Laboratory 1 & 2: Essential Oil Extractor Plant Modelling

Group 59 - Goodwill Gwala (674390), Thandi Magoro (1707061), 1608789 (Nomthandazo Magwaza)
ELEN3016 Control I, 17 September 2021
The University of the Witwatersrand, Johannesburg, South Africa
School of Electrical and Information Engineering

***Abstract*— The implementation of the designed plant system was a success. Due to the delay in the equation derived, the ordinary differential equation changes to be a delayed differential equation. Thus the solution to this equation is transcendental and infinity, with no well behaved closed form. Therefore for the delayed differential equation backward Euler numerical method is used. Simulink simulation results obtained at initial condition matches those obtained from `Matlab`.**

I. Introduction

This report documents the laboratory exercise that investigates the performance of a preliminary Essential Oil Extractor plant. A mathematical model of the negative feedback plant system using delayed differential equations and backward Euler integration is derived. The model is simulated in `Matlab`, the native `Matlab` code is verified using `Simulink`. Gain scheduling is employed in implementing a simple gain control such that the plant temperature does not exceed the threshold temperature. The implemented delay is contrasted with 1^{st} and 2^{nd} Order Padé approximation and their performances verified using a comparative analysis.

II. System Overview

Figure 1 illustrates the system model under investigation presented in block-diagram form.

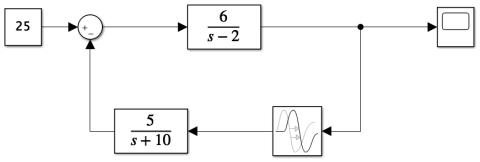

Fig. 1. System Overview [Simulink]

Using the general closed loop equation and some algebraic simplification, the system transfer function is illustrated by equation (1).

$$H(s) = \frac{6s + 60}{s^2 + 8s - 20 + 30e^{-0.2s}} \tag{1}$$

III. Mathematical Model

The system of differential equations for the system presented above can be obtained by representing the transfer function in state space.

Using state variables $z = x_1$, $\frac{dz}{dt} = x_2 = \dot{x_1}$ and $\frac{d^2z}{dt^2} = \dot{x_2}$, the differential equations are as follows.

$$\begin{gathered} \dot{x_1} = [0]x_1 + [1]x_2, \\ \dot{x_2} = [20]x_1 + [-30]x_1(t - 0.2) + [-8]x_2 + u, \\ y = [60]x_1 + [6]x_2 \end{gathered} \tag{2}$$

IV. Backward Euler Model

The backward Euler method is a rudimentary derivative from first principles that approximates a function using previous values, this is represented as follows:

$$x_{n+1} = x_n + \Delta t * f(t_{n+1}, x_{n+1}) \tag{3}$$

Substituting the first equation of (2) into (3) yields,

$$x_1[n+1] = x_1[n] + \Delta t * x_2[n+1] \tag{4}$$

Similarly for the second of (2) differential equation,

$$\begin{gathered} x_2[n+1] = x_2[n] + [20\Delta t]x_1[n+1] + \\ [-30 * \Delta t]x_1][n - m + 1] + [-8\Delta t]x_2[n+1] + u\Delta t \end{gathered} \tag{5}$$

Where m is the constant delay split into sub-multiple step-sizes such that it is an integer interval of said delay. Substituting (4) into (5),

$$x_2[n+1] = [\Omega_0]x_1[n+1] + [-\Omega_0]x_1[n] \tag{6}$$

Subsequently, substituting (5) into (6) and isolating $x_1[n+1]$,

$$\begin{gathered} x_1[n+1] = A_{11}x_1[n] + A_{12}x_1[n - m + 1] + \\ A_{13}x_2[n] + B_1 * u \end{gathered} \tag{7}$$

gives the first state equation. Similarly, back substituting (7) gives the second state equation,

$$\begin{gathered} x_2[n+1] = A_{21}x_1[n] + A_{22}x_1[n - m + 1] + \\ A_{23}x_2[n] + B_2 * u \end{gathered} \tag{8}$$

The coefficients are explicitly defined in the `Matlab` code, Appendix A.

