Session 4
Modeling and Digital Simulation Case Studies

Purpose

One of the objectives of this session is to get you acquainted with the basics of
SIMULINK, which is a graphical modeling, simulation, and prototyping environment
used extensively in industry. We will not be able to cover the vast capability of
SIMULINK with few examples, and you are expected to explore various features and
graphical programming techniques of SIMULINK on your own.   The other objectives of
this lab are to find the mathematical model for some basic physical systems, to obtain a
digital simulation diagram for the resulting differential equations, and to obtain the
system's step response and investigate the effect of damping on the system response.

Reference Computational Aids in Control Systems Using MATLAB, H. Saadat,
McGraw-Hill 1993. An updated version is on Saadat's website for EECS
students. The password is available from your instructor.

Introduction

The dynamic performance of physical systems is obtained by utilizing the related
physical laws governing the systems. Many dynamic systems contain energy storage
elements such as masses and springs in the mechanical system, or inductors and
capacitors in an electric circuit. Because of the principle of conservation of energy,
instantaneous changes in system variables are not possible. Therefore, the system
variables will go through some transients before settling to their steady-state values.

All physical systems are nonlinear to some extent. In order to model the system with
linear time-invariant differential equations for transfer function and state space model, the
system must first be linearized, or its range of operation be confined to a linear range.

The next step in designing a practical control system is to simulate the model on a
computer to obtain the system response to various signals and disturbances. Next,
introduce appropriate controllers to achieve the desired system response. This process of
design and analysis is repeated until a satisfactory control system is obtained before
implementing the design on the hardware.

One of the most powerful tools for modeling and simulation of dynamic systems is
SIMULINK, a toolbox extension of MATLAB. SIMULINK is very easy to learn. A
system in block diagram representation is built easily and the simulation results are
displayed quickly. Simulation algorithms and parameters can be changed in the middle of
a simulation with intuitive results, thus providing the student with a ready-access learning
tool for simulating many of the operational problems found in the real world.

SIMULINK is particularly useful for studying the effects of nonlinearities on the behavior of the system, and as such, it is also an ideal research tool. Simulink has many advanced features for simulating a complex control system, such as the creation of the new sub-system blocks and *masking blocks* through M-files, C programs, or SIMULINK block diagrams, for easy integration in your system's model. This allows an extension of the SIMULINK graphical functions to suit your own needs of analysis and design. The SIMULINK demos and User's Guide for SIMULINK are very helpful in explaining the advanced usage and extension of SIMULINK block library. Also refer to Chapter 1 in "Computational Aids in Control Systems Using MATLAB, Hadi Saadat". Get the password to download this supplementary textbook from the author's personal web page.

Case Study 1 Mechanical Translational System

Consider a simple mechanical system consisting of a mass, a spring and a shock absorber known as *dashpot* or *piston* shown in Figure 4.1 (a). Where M is the mass, B is the frictional coefficient, K is the spring constant, $f(t)$ is the external force, $x(t)$ the displacement and $u(t)$ the velocity. Three forces influence the motion of the mass, namely the applied force, the frictional force, and the spring force as shown on the free-body diagram in Figure 4.1(b)
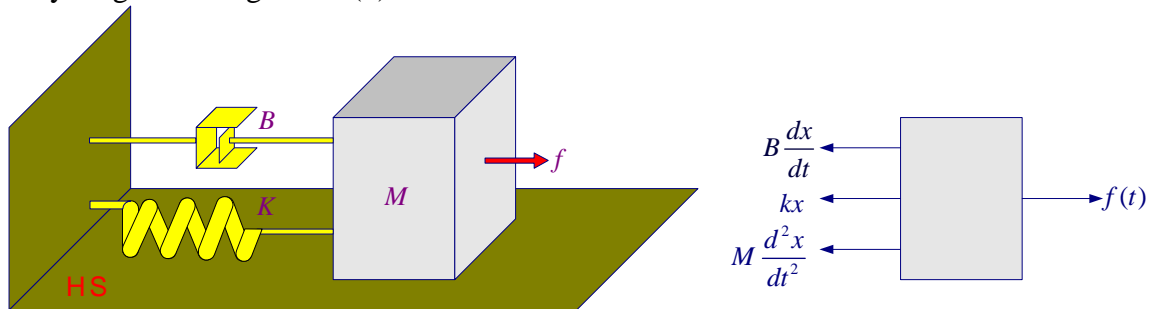

