Modeling And Simulation of CSTR

Group 2: Chang Chen, Yuanyi Huang, Sheena Kapoor, Ambrish Abhijnan

Objectives

- This project aims to model and simulate an ODE system for a Non-Isothermal Continuous Stirred Tank Reactor.
    - Ordinary Differential Equation (ODE) System
        - Transform to Discrete-Time State-Space Representation
        - Step Response Model
    - Machine Learning Simulation
        - Neural networks, Multilayer Perceptron (MLP)

Introduction

- A CSTR is a reaction vessel in which reagents, reactants and often solvents flow into the reactor while the product(s) of the reaction concurrently exit(s) the vessel.
- Model predictive control (MPC) is being applied extensively in the chemical industry and many of these applications are viewed as essential for a number of process control problems.

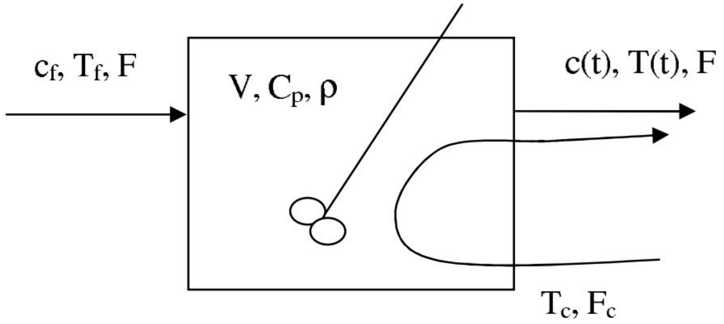


Fig 1. An isothermal CSTR setup

ODE System

Setting up the ODE system:

$$V\frac{dc}{dt} = F(c_f - c(t)) - Vk_{10}exp(-E/RT)c(t), c(0) = c_{init} \tag{1}$$

$$\rho C_p V\frac{dT}{dt} = F\rho C_p(T_f - T(t)) + \Delta H V k_{10}exp(-E/RT)c(t) + U(F_c)A_c(T_c - T), T(0) = T_{init} \tag{2}$$

Combine the constant variables, to obtain reduced form as:

$$\theta = \frac{V}{F}, \alpha u = \frac{UA_c}{\rho C_p V}, y_f = \frac{\rho C_p T_f}{\Delta H} \tag{3}$$

$$n = \frac{E\rho C_p}{R\Delta H}, y_c = \frac{\rho C_p T_c}{\Delta H} \tag{4}$$

ODE System (Cont.)

Redefined the states as:

$$c \leftarrow \frac{c}{c_f}, T \leftarrow \frac{\rho C_p T}{\Delta H} \tag{5}$$

Simplified ODE system:

$$\frac{dc}{dt} = \frac{1 - c(t)}{\theta} - k_{10} exp(-E/RT)c(t), c(0) = c_{init} \tag{6}$$

$$\frac{dT}{dt} = \frac{(y_f - T(t))}{\theta} + k_{10} exp(-E/RT)c(t) + \alpha u(y_c - T), T(0) = T_{init} \tag{7}$$

Discrete-Time State-Space Representation

So that we have the states, inputs and outputs:

$$x = y = \begin{bmatrix} c \\ T \end{bmatrix} \tag{8}$$

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{\theta} \\ u \end{bmatrix} \tag{9}$$

And the ODE system can be rewritten as :

$$\frac{dC}{dt} = (1-c)u_1 - k_{10}\exp(\frac{-n}{T})c \tag{10}$$

$$\frac{dT}{dt} = (y_f - T)u_1 - k_{10}\exp(\frac{-n}{T})c + \alpha u_2(y_c - T) \tag{11}$$

Based on:

$$\dot{x}' = Ax' + Bu' \tag{13}$$

$$y' = Cx' + Du' \tag{14}$$

We have:

$$A_{ij} = \frac{\partial f_i}{\partial x_j}|_{x_s,u_s}, B_{ij} = \frac{\partial f_i}{\partial u_j}|_{x_s,u_s} \tag{15}$$

$$C_{ij} = \frac{\partial g_i}{\partial x_j}|_{x_s,u_s}, D_{ij} = \frac{\partial fg_i}{\partial u_j}|_{x_s,u_s} \tag{16}$$

Discrete-Time State-Space Representation (Cont.)

