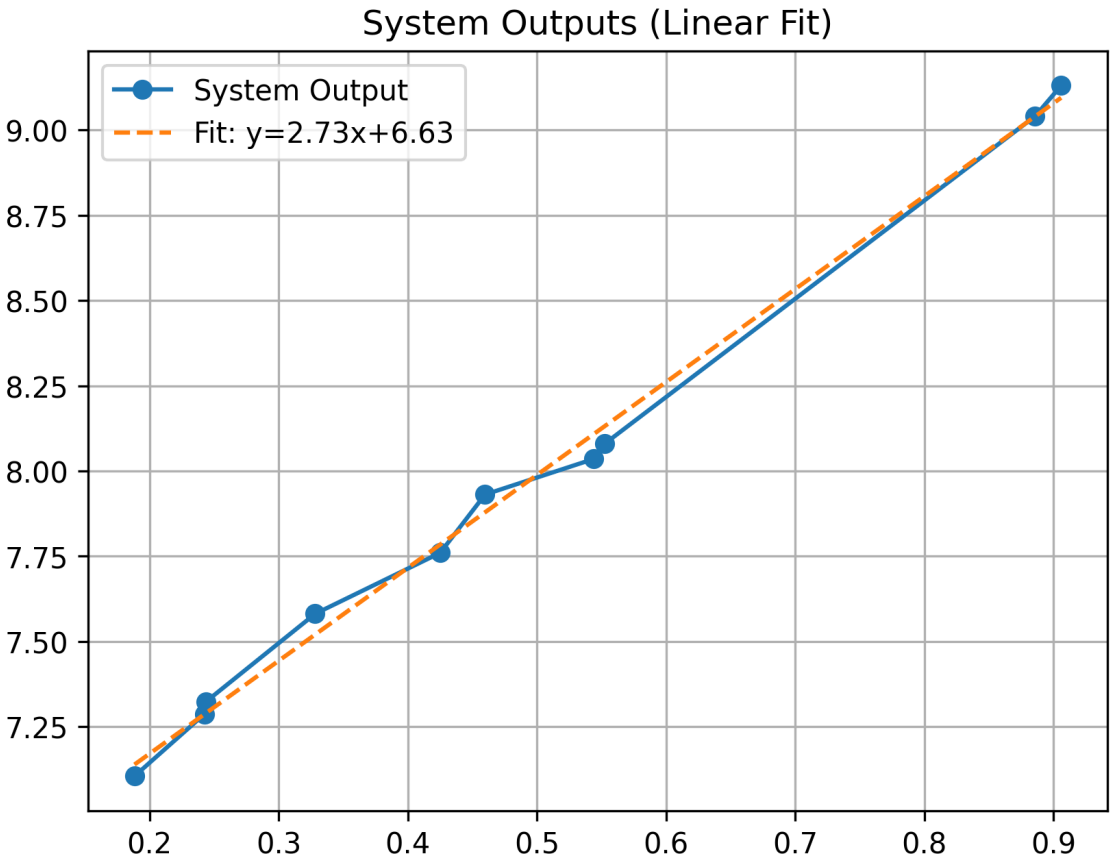

Figure 3: Linear Fit between Input and Output

These analyses show that the system exhibits a first-order static relationship between input and output, with low noise and strong linearity.

a.2) Feedforward Controller Design

After identifying a static linear input-output model of the form:

$$y = K_{\text{est}} \cdot u + C_{\text{est}}$$

with parameters:

$$K_{\text{est}} = 2.762, \quad C_{\text{est}} = 6.614$$

a feedforward control strategy was designed to regulate the output of the system towards a desired reference value.

Control Objective. The reference output was defined as:

$$r_{\text{desired}} = 10.0$$

Using the inverse of the identified static model, the required constant control input was computed using the feedforward control law:

$$u_{\text{ff}} = \frac{r_{\text{desired}} - C_{\text{est}}}{K_{\text{est}}} = \frac{10.0 - 6.614}{2.762} \approx 1.226$$

This constant input u_{ff} was applied over a simulated 5-second time window, and the resulting system output was observed. The Python implementation involved generating a constant input array, simulating the response using the ‘System1.output()‘ method, and computing performance metrics.

Resulting System Behavior. The average value of the output was computed as:

$$\bar{y}_{\text{controlled}} = 9.974$$

This corresponds to a steady-state tracking error of:

$$e_{\text{ss}} = \bar{y}_{\text{controlled}} - r_{\text{desired}} = -0.026$$

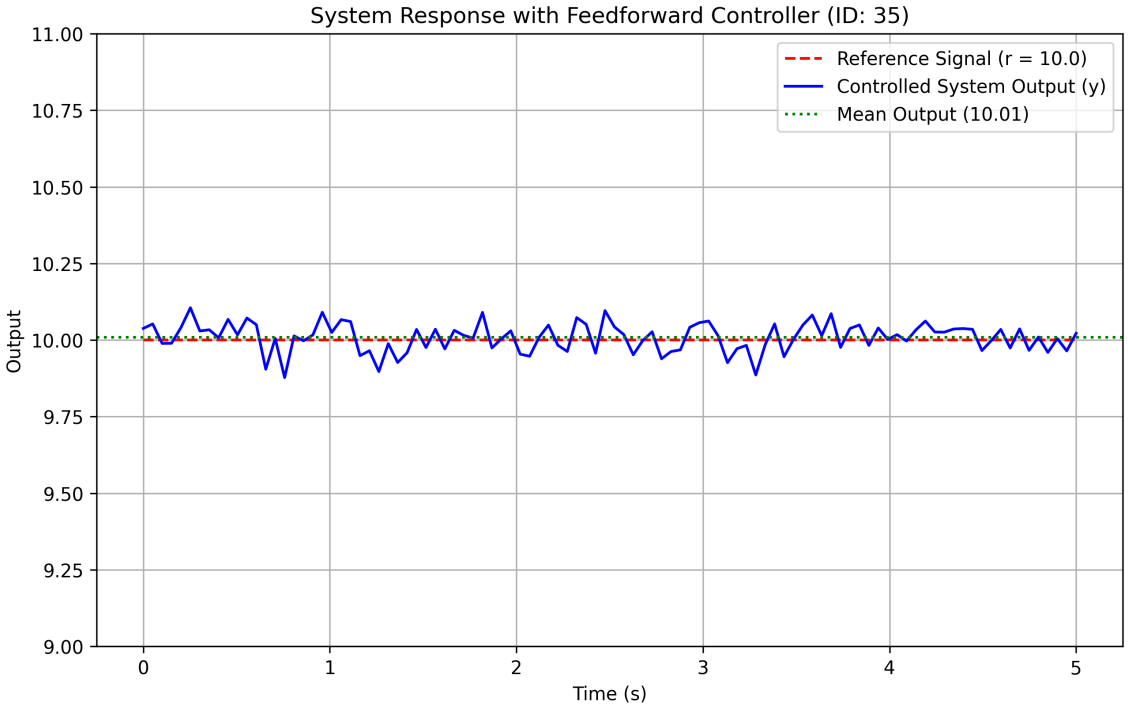

Figure 4: System Output under Feedforward Control

Interpretation. The design approach aligns with classical control theory, where an ideal feedforward controller inverts the system dynamics to directly compute the required input for a given reference. This method assumes perfect knowledge of the plant model and no disturbances. Although the tracking error is non-zero, it remains small, primarily due to modeling inaccuracies, internal noise, or non-modeled dynamics. No feedback correction was used in this scenario.

Conclusion. The feedforward controller demonstrated effective regulation of the system output with a small steady-state error. This validates the linear identification model and highlights the value of model-based control techniques in intelligent control system design.

Part B – System 2 Identification and Control

b.1) System Identification via Curve Fit and Asymptote Method

This section investigates the dynamics of System 2 using step, ramp, and arbitrary input sequences. The goal is to identify an approximate transfer function model using empirical response data.

