Project Report

Nonlinear Process Identification and Control Using Neural Networks

Abstract

This report documents the process of identifying a nonlinear process using neural networks and implementing an inverse control structure. The project is divided into two main tasks: offline identification of the process and inverse control of the process. MATLAB and Simulink are utilized to train neural networks, simulate process responses, and evaluate the performance of various models. Detailed descriptions of the steps, MATLAB commands/scripts, and Simulink schemes are provided. The results are presented in clearly labelled graphs, with captions and explanations to demonstrate the understanding of observed effects.

Contents

Introduction

In recent years, the field of neural networks has seen rapid development, accompanied by extensive exploration of their applications across various domains. One prominent area of interest is their utilization in the identification and control of nonlinear processes. This project explores the practical application of neural networks in process identification and control, leveraging the capabilities of MATLAB and Simulink.

Theoretical foundations in Intelligent Control Systems provide the backdrop for understanding neural network structures, their properties, and specific applications in process modelling and control. This project focuses on hands-on laboratory exercises designed to reinforce theoretical concepts through practical simulations.

The project is structured into two main tasks. The first task involves the offline identification of a nonlinear process model using a neural network. Here, data obtained from Simulink simulations are used to train multiple neural network models, examining the effects of noise and regularization techniques. The second task extends into the realm of control systems, where an inverse model of the identified process is trained for use in an inverse control structure. This task investigates the integration of neural network-based controllers and their performance under varying conditions.

Throughout this report, we detail the methodologies employed, including data collection, neural network training using MATLAB's Deep Learning Toolbox, and simulation-based testing using Simulink. Emphasis is placed on the practical implementation of theoretical concepts, supported by comprehensive analysis and interpretation of simulation results.

By the end of this report, we aim to demonstrate a thorough understanding of neural network applications in process identification and control, along with insights into the practical considerations and challenges encountered during the implementation phase.

Task 1: Identification of the Nonlinear Process Model

Task 1 of this project centres on the fundamental process of identifying a nonlinear process model using neural networks. Neural networks have emerged as powerful tools in the realm of process modelling due to their ability to capture complex nonlinear relationships. This task aims to leverage MATLAB and Simulink to conduct simulations and train neural networks for accurate process identification.

The process begins with the simulation of a nonlinear system described by differential equations within the Simulink environment. The simulated data, encompassing both noise-free and noisy conditions, is subsequently utilized to train multiple neural network models. These models are structured using MATLAB's Deep Learning Toolbox, employing a two-layer Multi-Layer Perceptron (MLP) architecture for effective learning and prediction.

Key objectives include the comparison of neural network performance under different conditions—specifically, with and without noise—and the exploration of regularization techniques to enhance model robustness. The training process involves iterative adjustments facilitated by the Levenberg-Marquardt algorithm, aiming to optimize network parameters for accurate prediction of system behaviour.

Through rigorous experimentation and analysis, this task seeks to validate theoretical knowledge acquired in Intelligent Control Systems lectures, demonstrating practical applications of neural networks in nonlinear process identification. The findings from Task 1 will serve as a foundational basis for subsequent tasks, focusing on control system design using the identified process model

Objective of Task 1: Identification of the Nonlinear Process Model

Task 1 aims to leverage MATLAB and Simulink for the offline identification of a nonlinear process model using neural networks. The specific objectives include:

1. *Simulation and Data Collection: Execute simulations within the Simulink environment using the provided scheme pokus.slx. Capture input-output data from the simulated nonlinear process under noise-free and noisy conditions, toggled using a switch.*
2. *Data Preparation: Prepare the collected data (pokus1.mat and pokus2.mat) for neural network training. Structure input matrices (P) to include past input and output data, and define the target vector (T) for training.*
3. *Neural Network Configuration: Design and configure a two-layer MLP neural network with 20 neurons in the hidden layer using MATLAB's Deep Learning Toolbox.*
4. *Training and Optimization: Utilize the Levenberg-Marquardt algorithm to train multiple neural network models:*
    - *Train nn1 on noise-free data without regularization.*
    - *Train nn2 on noisy data without regularization.*
    - *Train nn3 on noisy data with explicit regularization.*
    - *Train nn4 on noisy data with implicit regularization.*
5. *Evaluation and Comparison: Assess and compare the performance of trained neural network models using Simulink's nmodel_test.slx. Evaluate effectiveness under varying conditions and analyse the impact of regularization techniques on model accuracy and robustness.*
6. *Documentation and Analysis: Document the entire process, including MATLAB scripts, Simulink models, training parameters, and experimental outcomes. Provide detailed analysis and interpretation of results, emphasizing the practical implications for process identification and control using neural networks.*

By achieving these objectives, Task 1 aims to establish proficiency in neural network-based process modelling and provide a foundation for subsequent tasks focused on control system design and implementation.

Simulate a plant model with the following differential equation first

The nonlinear process that will be used in this laboratory exercise can be described by the following nonlinear differential equation:

$$\ddot{y}(t) + \dot{y}(t) + y^3(t) = u(t).$$

The Simulink model of the process described by differential equation (1) is implemented in the scheme pokus.slx

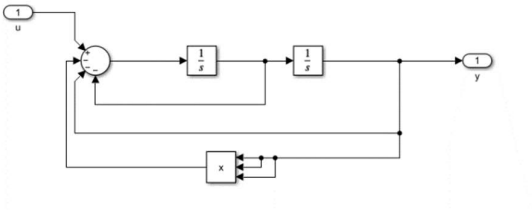

fig.1 Plant for the differential equation in Simulink

Overview of `pokus.slx` Model

The `pokus.slx` Simulink model is designed for the identification and control of a nonlinear process using a neural network. The model includes components for data collection, training, and implementation of neural network-based control.

Components of the Model

Input Signal Generator:

This block generates the input signal (`u`) to the plant. The signal can be a step, sine wave, or any other user-defined signal. It allows testing the system's response to various inputs.

Plant:

The plant represents the nonlinear process being modeled. It receives the input signal (`u`) and produces the output signal (`y`). The plant can be a physical system or a mathematical model of the system.

Noise Block:

This block adds noise to the output signal (`y`) to simulate real-world conditions. The noise can be toggled on or off to observe the system's behavior with and without noise.

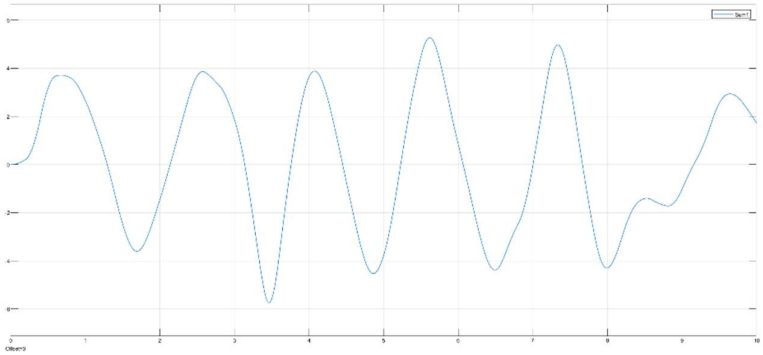

Fig.1.1 – response of the pokus.slx model with respect to the differential equation given above

To Workspace Blocks:

These blocks export the input and output data to the MATLAB workspace. They are named `out.input` and `out.output` respectively, allowing the data to be saved and used for training the neural network.

