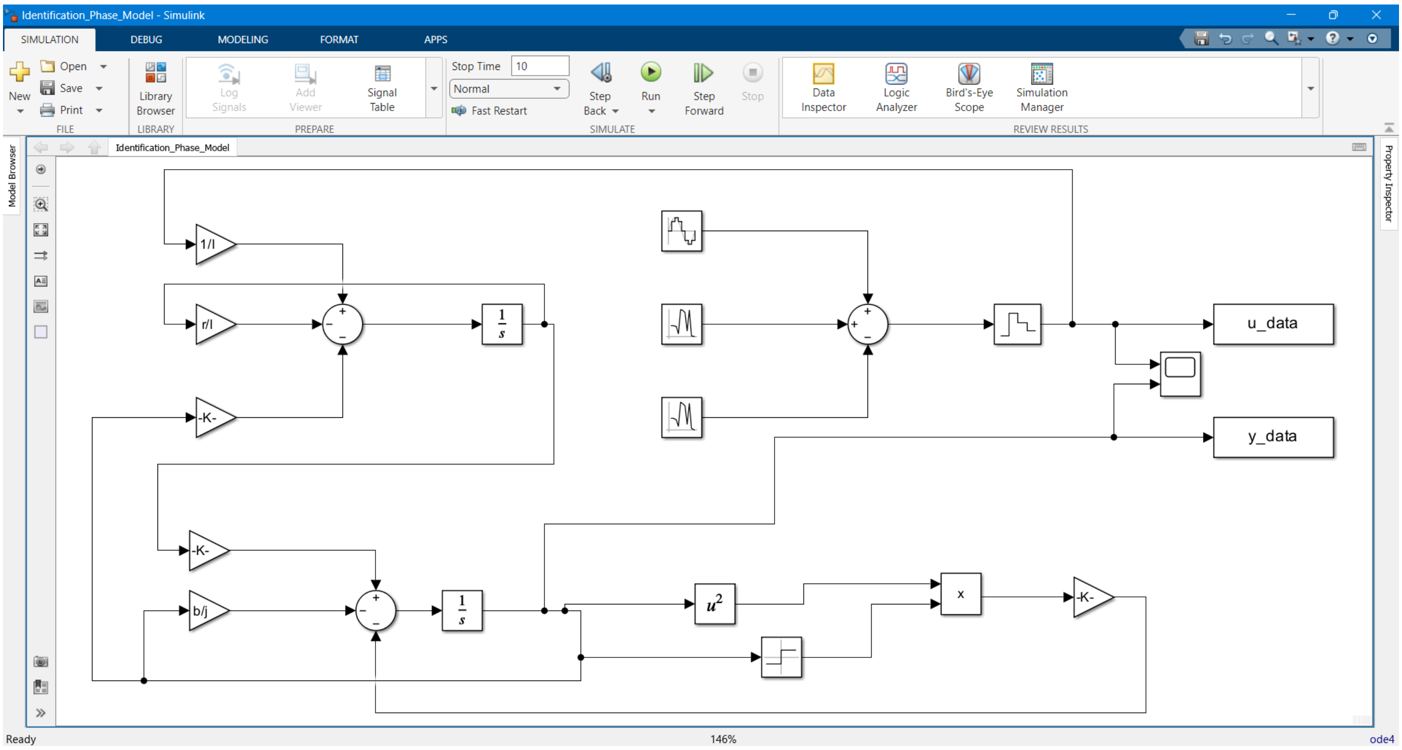
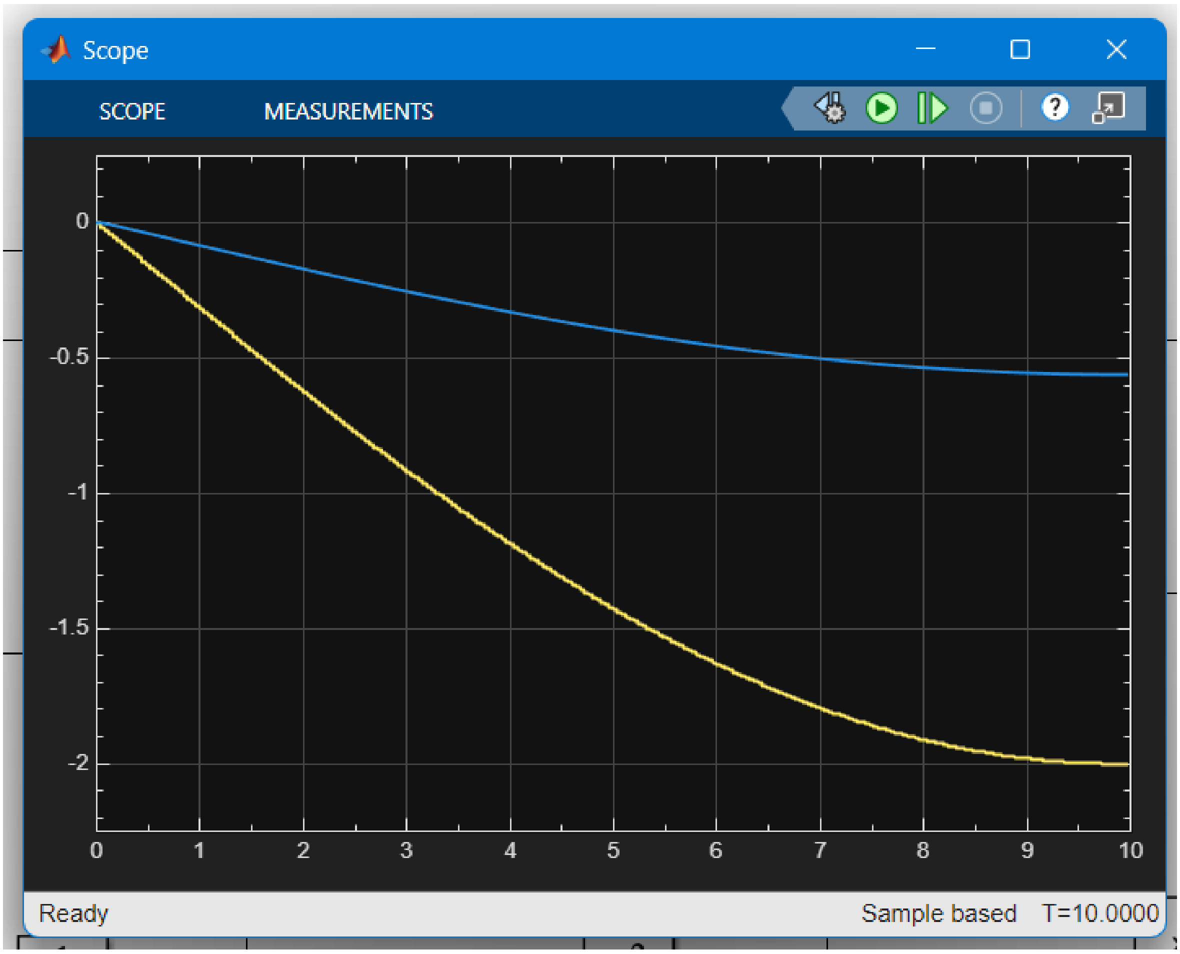
ANN Control of DC Motor with Nonlinear Pump Load

