Skill Oriented Laboratory – I -MATLAB & Simulink (P18EEL58)

List of Experiments

MATLAB

1. Introduction to MATLAB

2. Basic matrix operations

3. Solution of linear equation

4. Solution of Linear equations for Underdetermined and Over determined cases.

5. Determination of Eigen values and Eigen vectors of a Square matrix.

6. Solution of Difference Equations.

7. Solution of Difference Equations using Euler Method.

8. Solution of differential equation using 4th order Runge- Kutta method.

9. Determination of roots of a polynomial.

10. Determination of polynomial using method of Least Square Curve Fitting.

11. Determination of polynomial fit, analyzing residuals, exponential fit and error bounds from the given data.

12. Determination of time response of an R-L-C circuit.

Simulink simscape

1. Creating a simple circuit

2. Determination of R, L & C Responses

3. Design of RC, RL, RLC Circuits

4. Design of Half wave and Full wave Rectifier Circuits.

5. Simulating Motor control Techniques

1. Introduction to MATLAB

What Is MATLAB?

The name MATLAB stands for ***matrix laboratory*.**

MATLAB® is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions areexpressed in familiar mathematical notation. Typical uses include

- •Math and computation
- •Algorithm development
- •Data acquisition
- •Modeling, simulation, and prototyping
- •Data analysis, exploration, and visualization
- •Scientific and engineering graphics
- •Application development, including graphical user interface building

MATLAB features a family of add-on application-specific solutions called ***toolboxes***. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

The MATLAB System

The MATLAB system consists of five main parts:

Development Environment. This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, an editor and debugger, and browsers for viewing help, the workspace, files, and the search path.

The MATLAB Mathematical Function Library. This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

The MATLAB Language. This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both “programming in the small” to rapidly create quick and dirty throw-away programs, and “programming in the large” to create complete large and complex application programs.

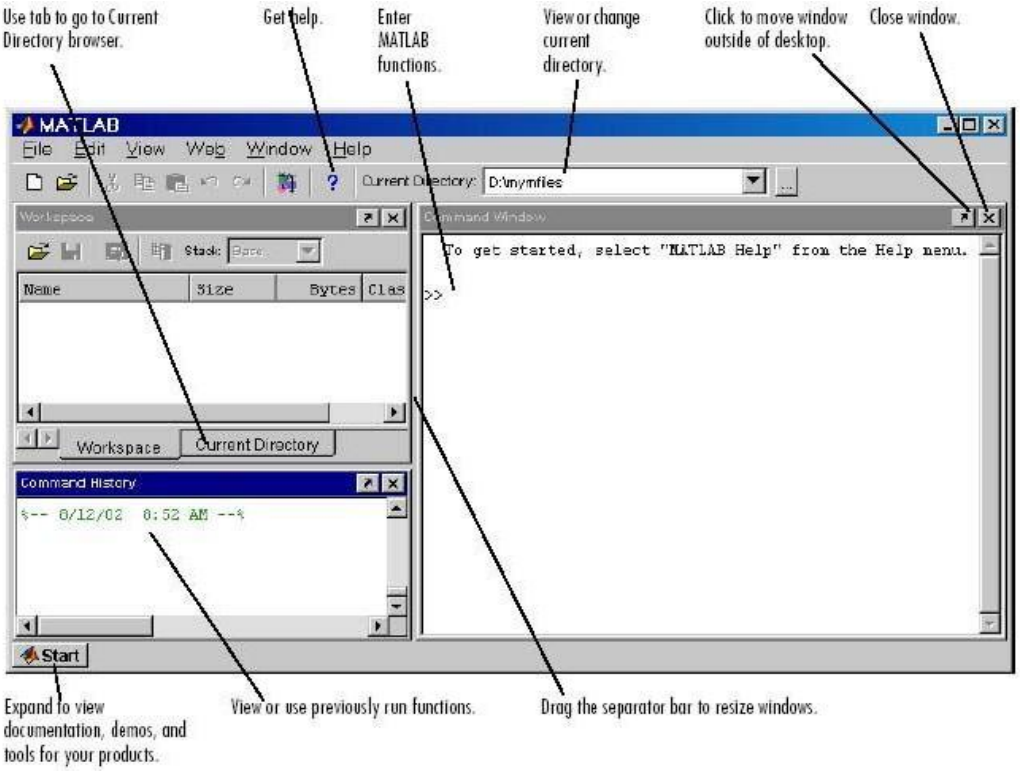
Graphics. MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize

the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

The MATLAB Application Program Interface (API). This is a library that allows you to write C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for readingand writing MAT-files.

MATLAB Desktop

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical userinterfaces) for managing files, variables, and applications associated with MATLAB.
The first time MATLAB starts, the desktop appears as shown in the following illustration.



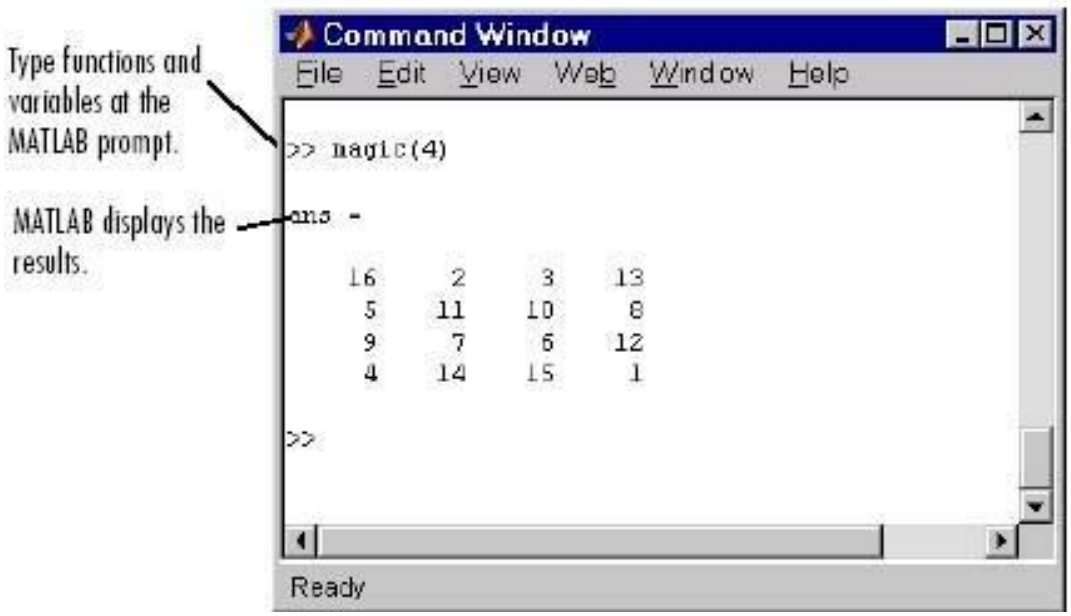
Desktop Tools

This section provides an introduction to the MATLAB desktop tools. You can also use MATLAB functions to perform most of the features found in the desktop tools. The tools are

1. "Command Window"
2. "Command History"
3. "Start Button and Launch Pad"
4. "Help Browser"
5. "Current Directory Browser"
6. "Workspace Browser"
7. "Array Editor"
8. "Editor/Debugger"
9. "Profiler"

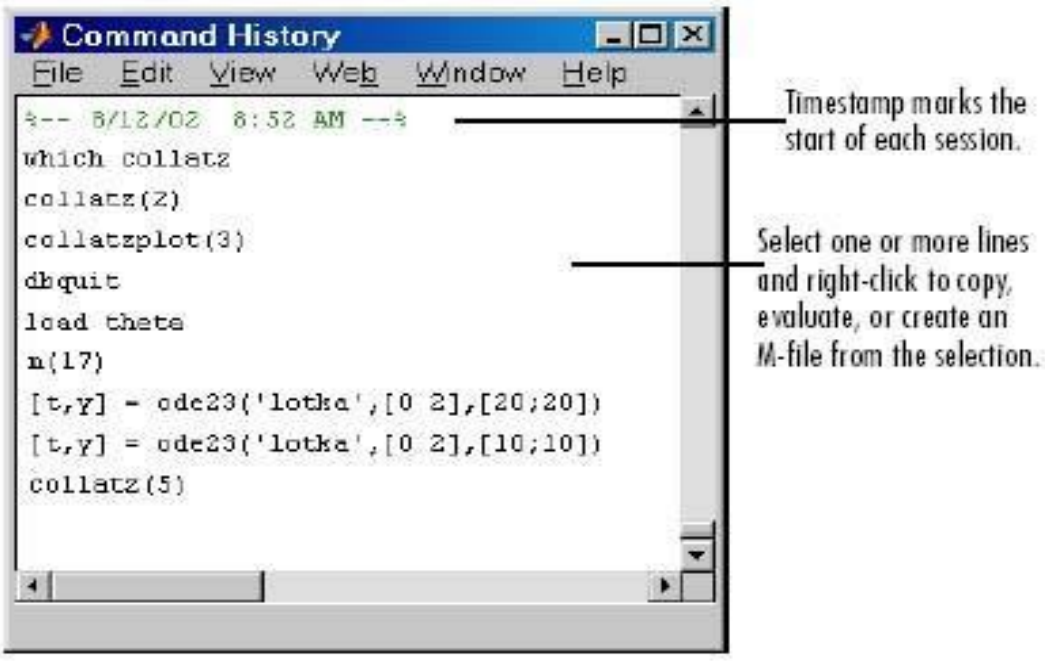
1. Command Window

Use the Command Window to enter variables and run functions and M-files.



2. Command History

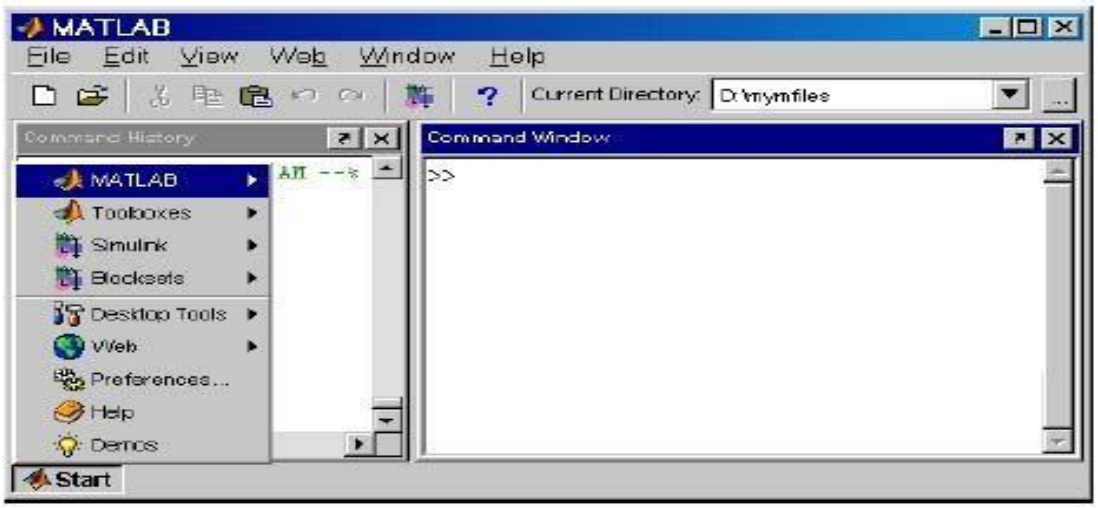
Statements you enter in the Command Window are logged in the Command History. In the Command History, you can view previously run statements, and copy and execute selected statements.



4

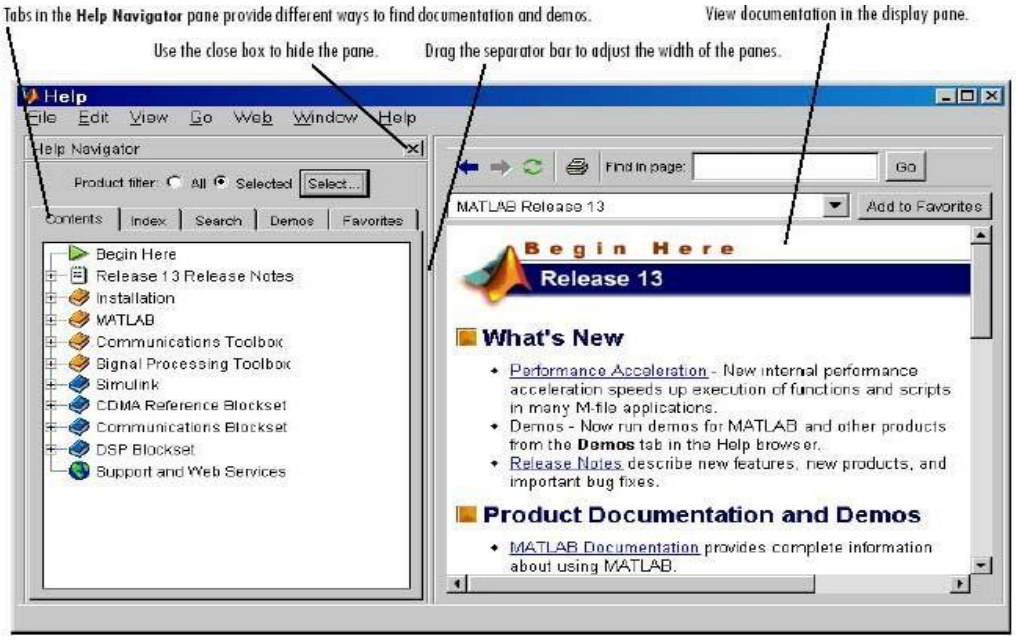
3. Start Button and Launch Pad

The MATLAB Start button provides easy access to tools, demos, and documentation. Just click the button to see the options.



4. Help Browser

Use the Help browser to search and view documentation and demos for all your MathWorks products. The Help browser is a Web browser integrated into the MATLAB desktop that displays HTML documents.To open the Help browser, click the help button in the toolbar, or type help browser in the Command Window.



For More Help

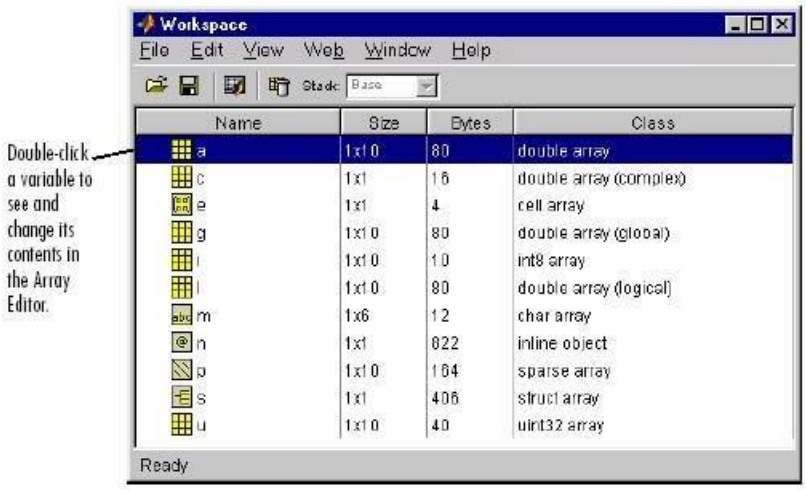
- In addition to the Help browser, you can use help functions. To get help for a specific function, use doc.
- For example, *doc format* displays documentation for the format function in the Help browser.
- If you type help followed by the function name, a briefer form of the documentation appears in the Command Window.
- Other means for getting help include contacting Technical Support (*http://www.mathworks.com*) and participating in the MATLAB file exchange, (*http://www.mathworks.com/matlabcentral/fileexchange/loadCategory.do*), MATLAB central (http://www.mathworks.com/matlabcentral/) and MATLAB group news (http://newsreader.mathworks.com/WebX?14@@/comp.soft-sys.matlab).

5. Current Directory Browser

MATLAB file operations use the current directory and the search path as reference points. Any file you want to run must either be in the current directory or on the search path.
A quick way to view or change the current directory is by using the Current Directory field in the desktop toolbar as shown below.

