5ELEN018W - Tutorial 6 Exercises: Revisiting Transformations - Bode Plots in Simulink - Writing Code for Simulations

1 Homogeneous Matrix Transformations

Consider the following homogeneous transformation matrix:

$$\begin{pmatrix} -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & -1 & 3 \\ 0 & 0 & 0 & 1.0000 \end{pmatrix} \tag{1}$$

Calculate the inverse transformation. Do this on paper with the usage of a calculator if needed. Do not use software such as Matlab.

Hint: Look at lecture 3 and at the last slide for a quick calculation of the inverse.

2 Bode Plots in Simulink

Consider a dynamic system describing a watertank (Figure 1). Water enters the tank from the top at a rate proportional to the voltage, V, applied to the pump. The water leaves through an opening in the tank base at a rate that is proportional to the square root of the water height, H, in the tank. The presence of the square root in the water flow rate results in a nonlinear plant.

The dynamic system is described by the following differential equation:

$$\frac{d}{dt}Vol = A\frac{dH}{dt} = bV - \alpha\sqrt{H} \tag{2}$$

where H is the height of the water in the tank, Vol is the volume of the water in the tank, b is a constant related to the flow rate into the tank, A relates to the area of the tank and α is a constant related to the flow rate out of the tank.

The following values should be used: $A = 20, \alpha = 2, b = 5, H_{ref} = 10$, where H_{ref} is the desired water level in the tank.

The Simulink model with the PID controller can be opened with the following commands in Matlab:

```
model = 'watertank';
open_system(model)
```

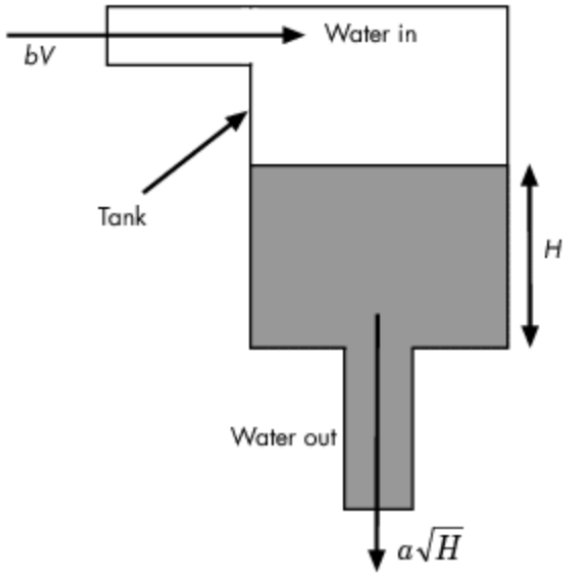

Figure 1: The watertank dynamic system.

We will see how to Simulink can linearise the model and then use the linearised model to plot the Bode diagram for the chosen input-output system.

This tutorial will be recorded so that you can watch the video again and repeat similar steps for any dynamic system for a chosen input-output system.

3 Creating Transfer Functions in Matlab

The transfer function of a dynamic system can be created in Matlab by using similar code as the one described in the last lecture (Slides 7–8).

Implement in Matlab the following transfer function:

$$\frac{5s + 15}{77s^3 + 4s^2 + 1000s + 10} \tag{3}$$

4 Drawing Bode Plots in Matlab

1. Implement in Matlab the following transfer function:

$$\frac{0.5s + 5}{0.0002s^4 + 0.0064s^3 + 0.512s^2 + s} \tag{4}$$

2. Draw the Bode plot in Matlab by calling the `bode` function and passing it a single argument corresponding to a variable which is assigned to the transfer function in the previous step.

3. Compare your diagram with the one given in slide 11 of the lecture. Do they match?

4. Is the system stable or unstable? Justify your answer.

5 Implementing a Dynamic System in Code (not Simulink)

Consider the cruise control system from the previous tutorial:

$$m\dot{v} + bv = u \tag{5}$$

where v is the speed of the car, u is the control action and b is the damping coefficient due to friction.

For the purposes of this exercise, the following values should be used for the system: $m = 1000, b = 50$. The desired (reference) speed is $v_{ref} = 10$.

Implement in Java a PID controller to bring the system to the desired speed. Use $t = 10.0$ as the total simulation time. The parameters of the PID controller are: $K_p = 800, K_i = 0, K_d = 40$ (i.e. this is a PD controller).

The next page lists the code solution to it, the same code can be downloaded from the following link:

`https://dracopd.users.ecs.westminster.ac.uk/DOCUM/courses/5elen018w/CruiseControl.java`

```
import java.io.*;

class CruiseControl {
    static double v = 0;  // current speed initialised to 0
    static double previous_v = 0;  // the speed at the previous time step
    static double dt = 0.001;  // time step for the simulation


    /* Implementents the dynamic system (plant) - system input is action u
       and the method returns the output of the plant */
    static double plant(double action_u) {
        // m \dot{v} + b v  = u
        // £m=1000, b=50, u = 500£
        double m = 1000.0;
        int b = 50;

        double v_dot = (action_u - b*v)/m;

        double new_speed = v + v_dot*dt;
        return new_speed;
    }


    public static void main(String[] args) {
        PrintWriter pw = null;

        try {
            pw = new PrintWriter("myfile.txt");
        }
        catch(Exception ex) {
            ex.printStackTrace();
        }

        double start_time = 0;
        double end_time = 10.0;

        double current_time = start_time;

        double v_ref = 10;  // the desired speed

        int K_p = 800;  // proportional gain
        int K_i = 0;    // integral gain
        int K_d = 40;   // derivative gain

        double previous_error = 0;
        double integral = 0;


        /* simulate the system operation from the beginning till
```

```
            the end of the simulation */
        while (current_time <= end_time) {
            double error = v_ref - v;

            // I(ntegral) component of the PID controller
            integral = integral + error*dt;

            // D(erivative) component of the PID controller
            double deriv = (error - previous_error)/dt;

            // the output (action) of the PID controller
            double action = K_p*error + K_i*integral + K_d*deriv;

            // remember the last error when the previous action
            // was applied to the plant
            previous_error = error;

            // apply the new action to the plant to calculate the new (current) speed
            v = plant(action);

            System.out.println("Time: " + current_time + " Action: " +
                               action + ", Speed=" + v);
            pw.println(v + " " + current_time);

            // advance the time
            current_time += dt;
        }

        pw.close();
    }
}
```