Ministry of Higher Education and Scientific Research

Mohamed KHIDER University of Biskra

Faculty of Science and Technology

Department of Electrical Engineering

Course material

Simulation software

(Logiciels de simulation)

For: 3rd year Electrotechnics Degree

Prepared by:

Dr. SAADI Ramzi

Table of Contents

Chapter 1
Getting started with MATLAB

1.1 Introduction

Created by MathWorks, MATLAB is an interactive system for scientific programming, for numerical computation and graphic visualization, based on the matrix representation of data, whose name is derived from Matrix Laboratory. It is a multi-platform tool that is available for Windows environments, Unix (and other).

MATLAB provides a slew of built-in commands and mathematical functions to help with mathematical modeling and data analysis. Its diverse applications include machine learning, signal and image processing, computer vision, and communications, as well as computational finance, control design, and robotics.

Users of MATLAB can create graphical user interfaces using its own functions in addition to these. Simulink, a tool for model-based design and simulation of systems spanning several dynamic and embedded domains, is also included.

1.2 MATLAB Environment

When you start MATLAB, the desktop environment is displayed with a default interface layout.

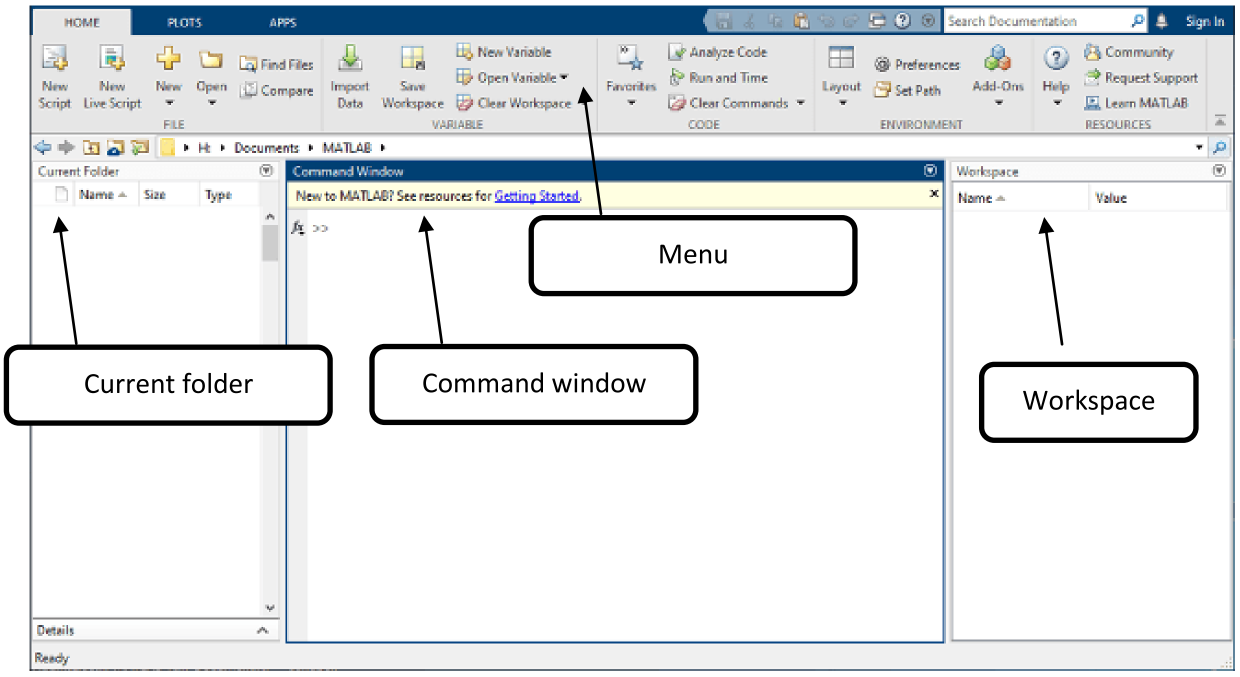


Figure 1-1 MATLAB Environment version 2018

The desktop environment provides the following panels:

The menu groups together basic Matlab commands such as save, display, etc ......

Current Folder – Access your files.

Command Window – Enter command line commands, indicated by the prompt (>>).

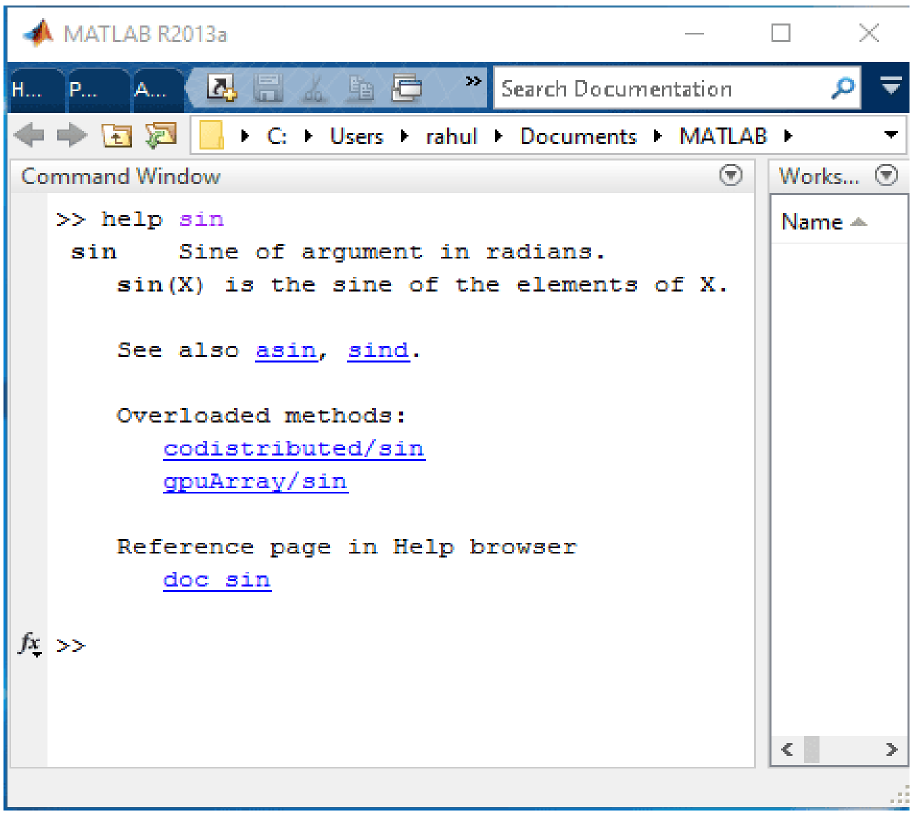
Workspace – Explore data you create or import from files.

1.3 Presentation and generalities

1.3.1 Getting Help

The Matlab help is well adapted to self-learning due to the various functions and methods available (particularly thanks to its toolboxes). Typing 'help' displays all of Matlab's contents, i.e. the families of functions it includes. Typing 'help family' reveals all of the functions in the selected family, whereas 'help function' displays the function's definition, options, and syntax.

The command 'help' lists all primary help topics in the Command Window. The format 'help name' displays the help text for the functionality specified by name, such as a function, method, class, or variable.



```
>> help sin
 sin    Sine of argument in radians.
    sin(X) is the sine of the elements of X.

    See also asin, sind.

    Overloaded methods:
       codistributed/sin
       gpuArray/sin

    Reference page in Help browser
       doc sin

>>
```

Figure 1-2 Help window

1.3.2 First interaction with MATLAB

To start Matlab, double-click on the Matlab icon. A window opens in which you then receive the "prompt" >>. Every command given to Matlab is interpreted, the result is displayed, and the user receives the prompt >> again for the next command: this is interactive mode.

You can execute commands here. Try typing disp ('Hello, World!') and press Enter. This command will display the text "Hello, World!" in the Command Window.

A line of instructions in Matlab is often constructed as follows:

```
>> [output_variables] = function_name(input_variables);
```

Here is a breakdown:

1. **output_variables:** These are the variables where you will store the output from the function. If the function outputs more than one value, the output variables should be enclosed in square brackets.
   For example: [output1, output2] = function_name(input_variables);. If the function only outputs one value, you don't need the square brackets. For example: output = function_name(input_variables);.
2. **function_name:** This is the name of the function you want to call. Matlab has a vast library of built-in functions, but you can also create your own custom functions.
3. **input_variables:** These are the variables that you pass into the function as input. Depending on the function, you may need to pass one or more inputs, and they should be separated by commas. For example: output = function_name(input1, input2);.
4. **; (semicolon):** At the end of the line, you use a semicolon. The semicolon suppresses the output from being displayed in the command window. If you want to see the output, you can omit the semicolon.

To calculate a mathematical expression, simply write it as follows:

```
>> 5 + 2
ans=
     7
```

To create a variable, use the simple structure: 'variable = definition', without worrying about the type of the variable.

Here is an example of simple arithmetic operations in Matlab with variable :

```
>> a = 5;
>>b = 10;
>>c = a + b; % This will add a and b and store the result in c.
```

The same example without ; (semicolon):

```
>> a = 5
a=
    5
>>b = 10
a=
    10

>>c = a + b % This will add a and b and store the result in c.
c =
    15
```

In defining variable names in Matlab, only alphanumeric characters and the underscore symbol '_' should be used, and the name must start with a letter. It's important to note that Matlab is case-sensitive, meaning it treats 'A' in uppercase and 'a' in lowercase letters as separate identifiers.

1.3.3 The main opérations, constants, functions, and commands:

Opérations

here are some basic operations you can perform in MATLAB:

1. **Arithmetic Operations:**
   - Addition: **a + b**
   - Subtraction: **a - b**
   - Multiplication: **a * b**
   - Division: **a / b**
   - Exponentiation: **a^b**
   - Modulus: **mod(a, b)**
2. **Matrix Operations:**
   - Transpose: **A'**
   - Matrix multiplication: **A * B**
   - Element-wise multiplication: **A .* B**
   - Inverse: **inv(A)**
   - Determinant: **det(A)**
3. **Relational Operations:**
   - Equality: **a == b**
   - Inequality: **a ~= b**

- Less than: **a < b**
- Greater than: **a > b**
- Less than or equal to: **a <= b**
- Greater than or equal to: **a >= b**

4. **Logical Operations:**
    - AND: **a & b**
    - OR: **a | b**
    - NOT: **~a**
5. **Trigonometric Operations:**
    - Sine: **sin(a)**
    - Cosine: **cos(a)**
    - Tangent: **tan(a)**

Remember, MATLAB uses a vectorized system, so many operations can be performed on arrays or matrices directly.

Constants

Some of the most common constants or special values that you might use in MATLAB are:

- **pi:** The mathematical constant (pi) that represents the circumference to diameter ratio of a circle. You may simply type pi in MATLAB.
- **Inf:** This stands for positive infinity. You can make it in MATLAB using Inf.
- **-Inf:** This stands for negative infinity. You can make it in MATLAB by using the -Inf command.
- **NaN:** This stands for 'Not a Number' and is used to represent undefined or unrepresentable values such as the result of 0/0. You may make it in MATLAB using NaN.
- **i or j:** The imaginary unit is represented by both i and j. In MATLAB, you can generate imaginary numbers such as 1 + 2i or 1 + 2j.
- **eps:** This is the smallest difference between two distinct real numbers that the computer can represent. In MATLAB, you can get this value with **eps**.
- **realmax:** This is the largest positive floating-point number that the computer can represent. In MATLAB, you can get this value with **realmax**.
- **realmin:** This is the smallest positive floating-point number that the computer can represent. In MATLAB, you can get this value with **realmin**.

Functions

While working in MATLAB, you issue commands that create variables and call functions. For an introduction, see Enter Statements in Command Window.

- **sin, cos, tan** - Trigonometric functions.
- **asin, acos, atan -** Inverse trigonometric functions.
- **exp -** Exponential function.
- **log, log10, log2 -** Logarithmic functions.
- **sqrt -** Square root function.
- **abs -** Absolute value function.
- **round,** ceil, floor - Rounding functions.
- **max, min -** Functions to find maximum and minimum values.
- sum, prod - Functions for summation and product.
- **mean, median, std, var -** Statistical functions.

Commands

Some of the most common commands that you might use in MATLAB are:

ans	Most recent answer
clc	Clear Command Window
clear	`clear` - Removes variables from memory.
diary	Log Command Window text to file
format	Set output display format
whos	Lists current variables.
home	Send cursor home
load/save	Loads/saves workspace variables from/to .mat files.
iskeyword	Determine whether input is MATLAB keyword
more	Control paged output in Command Window
commandwindow	Select the Command Window
commandhistory	Open Command History window

Objects

DisplayFormatOptions	Output display format in Command Window

These are just some of the basic and common constants, functions, and commands in MATLAB. There are many more that serve different purposes depending on the specific needs of your computation tasks.

1.3.4 Example

Here are some sample examples relating to the previously defined elementary functions.

```
a = 2;
b = 3;
c = a^b; % This will raise a to the power b and store the result in c.

a =

     2


b =

     3


c =

     8

a = pi/4
val_sin = sin(a)  % Sine of a
val_cos = cos(a)  % Cosine of a
val_tan = tan(a)  % Tangent of a

a =

    0.7854


val_sin =

    0.7071


val_cos =

    0.7071



val_tan =

    1.0000

a = 2;
val_exp = exp(a)  % e raised to the power a
val_log = log(a)  % Natural Logarithm (base e) of a
```

```
val_log10 = log10(a)  % Logarithm (base 10) of a
val_exp =

    7.3891


val_log =

    0.6931


val_log10 =

    0.3010
```

To calculate y = e $^{-a}$ cos(x) + 4√y, for a = 2, x = 4, and y = 6, we enter the following commands in MATLAB,

```
>> a = 2; x = 4; y = 6;
>> y = exp(-a)*cos(x)+4*sqrt(y)
y =

    9.7095
```

Chapter 2
Data Types and Variables

Similar to all programming languages, MATLAB provides the functionality to define variable data. A variable is indicated by an identifier composed of a blend of letters and numbers. In MATLAB, there is no need to declare the type or size of a variable. The type and size are automatically discerned based on the mathematical expression or the value allocated to the variable.

2.1 Data types

In MATLAB, variables can hold diverse types of data, and they are dynamically typed, which implies that there's no requirement to predefine their type or dimension. Here are several standard types of data that can be managed in MATLAB:

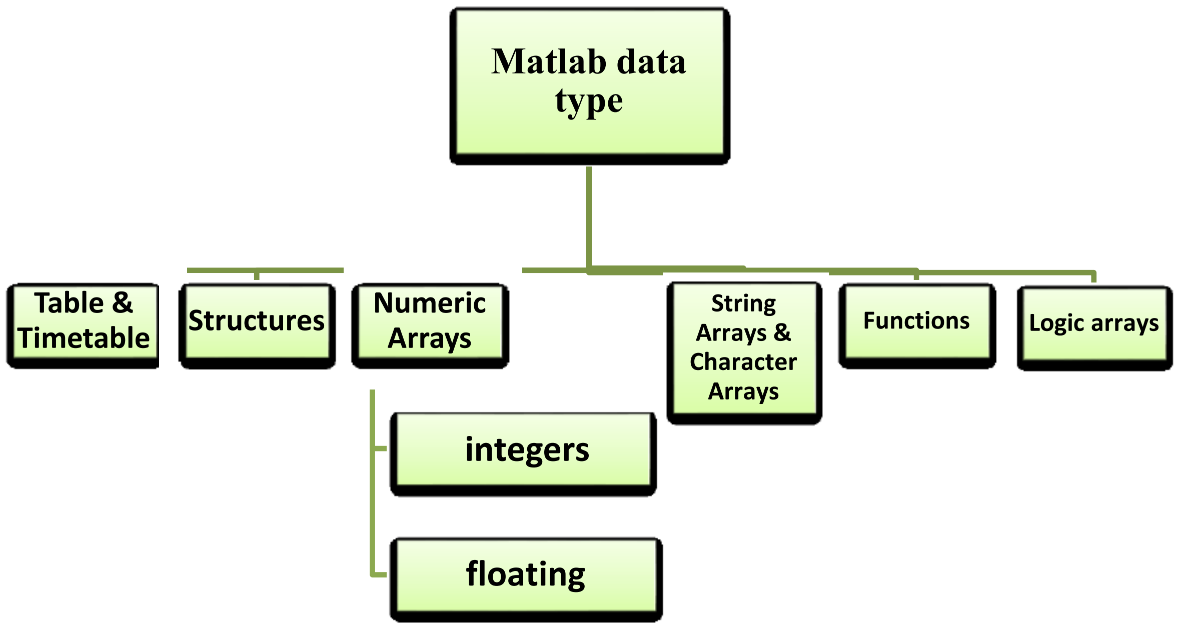


Figure 0-1 MATLAB Data types

- **Numeric Types:** Numeric data is the most frequently encountered data type in MATLAB. By default, MATLAB stores all numeric values as double-precision floating-point numbers. The primary numeric types, **double** and **single**, are used to store double-precision and single-precision numbers, respectively.
- **Integer Types:** MATLAB allows for data storage as integers of various sizes. These types include **int8**, **int16**, **int32**, **int64** for signed integers, and **uint8**, **uint16**, **uint32**, **uint64** for unsigned (non-negative) integers.