6. Workspace Browser

The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces. To view the workspace and information about each variable, use the Workspace browser, or use the functions *who* and *whos*.



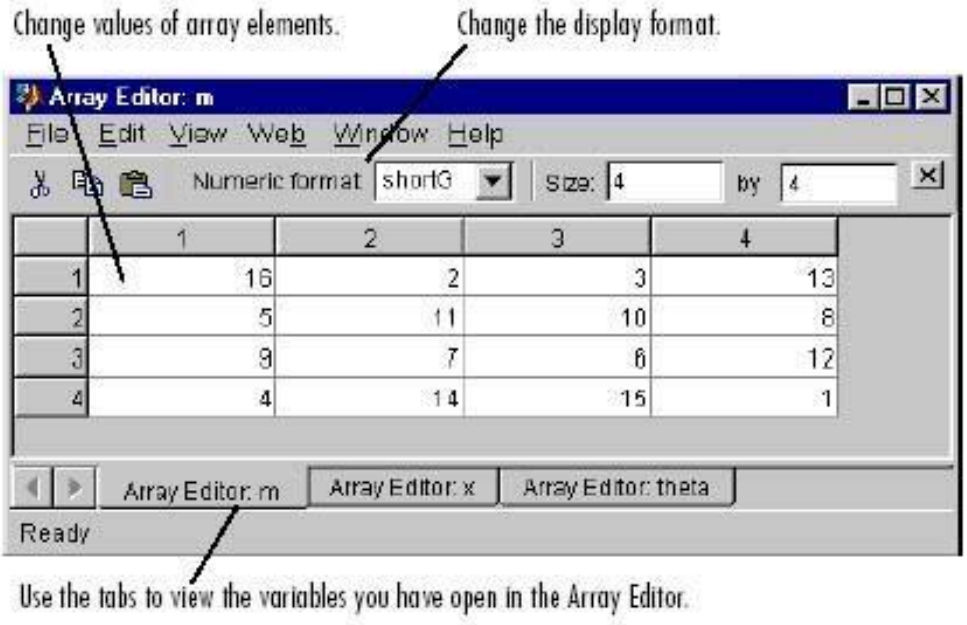
To delete variables from the workspace, select the variable and select Delete from the Edit menu. Alternatively, use the clear function. The workspace is not maintained after you end the MATLAB session.

To save the workspace to a file that can be read during a later MATLAB session, select Save Workspace As from the File menu, or use the save function. This saves the workspace to a binary file called a MAT-file, which has a .mat extension.

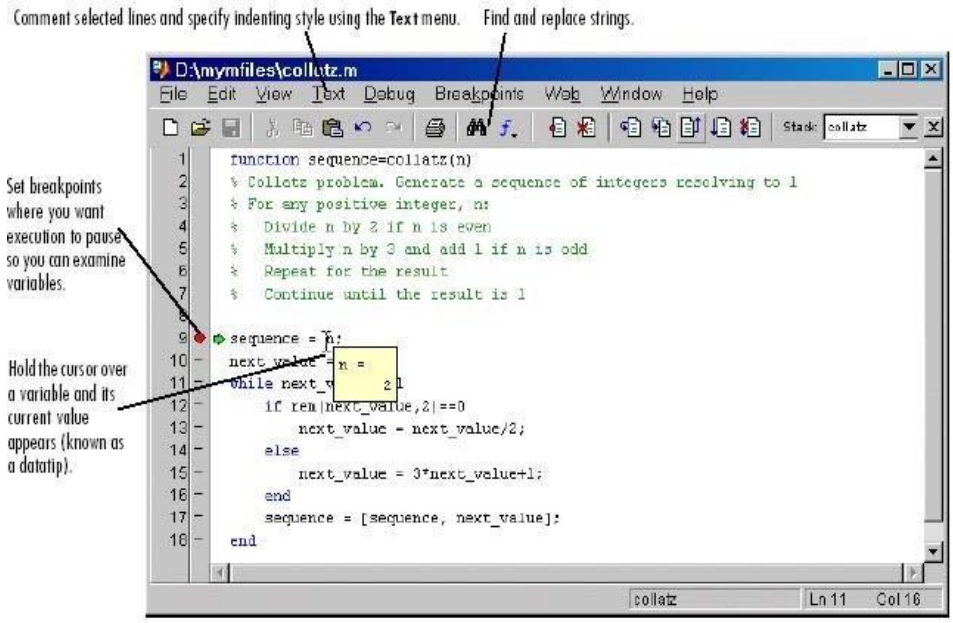
There are options for saving to different formats. To read in a MAT-file, select Import Data from the File menu, or use the load function.

7. Array Editor

Double-click a variable in the Workspace browser to see it in the Array Editor. Use the Array Editor to view and edit a visual representation of one- or two-dimensional numeric arrays, strings, and cell arrays of strings that are in the workspace.



8. Editor/Debugger

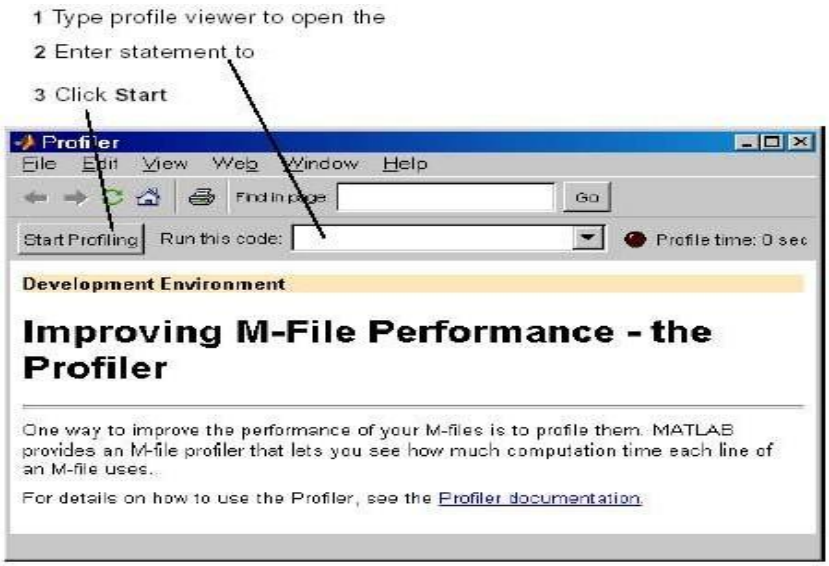


The Editor/Debugger provides a graphical user interface for basic text editing, as well as for M-file debugging.

You can use any text editor to create M-files, such as Emacs, and can use preferences (accessible from the desktop File menu) to specify that editor as the default. If you use another editor, you can still use the MATLAB Editor/Debugger for debugging, or you can use debugging functions, such as dbstop, which sets a breakpoint. If you just need to view the contents of an M-file, you can display it in the Command Window by using the type function.

9. Profiler

MATLAB includes a graphical user interface, the Profiler, to help you improve the performanceof your M-files.



LIBRARY FUNCTIONS:

Some of the important functions are discussed here. But it is not exhaustive. Other import functions are dealt whenever and/or where ever there they are necessary for future experiments or for simulations. Use MatLab '**help' command** to learn more about those used functions. Note that all the function names must be written or typed with small letters only and MatLab is case sensitive. i.e The variables 'Voltage' , voltage & VOLTAGE are treated as 3 different variables.

Be careful while typing the programs. All program file name must be a single word starting with an alphabet not with numbers. eg; mypgm1.m or 'my_pgm. m' or 'my_1_prog.m'

clc → clears the command window and homes the cursor.

clear all - Clear all the variables and functions from memory.

CLEAR removes all variables from the workspace. CLEAR VARIABLES does the same thing.

close all - Closes (removes) all the opened figure windows. CLOSE, by itself, closes the current figure window.

exp( ) - EXP(X) is the exponential of the elements of X, e to the power X($e^{x)}$.

input( ) - Prompt the user for input. e.g. R = input('How many apples'); gives the user the prompt in the text string and then waits for input from the keyboard. The input can be any MATLAB expression, which is evaluated, using the variables in the current workspace, and the result returned in R. If the user presses the return key without entering anything, INPUT returns an empty matrix.

tf ( ) - Construct transfer function or convert to transfer function. Construction or syntax or usage:

SYS = tf(NUM,DEN) creates a continuous-time transfer function SYS with numerator NUM anddenominator DEN.

linspace( ) - LINSPACE Linearly spaced vector.

LINSPACE(X1, X2) generates a row vector of 100 linearly equally spaced points between X1and X2.

ones( ): ONES(N) is an N-by-N matrix of ones. ONES(M,N) or

ONES([M,N]) is an M-by-N matrix of ones.

zeros( ) - ZEROS(N) is an N-by-N matrix of Zeros. ZEROS(M,N) or ZEROS([M,N]) is an M-by-N matrix of zeros

plot( ) - PLOT or draw the Linear plot /graph in the figure window. PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix, then the vector is plotted versus the rows or columns of the matrix, whichever line up.

subplot( ): SUBPLOT Create axes in tiled positions. H = SUBPLOT(m,n,p), or SUBPLOT(mnp), breaks the Figure window into an m-by-n matrix of small axes, selects the p-thaxes for the current plot, and returns the axis handle. The axes are counted along the top row of the Figure window, then the second row, etc.

stem( ) - STEM - Discrete sequence or "stem" plot. STEM(Y) plots the data sequence Y as stems from the x axis terminated with circles for the data value. STEM(X,Y) plots the data sequence Y at the values specified in X.

title( ) - TITLE used for the Graph title. TITLE('text') adds text at the top of the current axis.

xlabel( ) - X-axis labeling. XLABEL('text') adds text beside the X-axis on the current axis.

ylabel( ) - Y-axis labeling. YLABEL('text') adds text beside the Y-axis on the on the current axis.

Basic Programs in MATLAB

Ex.1 – Addition of two numbers

```
a = 3;
b = 5;
c = a+b
```

Output:

8

Remarks – (1) the semicolon at the end of a statement acts to suppress output (to keep the program running in a quiet mode). (2) the third statement C is equal to a + b is not followed by a semicolon so the content of the variable C is dumped as output.

Ex.2- Meaning of a = b

In Matlab and in any programming language, the statement a = b doesn't mean a equals b. Instead it prompts the action of replacing the content of a by the content of b

```
a =3;
b=a;
```

Output=3

Remarks- think of the two variables **a** and **b** as to bucket labeled **a** and **b.** The first statement put the number 3 into bucket **a** 2nd statement put the content of bucket a into bucket **b** such that we know have 3 in bucket **b. (**the content of bucket **a** remained unchanged after this action) the third statement dump the content of bucket **b** so the final output is 3

Ex.3- Basic Math operation

```
a=3;
b=9;
c=2*a+b^ 2-a*b+b/a-10
```

Output

53

Remark- the right hand side of the third statement includes are four of the basic arithmetic operation addition, subtraction, multiplication and division in their usual meaning. It also include the symbol ^ which means to **the power of**, So **b^2** means (the content of b) to the power of 2, that is 9^2 = 81. The right hand side of the statement is first evaluated: RHS is equal to 2*3 + 9^2 – 3*9 +9/3-10. The content of right hand side now 53, is then assigned to the variable **c** in the left hand side. since this statement is not followed by a semicolon the content of **C** is dumped as the final output

Ex.4 – Formatted output

```
a=3;
b=a*a;
c=a*a*a;
d=sqrt(a);
fprintf ('%4u square equals %4u\r',a,b)
fprintf ('%4u cube equals %4u\r',a,c)
fprintf ('The square root of %2u is %6.4f\r'a,d)
```

Output

```
3 square equals 9
3 cube equals 27
The square root of 3 is 1.7321
```

Remarks : The content of printf is formatted output ,using the format specified in the first string '…..' in the parenthesis.The "%4u"(4-digit integer) and "%6.4f"(real number that preserve 4 digits to the right of the floating point) are the format for the variable for output.The "sqrt" in the 4th statement is the intrinsic function for square root.

Ex.5: Continuation to next line

```
Summation1=1+3+5+7…

+9+11
```

Note: The three periods(…)allow continuation to the next line of commands. The two lines in the above example are essentially one line of "summation 1 is equal to 1 + 3 + 5 + 7 + 9 + 11".

Ex.6:clear a variable

```
c1=3;
c2=c1+5;
clear c1
c1
```

Output:

```
? ? ? Undefined function or variable 'c1'.
```

Remarks: we see an error message because the variable c1 no longer exist. It is purged from the Computer memory by the "clear" command. Note that the command doesn't just act to delete the content of a variable, but it kills the variable outright. The third statement can be useful if c1 is a big array that occupies a lot of memory but is no longer needed for the rest of the program .The third statement only kills c1,while c2(=8) still exists. A "clear" command not followed by any variable will kill all variables.

Ex.7- For loop

```
b = 3;
for k = 1:5
b
end
```

Output
3
3
3
3
3

Remark- The blue color segments in line 2-4 forms a **for loop.** The statement sand witch between for k = 1:5 and end is repeated 5 times with the k index going from 1 to 5 step 1.

Ex.8- Array within a loop

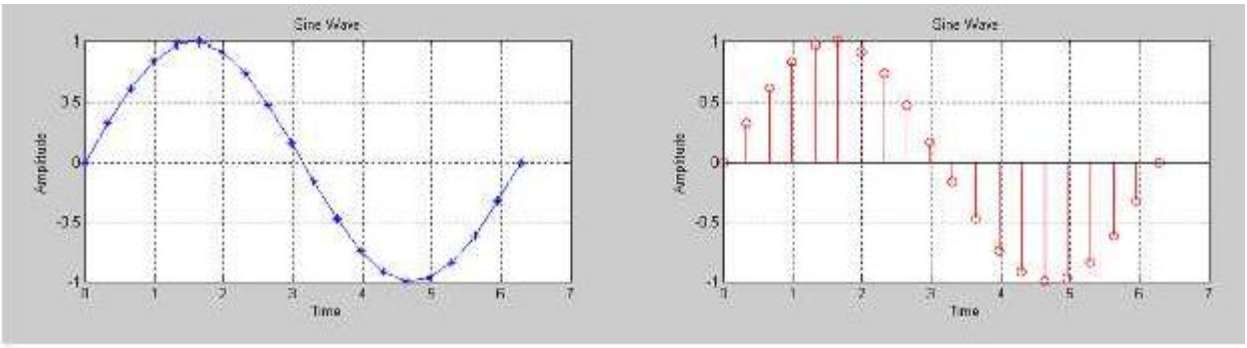
```
b = [3 8 9 4 7 5];
sum1 = 0;
for k = 1:4
sum1 = sum1 + b(k);
end
sum1
```

Output
24

Ex.9 – Generation of sine wave

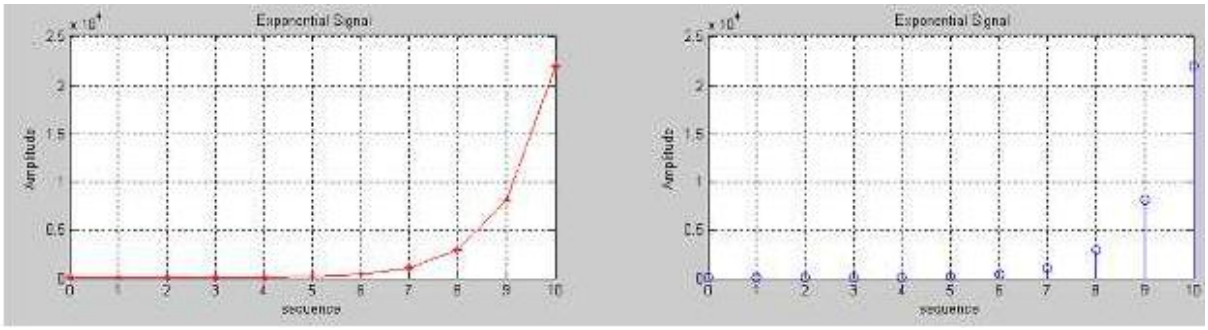
```
A = input('Enter the Value of Amplitude A =
');
S = input('Enter the No. of Sample S = ');
x =
linspace(0,(2*pi),S)
;
y = A*sin(x);

subplot(2,2
,1);
plot(x,y,'*
-b');grid
on;
xlabel('Time'); ylabel('Amplitude');
title('Sine Wave')
```

```
subplot(2,2,2);
stem(x,y,'
or');grid
on;
xlabel('Time'); ylabel('Amplitude');title('Sine Wave')
```



Ex.9 – Generation of Exponential wave

```
n2=input('Enter the No. of sequence
N=');
t=0:n2;
y2=exp(t);

subplot(2,2,1);
plot(t,y2,'*-r');
grid on;
xlabel('sequence'); ylabel('Amplitude'); title(' Exponential Signal');

subplot(2,2,2);
stem(t,y2,'ob');
grid on;
xlabel('sequence');  ylabel('Amplitude'); title(' Exponential Signal');
```



2. Basic matrix operations

The basic matrix operations are addition(+), subtraction(-), multiplication (*), and conjugate transpose(‘) of matrices. In addition to the above basic opera- tions, MATLAB has two forms of matrix division: the left inverse operator \ or the right inverse operator /.

