# FLIGHT CONTROL LAB

# (R22D7683)

**LAB MANUAL**

## I M. Tech II Semester (2023-2024)

## Department of Aeronautical Engineering


## MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

## (Autonomous Institution – UGC, Govt. of India)

Affiliated to JNTU, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Kompally), Secunderabad – 500100, Telangana State, India

## MRCET VISION

To become a model institution in the fields of Engineering, Technology and Management. To have a perfect synchronization of the ideologies of MRCET with challenging demands of International Pioneering Organizations.

## MRCET MISSION

To establish a pedestal for the integral innovation, team spirit, originality and competence in the students, expose them to face the global challenges and become pioneers of Indian vision of modern society.

## MRCET QUALITY POLICY.

To pursue continual improvement of teaching learning process of Undergraduate and Post Graduate programs in Engineering & Management vigorously. To provide state of art infrastructure and expertise to impart the quality education.

# PROGRAM OUTCOMES
# (PO's)

## Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design / development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.
12. **Life- long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

# DEPARTMENT OF AERONAUTICAL ENGINEERING

## VISION

Department of Aeronautical Engineering aims to be indispensable source in Aeronautical Engineering which has a zeal to provide the value driven platform for the students to acquire knowledge and empower themselves to shoulder higher responsibility in building a strong nation.

## MISSION

The primary mission of the department is to promote engineering education and research. To strive consistently to provide quality education, keeping in pace with time and technology. Department passions to integrate the intellectual, spiritual, ethical and social development of the students for shaping them into dynamic engineers

## QUALITY POLICY STATEMENT

Impart up-to-date knowledge to the students in Aeronautical area to make them quality engineers. Make the students experience the applications on quality equipment and tools. Provide systems, resources and training opportunities to achieve continuous improvement. Maintain global standards in education, training and services.

## PROGRAM EDUCATIONAL OBJECTIVES – Aeronautical Engineering

1. **PEO1 (PROFESSIONALISM & CITIZENSHIP):** To create and sustain a community of learning in which students acquire knowledge and learn to apply it professionally with due consideration for ethical, ecological and economic issues.
2. **PEO2 (TECHNICAL ACCOMPLISHMENTS):** To provide knowledge based services to satisfy the needs of society and the industry by providing hands on experience in various technologies in core field.
3. **PEO3 (INVENTION, INNOVATION AND CREATIVITY):** To make the students to design, experiment, analyze, and interpret in the core field with the help of other multi disciplinary concepts wherever applicable.
4. **PEO4 (PROFESSIONAL DEVELOPMENT):** To educate the students to disseminate research findings with good soft skills and become a successful entrepreneur.
5. **PEO5 (HUMAN RESOURCE DEVELOPMENT):** To graduate the students in building national capabilities in technology, education and research

## PROGRAM SPECIFIC OUTCOMES – Aeronautical Engineering

1. To mould students to become a professional with all necessary skills, personality and sound knowledge in basic and advance technological areas.
2. To promote understanding of concepts and develop ability in design manufacture and maintenance of aircraft, aerospace vehicles and associated equipment and develop application capability of the concepts sciences to engineering design and processes.
3. Understanding the current scenario in the field of aeronautics and acquire ability to apply knowledge of engineering, science and mathematics to design and conduct experiments in the field of Aeronautical Engineering.
4. To develop leadership skills in our students necessary to shape the social, intellectual, business and technical worlds.

## COURSE OUTCOMES

1.Basic knowledge on mathematical programming language.

2.Develop skills in programming language.

3.Ability to model aerospace problems through mathematical models.

4. Revise computational strategies for developing applications.

5.Ability to develop Simple to Complex applications using programming language.

# I Year M. Tech, ANE-II Semester

# (R22D7683) FLIGHT CONTROL LAB

## LIST OF EXPERIMENTS:-

The student is expected to conduct 8 exercises

1. Introduction to modeling software.
2. Programs using mathematical functions and plotting functions.
3. Program to solve differential equations.
4. Program to solve system of equations using numerical methods.
5. Program to ISA profile for flight.
6. Program to find aerodynamic static stability study of a symmetrical airfoil.
7. Program the stability analysis using Root locus, Bode plot techniques.
8. Design of lead, lag and lead –lag compensator for aircraft dynamics.
9. Introduction to Standard Simulink libraries, Simulink aerospace blockset, Building Simulink linear models: transfer function.
10. Simulation of Aircraft motion-longitudinal dynamics, lateral dynamics.

Software Required: MATLAB OR SCI lab software

## REFERENCES:


1. MATLAB an Introduction with Applications Fifth Edition AMOS GILAT by WILEY Publications
2. Programming in SCI lab by VINU V DAS NEW AGE INTERNATIONAL PUBLICATIONS

# INDEX

# EXPERIMENT -1

# INTRODUCTION TO MATLAB SOFTWARE

**Aim:** To study and solve following types of problems using MATLAB

(a) Simple arithmetic operations using command window.

(b) Creating arrays and mathematical operations with arrays(dot and cross product, inverse, transpose, Eigen values, solutions of linear equations).

(c) Creating simple script file.

**Equipment and material needed:** Core 2 duo processor with 1GB RAM and MATLAB software Version 2008a or higher. In addition, thorough understanding of chapter 1, 2 and 3 of Reference book 1(MATLAB: An Introduction with Applications by Amos Gilat).

**Brief introduction about MATLAB and algorithm:** The MATLAB® is a very popular language for technical computing by students, engineers, and scientists in universities, research institutes, and industries all over the world. The software is popular and easy to use. A brief on few features are explained below. For complete coverage you should go through all the chapters of Reference 1. These will also be explained to you during the LAB.

It is utmost important that you bring this reference book during the LAB for ease of understanding and quick reference. It will be of help if the concerned faculty keeps the copy of the book in the Lab for their reference and demonstration.

**Command Window:** Command Window is used to enter variables and to run functions and M-file scripts.

**Command History:** Statements entered in the Command Window are logged in the Command History. From the Command History, one can view and search for previously run statements, as well as copy and execute selected statements. One can also create an M-file from selected statements.

**Current Directory:** A quick way to view or change the current directory is by using the current directory field in the desktop toolbar

**Workspace:** The MATLAB® workspace consists of the set of variables built up during a MATLAB session and stored in memory. Variables are added to the workspace by using functions, running M-files, and loading saved workspaces.

**Editor:** Editor is used to create and debug M-files, which are programs we write to run MATLAB® functions. The Editor provides a graphical user interface for text editing, as well as for M-file debugging. To create or edit an M-file one use File > New or File > Open, or use the edit function.

**Keep the following points in mind when working with MATLAB.**

- Upper and lower-case characters are not equivalent (MATLAB is case sensitive).
- Typing the name of a variable will cause MATLAB to display its current value.
- A semicolon at the end of a command suppresses the screen output.
- MATLAB uses both parentheses, (), and square brackets, [], and these are not interchangeable.
- The up arrow and down arrow keys can be used to scroll through previous commands. Also, an old command can be recalled by typing the first few characters followed by up arrow.
- One can type help topic to access online help on the command, function or symbol topic.
- You can quit MATLAB by typing exit or quit.
- First character of the saved M-file should be a letter and not a number. So do not save file using your roll no in the beginning of the file name.
- Having entered MATLAB, you should work through this tutorial by typing in the text that appears after the MATLAB prompt, >>, in the Command Window. After showing you what to type, we display the output that is produced. Faculty will demonstrate and help you in thefollowing tutorial.

  **Defining Scalar Variable:** Do not use reserved key words as variables (e.g. break,else, while etc.).

```
>> x=15;
X=15
>> x=3*x-12
X=
33
>>E=sin(x)^2+cos(x)^2
E=
1
>>
```

**Creating Arrays and Mathematical Operations with Arrays**: The array is a fundamental form that MATLAB uses to store and manipulate data. It is a list of numbers arranged in rows and/or columns. The simplest array is a row or a column of numbers(one-dimensional). A more complex array (2-D) is a collection of numbers arranged in rows and columns. In science and engineering 1-D arrays frequently represents vectors, and two-dimensional arrays often represent **matrices**. Faculty will demonstrate to you how to create and perform mathematical operations on arrays.

-Create an array from given population data in Table 2-1: (Referene-1 text book chapter-2).

**Table2-1: Population data**

| Year | 1984 | 1986 | 1988 | 1990 | 1992 | 1994 | 1996 |
|---|---|---|---|---|---|---|---|
| Population(millions) | 127 | 130 | 136 | 145 | 158 | 178 | 211 |

>>yr= [1984,1986,1988,1990,1992,1994,1996]

yr=

1984 1986 1988 1990 1992 1994 1996

pop=[127,130,136,145,158,178,211]

**-create a vector by specifying first term,the spacing and the last term**.

Example:

> X=[1:2:13]

X

1 3 5 7 9 11 13

**-create a vector using linear spacing by specifying the first and last terms**, and the number of terms:

Va= linspace(0,8,6)

Va=

0 1.6000 3.2000 4.8000 6.4000 8.0000

Some of the useful commands for matrix manipulations are tabulated below with examples.

**Table-1: Useful functions on Matrix operations**

| S.No | **Command** | **Description** |
|---|---|---|
| 1 | A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1] | Matrix |
| 2 | A' | Transpose of Matrix |
| 3 | diag(A) | Diagonal elements of matrix |
| 4 | A(4,2) | the number in the fourth row and second column |
| 5 | 100:-7:50 | a row vector containing the integers from 100 to 50 with decrement of 7 |
| 6 | sum(A(1:4,4)) | computes the sum of the fourth column |
| 7 | Z=zeros(2,4) | 2x4 matrix with all zeros |
| 8 | C=ones(1,3) | 1x3 matrix with all ones |
| 9 | A(:,2) = [] | Deleting second row |
| 10 | E = A([1,1,1],:) | copies the first row of *A three* times to create a new matrix |
| 11 | det(A) | Determinant of matrix |
| 12 | X = inv(A) | Inverse of a matrix |
| 13 | e = eig(A) | Eigen values of a matrix |
| 14 | eye(3x3) | Creates 3x3 Identity matrix |
| 15 | linspace(a,b,n) | creates a row vector of *n* regularly spaced elements between *a* and *b* |

| | | |
|---|---|---|
| 16 | function [out1, out2, ...] = funname(in1, in2, ...) | *out1, out2, ...,* are the function outputs, *in1, in2, ...* are its inputs and *funname* is the function name then the function can be called in the command window or in other m-files. |

**Creating script file (M-file): A** script file is a list of MATLAB commands, called a program that is saved in a file. When the script file is executed(run), MATLAB executes the commands. You will be demonstrated by the faculty, how to create, save and run a simple script file in which commands are executed in order in which they are listed, and in which all the variables are defined within the script file. On similar line we can write functions for various applications.

**Result: -**Students should be able to formulate and solve simple problems using various functions including Arrays and Matrices available in MATLAB.

## EXERCISES

Solve the flowing problems given in Chapter4 of Reference book 1.

1. Problem 8

2. Problem 16

3. Problem 13

4. Problem 25

5. Problem 28

6. Problem 29

7. Compute the array and matrix product of $A = \begin{bmatrix} 8 & 7 & 11 \\ 6 & 5 & -1 \\ 0 & 2 & -8 \end{bmatrix}$, $B = \begin{bmatrix} 2 & 1 & 2 \\ -1 & 6 & 4 \\ 2 & 2 & 2 \end{bmatrix}$

8. Find a solution to the following set of equations:

$$x + 2y + 3z = 12$$
$$-4x + y + 2z = 13$$
$$9y - 8z = -1$$

What is the determinant of the coefficient matrix?

9. Write a function ‘altitude’ which takes static pressure in millibar as input argument and computes the pressure altitude in meters using standard atmosphere. (Kindly refer to any book on aerodynamics for relation between pressure and altitude in standard atmosphere.

# EXPERIMENT -2

# PROGRAMS FOR TWO-DIMENSIONAL (2-D) AND THREE-DIMENSIONAL (3-D)PLOTTING

**Aim:** To study the programs for creating 2D & 3D Plots.

**Equipment needed:** Core 2 duo processor with 1GB RAM and MATLAB software Version 2008a or higher

## Theory:

**Plotting:** Plots are very useful tool for presenting information. This is true in any field, but specially in science and engineering, where MATLAB is mostly used. MATLAB has many commands that can be used for creating different types of plots. These include standard plots with linear axes, plots with logarithmic and semi-logarithmic axes, polar plots, 3-D contour surfaces and mesh plots, and many more. In this experiment you will learn how MATLAB can be used to create and format many types of 2-D and 3-D plots.

**2-D Plot of a Function**. In many situations there is a need to plot a given function. This can be done by using the 'plot' or the 'fplot' command. In order to plot a function y =f(x) with the plot command, the user needs to first create a vector of x for the domain over which the function will be plotted. Then a vector y is created with the corresponding values of f(x) by using element-by-element calculations as explained in Experiment number 1. Once the two vectors are defined, they can be used in the plot command. The fplot command plots a function with the form y=f(x) between specified limits. The command has the form

fplot('function',limits,'line specifiers')

'function' can be typed directly as a string inside the command. For example if the function that is being plotted is f(x) = 8x^2+5 cos(x),it is typed as:'8*x^2+5*cos(x)'. the function can include MATLAB built-in functions and functions that are created by the users.

**Plotting multiple graphs in the same plot**. In many situations there is a need to make several graphs in the same plot. There are three methods to plot multiple graphs in one figure. One is by using the plot command, the second by using the hold on and hold off and the third is by using the line command.

**Three-Dimensional Graphics**: MATLAB provides a variety of functions to display 3-D data. Some functions plot lines in 3-D, while others draw surfaces and wire frames.In addition, color can be used to represent a fourth dimension. When color is used in this manner, it is called pseudo color, since color is is not inherent or natural property of the underlying data in the way that color in a photograph is natural characteristic of the image.

(a) **LINE PLOTS**. General format of the command is plot3(x1,y1,z1,S1,x2,y2,z2,S2,…) where xn,yn and zn are vectors or matrices and Sn are optional character strings specifying color, marker symbol,and or line style.

(b) **MESH PLOT**. MATLAB defines a mesh surface by the z-coordinates of points above a rectangular grid in the x-y plane. It formats a mesh plot by joining adjacent points with straight lines. The result looks like a fishing net with knots at the data points.

(c) **SURFACE PLOTS**. A surface plot is like a mesh plot, except that the spaces between the lines called patches are filled in. Plots of this type are generated using the surf function.

All the above features will be demonstrated to you by the faculty with examples discussed below. Some important commands for plotting are tabulated below for quick reference.

**Table 2: Important plot commands in MATLAB**

| S.No | Command | Description |
|---|---|---|
| 1 | plot(*x, y*) | Plots the variation of y with respect to x |
| 2 | xlabel('x') | Gives the label for x axis |
| 3 | ylabel('cos(x)') | Gives the label for y axis |
| | title('plot name') | Gives the name of plot |
| 4 | fplot ( 'function string,' [*x*start, *x*end]) | The function fplot gets around our choice of interval used to generate the plot, and<br>instead decides the number of plotting points to use for us. |
| 5 | plot(t,f,t,g,'--') | To plot multiple functions, call the plot(*x, y*) command with multiple pairs *x, y* defining the independent and dependent variables used in the plot in pairs. This is followed by a character string enclosed in single quotes to tell us what kindof line to use to generate the second curve |
| 6 | 'Linewidth'2 | Increases thickness of curve |
| 7 | legend('sinh(x)','cosh(x)') | To name the curves |
| 8 | plot(x,y,'r',x,z,'b') | Differentiates the curves with colors r-red, b-blue, g-green, k-black, w-white, y-yellow, m-magenta, c-cyan. |
| 9 | axis([xmin xmax ymin ymax]) | Plot range |
| 10 | subplot(1,2,1) | Creates the plot with 2 panes and 1 row, and that this particular plot will appear in the first pane |
| 11 | polar ( theta, r) | Creates Polar plots |
| 12 | bar(x,y) | Creates bar chart |
| 13 | [x,y] = meshgrid(–5:0.1:5,–3:0.1:3); | Generate a matrix of elements that give the range over *x* and *y* we want to use along with the specification of increment in each case. |
| 14 | mesh(x,y,z) | 3D plots |
| 15 | surf(x,y,z) | Shaded surface plots |

**Example 1: Plotting multiple graphs in the same plot using fplot and hold command.**

Plot three sine waves with different phases. For the first, use a line width of 2 points. For the second, specify a dashed red line style with circle markers. For the third, specify a cyan, dash-dotted line style with asterisk markers.

**Code:**

```
fplot(@(x) sin(x+pi/5),'Linewidth',2);
holdon
fplot(@(x) sin(x-pi/5),'--or');
fplot(@(x) sin(x),'-.*c')
holdoff
```

**Result:** -The plot is shown in Fig 2.1**.** The default limit for x : -5,5

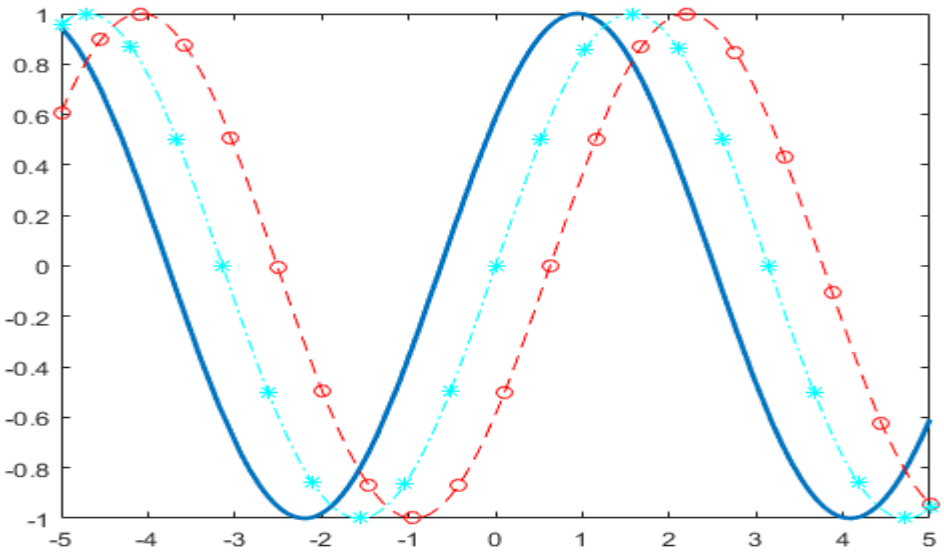



**Fig 2.1**

**Example 2:** Polar plot of a function r=3 cos^2(0.5θ) + θ for 0<= θ<= 2*pi

```
t= linspace (0,2*pi,200);
r=3*cos(0.5*t).^2+t;
polar(t, r)
```

Result: polar plot is shown in Fig 2.2

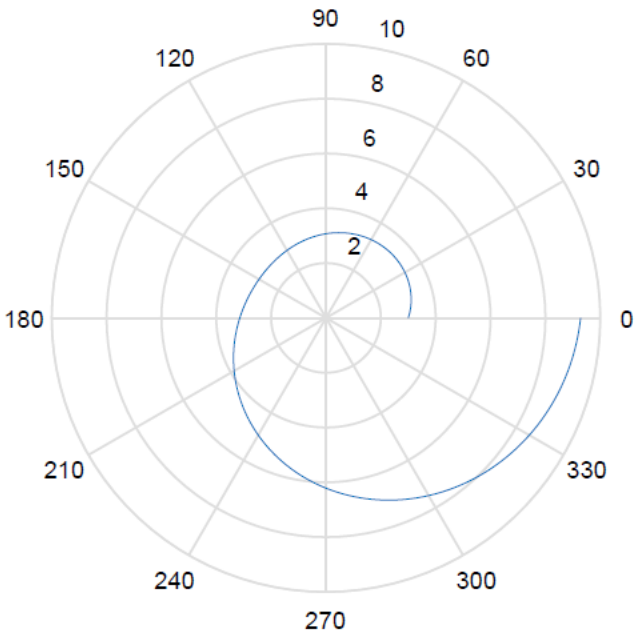
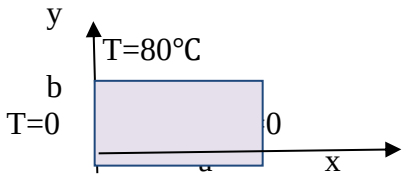



**Fig 2.2 Polar plot**

**Example 3 for 3-D plot.** Three sides of a rectangular plate (a=5m, b=4m) are kept at temperature of 0 deg C and one side is kept at a temperature T1=80 deg C, as shown in the figure. Determine and plot the temperature distribution T(x,y) in the plate.




T=0

**Solution.** The temperature distribution, T(x,y) in the plate can be determined by solving the 2-D heat equation. For given boundary conditions T9x,y) can be expressed analytically by a Fourier series:

$$T(x,y)=\frac{4T_1}{\pi}\sum_{n=1}^{\infty}\frac{\sin\left[(2n-1)\frac{\pi x}{a}\right]}{(2n-1)}\frac{\sinh\left[(2n-1)\frac{\pi y}{a}\right]}{\sinh\left[(2n-1)\frac{\pi b}{a}\right]}$$

A program in a script file that solves the problem is listed below. The program follows these steps:

(a) Create an X,Y grid in the domain $0\leq x \leq a$and $0\leq y \leq b$. The length of the plate, a, is divided into 20 segments, and the width of the plate, b, is divided into 16 segments.

(b) Calculate the temperature at each point of the mesh. The calculation are done point by point using a double loop. At each point the temperature is determined by adding k terms of the Fourier series.

(c) Make a surface plot of T.

```
% 3-D plot for heat equation PLMM lab
%
% script file
 a=5;
 b=4;na=20;nb=16;T0=80;k=5;
x=linspace(0,a,na);
y=linspace(0,b,nb);
[X,Y]=meshgrid(x,y);
for i=1:nb
for j=1:na
    T(i,j)=0;
for n=1:k
    ns=2*n-1;       % third loop, n, is the
% term of the Fourier series,% k is the number of  %terms
T(i,j)=T(i,j)+sin(ns*pi*X(i,j)/a).*sinh(ns*pi*Y(i,j)/a)/(sinh(ns*pi*b/a)...
*ns);
end
   T(i,j)=T(i,j)*4*T0/pi;
end
end
mesh(X,Y,T)
xlabel( 'x  (m)'); ylabel('y  (m) ');
zlabel(' T  ( ^0C) ')
```

Program was executed with two different values of k ( 5 &50). The mesh plots are shown in each case in the figure 2.3 and 2.4. the temperature should be uniformly at  y=4 m. Note the effect of k on the accuracy at y=4m.