- **Logical Type:** The **logical** data type is used for storing boolean values, which can either be **true** (1) or **false** (0).
- **Character and String Types:** Character arrays or **char** are sequences of characters, and **string** arrays are a newer addition to MATLAB (introduced in R2016b) specifically tailored to handle text.
- **Cell Arrays:** These are arrays where each element can hold any type of data. Cell arrays can be created using the **cell** function or by using curly braces {}.
- **Structures:** Structures are unique data types that bundle related data of varying types and sizes using containers known as fields. Structures can be created using the **struct** function.
- **Tables:** Table data types hold heterogeneous data in a tabular format, where each column can represent a different data type.
- **Functions:** Function handles are a type of variable that can be passed to other functions. These handles can reference both built-in and user-defined functions.

2.2 Variables types

2.2.1 Complex numbers in Matlab

In MATLAB, complex numbers are numbers with a real and an imaginary part. They are represented as "a + bi", where "a" is the real part and "b" is the imaginary part. a + bi and rexp(it) or rexp(it) with a, b, r, and t as real type variables are other forms to represent complex numbers in Matlab. In this context, it means that the complex number can be represented in either Cartesian form as "a + bi" or in Polar form as "rexp(it)", where "a", "b", "r", and "t" are variables of real number type.
Matlab contains several predefined functions for manipulating complex numbers.
We can write:

```
z = 4+i*8
z = 4+8i
z =4*exp(i*8)
```

```
z =

   4.0000 + 8.0000i


z =

   4.0000 + 8.0000i
```

```
z =

  -0.5820 + 3.9574i
```

Or

```
a = 3;  % real part
b = 4;  % imaginary part
z = complex(a, b) % Cartesian form
```

```
z =

   3.0000 + 4.0000i
```

The abs() function gives the magnitude (or modulus) of the complex number, and the angle() function provides the phase (or argument). The phase is in radians. For example:

```
m = abs(z);  % returns the magnitude of z
p = angle(z);  % returns the phase of z
```

To convert Cartesian form (a + bi) to Polar form (rexp(it)):

```
a = 3;  % real part
b = 4;  % imaginary part
z = complex(a, b)  % Cartesian form
r = abs(z);  % magnitude
t = angle(z);  % angle in radians
z_polar = r * exp(1i * t)  % Polar form
```

```
z =

   3.0000 + 4.0000i


z_polar =

   3.0000 + 4.0000i
```

MATLAB additionally contains multiple commands for dealing with complex numbers like :

abs	Returns the magnitude of z.
angle	Returns the phase angle of z in the interval [-π, π] in radians.
complex	Create complex array
conj	Returns the complex conjugate of z.
cplxpair	Sort complex numbers into complex conjugate pairs
i	Imaginary unit

imag	Extracts the imaginary part of the complex number z.
isreal	Returns 1 (true) if z has no imaginary part, and 0 (false) otherwise.
j	Imaginary unit
real	Extracts the real part of the complex number z.
sign	Sign function (signum function)
unwrap	Unwraps radian phases by changing absolute jumps greater than π to their 2π complement.

2.2.2 Boolean variables in MATLAB

In MATLAB Boolean data is represented by the logical data type . This data type uses the numbers 1 and 0, accordingly, to represent the true and false states. Some MATLAB operations and functions return logical values to signify the occurrence of a condition. These logical values can be used as indexes for arrays or to run conditional code.

Here an exemple of Boolean function in matlab

```
a = true
b = false
f = a && b % Logical AND
g = a || b  % Logical OR
h = ~a      % Logical NOT
```

```
a =

  logical

   1

b =

  logical

   0

f =

  logical

   0
g =

  logical

   1

h =     0
```

the most used commands for dealing with boolean numbers are :

&	Find logical AND
Short-Circuit &&	Logical AND with short-circuiting
~	Find logical NOT
|	Find logical OR
Short-Circuit ||	Logical OR with short-circuiting
xor	Find logical exclusive-OR
all	Determine if all array elements are nonzero or true
any	Determine if any array elements are nonzero
false	Logical 0 (false)
find	Find indices and values of nonzero elements
islogical	Determine if input is logical array
logical	Convert numeric values to logicals
true	Logical 1 (true)

2.2.3 String of characters

A string of characters is an array of characters .If you want to enter a string of characters, you need to type it in quotes

```
>>S = "Your string here";
```

Here is an example:

```
>>S = "Hello, Welcom";
```

In this case, S is a string containing the text "Hello, Welcom".

A variable of string type being interpreted as an array of characters, it is possible to manipulate each letter of the string by referring to its position in the string

```
>>S1='nice'
S2='day'
S = [S1,S2]
S(1), S(4), S(1:5)
```

```
S1 =
    'nice'

S2 =

    'day'
S =

    'niceday'

ans =

    'n'
ans =

    'e'

ans =

    'niced'
```

If a string needs to include an apostrophe character ('), this character has to be duplicated within the string (for instance, to assign an apostrophe character to a variable, we'll need to write ''''

This statement refers to the common programming practice of escaping certain characters (like the apostrophe) that have special meanings in code. By doubling the apostrophe, the Matlab understands that you want to include an actual apostrophe in the string, and not use it to indicate the end of the string.

```
>>s = 'It's a nice day today'

 s = 'It's a nice day today'
         ↑
Invalid expression. Check for missing multiplication operator, missing or
unbalanced delimiters, or other syntax error. To construct
matrices, use brackets instead of parentheses.

>> S = 'It''s a nice day today'

s =
    'It's a nice day today'
```

To create a string with an apostrophe, you can use double quotes:

```
>>S = "It's a nice day today.";  % This creates a string
```

2.2.4 The Arrays

An array is a fundamental data structure, which can store a fixed-size collection of elements of the same data type. If the elements are arranged horizontally, we say that the vector is a row array. On the other hand, if the elements are arranged vertically, we say it's a column array

One can generate an array in MATLAB by surrounding a set of elements in square brackets []. Following are a few illustrations of arrays of various types:

Row Vector (1D array) A row vector represents a one-dimensional array with horizontally aligned entries. Commas or spaces are utilized to differentiate between elements.

```
>>RV = [1 2 3 4 5]; % This function generates a row vector with entries. 1, 2, 3, 4, and 5
RV =

     1     2     3     4     5
Or
```

```
>>RV = [1, 2, 3, 4, 5]
```

Column Vector (1D array): A column vector is a one-dimensional array with vertically organized items. The semicolon characters are used to differentiate between items**.**

```
>>CV= [1; 2; 3; 4; 5]; % This function generates a coumn vector with entries 1, 2, 3, 4, and 5
CV =

     1
     2
     3
     4
     5
```

Or

```
CV= [1
 2
 3
 4
 5]
```

We can transpose the row array to a column array by

```
RV = [1, 2, 3, 4, 5]'

RV =

     1
     2
     3
     4
     5
```

To create a vector with consecutive numbers, you can use the colon (:) operator. The syntax is first element : last element .

```
>>A=[1:10]

A =

     1     2     3     4     5     6     7     8     9    10
```

One can create an array with a defined step using the colon (:) operator. The standard syntax is first element : step : last element .

```
>>A=[1:2:10]

A =

     1     3     5     7     9


>>A=[1:0.5:10]

A =



    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000    4.0000    4.5000
5.0000    5.5000    6.0000    6.5000    7.0000    7.5000    8.0000    8.5000
9.0000    9.5000   10.0000
```

To access elements of an array we can use indexing. The index of the first element of an array in MATLAB is 1.

Array name (position) for this instruction we use the parenthesis () because the square brackets used only for array creation.

```
>>A = [5, 8, 3, 9, 5] % Array

 A(1)  % Accesses the first element of the array, 1
 A(3) % Accesses the third element of the array, 3

A =

     5     8     3     9     5


ans =

     5


ans =

     3

>>A(3:5) % from the tird element of the fifth elemnt of the array
```

```
ans =

     3     9     5

>> A(2:end)% from the second element of the last elemnt of the array

ans =

     8     3     9     5


>>A([2,4,1]) % only the second and the forth and the first elemnt

ans =

     8     9     5

>>A(2)=0 % Give the element 2 the number 0

A =

     5     0     3     9     5


>>A(6)=17 % add an element 6 and give it the number 17

A =

     5     0     3     9     5    17



>>A(6)=[] % delete the element number 6

A =

     5     0     3     9     5

>>A(3:5)=[]% delete element number 2 to number 5

A =

     5     0
```

There are various operations that can be performed on arrays in MATLAB, including: Arithmetic operations: you can perform multiply, divide, add, and subtract arrays. Be aware that these actions are carried out element-by-element. Here are a few instances:

```
>>A = [2 4 6 8]
B = [1 2 3 4]


C = A + B % Addition between A and B
```

```
D = A - B % Subtraction between A and B
E = A .* B % Element by element multiplication between A and B
F = A ./ B % Element by element division between A and B

A =

     2     4     6     8


B =

     1     2     3     4


C =

     3     6     9    12


D =

     1     2     3     4


E =

     2     8    18    32


F =

     2     2     2     2
```

To create array with components arranged at regular intervals and with a well-determined number of elements, a Matlab function called linspace can be used.

The syntax is linspace (fisrt element, last element, n), where fisrt element is the starting value, last element is the ending value, and n is the number of points to generate between the start and end.

The **linspace(a, b, n)** function generates a vector of n elements ranging from a to b

Example: To create a vector with 6 uniformly spaced points between 1 and 12, use the following formula:

```
>>A = linspace(1, 12, 6)

A =

    1.0000    3.2000    5.4000    7.6000    9.8000   12.0000
```

When n is not specified, the value 100 is assumed. As a result, linspace(1, 10) produces an array of 100 linearly separated points that ranges from 1 to 10.
The **logspace(a, b, n)** function generates an array of n elements ranging from 10^a to 10^b

```
>>logspace(0, 4, 6)

ans =

   1.0e+04 *

    0.0001    0.0006    0.0040    0.0251    0.1585    1.0000
```

2.2.5 Matrix in MATLAB

In MATLAB, a matrix is a two-dimensional numeric array containing an assortment of numbers organized as rows and columns. There are several ways to construct a matrix with Matlab .
The first way is to use square brackets **[]**. Here is an example of how to create a 3x3 matrix:

```
>>A = [1 2 3; 4 5 6;7 8 9]  % This creates a 3x3 matrix

A =

     1     2     3
     4     5     6
     7     8     9
```

The second, and perhaps the simplest, is to declare the matrix as

```
>>A=[1 2 3
4 5 6
7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9
```

The third is to declare the matrix as

```
>>A = [1 ,2, 3; 4, 5, 6;7, 8, 9]  % This creates a 3x3 matrix

A =

     1     2     3
     4     5     6
     7     8     9
```

The number of elements in each row (number of columns) must be the same in all rows of the matrix, otherwise an error will be signaled by MATLAB

An element of a matrix is referenced by its row and column numbers. A(i,j) denotes the i-th element of the j-th row of the matrix A. Thus, A(3 ,2) refers to the first element of the second row of A.

```
>>A(3,2)

ans =

     8
```

A matrix can be generated from vectors as shown in the following examples:

```
>>B=1:0.5:2.5
C=1:4
D=[2 4 6 8]
A=[D;C;B]

B =

    1.0000    1.5000    2.0000    2.5000


C =

     1     2     3     4


D =

     2     4     6     8


A =

    2.0000    4.0000    6.0000    8.0000
    1.0000    2.0000    3.0000    4.0000
    1.0000    1.5000    2.0000    2.5000
```

some examples of how to extract a sub-matrix in MATLAB

Example No.	Original Matrix A	Command to Extract Sub-matrix	results
1	A = [1 2 3; 4 5 6; 7 8 9];	B= A(1:2, 1:2);	B = [1 2; 4 5];
2	A = [1 2 3 4; 5 6 7 8; 9 10 11 12];	B = A(1:2, 3:4);	B = [3 4; 7 8];
3	A = [1 2 3; 4 5 6; 7 8 9];	B = A(3, 1:2);	B = [7 8];
4	A = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15];	B= A(:, 4:5);	B = [4 5; 9 10; 14 15];

5	A = [1 2 3; 4 5 6; 7 8 9];	B = A(2,:);	B [4 5 6]

The special character (') is used to denote the transpose of a matrix. The command B= A' produces the transpose matrix.

```
>>B=A'

B =

    2.0000    1.0000    1.0000
    4.0000    2.0000    1.5000
    6.0000    3.0000    2.0000
    8.0000    4.0000    2.5000
```

2.2.5.1 Arithmetic operations

Addition +, subtraction -, multiplication *, powers ^ are used with the usual matrix syntax

Example No.	Matrix A	Matrix B	Operation	Command in MATLAB	Result
1	A = [1 2; 3 4];	B = [5 6; 7 8];	Addition	C = A + B;	C = [6 8; 10 12];
2	A = [1 2; 3 4];	B = [5 6; 7 8];	Subtraction	C = A - B;	C = [-4 -4; -4 -4];
3	A = [1 2; 3 4];	B = [5 6; 7 8];	Multiplication	C = A * B;	C = [19 22; 43 50];
4	A = [2 3; 4 5];		Power	C = A^2;	C = [16 21; 28 37];

2.2.5.2 tomatic generation of matrices in MATLAB,

Here are some examples of the most common instructions for automatic generation of matrices in MATLAB

No.	Command	Description	Example	Result
1	zeros(n, m)	Generates an n-by-m matrix filled with zeros.	A = zeros(2, 3);	A = [0 0 0; 0 0 0];

2	ones(n, m)	Generates an n-by-m matrix filled with ones.	A = ones(3, 2);	A = [1 1; 1 1; 1 1];
3	eye(n)	Generates an n-by-n identity matrix.	A = eye(3);	A = [1 0 0; 0 1 0; 0 0 1];
4	rand(n, m)	Generates an n-by-m matrix with random numbers between 0 and 1.	A = rand(2, 2);	A = [0.8147 0.9134; 0.9058 0.6324];*
5	linspace(a, b, n)	Generates a row vector with n linearly spaced elements between a and b.	v = linspace(1, 10, 10);	v = [1 2 3 4 5 6 7 8 9 10];
6	diag(v)	Creates a square diagonal matrix from the vector v.	A = diag([1 2 3]);	A = [1 0 0; 0 2 0; 0 0 3];
7	reshape(A, m, n)	Reshapes the matrix A into an m-by-n matrix, maintaining the elements.	B = reshape([1 2 3 4 5 6], 2, 3);	B = [1 3 5; 2 4 6];

2.2.5.3 Handling matrices in MATLAB

The most commonly used instructions for handling matrices in MATLAB are

Example No.	Command	Description
1	size(A)	Returns the size of matrix A.
2	length(A)	Returns the length of matrix A (the largest dimension).
3	A' or transpose(A)	Transposes matrix A.
4	A(i, j)	Accesses the element at the i-th row and j-th column of matrix A.
5	A(i, :) or A(:, j)	Accesses the i-th row or j-th column of matrix A.
6	A(i:j, k:l)	Extracts a sub-matrix from A starting from the i-th row and k-th column to the j-th row and l-th column.
7	det(A)	Computes the determinant of matrix A.
8	inv(A)	Computes the inverse of matrix A.
9	eig(A)	Computes the eigenvalues of matrix A.

10	rank(A)	Computes the rank of matrix A.

Here some example

```
 >>A = [1 2 3; 4 5 6; 7 8 9]
size(A)

A =

     1     2     3
     4     5     6
     7     8     9


ans =

     3     3

>>length(A)

ans =

     3
>>A'

ans =

     1     4     7
     2     5     8
     3     6     9

>>det(A)

ans =

   6.6613e-16


>>inv(A)

ans =

   1.0e+16 *

   -0.4504    0.9007   -0.4504
    0.9007   -1.8014    0.9007
   -0.4504    0.9007   -0.4504

>>eig(A)

ans =

   16.1168
   -1.1168
   -0.0000
```

```
>>rank(A)

ans =

     2
```

Exemple 2

Determinant, Inverse, and Rank of a 4x4 Matrix:

```
M = [3 , 5, 10, 12;
     4, 6, 7, 5;
     1, 2, 8, 1;
     2, 8, 3, 9];
% Compute the determinant of matrix M
determinant = det(M);
% Check if the matrix is invertible
if determinant ~= 0
    inverseofM = inv(M);
else
    inverseofM = 'Matrix is singular and cannot be inverted.';
end


% Compute the rank of matrix M
rank = rank(M);
% Display the results
disp(['Determinant of matrix M: ', num2str(determinant)]);
disp('Inverse of matrix M:');
disp(inverseofM);
disp(['Rank of matrix M: ', num2str(rank)]);
```

```
Determinant of matrix M: 1044
Inverse of matrix M:
    0.0517    0.4320   -0.3410   -0.2711
   -0.1552   -0.0182    0.1341    0.2021
    0.0172   -0.0412    0.1456   -0.0163
    0.1207   -0.0661   -0.0920   -0.0029

Rank of matrix M: 4
```

The values of the coefficients of polynomials appear as arrays. In descending powers, the values of the coefficients are listed .