Ex.1-Matrices of the same dimension may be subtracted or added. Thus if E and F are entered in MATLAB as

```
E = [7 2 3; 4 3 6; 8 1 5];

F = [1 4 2; 6 7 5; 1 9 1];
```

and

```
G = E - F
H = E + F
```

then, matrices G and H will appear on the screen as

```
G =
   6   -2    1
  -2   -4    1
   7   -8    4

H =
   8    6    5
  10   10   11
   9   10    6
```

A scalar (1-by-1 matrix) may be added to or subtracted from a matrix. In this particular case, the scalar is added to or subtracted from all the elements of another matrix.

Ex.2

```
J = H + 1
```

gives

```
J =
   9    7    6
  11   11   12
  10   11    7
```

Ex.3. Matrix multiplication is defined provided the inner dimensions of the two operands are the same. Thus, if X is an n-by-m matrix and Y is i-by-j matrix,

X*Y is defined provided m is equal to i.

Since E and F are 3-by-3 matrices, the product Q = E*F

results as

```
Q =
    22   69   27
    28   91   29
    19   84   26
```

Any matrix can be multiplied by a scalar. For example,

```
2*Q
```

gives

```
ans =
    44   138   54
    56   182   58
    38   168   52
```

Ex.4

Matrix division can either be the left division operator \ or the right division operator /. The right division a/b, for instance, is algebraically equivalent to a b while the left division a\b is algebraically equivalent to b/ a .

If Z*I = V and Z is non-singular, the left division, Z\V is equivalent to MATLAB expression

$$I = inv(Z)* V$$

where **inv** is the MATLAB function for obtaining the inverse of a matrix. The right division denoted by V/Z is equivalent to the MATLAB expression

$$I = V* inv(Z)$$

There are MATLAB functions that can be used to produce special matrices. Examples are given

Function	Description
ones(n,m)	Produces n-by-m matrix with all the elements being unity
eye(n)	gives n-by-n identity matrix
zeros(n,m)	Produces n-by-m matrix of zeros
diag(A)	Produce a vector consisting of diagonal of a square matrix A

Ex.5. Element by Element multiplication of two matricies

```
a = [2 3 5];
b = [2 4 9];
c = a.*b
```

Output:

```
c = 4   12   45
```

Remark: The period preceding the mathematical operation, "*", indicates that the operation will be performed element-by-element. In this case, the content of c is

```
c = [a(1)*b(1)  a(2)*b(2)  a(3)*b(3)]
```

Also, c is automatically assigned as a 1-D array with 3 elements

Ex.6- Elementary function with Vectorial variable

```
a = [2 3 5];
b = 2*a.^2+3*a+4
```

Output:

```
b = 18  31  69
```

Remark: The content of b is

```
b = [2*(a(1))^2+3*a(1)+4    2*(a(2))^2+3*a(2)+4    2*(a(3))^2+3*a(3)+4].
```

Ex. 7 Assign content of Array

```
a = [0:0.5:4];
a
```

Output:

```
a = 0  0.5  1  1.5  2  2.5  3  3.5  4
```

Ex.8. Extracting individual elements of matrix

```
A = [3  5; 2  4];
c = A(2,2)+A(1,2)
```

Output:

```
c = 9
```

Remark: With the given A matrix, we have A(1,1) = 3, A(1,2) = 5, A(2,1) = 2, and A(2,2) = 4.

Ex.9. Usage of index for matrix

```
A = [3  5; 2  4];
norm1 = 0;
for m = 1:2
for n = 1:2
  norm1 = norm1+A(m,n)^2;
end
end
norm1 = sqrt(norm1)
```

Output:

```
norm1 = 7.348
```

Remark: This program calculates the Euclidean norm of the A matrix.

3. Solution of linear equation

• **\ operator : A \ B** is the matrix division of A into B, which is roughly the same as **INV(A) * B**. If A is an NXN matrix and B is a column vector with N components or a matrix with several such columns, then **X = A \ B** is the solution to the equation **A * X = B**. A warning message is printed if A is badly scaled or nearly singular. **A\EYE(SIZE(A))** produces the inverse of A.

linsolve operator : X = LINSOLVE(A, B) solves the linear system A * X = B using LU factorization with partial pivoting when A is square, and QR factorization with column pivoting. A warning is given if A is ill conditioned for square matrices and rank deficient for rectangular matrices.

Ex.1 : Non-homogeneous System Ax = b, where A is a square and is invertible. In our example we will consider the following equations:

2x + y – z = 7

x – 2y +5z = -13

3x + 5y -4z = 18

We will convert these equations into matrices A and b :

```
% declaring the matrices based
on the equations
A = [2 1 -1; 1 -2 5; 3 5
-4]
b = [7; -13; 18]

Output

A =    2 1 -1
       1 -2 -5
       3 5 -4

b =    7
      -13
      18
```

4. Solution of Linear equations for Underdetermined and over determined cases.

Underdetermined

These are equations in which m>n. It means that the provided information is insufficient to give a solution to the problem.

Ex.1. 4x + 5y =6

Mathematically, the solution is y = (6-4x)/5. It means that the x value can range from -inf to inf as long as it works with the provided y. If Matlab is used to solve such equations, it will give only one value and the other set to 0.

```
A= [4 5];
b= 6;
X= A\b
```

The output will be;

```
X = 0

    1.2000
```

Over determined

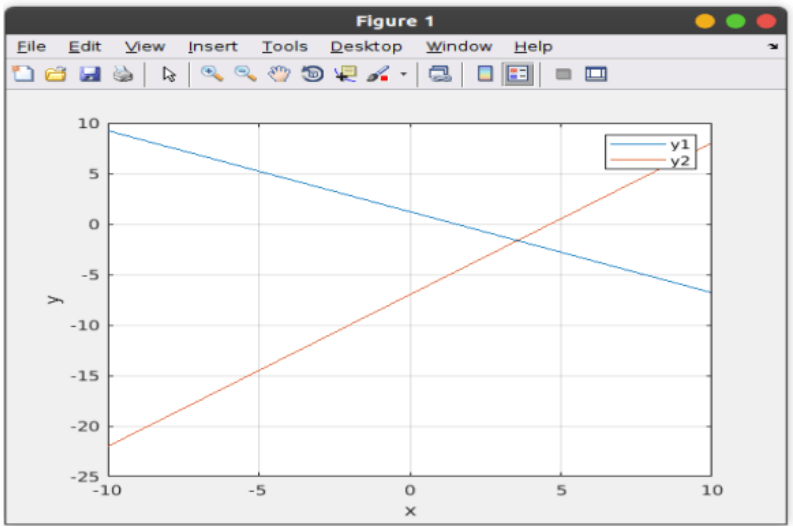
These kind of equations mainly occur when m>n. Here, the information provided by the equations is too much. The equation Ax=b cannot be satisfied simultaneously by any value of the vector x. When you solve the equation using Matlab, it will give an output, but it does not satisfy the matrix rule a*x=b.

Ex.2

$$\begin{cases} 4x + 5y = 6 \\ 3x - 2y = 14 \end{cases}$$

For the first equation, y1 = -(4x - 6)/5, plot y2 and y1 against x.

```
x = [ -10;10];
y1 = (-4*x + 6) /5;
y2 = (3*x -14)/2;
plot (x, y1, x, y2);
grid on
xlabel (‘x’);
ylabel (‘y’);
legend (‘y1’, ‘y2’)
```



5. Determination of Eigen values and Eigen vectors of a Square matrix.

An eigenvalues and eigenvectors of the square matrix A are a scalar λ and a nonzero vector v that satisfy

$$\mathbf{Av} = \lambda\mathbf{v}$$

In this equation, **A** is a n-by-n matrix, **v** is non-zero n-by-1 vector, and λ is the scalar (which might be either real or complex). Any value of the λ for which this equation has a solution known as eigenvalues of the matrix **A**. It is also called the **characteristic value.** The vector, **v**, which corresponds to this equation, is called eigenvectors**.** The eigenvalues problem can be written as,

$$\mathbf{A}\cdot\mathbf{v}-\lambda\cdot\mathbf{v}=0$$
$$\mathbf{A}\cdot\mathbf{v}-\lambda\cdot\mathbf{I}\cdot\mathbf{v}=0$$
$$(\mathbf{A}-\lambda\cdot\mathbf{I})\cdot\mathbf{v}=0$$

If **v**is a non-zero, this equation will only have the solutions if

$$|\mathbf{A}-\lambda\cdot\mathbf{I}|=0$$

This equation is called the characteristic equations of **A**, and is a n^{th} order polynomial in λ with n roots. These roots are called the eigenvalue of **A**. We will only handle the case of n distinct roots; through which they may be repeated. For each eigenvalue, there will be eigenvectors for which the eigenvalue equations are true.

Matlab allows the users to find eigenvalues and eigenvectors of matrix using **eig()** method. Different **syntaxes** of **eig()** method are:

- e = eig(A)
- [V,D] = eig(A)
- [V,D,W] = eig(A)
- e = eig(A,B)

Ex.1

e = eig(A)

- It returns the vector of **eigenvalues** of **square matrix A**

```
% Square matrix of size 3*3
A = [0 1 2;
   1 0 -1;
   2 -1 0];
disp("Matrix");
disp(A);

% Eigenvalues of matrix A
e = eig(A);
disp("Eigenvalues");
disp(e);
```

Output

```
Matrix
     0     1     2
     1     0    -1
     2    -1     0

Eigenvalues
   -2.7321
    0.7321
    2.0000
```

Ex.2. [V,D] = eig(A)

- It returns the **diagonal matrix D** having diagonals as **eigenvalues.**
- It also returns the matrix of **right vectors as V**.
- Normal **eigenvectors** are termed as **right eigenvectors**.
- **V** is a collection of N eigenvectors of each N*1 size(A is N*N size) that satisfies **A*V = V*D**

```
% Square matrix of size 3*3
A = [8 -6 2;
   -6 7 -4;
   2 -4 3];
disp("Matrix");
disp(A);

% Eigenvalues and right eigenvectors of matrix A
[V,D] = eig(A);
disp("Diagonal matrix of Eigenvalues");
disp(D);
disp("Right eigenvectos")
disp(V);
```

Output

```
Matrix :
     8    -6     2
    -6     7    -4
     2    -4     3

Diagonal matrix of Eigenvalues :
    0.0000         0         0
         0    3.0000         0
         0         0   15.0000

Right eigenvectors :
    0.3333    0.6667   -0.6667
    0.6667    0.3333    0.6667
    0.6667   -0.6667   -0.3333
```

Ex.3

[V,D,W] = eig(A)

- Along with the diagonal matrix of eigenvalues **D** and right eigenvectors **V**, it also returns the **left eigenvectors** of matrix A.
- A left eigenvector **u** is a **1*N matrix** that satisfies the equation **u*A = k*u**, where **k** is a **left eigenvalue of matrix A**.
- **W** is the collection of **N left eigenvectors** of A that satisfies **W'*A = D*W'**.

```
% Square matrix of size 3*3
A = [10 -6 2;
    -6 7 -4;
    2 -4 3];
disp("Matrix :");
disp(A);

% Eigenvalues and right and left eigenvectors
% of matrix A
[V,D,W] = eig(A);
disp("Diagonal matrix of Eigenvalues :");
disp(D);
disp("Right eigenvectors :")
disp(V);
disp("Left eigenvectors :")
disp(W);
```

Output

```
Matrix :
    10    -6     2
    -6     7    -4
     2    -4     3

Diagonal matrix of Eigenvalues :
    0.1618         0         0
         0    3.8694         0
         0         0   15.9688

Right eigenvectors :
    0.2411    0.6460   -0.7242
    0.6389    0.4561    0.6195
    0.7305   -0.6121   -0.3028

Left eigenvectors :
    0.2411    0.6460   -0.7242
    0.6389    0.4561    0.6195
    0.7305   -0.6121   -0.3028
```

Ex.4.

e = eig(A,B)

- It returns the **generalized eigenvalues** of two square matrices **A and B** of the same size.
- A generalized eigenvalue **λ** and a corresponding **eigenvector v** satisfy **Av=λBv**.

```
% Square matrix A and B of size 3*3
A = [10 -6 2;
      -6 7 -4;
      2 -4 3];
B = [8 6 1;
      6 17 2;
      -1 4 3];
      disp("Matrix A:");
disp(A);
disp("Matrix B:");
disp(B);

% Generalized eigen values
% of matrices A and B
e = eig(A,B);
disp("Generalized eigenvalues :")
disp(e);
```

Output

```
Matrix A:
    10    -6     2
    -6     7    -4
     2    -4     3

Matrix B:
     8     6     1
     6    17     2
    -1     4     3

Generalized eigenvalues :
    4.4423
    0.7271
    0.0117
```

6. Solution of Difference Equations.

Ex.1 Solve this differential equation.

dy / dt = ty

```
syms y(t)
ode = diff(y,t) == t*y
ySol(t) = dsolve(ode)
```

```
Output

ode(t) =
diff(y(t), t) == t*y(t)

ySol(t) =
 C3*exp(t^2/2)
```

Ex2. Second-Order ODE with Initial Conditions

$$\frac{d^2y}{dx^2} = \cos(2x) - y,$$
$$y(0) = 1,$$
$$y'(0) = 0.$$

Define the equation and conditions. The second initial condition involves the first derivative of y. Represent the derivative by creating the symbolic function Dy = diff(y) and then define the condition using Dy(0)==0.

```
syms y(x)
Dy = diff(y);

ode = diff(y,x,2) == cos(2*x)-y;
cond1 = y(0) == 1;
cond2 = Dy(0) == 0;
Solve ode for y. Simplify the solution using the simplify function.

conds = [cond1 cond2];
ySol(x) = dsolve(ode,conds);
ySol = simplify(ySol)
ySol(x) =
1 - (8*sin(x/2)^4)/3
```

output

```
ySol(x) =

1 - (8*sin(x/2)^4)/3


ySol(x) =

1 - (8*sin(x/2)^4)/3
```

7. Solution of Difference Equations using Euler Method.

Euler's method is a numerical method to solve first order first degree differential equation with a given initial value. It is the most basic explicit method for numerical integration of ordinary differential equations and is the simplest Runge–Kutta method. The Euler method is a first-order method, which means that the local error (error per step) is proportional to the square of the step size, and the global error (error at a given time) is proportional to the step size.