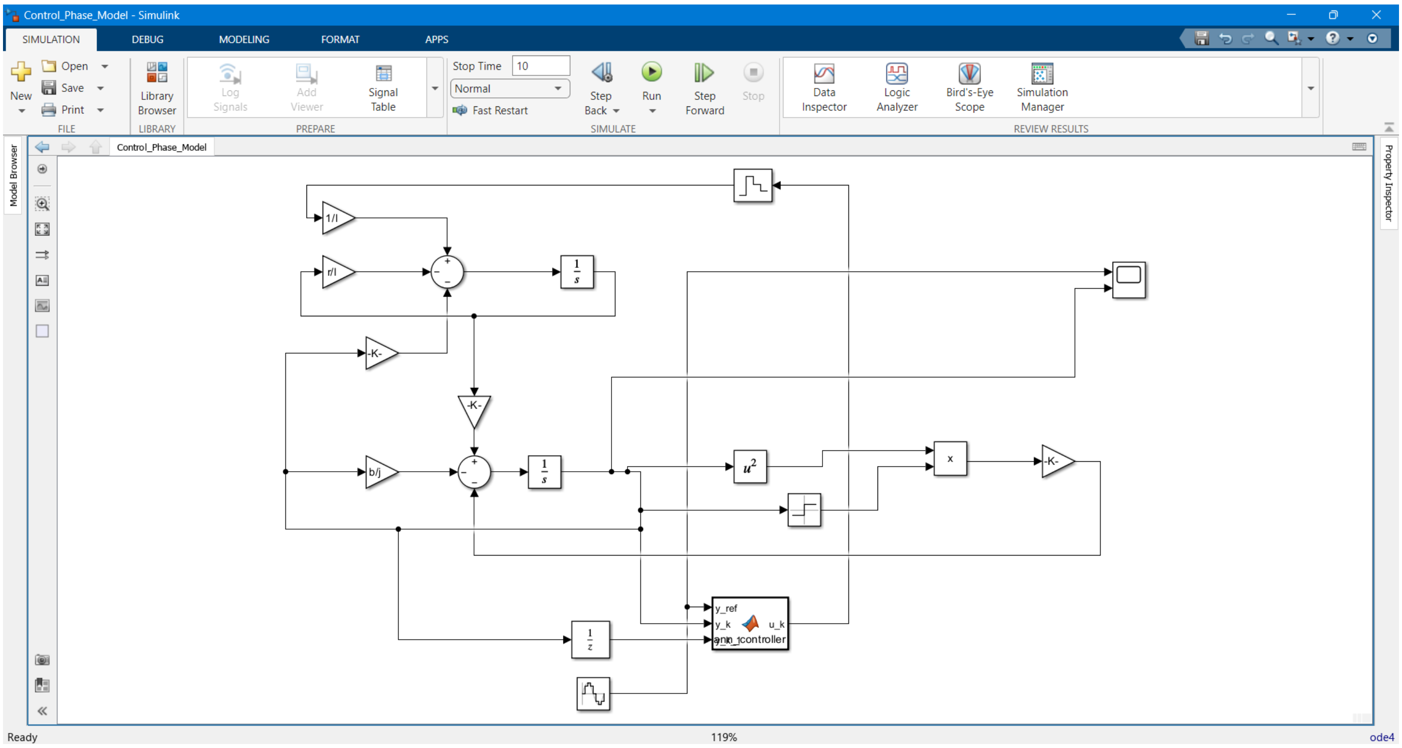
Identification phase model and scope



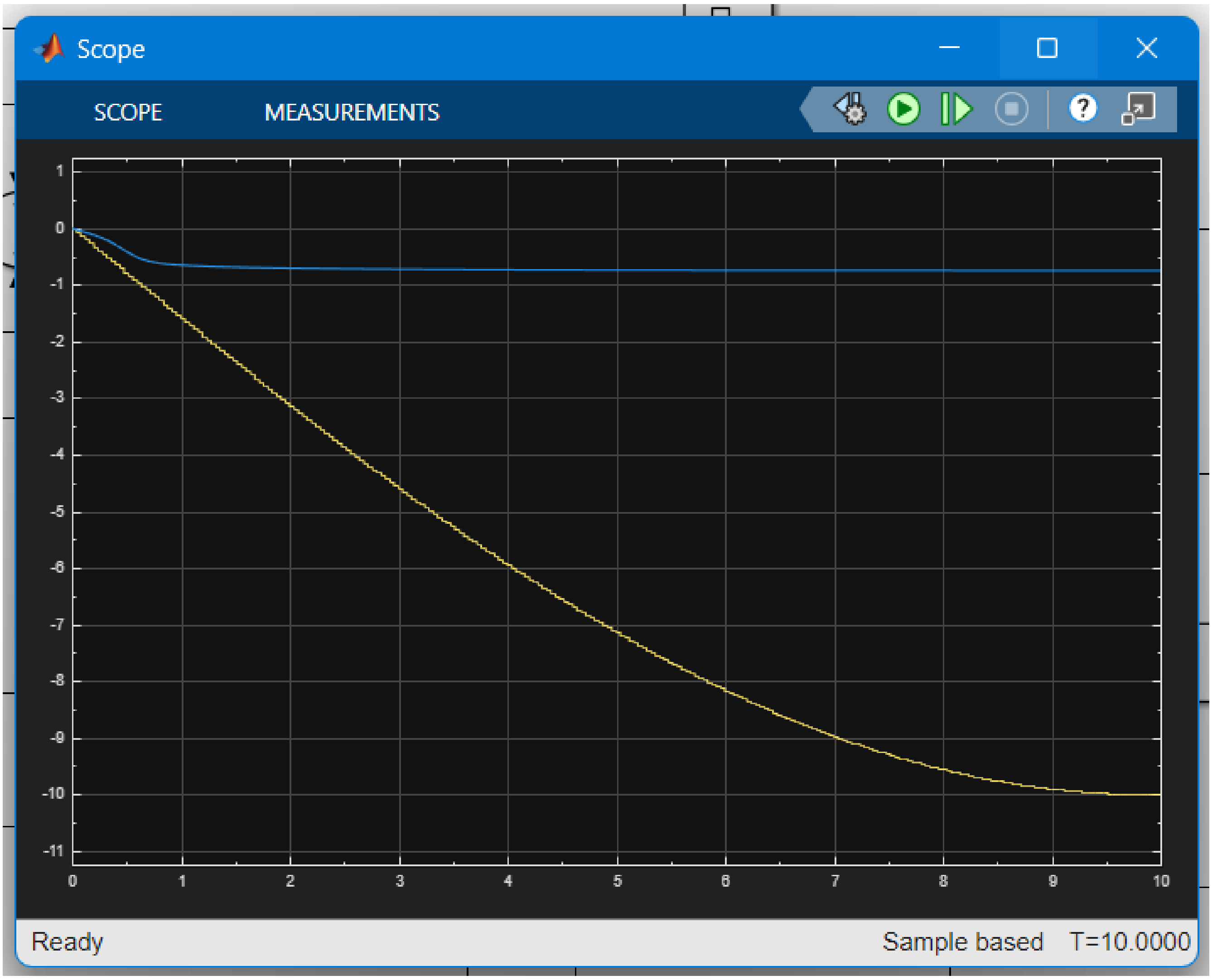
scope



Control phase model and scope



scope



e

M-files

setup_params.m × output_excel.m × +

C:\Users\moata\Documents\MATLAB\setup_params.m

```matlab
% System Parameters
r = 7.56;         % Motor armature resistance (Ohm)
l = 0.055;        % Motor armature inductance (H)
km = 3.475;       % Motor torque constant (N.m/A)
j = 0.068;        % Motor inertia (Kg.m^2)
b = 0.03475;      % Motor friction constant (N.m)/(rad/sec)
mu = 0.0039;      % Motor fan load torque constant (N.m)/(rad/sec)^2
```

setup_params.m × output_excel.m × +

C:\Users\moata\Documents\MATLAB\output_excel.m

```matlab
% --- Step 1: Data Preparation for ANN Training ---
% This script runs after the Simulink model produces 'u_data' and 'y_data'.

% We are training the network: u(k-1) = f(y(k), y(k-1), y(k-2))
%
% So, Inputs = [y(k), y(k-1), y(k-2)]
%    Target = u(k-1)

% Ensure 'y_data' and 'u_data' are column vectors
if size(y_data, 2) > size(y_data, 1)
    y_data = y_data';
end
if size(u_data, 2) > size(u_data, 1)
    u_data = u_data';
end

% Get the total number of samples
N = length(y_data);

% We need to create vectors that align.
% The first valid row we can create is for k=3, which gives:
% y(3), y(2), y(1)  and  u(2)
%
% The last valid row is for k=N, which gives:
% y(N), y(N-1), y(N-2) and u(N-1)

% Create the time-shifted vectors
y_k   = y_data(3:N);     % y(k)
y_k_1 = y_data(2:N-1);   % y(k-1)
y_k_2 = y_data(1:N-2);   % y(k-2)

% Create the target vector
u_k_1 = u_data(2:N-1);   % u(k-1)
```