The polyval command allows evaluating the polynomial p (the polynomial function) at given points. The syntax is polyval(p,x) where x is a numeric value or a array. In the second case, you get a array containing the values of the polynomial function at the different points specified in the x array.

Here are some common operations and functions in MATLAB for dealing with polynomials:

Defining a Polynomial: As mentioned, you can define a polynomial by creating a vector of its coefficients.

```
P = [4 3 2]; % Represents the polynomial 4x^2 + 3x + 2

P =

    4     3     2
```

You can evaluate a polynomial at specific points using the polyval function.

```
>>x = 2;
c= polyval(P, x) % Evaluates the polynomial at x = 2

c =

    24
```

The roots function can be used to find the roots of a polynomial.

```
>>r = roots(P) % Finds the roots of the polynomial

r =

  -0.3750 + 0.5995i
  -0.3750 - 0.5995i
```

Chapter 3
The Graphics

Graphics in MATLAB are a crucial component of the software since they allow you to visually comprehend data. Your results can be shown in the form of static or animated 2D or 3D graphs. These visualizations can be used to analyze data as well as convey your findings to others.

3.1 Management of graphic windows

Creating, arranging, and customizing figure windows are frequently involved in managing visual windows in MATLAB. Here are a few typical commands and actions for controlling graphic windows:

Creating New Figures: The figure function is used to create a new figure window.

```
Figure
```

Numbering Figures: You can also number your figures, which is useful when creating multiple figure windows.

```
figure(1)
figure(2)
```

Closing Figures: You can close a specific figure using the close command with the figure number, or close all figures using close all.

```
close(1) % Closes figure 1
close all % Closes all figures
```

Setting the Figure Size and Position: You can specify the size and position of the figure window using the set command and the 'Position' property.

```
set(gcf, 'Position', [x y width height])
```

Saving Figures: Figures can be saved for later use or for inclusion in publications using the saveas function.

```
saveas(gcf,'filename.png')
```

Where gcf (get current figure) returns the current figure handle

3.2 2D graphical representation

The most basic function for creating a graph in MATLAB is the **plot** function, which generates a 2D line plot . You can draw a series of points using the coordinates (x_i, y_i), where i=1,..., N using the plot command.

The syntax is **plot(x,y)** where x is the vector containing the xi values on the x-axis and y is the vector containing the y_i values on the y-axis.
Here is example of draw with plot function

```
>> x = 0:0.01:2*pi;
y = sin(x);
plot(x,y)
```

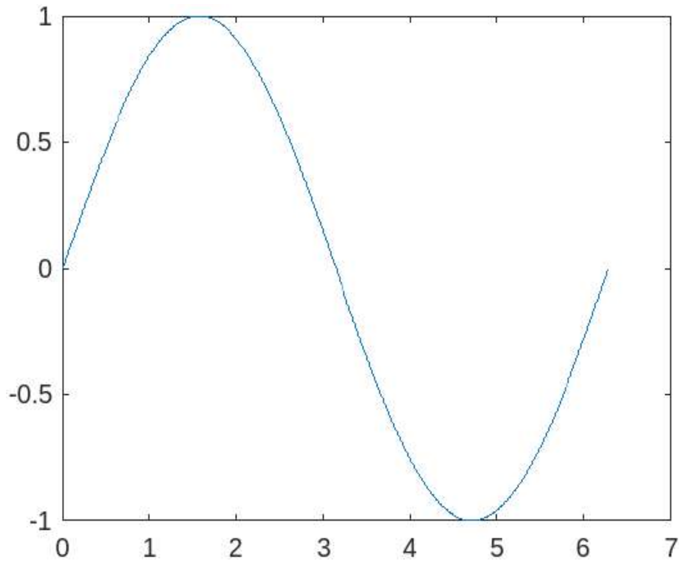


Figure 0-1 plot of exemple 1

The second exmaple here we have only a single array as an argument: it considers the values of the array as the elements of the Y-axis (ordinates), and their relative positions will define the X-axis (abscissae).

```
>>A=[1 3 5 8 4 2 10 7 1]
plot(A)
A =

     1     3     5     8     4     2    10     7     1
```

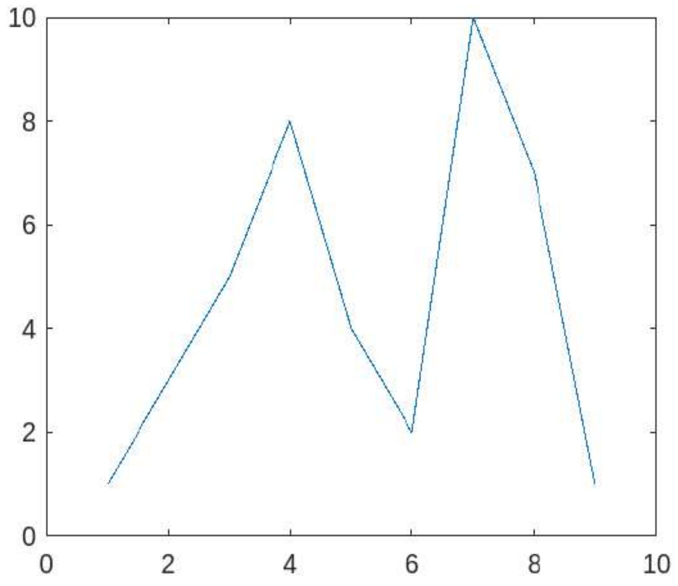


Figure 0-2 plot of exemple 2

The third exmaple here we have only a matrix as an argument. Since A contains three columns, plot(A) in this example generates three distinct line plots. A column in the matrix corresponds to each line plot. The y-axis displays the values of the components in each column, while the x-axis displays the index of each row in the matrix.

```
>>A=[2 4 6;8 9 11;3 6 8]
plot(A)

A =

     2     4     6
     8     9    11
     3     6     8
```

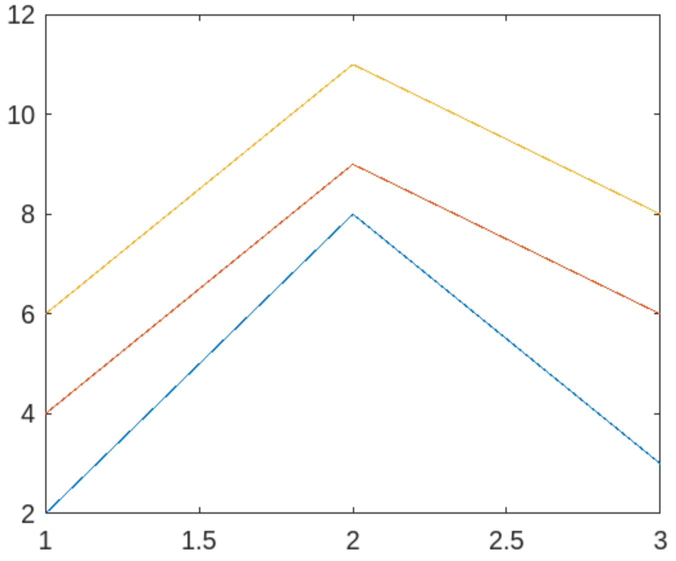


Figure 0-3 plot of exemple 3

3.2.1 Improve the readability of a figure

Improving the visibility of a figure in MATLAB entails applying a number of elements that aid in the understanding of the figure. Colors, line styles, labels, titles, legends, and axis scales may all need to be tweaked.

For this, a new argument is added (which we can call a feature) of string type to the plot function like this:

plot(x, y, ' **feature**')

Here is a table summarizing some common functions used to improve the readability of a figure in MATLAB:

Function	**Description**	**Example Usage**	**Example Effect**
xlabel('...')	Labels the x-axis	xlabel('Time (s)')	Labels the x-axis as "Time (s)"
ylabel('...')	Labels the y-axis	ylabel('Voltage (V)')	Labels the y-axis as "Voltage (V)"
title('...')	Adds a title to the plot	title('Time vs Voltage')	Adds the title "Time vs Voltage" to the plot
legend('...','...')	Adds a legend to the plot	legend('Experiment','Model')	Adds a legend with "Experiment" and

			"Model"
grid on	Turns on the grid lines	grid on	Turns the grid lines on the plot
xlim([xmin xmax])	Sets the limits of the x-axis	xlim([0 10])	Sets the x-axis limits from 0 to 10
ylim([ymin ymax])	Sets the limits of the y-axis	ylim([-1 1])	Sets the y-axis limits from -1 to 1
plot(..., 'LineStyle')	Changes the line style	plot(x, y, '--')	Plots y vs x with a dashed line
plot(..., 'Color')	Changes the line color	plot(x, y, 'r')	Plots y vs x with a red line
plot(..., 'Marker')	Adds markers to the line	plot(x, y, 'o')	Plots y vs x with circle markers at each data point
subplot(m,n,p)	Creates a grid of m by n subplots and makes the p-th subplot active	subplot(2,1,1)	Divides the figure window into 2 subplots and makes the first subplot active
figure	Creates a new figure window	figure	Opens a new figure window

Exemple :
Here is an example of how to plot a figure with a specific color using MATLAB. Let's plot a sine wave with a red line color:

```
>>x = linspace(0,2*pi,100); % Create an array of x values from 0 to 2pi
y = sin(x); % Compute the corresponding y values
plot(x, y, 'r') % Plot y vs x with a red line
```

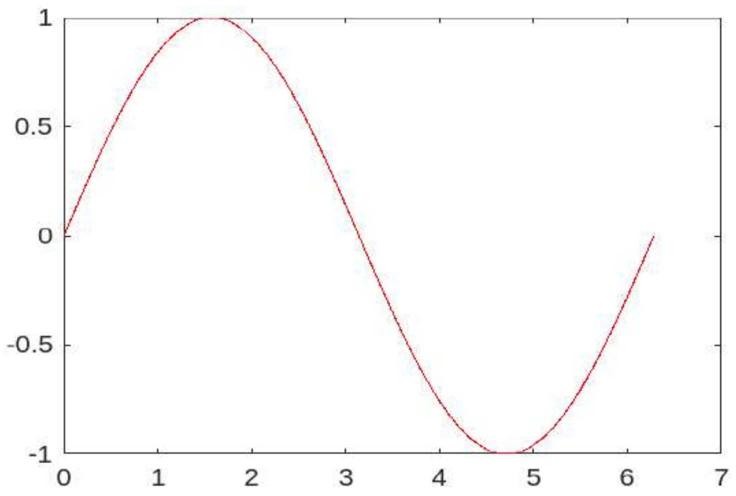


Figure 0-4 plot a figure with a specific color

In the plot function, the 'r' specifies the color of the line. For other color options include 'b' for blue, 'g' for green, 'k' for black, 'm' for magenta, 'c' for cyan, 'y' for yellow, and 'w' for white.

It is preferable to provide a textual description in a figure to help the user comprehend the meaning of the axes and the aim or interest of the concerned visualization.

- To give a title to a figure containing a curve, we use the **title** function like this:

```
title('Figure title ');
```

- To give a labels the x-axis, we use the **xlabel** function like this

```
xlabel('...')
```

- To give a labels the y-axis, we use the **ylabel** function like this

```
ylabel('...')
```

- to toggle the visibility of grid lines in your plot, we can use the **grid** function

```
grid on
```

- To make the line of the graph bolder in MATLAB, you can adjust the '**LineWidth**' property of the plot

```
% Generate an array of 100 values evenly spaced from 0 to 2pi
x = linspace(0, 2*pi, 100);


% Calculate the sine of each value in x
y = sin(x);


% Plot y vs x
figure; % create a new figure
plot(x, y, 'LineWidth', 3);
grid on
% Label the plot
title('Sine Wave');
xlabel('x');
ylabel('sin(x)');
```

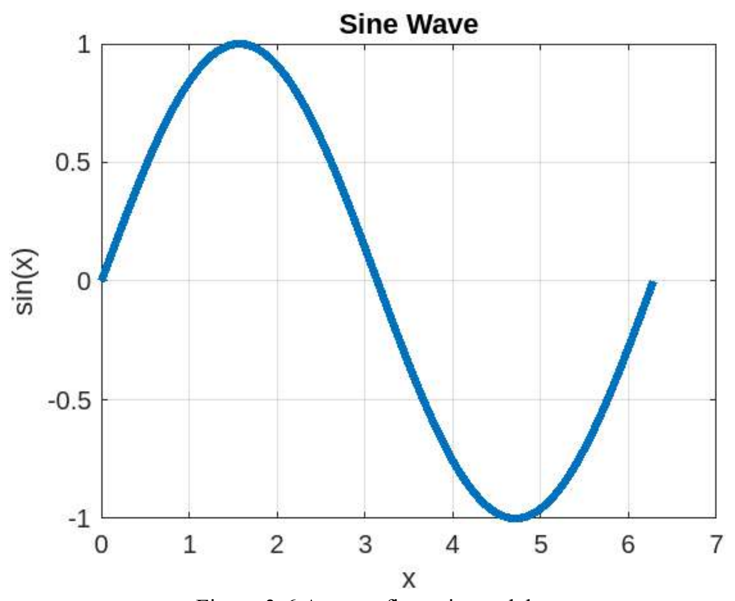


Figure 3-6 Anotate figure in matlab

3.2.2 The diagrams in Matlab

The MATLAB language does not only allow the display of points for plotting curves, but it also offers the possibility to draw bar graphs and histograms

Here's a general overview of creating bar graphs and histograms in MATLAB:

Bar Graphs:

MATLAB's bar() function allows you to create bar graphs. It can be used for both simple and grouped data.

```
% Simple bar graph
data = [5 15 25 35];
bar(data);
title('Simple Bar Graph');
xlabel('Categories');
ylabel('Values');
```

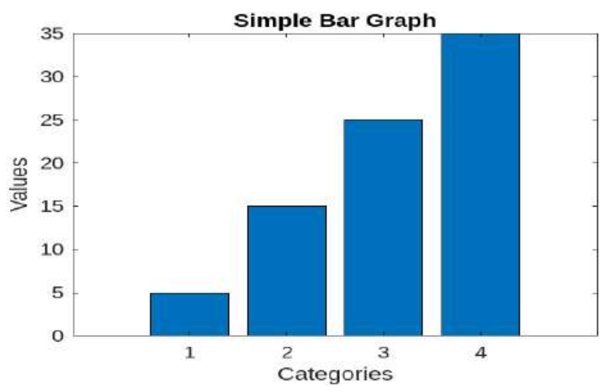


Figure 3-7 simple bar graph in matlab.

```
% Grouped bar graph
data = [5 10 40; 2 20 60; 12 24 36];
bar(data);
title('Grouped Bar Graph');
xlabel('Categories');
ylabel('Values');
legend('section 1', 'section 2', 'section 3');
```

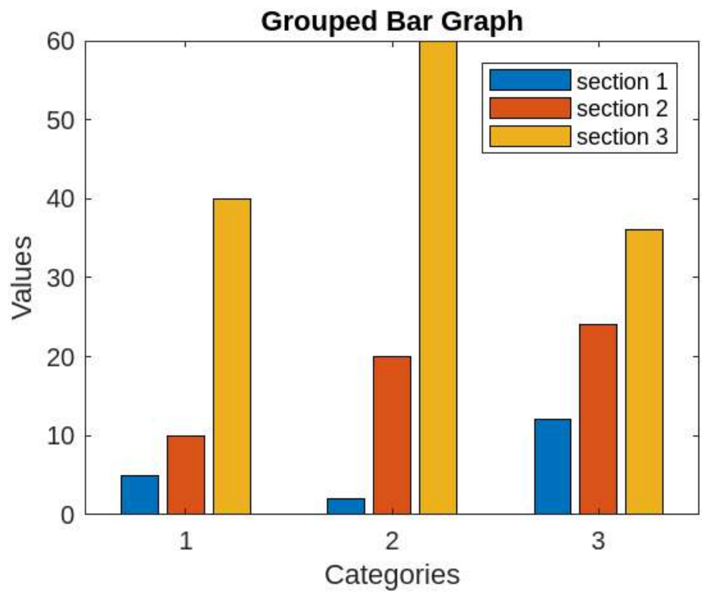


Figure 3-8 simple bar graph in matlab.

Histograms:

Histograms are used to represent frequency distributions. MATLAB's histogram() function makes it easy to generate histograms.

```
 data = randn(1, 2000); % Random data from a standard normal distribution
histogram(data, 60);  % 60 bins
title('Histogram');
xlabel('Data Values');
ylabel('Frequency');
```

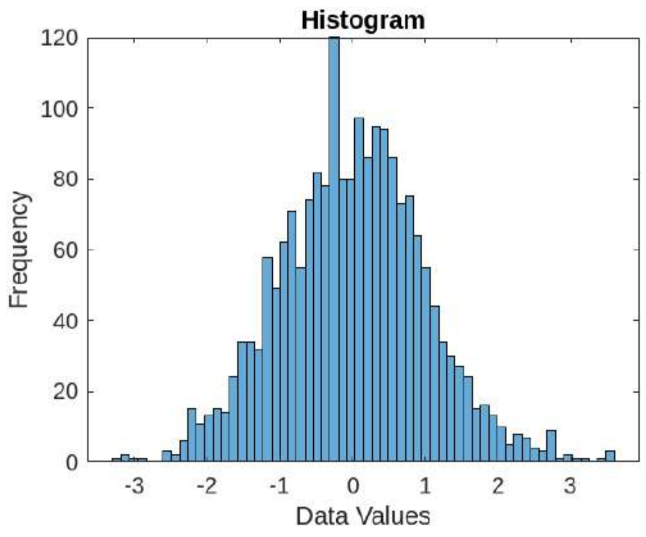


Figure 3-9 simple bar graph in matlab.