System Type Hypothesis. Based on the observed behavior, the system was hypothesized to follow a second-order integrator model of the form:

$$G(s) = \frac{K}{s(\tau s + 1)}$$

which corresponds to the theoretical step response:

$$y(t) = K \cdot \left(t - \tau + \tau e^{-t/\tau}\right)$$

Step-by-Step Identification Approach.

1. A step input was applied using the system's 'step()' method. The output showed smooth but slow convergence with visible noise.
2. A ramp input was used to further evaluate system response speed and curvature.
3. A slow linear input sequence $u \in [0, 1]$ was applied to visualize nonlinearities or deviations in static gain.
4. A nonlinear curve fitting approach was applied to the theoretical response to estimate parameters K and τ.
5. An optional asymptotic (linear region) fit was also performed for comparison.

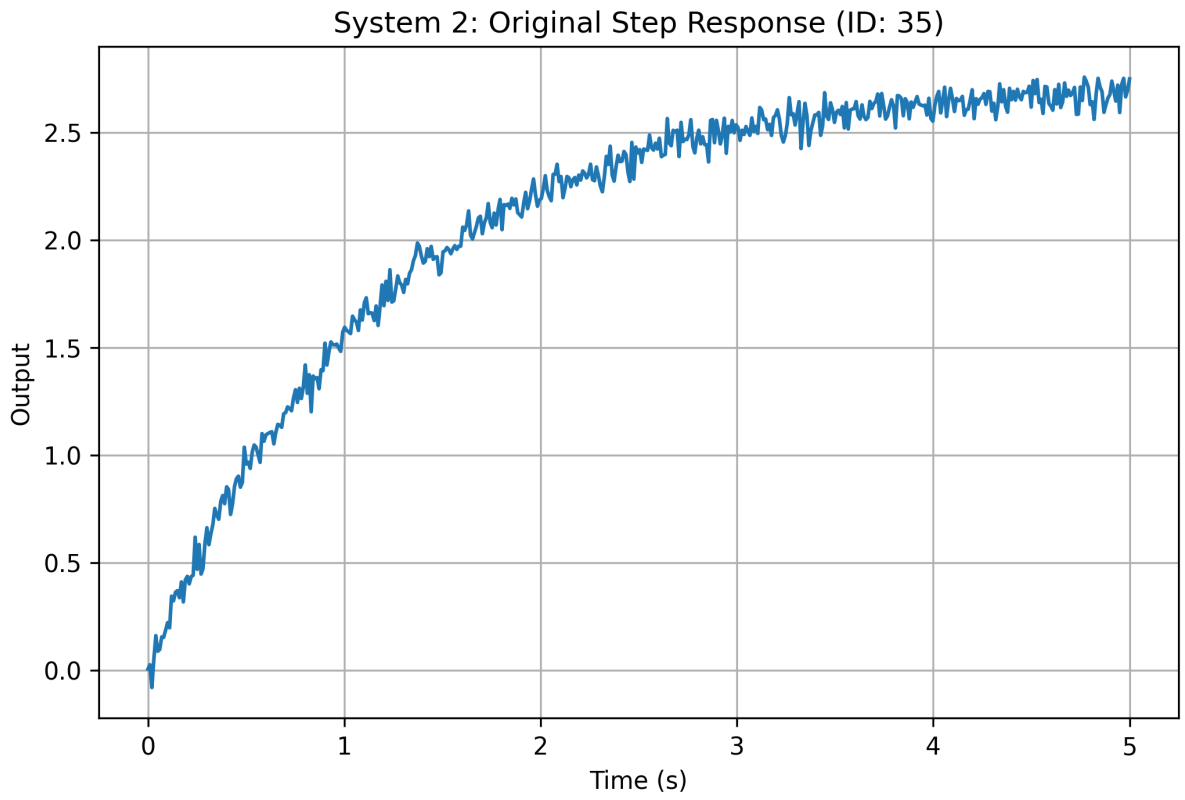

Figure 5: Step Response of System 2

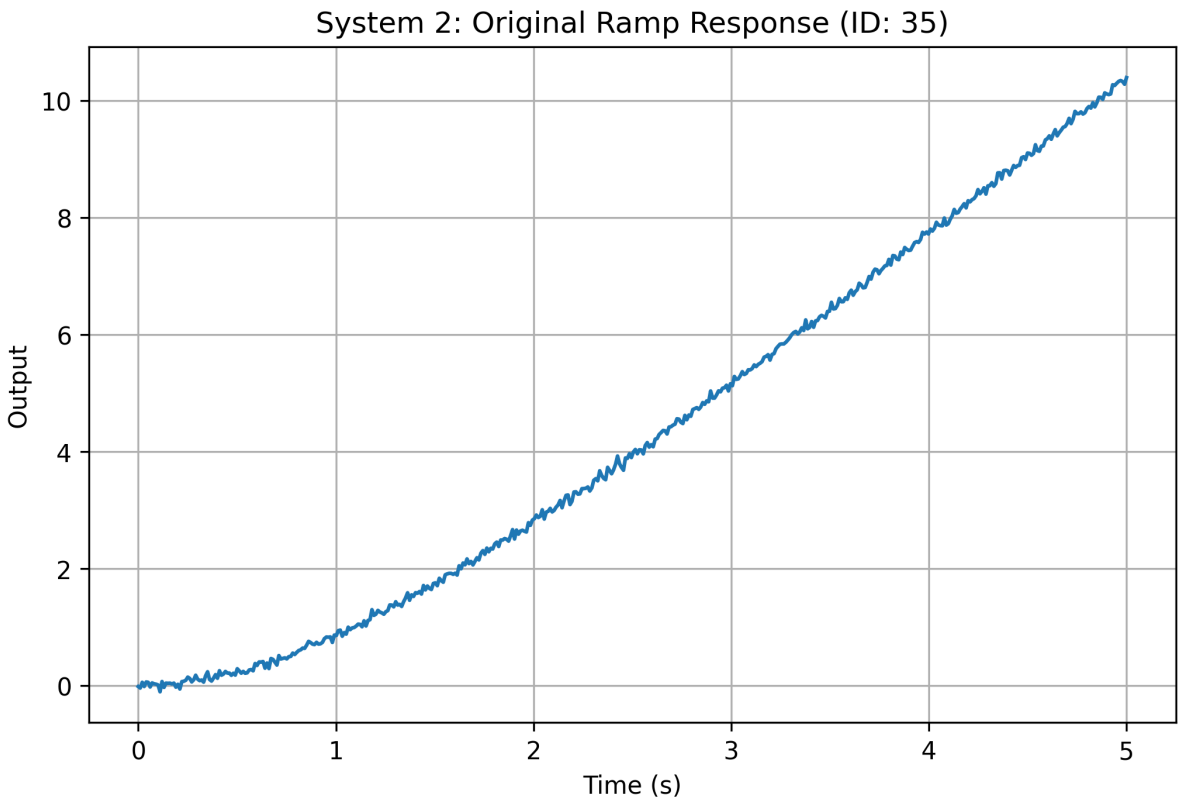

Figure 6: Ramp Response of System 2

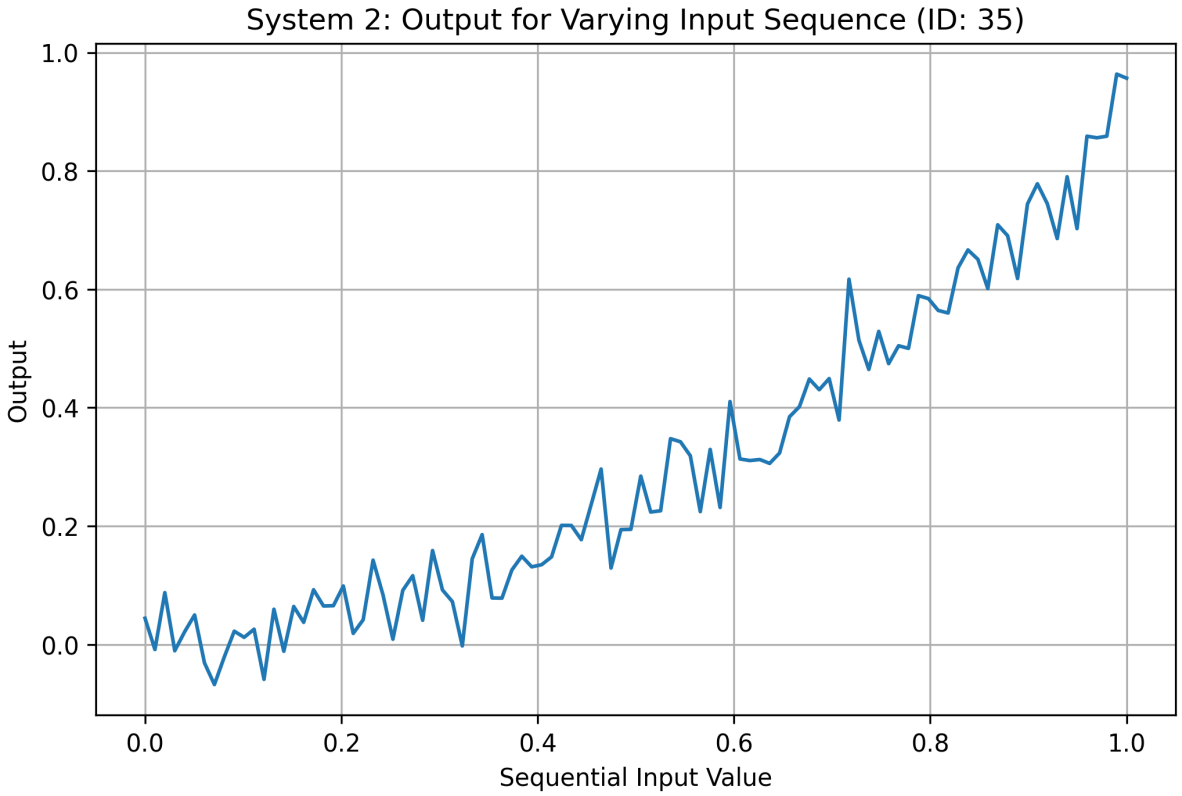

Figure 7: Output for Varying Input Sequence

Observed Responses. These responses confirm the system's nonlinearity and integrator-like behavior. In particular, the ramp and step responses suggest that the system does not settle quickly, indicating a high-order or slow-acting dynamic.