Example:

Enter initial value of x i.e. x0: 0
Enter initial value of y i.e. y0: 0.5
Enter the final value of x: 2
Enter the step length h: 0.2
x y
0.000 0.500
0.200 0.600
0.400 0.760
0.600 0.992
0.800 1.310
1.000 1.732
1.200 2.279
1.400 2.975
1.600 3.850
1.800 4.940
2.000 6.288

The Program:

```
%function t=t(n,t0,t1,y0)
function y=y(n,t0,t1,y0)
h=(t1-t0)/n;
t(1)=t0;
y(1)=y0;
for i=1:n
t(i+1)=t(i)+h;
y(i+1)=y(i)+h*ex(t(i),y(i));
end;
V=[t',y']
plot(t,y)
```

8. Solution of differential equation using 4th order Runge- Kutta method.

The Runge-Kutta method is the most popular method for solving ordinary differential equations (ODEs) by means of numerical approximations. There are several version of the method depending on the desired accuracy. The most popular Runge-Kutta method is of fourth order, which in simpler terms means that the error is of the order of h4 , abbreviated as O(h4). We will attempt to solve a simple differential equation with the simplest Runge-Kutta method available. The formulas for the fourth-order Runge-Kutta are

Compute slope at four places within each step

$$k_1 = f(t_j, y_j)$$

$$k_2 = f(t_j + \frac{h}{2}, y_j + \frac{h}{2}k_1)$$

$$k_3 = f(t_j + \frac{h}{2}, y_j + \frac{h}{2}k_2)$$

$$k_4 = f(t_j + h, y_j + hk_3)$$

Use weighted average of slopes to obtain y_{j+1}

$$y_{j+1} = y_j + h\left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right)$$

$$\text{LDE} = \text{GDE} = \mathcal{O}(h^4)$$

Where,

h = (b-a)/n, the step size of the independent variable

n = the number of values of the independent variable in which we wish to calculate the approximate solution

[a, b] = the limits of the t interval.

k1, k2, k3, k4 Runge-Kutta parameters or functions (actually 'slopes')

f(ti,yi)=function evaluated at ti and yi

$$f(t_i + \frac{1}{2}h,\ y_i + \frac{1}{2}k_1) = \text{function evaluated at } t_i + \frac{1}{2}h,\ y_i + \frac{1}{2}k_1$$

In order to calculate a new point in the solution yi+1 you need the previous solution yi and k1 , k2 , k3 , and k4 in that order. Then the calculation sequence is k1, k2, k3, k4, and then yi+1.

Ex.1. MATLAB program to implement the fourth-order Runge-Kutta method to solve

$$y' = 3e^{-t} + 0.4y,\ \ y(0) = 5\ on\ \ t = [0,3]\ .$$

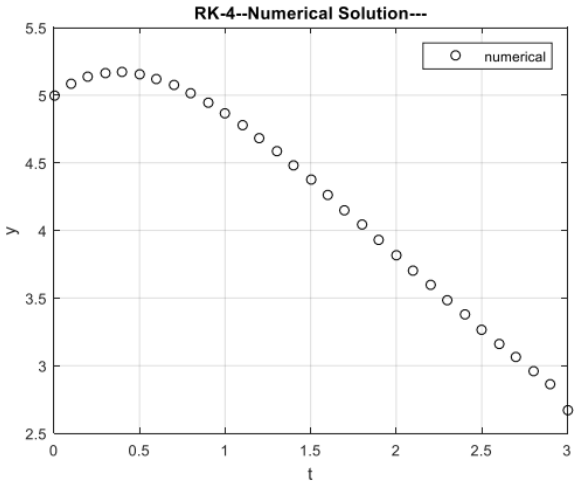
For h= 0.1; The RK method is highly accurate for small h.

```
clc, clear all, close all

% instruction to write results on external
filefid=fopen('RKout.m','w');

h=0.1; a=0; b=3;              % h is the step size, t=[a,b] t-
range
t = a:h:b;                    % Computes t-array up to t=3
y = zeros(1,numel(t));      % Memory preallocation
y(1) = 5;          % initial condition; in MATLAB indices start at 1

Fyt = @(t,y) 3.*exp(-t)-0.4*y;     % change the function as you desire
                    % the function is the expression after (t,y)
% table title
fprintf(fid,'%7s %7s %7s %7s %7s %7s %7s
\n','i','t(i)','k1','k2','k3','k4','y(i)');

for  ii=1:1:numel(t)
    k1 = Fyt(t(ii),y(ii));
    k2 = Fyt(t(ii)+0.5*h,y(ii)+0.5*h*k1);
    k3 = Fyt((t(ii)+0.5*h),(y(ii)+0.5*h*k2));
    k4 = Fyt((t(ii)+h),(y(ii)+h*k3));


y(ii+1) = y(ii) + (h/6)*(k1+2*k2+2*k3+k4);  % main equation
% table data
fprintf(fid,'%7d %7.2f %7.3f %7.3f',ii, t(ii), k1, k2);
fprintf

end

 (fid,' %7.3f %7.3f %7.3f \n', k3, k4, y(ii));


y(numel(t))=[ ];       % erase the last computation of y(n+1)

% Solution PLOT:
plot(t,y,' ok')
title('RK-4--Numerical Solution---');
ylabel('y');    xlabel('t');
                legend('numerical'); grid on

fclose(fid);
```

MATLAB solution

RKout.m (output file):

i	t(i)	k1	k2	k3	k4	y(i)
1	0.00	1.000	0.834	0.837	0.681	5.000
2	0.10	0.681	0.535	0.538	0.401	5.084
3	0.20	0.401	0.273	0.276	0.156	5.138
4	0.30	0.156	0.045	0.047	-0.057	5.165
5	0.40	-0.057	-0.154	-0.152	-0.242	5.170
6	0.50	-0.242	-0.326	-0.324	-0.402	5.155
7	0.60	-0.402	-0.475	-0.473	-0.540	5.122
8	0.70	-0.540	-0.602	-0.601	-0.658	5.075
9	0.80	-0.658	-0.711	-0.709	-0.758	5.015
10	0.90	-0.758	-0.802	-0.801	-0.842	4.944
11	1.00	-0.842	-0.879	-0.878	-0.912	4.864
12	1.10	-0.912	-0.942	-0.942	-0.969	4.776
13	1.20	-0.969	-0.994	-0.993	-1.015	4.682
14	1.30	-1.015	-1.035	-1.035	-1.052	4.583
15	1.40	-1.052	-1.067	-1.067	-1.080	4.479
16	1.50	-1.080	-1.091	-1.090	-1.100	4.372
17	1.60	-1.100	-1.107	-1.107	-1.113	4.263
18	1.70	-1.113	-1.118	-1.117	-1.121	4.153
19	1.80	-1.121	-1.122	-1.122	-1.123	4.041
20	1.90	-1.123	-1.122	-1.122	-1.121	3.929
21	2.00	-1.121	-1.118	-1.118	-1.115	3.817
22	2.10	-1.115	-1.110	-1.110	-1.105	3.705
23	2.20	-1.105	-1.099	-1.099	-1.093	3.594
24	2.30	-1.093	-1.086	-1.086	-1.078	3.484
25	2.40	-1.078	-1.070	-1.070	-1.061	3.375
26	2.50	-1.061	-1.052	-1.052	-1.042	3.268
27	2.60	-1.042	-1.032	-1.033	-1.022	3.163
28	2.70	-1.022	-1.012	-1.012	-1.001	3.060
29	2.80	-1.001	-0.990	-0.990	-0.979	2.959
30	2.90	-0.979	-0.967	-0.968	-0.956	2.860
31	3.00	-0.956	-0.944	-0.944	-0.932	2.763


RK-4--Numerical Solution---
numerical
y
t
5.5
5
4.5
4
3.5
3
2.5
0
0.5
1
1.5
2
2.5
3

9. Determination of roots of a polynomial

Roots of a polynomial are the values for which the polynomial equates to zero. So, if we have a polynomial in 'x', then the roots of this polynomial are the values that can be substituted in place of 'x' to make the polynomial equal to zero. Roots are also referred to as Zeros of the polynomial.

1. If a polynomial has real roots, then the values of the roots are also the x-intercepts of the polynomial.
2. If there are no real roots, the polynomial will not cut the x-axis at any point.

In MATLAB we use 'roots' function for finding the roots of a polynomial.

Syntax:

R = roots (Poly)

Description:

- R = roots (Poly) is used to find the roots of the input polynomial
- The input polynomial is passed as an argument in the form of a column vector
- For a polynomial of degree 'p', this column vector contains 'p+1' coefficients of the polynomial

Ex.1. x^2 – x – 6

```
Poly = [1 -1 -6];
%Creating the column vector for the input polynomial]
R = roots(Poly)
Output
Poly = 1 -1 -6
R = 3
    -2
```

Ex.2 x^3 –5x^2 + 2x+8

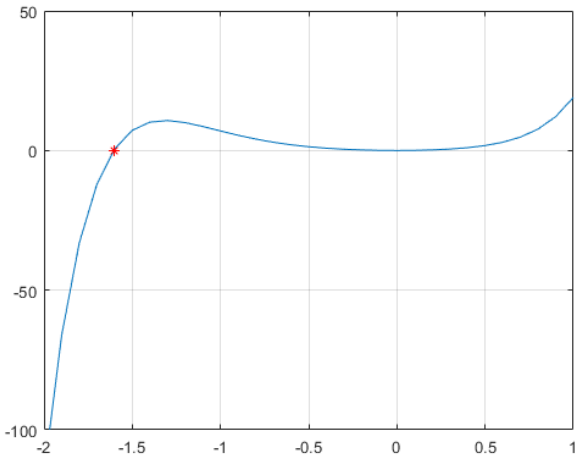
```
Poly = [1 -5 2 8];
R = roots(Poly)
Output
R = 4.0000
       2.0000
      -1.0000
```

Ex.3. Roots in a Specific Interval

Use the fzero function to find the roots of a polynomial in a specific interval. Among other uses, this method is suitable if you plot the polynomial and want to know the value of a particular root.

create a function handle to represent thepolynomial $3x^7+4x^6+2x^5+4x^4+x^3+5x^2$.

```
p = @(x) 3*x.^7 + 4*x.^6 + 2*x.^5 + 4*x.^4 + x.^3 + 5*x.^2;
Plot the function over the interval [-2,1].

x = -2:0.1:1;
plot(x,p(x))
ylim([-100 50])
grid on
hold on
Z = fzero(p, -1.5)
Z = -1.6056
plot(Z,p(Z),'r*')
```



Ex.5. Represent Roots of High-Degree Polynomial

Represent the roots of the polynomial x^3+1 using root. The root function returns a column vector. The elements of this vector represent the three roots of the polynomial.

```
syms x
p = x^3 + 1;
root(p,x)

ans =

 root(x^3 + 1, x, 1)
 root(x^3 + 1, x, 2)
 root(x^3 + 1, x, 3)
root(x^3 + 1, x, 1)
```

represents the first root of p, while root(x^3 + 1, x, 2) represents the second root, and so on. Use this syntax to represent roots of high-degree polynomials.

10. Determination of polynomial using method of Least Square Curve Fitting.

Curve Fitting Toolbox™ software uses the method of least squares when fitting data. Fitting requires a parametric model that relates the response data to the predictor data with one or more coefficients. The result of the fitting process is an estimate of the model coefficients.

To obtain the coefficient estimates, the least-squares method minimizes the summed square of residuals. The residual for the *i*th data point r_i is defined as the difference between the observed response value y_i and the fitted response value $\hat{y}_i$, and is identified as the error associated with the data.

The Curve Fitting Toolbox is a collection of graphical user interfaces (GUIs) and M-file functions built on the MATLAB® technical computing environment.

The toolbox provides you with these main features:

• Data preprocessing such as sectioning and smoothing

• Parametric and nonparametric data fitting: - You can perform a parametric fit using a toolbox library equation or using a custom equation. Library equations include polynomials, exponentials, rationals, sums of Gaussians, and so on. Custom equations are equations that you define to suit your specific curve fitting needs. - You can perform a nonparametric fit using a smoothing spline or various interpolants.

• Standard linear least squares, nonlinear least squares, weighted least squares, constrained least squares, and robust fitting procedures

• Fit statistics to assist you in determining the goodness of fit

• Analysis capabilities such as extrapolation, differentiation, and integration

• A graphical environment that allows you to: - Explore and analyze data sets and fits visually and numerically - Save your work in various formats including M-files, binary files, and workspace variables

The Curve Fitting Toolbox consists of two different environments:

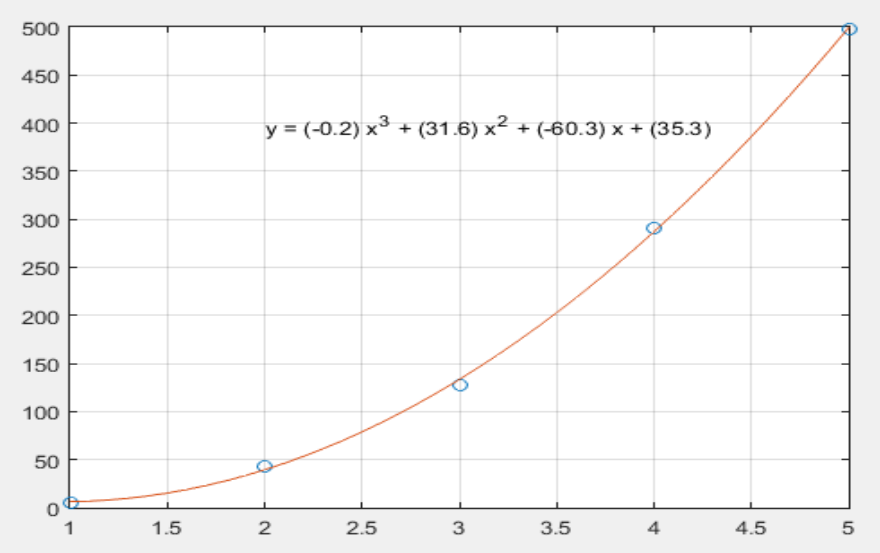
• The Curve Fitting Tool, which is a graphical user interface (GUI) environment

• The MATLAB command line environment You can explore the Curve Fitting Tool by typing cftool Click the GUI Help buttons to learn how to proceed. Additionally, you can follow the examples in the tutorial sections of this guide, which are all GUI oriented.

Ex. Program for Polynomial curve fitting

```
x = [1 2 3 4 5];

y = [5.5 43.1 128 290.7 498.4];
p = polyfit(x,y,3)
x2 = 1:.1:5;
y2 = polyval(p,x2);
plot(x,y,'o',x2,y2)
grid on
s = sprintf('y = (%.1f) x^3 + (%.1f) x^2 + (%.1f) x +
(%.1f)',p(1),p(2),p(3),p(4));
text(2,400,s)
```

```
Output
p =
   -0.1917   31.5821  -60.3262   35.3400
```



11. Determination of polynomial fit, analyzing residuals, exponential fit and error bounds from the given data.

Polynomial Fit Functions

Function	Description
polyfit	polyfit(x,y,n) finds the coefficients of a polynomial p(x) of degree n that fits the y data by minimizing the sum of the squares of the deviations of the data from the model (least-squares fit).
polyval	polyval(p,x) returns the value of a polynomial of degree n that was determined by polyfit, evaluated at x.