Figure 4.1 (a) Mechanical translational system.   **(b)** Free-body diagram,

Applying Newton's law of motion, we have

$$M\frac{d^2x(t)}{dt^2} + B\frac{dx(t)}{dt} + Kx(t) = f(t) \qquad (4.1)$$

The transfer function model is obtained by taking the Laplace transform, which results in

$$G(s) = \frac{X(s)}{F(s)} = \frac{1}{Ms^2 + Bs + K} \qquad (4.2)$$

As stated in the lecture for more complicated mechanical systems it is easier to draw the electric circuit force-voltage analogy in place of the free-body diagram. In force-voltage analogy, mass M is analogous to inductance, spring compliance $\frac{1}{K}$ is analogous to capacitance, frictional coefficient B is analogous to resistance, and velocity is analogous to current. The key point in drawing the electric circuit analogy is to identify the displacement or velocity of each element and draw the circuit accordingly. The circuit can be drawn in the s-domain to find the transfer function or in the time-domain suitable

for obtaining the state space model. The electric circuit analogy for this mechanical system is shown in Figure 4.2.
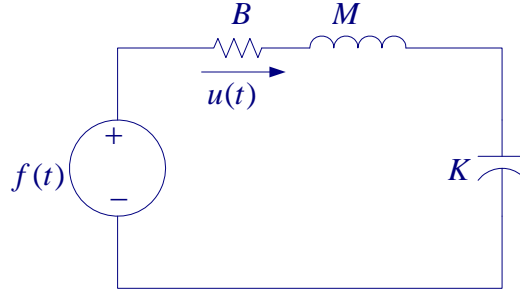

Figure 4.2 Electric circuit analogy

Applying Kirchhoff's voltage law, we have

$$M\frac{du(t)}{dt^2} + Bu(t) + K\int_0^t u(t) = f(t)$$

Since $u(t) = \frac{dx(t)}{dt}$, we have

$$M\frac{d^2x(t)}{dt^2} + B\frac{dx(t)}{dt} + Kx(t) = f(t)$$

Which is the same as (4.1). Equation (4.1) can also be written in state space form by selecting the two state variables as displacement and velocity, i.e., $x_1(t) = x(t)$ and

$$x_2(t) = u(t) = \frac{dx(t)}{dt}, \text{ then}$$

$$\frac{dx_1(t)}{dt} = x_2(t)$$

$$\frac{dx_2(t)}{dt} = \frac{1}{M}f(t) - \frac{K}{M}x_1(t) - \frac{B}{M}x_2(t)$$

(4.3)

and we define the output as the two state variables, namely $y_1 = x_1(t)$ and $y_2 = x_2(t)$
In matrix form, $\dot{x}(t) = Ax(t) + Bu(t)$, and $y(t) = Cx(t) + Du(t)$, the state and output equations are

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -K/M & -B/M \end{bmatrix}\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1/M \end{bmatrix}f(t), \text{ and } \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \quad (4.4)$$

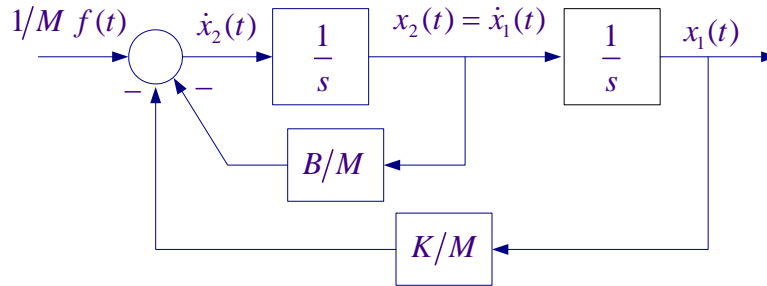The simulation diagram for equations in (4.3) is shown in Figure 4.3

Figure 4.3 Simulation diagram for the mechanical system. Case study 1

With the system initially at rest, a force of $f(t) = 32$ Newton is applied at time $t = 0$.
Mass $M = 2$ Kg, the spring constant $K = 32$ and the frictional coefficient B can be
adjusted to obtain a desirable response.
The system characteristic equation given by (4.2) is

$$s^2 + \frac{B}{M}s + \frac{K}{M} = 0 \tag{4.5}$$

This is the same form as the standard second-order transfer function

$$s^2 + 2\zeta\omega_n s + \omega_n^2 = 0 \tag{4.6}$$

1. Perform the following analysis:

(a) The dashpot damping is adjusted to $B = 2$ N-s/m. Determine the natural frequency of
oscillation ω_n, damping ratio ζ, $P.O. = e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}} \times 100$, peak time $t_p = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}}$, and

settling time $t_s \cong 4\tau$.
(b) The dashpot damping is adjusted to $B = 56$ N-s/m. Determine, damping ratio ζ,
response time constants τ_1 and τ_2, and settling time $t_s \cong 4 \times \max(\tau_1, \tau_2)$.