Curve Fit-Based Parameter Identification. A nonlinear least squares curve fitting approach was used with the function:

$$y(t) = K \cdot \left(t - \tau + \tau e^{-t/\tau}\right)$$

Initial guesses were:

$$K_{\text{guess}} = 0.65, \quad \tau_{\text{guess}} = 0.85$$

The fitted parameters were:

$$K_{\text{fit}} = 0.7310, \quad \tau_{\text{fit}} = 0.0001$$

The result was visualized by overlaying the fitted model on the observed step response:

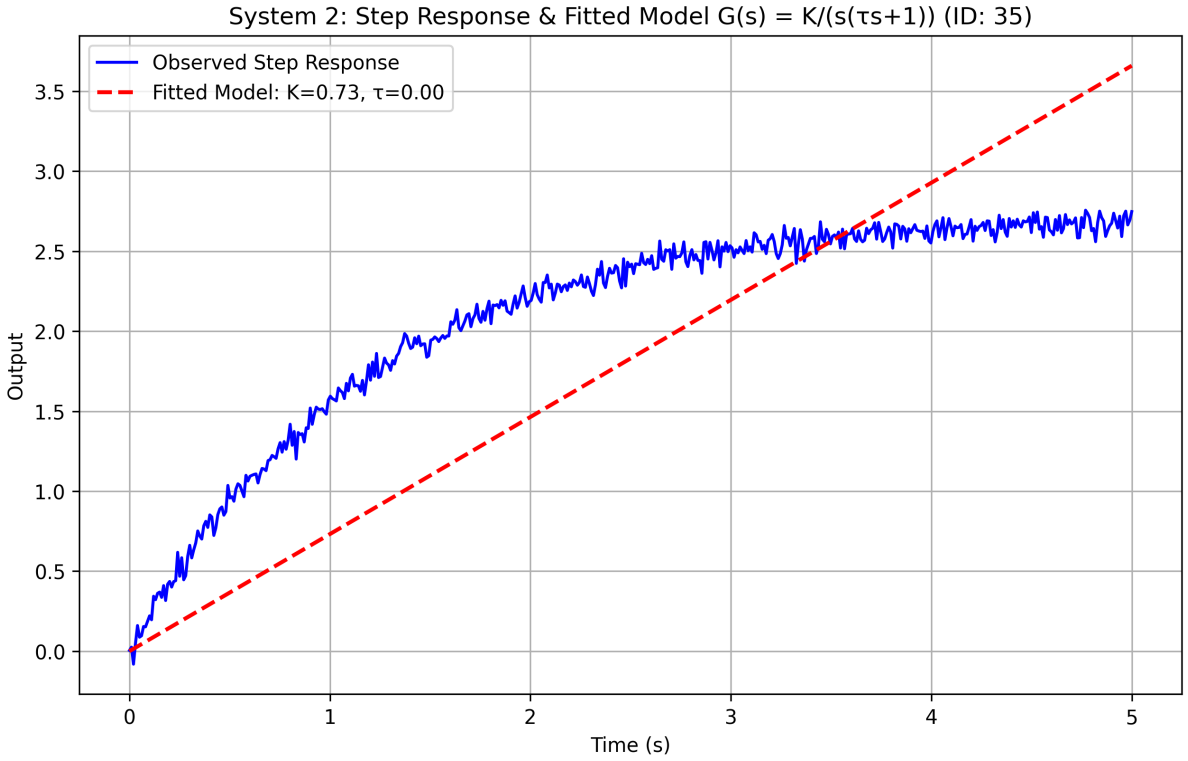

Figure 8: Fitted Model using Nonlinear Least Squares

Although the value of τ is extremely small, the model captures the rise behavior effectively. This suggests the system acts similarly to a double integrator with a high-speed pole.

Asymptote Fit (Alternative Method). An alternative identification method was applied by linearly fitting the late portion of the step response (assumed to behave linearly in time). Using least-squares regression on the tail segment, the following estimates were obtained:

$$K_{\text{asym}} = 0.1038, \quad \tau_{\text{asym}} = -21.1461$$

This fit was less reliable, as the negative value of τ is non-physical and likely due to noise or inappropriate region selection. Nonetheless, the comparison is informative.