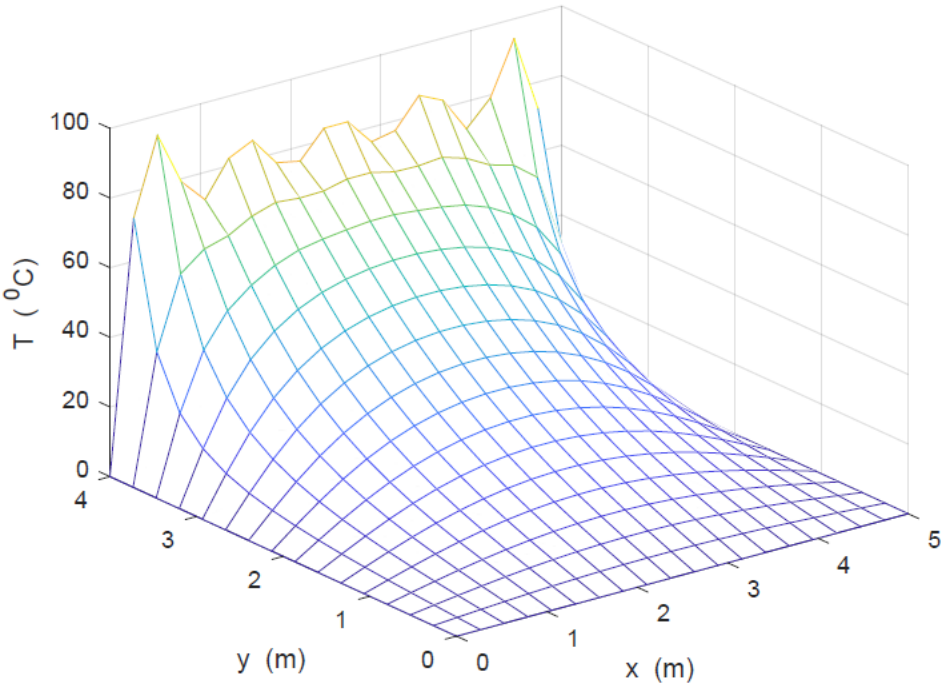



Fig 2.3: for k=5;

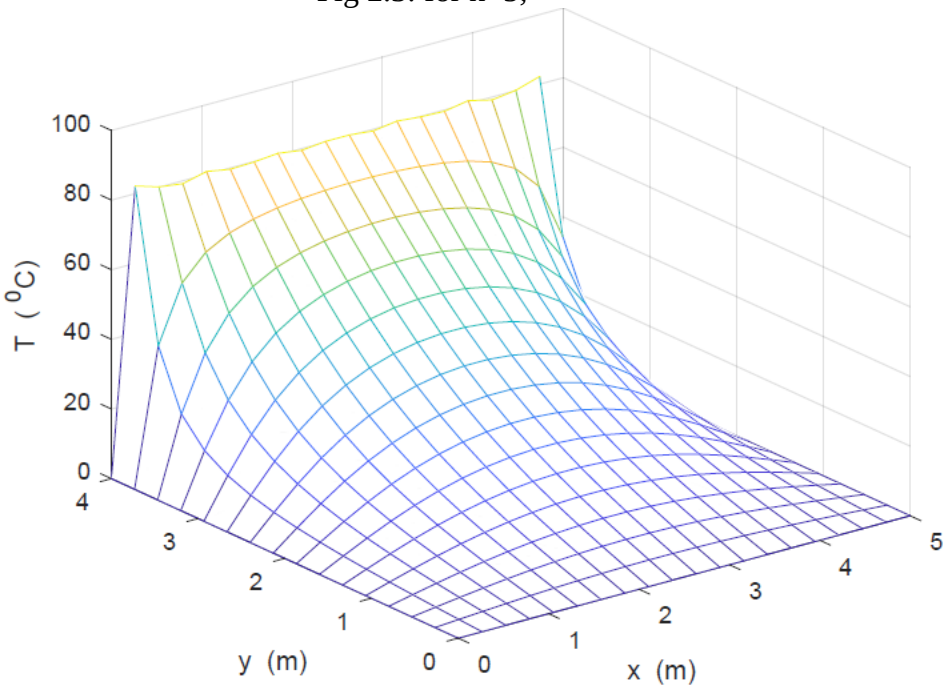



Fig 2.4: For k=50

## EXERCISES

**1.** Acycloid is a curve traced by a point on a circle that rolls along a line. The parametric equation of a cycloid is given by

x=r(t-sint) and y=r(1-cost)

Plot a cycloid with r=1.5 and $0 \leq t \leq 4\pi$

2. Solve the problem No 24 relating to NACA airfoil given in chapter 5 of reference book 1.

3. Solve the problem no 31 relating to simply supported beam given in Chapter 5 of reference 1

4. Solve the problem no 26 relating to vibrations of chapter 5 of reference 1.

**5.** Solve the problem no 19 of chapter 5 of reference 1 relating to tensile strength**.**

6. Solve the problem no 14 of chapter 10 of reference 1 relating to defect in crystal lattice.

7. Solve problem No 19 of chapter 10 of reference book 1.

8. In a study of the effect of various factors on the growth performance of activated sludge, the oxygen uptake rate was measured at various temperatures.

| Temperature °C | Oxygen uptake rate grams oxygen per gram dry Weight |
|---|---|
| 5 | 0.01 |
| 10 | 0.04 |
| 15 | 0.10 |
| 20 | 0.20 |
| 25 | 0.25 |
| 30 | 0.28 |
| 35 | 0.30 |
| 40 | 0.25 |
| 45 | 0.02 |

Write a script M-file that generates a plot of these data, including title, labels, and grid. Print the generated graph.

# EXPERIMENT 3

# PROGRAM TO SOLVE DIFFERENTIAL EQUATIONS

**Aim:** To learn various functions available in MATLAB to solve initial value problems(IVPs) and Boundary value problems(BVPs).

**Equipment needed:** Core 2 duo processor with 1GB RAM and MATLAB software Version 2008a or higher.

## Theory:

MATLAB has the capability to solve a wide variety of problems involving differential equations. In this exercise we will learn how to solve IVPs because they appear most often in applications. The IVP solvers in MATLAB compute the time history of a set of coupled first-order differential equations with known initial conditions. In mathematical terms, these problems have the form

$$\dot{\boldsymbol{y}}=\mathbf{f}(\boldsymbol{t},\boldsymbol{y}) \qquad \mathbf{y}(\boldsymbol{t_0})=\boldsymbol{y_0}$$

Which is vector notation for the set of differential equations

$\dot{y_1}= f_1(t, y_1, y_2 \dots y_n)$ $y_1(t_0)=y_{10}$

$\dot{y_2}= f_2(t, y_1, y_2 \dots y_n)$ , $y_2(t_0) =y_{20}$

$\dot{y_n}= f_n(t, y_1, y_2 \dots y_n)$ $y_n(t_0) =y_{n0}$

Where $\dot{y_i} =\frac{dy_i}{dt}$ , n is the number of first order differential equations and $y_{i0}$ is the initial conditions associated with the ith equations. When an initial value problem is not specified as a set of first order differential equations, it must be rewritten as one. For example, consider the classic **van der Pol equation**

$\ddot{x}$ -μ$(1 - x^2)\dot{x}$ + x=0;

Where μis a parameter greater than zero. If we choose $y_1$=x and $y_2$ =dx/dt, the van der Pol equation becomes

$\dot{y_1}= y_2$

$\dot{y_2}=$ μ$(1 - {y_1}^2)y_2$ - $y_1$

**ODE Suite Solvers:**

The MATLAB ODE suite offers various initial value problems solvers. Each has characteristics appropriate for different initial value problems. The calling syntax for each solver is identical, making it relatively easy to change solvers for a given problem. Some of them are given below

(a) **Ode23**: an explicit one-step Runge-Kutta low-order ($2^{nd}$-to $3^{rd}$-order) solver. Suitable for problems that exhibit mild stiffness, problems where lower accuracy is acceptable, or problems where f(t,y) is not smooth(e.g. discontinuities)

(b) **Ode23s**: an implicit one-step modified Rosen Brock solver of order two. Suitable for stiff problems where lower accuracy is acceptable or where f(t,y) is discontinuous. **Stiff are generally described as problems in which the underlying time constants vary by several orders of magnitude or more**.

(c) **Ode23tb**: An implicit trapezoidal rule followed by a backward differentiation of order two. Similar to ode23s. it can be more efficient than ode15s for crude tolerance.

**(d) Ode45**: An explicit one-step Runge Kutta medium-order ($4^{th}$- to $5^{th}$-order) solver. Suitable for non-stiff problems that require moderate accuracy. **This is typically the first solver to try on a new problem.**

**Algorithm:** Before a set of differential equations can be solved, they must be coded in a function M-file as ydot = odefile(t,y).That is, the file must accept a time t and a solution y and return values for the derivatives.

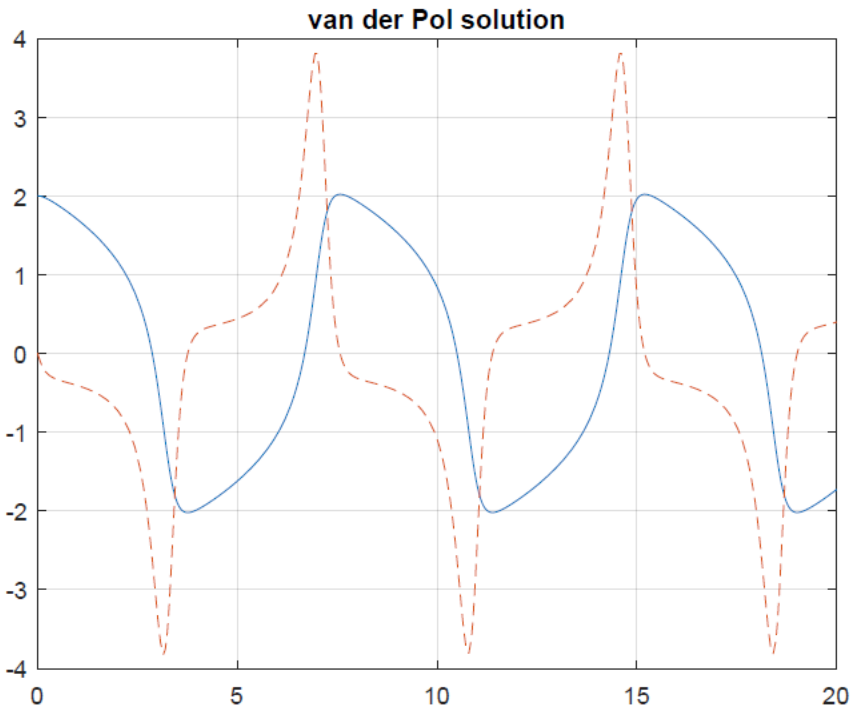
**Example**: Solve the van der Pol equation using ode45 function. For the van der Pol equation, create ODE function file as written below:

```
function ydot= vdpol(t,y)

%VDPOL van der pol equation.

% Ydot= VDPOL(t ,Y)

% Ydot(1) =Y(2)

% Ydot(2) =mu*(1-Y(1)^2)*Y(2)-Y(1)

% mu=2

Mu=2;

Ydot= [ y(2) ; mu*(1-y(1)^2)*y(2)-y(1)];

end
```

Note that the input arguments are t and y, but that this particular function does not use t. Note also that the output ydot must be a column vector.
Given the preceding ODE file, this set of ODEs is solved by using the following commands**:**

```
>>tspan=        [0 20];          % time span to integrate over
>>y0=[2;0];                      % initial conditions(must be column)
>>[t,y]=ode(@vdpol,tspan,y0);
>>size(t)                 % number of time points
Ans=
333     1
>>size(y)
Ans=
333              2
>>plot(t,y(:,1),t,y(:,2),'—')
>>)
>>title(' van der Pol Solutions')
```

Plot is shown on the next page.



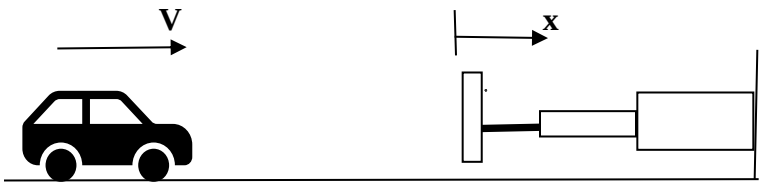

You can try with μ=0.8 and different initial conditions (e.g 0.1, 0 .1, and 2,3). Plot y(1) on x-axis and y(2) on y-axis. What do you conclude about van der Pol equation?

## EXERCISES

1. A safety bumper is placed at the end of a race track to stop out-of-control cars. The bumper is designed such that the force that the bumper applies to the car is a function of the velocity v and the displacement x of the front edge of the bumper according to the equation:

$$F = K v^3 (x+1)^3$$

Where K=30 (s kg)/m $^5$ is a constant.A car with a mass m of 1,500 kg hits the bumper at a speed of 90 km/h. Determine and plot the velocity of the car as a function of its position for $0 \leq x \leq 3$ m.




2. Problem No 35(pertaining to airplane parachute) of chapter 9 of Reference No 1.
3. Problem No 30 of reference No 1 chapter 9.
4. Problem No 31 of Reference No 1 chapter 9.

# EXPERIMENT 4

# SYSTEMS OF EQUATIONS USING NUMERICAL METHODS

**Aim:** To learn various methods for solving systems of equations using numerical methods.

**Equipment needed:** Core 2 duo processor with 1GB RAM and MATLAB software Version 2008a or higher.

**Theory:** Numerical methods are commonly used for solving mathematical problems that are formulated in science and engineering where it is difficult or impossible to obtain exact solutions. MATLAB has a large library of functions for numerically solving a wide variety of mathematical problems. Faculty will demonstrate methods of solving integration problems, finding minimum or maximum of a function

Some useful functions for numerical methods are

| **Command** | **Description** |
|---|---|
| integral | integrate a function |
| feval | evaluate the value of a math function |
| fzero | solves an equation with one variable |
| fminbnd | determines the minimum of a function |
| trapz | used for integrating functions that is given as data points. |

Example: Use numerical integration to calculate the following integral

$$\int_0^8 (xe^{-.8x} + 0.2)\, dx$$

```
>> f=@(x) x.exp(-x.^0.8)+0.2;

>> integral(f,0,8)
Ans=
  3.11604
```

**EXERCISES**

1. The orbit of Pluto is elliptical in shape, with a=5.9065 $\times 10^9$ km and b=5.7208 $\times 10^9$ km. The perimeter of an ellipse can be calculated by

   $$P=4a\int_0^{\frac{\pi}{2}} \sqrt{1 - k^2 \sin\theta^2}\, d\theta$$

2. Solve problem no 19 of chapter 9 of Reference 1.

3. Solve the problem no 20 of Reference No 1.

# EXPERIMENT 5

# ISA PROFILE FOR FLIGHT

**Aim:**

To extract data of atmospheric conditions at different altitudes for aircraft equation of motion.

**Software used:**

MAT LAB

**Formulae:**

$$\frac{T - T_1}{h - h_1} = \frac{dT}{dh} \equiv \alpha$$

| Variables | Gradient layer | Isothermal layer |
|---|---|---|
| Pressure | $\frac{P}{P_1} = \left(\frac{T}{T_1}\right)^{-\frac{g_o}{\alpha R}}$ | $\frac{P}{P_1} = e^{-[g_o/(RT)](h-h_1)}$ |
| Density | $\frac{\rho}{\rho_1} = \left(\frac{T}{T_1}\right)^{-\left(\frac{g_o}{\alpha R}+1\right)}$ | $\frac{\rho}{\rho_1} = e^{-[g_o/(RT)](h-h_1)}$ |

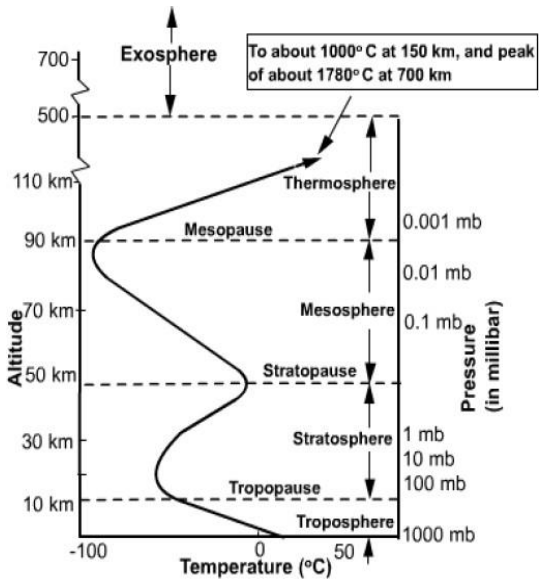



**Fig.2.1 Plot of Altitude versus temperature and pressure**

Lapse rate for 0 m to 11000 m= -0.0065
Lapse rate for 11000 m to 20000 m= 0
Lapse rate for 20000 m to 32000 m= 0.001

# Program code:

```
% ISA PROFILE
% t(1)=288.15;
% x(1)=0;
% i=1;
%  for h=1000:1000:11000
%  x(i+1)=h;
%  lp=-0.0065;
%  alt(i)=x(i+1)-x(i);
%  t(i+1)=t(i)+lp*alt(i);
% i=i+1;
% end
 clc; clear all;
t(1)=288.15;
x(1)=0;i=1;
for h=1000:1000:32000
x(i+1)=h;
 if x(i)>=0 && x(i)<=11000lp=-
0.0065;
alt(i)=x(i+1)-x(i);
t(i+1)=t(i)+lp*alt(i);
elseif x(i)>=11000 &&
 x(i)<=20000
t2(i)=t(end);
lp=0; alt(i)=x(i+1)-x(i);
t(i+1)=t2(i)+lp*alt(i);else
 %disp('a')
t3(i)=t(end);
lp=0.001;
alt(i)=x(i+1)-x(i);
 t(i+1)=t3(i)+lp*alt(i);end
i=i+1; end
plot(t,x)
xlabel('temperature')
ylabel('altitute')
```

## Observations:

Variation of pressure and density with altitude is observed to be an exponential decay in isothermal region. It is observed that aircraft can move from sea level conditions to stratosphere because of decrease in density and pressure.

## Result:

Data of atmospheric conditions at different altitudes for aircraft equation of motion is extracted andplotted.

## Viva questions:

1. What is MATLAB?
2. What is linspace?
3. Write ratio command?
4. Write the syntax for loop?
5. Write the syntax for if, elseif loop?
6. Give formula for pressure variation with altitude?
7. Give formula for density variation with altitude?
8. Give formula for gravity variation with altitude?

# EXPERIMENT 6

## AERODYNAMIC STATIC STABILITY STUDY OF A SYMMETRICAL AIRFOIL

**Aim:**

Carryout aerodynamic static stability study of a symmetrical airfoil and draw a plot for Cm versesangles of attack.

## Materials and Equipments used:

1. Low speed wind tunnel
2. Six component balance
3. Wing model of the symmetrical airfoil
4. Tools like spanner, screw driver
5. Spirit level to measure angle of attack of the tested wing

## Methodology

### A) Formulae:

The longitudinal stability of an aircraft refers to the aircraft's stability in the pitching plane - the plane which describes the position of the aircraft's nose in relation to its tail and the horizon.

$$Cm_0 > 0 \quad \text{.....(1)}$$

$$dCm/d\alpha < 0 \quad \text{.....(2)}$$

# B) Model Description and Drawing:

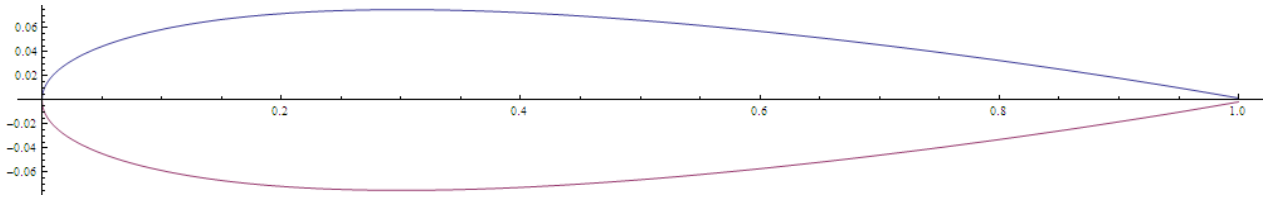



Fig.8.1. Symmetrical Airfoil NACA 0012

2. Aircraft stability ($dC_m/dC_N < 0$ which is the same as $dC_m/d\alpha < 0$). If this value is negative, the airplane is stable.

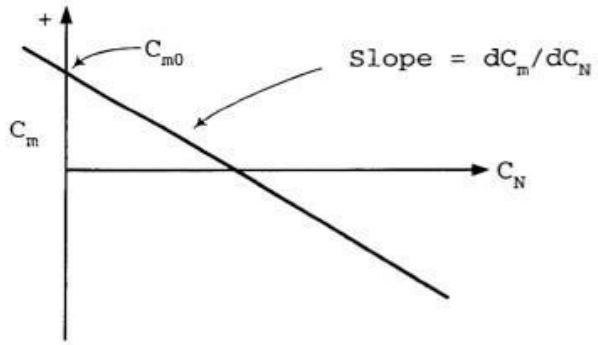



Figure 13-2. $C_m$ versus $C_N$.

Note: The notation $C_{m_{C_N}}$ is equivalent to $dC_m/dC_N$.

If a disturbance causes a lower $C_N$ (lower $\alpha$) then it also causes a positive $C_m$ which increases $C_N$ (and $\alpha$) and vice versa.

Fig.8.2. Aircraft Stability Criteria

## C) Procedure:

1. Fix the model inside the test section and make sure the angle of attack (AOA) is as per requirement.
2. Fix the Six component balance as per given condition and connect with power plugs.
3. Make it zero setting.
4 See and ensure safety of Wind Tunnel and make sure that no foreign object debris (FOD) are present.
5. Note the initial value of inclined tube manometer (hi).
6. Start the Wind Tunnel slowly and run for few minutes in that slow speed.

7. Now increase the Wind Tunnel RPM as per required speed in steps and should stabilize in that speed.
8. Observed the reading in Six Component display and note it down the Cm values.
9. Find the moment and dividing ½ ρ $V^2$ S c. This will give Cm that is coefficient of moment.
10. Prepare the table and fill the entire column.
11. Plot a graph the Cm verses angle of attack.
12. Find the maximum Cm by drawing tangent from the vertical axis and get the corresponding value ofAOA where it cuts the x axis called trim.
13. This AOA is the required AOA for trimmed condition.

**Result and discussion:**

Write the physics behind the result. Write and discuss the importance of the result. Give significant pointof interest and your logic behind the result.

**Conclusions:**

In this write the result main point obtained. How it can be applied for real scenario?

# EXPERIMENT -7

# Stability analysis using Root locus, Bode plot techniques

**Aim:** Write a code for stability analysis using root locus and bode plot techniques.

**Equipment needed:** Core 2 duo processor with 1GB RAM and MATLAB software Version 2008a or higher or Sci-lab.

**Theory:**

The **root locus technique in control system** was first introduced in the year 1948 by Evans. Any physical system is represented by a transfer function in the form of

$$G(s) = k \times \frac{numerator\ of\ s}{denomerator\ of\ s}$$

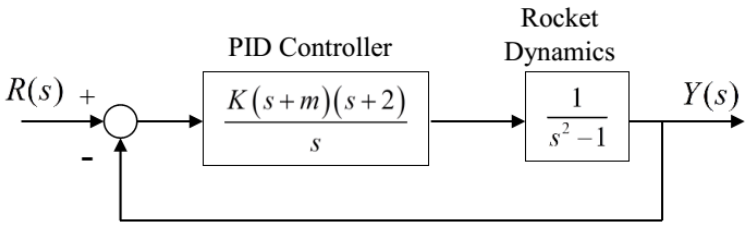
We can find poles and zeros from G(s). The location of poles and zeros are crucial keeping view stability, relative stability, transient response and error analysis. When the system is put to service stray inductance and capacitance get into the system, thus changes the location of poles and zeros. In **root locus technique in control system** we will evaluate the position of the roots, their locus of movement and associated information. These information will be used to comment upon the system performance. Now before I introduce what is a root locus technique, it is very essential here to discuss a few of the advantages of this technique over other stability criteria.

## Procedure to Plot Root Locus

Keeping all these points in mind we are able to draw the **root locus plot** for any kind of system. Now let us discuss the procedure of making a root locus.

1. Find out all the roots and poles from the open loop transfer function and then plot them on the complex plane.
2. All the root loci starts from the poles where k = 0 and terminates at the zeros where K tends to infinity. The number of branches terminating at infinity equals to the difference between the number of poles & number of zeros of G(s)H(s).
3. Find the region of existence of the root loci from the method described above after finding the values of M and N.
4. Calculate break away points and break in points if any.
5. Plot the asymptotes and centroid point on the complex plane for the root loci by calculating the slope of the asymptotes.

6. Now calculate angle of departure and the intersection of root loci with imaginary axis.
7. Now determine the value of K by using any one method that I have described above. By following above procedure you can easily draw the **root locus plot** for any open loop transfer function.
8. Calculate the gain margin.
9. Calculate the phase margin.
10. You can easily comment on the stability of the system by using Routh Array.