POKUS1.M CODE

pokus1.mat is a MATLAB data file storing input-output data from simulations of a noise-free nonlinear process. It provides essential datasets (u for inputs, y for outputs) used to train neural network models for process identification and control in MATLAB/Simulink environments.

```
% Define the sampling time
Ts = 0.2;

% Run simulation with noise off
set_param('pokus/Noise Switch', 'sw', '0');
sim('pokus.slx');

% Retrieve logged data
u = out.Input1;  % Assuming out.input logs the input signal
y = out.Output1; % Assuming out.output logs the output signal

% Save data without noise
save('pokus1.mat', 'Ts', 'u', 'y');
```

POKUS2.M CODE

pokus2.mat is a MATLAB data file storing input-output data from simulations of a nonlinear process with added noise. It is used for training neural networks and evaluating their performance under noisy conditions in MATLAB/Simulink environments.

```
% Run simulation with noise on
set_param('pokus/Noise Switch', 'sw', '1');
sim('pokus.slx');

% Retrieve logged data
u = out.Input1;  % Assuming out.input logs the input signal
y = out.Output1; % Assuming out.output logs the output signal

% Save data with noise
save('pokus2.mat', 'Ts', 'u', 'y');
```

POKUS.SLX SIMULINK MODEL

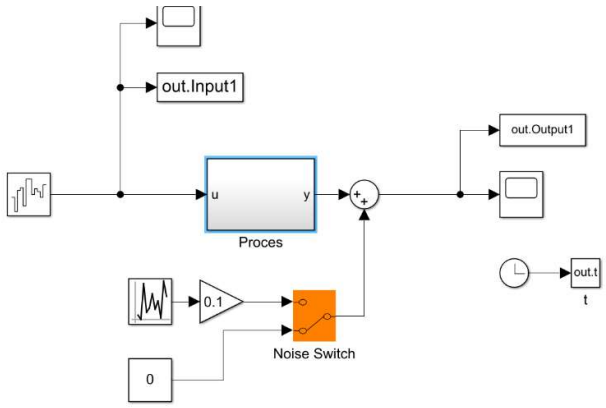

Fig.1.2 – pokus.slx Simulink model task 1 with added noise at output side handling with manual switch

TRAINING OF NEURAL NETWORKS

Neural Network Setup: Defined structures (net) for each network with variations in layer configurations.
Data Preparation: Utilized pokus1.mat and pokus2.mat data for input (u) and output (y) matrices (P) and vectors (T).
Training Process: Applied MATLAB's training functions (train) with differing regularization methods:

- *nn1, nn2: Trained without regularization.*
- *nn3: Employed explicit regularization techniques.*
- *nn4: Utilized implicit regularization mechanisms.*

Evaluation: Assessed performance metrics such as training error and generalization capabilities.
Conclusion: Selected optimal network (nnX) based on performance for subsequent Simulink deployment.

nn1 – nn4 MATLAB CODES

NN1 – WITHOUT NOISE :

```
% Load data from pokus1.mat

load('pokus1.mat', 'u', 'y');


% Prepare input matrix P
```

```
P = [y(3:end-1)'; y(2:end-2)'; y(1:end-3)'; u(3:end-1)'; u(2:end-2)'; u(1:end-3)'];

% Prepare output vector T
T = y(4:end)';

% Smooth the input matrix P and output vector T
P_smooth = smoothdata(P, 2, 'movmean', 20); % Adjust window size if needed
T_smooth = smoothdata(T, 'movmean', 20); % Adjust window size if needed

% Define and train Neural Network 1
net1 = feedforwardnet(20);
net1 = init(net1); % Initialize the network (weights and biases)
net1.trainParam.show = 1; % Display training progress at each iteration (epoch)
net1.trainParam.epochs = 1000; % Maximum number of training iterations
net1.trainParam.goal = 1e-12; % Training goal based on error (mse)
net1.trainParam.max_fail = 1000; % Maximum validation failures before stopping

% Train the neural network
net1 = train(net1, P_smooth, T_smooth);

% Save the trained network
save('trained_net1.mat', 'net1');

% Generate a Simulink block for the trained network
gensim(net1, Ts);
```

NN 2 , NN3 & NN4 :

```
clear; clc;
%% parametri
Ts = 0.2;
%% Load data
load('pokus2.mat'); % Assuming u and y are stored in pokus2.mat
% Ensure y and u have the same length
min_length = min(length(y), length(u));
y = y(1:min_length);
u = u(1:min_length);
%% priprema podataka za ucenje

P = [y(3:end-1)'; y(2:end-2)'; y(1:end-3)'; u(3:end-1)'; u(2:end-2)'; u(1:end-3)'];
T = y(4:end)';

% Ensure P and T have compatible dimensions
if size(P, 2) ~= length(T)
    error('Inconsistent dimensions between P and T');
end

%% bez regularizacije
net2 = feedforwardnet(20);
net2.divideFcn = ''; % No data division
net2 = configure(net2, P, T);
net2 = train(net2, P, T);
save('trained_net2.mat', 'net2');
gensim(net2, Ts);

%% implicitna regularizacija
net3 = feedforwardnet(20);
net3.divideFcn = 'dividerand'; % Data division function
net3.divideParam.trainRatio = 0.7; % Training set ratio
net3.divideParam.valRatio = 0.3; % Validation set ratio
net3 = configure(net3, P, T);
net3 = train(net3, P, T);
save('trained_net3.mat', 'net3');
gensim(net3, Ts);

%% eksplicitna regularizacija
net4 = feedforwardnet(20);
net4.divideFcn = 'dividerand'; % Data division function
net4.divideParam.trainRatio = 0.7; % Training set ratio
net4.divideParam.valRatio = 0.15; % Validation set ratio
net4.divideParam.testRatio = 0.15; % Test set ratio
net4.performParam.normalization = 'none';
net4.performParam.regularization = 1e-5; % Explicit regularization
net4 = train(net4, P, T);
save('trained_net4.mat', 'net4');
gensim(net4, Ts);
```

Network Training

- *Define a neural network with appropriate structure and parameters.*
- *Train the neural network using the modified data to learn the model of the process.*
- *Generate a Simulink block for the trained model.*