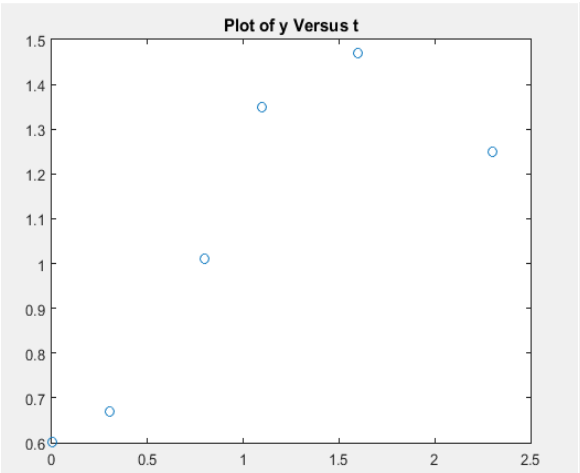
Ex. Program for the determination of polynomial fit, analyzing residuals, exponential fit and error bounds

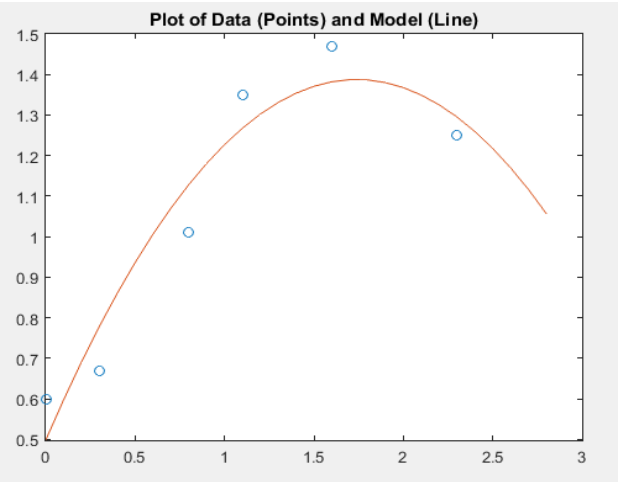
```
t = [0 0.3 0.8 1.1 1.6 2.3];
y = [0.6 0.67 1.01 1.35 1.47 1.25];
plot(t,y,'o')
title('Plot of y Versus t')
%%
p = polyfit(t,y,2)
t2 = 0:0.1:2.8;
y2 = polyval(p,t2);
figure
plot(t,y,'o',t2,y2)
title('Plot of Data (Points) and Model (Line)')

%% Residuals

y2 = polyval(p,t);
res = y - y2;
figure, plot(t,res,'+')
title('Plot of the Residuals')

%%5th degree polynomial fit

p5 = polyfit(t,y,5)
y3 = polyval(p5,t2);
figure
plot(t,y,'o',t2,y3)
title('Fifth-Degree Polynomial Fit')

%%Exponential fit

t = [0 0.3 0.8 1.1 1.6 2.3]';
y = [0.6 0.67 1.01 1.35 1.47 1.25]';
X = [ones(size(t))  exp(-t)  t.*exp(-t)];
a = X\y
T = (0:0.1:2.5)';
Y = [ones(size(T))  exp(-T)  T.*exp(-T)]*a;
plot(T,Y,'-',t,y,'o'), grid on
title('Plot of Model and Original Data')
```

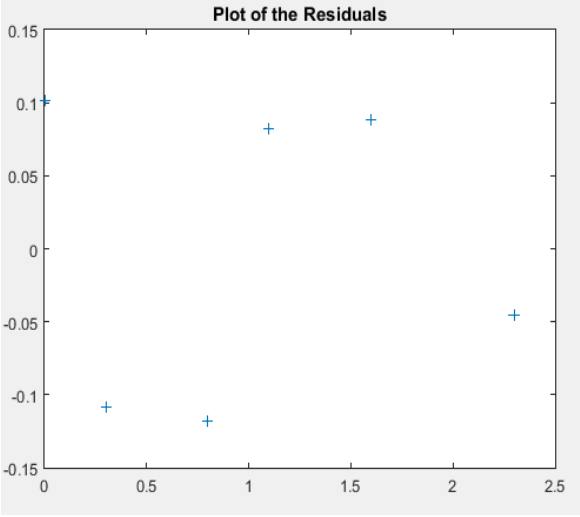
```
%%Error bound

x1 = [.2 .5 .6 .8 1.0 1.1]';
x2 = [.1 .3 .4 .9 1.1 1.4]';
y  = [.17 .26 .28 .23 .27 .24]';
X = [ones(size(x1))  x1  x2];
a = X\y
Y = X*a;
MaxErr = max(abs(Y - y))
```

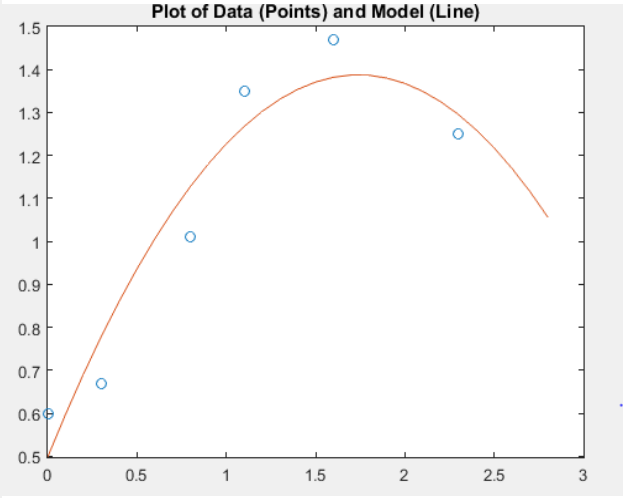
Output

```
p =
   -0.1917   31.5821  -60.3262   35.3400

p =
   -0.1917   31.5821  -60.3262   35.3400

p =
   -0.2942    1.0231    0.4981

p5 =
 0.7303   -3.5892    5.4281   -2.5175    0.5910    0.600

a =

    1.3983
   -0.8860
    0.3085

a =
    0.1018
    0.4844
   -0.2847

MaxErr =
    0.0038
```


Plot of y Versus t


Plot of Data (Points) and Model (Line)

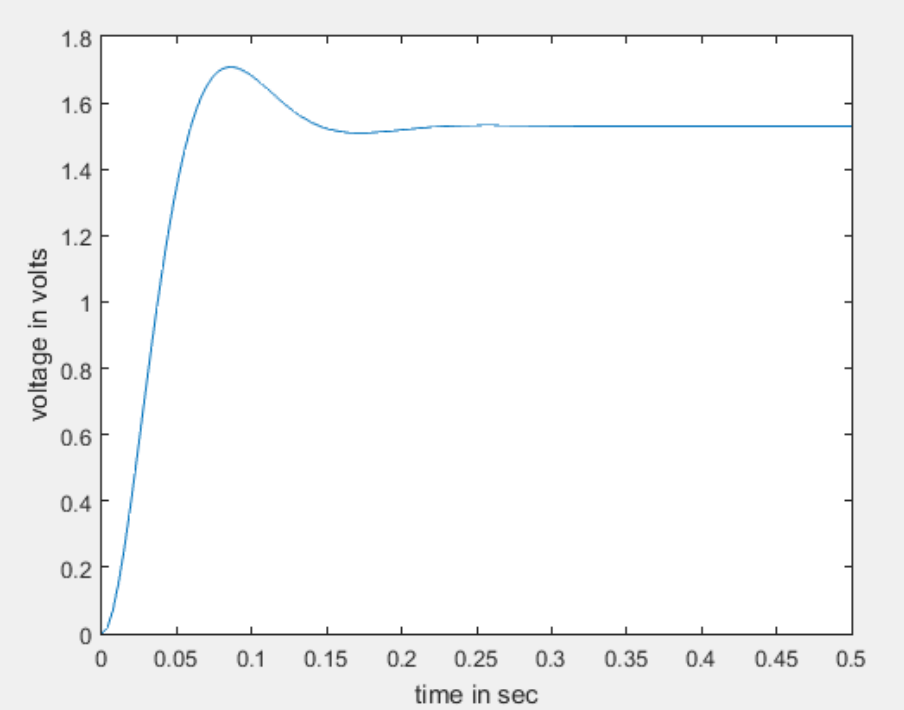

Plot of the Residuals


Plot of Data (Points) and Model (Line)

12. Determination of time response of an R-L-C circuit.

```
s = tf('s');
          R = 10;                  % resistor resistance
          Req = 40;                % inductor equivalent series resistance
(ESR)
          L = 1;                   % inductor inductance
          C = 510*10^-6;           % capacitor capacitance
          ei = 1.53;               % battery voltage
          tstep = 1.22;            % time step occurred
          G = 1/(C*L*s^2 + C*(R+Req)*s + 1);     % model transfer function
         [y,t] = step(G*ei,0.5);
          plot(t,y)
          xlabel('time in sec')
          ylabel('voltage in volts')
```

Output:



SIMULINK

Introduction to Simulink

Simulink is a software package for modeling, simulating and analyzing dynamical systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Simulink can easily accommodate multirate systems as well. Simulink offers a friendly, graphical environment, in which you can model, systems in the form of block diagrams, by simply clicking and dragging blocks into a model window. You can run a model at the push of a button and modify it easily. The graphical nature of Simulink models makes them easy forothers to read and understand.

Simulink has a comprehensive block library of sinks, sources, subsystems (linear, nonlinear, and time-varying), connectors, and powerful conditionally-enabled blocks. You can also customize your own blocks.You can use all the facilities of MATLAB when running Simulink. You can invoke familiarMATLAB expressions and M-file functions as temporary utilities, for instance to control display and visualization. You can even encapsulate them as blocks and place them in Simulink models. Using Simulink, you can see data displayed in scopes as the simulation unfolds. This makes tracing and debugging a model, by quick, proof-of-concept demonstrations, much faster and easier than by working directly from the command line. Outside the simulation environment, Simulink serves as the primary link for targeting to chips, boards, and co-simulation platformsby means of automatic code generation.

Blocksets are comprehensive collections of Simulink blocks that extend the Simulink environment to solve particular classes of problems. Areas in which blocksets are available include digital signal processing, communications, embedded target for TI C6000, Xilinx and many others.

What Is the Communications Blockset

The Communications Blockset is a collection of Simulink® blocks for designing and simulating communication systems. With the Communications Blockset, you can design models in the formof block diagrams, using simple click-and-drag mouse operations. You can run simulations on a model at the push of a button, and change parameters while the simulation is running. The Communications Blockset contains ready-to-use blocks to model various processes within communication systems, including

- Signal generation
- Source coding
- Error-correction
- Interleaving
- Modulation/demodulation
- Transmission along a channel
- Synchronization

In addition, you can create specialized blocks, to implement your own algorithms.

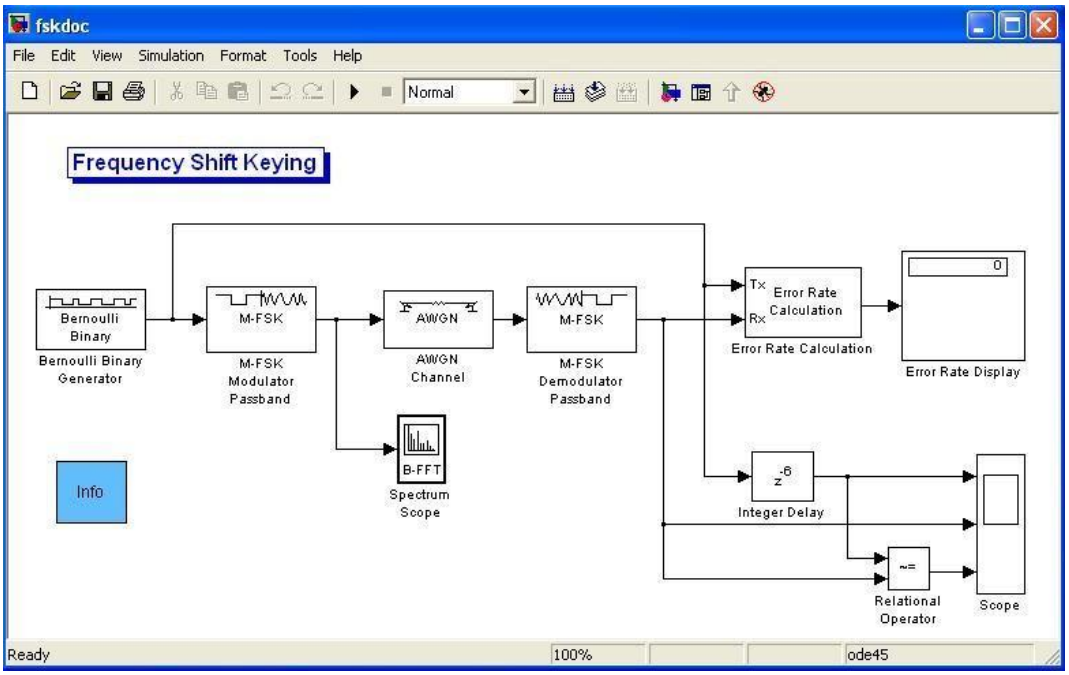
All the power of MATLAB® is available to you when you use the Communications Blockset. You can run simulations from the command line and invoke MATLAB expressions and M-files.

the basic elements that make up Simulink models – and libraries.This parts covers the following topics:

- •What You Need to Use the Models in This Book
- •Starting Simulink
- •Running a Model
- •Changing a Block's Parameters
- •Changing Simulation Parameters
- •Libraries

Starting Simulink

Unlike MATLAB, there is no special command window for Simulink. It works in the background whenever you build and run models. Before using Simulink, you must first start MATLAB. You can then load an existing Simulink model by typing its name at the MATLAB prompt.
For example, type " *fskdoc*" to bring up the model shown below.



Notice that the model *fskdoc* looks like a standard block diagram. The blocks represent various processes in the model.
For example, "Bernoulli Binary Generator" block at the upper left is a source that generates a Bernoulli random binary number. The Results block is an error rate display, which displays the bit error rate. The lines between blocks represent the passage of data through the model. You canfind the blocks in this model in Simulink libraries.

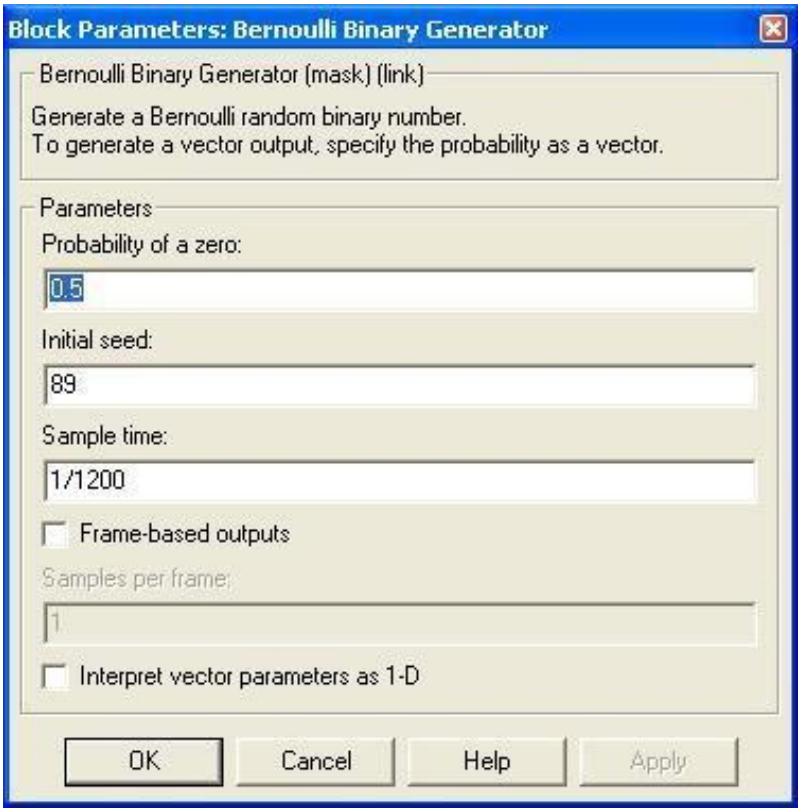
Running a Model

You can run a model by selecting the Simulation menu at the top of the model window, and then selecting Start or simply by clicking on the symbol at the top of the of the model window. The square symbol next to it stops the model.

Start the model *fskdoc* and observe the resulting burst of activity. Even if you don't understand exactly what is going on in the model, you can readily see the exciting dynamics of what Simulink can do for you. It provides incisive, visually compelling simulations that give apanoramic overview of all areas of processing. Using Simulink is more like working with laboratory equipment than computing.