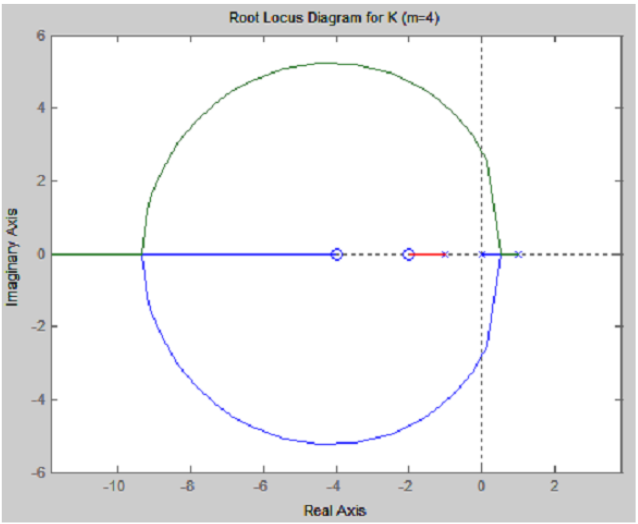



The characteristic equation of the closed-loop system is $1+GH(s)=0$ or $1+KP(s)=0$. Substituting the transfer functions from the block diagram gives

$$1+K\left[\frac{(s+4)(s+2)}{s(s^2-1)}\right]=1+K\left[\frac{s^2+6s+8}{s^3-s}\right]=0$$

The MATLAB commands that produce the root locus diagram are:

```
>> num=[1,6,8];
>> den=[1,0,-1,0];
>> sys=tf(num,den)

Transfer function:
        s^2 + 6 s + 8
        -------------
          s^3 - s

>> rlocus(sys)
>> axis('equal')
>> title('Root Locus Diagram for K (m=4)')
```




## EXERCISES

1. Draw bode plot of same system .

# EXPERIMENT -8

# Design of lead, lag and lead –lag compensator

**Aim:** Write a code to design of lead, lag and lead –lag compensator.

**Equipment needed:** Core 2 duo processor with 1GB RAM and MATLAB software Version 2008a or higher or Sci-lab.

## Theory:

Lead and lag compensators are used quite extensively in control. A lead compensator can increase the stability or speed of reponse of a system; a lag compensator can reduce (but not eliminate) the steady-state error. Depending on the effect desired, one or more lead and lag compensators may be used in various combinations.

Lead, lag, and lead/lag compensators are usually designed for a system in transfer function form.

## Lead or phase-lead compensator using root locus

A first-order lead compensator *C*(*s*) can be designed using the root locus. A lead compensator in root locus form is given by

$$C(s) = K_c \frac{(s - z_0)}{(s - p_0)} \quad (1)$$

where the magnitude of *z0* is less than the magnitude of *p0*. A phase-lead compensator tends to shift the root locus toward to the left in the complex *s*-plane. This results in an improvement in the system's stability and an increase in its response speed.
How is this accomplished? If you recall finding the asymptotes of the root locus that lead to the zeros at infinity, the equation to determine the intersection of the asymptotes along the real axis is the following.

$$a = \frac{\sum(poles) - \sum(zeros)}{(\#poles) - (\#zeros)} \quad (2)$$

When a lead compensator is added to a system, the value of this intersection will be a larger negative number than it was before. The net number of zeros and poles will be same (one zero and one pole are added), but the added pole is a larger negative number than the added zero. Thus, the result of a lead compensator is that the asymptotes' intersection is moved further to the left in the complex plane, and the entire root locus is shifted to the left as well. This tends to increase the region of stability and the system's response speed.

In MATLAB a phase-lead compensator in root locus form is implemented using the following commands (where `Kc`, `z`, and `p` are defined).

```
s = tf('s');

C_lead = Kc*(s-z)/(s-p);
```

We can interconnect this compensator *C*(*s*) with a plant *P*(*s*) using the following code.

```
sys_ol = C_lead*P
```

# Lead or phase-lead compensator using frequency response

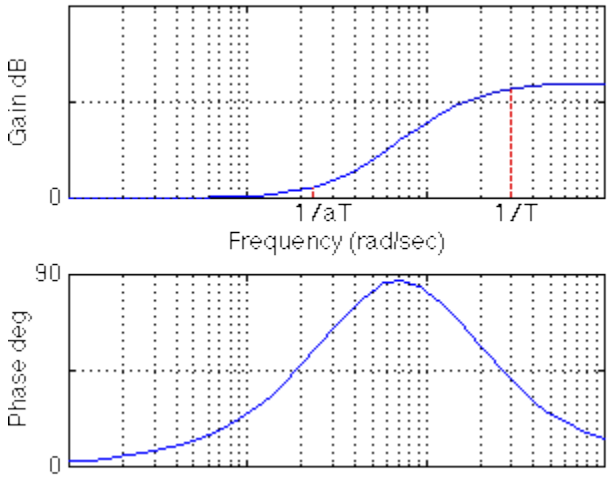
A first-order phase-lead compensator can also be designed using a frequency reponse approach. A lead compensator in frequency response form is given by the following.

$$C(s) = \frac{1 + aTs}{1 + Ts} \qquad [a > 1] \tag{3}$$

Note that this is equivalent to the root locus form repeated below

$$C(s) = K_c \frac{(s - z_0)}{(s - p_0)} \tag{4}$$

with $p$ =1 / $T$, $z$ = 1 / $aT$, and $Kc$ = $a$. In frequency response design, the phase-lead compensator adds positive phase to the system over the frequency range 1 / $aT$ to 1 / $T$. A Bode plot of a phase-lead compensator $C(s)$ has the following form.




The two corner frequencies are at 1 / $aT$ and 1 / $T$; note the positive phase that is added to the system between these two frequencies. Depending on the value of $a$, the maximum added phase can be up to 90 degrees; if you need more than 90 degrees of phase, two lead compensators in series can be employed. The maximum amount of phase is added at the center frequency, which is calculated according to the following equation.

$$\omega_m = \frac{1}{T\sqrt{a}} \tag{5}$$

The equation which determines the maximum phase is given below.

$$\sin\phi = \frac{a - 1}{a + 1} \tag{6}$$

Additional positive phase increases the phase margin and thus increases the stability of the system. This type of compensator is designed by determining $a$ from the amount of phase needed to satisfy the phase margin requirements, and determing $T$ to place the added phase at the new gain-crossover frequency.

Another effect of the lead compensator can be seen in the magnitude plot. The lead compensator increases the gain of the system at high frequencies (the amount of this gain is equal to $a$). This can increase the crossover frequency, which will help to decrease the rise time and settling time of the system (but may amplify high frequency noise).

In MATLAB, a phase-lead compensator $C(s)$ in frequency response form is implemented using the following code (where `a` and `T` are defined).

```
s = tf('s');

C_lead = (1+a*T*s)/(1+T*s);
```

We can then interconnect it with a plant *P*(*s*) using the following code.

```
sys_ol = C_lead*P;
```

## Lag or phase-lag compensator using root locus

A first-order lag compensator *C*(*s*) can be designed using the root locus. A lag compensator in root locus form is given by the following.

$$C(s) = \frac{(s - z_0)}{(s - p_0)} \qquad (7)$$

This has a similar form to a lead compensator, except now the magnitude of *z0* is greater than the magnitude of *p0* (and the additional gain *Kc* is omitted). A phase-lag compensator tends to shift the root locus to the right in the complex *s*-plane, which is undesirable. For this reason, the pole and zero of a lag compensator are often placed close together (usually near the origin) so that they do not appreciably change the transient response or stability characteristics of the system.

How does the lag controller shift the root locus to the right? Below is repeated the equation for finding where the asymptotes of the root locus intersect along the real axis.

$$a = \frac{\sum(poles) - \sum(zeros)}{(\#poles) - (\#zeros)} \qquad (8)$$

When a lag compensator is added to a system, the value of this intersection will be a smaller negative number than it was before. The net number of zeros and poles will be the same (one zero and one pole are added), but the added pole is a smaller negative number than the added zero. Thus, the result of a lag compensator is that the asymptotes' intersection is moved to the right in the complex plane, and the entire root locus is shifted to the right as well.

It was previously stated that a lag compensator is often designed to minimally change the transient response of system because it generally has a negative effect. If the phase-lag compensator is not supposed to change the transient response noticeably, what is it good for then? The answer is that a phase-lag compensator can improve the system's steady-state response. It works in the following manner. At high frequencies, the lag compensator will have unity gain. At low frequencies, the gain will be *z0* / *p0* which is greater than 1. This *z0* / *p0* factor will multiply the position, velocity, or acceleration constant (*Kp*, *Kv*, or *Ka*), and the steady-state error will thus decrease by the same factor.

In MATLAB, a phase-lag compensator *C*(*s*) in root locus form is implemented by employing the following code where it is again assumed that `z` and `p` are previously defined.

```
s = tf('s');

C_lag = (s-z)/(s-p);
```

We can also interconnect the compensator with a plant *P*(s) as follows.

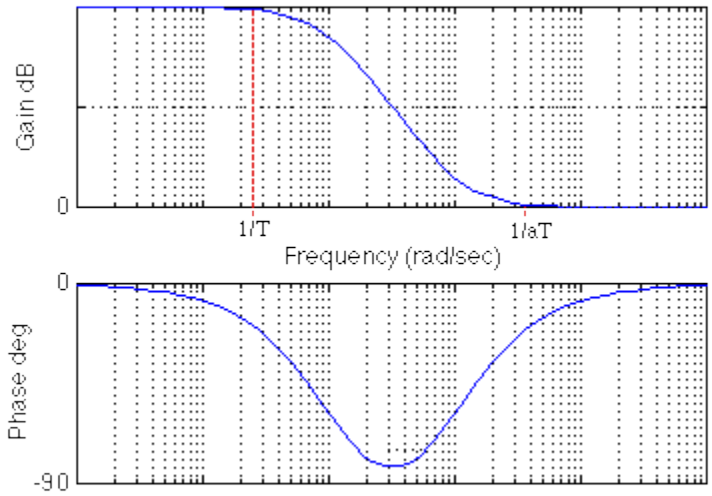
```
sys_ol = C_lag*P;
```

## Lag or phase-lag compensator using frequency response

A first-order phase-lag compensator also can be designed using a frequency response approach. A lag compensator in frequency response form is given by the following.

$$C(s) = \frac{1}{a}\left(\frac{1+aTs}{1+Ts}\right) \qquad [a<1] \tag{9}$$

The phase-lag compensator looks similar to phase-lead compensator, except that *a* is now less than 1. The main difference is that the lag compensator adds negative phase to the system over the specified frequency range, while a lead compensator adds positive phase over the specified frequency. A Bode plot of a phase-lag compensator has the following form.




The two corner frequencies are at 1 / *T* and 1 / *aT*. The main effect of the lag compensator is shown in the magnitude plot. The lag compensator adds gain at low frequencies; the magnitude of this gain is equal to *a*. The effect of this gain is to cause the steady-state error of the closed-loop system to be decreased by a factor of *a*. Because the gain of the lag compensator is unity at middle and high frequencies, the transient response and stability are generally not impacted much.

The side effect of the lag compensator is the negative phase that is added to the system between the two corner frequencies. Depending on the value *a*, up to -90 degrees of phase can be added. Care must be taken that the phase margin of the system with lag compensation is still satisfactory. This is generally achieved by placing the frequency of maximum phase lag, *wm* as calculated below, well below the new gain crossover frequency.

$$\omega_m = \frac{1}{T\sqrt{a}} \tag{10}$$

In MATLAB, a phase-lag compensator *C*(*s*) in frequency response form is implemented using the following code, again assuming that `a` and `T` are defined.

```
s = tf('s');
```

```
C_lag = (a*T*s+1)/(a*(T*s+1));
```

We can again interconnect the compensator with a plant *P*(s) as follows.

```
sys_ol = C_lag*P;
```

## Lead-lag compensator using either root locus or frequency response

A lead-lag compensator combines the effects of a lead compensator with those of a lag compensator. The result is a system with improved transient response, stability, and steady-state error. To implement a lead-lag compensator, first design the lead compensator to achieve the desired transient response and stability, and then design a lag compensator to improve the steady-state response of the lead-compensated system.

# EXPERIMENT -9

# Introduction to Standard Simulink libraries, Building Simulink linear models: transfer function

**Aim:** Basic introduction of simulation libraries and linear model for transfer function.

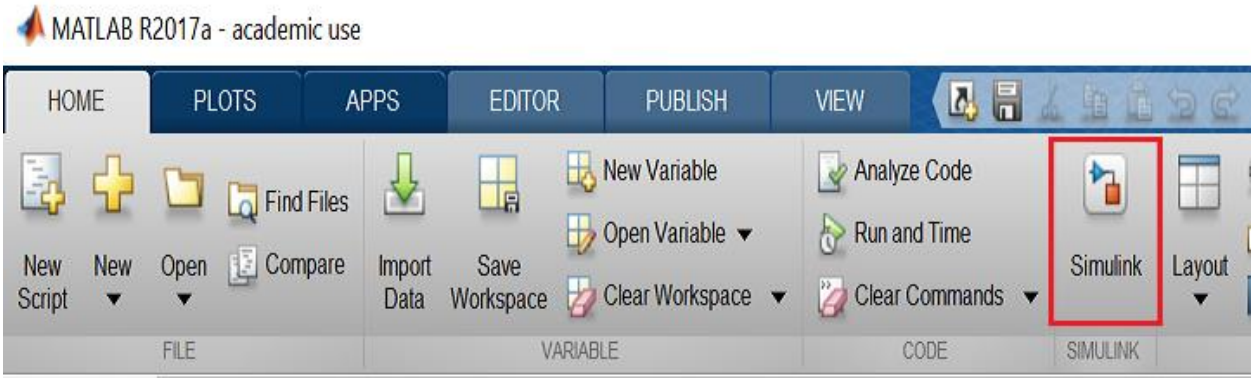
**Equipment needed:** Core 2 duo processor with 1GB RAM and MATLAB software Version 2008a or higher or Sci-lab.

## Theory:

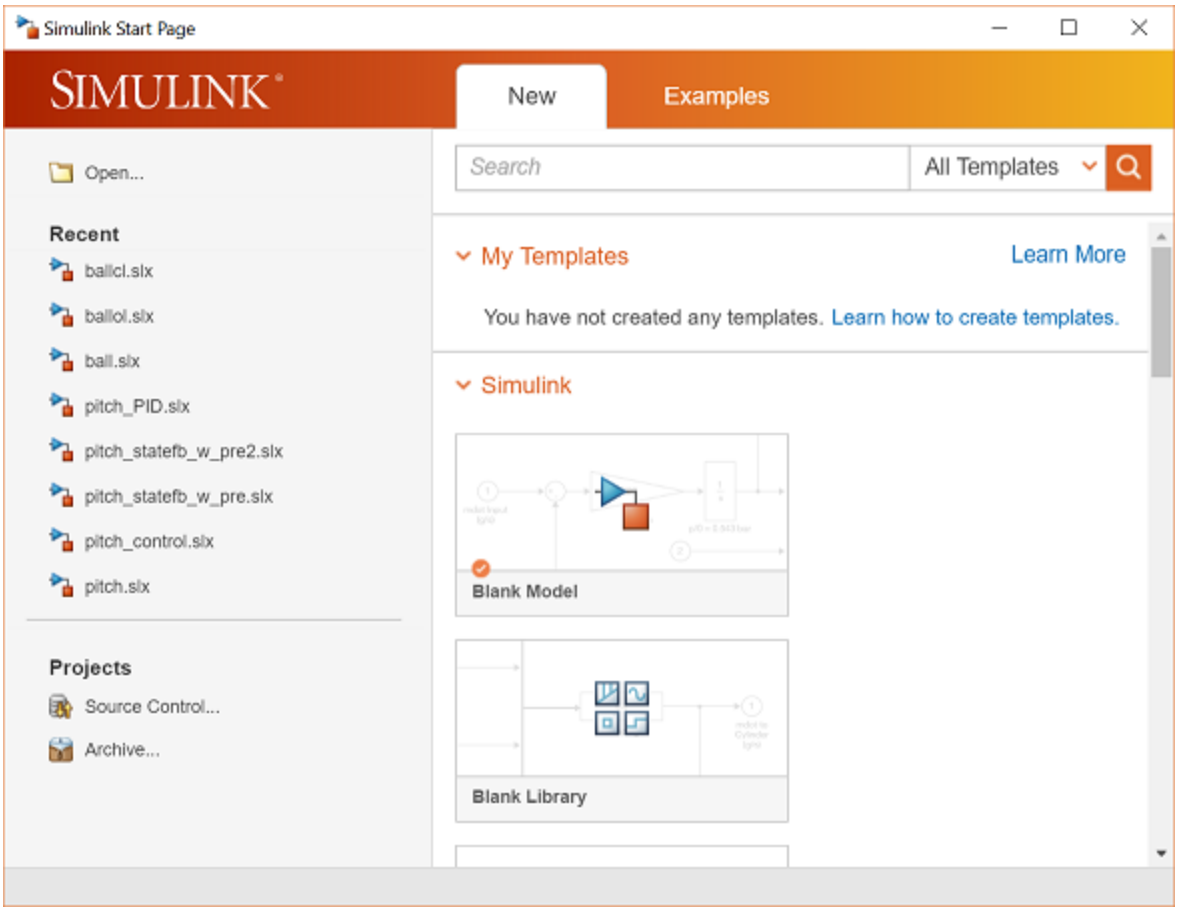
Simulink is started from the MATLAB command prompt by entering the following command:

```
simulink
```

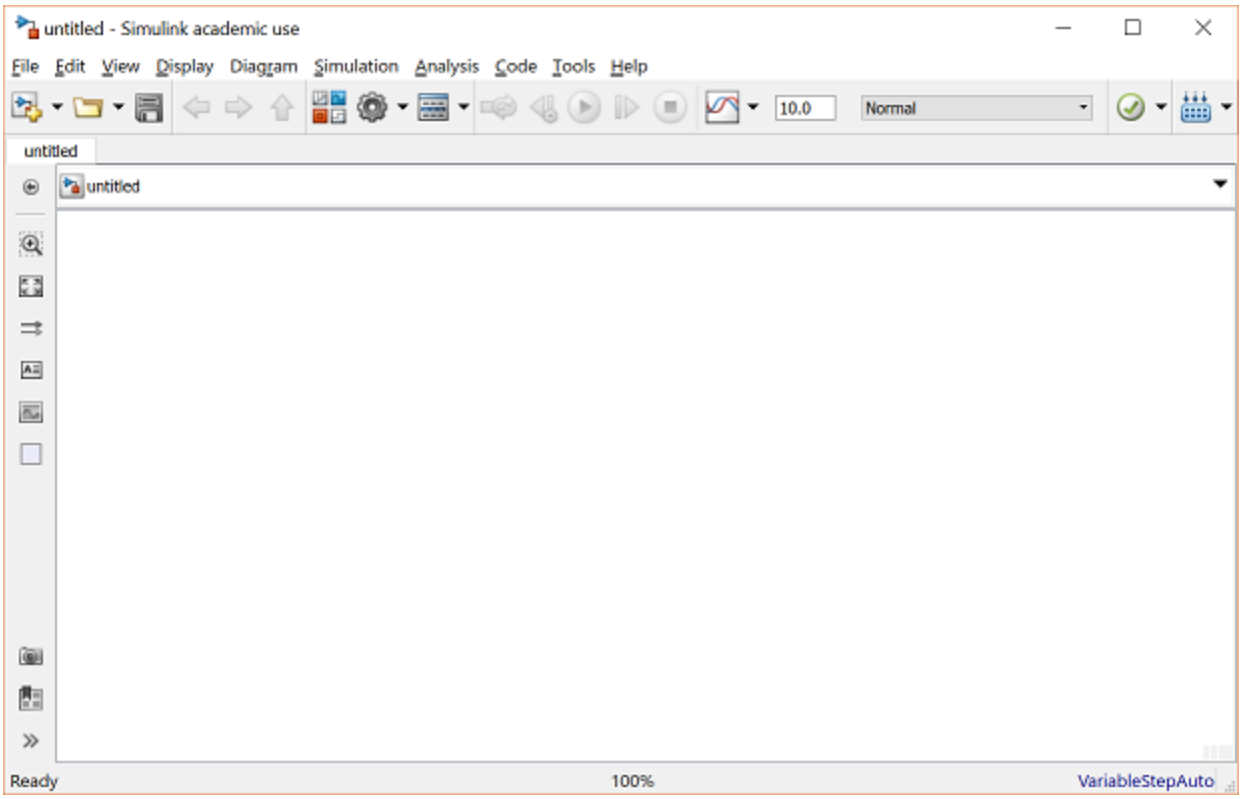
Alternatively, you can hit the **Simulink** button at the top of the MATLAB window as shown here:




When it starts, Simulink brings up a single window, entitled **Simulink Start Page** which can be seen here.




Once you click on **Blank Model**, a new window will appear as shown below.




# Model Files

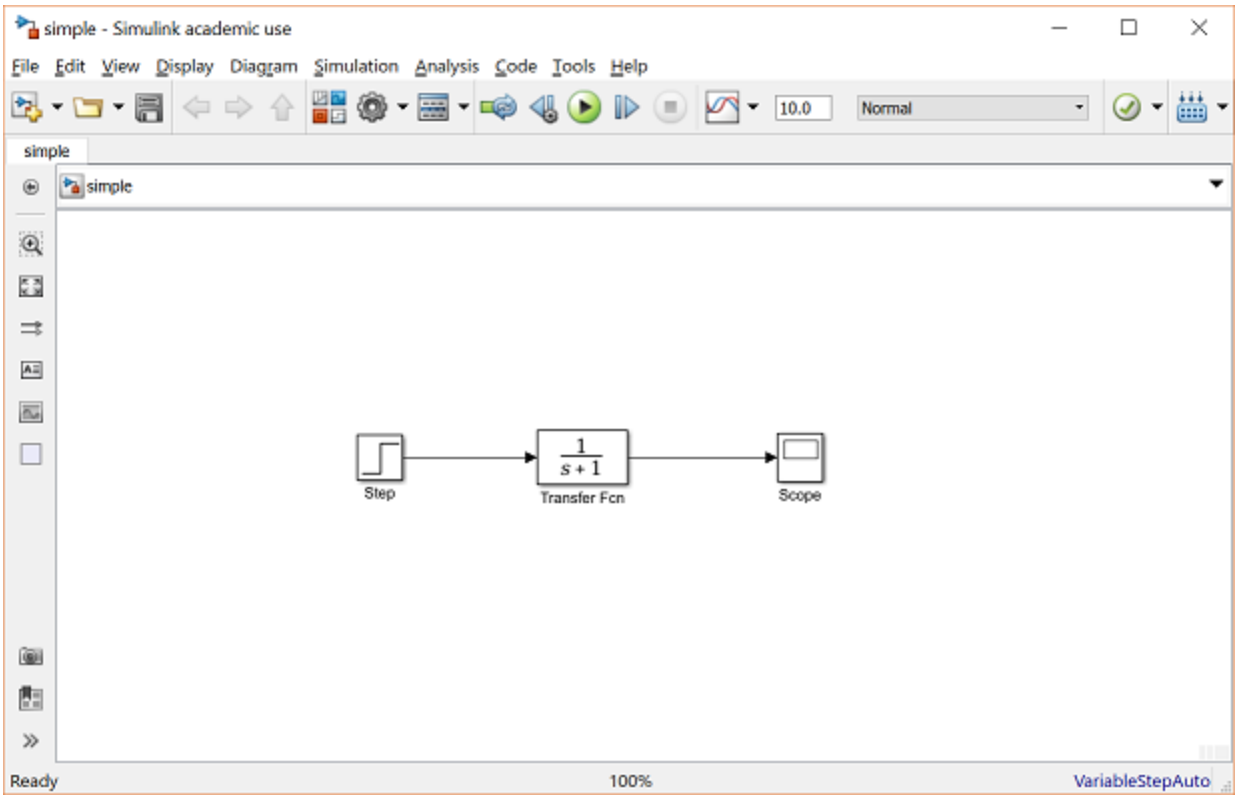
In Simulink, a model is a collection of blocks which, in general, represents a system. In addition to creating a model from scratch, previously saved model files can be loaded either from the **File** menu or from the MATLAB command prompt. As an example, download the following model file by right-clicking on the following link and saving the file in the directory you are running MATLAB from.

simple.slx

Open this file in Simulink by entering the following command in the MATLAB command window. (Alternatively, you can load this file using the **Open** option in the **File** menu in Simulink, or by hitting **Ctrl-O** in Simulink).

```
simple
```

The following model window should appear.