3.2.3 3D Plots

Three-dimensional plots typically display a surface defined by a function in two variables, z=f(x,y)

Surface plots, mesh plots, and contour plots can be created for 3D data visualization.

```
[X,Y] = meshgrid(-5:0.25:5);
Z = X.^2 + Y.^2;
surf(X,Y,Z); % for surface plot
```

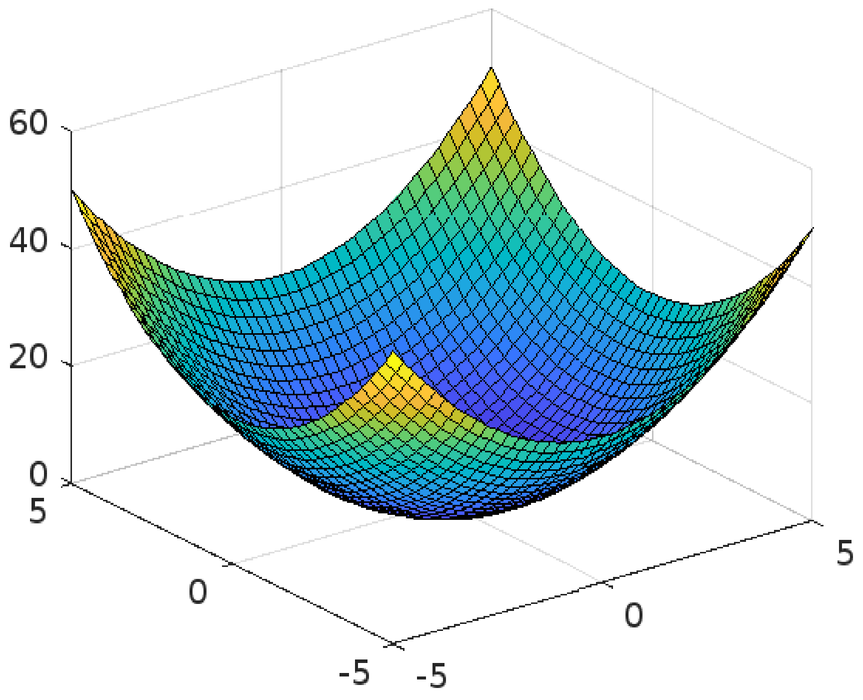


Figure 3-19 3D plot in matlab.

Exemple :

Create a 3D plot for the function $z = \sin(\sqrt{x^2 + y^2}$

```
[x, y] = meshgrid(linspace(-4, 4, 100), linspace(-4, 4, 100)); % Generate x, y data
z = sin(sqrt(x.^2 + y.^2));
figure;
surf(x, y, z); % Create a 3D surface plot
% Add labels, title, and colorbar
xlabel('X-axis');
ylabel('Y-axis');
zlabel('Z-axis');
title('3D Plot of z = sin(sqrt(x^2 + y^2))');
colorbar;
```

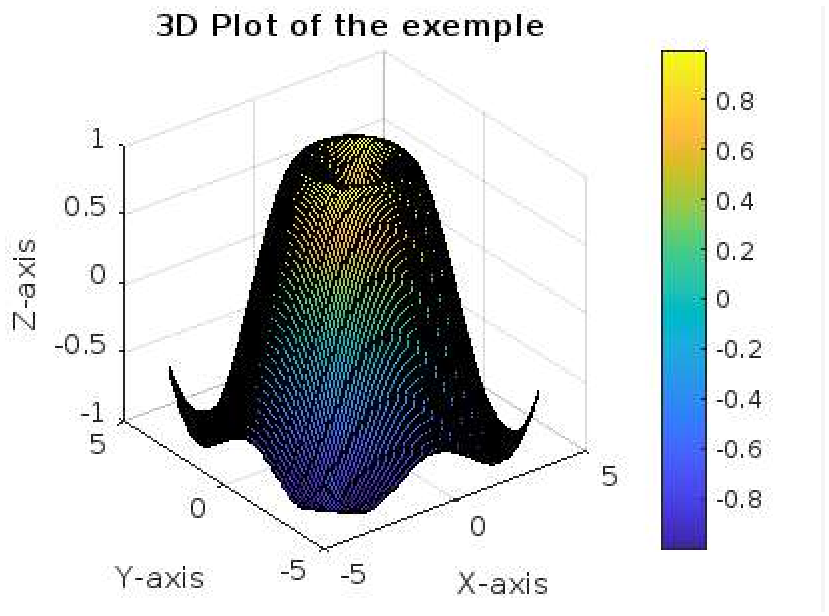


Figure 3-20 3D plot of the exemple.

Chapter 4
Programming in MATLAB

Programming in MATLAB is the process of creating scripts or functions that use the MATLAB language and resources to tackle a variety of issues. MATLAB (short for "Matrix Laboratory") is primarily intended for numerical computation, although it is also used for symbolic computation, data analysis, method creation, and modeling.

4.1 Arithmetic, logical operators and special characters

There are numerous operators and special characters in MATLAB that can be used to perform arithmetic operations, logical comparisons, and other specific tasks. Here's a quick rundown:

1. Arithmetic Operators:
    - + : Addition
    - - : Subtraction
    - * : Matrix multiplication
    - .* : Element-wise multiplication
    - / : Right division (similar to matrix division A/B)
    - ./ : Element-wise right division
    - \ : Left division (similar to matrix division B\A)
    - .\ : Element-wise left division
    - ^ : Matrix power
    - .^ : Element-wise power
    - ' : Complex conjugate transpose (Hermitian transpose)
    - .' : Non-conjugate transpose
2. **Logical Operators:**
    - & : Element-wise AND
    - | : Element-wise OR
    - ~ : NOT
    - && : Short-circuit AND (typically used in if statements and loops)
    - || : Short-circuit OR (typically used in if statements and loops)
    - == : Equal to
    - ~= : Not equal to
    - < : Less than
    - > : Greater than
    - <= : Less than or equal to

- >= : Greater than or equal to

3. **Special Characters:**
    - : : Colon operator (used for creating vectors, array indexing, and loops)
    - ; : Semicolon (used to suppress the output or to indicate the end of a row in a matrix)
    - , : Comma (used to separate matrix elements in a row or function arguments)
    - () : Parentheses (used for function calls, grouping expressions, and matrix indexing)
    - [] : Square brackets (used for matrix construction and concatenation)
    - % : Comment indicator (everything after this character on the same line is considered a comment)
    - ... : Line continuation (used to continue a command on the next line)
    - ' : Quote (used for creating character arrays and string transposition)
    - . : Decimal point and structure field access, as well as indicating element-wise operations.
4. **Other Special Characters:**
    - @ : Handle to a function
    - {} : Curly braces (used for cell array indexing and creation)
    - ! : System command indicator (allows you to run a system command from the MATLAB command prompt)
    - = : Assignment operator

4.2 The M-files

Scripts or functions with the.m extension are known as M-files. They are the main method for saving, sharing, and running MATLAB applications since they include MATLAB code. MATLAB employs.m files (also known as "M-files") as both script and function files. A MATLAB script (.m) is a text file containing a program, which is a series of commands written in the MATLAB programming language. MATLAB can execute it simply inputting the file name.

The MATLAB function file (.m) is identical to the script file, with the exception that it defines a system-level function rather than holding a script.

MATLAB scripts and functions must be stored in directories that are part of the PATH variable. Any text editor can be used to generate and edit these.m files.

Creating M-files:

1. Use the MATLAB Editor or any other text editor to create an M-file.
2. Save the file with a **.m** extension.

Running M-files:

1. Navigate to the directory containing your M-file using MATLAB's Current Folder browser or the **cd** command.
2. Type the name of the M-file (without the **.m** extension) in the Command Window.

4.3 Scripts and functions

You may utilize sets of commands by placing them in code files that contain scripts and functions. Scripts are the most basic form of code file since they save commands in their exact form as they would appear on the command line. Functions, on the other hand, are more adaptable and expandable.

4.3.1 Scripts:

- **Definition**: A script is a.m extension file that includes a series of MATLAB instructions. A script runs inside the current workspace when it is executed.
- **Usage**:

• Scripts are helpful for command sets you want to use frequently.

• They use or alter the variables in the primary workspace and have no predefined inputs or outputs.

• Usually used to aggregate a collection of MATLAB instructions or for simple jobs.

Exemple

Create a script in a file named exemple1.m that computes the area of a triangle:

```
b = 7;
h = 9;
a = 0.5*(b.*h)
```

You can run the script using the command line once saving the file:

```
exemple1
a =
  31.5000
```

4.3.2 Functions:

- **Definition:** An function is an M-file with the.m extension that begins with the function expression. Its is capable of accepting arguments for input and output.
- **Usage:**
  - Functions have their own local workspace, independent from the principal workspace, and are employed for activities that must be repeated with varied inputs.
  - Contains a piece of code to execute a specified purpose, enhancing reusability and adaptability..
- **Execution:** A function, similar to a script, is called by writing the function's title, however you may additionally provide appropriate input arguments and record output data.

Exemple

Create a Function in a file named area.m that computes the area of a triangle like the last exemple :

```
function A = area(c,d)
A = 0.5*(c.*d);
end
```

Without any changing the script, you may use the command prompt to execute the function once saving the file with varied c and d values:

```
Test1= area (1,5)
Test2= area (2,10)
Test3= area (3,6)
```

```
Test1 =
    2.5000
Test2 =
    10
Test3 =
     9
```

4.4 Control instructions

The control structures define the order of execution of the instructions for a Program .This order can become non-linear. Control structures offer ways to select which blocks of code to execute based on conditions, repeat blocks of code numerous times, and more, as opposed to programs running code from top to bottom in a sequential manner

Matlab has eight flow control structures, namely:

- **If, else, else if**
- **switch**
- **for**
- **while**
- **continue**
- **break**
- **try - catch**

- **if:**

The if statement evaluates a logical condition and executes a block of code if the condition is true.

```
if condition
% Code to execute if condition is true
End
```

- **else:**

The **else** statement can be used with an **if** statement to specify a block of code to be executed if the **if** condition is false.

```
if condition
   % Code to execute if condition is true
else
   % Code to execute if condition is false
end
```

- **elseif:**

The **elseif** statement allows you to test multiple conditions in an **if** statement. If the initial **if** condition is false, MATLAB checks the conditions of the **elseif** statements in order. If one of these conditions is true, MATLAB executes the corresponding block of code.

```
if condition1
% Code to execute if condition1 is true
elseif condition2
% Code to execute if condition1 is false but condition2 is true
else
% Code to execute if both condition1 and condition2 are false
End
```

Exemple

```
x = 3;
if x > 5
   disp('x is greater than 5');
elseif x < 5
   disp('x is less than 5');
else
   disp('x is equal to 5');
end
```

- **switch**

Syntax:

```
switch switch_expression
   case case_expression1
      % Code to execute if switch_expression matches case_expression1
   case case_expression2
      % Code to execute if switch_expression matches case_expression2
   ...
   otherwise
      % Code to execute if none of the case expressions match
End
```

The switch function examines an expression and decides one of multiple groups of statements to run. Every choice is a case.The switch block evaluates each case statement as long as one of them is true. A case exists when:

- Case_expression == switch_expression for numbers.
- strcmp(case_expression,switch_expression) == 1 for character vectors.
- Case_expression == switch_expression for objects that support the eq function. The extended eq function's output should be having a logical value or converted to a logical value.
- A cell array case_expression has a minimum of one member that satisfies switch_expression, as specified above for numbers, character vectors, and objects.

MATLAB runs the necessary statements and exits the switch block when a case expression is true.

Example

```
Studentname = 3; % Example value

switch Studentname
   case 1
      disp('Mohamed');
   case 2
      disp('Yacine');
   case 3
      disp('Farouk');
   case 4
      disp('Walid');
   case 5
      disp('Okba');
   case 6
      disp('Messaoud');
   case 7
      disp('Wahid');
   otherwise
      disp('Invalid Studentname. Please enter a number between 1 and 7.');
end
```

- **for:**

The **for** loop is an important control structure in MATLAB that is employed to iterate over a sequence of numbers or array elements. It enables you to run a block of code a set number of times.

Syntax:

```
for variable = start: step : end
   % Code to be executed in each iteration
end
```

Exemple

```
sum = 0;
for i = 2:2:10
   sum = sum + i;
end
```

```
sum = 0;
for i = 2:2:10
    sum = sum + i;
end
sum

sum =

    30
```

Another Exemple

```
matrix = zeros(7, 7)  % create a matrix 7x7
for i = 1:7
    for j = 1:7
        matrix(i, j) = i * j;
    end
end
disp(matrix);
```

```
 matrix =

   1      2      3      4      5      6      7
     2      4      6      8     10     12     14
     3      6      9     12     15     18     21
     4      8     12     16     20     24     28
     5     10     15     20     25     30     35
     6     12     18     24     30     36     42
     7     14     21     28     35     42     49
```

- **while**

In MATLAB, a **while** loop is a control structure that is used for repetitive operation determined by a condition. The while loop, opposite the for loop, maintains execution while as the stated condition stays valid.

Syntax:

```
while condition
   % Code to be executed while the condition is true
end
```

Exemple

```
value = 1;
while value <= 1000
   value = value * 2;
end
disp(value);
```

```
disp(value)
        1024
```

Another exemple

Doubling value of number

```
M = 1;
while M <= 50
   disp(M);  % Display current number
   M = M* 2;  % Double the number
End
```

```
M=    1

      2

      4

      8

     16

     32
```

- **continue**

The **continue** statement in MATLAB is used within loop structures (**for** and **while**) to skip the remaining portion of the current iteration and proceed directly to the next iteration of the loop.

Usage:

The **continue** statement is usually used in conjunction with a conditional statement (**if**) to skip specific iterations based on a condition.

Examples:

Using **continue** to skip even numbers and print odd numbers between 1 to 10:

```
for i = 1:10
    if mod(i, 2) == 0  % Checks if the number is even
        continue;      % Skips the rest of the loop for this iteration
    end
    disp(i);
end
```

- **Break**

The break statement allows users to quit a loop (for or while) early if particular circumstances or requirements are met.

Usage:

The break expression is commonly used alongside with an if statement to leave a loop once a certain condition is fulfilled.

Examples:

Using **break** to stop a loop once a number greater than 5 is encountered:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9];

for i = 1:length(numbers)
    if numbers(i) > 5
        break;  % Exits the loop
    end
    disp(numbers(i));
end
```

- **try - catch**

The try-catch technique in MATLAB allows users to elegantly manage failures and exceptions in your code. Instead of ending the application suddenly when an error occurs, the try-catch structure allows you to catch the issue and determine how to manage or report it.

The possibly problem-causing code is placed within the try block, and the error handling or recovery code is placed within the accompanying catch block.

Syntax:

```
try
    % Code that might cause an error
catch exception
    % Code to handle the error or display a message
end
```

Examples:

Assume that you would like to access the fifth element in an array but are unsure whether the array is long enough.

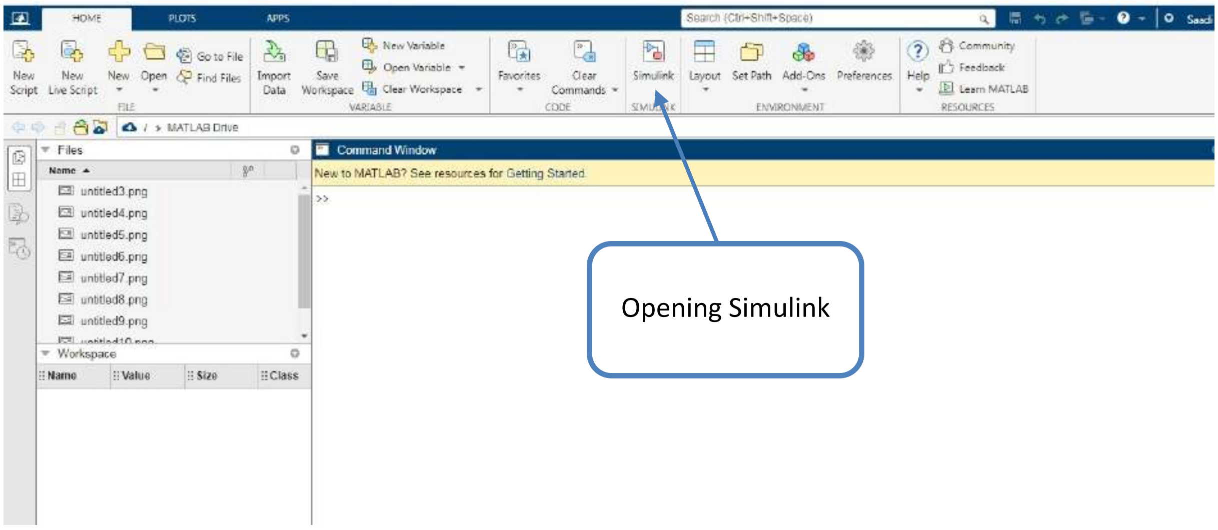
```
A = [1, 2, 3, 4];  % Array with 4 elements

try
    value = A(5);  % Attempt to access the fifth element
catch
    disp('The fifth element does not exist!');
end
```

Chapter 5
Getting started with SIMULINK

Simulink is an interface to graphical programming for modeling, simulating, and evaluating dynamic systems (such as control systems or signal processing systems). It is included with MATLAB.

Opening Simulink:

In MATLAB, you can open Simulink by typing simulink in the command window. Or clique the icon depicted ih the following figure :



New to MATLAB? See resources for Getting Started.