(c) Determine the frictional coefficient B for the response to be critically damped. What
is the response time constant and approximate settling time?

2. Digital Simulation using SIMULINK

To create a SIMULINK block diagram presentation of the system shown in Figure 4.3
double click on the SIMULINK icon on the MATLAB toolbar, or type *simulink* at the
MATLAB prompt. Click on the *create a new model* icon on the SIMULINK toolbar. An
untitled window for designing and simulating a new model will open. Double click on the
Simulink library icon; this will open nine subsystems libraries. Open the Source Library
and drag the Step Input block to the open new model window. Double click on the Step
Input to open its dialog box, set the parameters **Step time** to **0**, and Final value to 32. Get
two integrators from the Continuous library, three Gain blocks on one Sum block from
the Math Library, one Scope and one XY Graph from the Sink library, one Mux block
from Signals and Systems library. Open the Sum block dialog box and enter the required

summing point signs +--. Once you have dragged all the required blocks and placed them on the new model window, join the *in-ports* and out-ports to create the simulation model. The purpose of Mux block (Multiplex) is to combine the velocity and displacement signals into a composite signal so as to display both signals on one Scope. XYGraph is used to display the state trajectory, i.e., velocity versus displacement plot. Connect x_1 to the first input and x_2 to the second input of the XY Graph. Open the XY Graph dialog box, set the x-axis limits to 0, 2, the y-axis limits to ∓ 4 and Sample time to 0.01. An m-file named MSFanimation.m has been developed for animating the motion of the mass-spring-friction system during simulation. To add this animation, get an S-Function block form the functions & Tables library, place it on your model window and connect its input terminal to the signal coming from the output of the Mux block. Make sure that you have obtained MSFanimation.m and InAmin.m files from your instructor and placed it on a folder in the current directory. Open the S-Function block for its name enter **MSFanimation**, and for the S-Function parameter enter **0.01**. Set the gain blocks to the given values and the damping coefficient specified in part (a) above.

Solver page
Before, starting simulation, you must set the simulation parameters. Pull down the **Simulation** dialog box and select **Simulation Parameters**. Set the start time to 0 and the stop time to a suitable value. For solver option select Variable-step and any of the Continuous integration routine such as **ode45** or **ode23**. For more accurate response you may change the Relative tolerance from 1e-3 to 1e-5. If you select Fixed-step, again make sure you select a Continuous integration routine such as **ode4 (Runge-Kutta)**. You can also change the step size from auto to small value such as 0.0001. Follow the same procedure in the remaining case studies in this lab and make sure the Solver option is not set for discrete.

Simulate and obtain a print of the Scope. The scope yellow trace will not print well. Also, the Simulink XY Graph cannot be printed. A Script m-file named '**plotscope**' has been developed which captures the scope plot and produces a Figure plot. At the MATLAB prompt type

>> plotscope
then click on the Scope Figure (outside the plot area) and hit return you will have a Figure print. You can add label and legend commands or edit the graph. You can use this procedure for the XY Graph or the animation plot.

An alternative way to obtain a Figure plot is to place two To Workspace blocks from the Sink library and connect their inputs to x_1, and x_2 signals, and defining x1 and x2 for the variables. The time array can be obtained by feeding a Clock block into another To Workspace block and defining a variable t for time. After performing the simulation you can use plot function to obtain the desired Figure plot.

Repeat the simulation for the value of B given part (b) and the value determined in part (c) above. Document the plots obtained for the above three cases, determine and

summarize the time-domain specifications for each case. Comment on the nature of each response and discuss the effect of damping coefficient on the resulting response.

Figure 4.4 SIMULINK diagram for the mechanical system. Case study 1

Case Study 2 – Simple Pendulum

Consider the simple pendulum illustrated in Figure 4.5 where a mass of m kg is hung from a hinge by a rigid rod of length l meters. The rod is light enough that its mass can be neglected. The rod is displaced by angle θ radians from the equilibrium position.

Figure 4.5 Simple Pendulum

Assume a viscous friction for the motion with a damping coefficient of B Kg-s/m. The tangential velocity of the mass is $l\dfrac{d\theta}{dt}$. The tangential forces acting to restore the pendulum to equilibrium are

$$F_T = -mg\sin\theta - Bl\frac{d\theta}{dt}$$

Where g is the gravitational acceleration. Also from Newton's law, we have

$$F_T = ml\frac{d^2\theta}{dt^2}$$

Combining the above equations, we get

$$\frac{d^2\theta}{dt^2} + \frac{B}{m}\frac{d\theta}{dt} + \frac{g}{l}\sin\theta = 0 \qquad (4.7)$$

Equation (4.7) is nonlinear because of the $\sin\theta$ term.