A new model can be created by selecting **New** from the **File** menu in any Simulink window (or by hitting **Ctrl-N**).

# Basic Elements

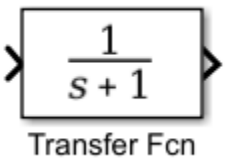
There are two major classes of items in Simulink: **blocks** and **lines**. Blocks are used to generate, modify, combine, output, and display signals. Lines are used to transfer signals from one block to another.

**Blocks**

There are several general classes of blocks within the Simulink library:

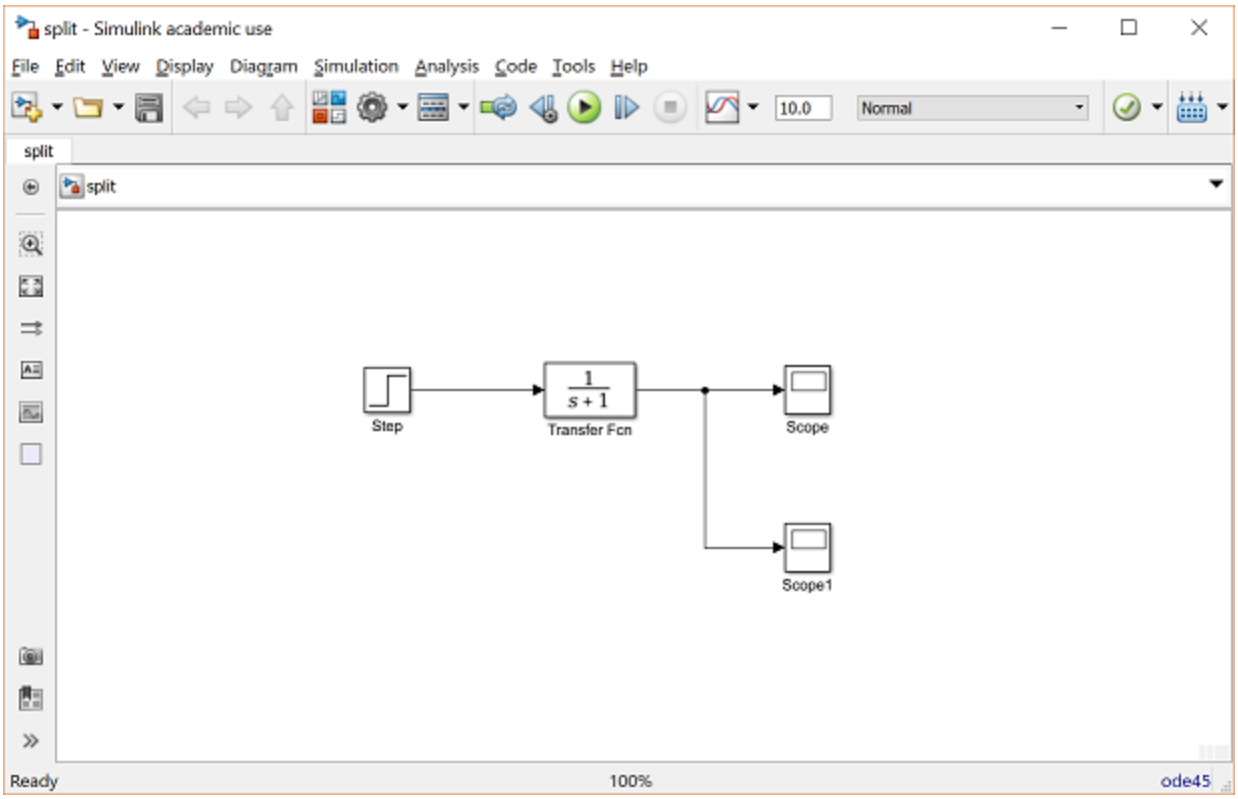
- Sources: used to generate various signals
- Sinks: used to output or display signals
- Continuous: continuous-time system elements (transfer functions, state-space models, PID controllers, etc.)
- Discrete: linear, discrete-time system elements (discrete transfer functions, discrete state-space models, etc.)
- Math Operations: contains many common math operations (gain, sum, product, absolute value, etc.)
- Ports & Subsystems: contains useful blocks to build a system

Blocks have zero to several input terminals and zero to several output terminals. Unused input terminals are indicated by a small open triangle. Unused output terminals are indicated by a small triangular point. The block shown below has an unused input terminal on the left and an unused output terminal on the right.



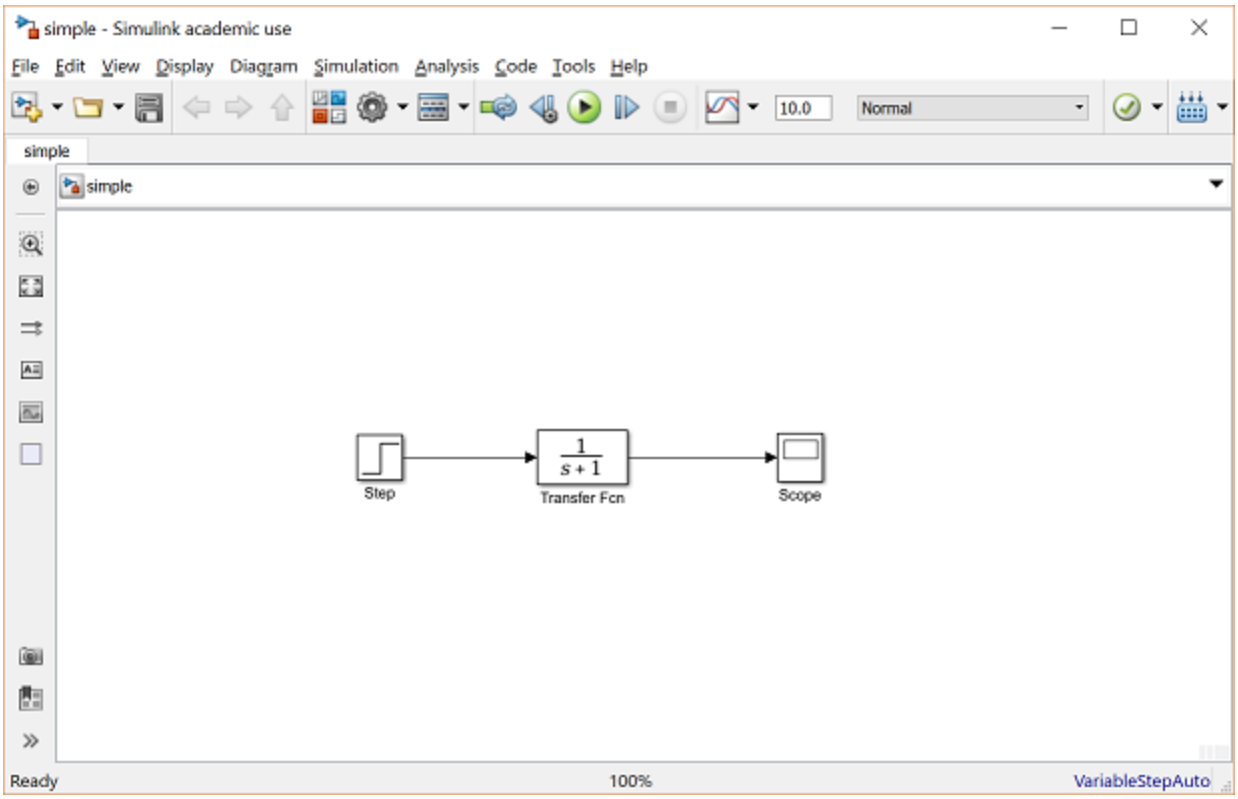

**Lines**

Lines transmit signals in the direction indicated by the arrow. Lines must always transmit signals from the output terminal of one block to the input terminal of another block. On exception to this is a line can tap off of another line, splitting the signal to each of two destination blocks, as shown below (right-click here and then select **Save link as ...** to download the model file called split.slx).




Lines can never inject a signal *into* another line; lines must be combined through the use of a block such as a summing junction.
A signal can be either a scalar signal or a vector signal. For Single-Input, Single-Output (SISO) systems, scalar signals are generally used. For Multi-Input, Multi-Output (MIMO) systems, vector signals are often used, consisting of two or more scalar signals. The lines used to transmit scalar and vector signals are identical. The type of signal carried by a line is determined by the blocks on either end of the line.

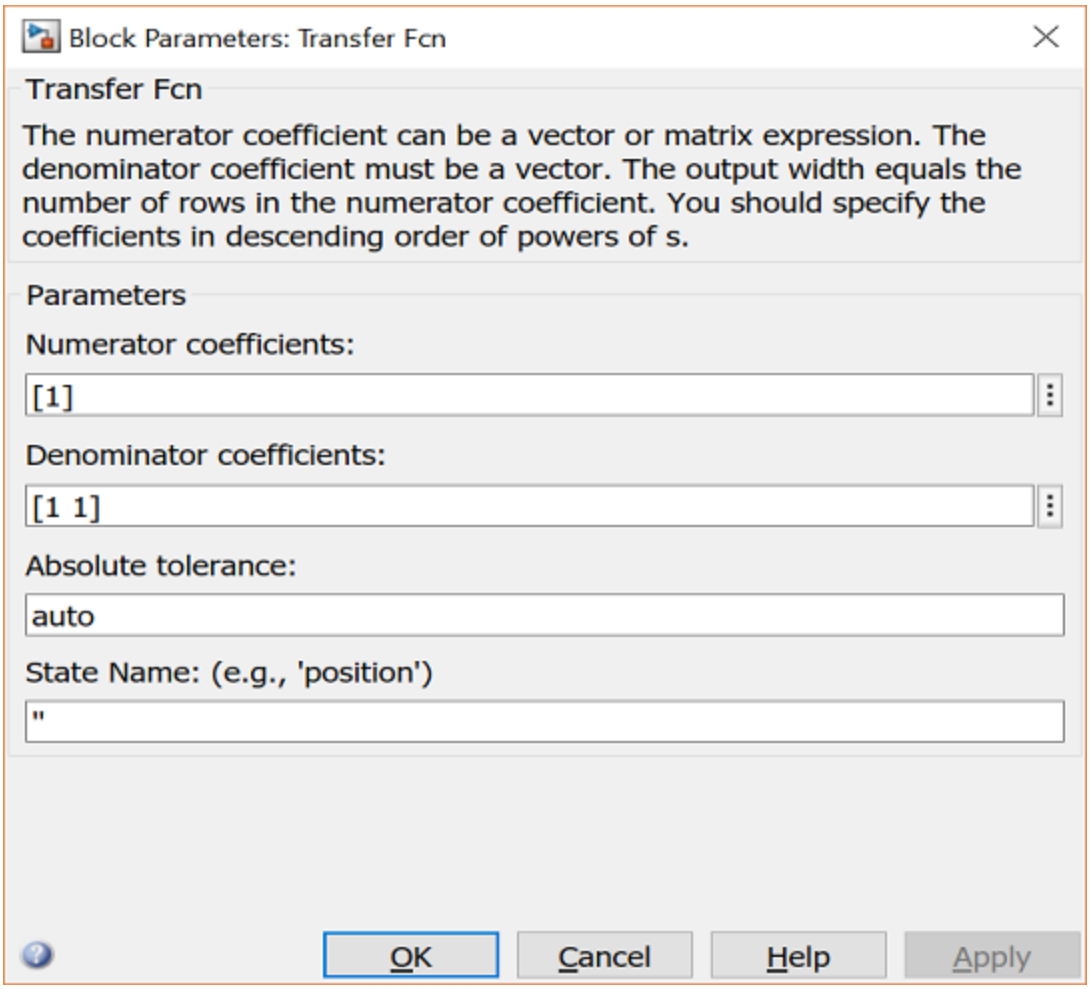
## Simple Example




The `simple` model consists of three blocks: Step, Transfer Function, and Scope. The Step is a **Source** block from which a step input signal originates. This signal is transferred through the **line** in the direction indicated by the arrow to the *Transfer Function* **Continuous** block. The *Transfer Function* block modifies its input signal and outputs a new signal on a line to the *Scope*. The *Scope* is a **Sink** block used to display a signal much like an oscilloscope.
There are many more types of blocks available in Simulink, some of which will be discussed later. Right now, we will examine just the three we have used in the `simple` model.

**Modifying Blocks**

A block can be modified by double-clicking on it. For example, if you double-click on the *Transfer Function* block in the `Simple` model, you will see the following dialog box.



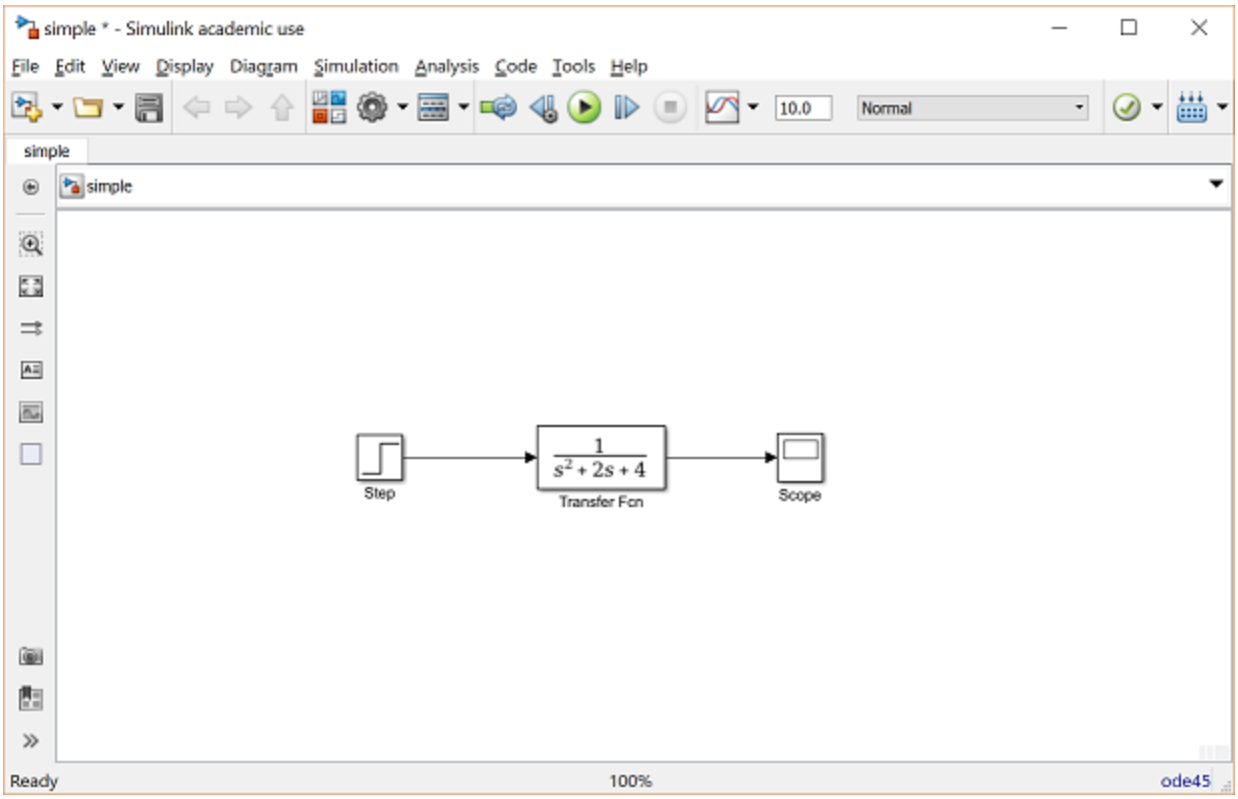

This dialog box contains fields for the numerator and the denominator of the block's transfer function. By entering a vector containing the coefficients of the desired numerator or denominator polynomial, the desired transfer function can be entered. For example, to change the denominator to

(1) $s^2 + 2s + 4$

enter the following into the denominator field

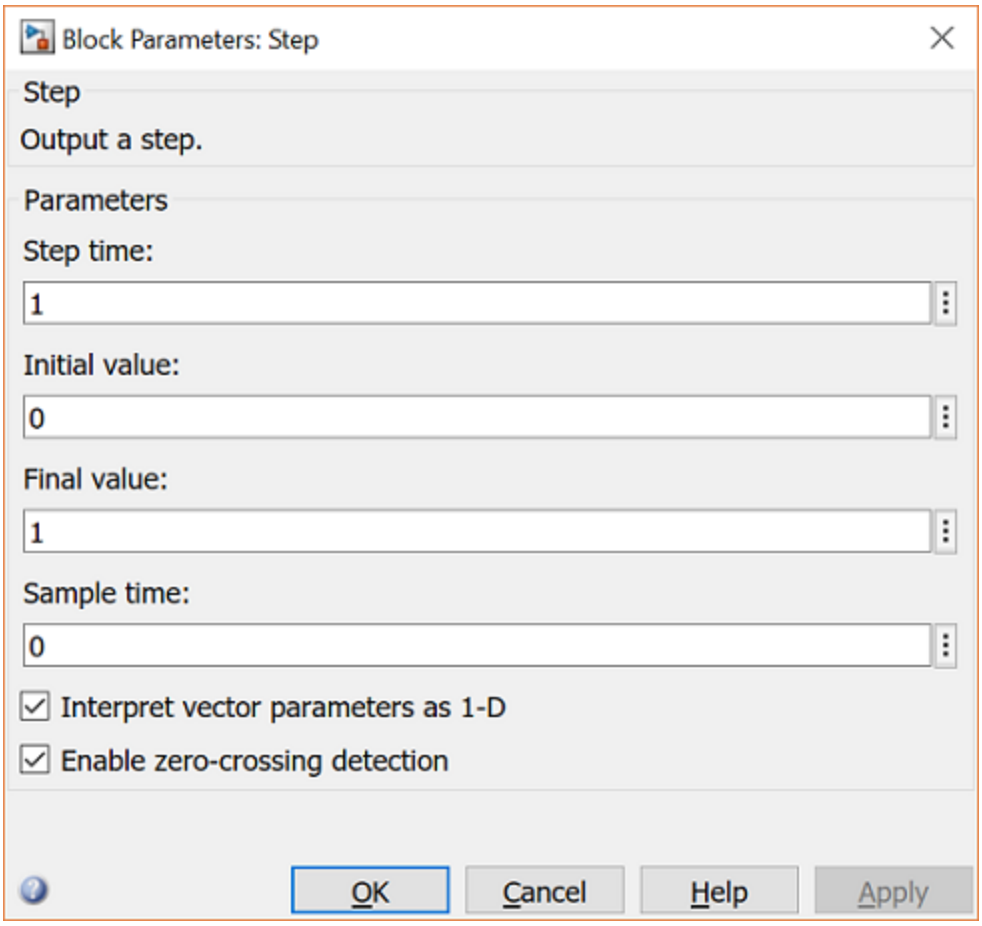
```
[1 2 4]
```

and hit the close button, the model window will change to the following,



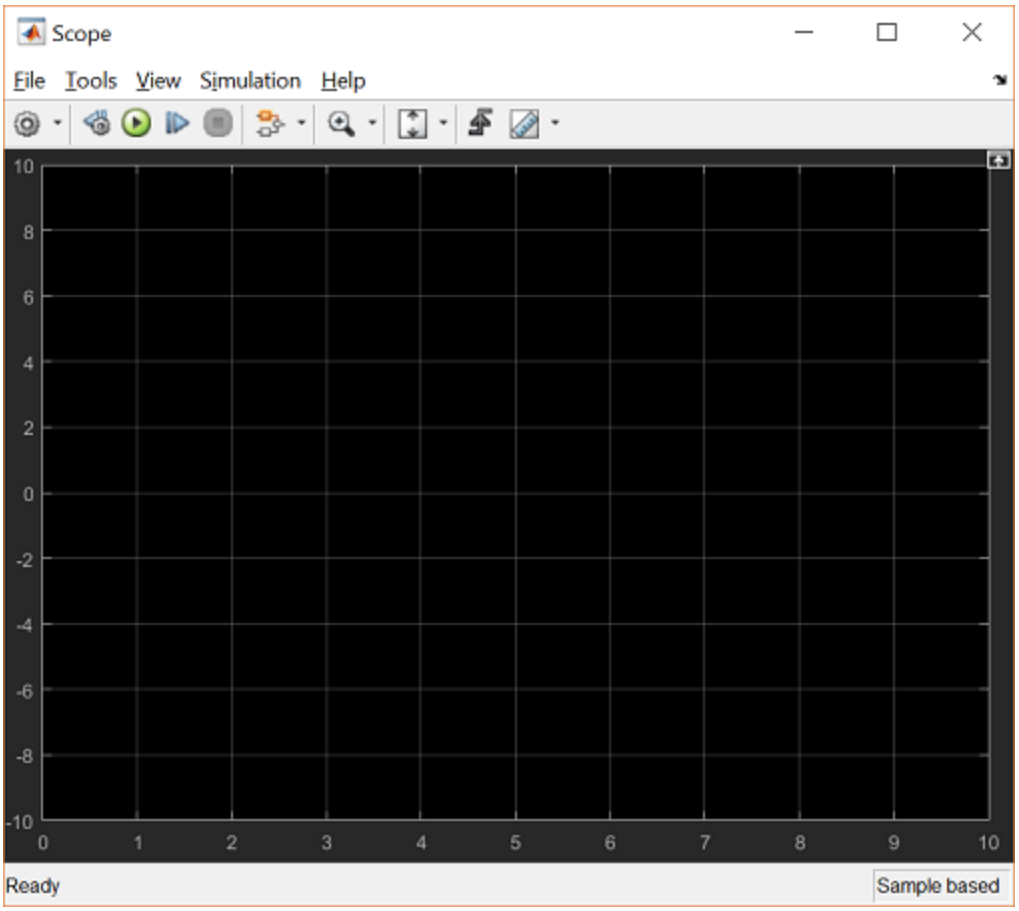

which reflects the change in the denominator of the transfer function.

The *Step* block can also be double-clicked, bringing up the following dialog box.




The default parameters in this dialog box generate a step function occurring at time = 1 sec, from an initial level of zero to a level of 1 (in other words, a unit step at t = 1). Each of these parameters can be changed. Close this dialog before continuing.

The most complicated of these three blocks in the *Scope* block. Double-clicking on this brings up a blank oscilloscope screen.

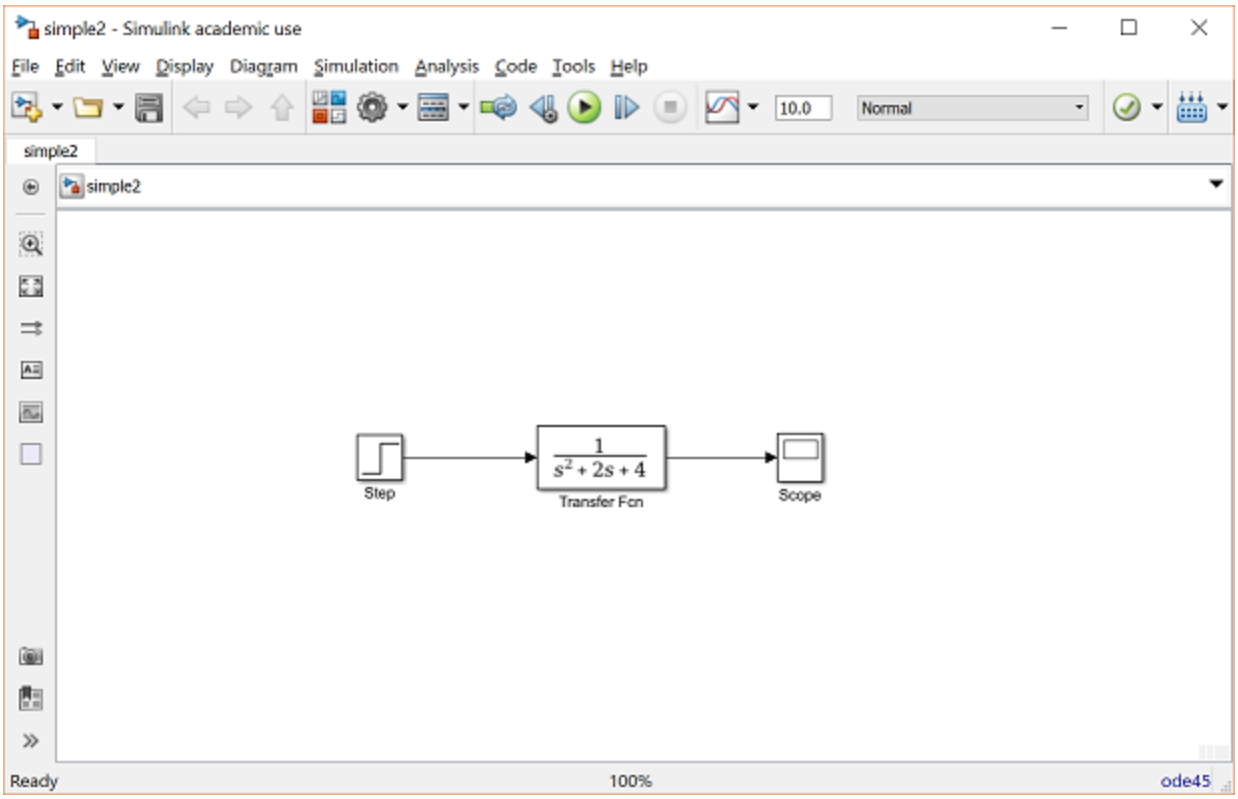



When a simulation is performed, the signal which feeds into the scope will be displayed in this window. Detailed operation of the scope will not be covered in this tutorial.