For result, we calculated A, B, C, and D:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} -0.52972684 & -0.37544061 \\ -0.47972684 & -0.49174061 \end{bmatrix} \quad (21)$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} 9.0560e-1 & 0 \\ -3.8190e-1 & -7.7025e-5 \end{bmatrix} \quad (22)$$

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (23)$$

Afterall, the calculation of ***Φ*** and ***Γ*** can be one-line-code: scipy.signal.cont2discrete

With formulas below,

$$\Phi = \exp A\Delta t \quad (24)$$

$$\Gamma = [\exp A\Delta t - I]A^{-1}B \quad (25)$$

We can find ***Φ*** and ***Γ***:

$$\Phi = \begin{bmatrix} 0.64327742 & -0.23212266 \\ -0.29659944 & 0.66676306 \end{bmatrix}$$

$$\Gamma = \begin{bmatrix} 7.73836547e-1 & 1.05088995e-5 \\ -4.67508473e-1 & -6.24495424e-5 \end{bmatrix} \quad (26)$$

Step Response Model

- The First Step Response models are determined by constructing a unit step input change to process operating steady- state. The coefficients of the models are the outputs at every time step.

$$S = [s_1 \; s_2 \; s_3 \; s_4 \; \ldots\ldots\ldots\ldots \; s_N]^T$$

- The reduced equation is in form as:

$$y(k) = S_N u(k-N) + \sum_{i=1}^{N-1} s_i \Delta u(k-i)$$

- In our code, the y_step term represents each of the output step response for each input variable.
- This complicated calculation can be accomplished by one-line-function `signal.dstep`

Deep Learning Simulation
- Data Generation

- **Features: u_1, u_2**
  - 500 points were picked between the lower bound and upper bound of two inputs.
  - 250,000 combinations in total!
- **Labels: y_1, y_2**
  - Solved by *scipy.integrate.solve_ivp* function
  - Time span: 150 mins
  - Time interval: 1 min
- **Split dataset into random training and testing subsets**
  - *sklearn.model_selection.train_test_split* function.
  - the size of the training set: 200,000.
  - the size of the testing set is 50,000.

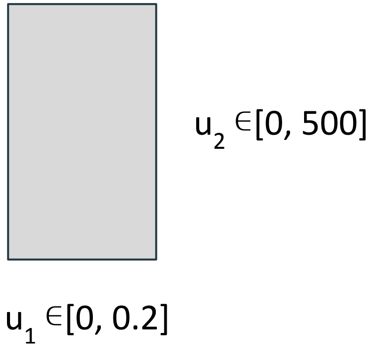


Fig 2. Range for inputs u_1, u_2.

Deep Learning Simulation -Data Processing

- We loaded the training and testing sets by customizing the *Dataset* Class in Pytorch.
- Then, we shuffled the training set and set the batch size by passing the dataset to the *Dataloader* Class in Pytorch.

```
class Data(Dataset):
  def __init__(self, x, y):
    self.x = x
    self.y = y

  def __len__(self):
    return len(self.x)

  def __getitem__(self, index):
    x = self.x[index]
    y = self.y[index]
    return x, y

trainset = Data(x_train, y_train)
trainloader = DataLoader(trainset, shuffle=True, batch_size=128, pin_memory=True)

testset = Data(x_test, y_test)
testloader = DataLoader(testset, shuffle=False, batch_size=128, pin_memory=True)
```

Deep Learning Simulation
-Multilayer Perceptron Model

- Classic neural networks: Multilayer Perceptron (MLP)
    - Input Dimension: 2 (u_1 and u_2)
    - Output Dimension: 300 (y_1 and y_2)
    - Number of hidden layers: 3
    - Number of neurons in each hidden layer: 1024