Changing a Block's Parameters

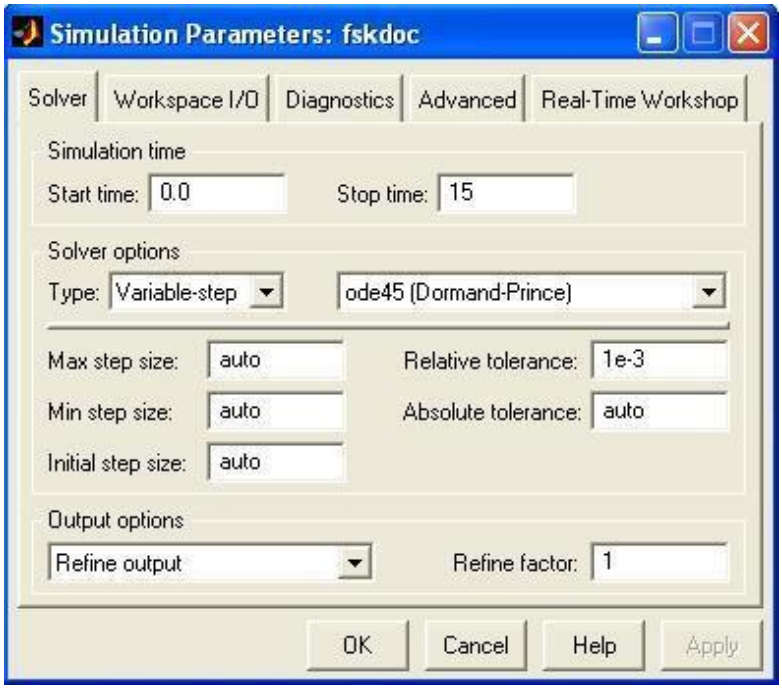
You can not change a block's parameters while a model is running. Stop the *fskdoc* model and double-click on the Bernoulli Binary Generator at the upper-left of the model window. A dialog box opens, presenting you with several parameters that control the block's operation.



Click the Initial seed parameter value, and overwrite the unity value by typing 23. Clicking the Apply button at the lower right corner of the dialog box changes the amplitude. Notice that the scope traces go off the scale.
Now, restore the Amplitude value to 89 and click OK. Then stop the demo.

Changing Simulation Parameters

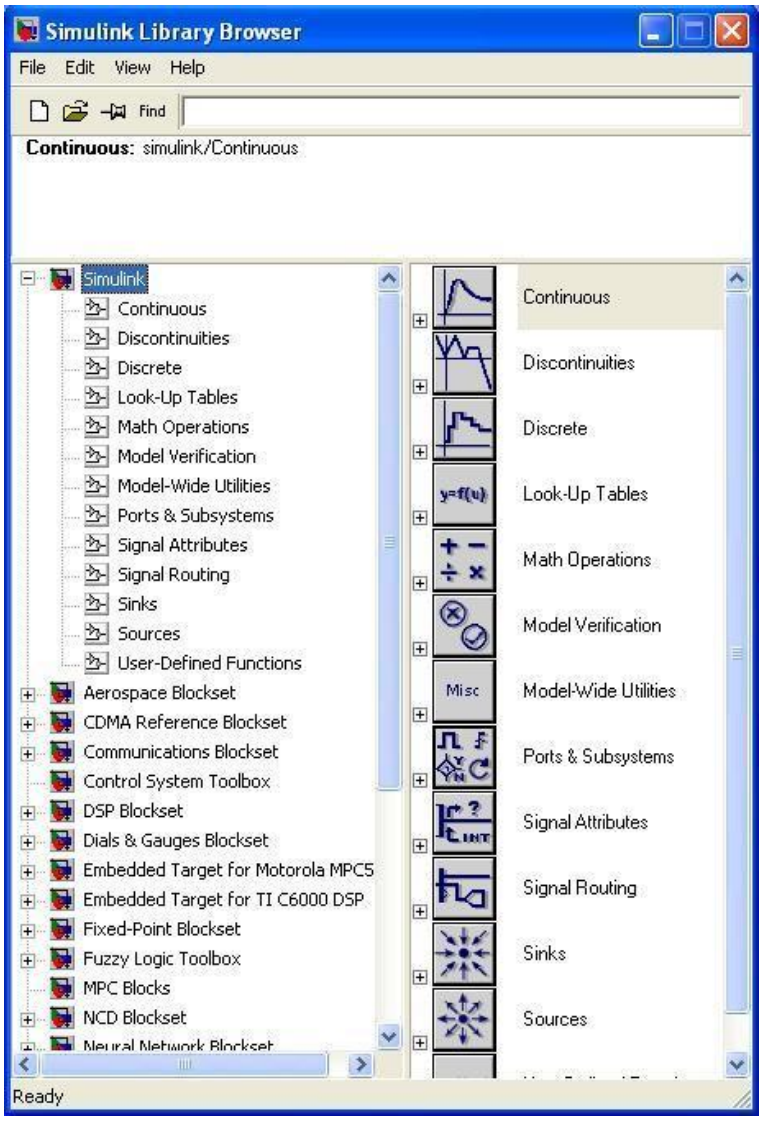
There are several parameters that control the overall simulation. You can view or change these by selecting the Simulation menu at the top of the model window, and then selecting Parameters,which opens a dialog window as in the figure below.



Change the Stop time value to 5 and select OK. Now start the model again. The simulation will stop after about five second. If you change the Stop time setting to inf (for infinite), the simulation will never terminate until you stop it manually.
Now, close the *fskdoc* model, selecting No when asked if you want to save the changes.

Libraries

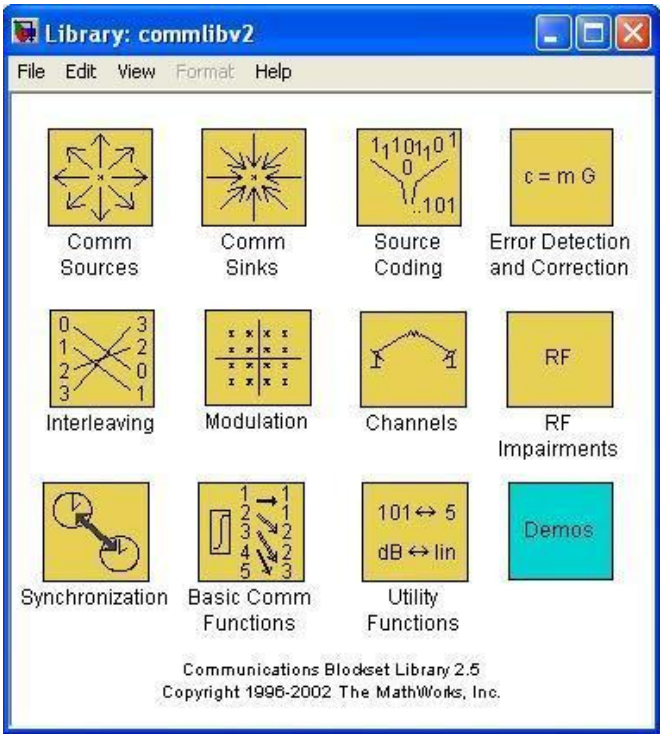
Simulink libraries are collections of Simulink blocks, which you can use to create models. To see all the available libraries, select the Simulink icon at the top of the MATLAB desktop, or type *simulink* at the MATLAB prompt. This opens the Simulink Library Browser, which lists thelibraries and their sublibraries in a tree structure, as shown below. The most important libraries for this manual are the Simulink library, the DSP blockset and the Communication Blockset library.



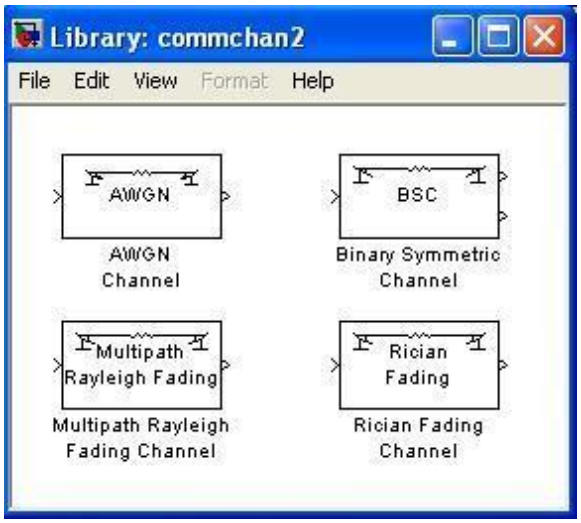
You can open a library to see its sublibraries by clicking on the + sign next to its name. For example, clicking on the + sign next to communication Blockset opens several sublibraries. You can open a sublibrary, such as "Comm Sinks", in the same way. Furthest to the right on the tree are the individual blocks.

Clicking on the name of a library or block will display a description of it at the bottom of the Browser window.

To open a library in a window, type the name of the library at the MATLAB prompt. For example, type *commlib* to open the communication Blockset library, as seen in the top windowin the figure below.

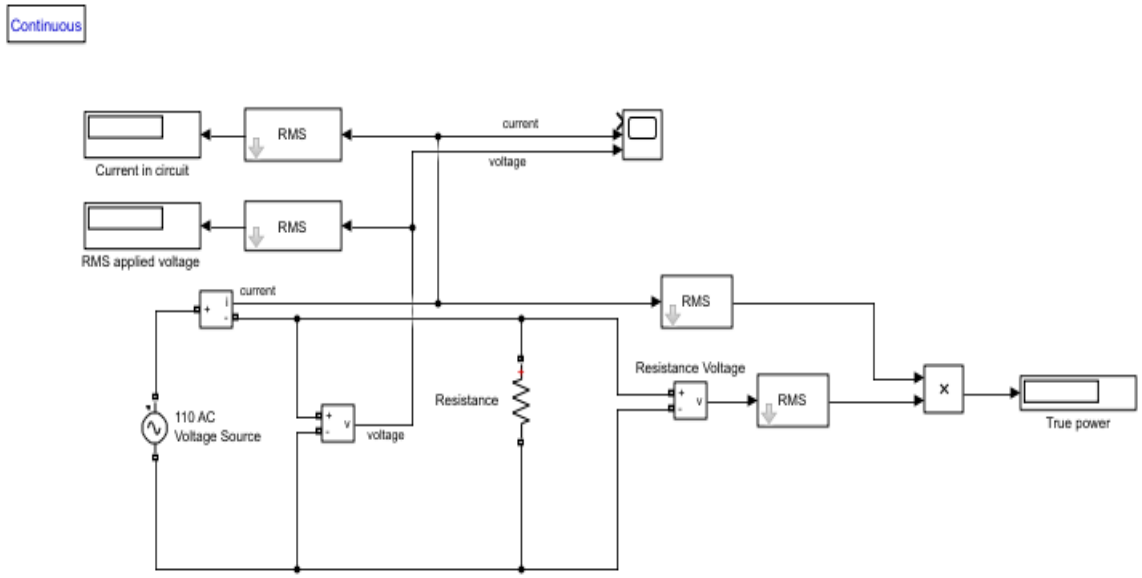


Each of the icons in this window represents a sublibrary. Click on one of the icons, such as theone labeled “Channels” and that sublibrary will appear, as seen in figure below.

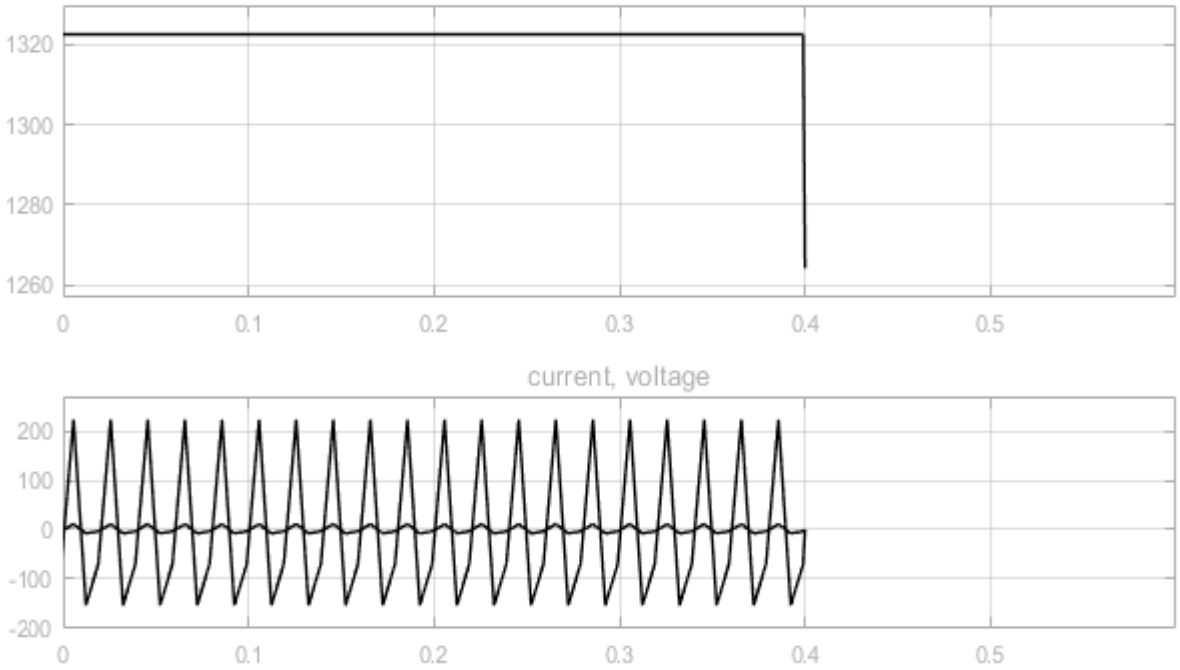


Determination of R, L & C Responses

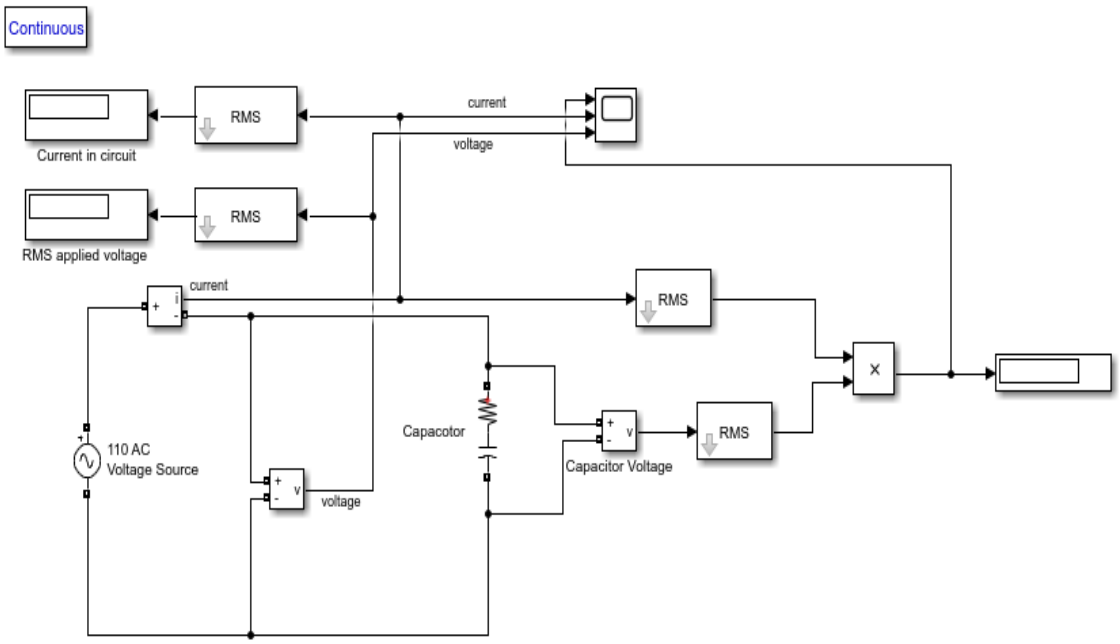
Ex.1 A circuit consists of a resistance of 20 ohm. A supply of 230V at 50 hz is applied across the circuit. Find the power factor.