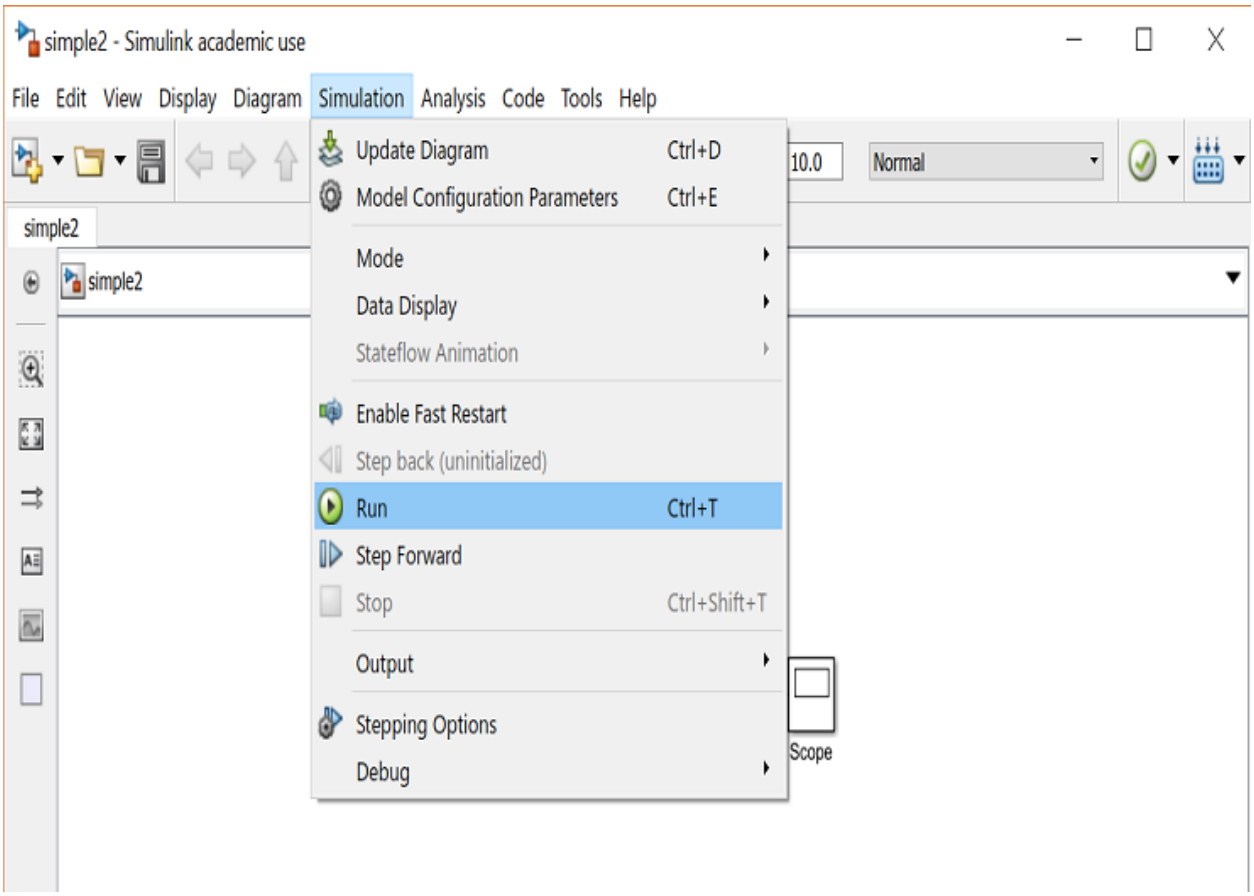
## Running Simulations

To run a simulation, we will work with the following model file:

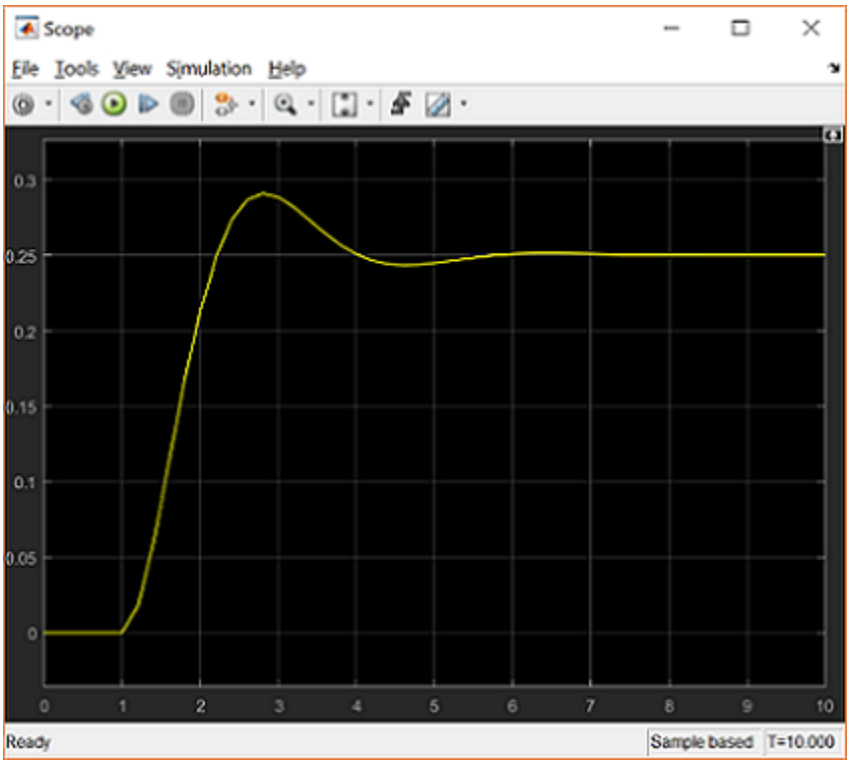
simple2.slx (right-click and then select **Save link as ...**)
Download and open this file in Simulink following the previous instructions for this file. You should see the following model window.




Before running a simulation of this system, first open the scope window by double-clicking on the scope block. Then, to start the simulation, either select **Run** from the **Simulation** menu, click the **Play** button at the top of the screen, or hit **Ctrl-T**.



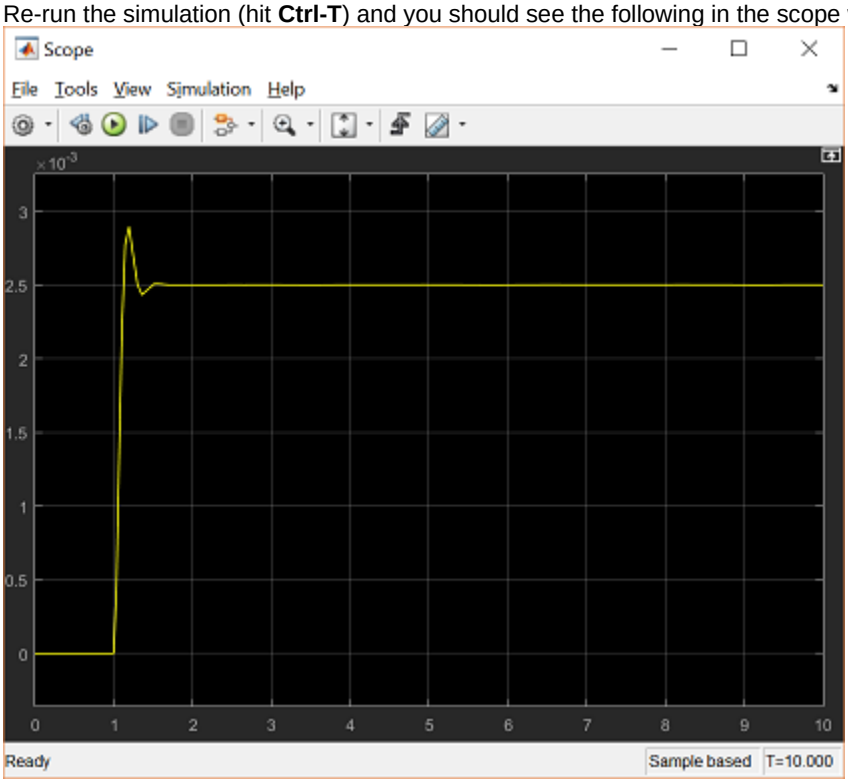

The simulation should run very quickly and the scope window will appear as shown below.



Note that the step response does not begin until t = 1. This can be changed by double-clicking on the *step* block. Now, we will change the parameters of the system and simulate the system again. Double-click on the *Transfer Function* block in the model window and change the denominator to:

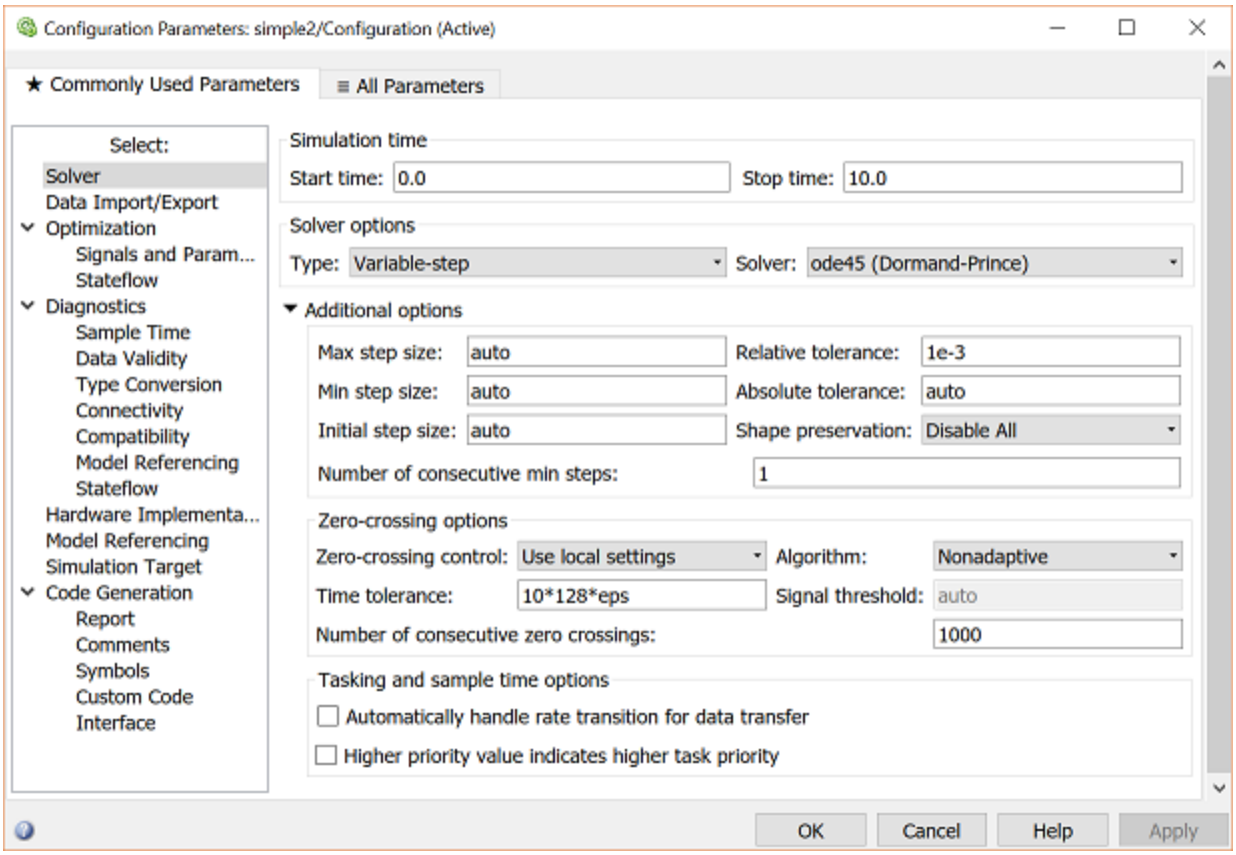
```
[1 20 400]
```

Re-run the simulation (hit **Ctrl-T**) and you should see the following in the scope window.

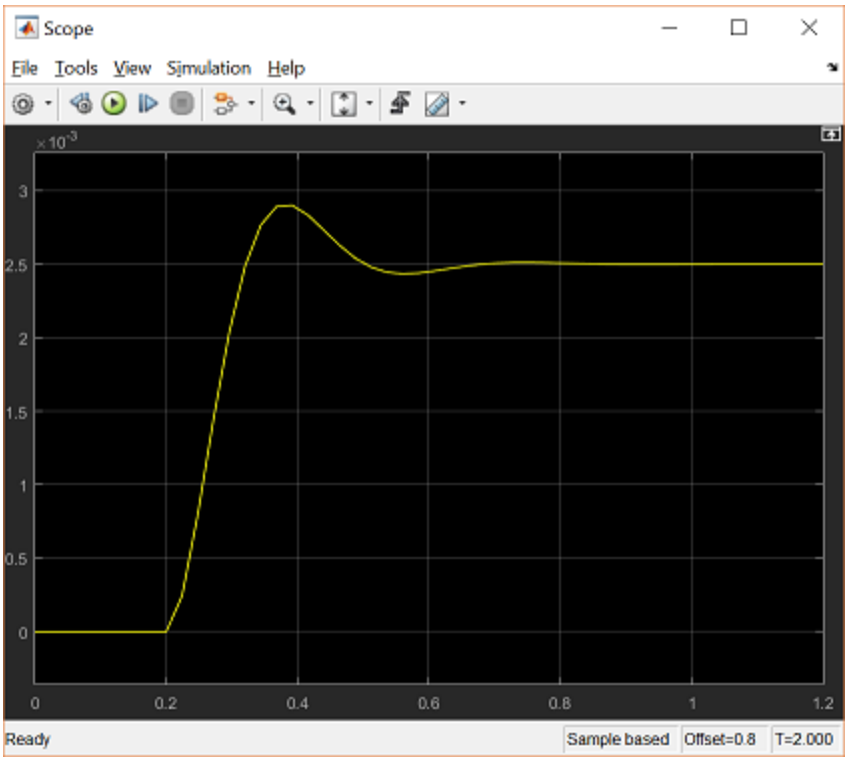



Since the new transfer function has a very fast response, it compressed into a very narrow part of the scope window. This is not really a problem with the scope, but with the simulation itself. Simulink simulated the system for a full ten seconds even though the system had reached steady state shortly after one second.

To correct this, you need to change the parameters of the simulation itself. In the model window, select **Model Configuration Parameters** from the **Simulation** menu. You will see the following dialog box.

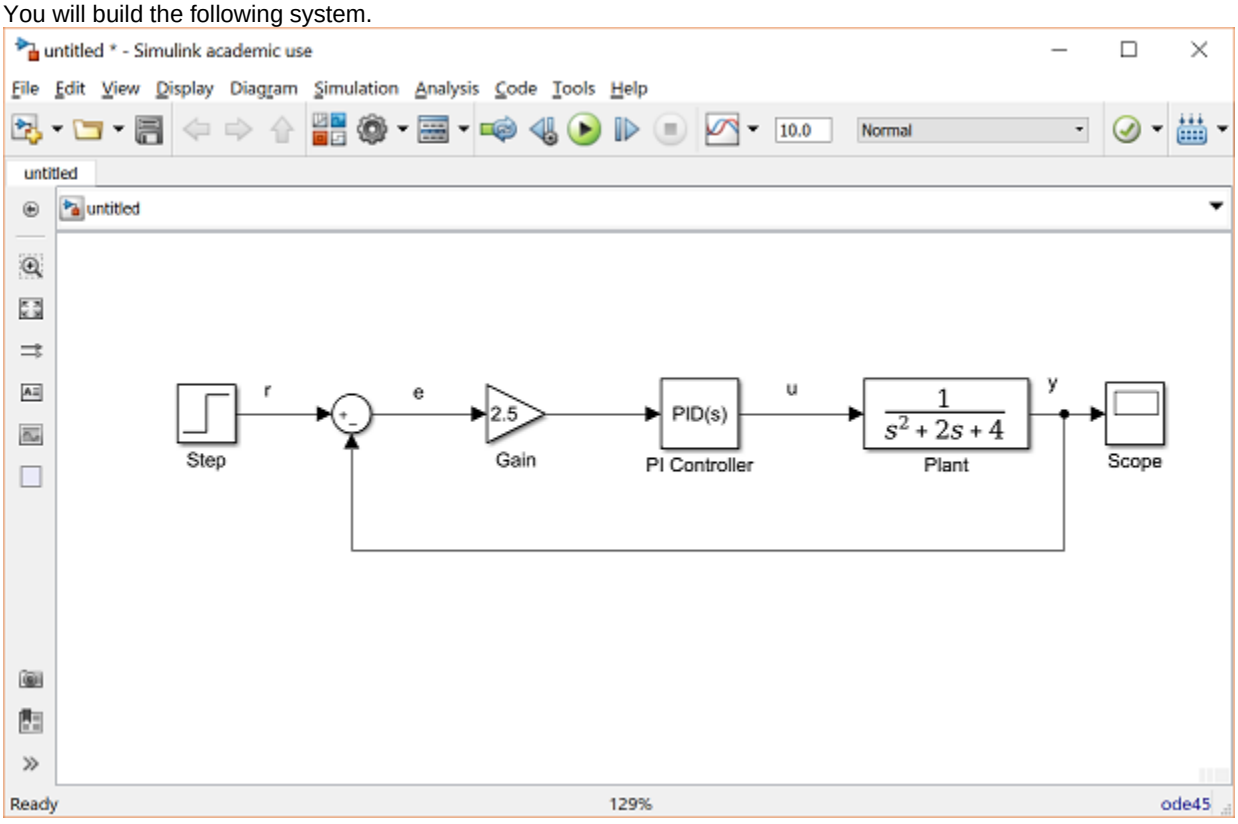



There are many simulation parameter options; we will only be concerned with the start and stop times, which tell Simulink over what time period to perform the simulation. Change **Start time** from 0.0 to 0.8 (since the step doesn't occur until t = 1.0). Change **Stop time** from 10.0 to 2.0, which should be only shortly after the system settles. Close the dialog box and rerun the simulation. Now, the scope window should provide a much better display of the step response as shown below.




## Building Systems

In this section, you will learn how to build systems in Simulink using the building blocks in Simulink's **Block Libraries**. You will build the following system.

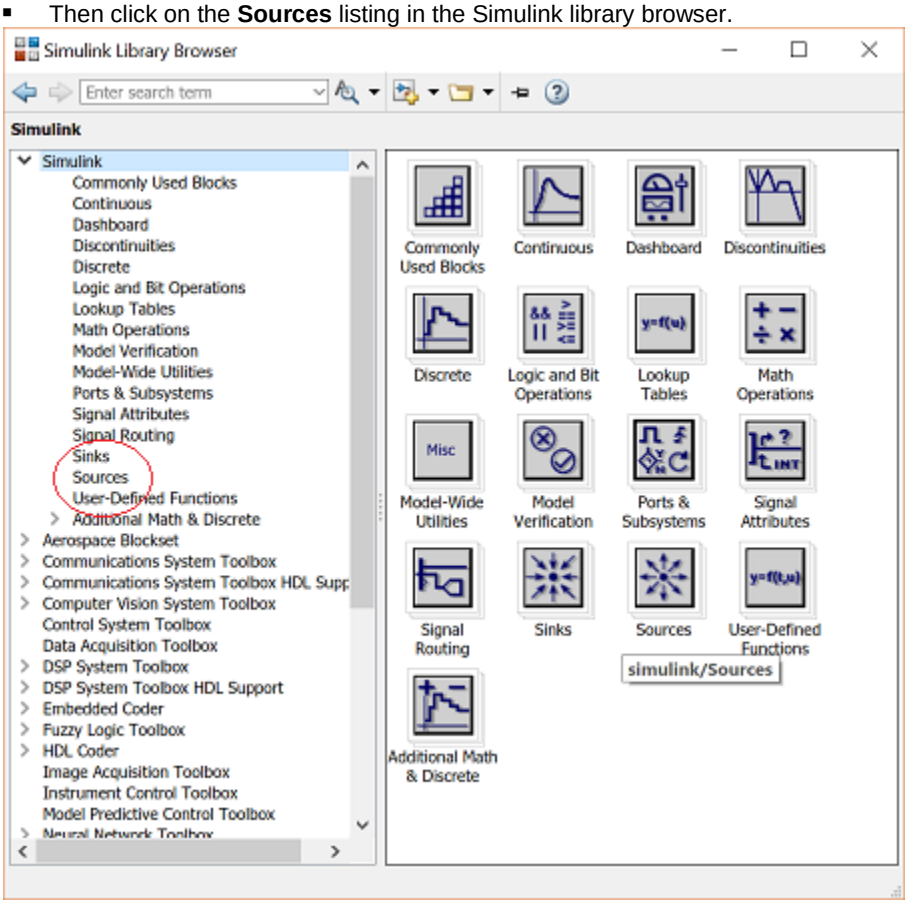



If you would like to download the completed model, right-click here and then select **Save link as ...**.

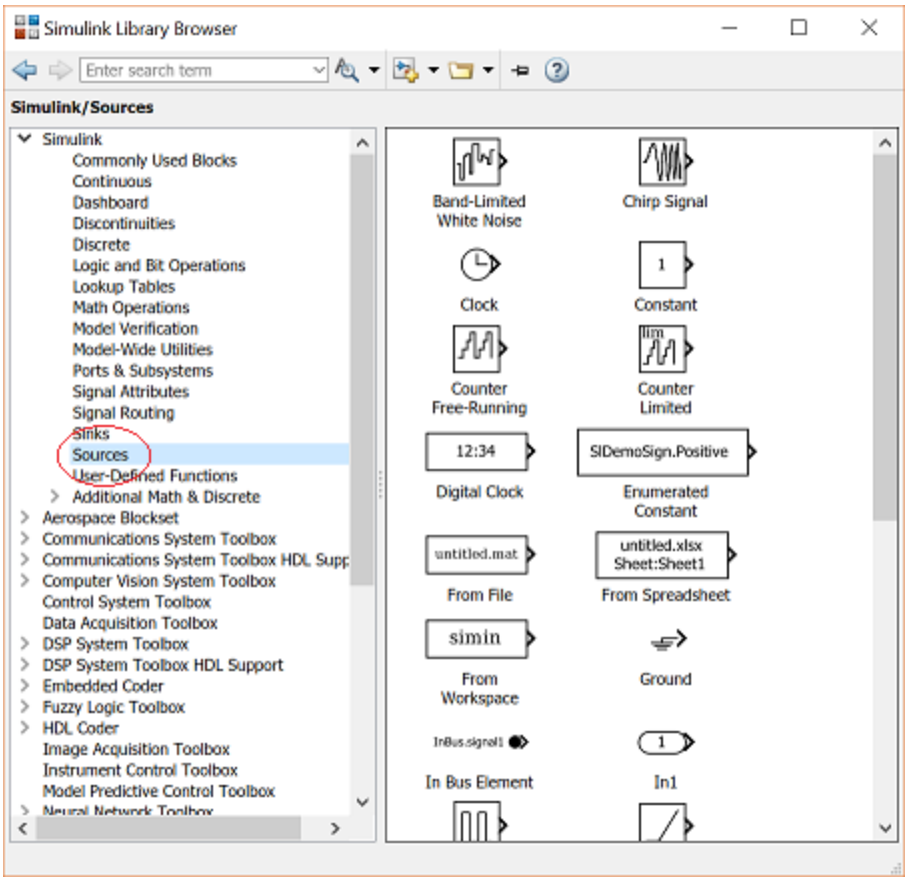
First, you will gather all of the necessary blocks from the block libraries. Then you will modify the blocks so they correspond to the blocks in the desired model. Finally, you will connect the blocks with lines to form the complete system. After this, you will simulate the complete system to verify that it works.