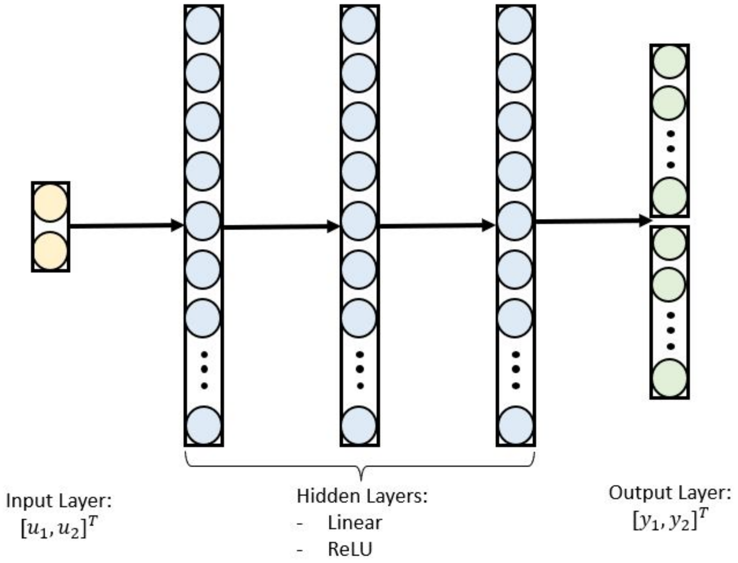


Fig 3. A schematic diagram of the Multi-Layer Perceptron Neural Network.

Deep Learning Simulation -MLP Model (Cont.)

- The objective function to train MLP is the Mean Squared Error Loss

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

- The optimizer to train MLP is Adam
  - Learning rate: 0.001
  - Weight decay: 0.0005

```python
class MLP(nn.Module):
  def __init__(self, in_dim, hidden_dims, out_dim, batchnorm=True):
    super(MLP, self).__init__()
    if batchnorm:
      layers = [nn.Linear(in_dim, hidden_dims[0]),
                nn.BatchNorm1d(hidden_dims[0]),
                nn.ReLU()]
      for i in range(len(hidden_dims)-1):
        layers.append(nn.Linear(hidden_dims[i], hidden_dims[i+1]))
        layers.append(nn.BatchNorm1d(hidden_dims[i+1]))
        layers.append(nn.ReLU())
        # layers.append(nn.Dropout(0.1))
    else:
      layers = [nn.Linear(in_dim, hidden_dims[0]),
                nn.ReLU()]
      for i in range(len(hidden_dims)-1):
        layers.append(nn.Linear(hidden_dims[i], hidden_dims[i+1]))
        layers.append(nn.ReLU())
        # layers.append(nn.Dropout(0.1))
    layers.append(nn.Linear(hidden_dims[-1], out_dim))
    self.nn = nn.Sequential(*layers)

  def forward(self, x):
    out = self.nn(x)
    return out

model = MLP(2, [1024, 1024, 1024], 300)
device = torch.device('cuda')
model.to(device)
```

```
MLP(
  (nn): Sequential(
    (0): Linear(in_features=2, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=1024, out_features=1024, bias=True)
    (4): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): Linear(in_features=1024, out_features=1024, bias=True)
    (7): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU()
    (9): Linear(in_features=1024, out_features=300, bias=True)
  )
)
```

```python
criterion = nn.MSELoss(reduction='none')
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, weight_decay=5e-4)
```

Results
-Concentration and Temperature Profile (Solve_ivp)