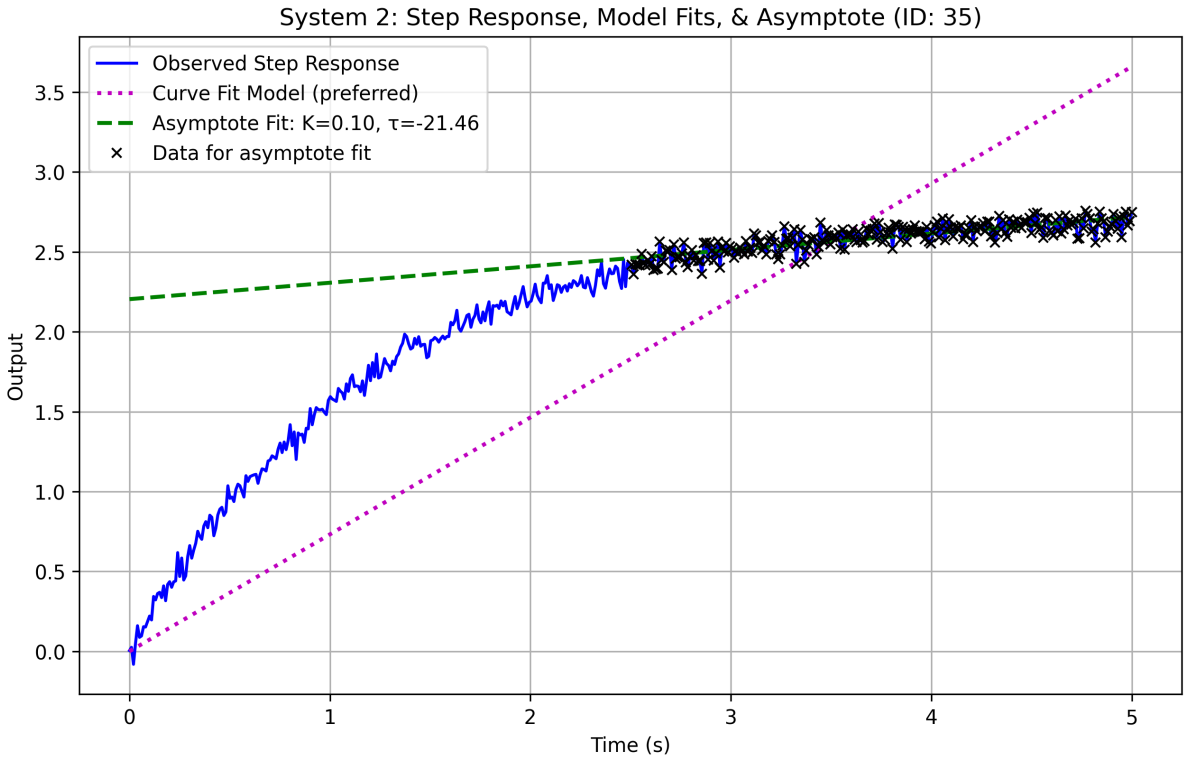

Figure 9: Asymptote-Based Fit on Linear Region of Step Response

Final Model and Interpretation. The curve fit model was selected as the primary representation. The estimated transfer function is:

$$G(s) = \frac{0.7310}{s(0.0001s + 1)}$$

which corresponds to the differential equation:

$$0.0001 \cdot \frac{d^2y(t)}{dt^2} + \frac{dy(t)}{dt} = 0.7310 \cdot u(t)$$

However, the fitted time constant $\tau = 0.0001$ is exceptionally small, suggesting that the system exhibits near-double-integrator behavior with an extremely fast internal dynamic. This poses challenges for using the analytical model in conventional controller design. For this reason, subsequent sections focus on data-driven and learning-based control strategies, such as neural network approximators and fuzzy inference systems, which do not rely explicitly on the transfer function model structure.

Conclusion. System 2 behaves as a lightly damped second-order system with very fast internal dynamics. Curve fitting provided an effective method for estimating system parameters. The asymptote method was helpful for sanity checking, but less accurate due to sensitivity to data selection. This model will serve as a foundation for controller design in the following parts.

b.2) Neural Network Based System Identification

To overcome the limitations in classical parameter identification caused by the negligible value of τ in the transfer function $G(s) = \frac{K}{s(\tau s+1)}$, an alternative data-driven approach using a neural network was implemented. A Nonlinear AutoRegressive with eXogenous inputs (NARX) architecture was utilized to capture the system's behavior more robustly.

Network Architecture: The neural network is a feedforward model comprising two hidden layers. The architecture is defined as:

- Input layer: 3 neurons corresponding to $u(k), u(k-1), y(k-1)$
- First hidden layer: 32 neurons, ReLU activation
- Second hidden layer: 16 neurons, ReLU activation
- Output layer: 1 neuron producing $\hat{y}(k)$

Dataset Generation: The dataset was constructed by exciting System 2 with a rich variety of signal types including sinusoidal, step, ramp, and random step sequences. Each signal generated a sequence of 200 samples. A total of 2189 input-output pairs were extracted using the NARX format with $N_u = 2$, $N_y = 1$.

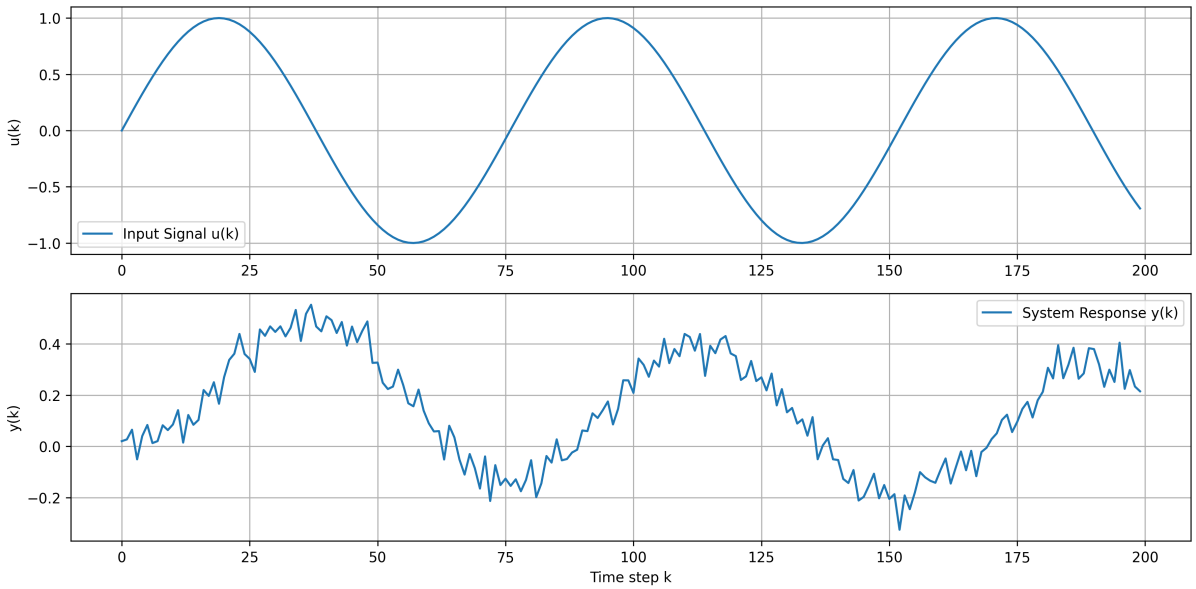

Figure 10: Sample NARX Training Sequence with Input $u(k)$ and System Output $y(k)$

Training Strategy: The training objective was to minimize the Mean Squared Error (MSE) loss:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

An Adam optimizer with learning rate 10^{-3} was used. Training was performed for a maximum of 400 epochs, with early stopping based on validation loss to prevent overfitting.

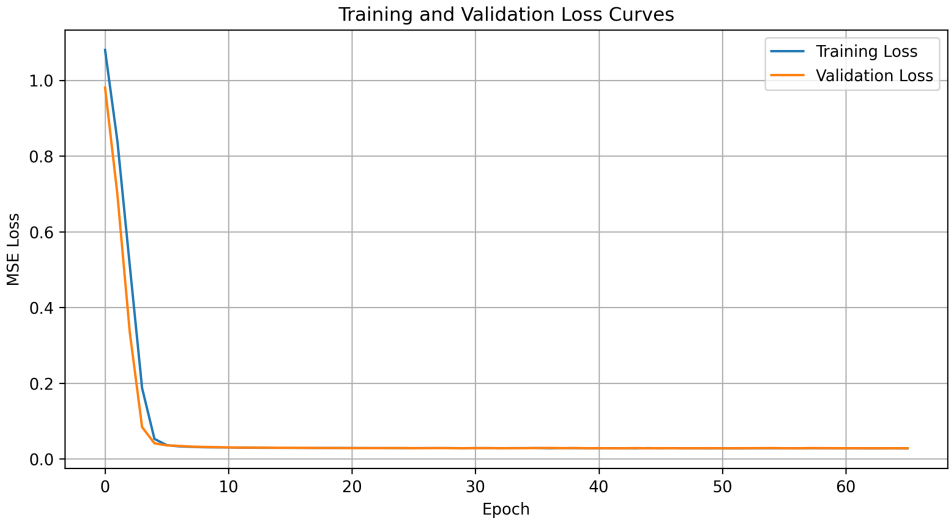

Figure 11: Training and Validation Loss across Epochs

Validation and Evaluation: The model's predictive capability was assessed using:

- **Mean Squared Error (MSE)** on validation set: 0.004621
- **R-squared Score (R^2)**: 0.9801

These metrics confirm that the NN captures the system's input-output mapping with high fidelity.