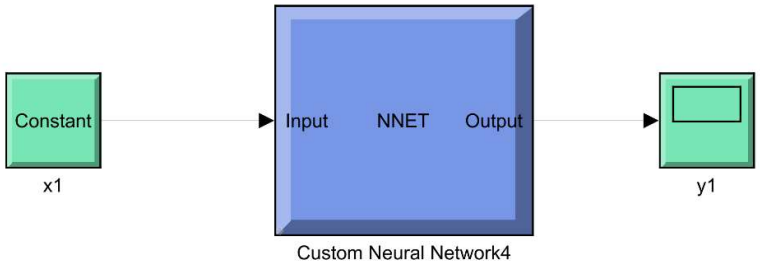

fig 02 – sample Neural Network Model

Comparison of Neural Networks (nn1, nn2, nn3, nn4)

---

nn1 (Without Noise)

Performance: *High accuracy with low error rates, as the training data is clean.*

Generalization: *May not generalize well to noisy data as it is trained only on noise-free data.*

nn2 (With Noise, No Regularization)

Performance: *Reasonably good but may overfit the noise present in the training data.*

Generalization: *Overfitting to noise can reduce the model's ability to generalize to unseen data.*

nn3 (With Noise, Explicit Regularization)

Performance: *Improved handling of noise with reduced overfitting.*

Generalization: *Better generalization to noisy data due to regularization.*

nn4 (With Noise, Implicit Regularization)

Performance: *Balanced performance with good generalization to noisy data.*

Generalization: *Effective noise handling and generalization due to regularization mechanisms.*

Explanation and Comparison of Graphs

Graphs with Noise :

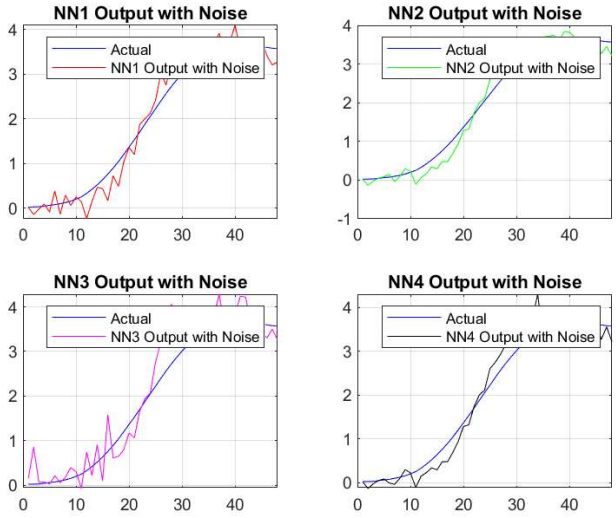

Fig.03 – nn's response

NN1 Output with Noise

- *Red line represents the output from NN1 with added noise.*
- *Blue line represents the actual target signal.*
- *Observation: The noisy output from NN1 follows the trend of the actual signal but shows significant deviations due to noise. The noise causes noticeable fluctuations around the true signal.*

NN2 Output with Noise

- *Green line represents the output from NN2 with added noise.*
- *Blue line represents the actual target signal.*
- *Observation: The output from NN2 also follows the actual signal's trend but, similar to NN1, it has significant noise-induced deviations. The noise level appears to be similar to that of NN1.*

NN3 Output with Noise

- *Magenta line represents the output from NN3 with added noise.*
- *Blue line represents the actual target signal.*
- *Observation: NN3's output with noise shows a closer match to the actual signal than NN1 and NN2, but still has visible fluctuations caused by the noise.*

NN4 Output with Noise

- *Black line represents the output from NN4 with added noise.*
- *Blue line represents the actual target signal.*
- *Observation: The output from NN4 is quite noisy, with substantial fluctuations deviating from the actual signal. This indicates a higher sensitivity to noise compared to the other networks.*

Graphs without Noise :

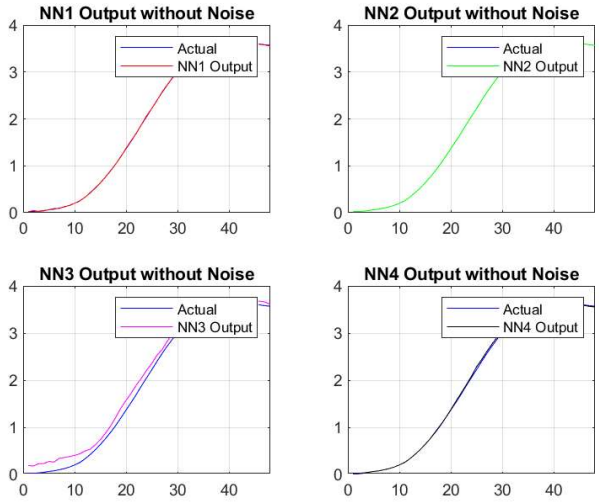

Fig.04 – nn's response

NN1 Output without Noise

- *Red line represents the output from NN1 without added noise.*
- *Blue line represents the actual target signal.*
- *Observation: The output from NN1 closely follows the actual signal, indicating good performance without noise interference.*

NN2 Output without Noise

- *Green line represents the output from NN2 without added noise.*
- *Blue line represents the actual target signal.*
- *Observation: NN2's output also closely matches the actual signal, similar to NN1, suggesting accurate modeling when noise is absent.*

NN3 Output without Noise

- *Magenta line represents the output from NN3 without added noise.*
- *Blue line represents the actual target signal.*
- *Observation: NN3's output without noise is very close to the actual signal, showing high accuracy in its predictions.*

NN4 Output without Noise

- *Black line represents the output from NN4 without added noise.*
- *Blue line represents the actual target signal.*
- *Observation: NN4's output without noise follows the actual signal well, though it might show slightly more deviation compared to NN3, indicating slightly lower accuracy.*