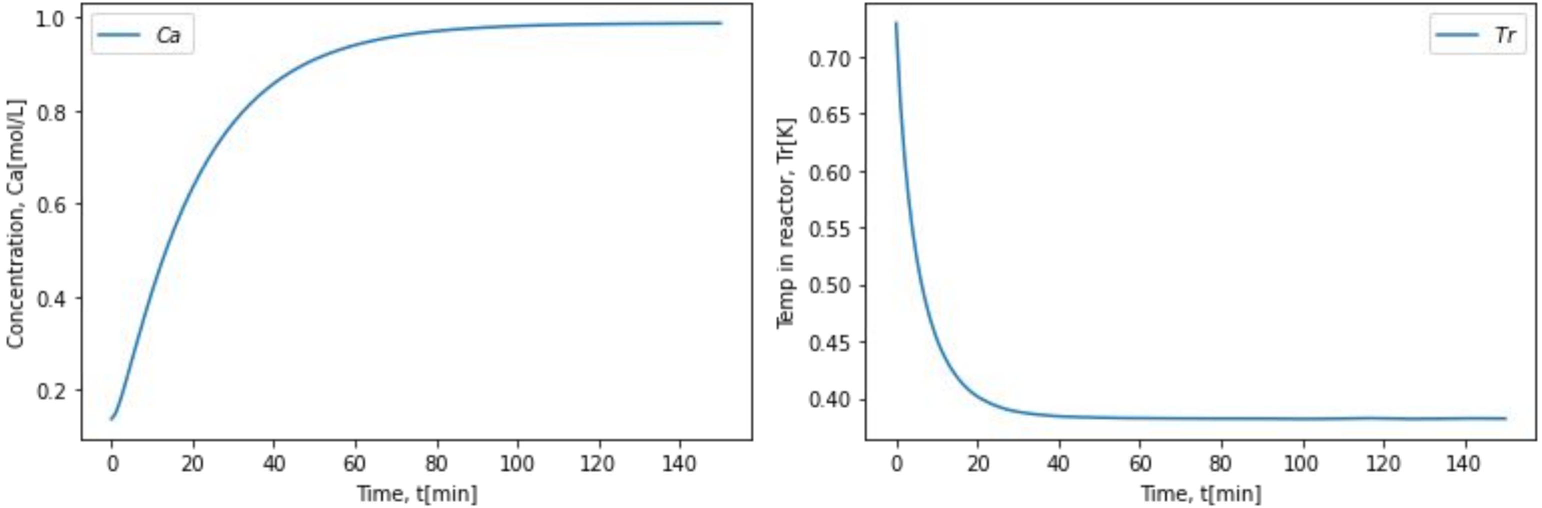
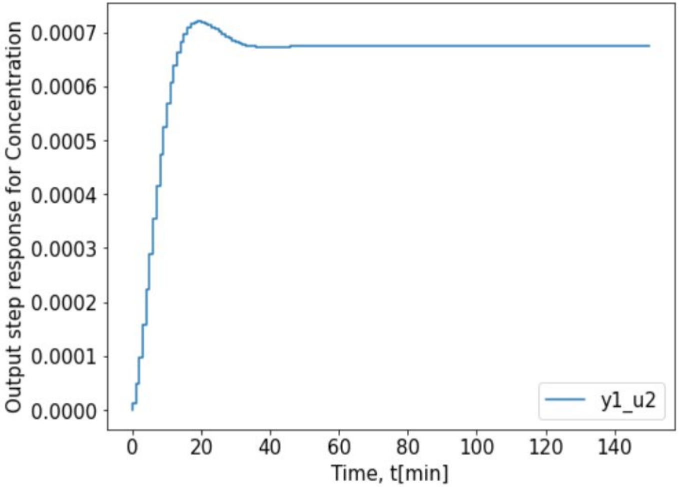


Fig 4. (a) Concentration profile vs time (left) (b) Temperature profile vs time (right)

Results
-Step Response (y_1_u_1, y_1_u_2)



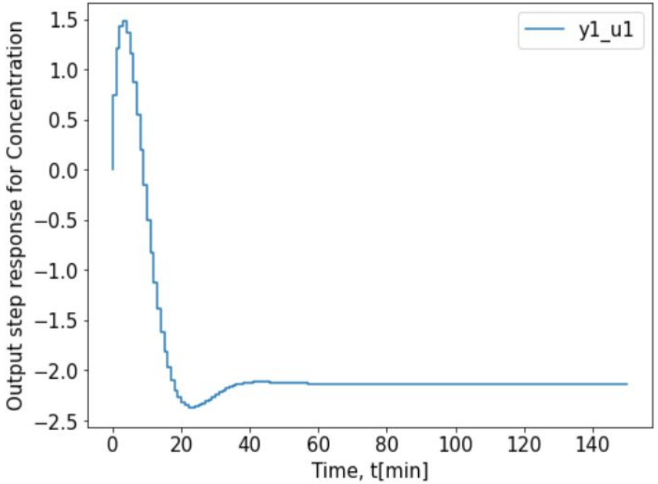


Fig 5. a) Output (y_1, concentration) step response for feed flow rate (u_1) vs time (Left);
b) Output (y_1, concentration) step response for heat transfer rate(u_2) vs time (Right)

Results
-Step Response (y_{2}_u_{1}, y_{2}_u_{2})