setup_params.m | output_excel.m

C:\Users\moata\Documents\MATLAB\output_excel.m

```matlab

% Check for length consistency (this should match)
% We have N-2 samples for training
if length(y_k) ~= length(u_k_1)
    error('Vector lengths do not match. Check data simulation.');
end

% Separate into final inputs and targets
ann_inputs = [y_k, y_k_1, y_k_2];    % Columns [y(k), y(k-1), y(k-2)]
ann_targets = u_k_1;                 % Column  [u(k-1)]

disp('ANN training data created successfully.');
disp('Input matrix size (ann_inputs):');
disp(size(ann_inputs));
disp('Target vector size (ann_targets):');
disp(size(ann_targets));


% --- Step 2: Export Data to Excel ---
% Combine inputs and targets for easy export
output_table = array2table([ann_inputs, ann_targets], ...
    'VariableNames', {'y_k', 'y_k_1', 'y_k_2', 'u_k_1'});

filename = 'ann_training_data.xlsx';
writetable(output_table, filename);

fprintf('Training data successfully exported to %s\n', filename);
```

Python Scripts

train_ann_identifier.py

train_ann_identifier.py > ...

```python
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import joblib
import warnings

# Suppress warnings
warnings.filterwarnings('ignore')

print("--- 1. Loading and Preparing Data ---")

# Load the data from Excel
data = pd.read_excel('ann_training_data.xlsx')

# Define features (X) and target (y)
features = ['y_k', 'y_k_1', 'y_k_2']
target = 'u_k_1'
X = data[features]
y = data[target]

print("--- 2. Scaling the Data ---")
scaler = StandardScaler()

# --- THIS IS THE FIX ---
# Fit the scaler on the NumPy array (.values), not the DataFrame (X)
# This trains the scaler WITHOUT feature names.
X_scaled = scaler.fit_transform(X.values)
# ---------------------

# Save the new scaler
scaler_filename = 'ann_identifier_scaler.pkl'
joblib.dump(scaler, scaler_filename)
print(f"Data scaler (trained on NumPy) saved to '{scaler_filename}'")