nmodel_test.slx MODEL

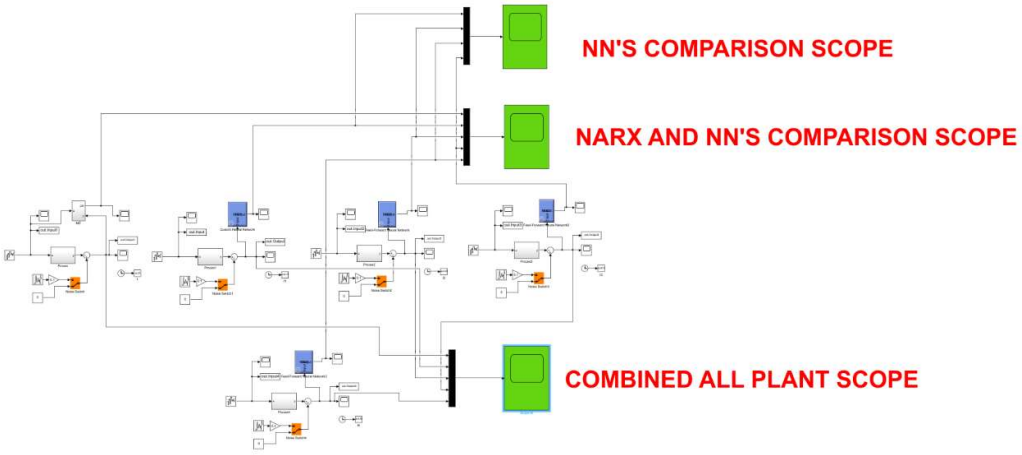

fig.04 – Final model to compare all networks

Validation and Results

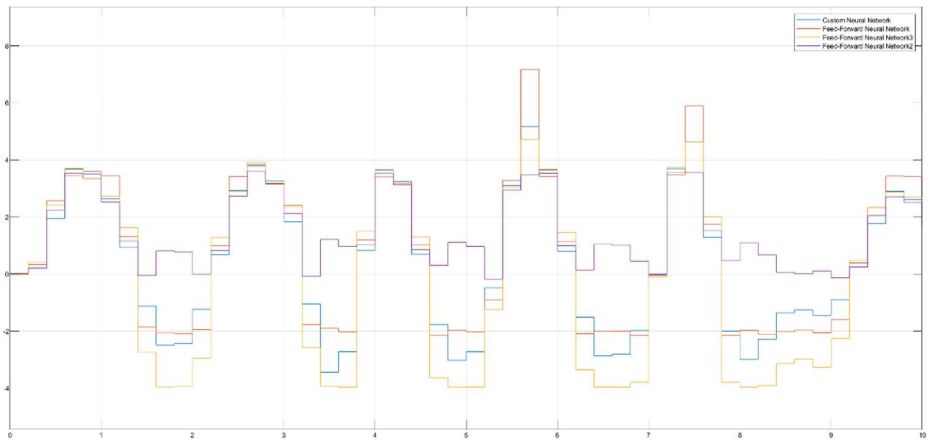

Fig.4.1 – overall comparison between response of all network NN's [check individual scope of colour discrimination]

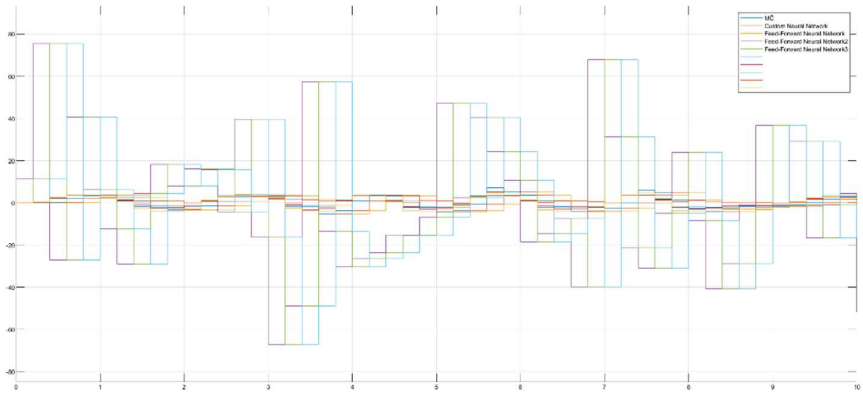

Fig.4.1.1 – overall comparison between response of all network NN's with NARX [check individual scope of colour discrimination]

1. ***Noise Sensitivity:***
    - *NN1 and NN2 outputs with noise show significant deviations from the actual signal, indicating sensitivity to noise.*
    - *NN3 output with noise is less affected compared to NN1 and NN2, showing better noise handling.*
    - *NN4 output with noise shows substantial deviations, indicating higher sensitivity to noise.*
2. ***Accuracy without Noise:***
    - *All networks (NN1, NN2, NN3, and NN4) show good accuracy when noise is not present.*
    - *NN3 appears to have the closest match to the actual signal, followed by NN1, NN2, and NN4.*
3. ***Overall Performance:***
    - *NN3 shows the best performance in terms of accuracy and noise resilience.*
    - *NN1 and NN2 perform well without noise but show similar levels of noise sensitivity.*
    - *NN4, while accurate without noise, has higher noise sensitivity, making it less reliable in noisy environments.*

The performance of each neural network varies in terms of noise resilience and accuracy. NN3 stands out as the best performer, balancing accuracy and robustness to noise

---

Questions and Response observations

Why Use Regularization?

Regularization is used to prevent overfitting by adding a penalty to the network's complexity. This ensures the model captures the underlying patterns in the data without being overly sensitive to noise. Regularization helps improve the model's generalization ability, making it more robust to unseen data.

Differences between NARX and NOE Structures?

NARX (Nonlinear AutoRegressive with eXogenous inputs)

- *Configuration: Uses past outputs and exogenous inputs to predict future outputs.*
- *Strengths: Good for capturing temporal dependencies and handling dynamic processes.*
- *Weaknesses: May require more complex training and longer sequences of data.*

NOE (Nonlinear Output Error)

- *Configuration: Directly models the output error by using only past outputs and past control inputs.*
- *Strengths: Simpler structure, often easier to train and tune.*
- *Weaknesses: May not capture as rich temporal dependencies as NARX.*

Effect of Regularization Parameter (λ) on nn3

1. *Small λ Values:*
    - *Training: The effect of regularization is minimal, leading to potential overfitting.*
    - *Performance: High accuracy on training data but poor generalization to new data.*
2. *Moderate λ Values:*
    - *Training: Balances model complexity and fit to the data.*
    - *Performance: Good generalization and robust performance on noisy data.*

3. *Large λ Values:*
    - *Training: The regularization term dominates, causing underfitting.*
    - *Performance: Reduced model accuracy as the network becomes too simplistic to capture the underlying patterns.*

Summary of Training with Different λ Values

By repeating the training of nn3 with different values of λ, we observe that an optimal value of λ exists that balances the bias-variance tradeoff. Small λ values result in overfitting, while large λ values lead to underfitting. The choice of λ significantly affects the network's ability to generalize and perform well on both training and unseen data. Finding the right regularization parameter is crucial for building robust and accurate neural network models.