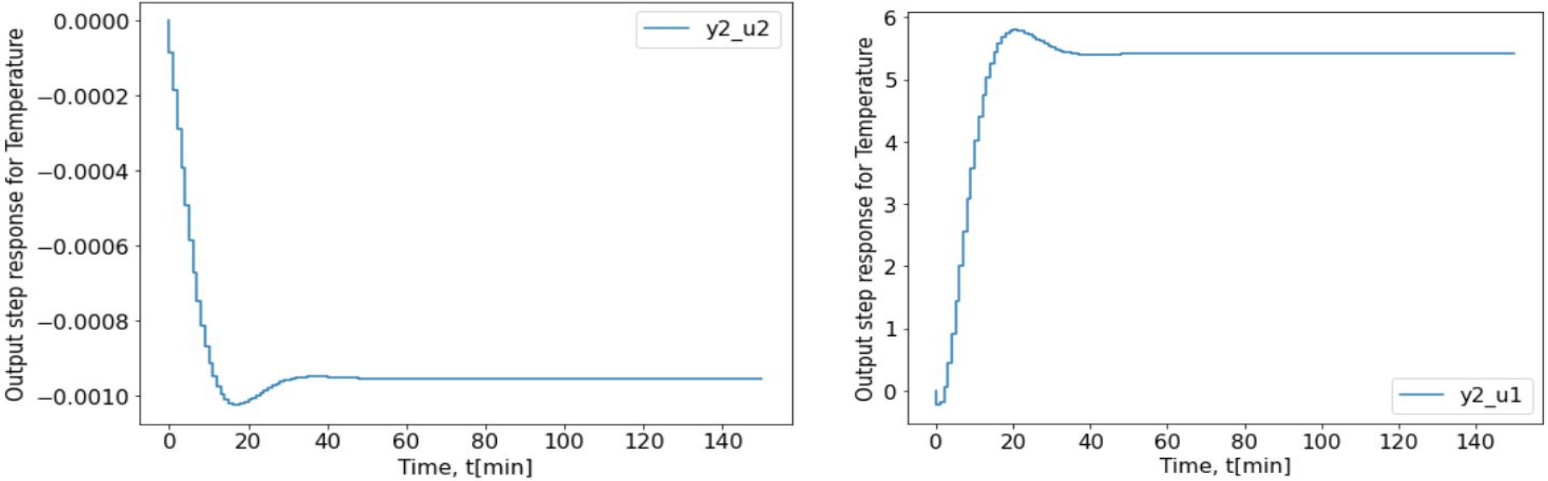


Fig 6. a) Output (y_2, temp) step response for feed flow rate (u_1) vs time (Left);
b) Output (y_2, temp) step response for heat transfer rate (u_2) vs time (Right)

Results

-Deep Learning Simulation for the ODE System

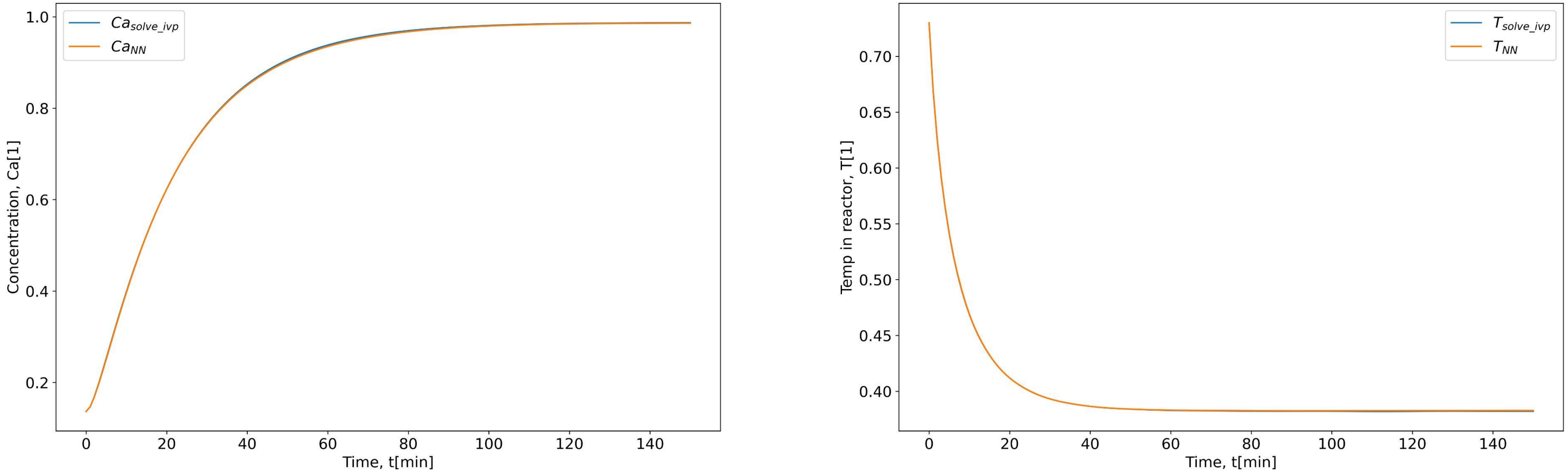


Fig 7. Concentration prediction by MLP for interpolation test (Left);
b) Temperature prediction by MLP for interpolation test (Right)