We can now write the above equation in state variable form. Let $x_1 = \theta$, and $x_2 = \dot{\theta}$ (angular velocity), then

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -\frac{B}{m}x_2 - \frac{g}{l}\sin x_1 \qquad (4.8)$$

The simulation diagram for equations in (4.8) is shown in Figure 4.6
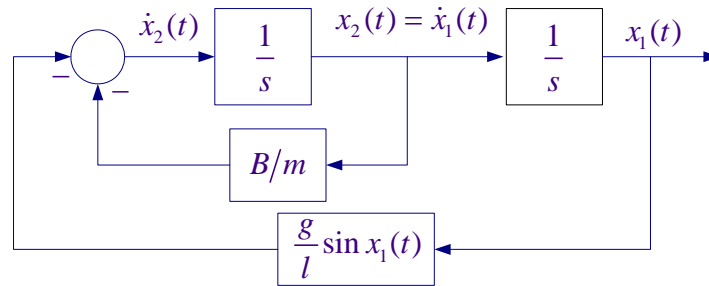

Figure 4.6 Simulation diagram for the pendulum. Case study 2

The mass m is displaced from the equilibrium by 0.5 radians ($28.65°$) at time $t = 0$. Mass $m = 0.5\,\text{Kg}$, the rod length is $l = 0.613\,\text{m}$ and the gravitational acceleration is 9.81 m/s^2. Two cases of frictional coefficient will be considered:

(a) $B = 0.05$ Kg-s/m
(b) $B = 4.0$ Kg-s/m

Create a SIMULINK block diagram presentation of the system shown in Figure 4.5. Drag all the required blocks and place them on a new model window. For the nonlinear term, get the Fcn block from the Function & Tables library. Use "u" for the input variable name, e.g. 16*sin(u). Specify the gain B/m and connect all blocks to create the simulation model shown in Figure 4.7. Open the last integrator dialog box and set the initial condition to 0.5 for the angular displacement. The initial velocity is zero. Therefore, set the Initial condition parameter for the first integrator to 0. Place one scope to display the angle θ (signal x_1) and another scope to display the velocity signal x_2. Use a XYGraph to display the state trajectory, i.e., velocity versus displacement plot. Connect

x_1 to the first input and x_2 to the second input. Open the XY Graph dialog box, and set the x-axis limits to -5, 0.5, the y-axis limits to -2, 2 and Sample time to 0.01.
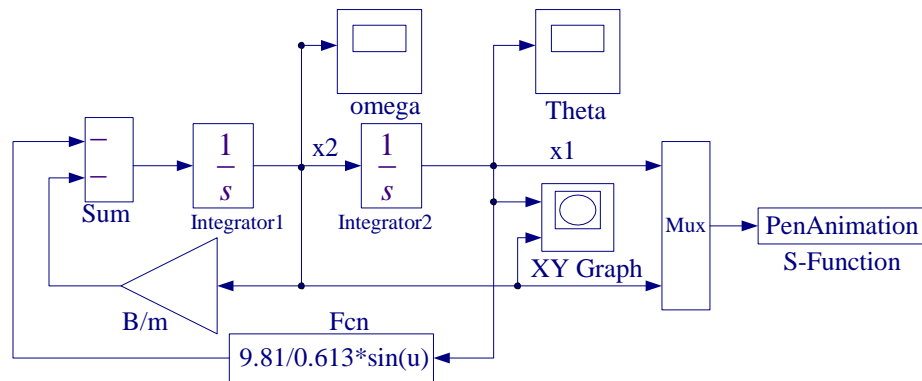

Figure 4.7 SIMULINK diagram for the pendulum. Case study 2

An m-file named PenAnimation.m has been developed for animating the pendulum swing during simulation. To add this animation, get a Mux block with two inputs. Connect x_1 to the top inport and x_2 to the lower inport. Next get an S-Function block from the Functions & Tables library, place it on the model window and connect its in-port terminal to the signal coming from the Mux block. Make sure that you have obtained this m file from your instructor and placed it on a folder in the path of MATLAB. Open the S-Function block, for the name enter PenAnimation and for the S-Function parameter enter 0.01.

(a) The frictional coefficient $B = 0.05$ Kg-s/m.