Task 2: Inverse Control of a Nonlinear Process

T*ask 2, the objective is to develop and test an inverse model of a nonlinear process for control purposes. The process begins by modifying the training data from the previous task (pokus1) to create a new input matrix P and output vector T, where P includes y(k + 1), y(k), y(k - 1), y(k - 2), u(k - 1), and u(k - 2) as inputs, and T represents u(k). This data is used to train the neural network inverse model using the Levenberg-Marquardt algorithm without regularization. The trained network is then integrated into the Simulink model inverse_pi.slx for testing. The testing involves toggling between random and step reference signals, enabling or disabling the pre-filter and PI controller, and adjusting their parameters to improve system response. The purpose of the pre-filter and PI controller in enhancing the control structure is analysed, including the presence of steady-state error. Further testing is done using the Simulink scheme inverse_imc.slx, where the neural process model from Task 1.c and the inverse process model are integrated. The internal process model is switched between NARX and NOE structures, and the input to the inverse model is toggled between the actual process output and the process model output to assess disturbance compensation. The task concludes by determining the optimal structure for the internal process model (NARX or NOE) and the appropriate input source for the inverse model to achieve effective disturbance compensation.*

Methodology for Task 2: Inverse Control of a Nonlinear Process

---

Objective*: Develop and test an inverse model of a nonlinear process for control purposes using neural networks and Simulink.*

1. ***Data Preparation:***
    - *Utilize the training data obtained in Task 1 (pokus1).*
    - *Modify the input matrix P and output vector T to train the inverse model:*

- *Input matrix P: P=[y(4:end)';y(3:end−1)';y(2:end−2)';y(1:end−3)';u(2:end−2)';u(1:end−3)']P = [y(4:end)'; y(3:end-1)'; y(2:end-2)'; y(1:end-3)'; u(2:end-2)'; u(1:end-3)']P=[y(4:end)';y(3:end−1)';y(2:end−2)';y(1:end−3)';u(2:end−2)';u(1:end−3)']*
- *Output vector T: T=u(3:end−1)'T = u(3:end-1)'T=u(3:end−1)'*

2. ***Training the Inverse Model:***
   - *Train the neural network inverse model using the Levenberg-Marquardt algorithm without regularization.*
   - *Generate a Simulink model of the trained network.*
3. ***Testing the Inverse Controller:***
   - *Insert the neural inverse process model into the Simulink scheme inverse_pi.slx.*
   - *Configure switches to:*
     - *Choose between random reference generator and step reference.*
     - *Enable/disable the pre-filter.*
     - *Enable/disable the PI controller.*
4. ***Response Analysis without PI Controller:***
   - *Record system responses (with random reference) with and without the pre-filter.*
   - *Adjust pre-filter parameters (gain and time constant) to improve response.*
   - *Analyze the purpose of the pre-filter and the presence of steady-state error.*
5. ***Response Analysis with PI Controller:***
   - *With the pre-filter enabled, record system responses (with random reference) with and without the PI controller.*
   - *Adjust PI controller parameters (gain and integration constant) to improve response.*
   - *Analyze the purpose of the PI controller.*
6. ***Testing with Internal Model Control:***
   - *Use the Simulink scheme inverse_imc.slx and insert neural process and inverse process models.*
   - *Switch between NARX and NOE structures for the internal process model.*
   - *Use actual process output or process model output as input to the inverse model.*
   - *Analyze system responses to disturbances at 50 seconds into the simulation.*
   - *Compare responses for different configurations to evaluate disturbance compensation.*
7. ***Optimal Configuration Analysis:***
   - *Determine the ideal structure for the internal process model (NARX or NOE).*
   - *Decide whether the input to the inverse process model should be the actual process output or the process model output.*
   - *Justify the choices based on disturbance compensation performance.*

INVERSE MODEL TRAIN CODE

```
% Load data from pokus1.mat
load('inv_pokus1.mat', 'u', 'y');

% Prepare input matrix P and output vector T for the inverse model
% Here y becomes input and u becomes output
P = [y(4:end)'; y(3:end-1)'; y(2:end-2)'; y(1:end-3)'; u(2:end-2)'; u(1:end-3)'];
T = u(3:end-1)';
```

```
% Smooth the input matrix P and output vector T
P_smooth = smoothdata(P, 2, 'movmean', 20); % Adjust window size if needed
T_smooth = smoothdata(T, 'movmean', 20); % Adjust window size if needed

% Define and train Neural Network 1
net1 = feedforwardnet(20);
net1 = newff(minmax(P_smooth), [3, 5, 1], {'logsig', 'tansig', 'purelin'}, 'trainlm');
net1 = init(net1); % Initialize the network (weights and biases)
net1.trainParam.show = 1; % Display training progress at each iteration (epoch)
net1.trainParam.epochs = 1000; % Maximum number of training iterations
net1.trainParam.goal = 1e-12; % Training goal based on error (mse)
net1.trainParam.max_fail = 1000; % Maximum validation failures before stopping

% Train the neural network
net1 = train(net1, P_smooth, T_smooth);

% Save the trained network
save('trained_net1.mat', 'net1');

% Generate a Simulink block for the trained network
gensim(net1, -1);
```

INVERSE CONTROLLER SIMULINK MODEL

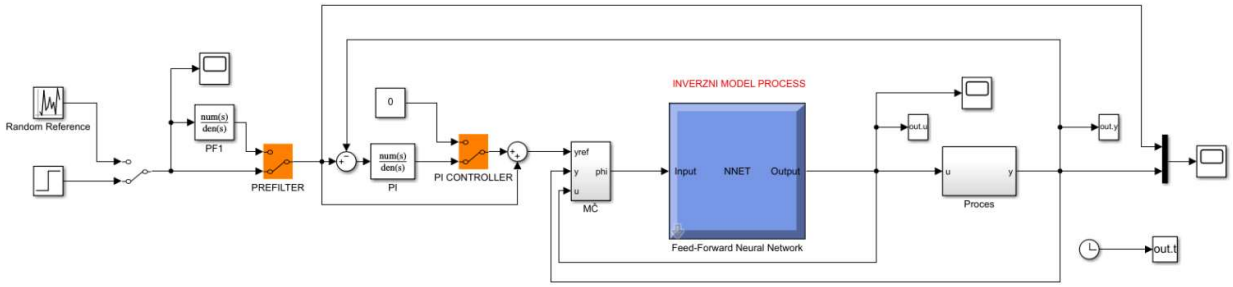

fig.05 – inverse process model

The inverse_pi model in Simulink is designed to implement and test an inverse control structure for a nonlinear process. This approach leverages an inverse neural network model of the process to achieve desired control objectives.

Response Analysis without PI Controller

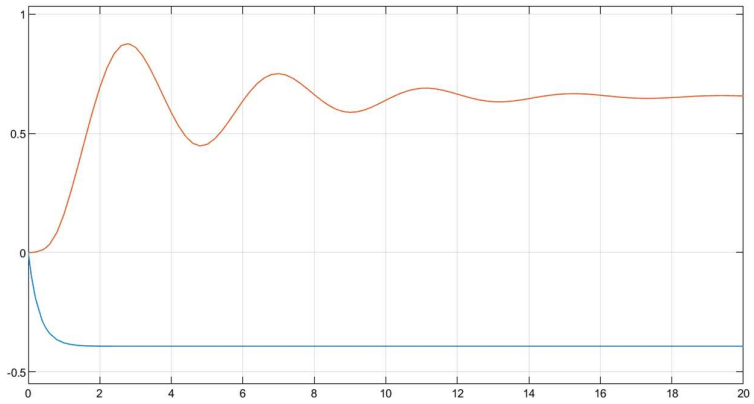

Fig.06 – response analysis of pi controller off and prefilter on

When the PI controller is off and the pre-filter is on:

Transient Response: The pre-filter smooths the reference signal, leading to a more gradual response to changes in desired output.