Results
-Deep Learning Simulation for the ODE System

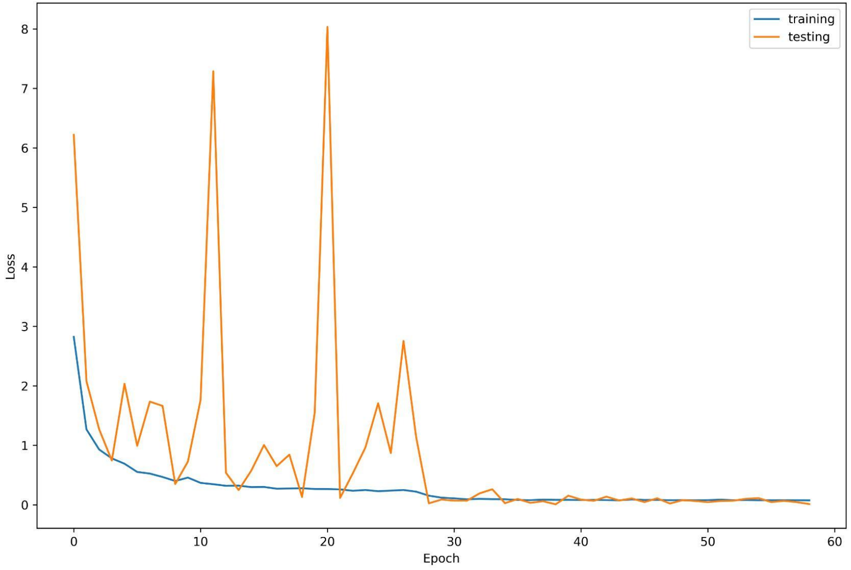


Fig 8. Training and Testing Loss vs Epoch

- Over the testing set, the root mean squared error (RMSE) of concentration at individual time step is 0.0073, which is 0.8 % of the average concentration (0.8603) at individual time step.
- The (RMSE of temperature at individual time step is 0.0021, which is 0.5 % of the average temperature (0.4002) at individual time step.

Great Results!

Reference

[1] AI vs. Machine Learning vs. Deep Learning vs. neural networks: What's the difference? IBM. (n.d.). Retrieved April 25, 2022, from https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks
[2] Mettler-Toledo International Inc. all rights reserved. (2022, February 8). Mettler Toledo Balances & Scales for Industry, lab, retail. METTLER TOLEDO Balances & Scales for Industry, Lab, Retail - METTLER TOLEDO. Retrieved April 24, 2022, from https://www.mt.com/us/en/home/products/L1 AutochemProducts/Chemical-Synthesis-and-Process-Development-Lab-Reactors/continuous-stirred-tank-reactors-cstr.html
[3] Biegler, L. T. (n.d.). Computers and Chemical Engineering - cepac.cheme.cmu.edu. Retrieved April 24, 2022, from http://cepac.cheme.cmu.edu/pasi2011/library/biegler/CACE_3678.pdf
[4] Tlacuhua, A.F.; Moreno, S.T.; Biegler, L.T. Global Optimization of Highly Nonlinear Dynamic Systems. Ind. Chem.Res 2008, 47, 2643 - 2655.
[5] Hicks, J; Mohan,A; Ray,W.A. The Optimal Control of Polymerization Reactors. The Canadian Jouranl of Chemical Engineering 1969, Vol. 47, 590 -596.
[6] Salahshour,E; Malekzadeh,M; Gordillo,F; Ghasemi,J. Quantum neural network-based intelligent controller design for CSTR using particle swarm optimization algorithm. Transaction of the Institute of Measurement and Control 2019, 41, 392-404.
[7] Shrivastava,P; Modeling and Control of CSTR using Model based Neural networks Predictive Control
[8] Shyamalagowri,M;Rajeswari,R. Neural Network Controller based Nonlinearity Identification Case Study: Nonlinear Process Reactor- CSTR. Advanced Material Research 2016, 985, 1326- 1334.
[9] Putrus,k.M. Implementation of Neural Control for Continuous Stirred Tank Reactor (CSTR), Al- Khwarizmin Engineering Journal 2011, Vol. 7, 39-55.
[10] Patterson D. W; Artificial Neural Networks Theory and Applications. Prentice Hall 1996, 220.

Thank you