```
>>
```

Figure 5-1 Opening Simulink.

5.1 The libraries of SIMULINK

Simulink offers a large number of block libraries which include a range of prebuilt blocks that you may use to build your models. Each of these building blocks has been created for a certain set of functions, ranging from simple arithmetic and logical calculations to large system-level simulations.

Here's an overview of some key libraries in Simulink:

1. **Frequently Used Blocks:** This section contains commonly used blocks such as Gain, Scope, Constant, and Mux.
2. **Sources:** the fundamental blocks that generate signals, such as Sine Wave, Pulse Generator, Step, and so on.
3. **Sinks:** Signal display or storage blocks such as Scope, Display, and To Workspace.

4. **Continuous and Discrete:** These types of libraries include blocks for modeling both continuous-time and discrete-time dynamic systems. Integrator, Derivative, and Filter designs can be found here.
5. **Math Operations:** This section contains building blocks for basic arithmetic operations, mathematical functions, and other computational activities.
6. **Logical and Bit Operations:** This section includes blocks for logical operations such as AND, OR, and NOT, as well as bitwise operations.
7. **Lookup Tables:** Used to create functions that map specified input values to specific output values without the need for computation.
8. **Signal Routing:** Blocks like Selector, Switch, and Multiport Switch that route signals based on certain conditions or selections.
9. **Model Verification:** Blocks that verify, claim, or ensure that certain requirements in your model are met.
10. **Function and Subsystem Blocks:** Enables programmers to wrap a model segment into a separate block or build MATLAB functions.
11. **Ports & Subsystems:** Contains blocks for defining input and output ports as well as creating subsystems.
12. **Bus Signals:** These are blocks that allow you to group many signals together as a bus.
13. **Domain-specific libraries:** Simulink provides specialized libraries for different uses or industries. • Simulink Control Design for systems of control, for example.
14. **Communications System Toolbox,** which allows you to create and simulate communication systems.
15. Signal Processing Toolbox for processes in signal processing.

Below, we present the library continous

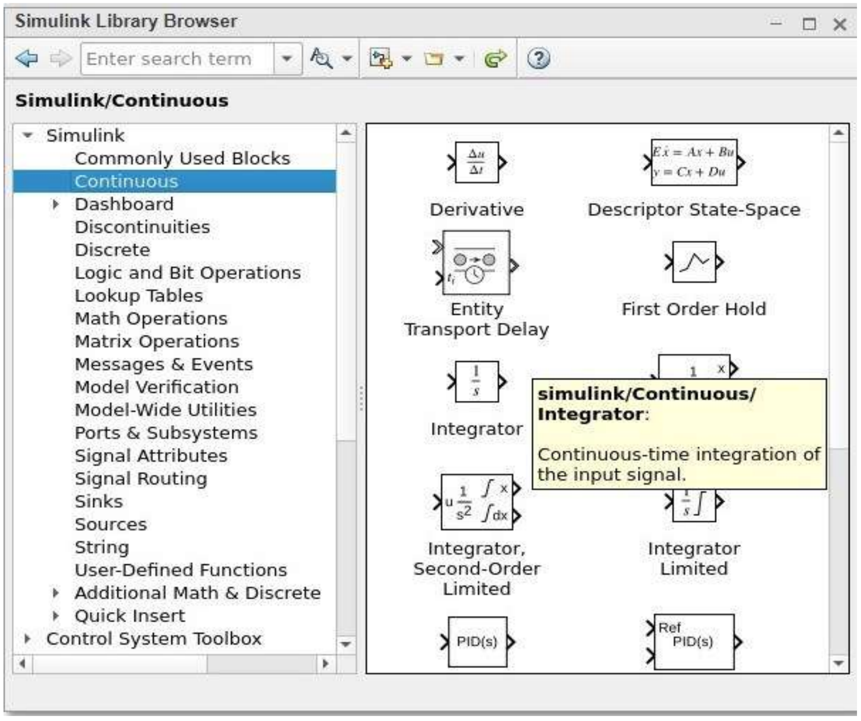


Figure 5-2 the library continous.

These are a few interesting blocks in Simulink's Continuous library:

1. **Integrator:** This block integrates the input signal over time. In differential equations or control systems, it is frequently used to indicate the integration action.

2. **Derivative :** This block computes the derivative of the input signal, as the name implies.

3. **Transfer Fcn:** A linear transfer function is represented by this block. The transfer function's numerator and denominator can be specified.

4. **State-Space:** A state-space representation of a system. You supply the system's matrices (A, B, C, D).

5. **PID Controller:** A device that is used to implement proportional-integral-derivative control techniques.

6. **Gain:** A constant factor multiplies the input signal.

7. **Saturation:** Sets the upper and lower saturation levels for the input signal.

5.2 Fast Getting started

Simulink launches a single window called Simulink Start Page, which might be viewed below.

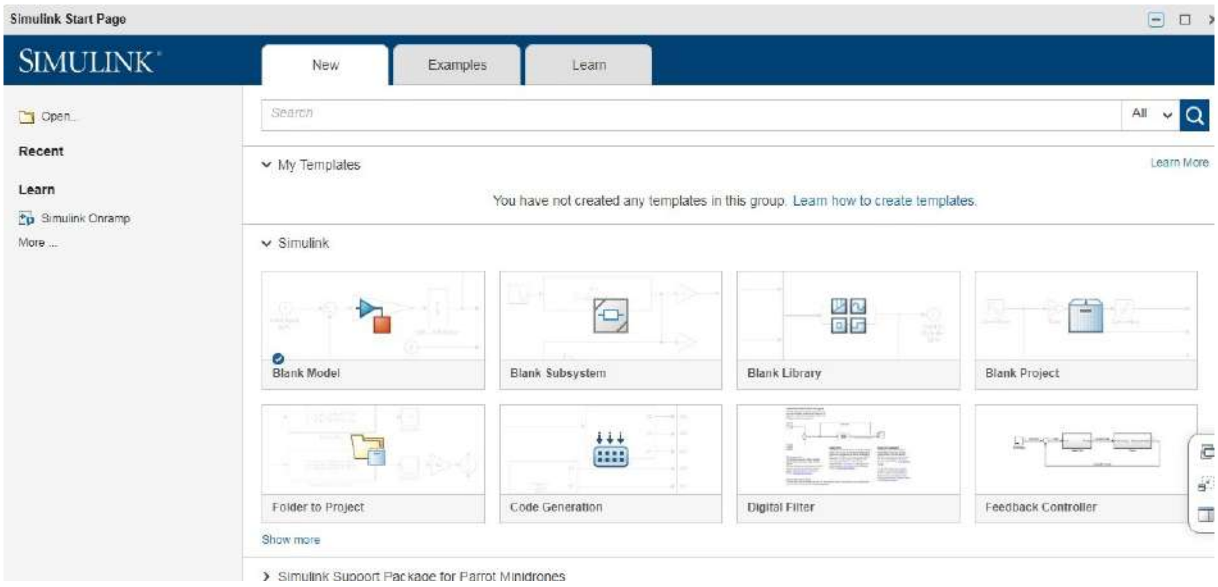


Figure 5-3 the Simulink Start Page.

A new window will emerge when ones select on Blank Model, as seen above.

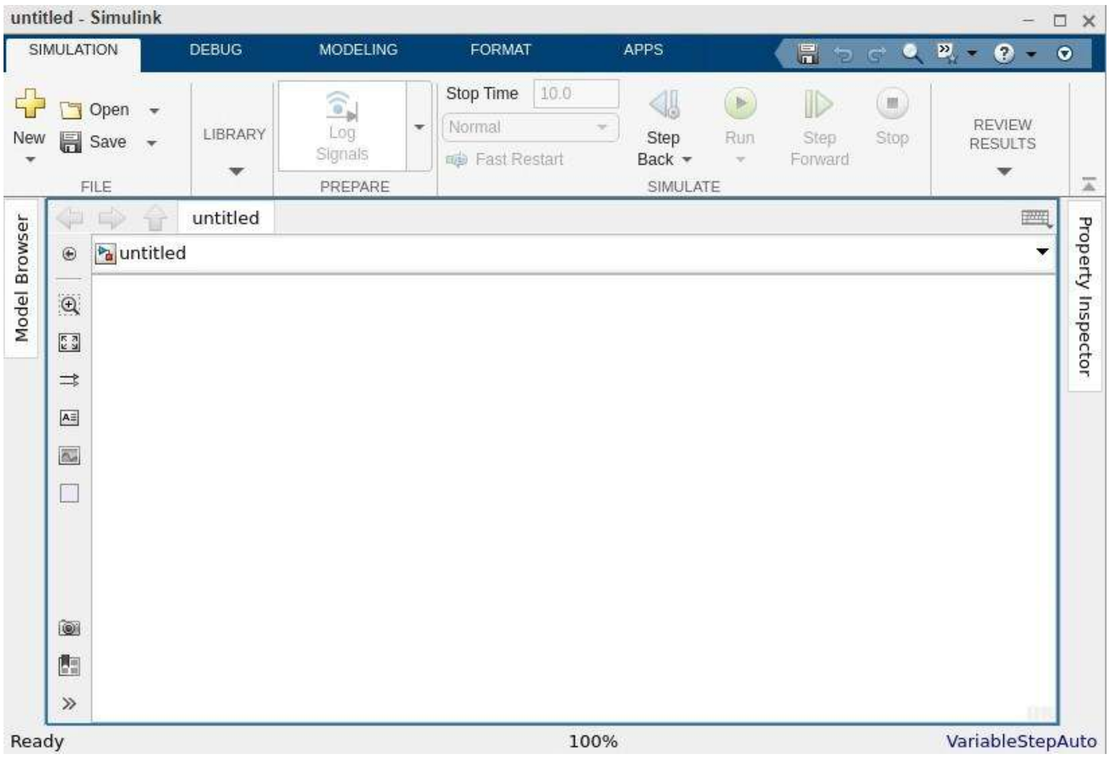


Figure 5-4 Blank Model.

A model in Simulink is an assembly of blocks that, in general, describes a system. Besides from starting from scratch, previously saved model files can be loaded using the File menu or the MATLAB command prompt.

We'll create a basic model with a sine wave as an input that feeds into a scope to visualize the waveform.

Step 1: Create a New Model

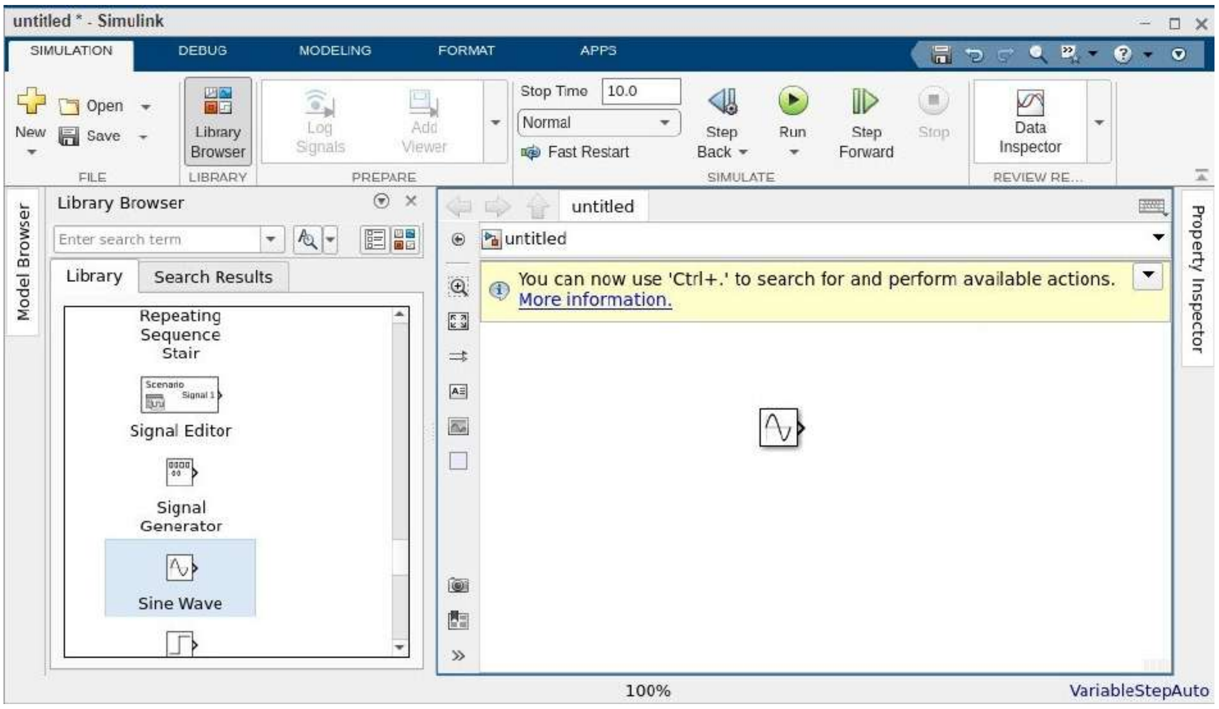
In the Simulink Start Page, click on the Blank Model option. This will open a new, empty Simulink model.