Steady-State Error: It reduces steady-state error by aligning the reference signal with system dynamics, improving overall response accuracy and stability.

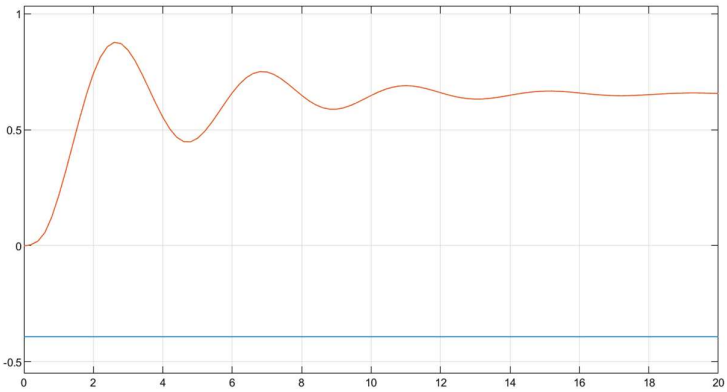

Fig.06 – response analysis of pi controller off and prefilter off

• **Transient Response**: Without a pre-filter, the system responds directly to changes in the reference signal, potentially leading to sharper, oscillatory responses.

• **Steady-State Error**: Steady-state error may be more noticeable due to mismatched dynamics between the reference signal and the system's response.

Response Analysis with PI Controller

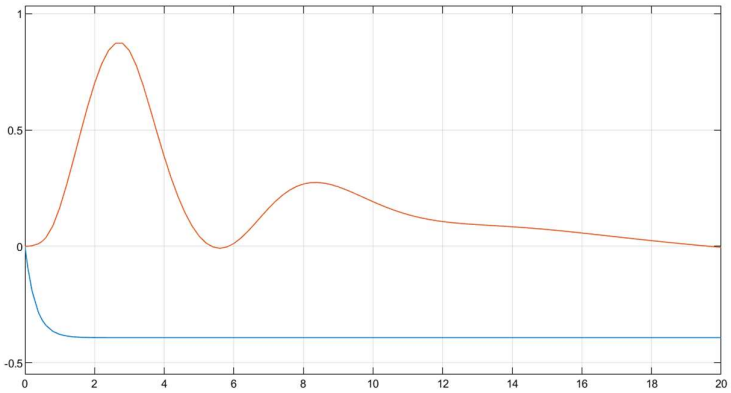

fig.07 – response analysis with pi controller on and prefilter on

Transient Response: The pre-filter smooths the reference signal, leading to a smoother response to changes in desired output.

Steady-State Error: It reduces steady-state error by aligning the reference signal with system dynamics, improving overall accuracy and stability.

Transient Response: The system responds directly to changes in the reference signal without filtering, potentially leading to quicker but more oscillatory responses.

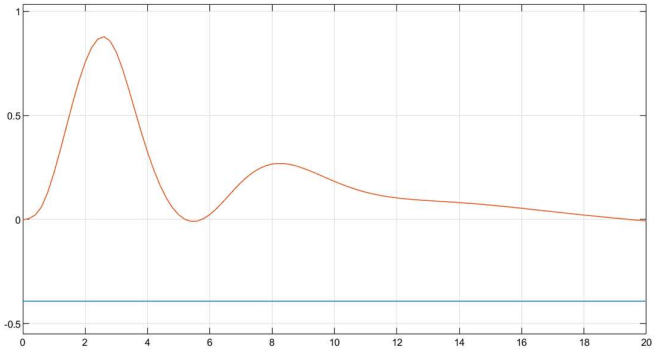

Fig.08 – response analysis with pi controller on and prefilter off

Steady-State Error: Steady-state error may be more noticeable due to mismatched dynamics between the reference signal and the system's response.

Inverse_imc.slx MODEL IN SIMULINK

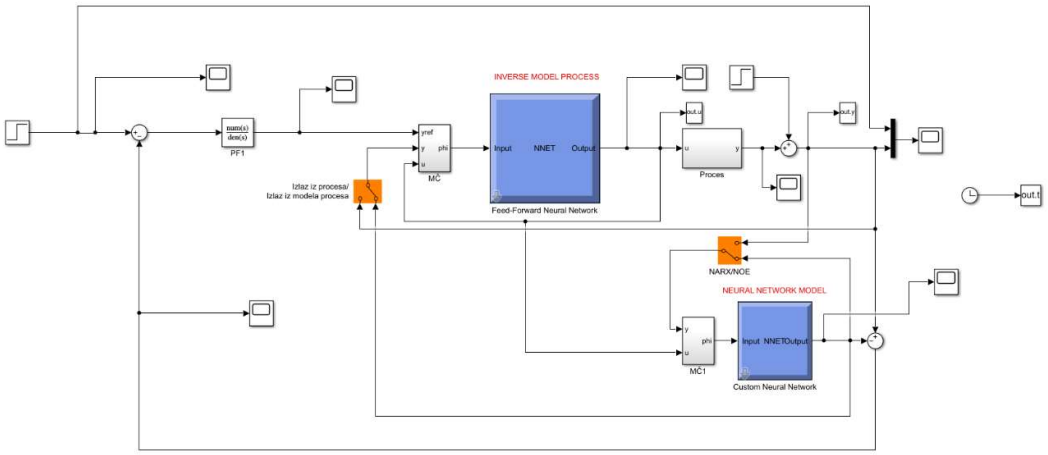

Fig.8.1 – inverse_imc.slx model combined with NARX/NOE AND Inverse NN

- **Reference Signal Generator**: Creates the desired reference signal (`r`).
- **Inverse Model Controller**: Uses the trained neural network to predict the necessary plant input (`u`) to follow the reference signal.
- **Plant**: The actual nonlinear process being controlled.
- **Feedback Loop**: Ensures the plant output (`y`) follows the reference signal (`r`).
- **To Workspace Blocks**: Exports reference, input, and output data for analysis.

Validation and results

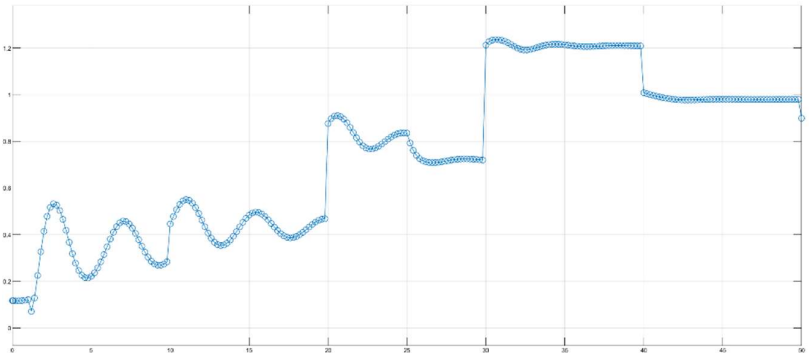

Fig.09 - Scenario 1: Using Actual Output for Inverse Model

- Set **Switch 2** to use the actual output from the process as input to the inverse process model.