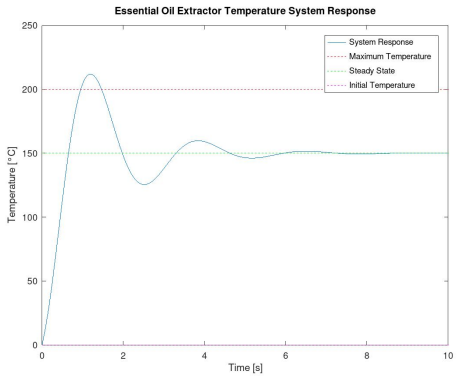

Fig. 2. System Response

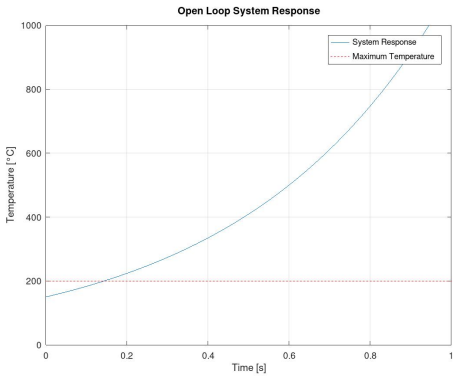

Fig. 4. Open Loop Response

V. Simulation Results

The simulation from the implemented model is illustrated below.

The native `Matlab` implementation and the reference model exhibit the same behaviour , this is a verification of the native implementation. The plant exceeds the 200 °C threshold during the transient. The combination of gain and initial conditions are investigated. Small change in the initial temperature has drastic effects on the response, this is attributed to non-linearity. Gain scheduling dictates that the system is to be compensated for $x_1(0)$ & $x_1(3.34)$. The former case can be easily achieved with $K_p \approx 1.1$. The controller fails to adequately compensate the system at $x_1 \geq 3.34$. The required gain-control scheme is realisable using a switch mechanism. The compensated response is illustrated below.

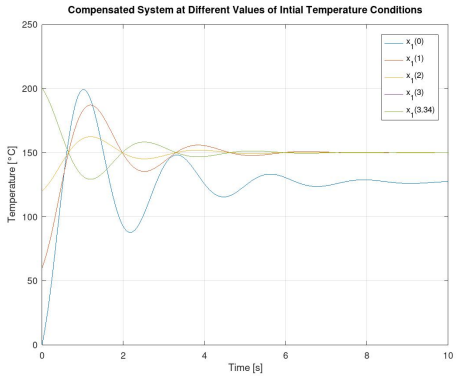

Fig. 3. Compensated System at Different Initial Conditions

A. Open Loop vs Closed Loop Performance

The open-loop differential equation is obtained using the same methodology previously described to obtain,

$$x[n+1] = [\frac{1}{1-2\Delta t}]x[n] + [\frac{6\Delta t}{1-2\Delta t}] \tag{9}$$

The open-loop system response is presented below,

As expected, the response is unstable. This is due to the RHS pole of the plant. The addition of the feedback path facilitates pole cancellation and ensures stability.

VI. Pade Approximations

A Pade approximation is a rational function that realises delays in physical dynamic systems. The Pade approximation essentially linearises the system which lends itself to linear methods using state space computation. Using (3), the system of equation is represented as follows:

$$\begin{aligned} \tilde{X}_{n+1} &= [I - \Delta tA]^{-1}\tilde{X}_n + [I - \Delta tA]^{-1}\Delta tB\tilde{U} \\ \tilde{Y} &= C\tilde{X} + D \end{aligned} \tag{10}$$

The Pade equations are substituted into the plant transfer function, replacing the original delay term. The coefficients of the newly obtained function serve as matrix entries in (10) The simulated response is demonstrated below.

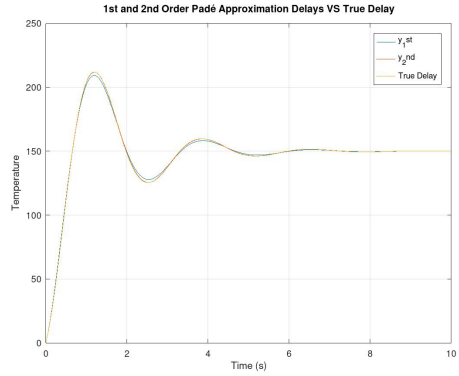

Fig. 5. Open Loop Response

There are minor discrepancies observed. A higher order approximation tends to improve accuracy of the delay.