Step 2: Add a Sine Wave Source

Navigate to the Library Browser in the Simulink model window by click on the library icon or hitting Ctrl+Shift+L.

Navigate to the Simulink category in the Library Browser and then to Sources.

Drag the Sine Wave block across the empty model.



Stop Time 10.0

Normal

Fast Restart

You can now use 'Ctrl+.' to search for and perform available actions. More information.

Figure 5-5 Add sin wave Block.

Step 3: Add a Scope to Visualize the Output

1. Still in the Library Browser, navigate to Sinks under the Simulink category.
2. Drag and drop the Scope block onto your model.

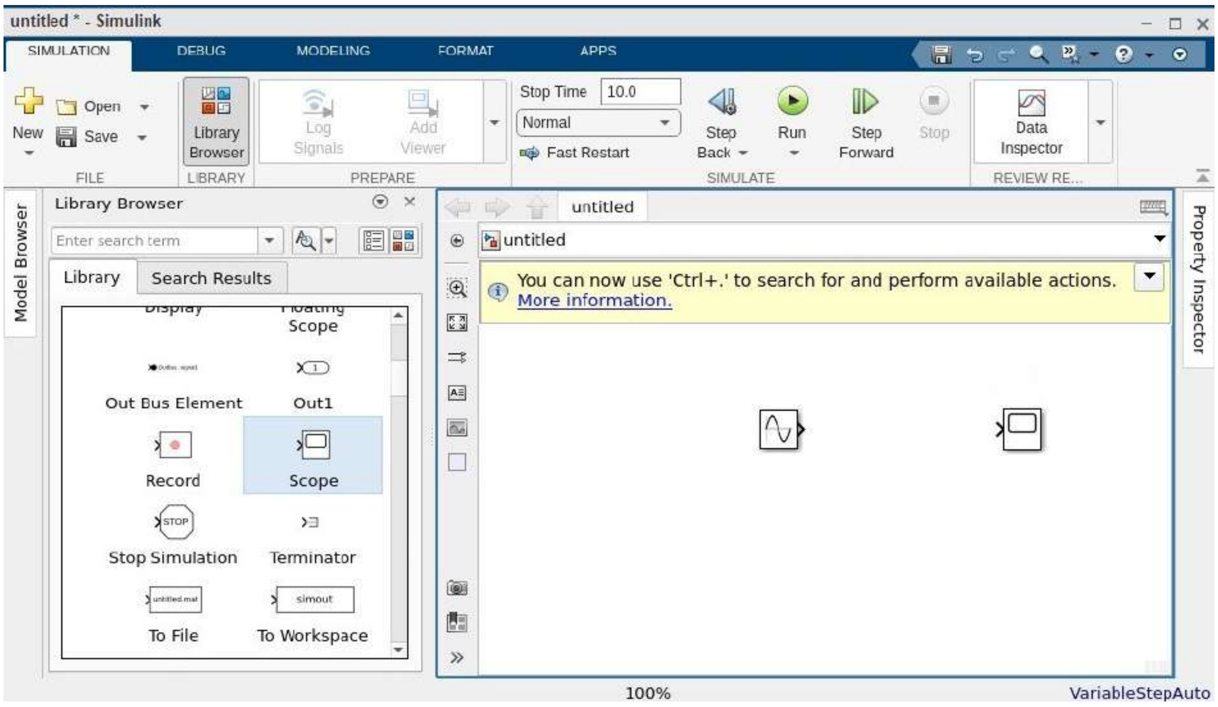


Figure 5-6 Add scope Block.

Step 4: Connect the Blocks

Select the Sine Wave block's output port (the arrow).

Establish a connection both of them by dragging a line to the Scope block's input port.

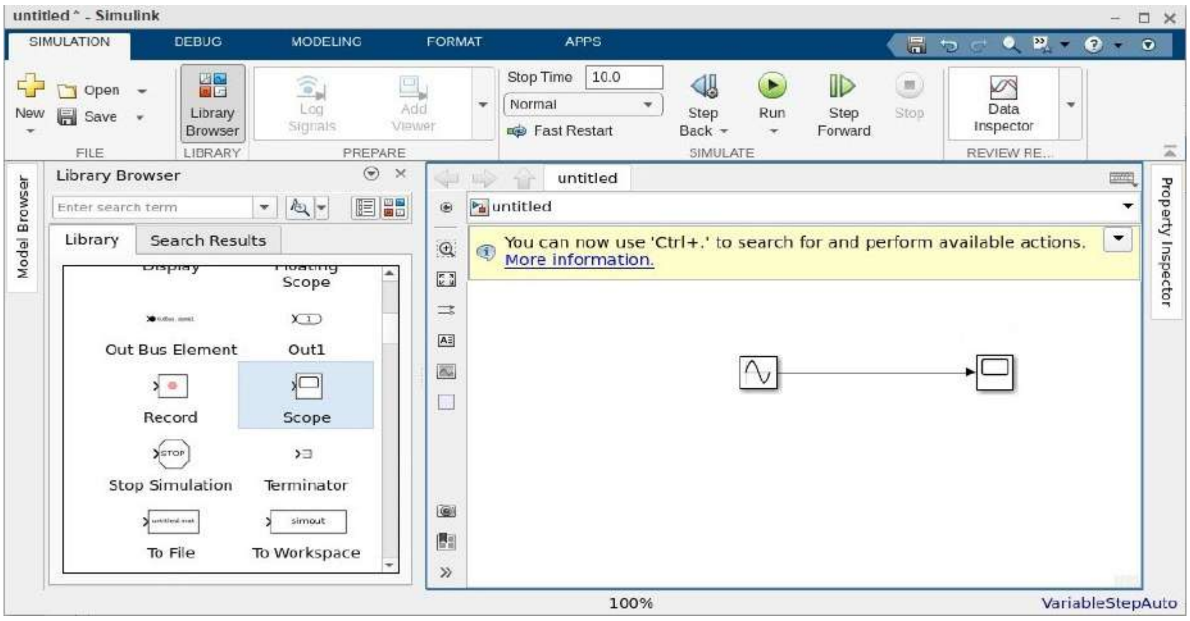
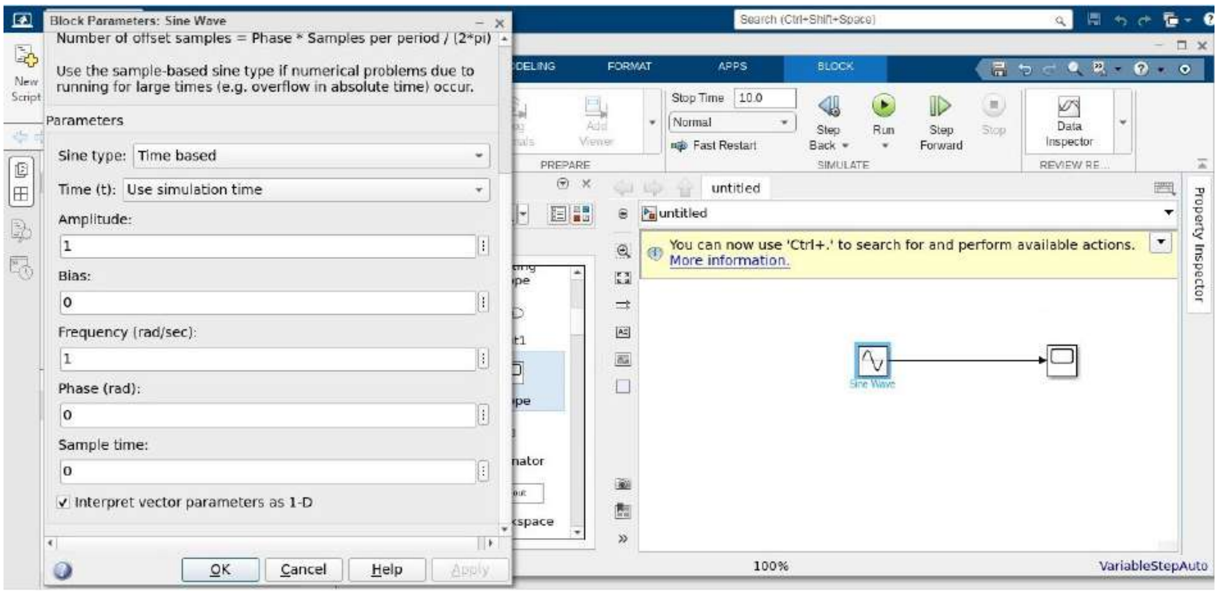


Figure 5-7 connect the two Blocks.

Step 5: Change Sine Wave Properties

1. Select the Sine Wave block by double-clicking it.
2. You can modify parameters such as as amplitude, frequency, phase, and so on.
3. After applying modifications, rerun the simulation to see the revised waveform in the scope.


Block Parameters: Sine Wave

Number of offset samples = Phase * Samples per period / (2*pi)

Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use simulation time

Amplitude: 1

Bias: 0

Frequency (rad/sec): 1

Phase (rad): 0

Sample time: 0

☑ Interpret vector parameters as 1-D

OK Cancel Help Apply

You can now use 'Ctrl+.' to search for and perform available actions. More information.

Figure 5-8 Change Sine Wave Properties.

Step 6: Simulate the Model and observe the Sine Wave

Click the Run icon in the simulation window.

The simulation will run, and although you may not notice much, the sine wave data will be supplied into the scoping.

In the model you are using, double-click the Scope block.

A window will appear that displays the sine wave over time.

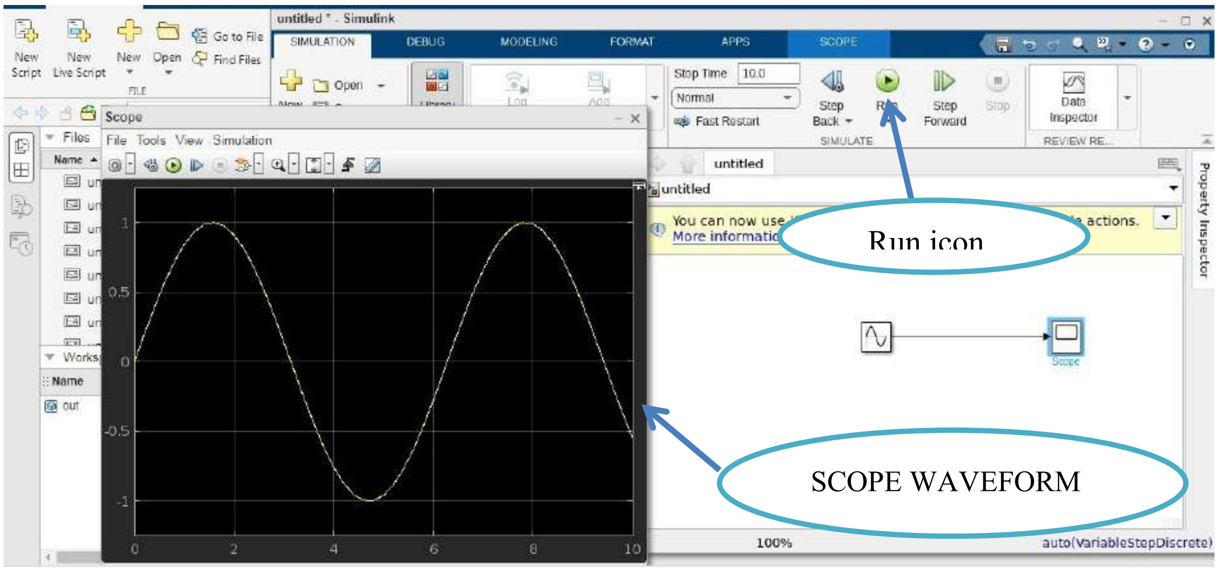


Figure 5-8 Simulate the Model and observe the Sine Wave

5.3 ExemplesOf Simulation With Simulink

The First Exemple

We will utilize Simulink to resolve the first order differential equation (ODE) as an illustration.

$$\frac{dx}{dt} = 4sint - 6x$$

Additional initial conditions of the form $x(t_0) = x_0$ at $t = t_0$ are required. To solve this issue, we'll set x(0) to 0.

The previous equation can be formally solved by integrating dx/ dt.

$$x(t) = \int (4sint - 6x(t))dt$$

At this point, using Simulink blocks, we can build the model in Simulink for simulating Equation.

To model an equation (or system) using simulink, follow the following steps:

- The selection of inputs and outputs of the equation (or system).
- Block Determination Required
- Simulation execution

After examination the equation we can found that we need the following blocks

– Sine Wave block from the Math Operations library

– Sum block from the Math Operations library,

– Integrator block from the Continuous library;

– Gain block from the Math Operations library,

– Scope block from the Sink library.

The first step is to oppen a new model and drags the necessary blocks

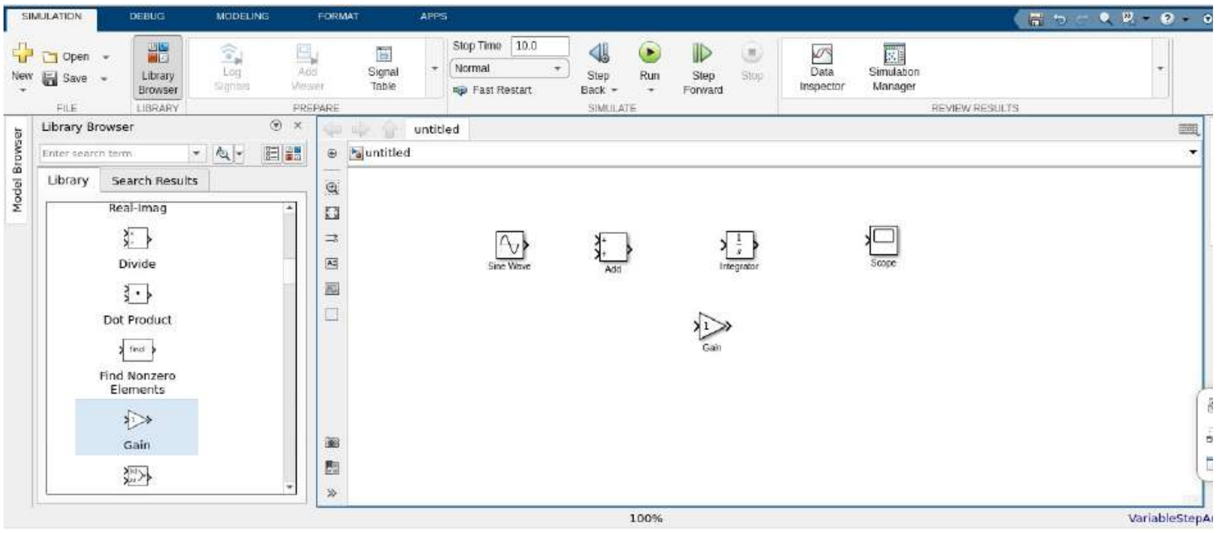
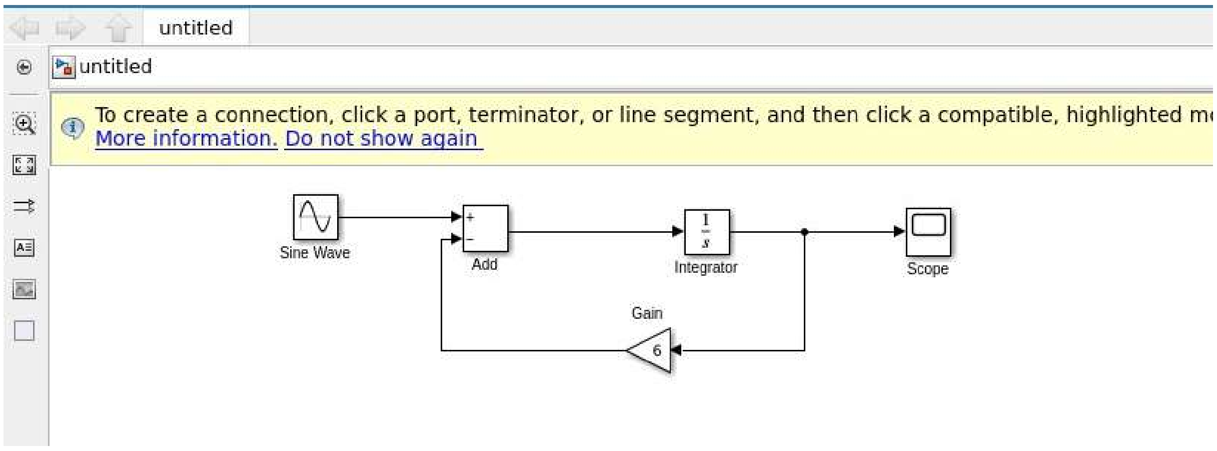


Figure 5-9 exmple step1

The sconde step is to connect the blocks According to the equation


untitled

To create a connection, click a port, terminator, or line segment, and then click a compatible, highlighted m
More information. Do not show again

Figure 5-10 exmple step2

The last step is to save the simulink file and star the simulation and observe the waveform with the scope block

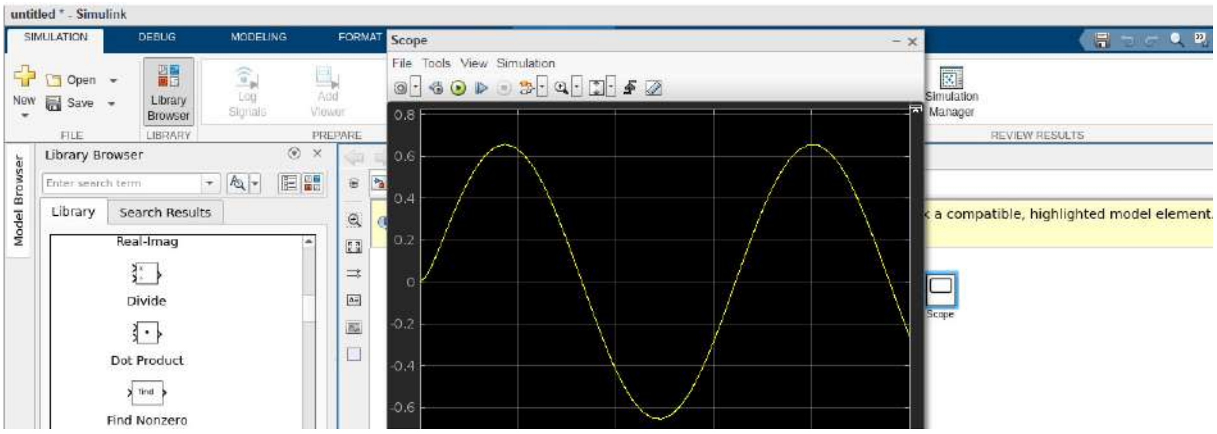


Figure 5-11 exmple step3

The Seconde exemple

In this exemple we use simuink to resolve sconde ordre ssystem RLC circuit

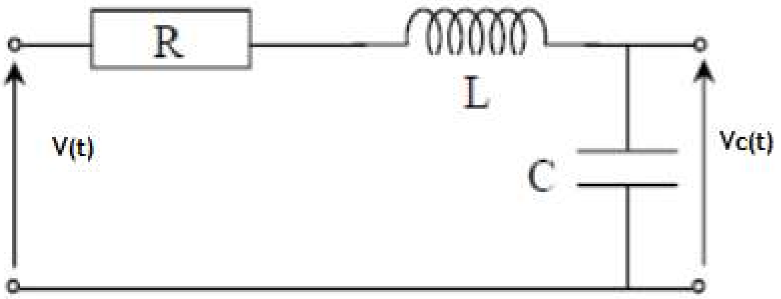


Figure 5-12 RLC circuit

Once powered by a voltage source, the differential equation for a series RLC circuit is