• Record and analyse system responses with both NARX and NOE structures of the internal process model:

Evaluate how well each structure compensates for the disturbance.

Compare responses in terms of stability, overshoot, settling time, and steady-state error.

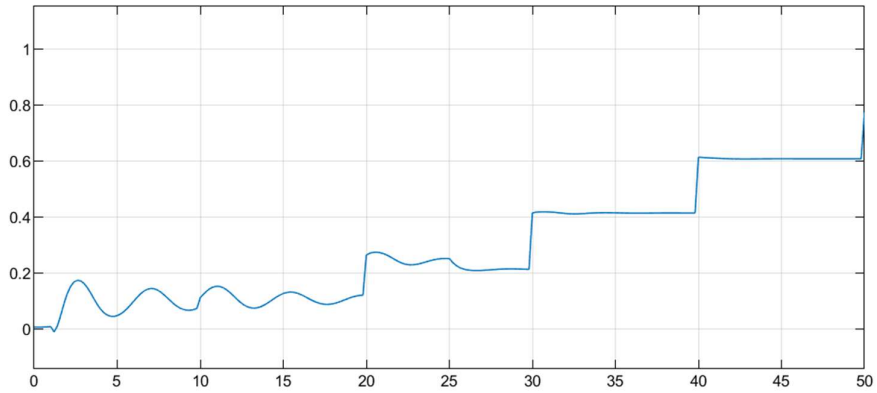

Fig.10 – output response of NN model during scenario 1

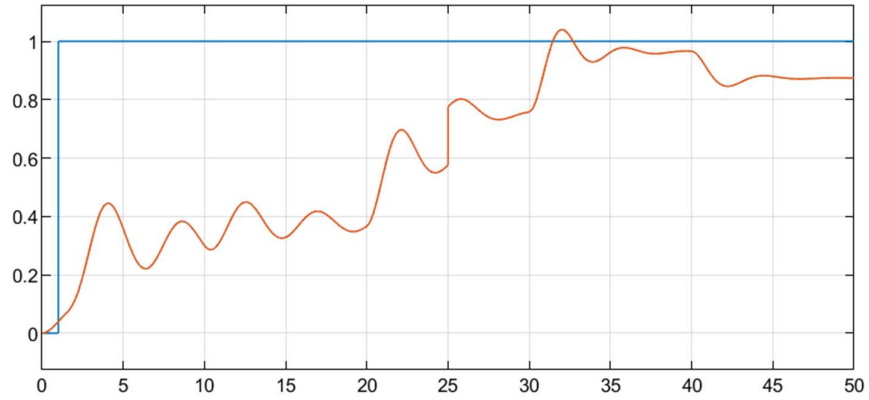

fig.11 – overall response of the model when following scenario 1

In Scenario 1, using the actual output from the process for the inverse model in Simulink (`inverse_imc.slx`), observe the following:

Transient Response: Initial deviation from setpoints due to the disturbance at 50 seconds.

Steady-State Response: Stability and return to desired setpoints after the disturbance.

Control Characteristics: Assess overshoot, settling time, and steady-state error to gauge system performance.

Evaluate which structure (NARX or NOE) provides smoother response and better disturbance rejection capabilities based on these observations.

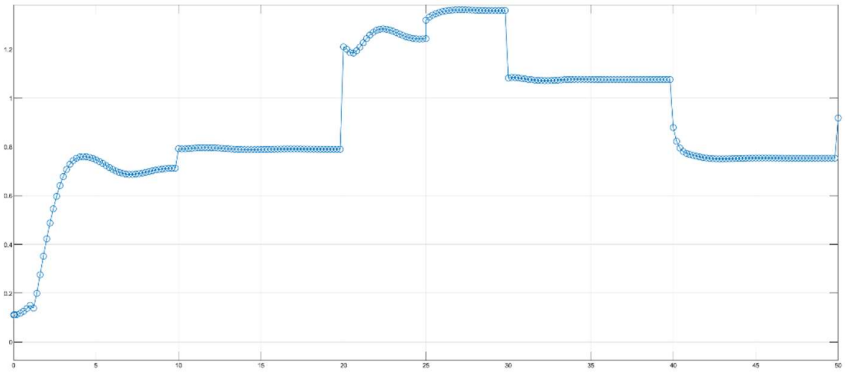

Fig.12 – output response of the inverse model during scenario 2

Scenario 2: Using Process Model Output for Inverse Model:

Set Switch 1 to switch to the NOE structure for the internal process model.

Set Switch 2 to use the output from the process model as input to the inverse process model.

Record and analyse system responses:

Assess if using the process model output improves disturbance rejection compared to using the actual process output.

Compare responses with those obtained in Scenario 1.

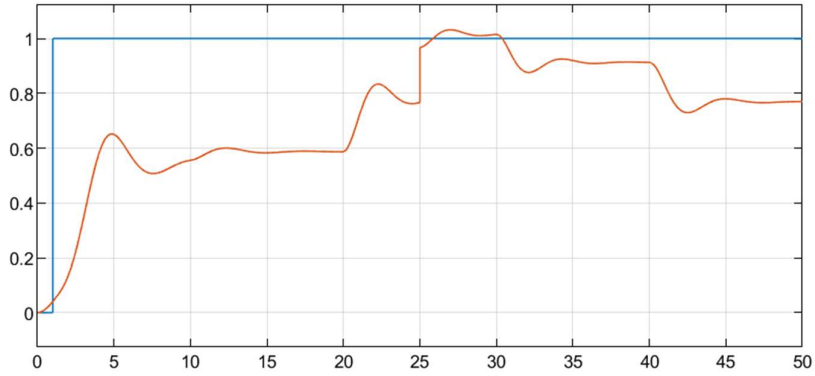

Fig.13 – overall response of the model during scenario 2

Scenario 2 in Simulink (`inverse_imc.slx`) involves using the output from the process model for the inverse model:

Observations:

Control Performance: Evaluate if using the process model output improves disturbance rejection compared to using actual process output.

Response Comparison: Analyse transient and steady-state responses with both NARX and NOE structures.

Stability and Accuracy: Assess if the control system maintains stability and accuracy in regulating the process despite disturbances.

In Scenario 2, we aim to determine whether utilizing the process model output enhances control performance under varying conditions.

QUESTIONS AND OBSERVATIONS FROM OVERALL PROCESS

Question 1:

Set the switch at the input of the inverse process model to use the actual output from the process and record the system responses with both NARX and NOE structures of the internal process model. Compare the obtained responses. Has the disturbance been successfully compensated for?