Simulate and print the zero-input response (natural response) for the angular displacement and state trajectory. You may want to reduce the simulation final time to a suitable value. Comment on the nature of response. When a system returns to its equilibrium point after a disturbance, the system is said to be stable. Is the system stable about its equilibrium point $\theta = 0$? How would you describe the system stability if the damping coefficient B were neglected? Use **plotsope** to capture the Scope trace and the XY Graph.

(b) Repeat for the frictional coefficient $B = 4$ Kg-s/m.

Linearization

Many control systems are designed to return to their equilibrium position when subjected to a small disturbance. Nonlinear systems are often linearized assuming small deviation from the equilibrium point. The nonlinear differential equation describing the motion of the pendulum can be linearized if the initial angle of deflection is small.

Let $\theta = \theta_0 + \Delta\theta$, substitute in (4.7) and expand the sine term. For small $\Delta\theta$ assuming $\sin\Delta\theta \square \Delta\theta$, $\cos\Delta\theta \square 1$, and $\cos\theta_0 \square 1$, show that (4.7) results in the linearized differential equation given by

$$\frac{d^2\Delta\theta}{dt^2} + \frac{B}{m}\frac{d\Delta\theta}{dt} + \frac{g}{l}\Delta\theta = 0 \tag{4.9}$$

This approximation is reasonably accurate for $-\pi/4 \le \theta \le \pi/4$.

(c) The state variable equation in terms of the small changes $\Delta\theta$ and $\Delta\dot{\theta}$ is the same as (4.8), except $\sin x_1$ is replaced by x_1. Copy the SIMULINK nonlinear model and paste it on the same model window, replace the Fnc block with a gain block and set the parameter to the value given by g/l. Eliminate the s-function in the duplicate model. In order to validate the linearized model use a Mux block with two inputs and connect its in-ports to the x_1 signal of each model and a Scope to its out-port terminal. Simulate for $B = 0.05$ in both models and obtain the response. State if the linearized model response is in close agreement with the nonlinear model. The characteristic equation of the linearized model is

$$s^2 + \frac{B}{m}s + \frac{g}{l} = 0 \tag{4.10}$$

(d) For the given values of l, and m, find the value of B for critically damped response. Set the B to this value in both models and repeat the simulation. Comment on the response.

Case Study 3 – Nonlinear Differential Equation with Saturation

One of the useful features of SIMULINK is the availability of nonlinear blocks, such as switch, relay, deadzone, backlash, rate-limiter, saturation, Coulomb friction, and many other nonlinear functions. These are very useful for studying the effects of nonlinearities on the behavior of the system. This study deals with the simulation of a nonlinear differential equation. The angular displacement of a dynamic system is given by

$$\frac{d^2\delta}{dt^2} + 4\frac{d\delta}{dt} + a_0\sin\delta = 30u(t) \tag{4.11}$$

where $u(t)$ is a unit step input and

$$\begin{aligned} a_0 &= 35.6 \quad \text{for} \quad 0 \le t \le t_c \\ a_0 &= 15 \quad \text{for} \quad t_c \le t \le \infty \end{aligned} \tag{4.12}$$

and δ is constrained as follows

$$-2\pi \le \delta \le 2\pi \tag{4.13}$$

t_c is a threshold switching time. Where large value of t_c may result in an unbounded response. Transforming to state variable form, let $x_1 = \delta$, and $x_2 = \dot{\delta}$, then

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -a_0\sin x_1 - 4x_2 + 30u(t) \end{aligned} \tag{4.14}$$

The SIMULINK diagram for the above system is shown in Figure 4.8. Equation (4.12) is represented by the Switch block and equation (4.13) is represented by the Saturation block.
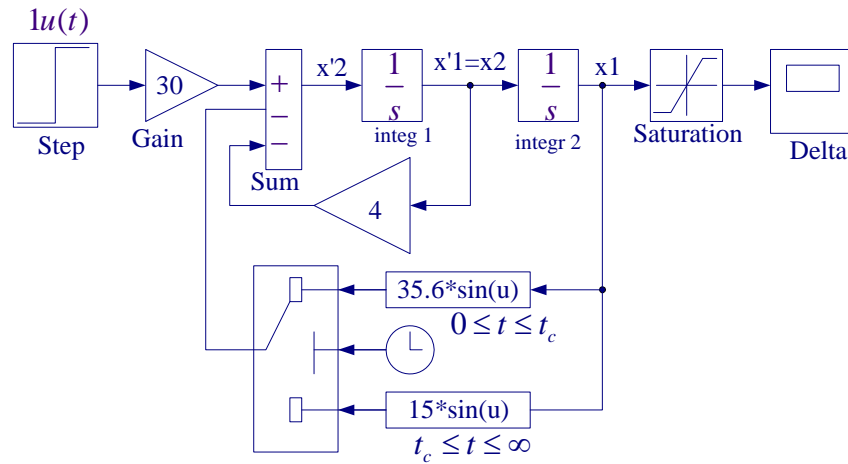

Figure 4.8 SIMULINK diagram for the Case Study 3.