$$L\frac{d^2i(t)}{dt^2} + R\frac{di(t)}{dt} + \frac{1}{C}i(t)dt = V(t)$$

Avec $i(t) = C\frac{dVc(t)}{dt}$

The state equation of this system is

$$\frac{d^2Vc(t)}{dt^2} = (V(t) - Vc(t) - RC\frac{dVc(t)}{dt})LC$$

Now Launch MATLAB.

In the MATLAB command opening, type simulink and hit Enter. The Simulink start page will then be displayed.

Click "Create Model" after selecting "Blank Model."

Applying differential equations to the RLC Circuit construction:

- **Block Integrators:** Drag two "Integrator" blocks from the Simulink library to the "Continuous" section. These will be utilized in order to integrate and distinguish the voltage Vc(t).
- **Blocks of Gain:** Use "Gain" blocks for the resistance and inductance.
- **Block Product:** Use "Product" block for the multiplication operation
- **Block Sums:** To tally the outcomes of the aforementioned procedures

- **Voltage Source:** Use "Step" block or constant from the "Sources" library to simulate the input voltage

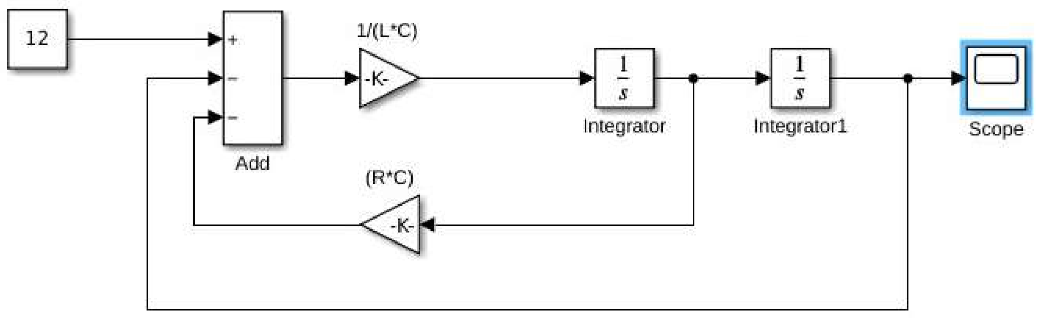


Figure 5-13 RLC circuit in simulink

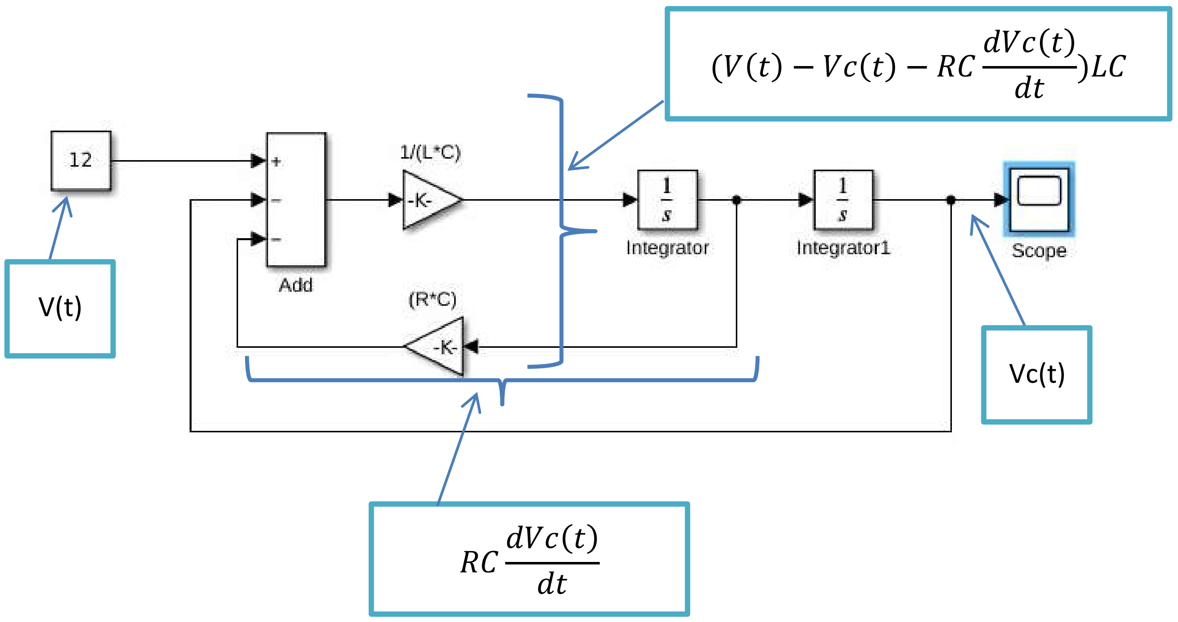


Figure 5-14 RLC circuit in simulink (equation description)

Running the Simulation:

Once everything is set up, click on the "Run" button in the Simulink window. After the simulation finishes, open the scope to view the results.

Save Your Work:

Save your model. Click on File > Save or press Ctrl + S.

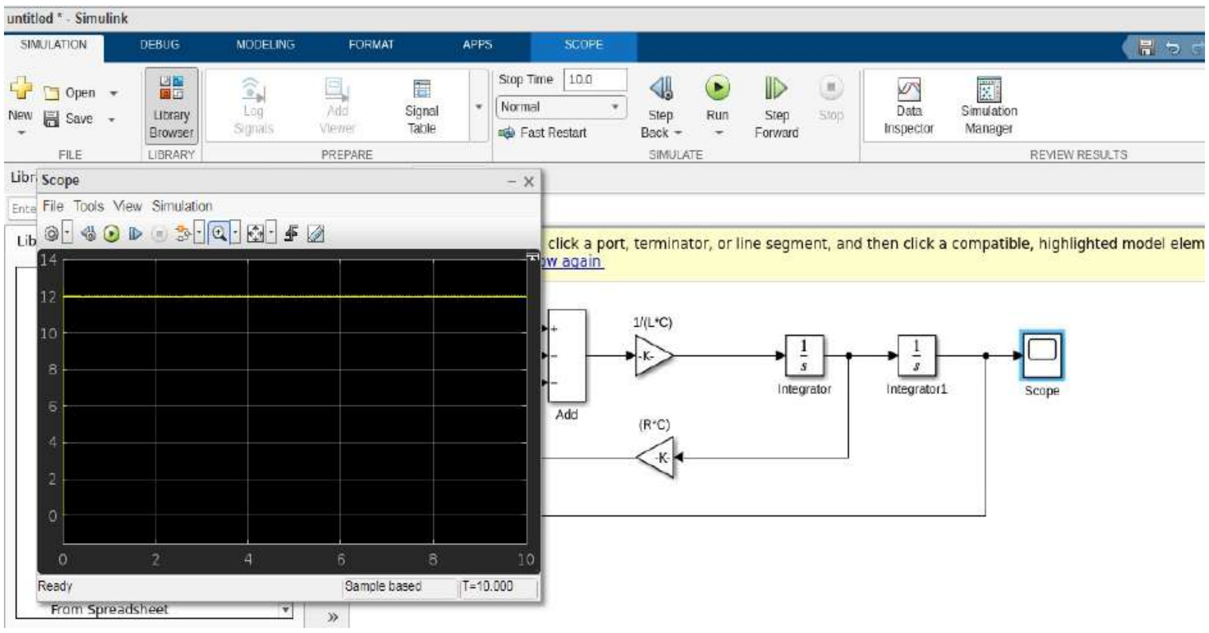


Figure 5-15 RLC circuit after simulation

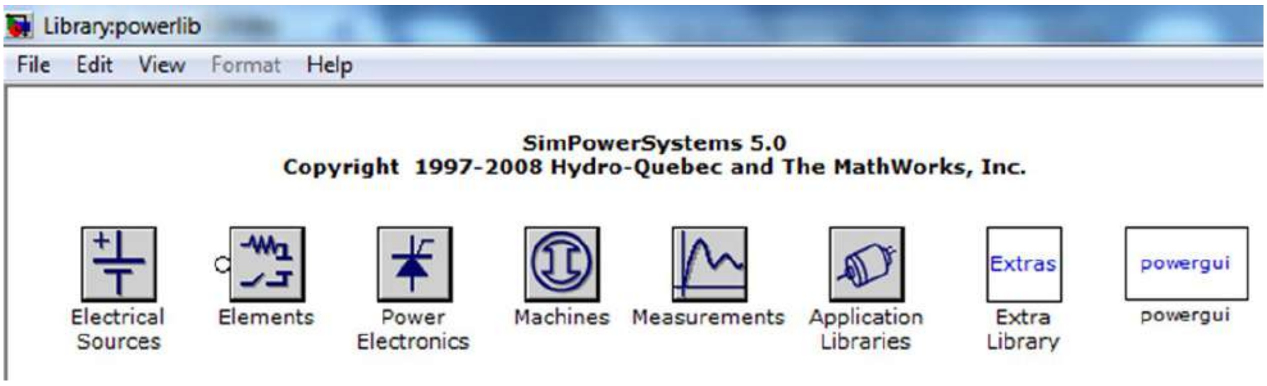
Chapter 6
Power System Blockset

For modeling and simulating electrical power systems in the MATLAB/Simulink environment, MathWorks provided the Power System Blockset toolbox. With an emphasis on power electronic and drive systems, it provides the ability to design, evaluate, and simulate electrical power systems.

6.1 Presentation of the Power System Blockset simulink

A collection of tools and libraries called "Power System Blockset" in MATLAB/Simulink are used for modeling, simulating, and analyzing electrical power systems.
Scientists and engineers are able to model and examine the functioning of electrical power systems using the systems and components provided by the Power System Blockset.
It is appropriate for a range of uses, including the generation, conversion, transmission, and consumption of electricity.

To open the main library of SimPowerSystems from Matlab, execute the following command: powerlib


Library:powerlib

SimPowerSystems 5.0
Copyright 1997-2008 Hydro-Quebec and The MathWorks, Inc.

Figure 0-1 Library powerlib

One can also access SimPowerSystems from Simulink:

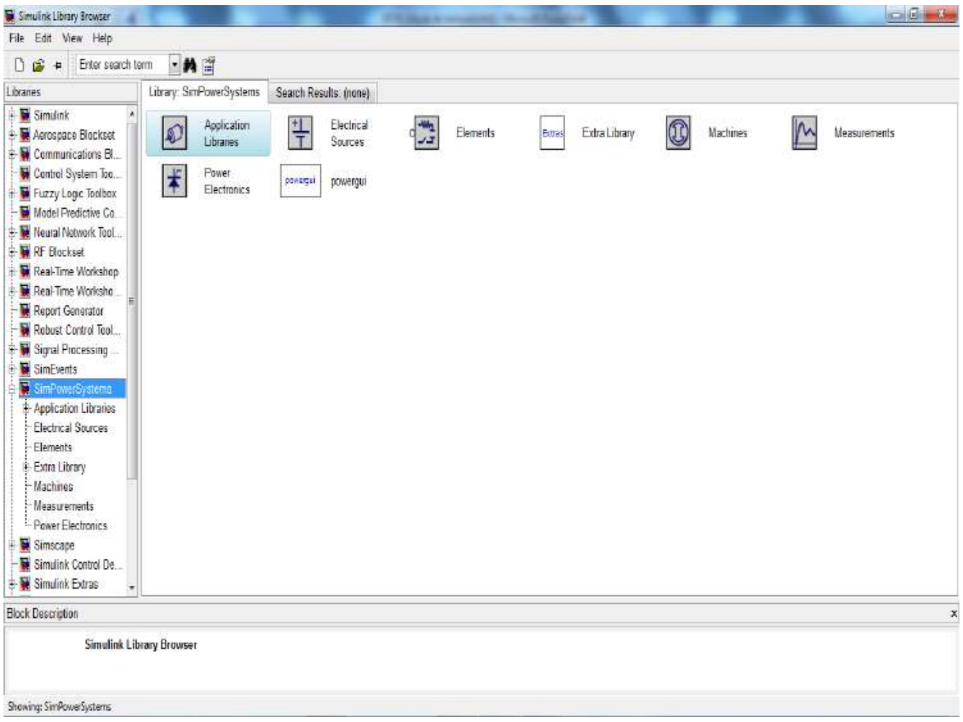


Figure 0-2 brows Library powerlib

SimPowerSystems contains various models grouped under several categories

- Electrical Sources: AC and DC voltage sources, current sources, and more.
- Lines and Transformers: Represents transmission lines, transformers with multiple winding configurations, etc.
- Machines: Synchronous machines, induction machines, and other electrical machinery.
- Loads: Different types of electrical loads, both dynamic and static.
- Protection & Control: Circuit breakers, relays, and controllers to protect and control the system.

6.1.1 Applications libraries

In the toolbox of SimPowerSystems, "Applications libraries" certainly corresponds to specific sets of components or preset systems customized for certain power system applications or research.

Th next figure represent the applications libraries

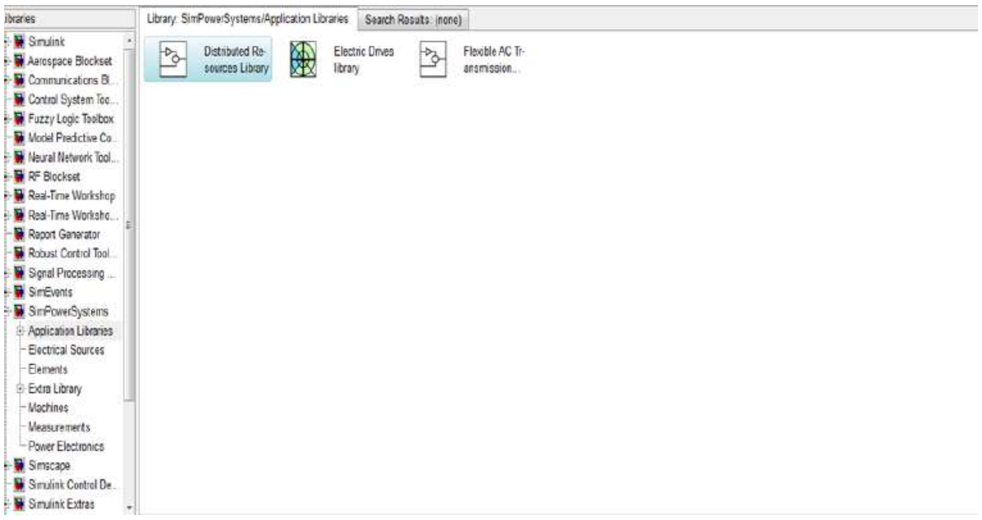


Figure 0-3 applications Library

6.1.2 Electrical sources library

Electrical sources are commonly encountered in toolboxes developed for power system and electrical engineering simulations. SimPowerSystems (later versions may refer to it as Simscape Electrical) is one such toolkit

The "Electrical Sources" library in SimPowerSystems (or Simscape Electrical) provides a variety of source components for generating electrical power in simulations. These elements are as follows:

- AC Voltage Source: Represents an ideal AC voltage source.