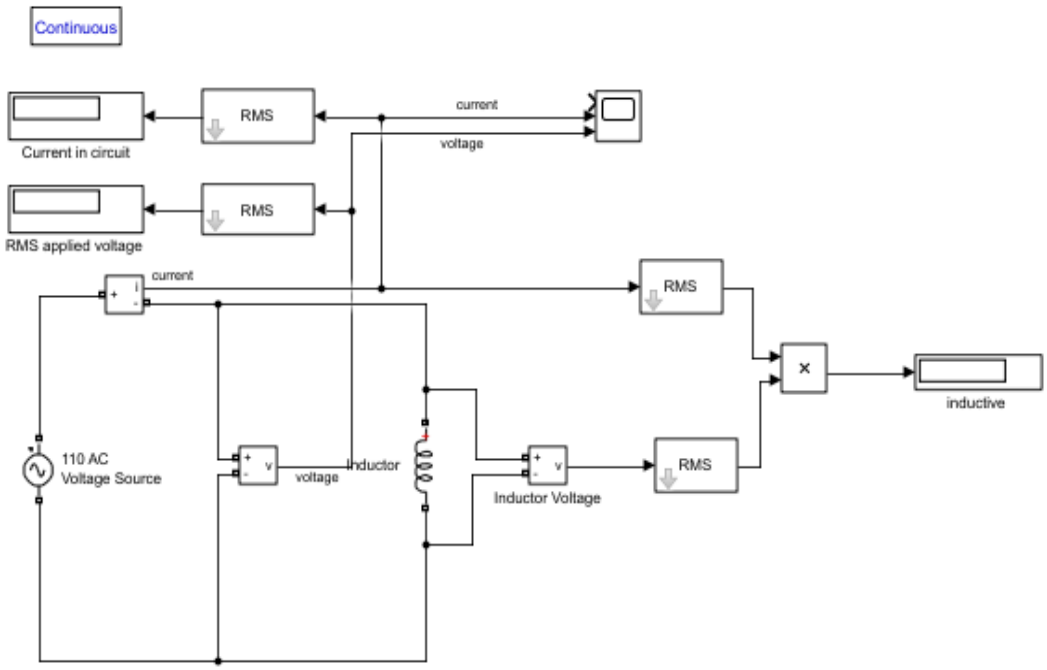
Output



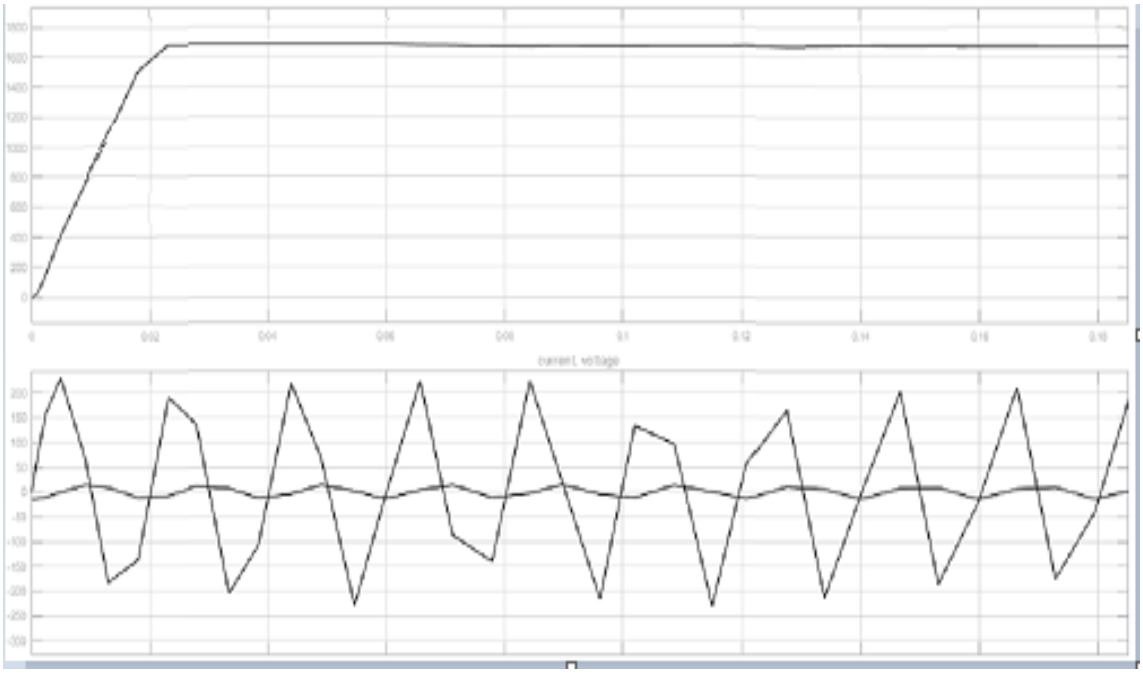
Ex.2 A circuit consists of a capacitance of 100uF with internal resistance is 0.01 ohm. A supply of 230V at 50 hz is applied across the circuit. Find the power factor.



Ex3 A circuit consists of a inductance of 0.05H. A supply of 230V at 50 hz is applied across the circuit. Find the power factor.

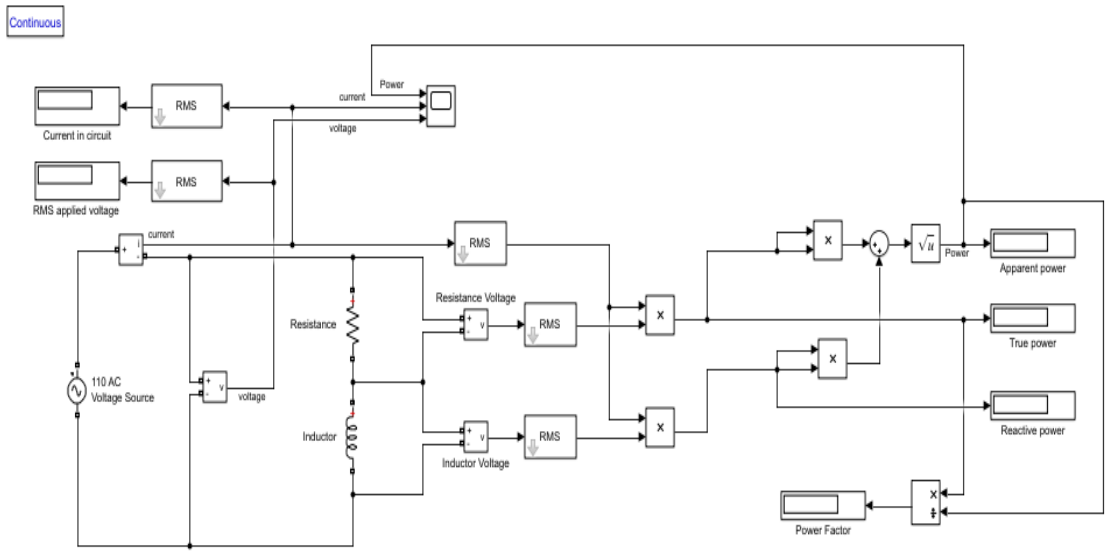


Output

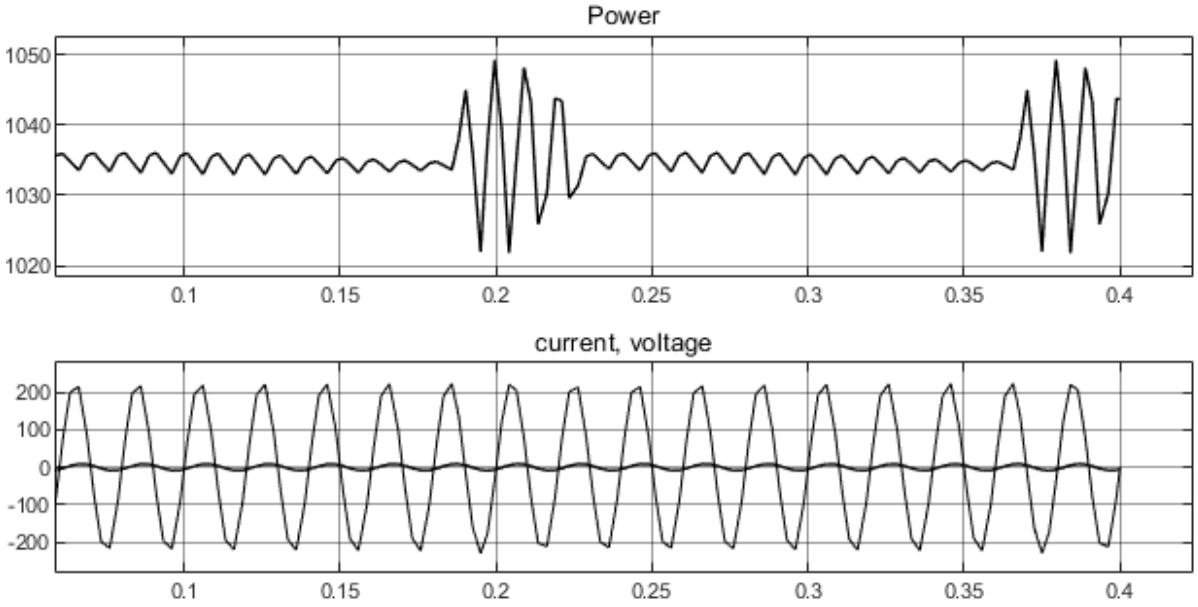


Design of RC, RL, RLC Circuits

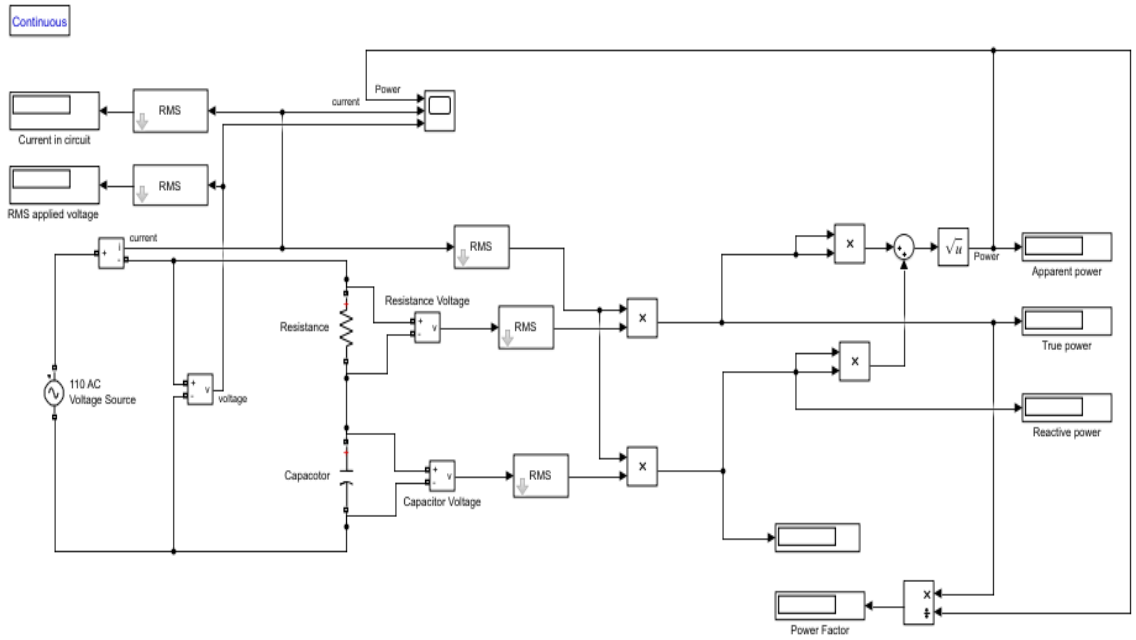
Ex.1. A circuit of a resistance 20 ohm**,** an inductance of 0.05H connected in series. A supply of 230 V at 50 Hz is applied across the circuit. Find the current, power factor and power consumed by the circuit.



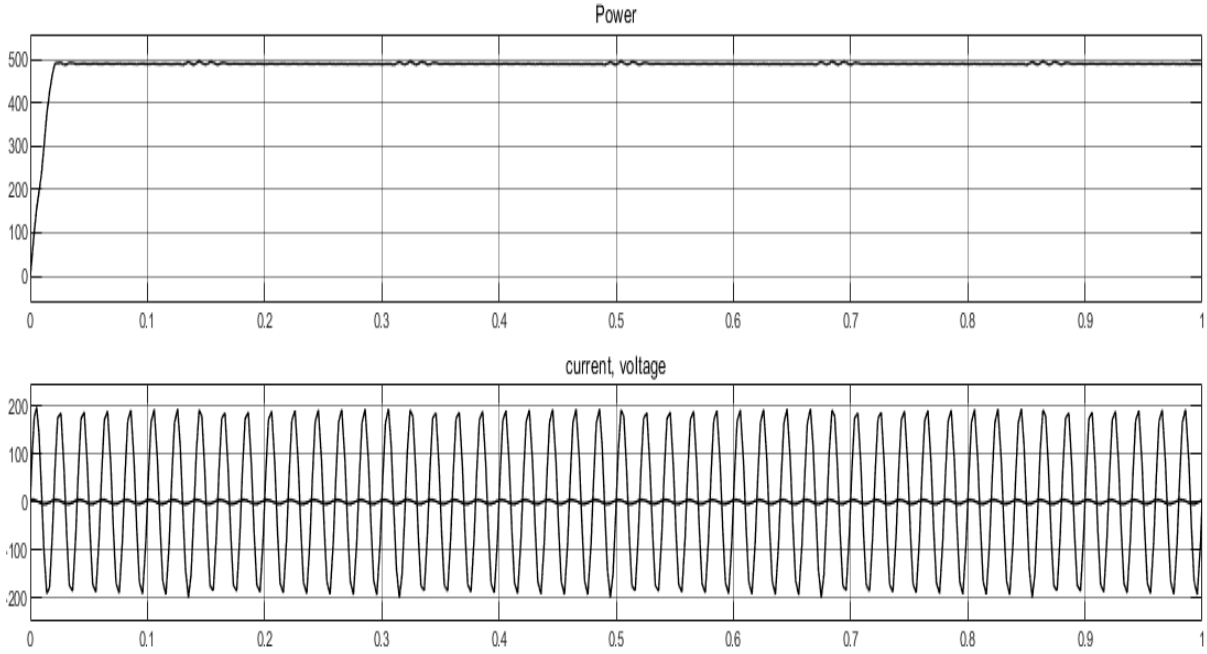
Output



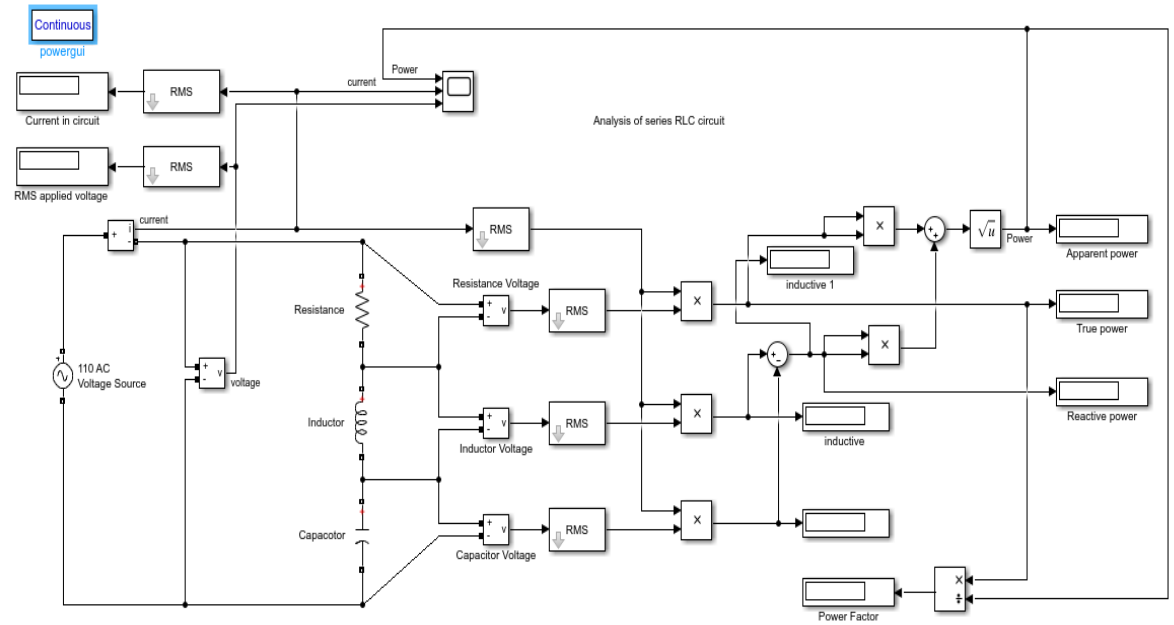
Ex.2. A circuit of a resistance 20 ohm**,** a capacitance of 100uF connected in series. A supply of 230 V at 50 Hz is applied across the circuit. Find the current, power factor and power consumed by the circuit.