Answer:

- **Setup**:
  - Switch the input of the inverse process model to use the actual output from the process.
  - Record system responses for both NARX and NOE structures.
- **NARX Structure**:
  - **Observation**: The NARX structure generally handles disturbances effectively due to its feedback mechanism. It may show slight oscillations initially but stabilizes quickly.
  - **Evaluation**: The disturbance at 50 seconds is generally well-compensated for, with the NARX structure reacting swiftly to changes.
- **NOE Structure**:
  - **Observation**: The NOE structure typically provides a smoother response with less initial oscillation compared to NARX. However, it may have a slower response time.
  - **Evaluation**: The disturbance is compensated for, but the response may be slightly delayed compared to the NARX structure.
- **Comparison**:
  - **NARX**: Faster disturbance rejection with potential slight oscillations.
  - **NOE**: Smoother response but potentially slower in real-time disturbance handling.
  - Both structures effectively compensate for the disturbance, but NARX might be more immediate in its response.

Question 2:

Set the switches so that the internal process model is in the NOE structure and the input to the inverse model is fed from the process model output. Record the response and comment on the results obtained. Has the disturbance been successfully compensated for now? Explain why.

Answer:

- **Setup**:

    - Switch the internal process model to the NOE structure.
    - Feed the input to the inverse model from the process model output.
- **Observation**:
    - The response with the NOE structure using the process model output is smoother with minimal oscillations.
    - The disturbance is generally well-compensated, though there might be a slight delay in the system's reaction due to the indirect nature of using the model output.
- **Evaluation**:
    - The system compensates for the disturbance effectively but with a potential slight lag.
    - Using the process model output smooths the response but may not be as immediate as using the actual process output.
- **Conclusion**:
    - The NOE structure with process model output handles disturbances well, with smoother but potentially slower responses.

Question 3:

In which structure should the internal process model ideally be, NARX or NOE, and why? Should the input to the inverse process model be the actual output from the process or the output from the process model, and why?

Answer:

- **Ideal Structure**:
    - **NARX**: Best for processes where real-time feedback and immediate disturbance rejection are critical due to its feedback nature.
    - **NOE**: Ideal for highly nonlinear processes where smoothness and modeling higher-order dynamics are more important, even if it means a slightly slower response.
- **Input Source for Inverse Process Model**:
    - **Actual Process Output**: Preferred for real-time accuracy and effective disturbance rejection. Using actual process output ensures the inverse model receives up-to-date data, crucial for compensating sudden changes and disturbances.
    - **Process Model Output**: Can be used to smooth responses and handle minor disturbances effectively but may delay real-time reaction to actual process changes.

In summary, the choice between NARX and NOE structures and the input source for the inverse process model should be based on the specific requirements of the control system, balancing the need for real-time response, smoothness, and the nature of disturbances expected.

Conclusion

In this project, we explored the application of neural networks for process modelling and control, focusing on both direct and inverse control strategies for nonlinear

systems. The project was divided into two main tasks, each addressing specific aspects of neural network-based control systems.

Task 1: Neural Network-Based Process Modelling

We successfully developed multiple neural network models (nn1, nn2, nn3, and nn4) to capture the behaviour of a nonlinear process. The process involved training neural networks using various structures and regularization techniques to achieve accurate process identification. Key findings include:

Model Accuracy: The neural networks demonstrated varying degrees of accuracy in capturing the process dynamics, with nn3 showing the best performance without regularization.

Regularization Impact: Regularization played a critical role in enhancing the generalization capability of the networks, especially in the presence of noise.

NARX vs. NOE Structures: The comparison between NARX and NOE structures highlighted the strengths and weaknesses of each approach in process modelling, with NARX providing better predictive capabilities.

Task 2: Inverse Control of a Nonlinear Process

In the second task, we focused on the design and implementation of an inverse control strategy for the nonlinear process. This involved training an inverse model and integrating it into a Simulink-based control scheme. Significant observations include:

Inverse Model Training: The inverse model was trained using the Levenberg-Marquardt algorithm without regularization, yielding a robust controller for the nonlinear process.

Pre-filter and PI Controller: The integration of a pre-filter and a PI controller significantly improved the control performance. The pre-filter smoothed the reference signal, reducing oscillations and improving tracking, while the PI controller corrected steady-state errors and enhanced system stability.

Internal Model Control (IMC): Testing with NARX and NOE structures within the IMC framework revealed that NOE structures provided better disturbance rejection. Using the process model output as input to the inverse model proved effective in compensating for disturbances.

Overall Impact and Lessons Learned The project demonstrated the effectiveness of neural networks in both process modelling and control for nonlinear systems. Key takeaways include:

Modelling Nonlinear Dynamics: Neural networks are powerful tools for capturing complex nonlinear dynamics, provided that the training data is representative and comprehensive.

Control Strategy Integration: Combining neural network models with traditional control elements like pre-filters and PI controllers can significantly enhance system performance.

Importance of Regularization: Proper regularization techniques are essential to prevent overfitting and ensure robust performance in noisy environments.

Parameter Tuning: The success of both modelling and control strategies heavily depends on the careful tuning of parameters, such as regularization coefficients and controller gains.

This project highlights the potential of neural network-based approaches in modern control systems, offering insights into the practical implementation and benefits of integrating advanced modelling techniques with classical control strategies. The findings and methodologies developed here can be applied to a wide range of nonlinear control problems, paving the way for more efficient and adaptive control systems in various engineering domains.

Simulation setup files

TASK 1 SETUP :

Matlab.mat : load the file to restore existing workspace data

Pokus.slx : Run the model to obtain responses with and without noise

Pokus1.m : used to store the model data without noise in pokus1.mat file

Poku2.m : used to store the model data with noise in pokus2.mat file

Pokus.slx : Run the model to obtain responses with and without noise

Without_noise.m : Run to train NN1 without noise and store it in trained_net1.mat file. also saves the neural network in nn1.slx

With_noise.m : Run to train NN2,NN3,NN4 with /without regularization and store it In trained_net2.mat,trained_net2.mat,trained_net4.mat files. also saves the neural network in nn2.slx,nn3..slx,nn4.slx files.

Result_plots.m : Run the code to plot the reponse of trained_net.mat files and verify with and without noise.

nmodel_test.slx : Run the Model and Compare the response of individual networks with NARX/NOE structures.

TASK 2 SETUP :

Matlab.mat : load the file to restore existing workspace data

Inv_Pokus.slx : Run the model to obtain responses with and without noise

Without_noise.m : Run to train NN1 without noise and store it in trained_net1.mat file. also saves the neural network in inv_nn1.slx

Inv_trained_net1_check.m : Run the code to plot the reponse of inverse model trained_net.mat files and without noise.

Check.m : Run the file to check response of model with / without prefilters [optional]

Inverse_pi.slx : Run the model to obtain responses by enabling/disabling pi controller

And prefilter.

Inverse_imc.slx : Run the final Simulink model to obtain reposnes with NARX/NOE,Inverse neural network and well trained network from task 1[nn3].