**Gathering Blocks**
Follow the steps below to collect the necessary blocks:

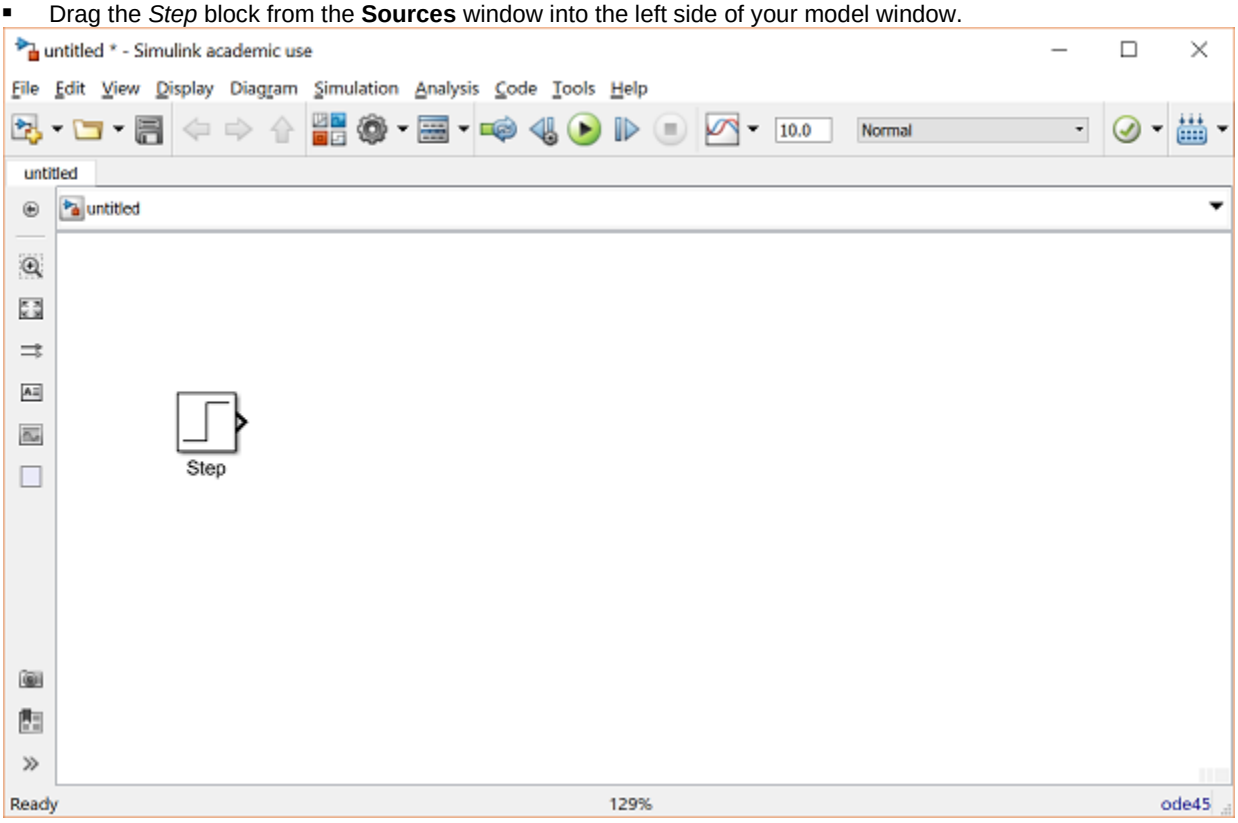
- Create a new model (**New** from the **File** menu or hit **Ctrl-N**). You will get a blank model window.
- Click on the **Tools** tab and then select **Library Browser**.
- Then click on the **Sources** listing in the Simulink library browser.



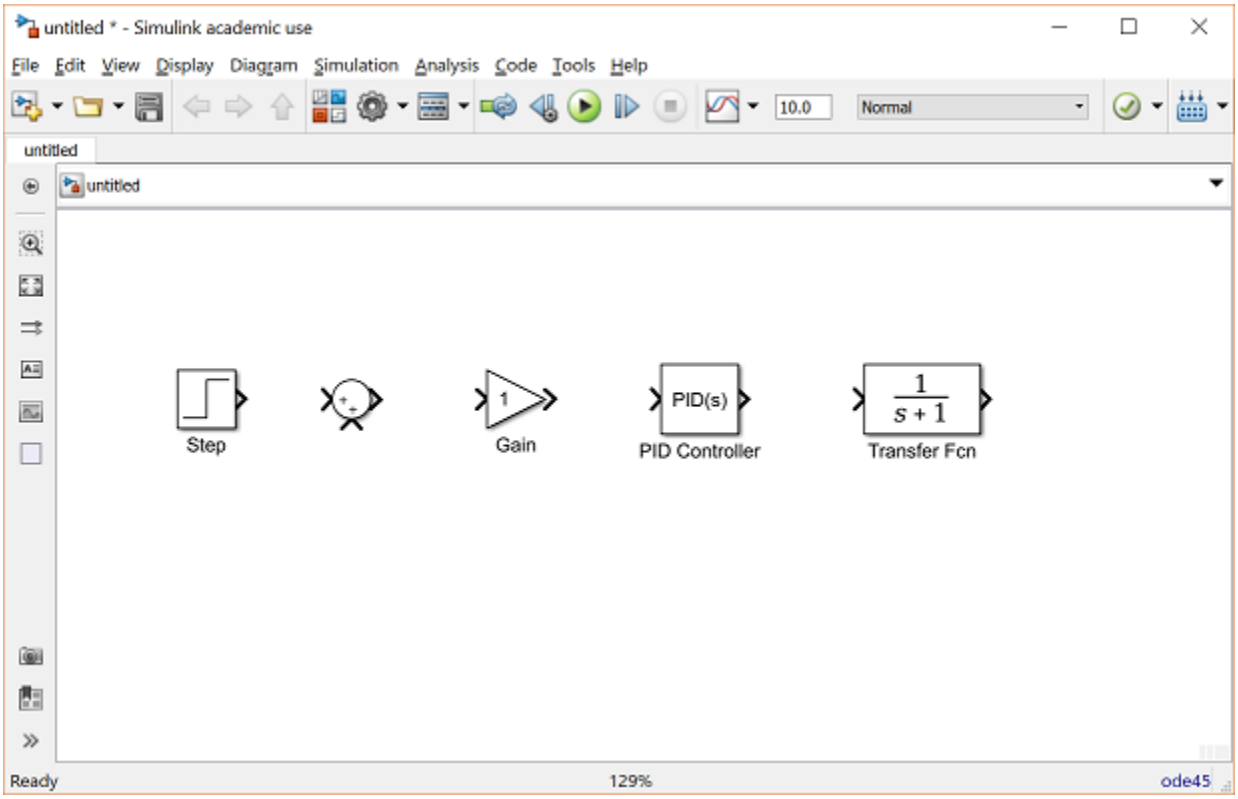

- This will bring up the **Sources** block library. **Sources** are used to generate signals.



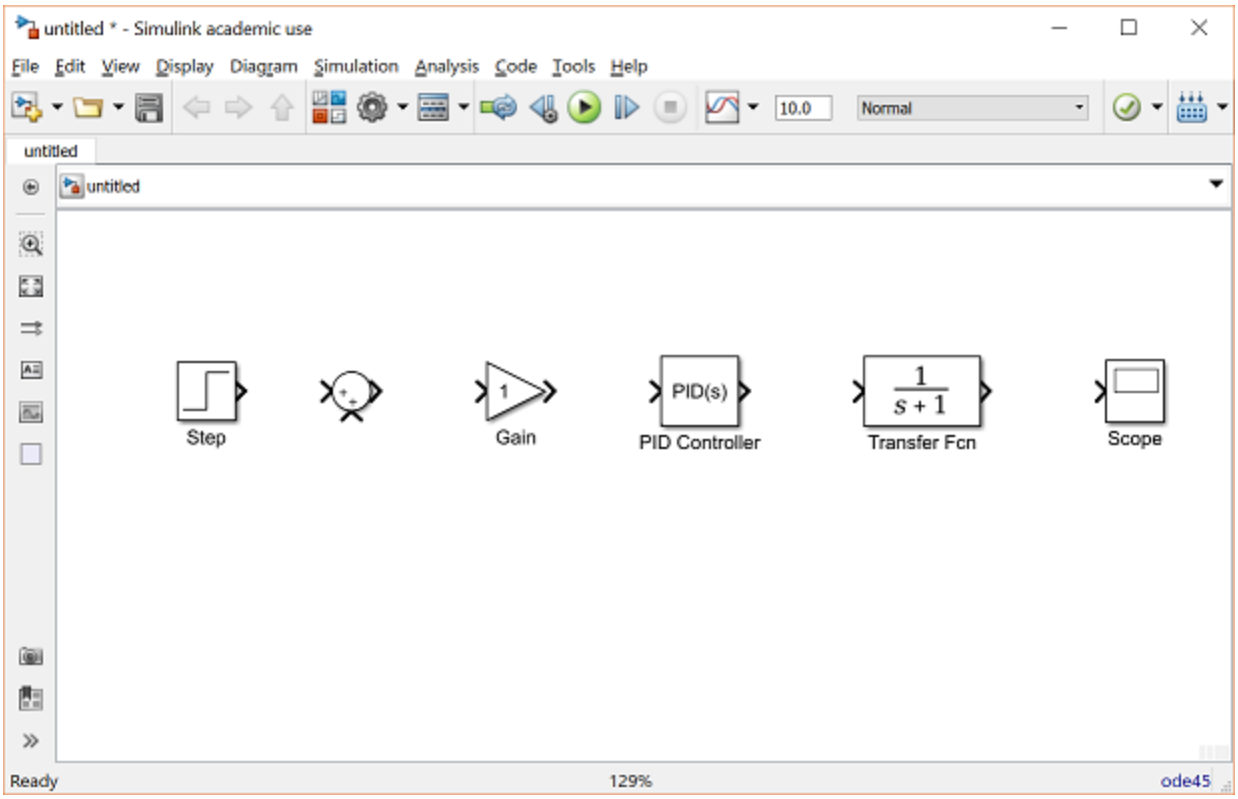

- Drag the *Step* block from the **Sources** window into the left side of your model window.



- Click on the **Math Operations** listing in the main Simulink window.
- From this library, drag a *Sum* and *Gain* block into the model window and place them to the right of the *Step* block in that order.
- Click on the **Continuous** listing in the main Simulink window.
- First, from this library, drag a *PID Controller* block into the model window and place it to the right of the *Gain* block.
- From the same library, drag a *Transfer Function* block into the model window and place it to the right of the *PID Controller* block.

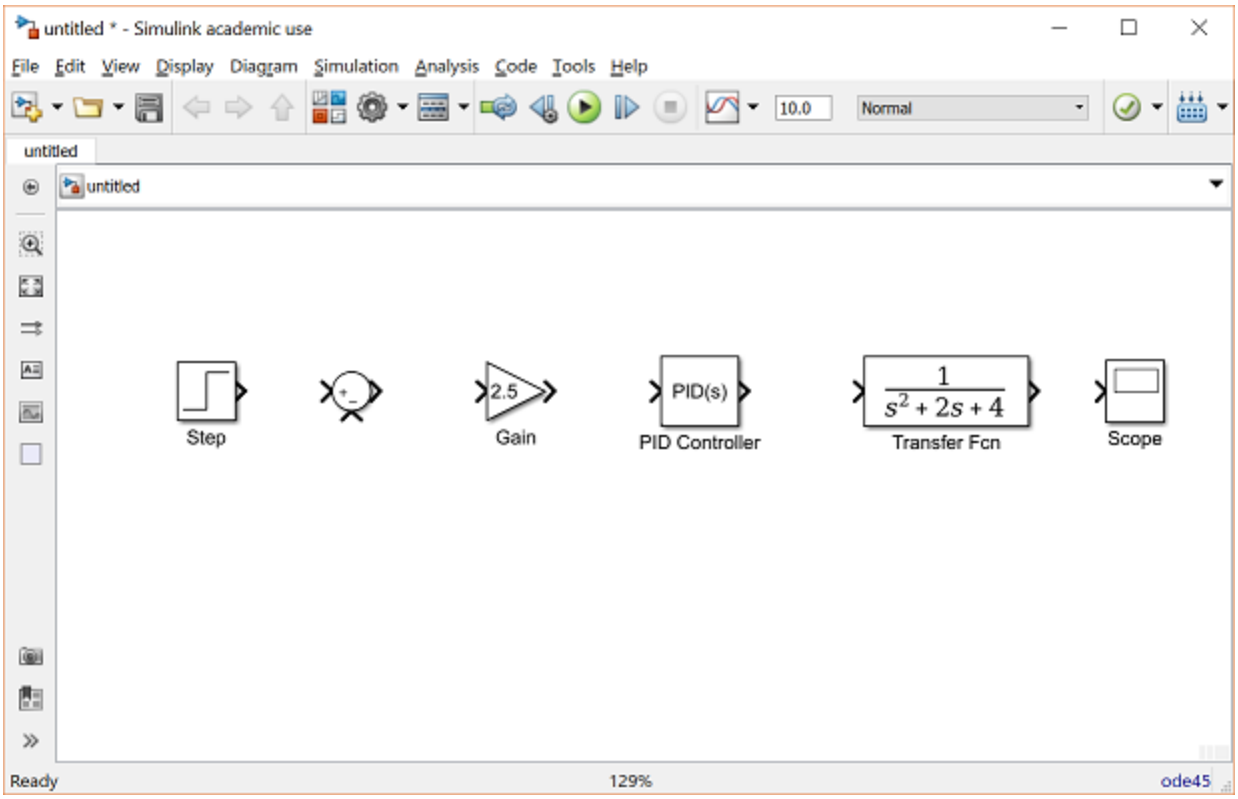



- Click on the **Sinks** listing in the main Simulink window.
- Drag the *Scope* block into the right side of the model window.

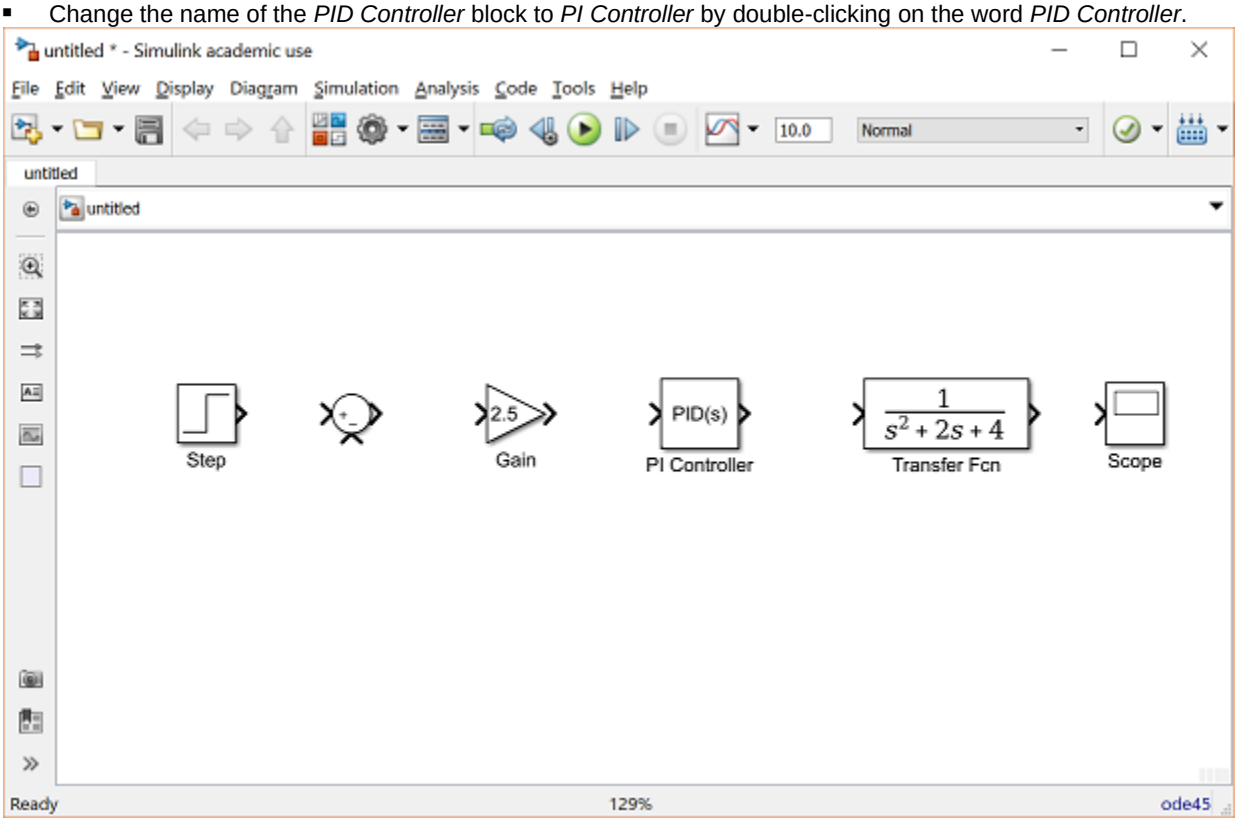



**Modify Blocks**
Follow these steps to properly modify the blocks in your model.

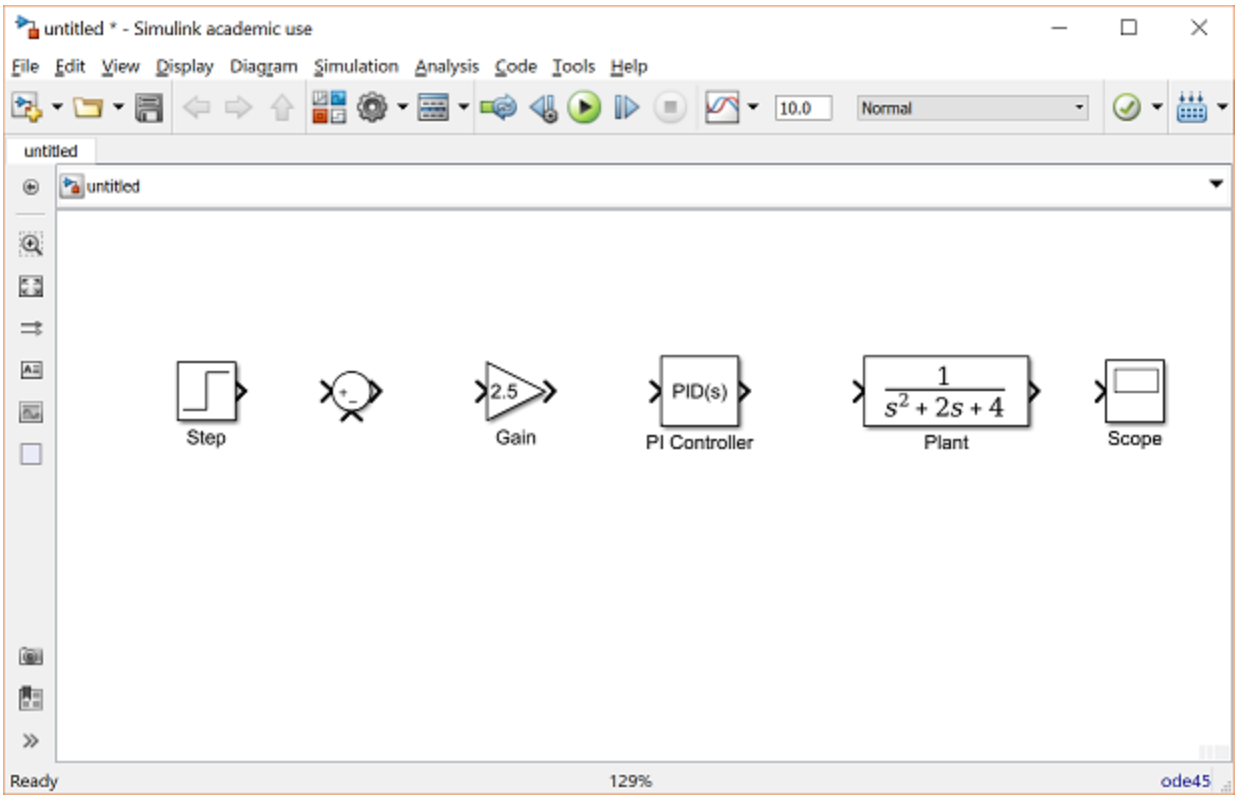
- Double-click on the *Sum* block. Since you will want the second input to be subtracted, enter +− into the list of signs field. Close the dialog box.
- Double-click the *Gain* block. Change the gain to 2.5 and close the dialog box.
- Double-click the *PID Controller* block and change the Proportional gain to 1 and the Integral gain to 2. Close the dialog box.
- Double-click the *Transfer Function* block. Leave the numerator [1], but change the denominator to [1 2 4]. Close the dialog box. The model should appear as:




- Change the name of the *PID Controller* block to *PI Controller* by double-clicking on the word *PID Controller*.