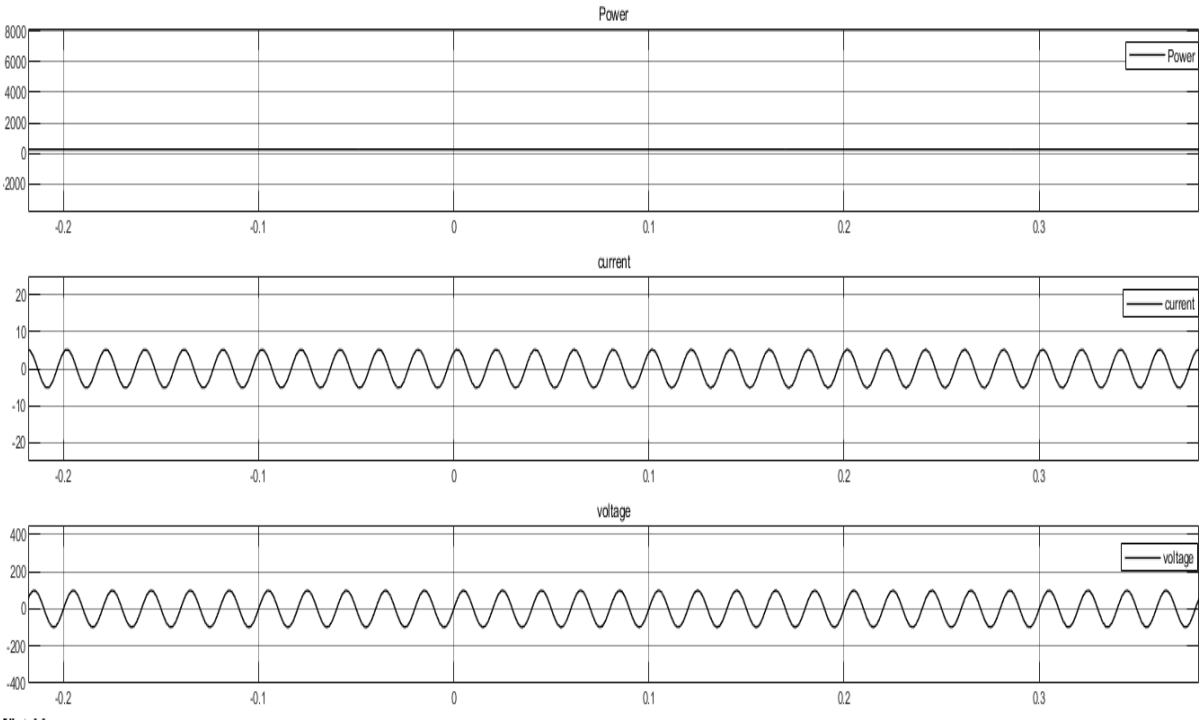
Output



Ex.3. A circuit of a resistance 10 ohm**,** an inductance of 16mH and capacitances of 150uF connected in series. A supply of 230 V at 50 Hz is applied across the circuit. Find the current, power factor and power consumed by the circuit.

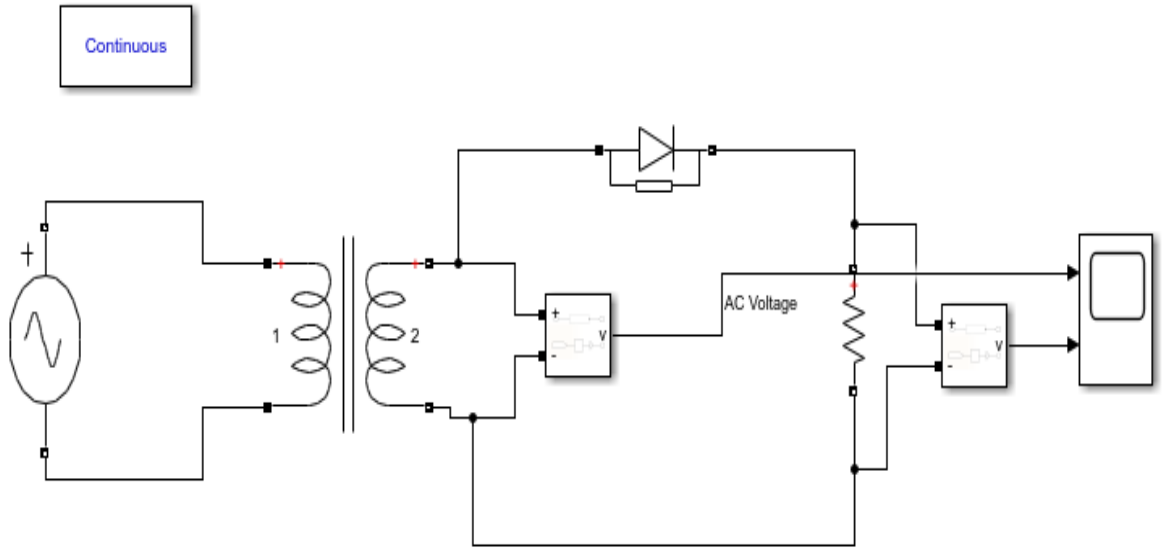


Output

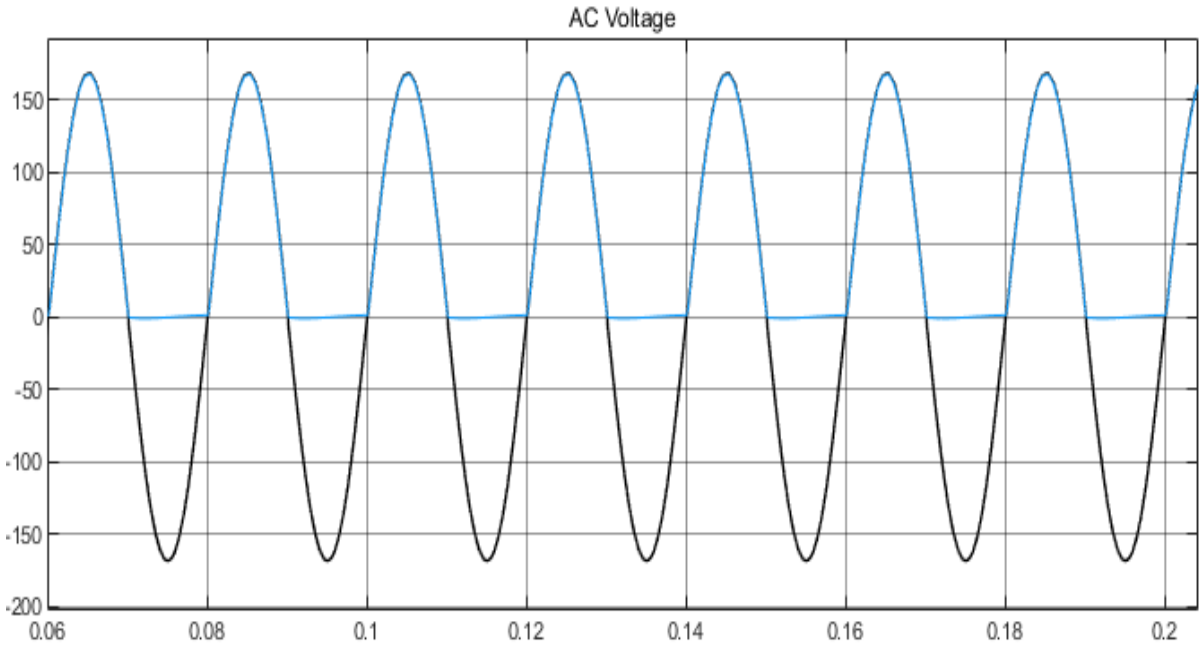


Design of Half wave and Full wave Rectifier Circuits

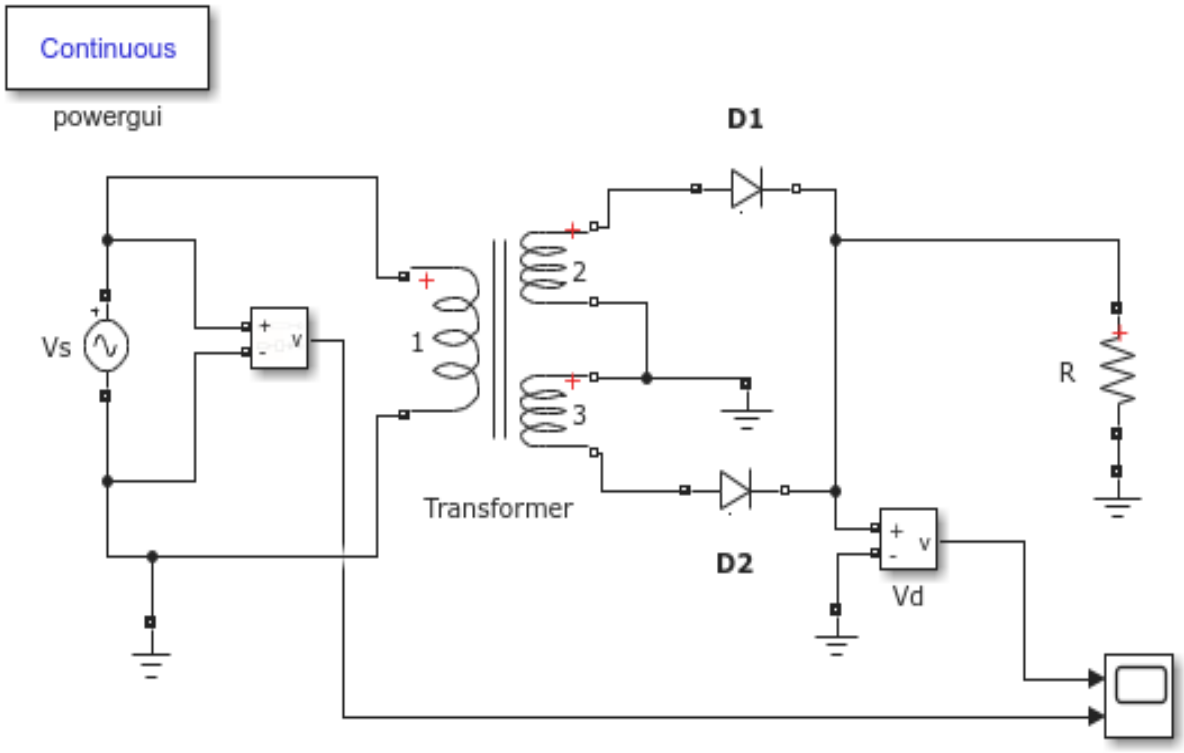
Ex.1 Design a half wave rectifier for given values V =110V, 10:1 turns ratio and Rl=500ohm



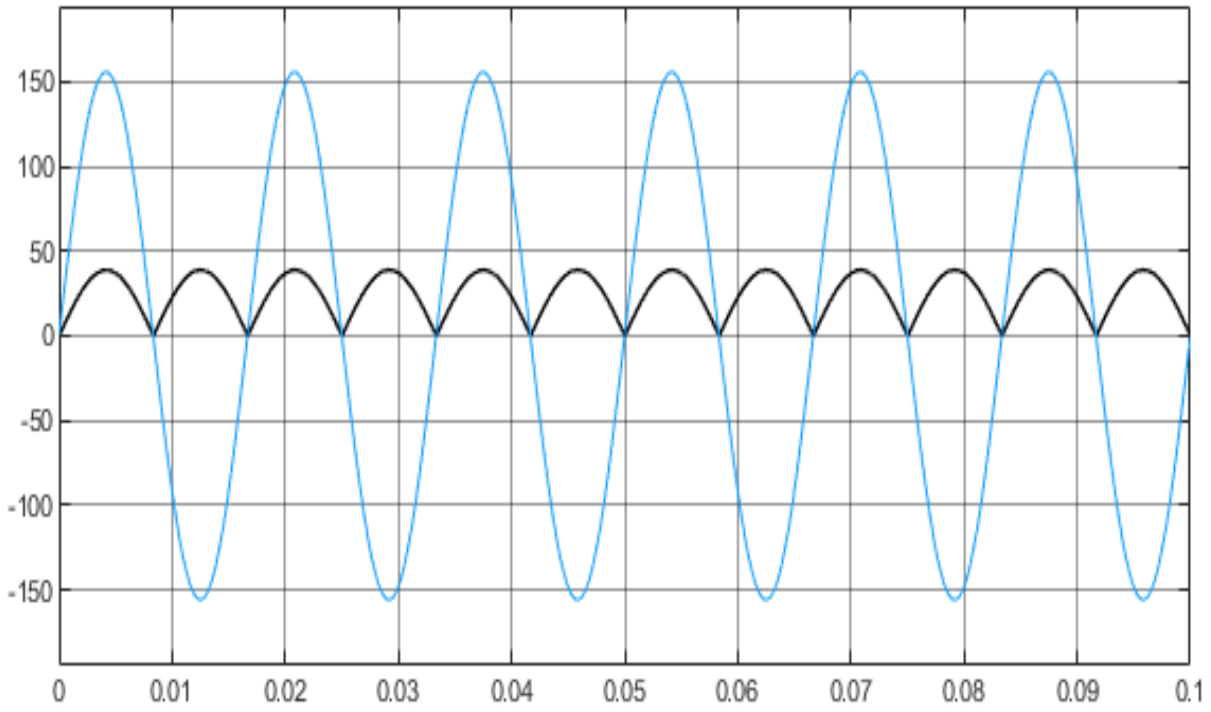
Output



Ex.2 Design a full wave rectifier for given values V =110V, 10:1 turns ratio and Rl=500ohm

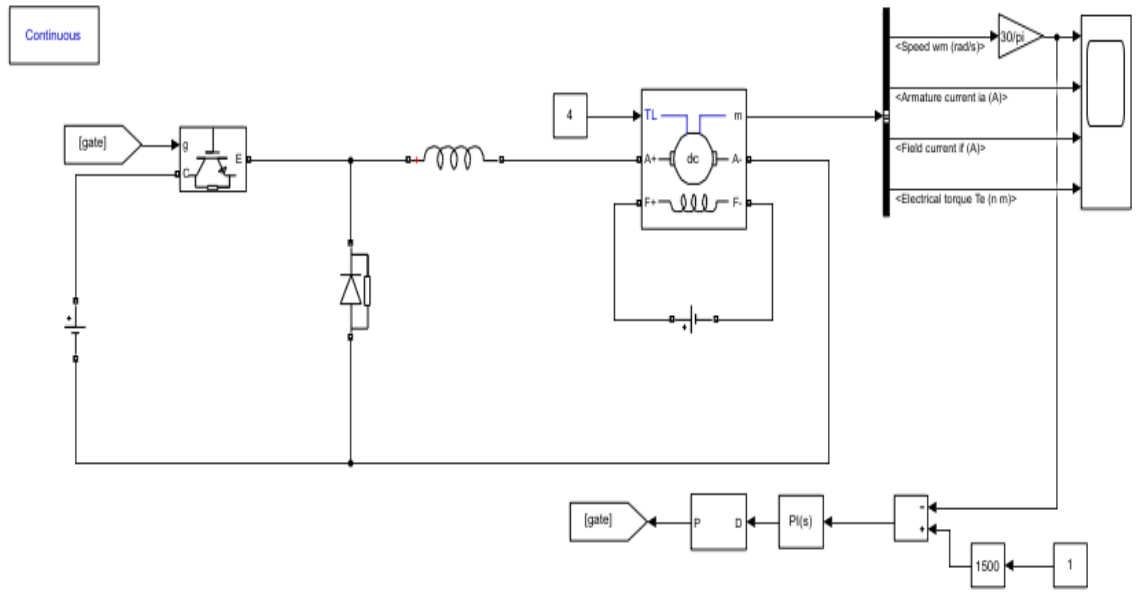


Output



Simulating Motor control Techniques

Ex 1. Speed control of DC Motor using Chopper



Output