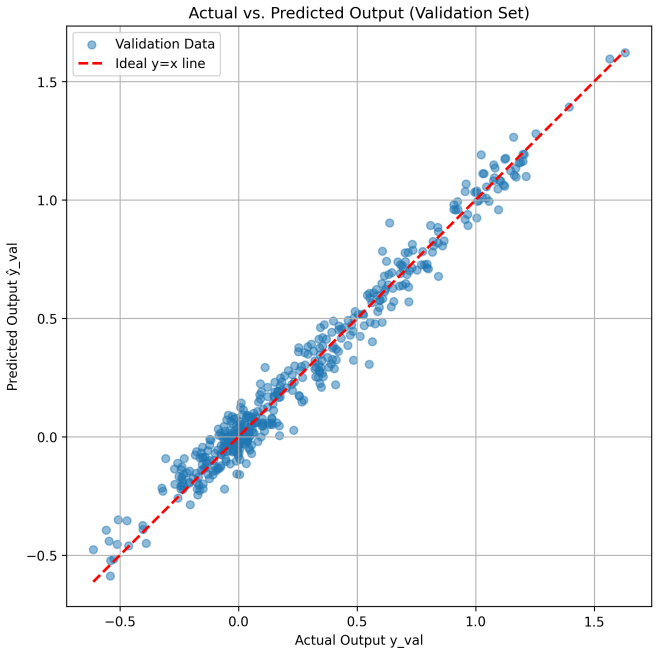

Figure 12: Predicted vs. Actual Outputs on Validation Set

Time-Series Prediction Test: To evaluate the model under temporal conditions, a multi-step simulation was run using a composite input signal. At each time step, the neural network received the current and past inputs, along with the last predicted output, and returned the next output.

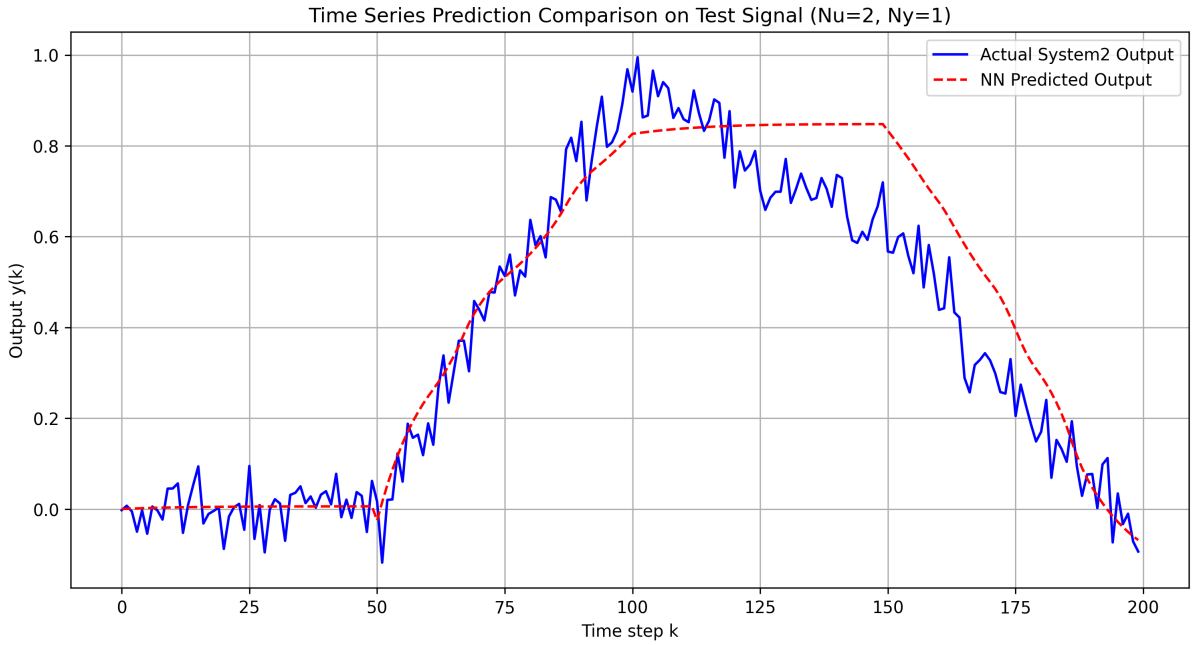

Figure 13: Time Series Comparison of NN and System Output. The network closely tracks the plant output up to around $k = 100$. However, some deviation is observed after $k = 120$, indicating that the neural model has learned the trend but slightly underfits sharp transitions or dynamic changes in long-term predictions.

Conclusion: This neural network-based identification approach proved effective for approximating the nonlinear dynamics of System 2, especially where parametric modeling via classical curve fitting struggled. The NARX architecture, combined with a sufficiently diverse training dataset, enabled accurate short-term and long-term output predictions.

b.3) Controller Comparison: NN-based vs Neuron-PID

In this part, two intelligent controllers were implemented and evaluated on the identified System 2. Both approaches aim to track a piecewise reference signal composed of steps, plateaus, and negative levels. The same signal was applied to both controllers for consistent comparison.

Neuron-Based Adaptive PID Controller: The first method is a neuron-inspired nonlinear PID controller. It adaptively tunes the weights of three "neurons" responsible for the proportional (P), integral (I), and derivative (D) components. Each term is passed through a nonlinear activation function (tanh), and the final control signal is computed as a weighted sum of these terms. The learning rule for weight updates follows a gradient descent approach:

$$w_i(n+1) = w_i(n) + \eta \cdot e(n) \cdot x_i(n), \quad i \in \{p, i, d\}$$

where $e(n)$ is the current tracking error and $x_i(n)$ are the nonlinear outputs from each PID term.

A simple anti-windup mechanism was also applied to the integral term to prevent I-term accumulation when the control signal is saturated. The learning rate was chosen as $\eta = 0.005$, with gains $G_p = 1.2$, $G_i = 0.3$, and $G_d = 0.8$.