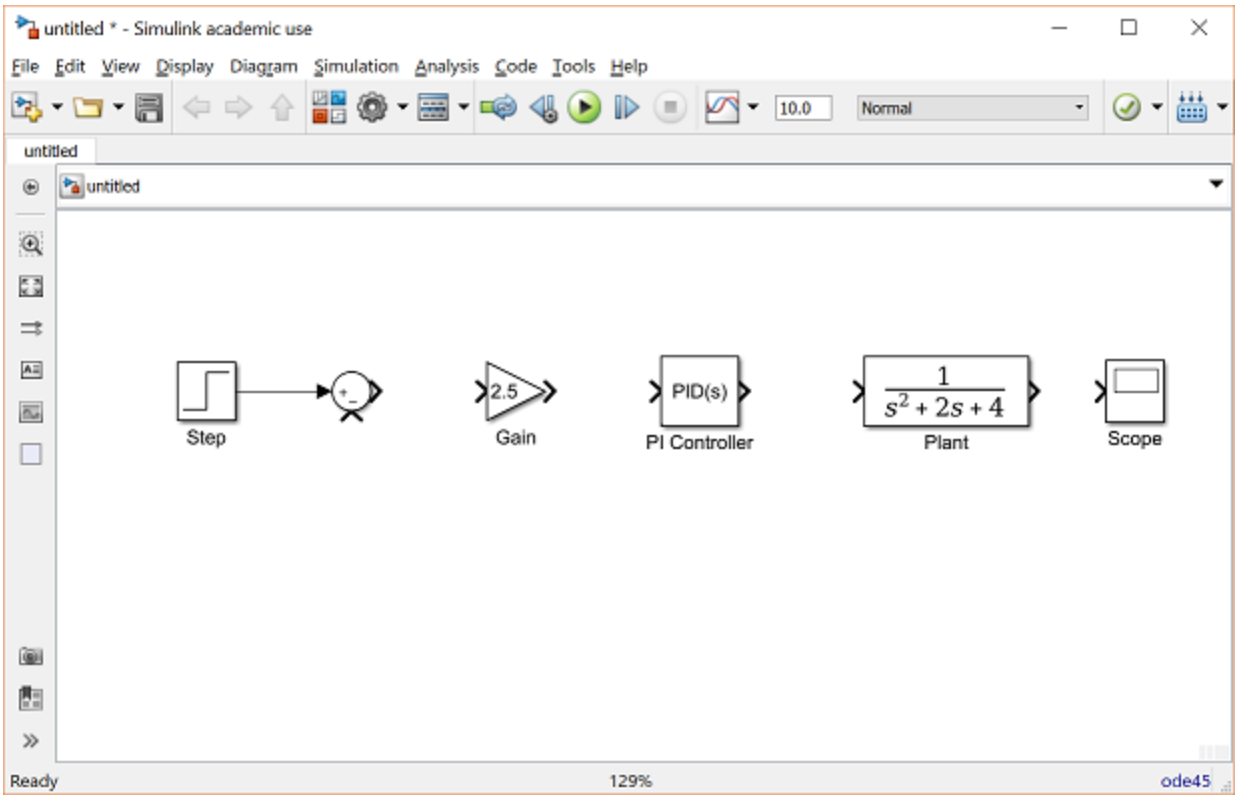

- Similarly, change the name of the *Transfer Function* block to *Plant*. Now, all the blocks are entered properly. Your model should appear as:



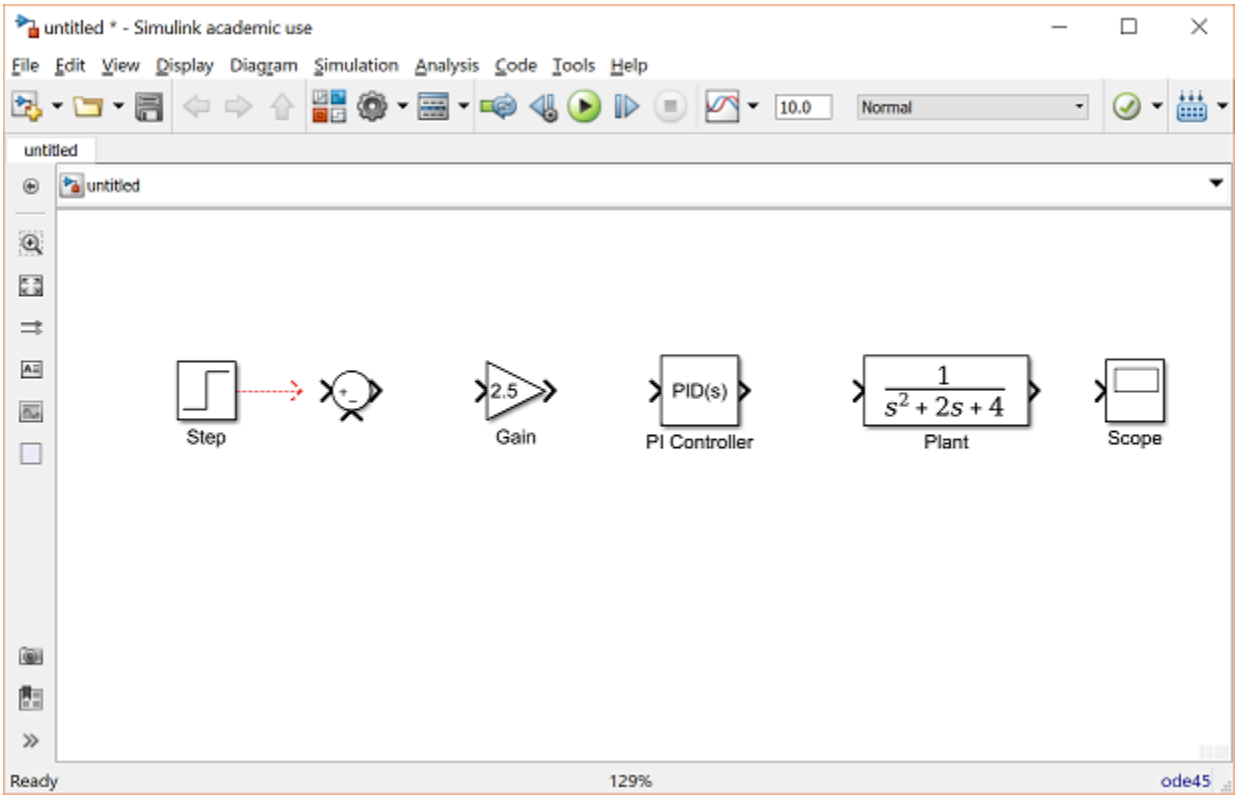

**Connecting Blocks with Lines**

Now that the blocks are properly laid out, you will now connect them together. Follow these steps.

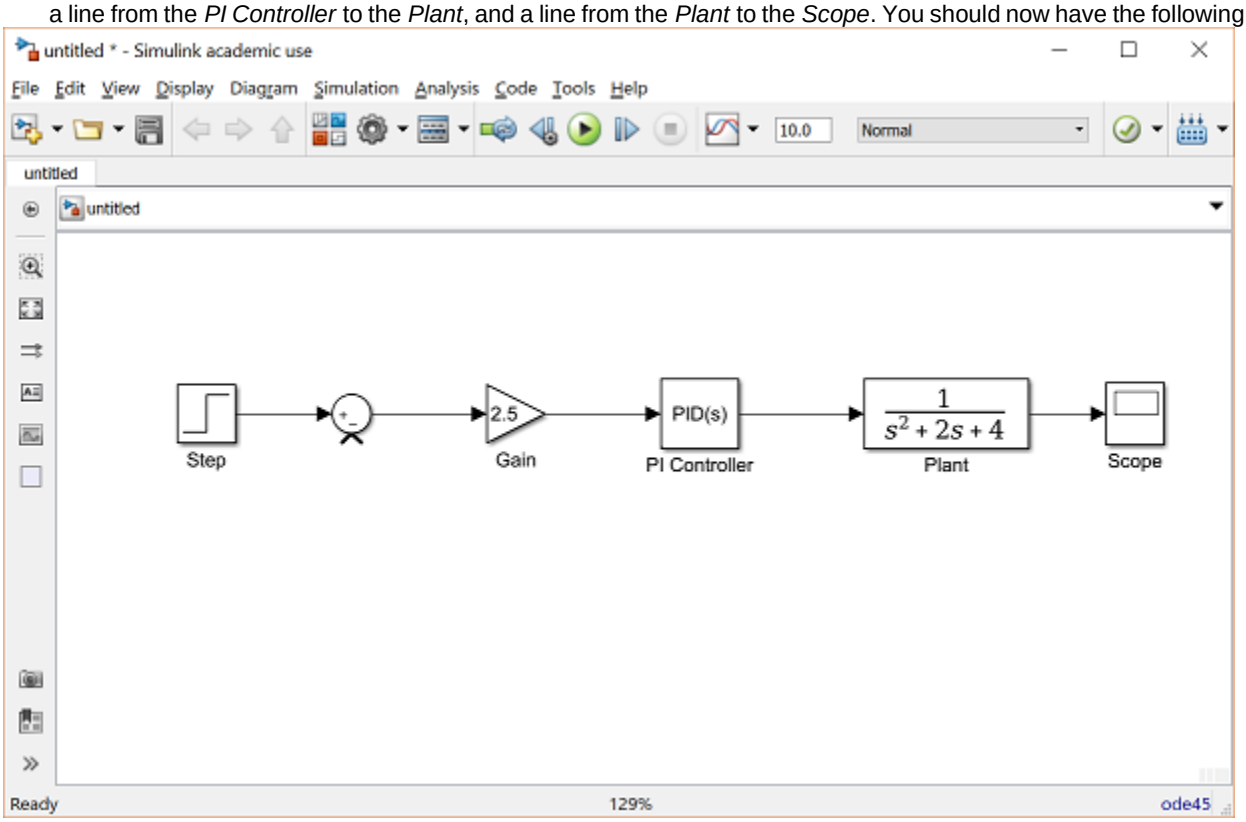
- Drag the mouse from the output terminal of the *Step* block to the positive input of the *Sum* input. Another option is to click on the *Step* block and then **Ctrl-Click** on the *Sum* block to connect the two togther. You should see the following.



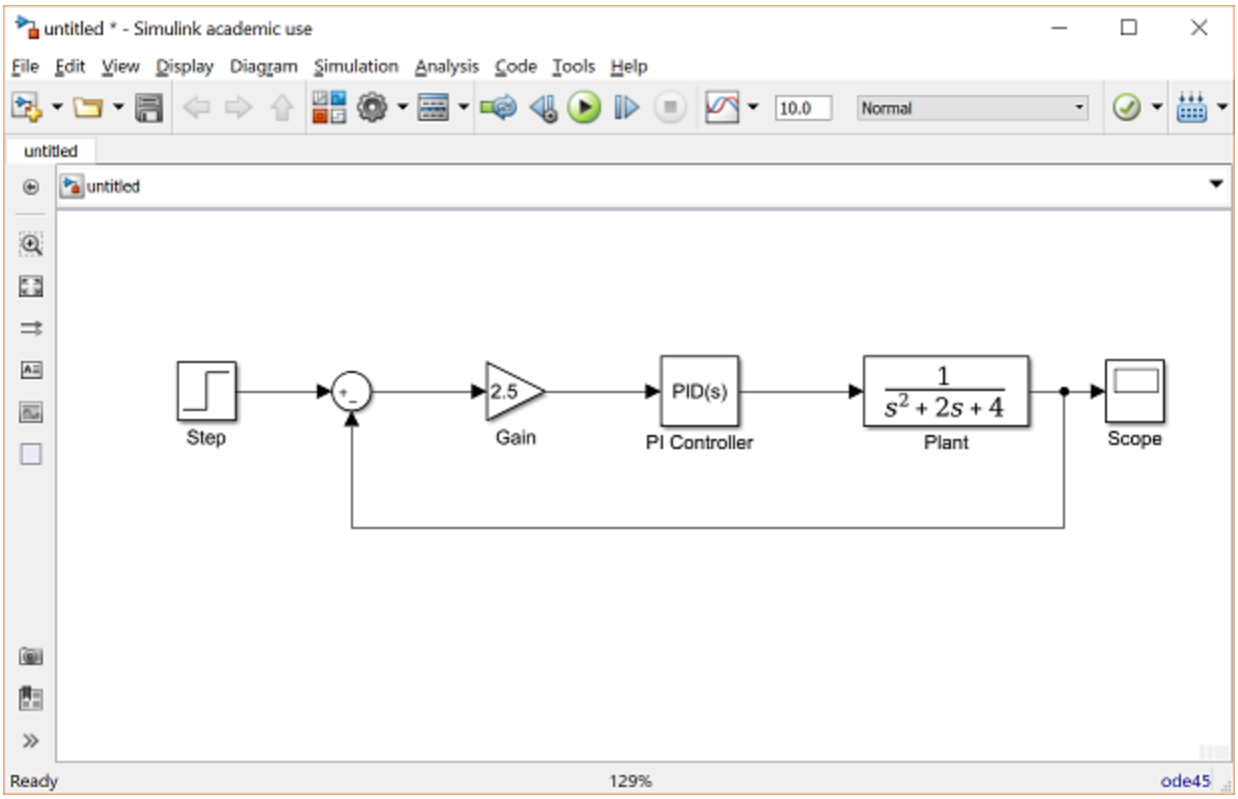

- The resulting line should have a filled arrowhead. If the arrowhead is open and red, as shown below, it means it is not connected to anything.



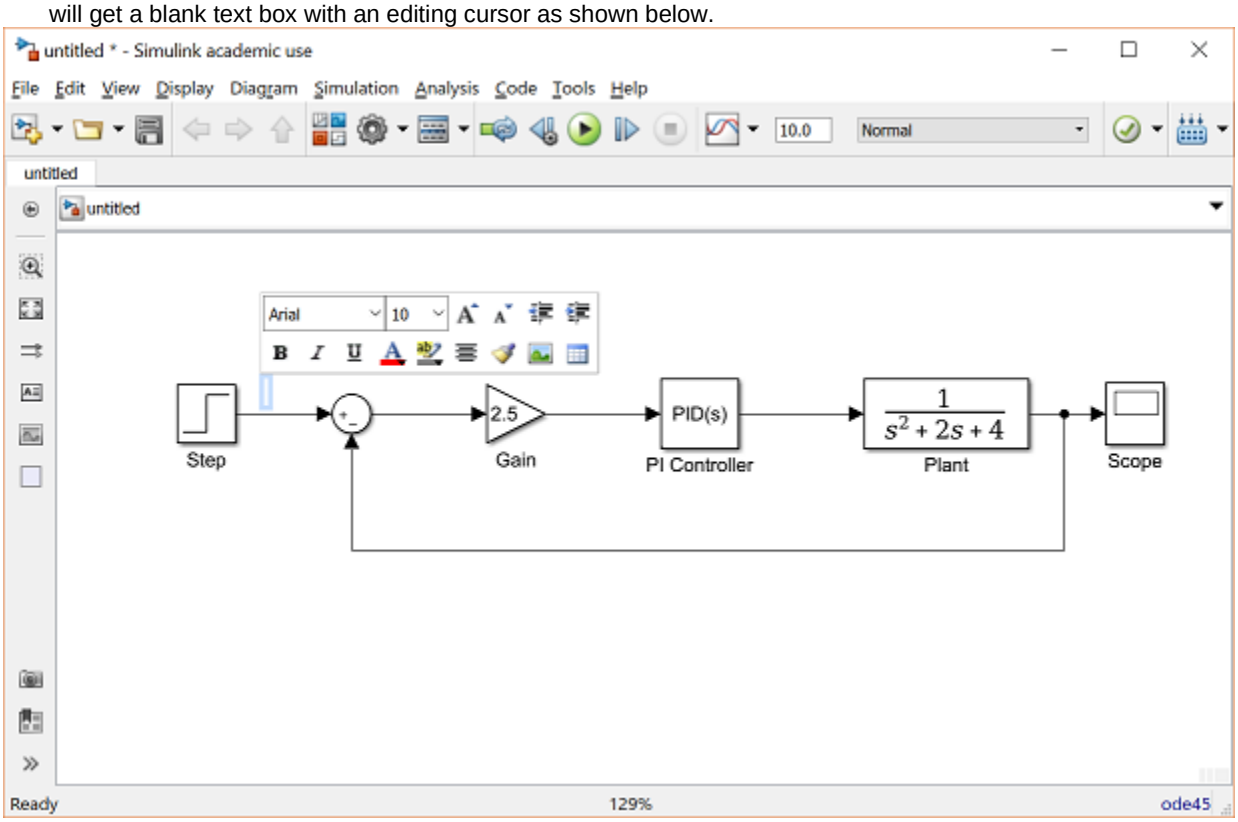
- You can continue the partial line you just drew by treating the open arrowhead as an output terminal and drawing just as before. Alternatively, if you want to redraw the line, or if the line connected to the wrong terminal, you should delete the line and redraw it. To delete a line (or any other object), simply click on it to select it, and hit the delete key.
- Draw a line connecting the *Sum* block output to the *Gain* input. Also draw a line from the *Gain* to the *PI Controller*, a line from the *PI Controller* to the *Plant*, and a line from the *Plant* to the *Scope*. You should now have the following.



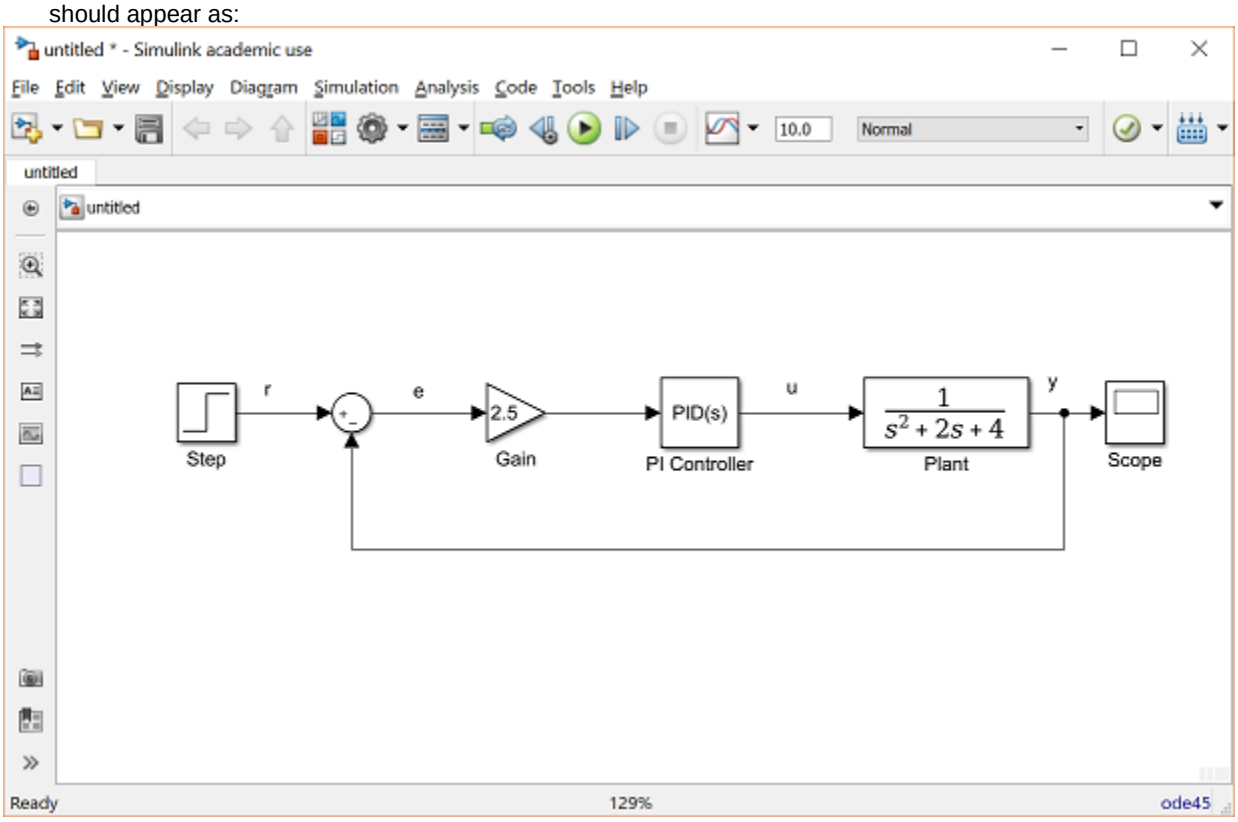

- The line remaining to be drawn is the feedback signal connecting the output of the *Plant* to the negative input of the *Sum* block. This line is different in two ways. First, since this line loops around and does not simply follow the shortest (right-angled) route so it needs to be drawn in several stages. Second, there is no output terminal to start from, so the line has to tap off of an existing line.
- Drag a line off the negative portion of the *Sum* block straight down and release the mouse so the line is incomplete. From the endpoint of this line, click and drag to the line between the *Plant* and the *Scope*. The model should now appear as follows.




- Finally, labels will be placed in the model to identify the signals. To place a label anywhere in the model, double-click at the point you want the label to be. Start by double-clicking above the line leading from the *Step* block. You will get a blank text box with an editing cursor as shown below.



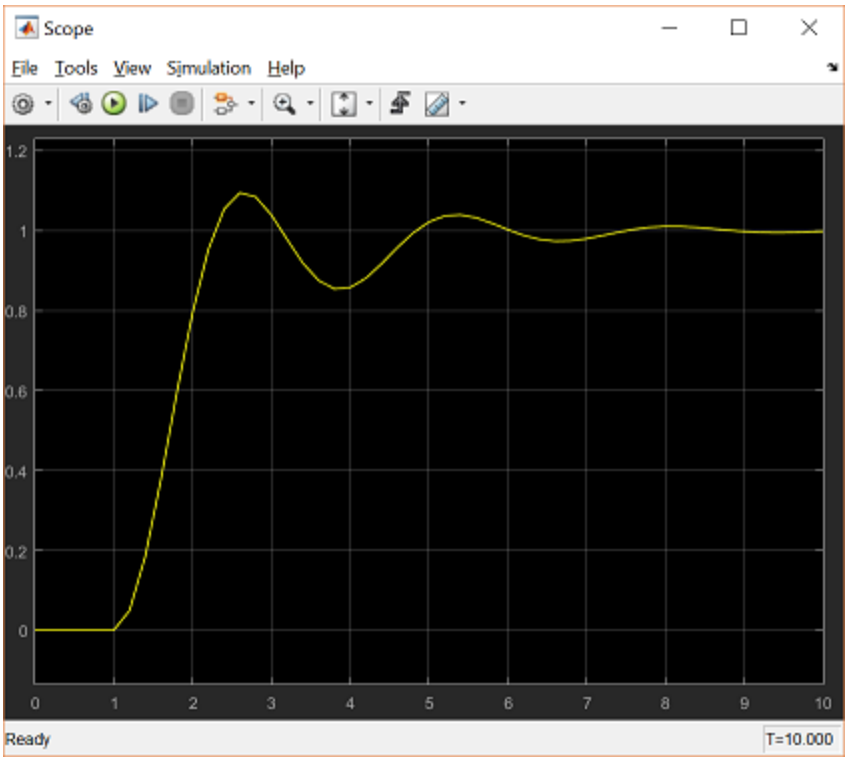
- Type an *r* in this box, labeling the reference signal and click outside it to end editing.
- Label the error (*e*) signal, the control (*u*) signal, and the output (*y*) signal in the same manner. Your final model should appear as:




To save your model, select **Save As** in the **File** menu and type in any desired model name. The completed model can be downloaded by right-clicking here and then selecting **Save link as ...**.

**Simulation**

Now that the model is complete, you can simulate the model. Select **Run** from the **Simulation** menu to run the simulation. Double-click on the _Scope_block to view its output and you should see the following:



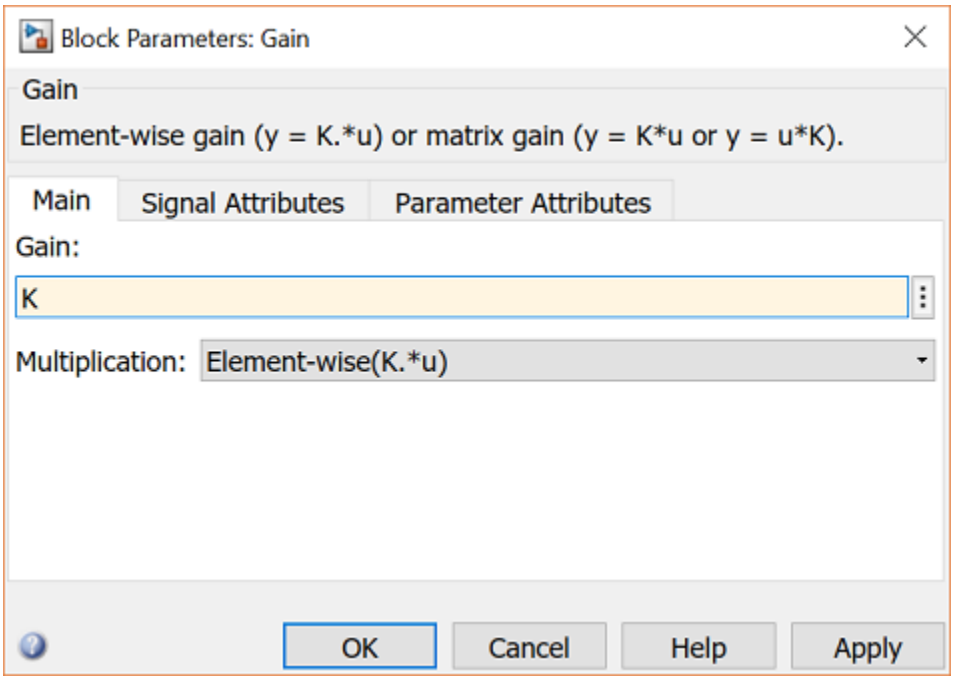

**Taking Variables from MATLAB**

In some cases, parameters, such as gain, may be calculated in MATLAB to be used in a Simulink model. If this is the case, it is not necessary to enter the result of the MATLAB calculation directly into Simulink. For example, suppose we calculated the gain in MATLAB in the variable `K`. Emulate this by entering the following command at the MATLAB command prompt.