Construct the above Simulink diagram, and in Step block set the **Step time** to **0**, Final time to 1. Open the Simulation Parameters dialog box, set the Stop time to 5 seconds and select ode45. Obtain the response for $t_c = 0.4$, and $t_c = 3$ seconds. Comment on the behavior of the response for each case.

Case Study 4 – Inverted Pendulum

This fourth study deals with the classic problem of balancing an inverted pendulum. This study demonstrates the control of an inherently unstable system of balancing systems that occurs in the areas of missile stabilization and robotics. This study also demonstrates the Linearization of a nonlinear system.

Figure 4.9 shows an inverted pendulum of length l and mass m supported by a frictionless pivot on a cart of mass M. It is to be balanced by means of a force u applied to the cart. That is, the cart must be moved in such a way that the pendulum is in upright position. In a physical system there would be sensors to measure the position and velocity of the cart and the angle θ measured from the vertical position. This is also a model of the attitude control of a space booster on takeoff.
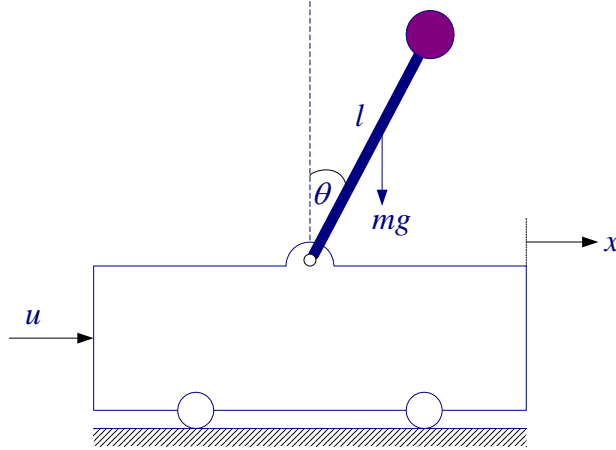
Figure 4.9 Inverted Pendulum on a cart.

This is similar to the balancing of a broomstick on the palm of your hand. The equilibrium condition is when $\theta(t)$, and $\dot{\theta}(t)$ returns to zero. The visual location of your hand and the position of the broomstick and the proper movement of your hand is the required feedback without which it is not possible to balance the broomstick.

The differential equations describing the motion of the system are obtained by summing the forces on the pendulum, which result in the following nonlinear equations.

$$
\begin{aligned}
(M+m)\ddot{x}+(mL\cos\theta)\ddot{\theta} &= (mL\sin\theta)\dot{\theta}^2 + u \\
(mL\cos\theta)\ddot{x}+mL^2\ddot{\theta} &= mgL\sin\theta
\end{aligned}
\tag{4.15}
$$

(a) Linearize the above equations in the neighborhood of the zero initial states. Hint: Substitute θ for $\sin\theta$, 1 for $\cos\theta$ and 0 for $\dot{\theta}^2$. With the state variables defined as $x_1 = \theta$, $x_2 = \dot{\theta}$, $x_3 = x$, and $x_4 = \dot{x}$, show that the linearized state equation is

$$
\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \dfrac{M+m}{ML}g & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\dfrac{m}{M}g & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ -\dfrac{1}{ML} \\ 0 \\ \dfrac{1}{M} \end{bmatrix} u
\tag{4.16}
$$

If we want to have all the state variables available as the output, we define the C matrix as an identity matrix and D is a 4×1 zero matrix

4.11

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad (4.17)$$

(b) The parameters of the inverted pendulum are $M = 4$ kg, $m = 0.2$ kg, $L = 0.5$ m, and $g = 9.81$ m/s. In a MATLAB script file define the system parameters and the A, B, C, D matrices.

```
M = 4;  m = 0.2;  g =9.81;  L =.5;
A = [   0              1        0       0
      (M+m)/(M*L)*g    0        0       0
           0           0        0       1
     -m/M*g            0        0       0]
B = [0;          -1/(L*M);   0;      1/M],   % Column vector
C = eye(4),                                  % Identity matrix
D = zeros(4, 1)                              % Column vector
x_0 = [0.1  0  0.1  0];
```