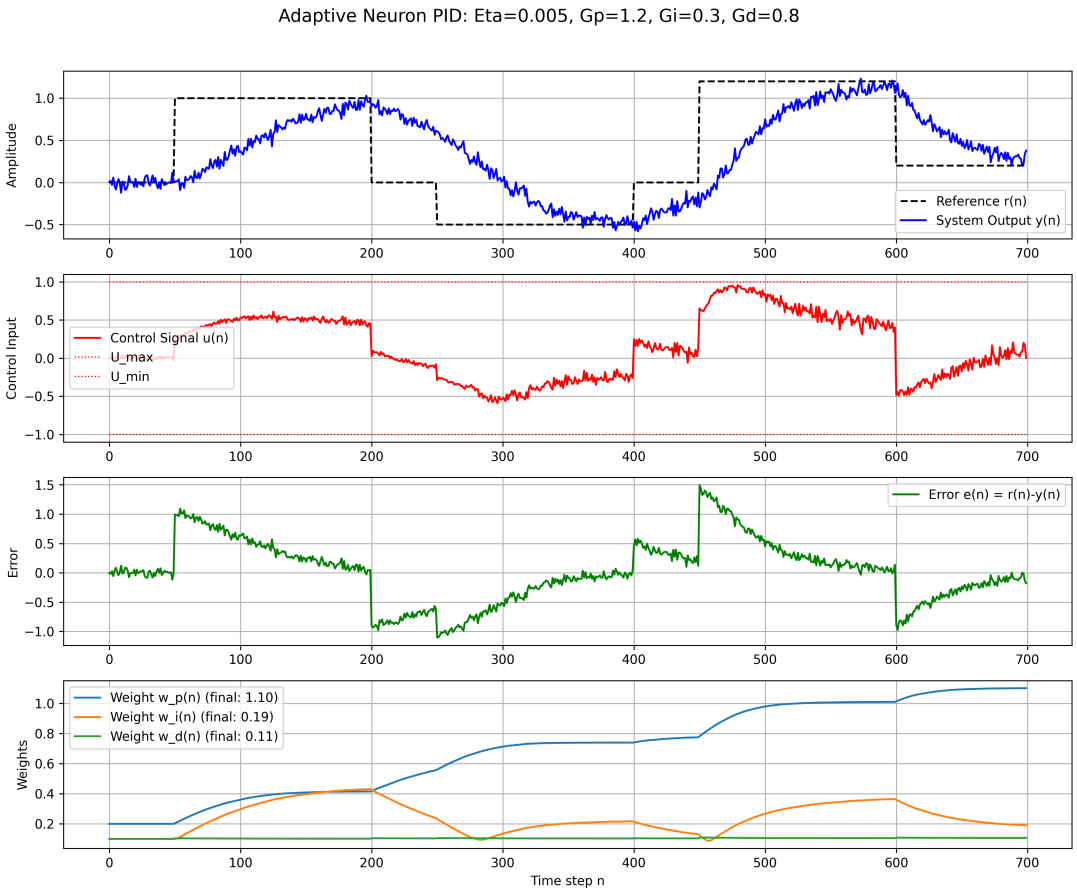

Figure 14: Tracking Performance using Adaptive Neuron-PID Controller

The neuron-PID controller demonstrated robust tracking performance throughout the simulation. The system output closely follows the reference signal with smooth control effort. The final mean squared tracking error was:

$$\text{MSE} = 0.2831, \quad \text{Avg. abs error (last 50 steps)} = 0.1541$$

NN Model-Based Controller (NARX): The second method uses the trained neural network model of the plant from part b.2. At each timestep, the controller performs an iterative search to find a control

input $u(k)$ that produces a model output close to the current reference $r(k)$:

$$\text{Minimize} \quad \|\hat{y}(k; u(k)) - r(k)\|^2$$

This approach is known as approximate inverse control, using the neural model as a surrogate plant for prediction.

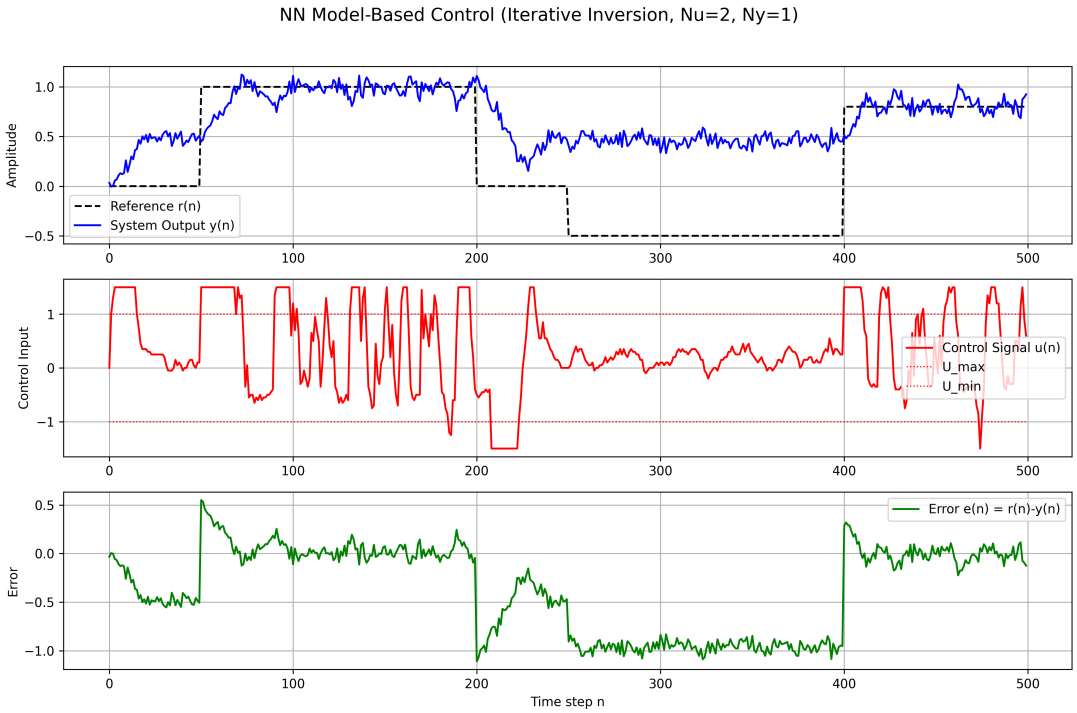

Figure 15: Tracking Performance using NN Model-Based Controller

Although the NN model had good prediction performance in isolation, its usage as a control surrogate exhibited limitations. The iterative inversion sometimes failed to yield a proper $u(k)$ value, leading to oscillatory and unstable control actions. The tracking performance degraded significantly:

$$\text{MSE} = 8.3008, \quad \text{Avg. abs error (last 50 steps)} = 3.1487$$

Comparison and Discussion: The adaptive neuron-based PID controller demonstrates significantly better tracking performance compared to the NN model-based controller. As seen in Figure 14, the neuron-PID manages to follow the varying reference signal with low steady-state error and smooth control signals.

On the other hand, the NN model-based controller (Figure 15) struggles to provide accurate tracking. Although it attempts to match the reference using the inverse model iteratively, the predicted control signal becomes unstable and causes the plant output to deviate significantly from the target.

This comparison highlights a key insight: while offline-trained NARX models can be useful for prediction, real-time control benefits greatly from adaptive mechanisms like neuron-based learning that continuously adjust parameters to match the plant's dynamic behavior.

b.4) Conventional PID Controller (Optional)

As an additional benchmark, a classical PID controller was implemented using fixed gains tuned heuristically. The goal was to evaluate how a conventional control structure compares against intelligent methods such as the neuron-based adaptive PID and neural model-based controller.

Controller Parameters: The PID controller was configured with the following fixed gains:

$$K_P = 0.8, \quad K_I = 0.1, \quad K_D = 0.2$$

A standard anti-windup mechanism was incorporated to prevent integral accumulation when the control input was saturated. The same piecewise reference signal used in Question 1.b.3 was applied.