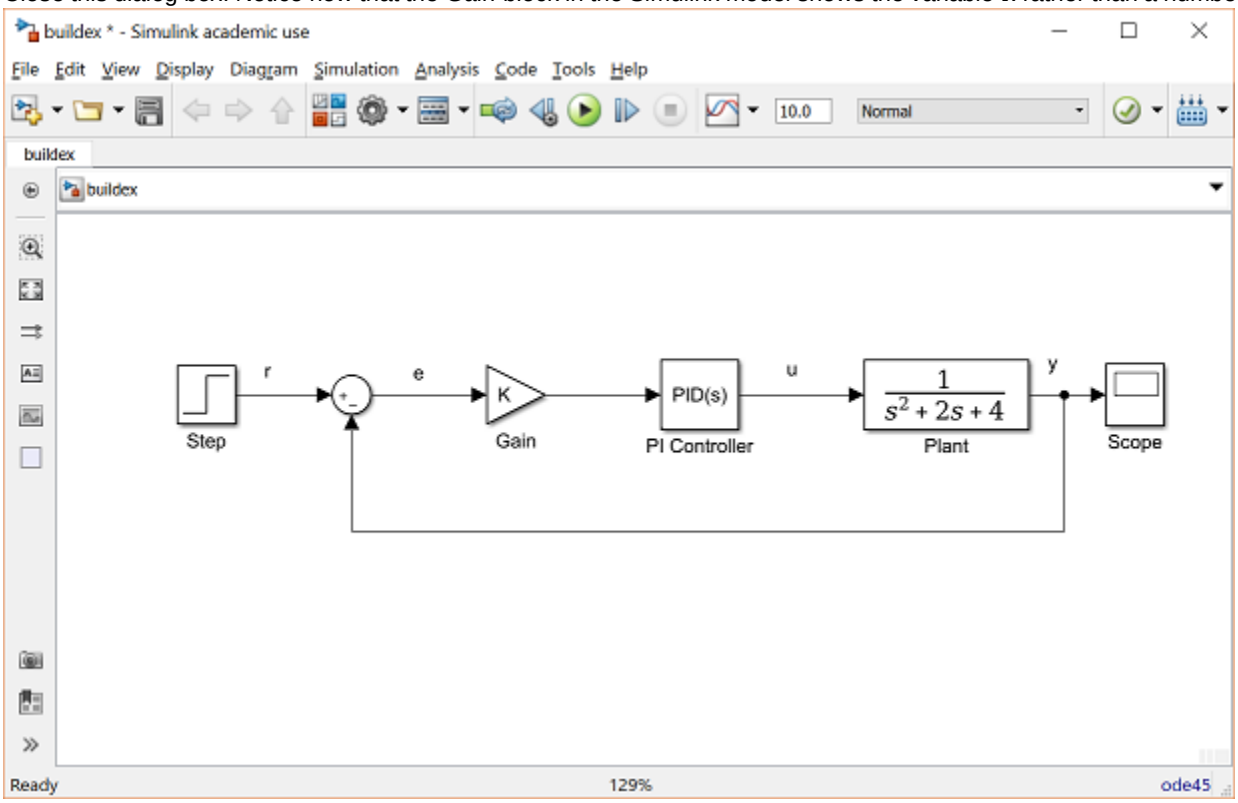
```
K = 2.5
```

This variable can now be used in the Simulink *Gain* block. In your Simulink model, double-click on the *Gain* block and enter the following the Gain field.

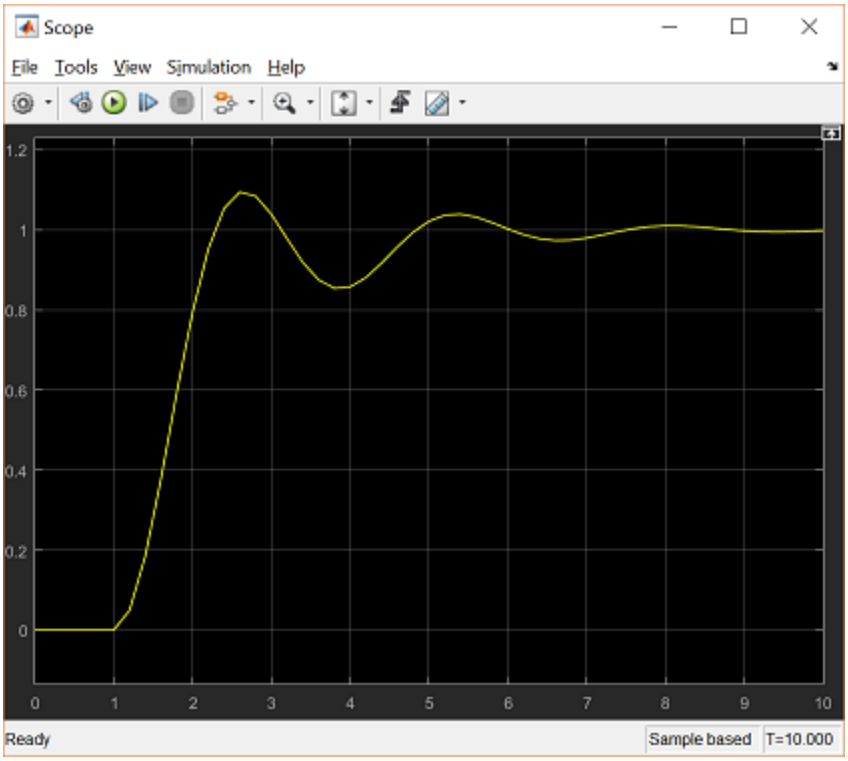
```
K
```



Close this dialog box. Notice now that the *Gain* block in the Simulink model shows the variable `K` rather than a number.



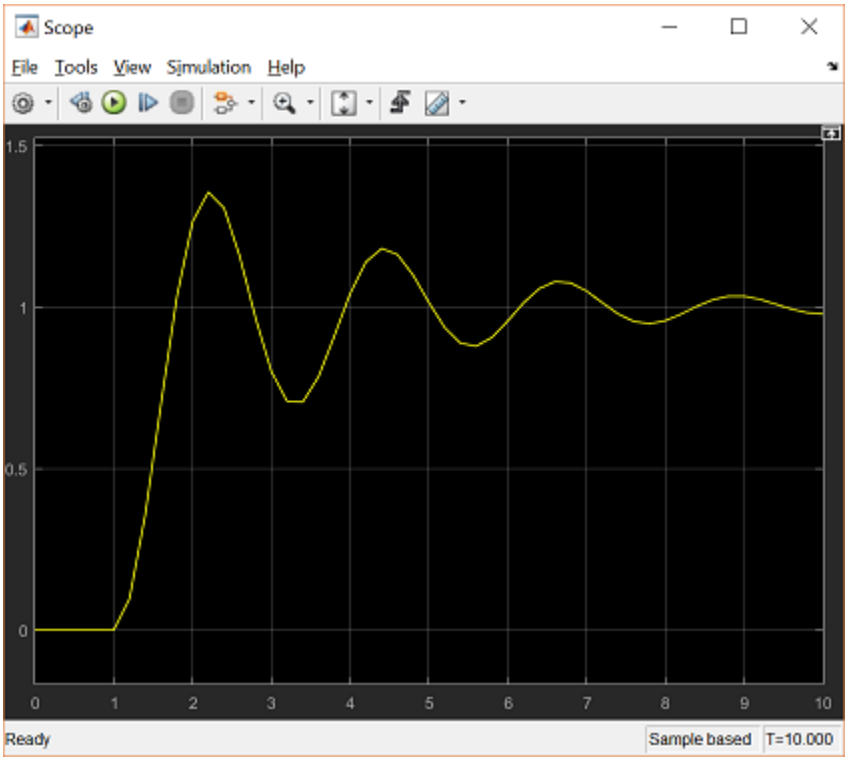

Now, you can re-run the simulation and view the output on the Scope. The result should be the same as before.



Now, if any calculations are done in MATLAB to change any of the variables used in the Simulink model, the simulation will use the new values the next time it is run. To try this, in MATLAB, change the gain, `K`, by entering the following at the command prompt.

```
K = 5
```

Start the Simulink simulation again and open the Scope window. You will see the following output which reflects the new, higher gain.




Besides variables and signals, even entire systems can be exchanged between MATLAB and Simulink.

# EXPERIMENT -10

# Simulation of Aircraft motion-longitudinal dynamics, lateral dynamics

**Aim:** Write a code to simulate the aircraft longitudinal and lateral dynamics.

**Equipment needed:** Core 2 duo processor with 1GB RAM and MATLAB software Version 2008a or higher or Sci-lab.

**Theory:** The motion of aircraft is necessary to define a suitable coordinate system for formulations of the equation of motion. It has two axis systems: earth axis system and body axis system. Earth axis system means that it is fixed to the earth surface. Body axis system is attached to the aircraft body. Moreover, the stability axis system is used as a reference for aerodynamic moment and forces. Lift forces and drag forces are transformed to normal forces and axial force respectively. In dynamic modeling of aircraft, the forces acting on the aircraft are mainly:

- The weight located at the center of gravity
- The thrust of the propeller acting in the x-direction.
- The aerodynamic forces of each part of the airplane (mainly the wing).

The aircraft equations of motions in order to formulate that must be considered as the follows;

- There is the flat earth.
- There is non-rotation mass.
- The aircraft is rigid body.
- The aircraft is symmetric.
- There is a constant wind.
- There is no rotating earth.

The symmetric flight produces the zero bank angle and all moments are zero in steady straight flight conditions. In dynamic modeling of aircraft, the forces acting on the aircraft are mainly:

- The weight located at the center of gravity
- The thrust of the propeller acting in the x-direction.
- The aerodynamic forces of each part of the airplane (mainly the wing).

The aircraft equations of motions in order to formulate that must be considered as the follows;

- There is the flat earth.
- There is non-rotation mass
- . • The aircraft is rigid body.
- The aircraft is symmetric.
- There is a constant wind.
- There is no rotating earth.

The symmetric flight produces the zero bank angle and all moments are zero in steady straight flight conditions. $\sum_{abs} F = ma$

$$\sum_{abs} M = I\omega$$

The velocity vector is related with six parameters: forward velocity, sideway velocity, vertical velocity, roll rate, pitch rate and yaw rate. The velocity vector, **v** in the body frame is defined as, where the angular velocity is expressed with respect to the north-east-down frame:

Velocity vector, **v=**
$$\begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \text{forward velocity} \\ \text{sideway velocity} \\ \text{vertical velocity} \\ \text{roll rate} \\ \text{pitch rate} \\ \text{yaw rate} \end{bmatrix}$$

6 DOF rigid-body forces and moments in the body frame are defined as follows:

$$\begin{bmatrix} X \\ Y \\ Z \\ L \\ M \\ N \end{bmatrix} = \begin{bmatrix} \text{forward force} \\ \text{sideway force} \\ \text{vertical force} \\ \text{rollmoment} \\ \text{pitchmoment} \\ \text{yawmoment} \end{bmatrix}$$

Dynamic pressure, $Q = \frac{1}{2}\rho V_T^2 = \frac{1}{2}\rho u_o^2$

On component form:

- $m(\dot{u} + qw - rv + g\sin\theta) = X$
- $m(\dot{v} + ur - pw + g\cos\theta\sin(\varphi)) = Y$
- $m(\dot{w} + pv - qu + g\cos\theta\sin(\varphi)) = Z$
- $I_x\dot{p} + I_{xz}(\dot{r} + pq) + (I_z - I_y)qr = L$
- $I_y\dot{q} + I_{xz}(p^2 - r^2) + (I_x - I_z)pr = M$
- $I_z\dot{r} - I_{xz}(\dot{p} + qr) + (I_y - I_x)pq = N$

Airflow angles: The two angles β slideslip angle and α angle of attack that are related to the flight direction deals with the air are expressed as follows:

Inverse
$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} v_T\cos\alpha\cos\beta \\ v_T\sin\beta \\ v_T\sin\alpha\cos\beta \end{bmatrix}$$ relationship:

$$\begin{bmatrix} v_T \\ \beta \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{u^2 + v^2 + w^2} \\ \sin^{-1}(v/v_T) \\ \tan^{-1}(w/u) \end{bmatrix}$$

$$\alpha = \tan^{-1}\left(\frac{w}{u}\right)$$

$$\beta = \tan^{-1}\left(\frac{v}{\sqrt{u^2 + w^2}}\right)$$

## 3 Physical Measurement and Mass Properties

Smart Fly is the airframe with small-size flying wing UAV that is used for aerial mapping system. The airframe is assumed to have aerodynamically efficient as the conventional stabilizers areremoved in Flying wings UAVs. The major dimensions of Smart One are presented in Table 4.1. Aspect ratio is calculated by using AR= $b^2/S$ and it has the cruise speed with 11.5 m/s. The aircraft is 5m/s of climb rate with the pitch angle of 7 degrees (or) 0.122 radians. The dynamic pressure is also calculated by using $1/2\rho u^2$.

## 4 Stability and Control Derivatives

The formulae for coefficient of X-axis and Z-axis are as follows:

- $C_{x0} = -C_{D0}\cos\alpha + C_{L0}\sin\alpha = -0.0237$
- $C_{x\alpha} = -C_{D\alpha}\cos\alpha + C_{L\alpha}\sin\alpha = 0.2972$
- $C_{x\delta e} = -C_{D\delta e}\cos\alpha + C_{L\delta e}\sin\alpha = -0.0143$
- $C_{xq} = -C_{Dq}\cos\alpha + C_{Lq}\sin\alpha = 0$
- $C_{z0} = -C_{D0}\sin\alpha + C_{L0}\cos\alpha = 0.0226$
- $C_{z\alpha} = -C_{D\alpha}\sin\alpha + C_{L\alpha}\cos\alpha = 0.4956$
- $C_{z\delta e} = -C_{D\delta}\text{esin}\alpha + C_{L\delta e}\cos\alpha = 0.573$
- $C_{zq} = -C_{Dq}\sin\alpha + C_{lq}\cos\alpha = 0$

***TABLE 1***
*UAV SPECIFICATIONS*

| Symbol | Values | Properties |
|---|---|---|
| b | 1.99m | Wing span |
| S | 0.076m² | Wing area |
| m | 0.0973kg | Empty weight |
| u | 9.67m/s | Forward velocity along the X-axis |
| w | 1.16m/s | Vertical velocity along the Z-axis |
| ρ | 1.225kg/m³ | Air pressure density |
| α | 0.119rads | Angle of attack |
| θ | 0.119rads | Pitch angle |
| $\delta_e$ | 0.0279rads | Elevator deflections angle |
| $\delta_T$ | 0.57rads | Throttle angle |
| c | 0.0381m | Chord |
| $I_x$ | 0.1124 | Rolling moment of inertia |
| $I_x$ | 0.0709 | Pitching moment of inertia |
| $I_x$ | 0.0562 | Yawing moment of inertia |
| Q | 296.45kg/m³ | Dynamic pressure |

Longitudinal stability derivatives formulae are presented as follow:

- Stability Derivative, $X_u = -6.68$
- Angle of Attack Derivative, $X_w = 4.1754$
- Elevator Deflection, $X_{\delta e} = -0.649$
- Thrust Deflection ,$X_{\delta T} = 0$
- Compressibility Effect Derivative ,
- $M_u = -0.01376$
- Dimensional Pitching Moment ,
- Derivative, $M_w = 0.05852$

- Pitching moment (Elevator Deflection) ,$M_{\delta e}$ = -1.1526
- Dimensionless Pitching Moment Derivative, $M_q$= -0.1179
- Pitching moment (Thrust Deflection) ,$M_{\delta T}$= 0
- Pitch Rate Derivative $X_q$ = -1.16
- Stability Derivative, $Z_u$ = -0.6276
- Angle of Attack Derivative, $Z_w$ =-3.0503
- Elevator Deflection ,$Z_{\delta e}$ = 26.0063
- Thrust Deflection, $Z_{\delta T}$= 0
- Pitch Rate Derivative, $Z_q$= 9.67

Lateral stability derivatives formulae are presented as follows:

- Roll Rate ,$Y_p$ = -0.05579
- Aileron Deflection Derivative ,$Y_{\delta a}$ = 0
- Yaw Rate Derivative, $Y_r$ = 0
- Sideslip Derivative $Y_\beta$ = -4.5129
- Rolling Moment , $L_p$ = -0.3295
- Rolling Moment $L_r$ ,= 0.0205
- Rolling Moment,$L_{\delta a}$ = 3.6299
- Roll Acceleration ,$L_\beta$ = 3.7096
- Yawing Moment, $N_{\delta a}$ = 3.0316
- Yawing Moment , $N_p$ = 0.02025
- Yawing Moment ,$N_r$ = -0.10266
- Yaw Acceleration ,$N_\beta$ = 0.79937

## 5 State Space System

Non –Linear equation of motion which had been derived based on Newton's second law in Chapter 3, are difficult to be used for control system design purpose. The linearized dynamic equations had been calculated by using small-disturbance theory in Chapter 3. Linear differential equation with constant coefficients, it is possible to write as a set of first-order differential equations in the form of a State – Space Model. The equation of motion or state equation of the linear time invariant multivariable system in the matrix form is written:

- $\dot{x}(t)=Ax(t)+Bu(t)$
- where;
- x(t) = the column vector of n state variables called the state vector

u(t) = the column vector of m input variables called the input vector
A = the (n/n) state matrix
B = the (n/m) input matrix

The corresponding output equation is written as follows.

- y(t) = Cx(t) +Du(t)
- where, y(t) = the column vector r of output variable called the output vector
- C = the (r/n) output matrix
- D = the (r/m) feed forward matrix

The matrices A, B, C and D have the constant elements for an LTI system

### 5.1 Longitudinal–Directional State Space System

Since four states variables u, w, q and θ in longitudinal motion of the aircraft, four transfer functions are required. Therefore, the longitudinal equation of motion can be written in the form of state space mode:

### 5.2 Lateral–Directional State Space System

There are four lateral directional motion of aircraft: side force, rolling moment, and yawing moment. The lateral motion can be written by using state space form in Equation

$$\begin{bmatrix}\Delta u\\ \Delta w\\ \Delta q\\ \Delta\theta\end{bmatrix}=\begin{bmatrix}X_u & Xw & Xq+w_0 & -g\cos\theta_0\\ Z_u & Zw & Zq+w_0 & -g\sin\theta_0\\ M_u & Mw & Mq & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\Delta u\\ \Delta w\\ \Delta q\\ \Delta\theta\end{bmatrix}+\begin{bmatrix}X\delta e & X\delta T\\ Z\delta e & Z\delta T\\ M\delta e & M\delta T\\ 0 & 0\end{bmatrix}\begin{bmatrix}\Delta\delta_e\\ \Delta\delta T\end{bmatrix}$$

$$\begin{bmatrix}\Delta u\\ \Delta w\\ \Delta q\\ \Delta\theta\end{bmatrix}=\begin{bmatrix}-6.68 & 4.1754 & 0 & -32.174\\ -0.6276 & -3.0503 & 19.34 & 0\\ -0.01376 & 0.05852 & -0.1179 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\Delta u\\ \Delta w\\ \Delta q\\ \Delta\theta\end{bmatrix}+\begin{bmatrix}-0.649 & \\ 26.0063 & \\ -1.1526 & \\ 0 & \end{bmatrix}\begin{bmatrix}\Delta\delta_e\\ \Delta\delta T\end{bmatrix}$$

$$\begin{bmatrix}\Delta\beta\\ \Delta p\\ \Delta\phi\\ \Delta r\end{bmatrix}=\begin{bmatrix}Y\beta & Yp & -(u0-Yr) & g\cos\theta_0\\ L\beta & Lp & Lr & 0\\ N\beta & 1 & Nr & 0\\ 0 & Np & \tan\theta_0 & 0\end{bmatrix}\begin{bmatrix}\Delta\beta\\ \Delta p\\ \Delta\phi\\ \Delta r\end{bmatrix}+\begin{bmatrix}0\\ L\delta a\\ N\delta a\\ 0\end{bmatrix}[\Delta\delta a]$$

$$\begin{bmatrix}\Delta\beta\\ \Delta p\\ \Delta\phi\\ \Delta r\end{bmatrix}=\begin{bmatrix}-0.2051 & -0.05579 & -21.9543 & 32.174\\ -0.1686 & -0.3295 & 0.0205 & 0\\ 0.03633 & 0.02025 & -0.10266 & 0\\ 0 & 1 & 0 & 0\end{bmatrix}\begin{bmatrix}\Delta\beta\\ \Delta p\\ \Delta\phi\\ \Delta r\end{bmatrix}+\begin{bmatrix}0\\ 3.6299\\ 3.0316\\ 0\end{bmatrix}[\Delta\delta a]$$

## 6 Transfer Function for Longitudinal and Lateral Dynamic System

Transfer functions are more convenient than state space system because its denominator gives the system pole locations in order to decide the system either stable or unstable condition. Pole location in s-plane is significant in control system because it can generate the corresponding response of a plant with step input. The response indicates the system is either dynamically stable or unstable curves. Therefore, the state space system can be transformed into four transfer functions

### 6.1 Longitudinal Transfer Function

$$G_p(s)=\frac{-1.153\,s2\ -9.684s-17.78}{s4+9.848\,s3+23.01\,s2-4.181\,s-2.532}$$

The pole locations of longitudinal motions are:-5.6842, -4.292, 0.3930,-0.2641

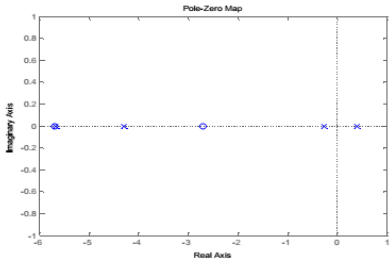



***Fig. 1.****Pole-Zero Map for longitudinal transfer function*

## 6.1 Longitudinal Transfer Function

$$G_p(s) = \frac{-1.153\,s2\ -9.684s\ -17.78}{s4+9.848\,s3+23.01\,s2-4.181\,s-2.532}$$

The pole locations of longitudinal motions are:-5.6842, -4.292, 0.3930,-0.2641

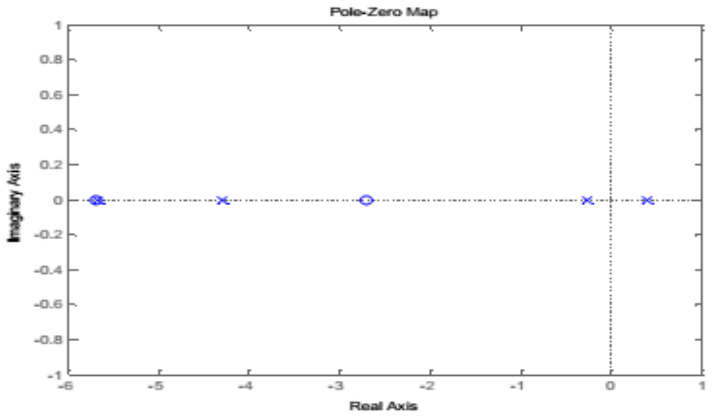



***Fig. 1.****Pole-Zero Map for longitudinal transfer function*

## 6.2 Lateral Transfer Function

$$G_p(s) = \frac{3.63\,s2\ +1.179s\ +14.21}{s4+0.6373\,s3+0.9102\,s2+\ 5.618s+0.5329}$$

The pole locations of lateral motions are:-0.6244 + 1.6441 0.6244 - 1.6441i,-1.7897,-0.0963.

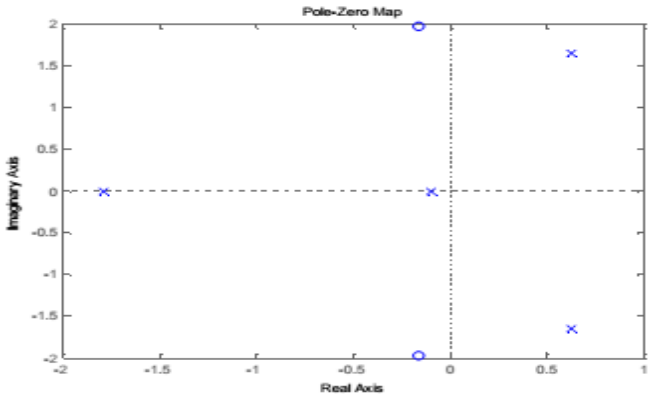



***Fig. 2*** *Pole-Zero Map for lateral transfer function*