References

[1] R.Burns. Advanced control engineering. Butterworth-Heinemann. 2001

APPENDIX A

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%AUTHORS : Group 59; %LAB 1 & 2 %Acknowledgements: Our sincerest gratitude to Mubanga Mofu for his guidance
%during the course of the completion of this lab exercise.
clc;
clear all;
close all;
%%%%%%%%%%%%%%%%%%%%%%%% Variable Declaration %%%%%%%%%%
tau_system = 2; T= 50*tau_system; tF(1) = 0;
int_val = 50; tau = 0.2; dt_sub = tau/int_val; u = 25; m = tau/dt_sub; x0 = [0 ; 0]; %initial conditions
X( : ,1) = x0; X_OL( :,1) = x0; X_GC( :,1) = x0;
if ~isinf(m) && floor(m) == m %ensure that dt_sub is an integer else use dt = 0.2
    dt = dt_sub;
else
    dt = tau; m = tau/dt;
end
%define the differential consant variables
P0 = 1/(dt); P1 = 1+8*dt; P2 = P0-((20*dt)/(P1));
A11 = P0/P2; A12 = (-30*dt)/(P1*P2); A13 = 1/(P1*P2); B_1  = dt/(P1*P2);
A21 = (A11*P0)-P0; A22 = A12*P0; A23 = A13*P0; B_2  = B_1*P0;
%Gain Scheduling
switch x0(1)
    case 0
        Kp = 1.1;
    otherwise
        Kp = 1;
end
for n=1:T/dt
    tF(n+1) = n*dt;
    if n-m+1<1 %ensure that the system is causal
        m_del = 1;
    else
        m_del = n-m+1;
    end
    %Closed Loop
    X(1,n+1) = A11*X(1,n) + A12*X(1,m_del) + A13*X(2,n) + B_1*u;
    X(2,n+1) = A21*X(1,n) + A22*X(1,m_del) + A23*X(2,n) + B_2*u;
    %Open Loop
    X_OL(n+1) = (1/(1-2*dt)) * X_OL(n) + (6*dt)/(1-2*dt)*u;
    %Gain Controller
    X_GC(1,n+1) = A11*X_GC(1,n) + Kp*A12*X_GC(1,m_del) + A13*X_GC(2,n) + B_1*u;
    X_GC(2,n+1) = A21*X_GC(1,n) + Kp*A22*X_GC(1,m_del) + A23*X_GC(2,n) + B_2*u;
end
%Closed Loop Output
Y = 60.*X(1,:) + 6.*X(2,:);
%Open Loop Output
Y_OL = 2.*X_OL +6*u;
%Compensated KP Controller Output
Y_KP = Kp*(60.*X_GC(1,:) + 6.*X_GC(2,:)); Y_Y_KP = [Y ; Y_KP];
%PADE APPROX
p = 2/tau;
%1st Order Coeffs
a3 = 1; a2 = 8+p; a1 = 8*p-50; a0 = 10*p; b3 = 0; b2 = 6; b1 = 60+6*p; b0 = 60*p;
%Transfer Function with Pade Approximation delay
b = [b3 b2 b1 b0]; a = [a3 a2 a1 a0];
x0_PD1 = [0;0;0]; %Pade 1st Order Initial Conditions
[A1,B1,C1,D1] = tf2ss(b,a); %Transform to State Space
X_PD1( :,1) = x0_PD1; % Initialise with Pade I.Cs
T_PD1(1) = 0; %Time
%2nd Order Coeffs
b4 = 0; b3 = 6;  b2 = 60+18*p;  b1 = 18*p^2+180*p; b0 = 180*p^2;
a4 = 1; a3 = 3*p+8; a2 = 3*p^2+24*p+10;  a1 = 24*p^2-150*p; a0 = 30*p^2;
b = [b4 b3 b2 b1 b0]; a = [a4 a3 a2 a1 a0];
x0_PD2 = [0;0;0;0]; %Pade 2nd Order Initial Conditions
[A2,B2,C2,D2] = tf2ss(b,a); %Transform to State Space
X_PD2( :,1) = x0_PD2; % Initialise with PD I.Cs
T_PD2(1) = 0; % Time
%Backward Euler for Pade Approximation
for n=1:T/dt
    T_PD1(n+1) = n*dt; COEF1 = inv(eye(3)-dt*A1); COEF2 = inv(eye(4)-dt*A2);
    X_PD1( :,n+1) = COEF1*X_PD1( :,n) + COEF1*dt*B1*u; %1st Order Pade
    X_PD2( :,n+1) = COEF2*X_PD2( :,n) + COEF2*dt*B2*u; %2nd Order Pade
end
% Padé Outputs
Y_PD1 = C1*X_PD1  + D1*u; Y_PD2 = C2*X_PD2 + D2*u; Y_OUT_PD = [Y_PD1; Y_PD2];
```