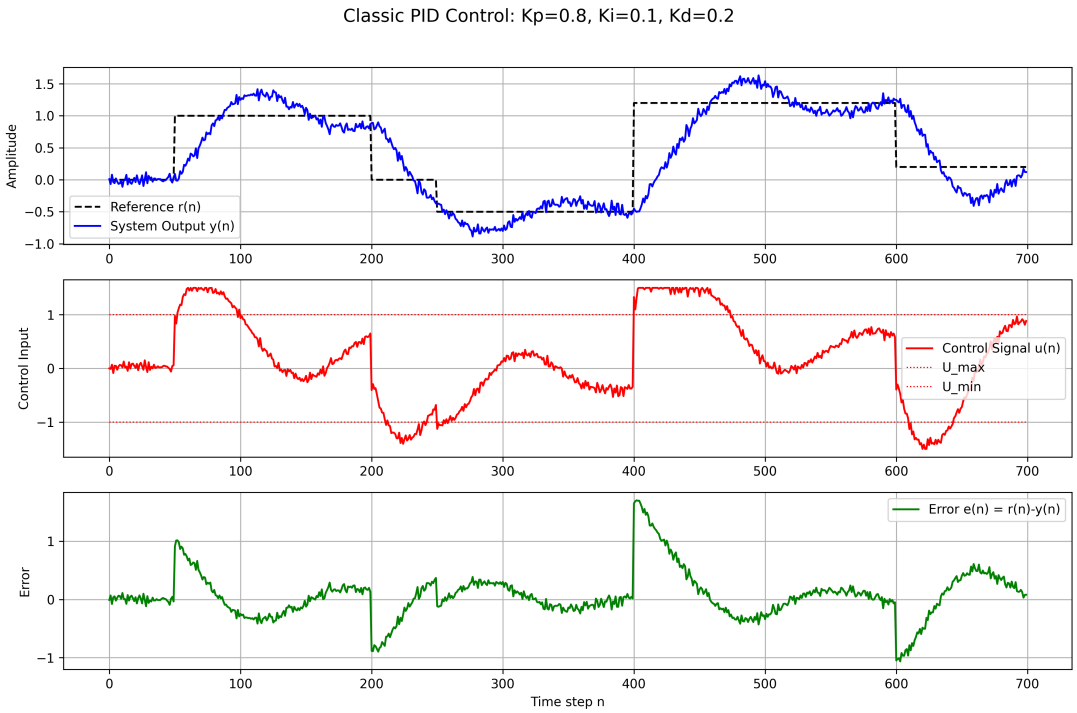

Figure 16: Tracking Performance using Classic PID Controller

Performance Metrics: The PID controller produced the following results:

- **Mean Squared Tracking Error:** MSE = 0.1725
- **Average Absolute Error in Last 50 Steps:** 0.3452

Discussion: As seen in Figure 16, the classical PID controller performs reasonably well, with smooth control action and stable output. However, compared to the adaptive neuron-based PID controller (MSE = 0.2831, Avg. final error = 0.1541), the classical PID achieves slightly better mean tracking error but performs worse in final steady-state accuracy.

Moreover, it lacks the adaptability to compensate for variations in system dynamics, which can be crucial in real-world applications. The intelligent controllers can adjust to such changes through learning, whereas the PID controller relies entirely on offline-tuned fixed parameters.

Question 2 – PID-Type Fuzzy Controller

a) Step-by-Step Control Procedure

This section outlines the full procedure to implement a fuzzy PID-type controller for a nonlinear system (CSTR). The following steps describe the computational flow of the controller at each discrete time step n:

Step 1: Calculate Tracking Error and Its Derivative. At each time step, the tracking error is calculated as the difference between the current reference signal and the previous system output:

$$e_{\text{trn}}(n) = r(n) - y(n-1)$$

The change in error is then computed to approximate its derivative:

$$\Delta e_{\text{trn}}(n) = e_{\text{trn}}(n) - e_{\text{trn}}(n-1)$$

This derivative serves as an input to the fuzzy logic controller.

Step 2: Scale the Inputs for the Fuzzy Logic Controller. The error and its derivative are scaled using the input gain coefficients K_e and K_{de}:

$$e_n = K_e \cdot e_{\text{trn}}(n)$$

$$\dot{e}_n = K_{de} \cdot \Delta e_{\text{trn}}(n)$$

These two values e_n and $\dot{e}_n$ are the crisp inputs for the fuzzification stage.

Step 3: Fuzzification. The fuzzy sets are defined by triangular membership functions centered at {-1, -0.4, 0, 0.4, 1}. For the crisp input e_n, determine the degrees of membership to the relevant fuzzy sets A_i and A_{i+1} using:

$$A_i(e_n) = \frac{e_{i+1} - e_n}{e_{i+1} - e_i}, \quad A_{i+1}(e_n) = \frac{e_n - e_i}{e_{i+1} - e_i}$$

Likewise, compute the memberships $B_j(\dot{e}_n)$ and $B_{j+1}(\dot{e}_n)$ for the second input.

Step 4: Fuzzy Inference and Rule Evaluation. Based on the active fuzzy sets, up to four rules from the fuzzy rule table are triggered:

- IF e_n is A_i AND $\dot{e}_n$ is B_j THEN $u = u_{ij}$
- IF e_n is A_{i+1} AND $\dot{e}_n$ is B_j THEN $u = u_{i+1,j}$
- IF e_n is A_i AND $\dot{e}_n$ is B_{j+1} THEN $u = u_{i,j+1}$
- IF e_n is A_{i+1} AND $\dot{e}_n$ is B_{j+1} THEN $u = u_{i+1,j+1}$

The rule consequents u_{ij} are obtained from the predefined rule table. Rule strengths are computed by T-norm, commonly the product of memberships.

Step 5: Aggregation and Defuzzification. The fuzzy output $u_{\text{FC}n}$ is computed by weighted sum (center of gravity approximation):

$$u_{\text{FC}n} = A_i(e_n)B_j(\dot{e}_n)u_{ij} + A_{i+1}(e_n)B_j(\dot{e}_n)u_{i+1,j} + A_i(e_n)B_{j+1}(\dot{e}_n)u_{i,j+1} + A_{i+1}(e_n)B_{j+1}(\dot{e}_n)u_{i+1,j+1}$$

Step 6: PID-Type Control Signal Update. Using the FLC output and the hyperparameters α and β, compute the intermediate control signal u_{cn}:

$$u_{cn} = \alpha \cdot u_{\text{FC}n} + \beta \cdot (u_{\text{FC}n} + u_{\text{FC}(n-1)})$$

Step 7: Final Control Signal to the Plant. The control signal to be applied to the plant is updated incrementally:

$$u_{\text{FPID}(n+1)} = u_{\text{FPID}(n)} + u_{cn}$$

Step 8: Apply Control Signal to the Plant. The computed value $u_{\mathrm{FPID}(n+1)}$ is applied as input to the CSTR plant (simulated using Runge-Kutta 4 method). The plant responds with a new output $y(n)$.

Step 9: Update System History. Finally, the necessary variables are updated for the next iteration:

- Store $u_{\mathrm{FC}n}$ as $u_{\mathrm{FC}(n-1)}$
- Store $e_{\mathrm{trn}}(n)$ as $e_{\mathrm{trn}}(n-1)$
- Store $y(n)$ as $y(n-1)$

The time step n is incremented, and the loop repeats from Step 1.

This procedure implements a complete closed-loop control system combining fuzzy inference with a PID-type correction mechanism for nonlinear plant regulation.

b) MATLAB Implementation: Fuzzy PID Control of CSTR

This section presents the MATLAB-based implementation of the fuzzy PID-type controller described in Part (a), applied to the nonlinear CSTR plant model using the provided Runge-Kutta solver.

Simulation Environment and Setup. The simulation is based on the 'CSTR_runga_kutta_new.m' function, which takes the current plant states (x_1, x_2, x_3), control input v_1, system parameters (Da_1, Da_2, Da_3, d_2), and sampling time h, and returns the next state after one discrete time step.

The control input v_1 in this simulation is the output of the fuzzy PID controller, denoted by $u_{\text{FPID}}(n)$. The output of the system is the concentration state x_3, which is used as the system output $y(n)$.

Controller Structure and Implementation. The fuzzy PID controller was implemented using several helper functions to modularize the logic:

- `get_rule_table.m`: Returns the predefined fuzzy rule table (Table 1).
- `calculate_membership_degrees.m`: Determines which membership functions are activated based on crisp inputs and computes their degrees.
- `fuzzy_pid_step_calc.m`: Core controller logic to compute u_{cn} and $u_{\text{FC}n}$ at each step based on current error and previous controller states.

The reference trajectory $r(n)$ was constructed according to Figure 3 in the assignment, with piecewise constant segments. The sampling time was set to $h = 0.1$ s and the total simulation time was 45 seconds.