Save the file as Lab4CS4aData.m

Launch Simulink, open a new model, get the State-Space block from the Continuous library and construct the above state model. Double click on the State-Space block to open its dialog box and for parameters type A, B, C, and D. Note that MATLAB is case sensitive. For the initial condition type, [0.1   0   0.1   0] or x_0. In the Simulation Parameters dialog box set Start Time to 0, Stop Time to 3 second. For Solver option use a variable step size and ode45 algorithm. Connect a Demux block to separate the signals, and use two scopes to display θ and x. Get an Inport block from the Source library and use it as an input terminal. Also get an Outport block from the Sink library and connect it to the signal for θ as shown in Figure 4.10. These terminals will enable us to find the system transfer function from the Simulink diagram.
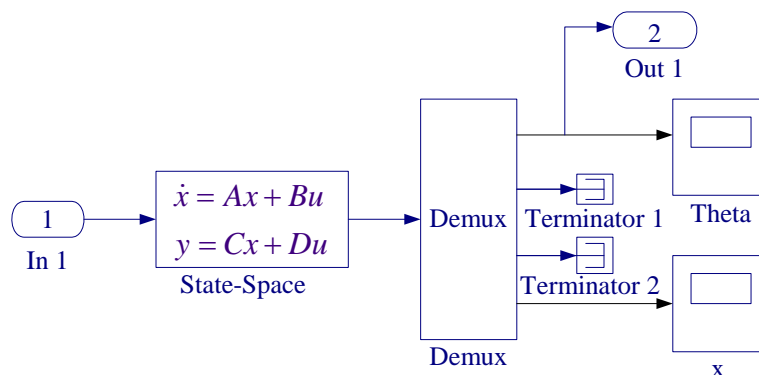

Figure 4.10 Simulink block diagram for the Inverted Pendulum.

Save the Simulink model as Lab4CS4a.mdl.

Run the script m-file Lab4CS4aData at the MATLAB prompt to calculate the A, B, C, and D matrices. These values are now defined and are available in Simulink. Start the Simulation in SIMULINK and obtain a plot of θ and x. Comment on the stability of the system. MATLAB provides the function **linmod** to extract a linear model in state variables or as a transfer function model using the Simulink file name as argument. At the MATLAB prompt type the following commands to obtain the linearized transfer function model, and roots of the characteristic equation.

```
[num, den]=linmod('Lab4CS4a')
r = roots(den)
```

You may have found that the angle θ increases without limit, i.e. the response is unbounded. Also you may find that a root of the characteristic equation is positive. This again confirms an unbounded response and we say that the system is unstable, that is, the inverted pendulum will fall over unless a suitable control force via state feedback is used.

(c) The purpose is to design a control system such that for a small initial disturbance the pendulum can be brought back to the vertical position ($\theta = 0$), and the cart can be brought back to the reference position ($x = 0$).

One approach in modern control systems, accomplished by the use of state feedback, is known as *pole-placement design*. The pole-placement design allows all roots of the system characteristic equation to be placed in desired locations. This results in a regulator with K constant gain vector. In pole-placement design the control is achieved by feeding back the state variables through a regulator with constant gains. Consider the control system presented in the state-variable form

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t)$$

(4.18)

Consider the block diagram of the system shown in Figure 4.11 with the following state feedback control.

$$u(t) = -Kx(t)$$

(4.19)

where K is a $k \times 1$ matrix of constant feedback gain. The control system input is assumed to be zero. The purpose of this system is to return all state variables to values of zero when the states have been perturbed.
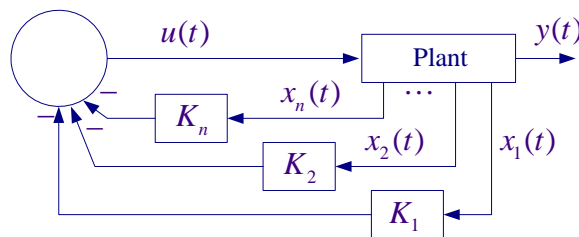
Figure 4.11: Control system design via pole placement.

Substituting (4.19) into (4.18), the closed-loop system state-variable representation is

$$\dot{x}(t) = [A - BK]x(t) = A_f x(t) \qquad (4.20)$$

The design objective is to find the gain matrix K such that the characteristic equation for the controlled system is identical to the desired characteristic equation. The derivation is straightforward; refer to "Computational Aids in Control Systems using MATLAB, Hadi Saadat, McGraw-Hill 1993, Chapter 8, page 170." A custom-made function named **[K, Af] = placepol(A, B, C, P)** is developed for the pole placement design. A, B, C are system matrices and P is a row vector containing the desired closed-loop poles. This function returns the gain vector K and the closed-loop system matrix Af. Also, the MATLAB Control System Toolbox contains two functions for pole-placement design. Function **K = acker(A, B, P)** is for single input systems, and function **K = place(A, B, p)**, which uses a more reliable algorithm, is for multi-input systems.