```

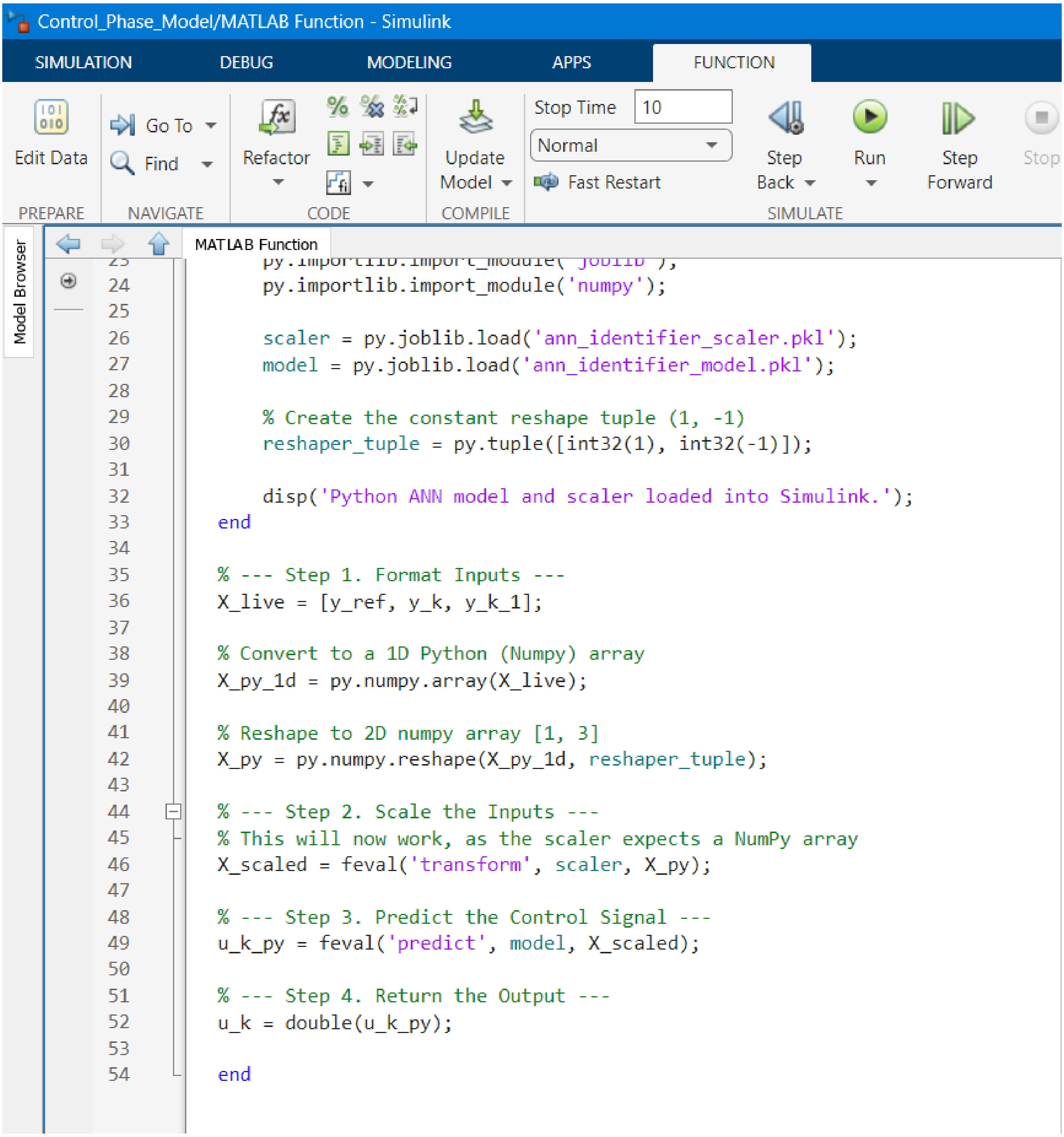
train_ann_identifier.py

```python
y = data[target]

print("--- 2. Scaling the Data ---")
scaler = StandardScaler()

# --- THIS IS THE FIX ---
# Fit the scaler on the NumPy array (.values), not the DataFrame (X)
# This trains the scaler WITHOUT feature names.
X_scaled = scaler.fit_transform(X.values)
# ----------------------

# Save the new scaler
scaler_filename = 'ann_identifier_scaler.pkl'
joblib.dump(scaler, scaler_filename)
print(f"Data scaler (trained on NumPy) saved to '{scaler_filename}'")


print("\n--- 3. Building and Training ANN Identifier ---")
ann_identifier = MLPRegressor(
    hidden_layer_sizes=(10,),
    activation='logistic',
    solver='adam',
    max_iter=1000,
    random_state=42,
    verbose=True
)

print("Starting ANN training...")
ann_identifier.fit(X_scaled, y)
print("Training complete.")

# Save the new model
model_filename = 'ann_identifier_model.pkl'
joblib.dump(ann_identifier, model_filename)
print(f"\nTrained ANN model saved to '{model_filename}'")
```

Control_Phase_Model/MATLAB Function - Simulink

MATLAB Function

```matlab
function u_k = ann_controller(y_ref, y_k, y_k_1)
% This function implements the ANN Controller
% Based on the equation: u(k) = f(y_ref(k), y(k), y(k-1))

% --- Declare Python functions as extrinsic ---
coder.extrinsic('py.importlib.import_module');
coder.extrinsic('py.joblib.load');
coder.extrinsic('py.numpy.array');
coder.extrinsic('py.numpy.reshape');
coder.extrinsic('py.tuple');
coder.extrinsic('int32'); % For creating the reshape tuple
% --------------------------------------------------

% --- Initialize output to define its size and type ---
u_k = 0.0;
% --------------------------------------------------

% Use persistent variables
persistent scaler model reshaper_tuple;

% Load the model on the first step
if isempty(model)
    py.importlib.import_module('joblib');
    py.importlib.import_module('numpy');

    scaler = py.joblib.load('ann_identifier_scaler.pkl');
    model = py.joblib.load('ann_identifier_model.pkl');

    % Create the constant reshape tuple (1, -1)
    reshaper_tuple = py.tuple([int32(1), int32(-1)]);

    disp('Python ANN model and scaler loaded into Simulink.');
end

% --- Step 1. Format Inputs ---
X_live = [y_ref, y_k, y_k_1];
```



معتز أحمد سمير عبدالعاطي 2305223