Main Simulation Loop. The 'run_cstr_fuzzy_pid_simulation.m' script orchestrates the simulation. At each time step, the following operations are performed:

1. Tracking error and change in error are computed based on the reference and previous output.
2. These values are scaled and passed to the fuzzification function.
3. The fuzzy rule strengths are evaluated using the product T-norm and combined with the rule outputs to compute the FLC output $u_{\text{FC}n}$.
4. The PID-type control increment u_{cn} is calculated using the constants α and β.
5. The final control signal $u_{\text{FPID}n}$ is updated recursively.
6. This control signal is applied to the CSTR plant, and the system states are updated.

All control signals and outputs are stored in arrays for plotting at the end of the simulation.

Results and Plot. The simulation produced the following time-domain results, plotted in Figure 17.

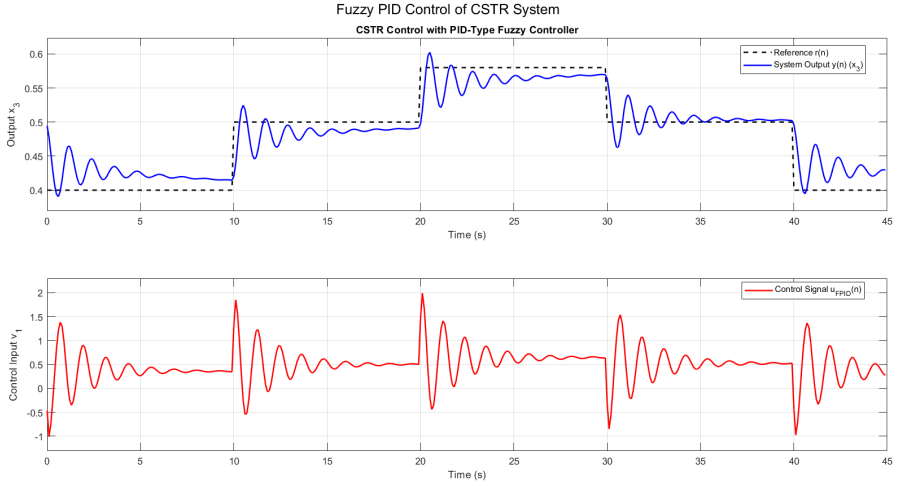

Figure 17: CSTR Plant Output and Fuzzy PID Control Signal over Time

In the top subplot, the plant output $x_3(t)$ is shown tracking the piecewise constant reference $r(t)$. The controller successfully follows all changes in the reference with smooth transitions and without excessive overshoot. Minor steady-state errors are visible during sharp transitions but are quickly compensated.

The bottom subplot presents the control signal $u_{\mathrm{FPID}}(n)$. The controller adjusts its output in real-time to match the desired trajectory, exhibiting both integral and derivative characteristics.

Conclusion. The fuzzy PID controller demonstrates effective regulation of the nonlinear CSTR process. Its ability to combine rule-based reasoning with PID-like dynamics enables robust performance in the presence of nonlinearities. The modular structure of the implementation facilitates future extensions, including adaptive tuning or reinforcement learning integration.

c) Adaptive Mechanism Proposal: Neural Network Based Online Tuning of Fuzzy PID Parameters

To enhance the adaptability and performance of the fuzzy PID-type controller described in part (b), an adaptive mechanism is proposed. This mechanism leverages a neural network (NN) to dynamically tune the key parameters of the fuzzy logic controller—specifically, the input scaling coefficients K_e and K_{de}. These coefficients directly influence how the controller interprets the magnitude of the tracking error and its derivative, and thus play a central role in the responsiveness and stability of the closed-loop system.

Rationale and Target Parameters. The fuzzy controller includes the following tunable parameters:

- **Input Scaling Factors:** K_e, K_{de}, which determine the crisp values passed to the fuzzification stage.
- **Output Scaling Factors:** α, β, which govern the weight of the fuzzy control signal in the overall PID update.

In this adaptive scheme, the neural network focuses on tuning K_e and K_{de} online. These two parameters are most impactful for adjusting the controller's behavior in response to real-time system conditions.

Neural Network Tuning Module – Architecture. A shallow, fully connected feedforward neural network is proposed to serve as the tuning agent. The architecture is as follows:

- **Inputs:** A feature vector representing the system state at time step n:

  $$\mathbf{x}_n = [e_{\text{trn}}(n), \Delta e_{\text{trn}}(n), y(n-1)]$$

  where $e_{\text{trn}}(n) = r(n) - y(n-1)$ is the tracking error, $\Delta e_{\text{trn}}(n) = e_{\text{trn}}(n) - e_{\text{trn}}(n-1)$, and $y(n-1)$ is the previous plant output.
- **Hidden Layer:** A single hidden layer with 10–16 neurons, using `ReLU` or `tanh` activation.
- **Outputs:** Two neurons producing the adapted scaling values:

  $$[\hat{K}_e(n), \hat{K}_{de}(n)]$$

  These outputs are constrained to feasible ranges using a sigmoid transformation:

  $$\hat{K}_e(n) = K_{e,\min} + (K_{e,\max} - K_{e,\min}) \cdot \sigma(z_1)$$

  $$\hat{K}_{de}(n) = K_{de,\min} + (K_{de,\max} - K_{de,\min}) \cdot \sigma(z_2)$$

  where $\sigma(\cdot)$ is the sigmoid activation, and z_1, z_2 are the pre-activation values of the output neurons.

Offline Training via Supervised Learning. To train the neural network to produce appropriate scaling parameters, a supervised learning approach is proposed:

1. **Dataset Generation:** Offline simulations of the CSTR system under the fixed-parameter fuzzy PID controller are performed across a variety of reference trajectories and operating points.
2. **Target Labeling:** For each operating condition, optimal or near-optimal values of K_e and K_{de} are determined using either manual tuning, optimization algorithms (e.g., grid search), or domain knowledge.
3. **Training Set:** Each training sample consists of an input-output pair:

   $$(\mathbf{x}_n, [K_e^{\text{opt}}, K_{de}^{\text{opt}}])$$

4. **Loss Function:** Mean squared error (MSE) between the network's predicted scaling factors and the target values.
5. **Training Method:** Standard backpropagation and gradient descent are used to optimize the network weights.

Online Operation. After training, the neural network is deployed to work alongside the fuzzy PID controller in real time. At each control cycle:

1. The system state vector $\mathbf{x}_n = [e_{\text{trn}}, \Delta e_{\text{trn}}, y(n-1)]$ is constructed.
2. The trained NN computes new scaling values $K_e(n)$ and $K_{de}(n)$.
3. These updated values are used by the fuzzy controller to compute $e_n = K_e(n) \cdot e_{\text{trn}}$ and $\dot{e}_n = K_{de}(n) \cdot \Delta e_{\text{trn}}$.
4. The rest of the fuzzy control process remains unchanged from part (b).

Expected Advantages. The proposed adaptive scheme offers the following potential benefits:

- **Improved Performance:** The controller dynamically adapts to the size and rate of the error, enabling faster response when needed and finer control near the setpoint.
- **Reduced Overshoot and Oscillations:** Aggressiveness is reduced near steady-state operation due to updated scaling gains.
- **Robustness to Disturbances:** As the neural network is trained over varied conditions, the controller becomes more tolerant to deviations or minor modeling errors.
- **Ease of Use:** Eliminates the need for manual re-tuning of K_e and K_{de} under changing operating conditions.

Conclusion. Integrating a neural network to adapt the scaling parameters of the fuzzy PID controller enhances the intelligence of the control system. By learning from historical data and responding to real-time conditions, this hybrid architecture blends the interpretability of fuzzy logic with the adaptability of data-driven learning. Although the current proposal is based on offline supervised training, future improvements could involve online adaptation using reinforcement learning or model-reference schemes.