An aspect of state variable design is state feedback design. In this study use the custom made function **[K, Af] = placepol(A, B, C, P)** and design a state feedback controller to place the closed-loop poles at

$$P = [-1 \pm j0.5, \ -4, \ -5]$$

Add the above two statements to the script file Lab4CS4aData4.m as shown below.

```
M = 4;  m = 0.2;  g =9.81;  L =.5;
A = [   0                  1        0       0
      (M+m)/(M*L)*g    0        0       0
          0              0        0       1
   -m/M*g              0        0       0]
B = [0;          -1/(L*M);    0;       1/M],    % Column vector
C = eye(4),                                     % Identity matrix
D = zeros(4, 1)                                 % Column vector
P =[-2+j*.5,  -2-j*.5,  -5,  -4];
[K, Af] = placepol(A, B, C, P)
```

Save as Lab4CS4bData.m. In Simulink, open the Lab4CS4a.mdl. Add the state feedbacks and set the gains to K(1), K(2), K(3), and K(4) as shown in Figure 4.12.
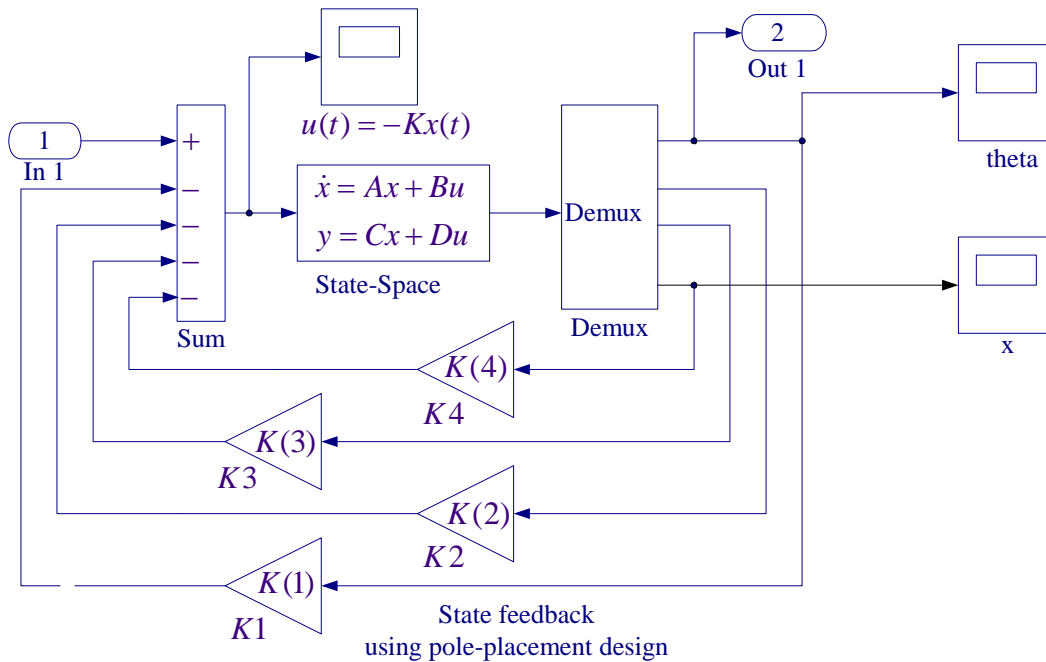
Figure 4.12 Control of Inverted Pendulum via pole placement.

Run the script file Lab4CS4bData to evaluate the gain matrix K for use in the Simulink. Rename the model as Lab4CS4b.mdl. Start the Simulation in SIMULINK and obtain a plot of θ and x, comment on the stability of the system. To see an inverted pendulum animation make sure you have obtained the m-file named 'InvPenAnimation.m'. After simulation type InvPenAnimation at the MATLAB prompt.

At the MATLAB prompt type the following commands to obtain the linearized transfer function model, and roots of the compensated characteristic equation.

```
[num, den]=linmod('Lab4CS4b')
r = roots(den)
```

Check for the roots of the compensated system. Are they the same as the specified values? Is the system stable that is, will the pendulum return to the vertical equilibrium position?