- DC Voltage Source: Represents an ideal DC voltage source.
- Controlled Voltage and Current Sources: Sources whose output can be controlled by external signals.
- Three-Phase Sources: For simulating three-phase electrical systems.
- Specialized Sources: These might include things like pulse generators, signal generators, or other specialized electrical sources.

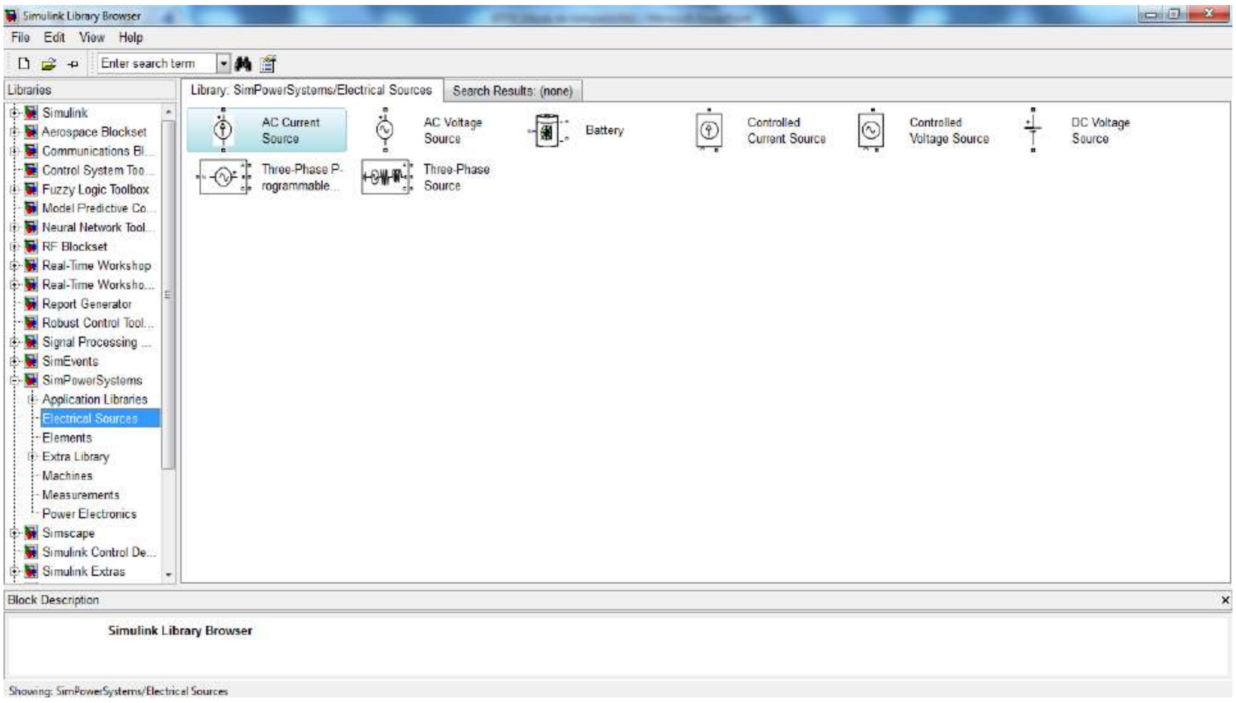


Figure 0-4 Electrical source Library

6.1.3 Element library

A collection of blocks or components that may be utilized for modeling and simulation could be referred to as a "element library." These might be included with the main Simulink program or extra toolboxes like SimPowerSystems (Simscape Electrical), SimMechanics, and others. These libraries are made up of pre-defined blocks that represent mathematical operations, system components, control elements, and so on.

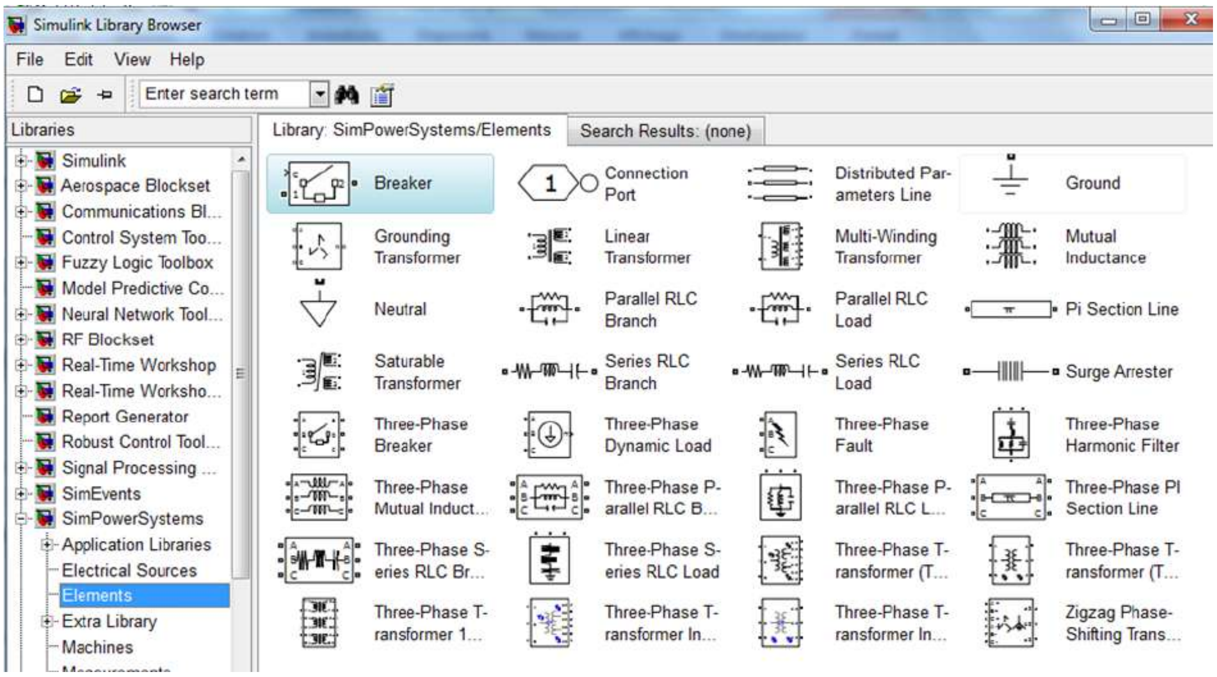


Figure 0-5 Element library

6.1.4 Extra Library

This library offers a variety of components and blocks that do not exactly fall into the basic categories of electrical sources, machines, power electronics, and so on, but are nonetheless valuable for a variety of modeling tasks.

Here's a quick rundown of some of the blocks you could find in the "Extra Library."

- Transformations are blocks that may be used to transform between different reference frames, which is very important for machine modeling.
- Measurement Blocks: These are specialized blocks that are used to measure certain parameters or circumstances in your model.
- User-defined functions or blocks: The library may include blocks that allow users to specify their own actions or equations in some cases.
- Utilities: These are blocks or systems that are meant to help in simulation, debugging, or analysis but are not components of a power system.

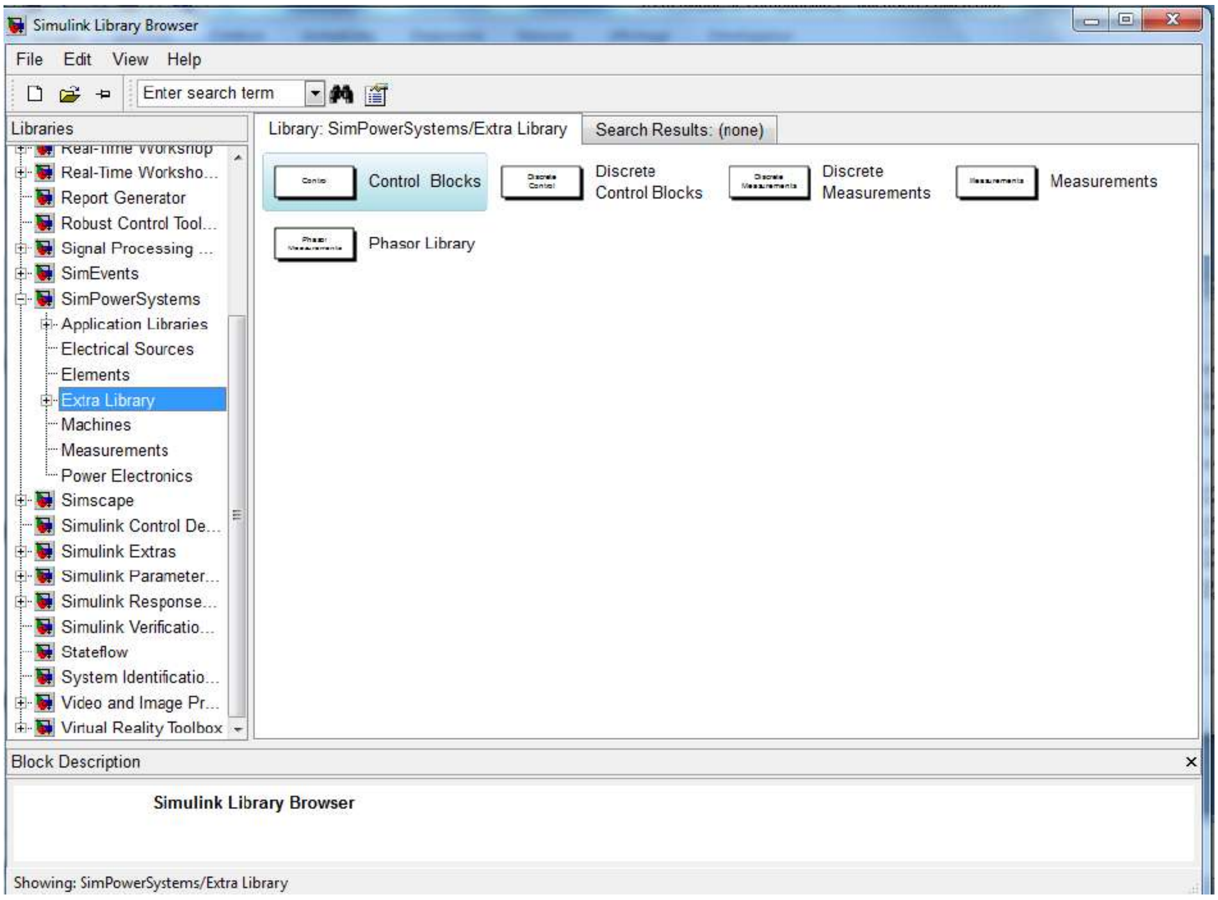


Figure 0-6 Extra Library

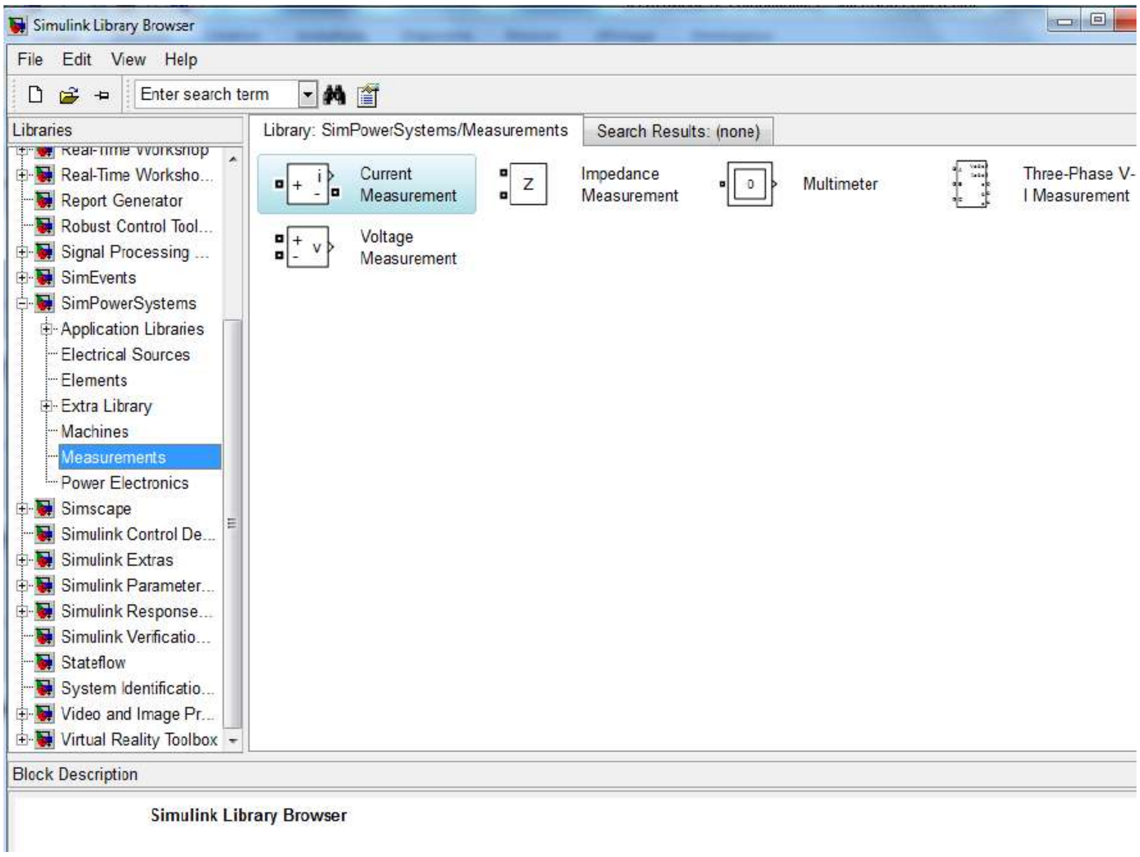


Figure 0-7 Mesurments Library

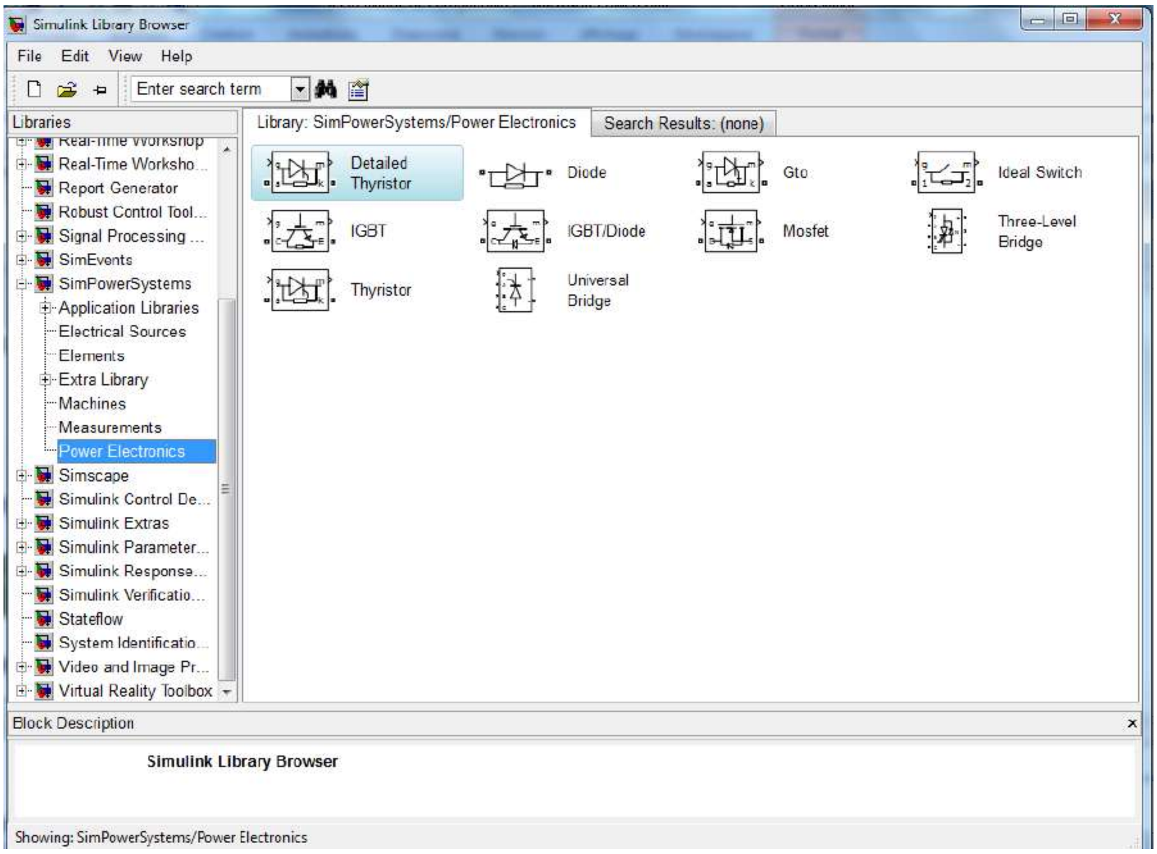


Figure 0-8 Power electronics Library

6.1.5 The Power GUI block

The "powergui" object is a powerful tool, usable with the acausal modeling of the specific library Simscape > SimPowerSystem. This block provides a graphical user interface for setting the simulation approach and accessing power system analysis tools.

Figure 0-9 Power GUI block

The powergui block provides to resolve your circuit using one of the following methods:

- Continuous, which employs a Simulink variable-step solution
- Discretization of the electrical system to provide a solution in discrete time
- The phasor mode simulation (steady-state phasor solution).

This block offers also analysis tools :

- FFT Analysis: This tool assists users in analyzing the frequency components of signals in their models. This is useful for harmonic analysis.
- Scopes & Visualization: Provides specialized scopes for visualizing three-phase signals, voltage/current phasors, and other parameters.
- Load Flow: Calculate the system's steady-state solution, including voltages, currents, active and reactive powers.
- Impedance Measurement: This analysis helps users to measure and visualize system impedance as a function of frequency.

By double-clicking on the powergui block present in the model, a window similar to the figure below opens, which includes the following sections:

1. Configure parameters
2. Steady-State Voltages and Currents
3. Initial States Setting
4. Load Flow and Machine Initialization
5. Use LTI Viewer
6. Impedance vs Frequency Measurement
7. FFT Analysis
8. Generate Report

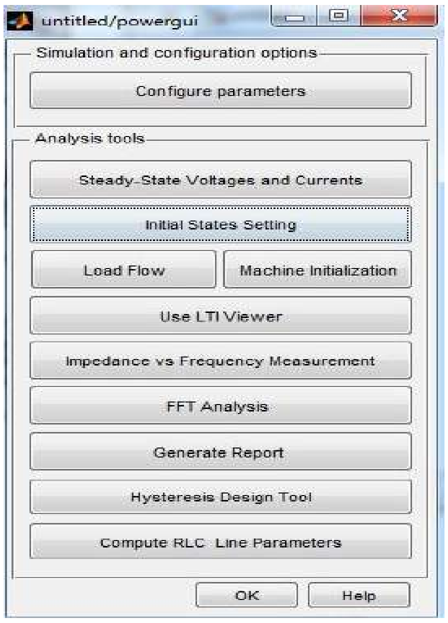


Figure 0-10 Power GUI window

6.1.6 Exemple boost converter

Boost converters have applications in electronics to provide a DC output voltage larger compared with the DC input voltage, thus increasing the supply voltage. Boost converters are frequently found in power supply for white LEDs, rechargeable batteries for electric vehicles, and a variety of different functions.

The theoretical step up equation of the boost converter is:

$$\frac{V_{out}}{V_{in}} = \frac{1}{(1-D)}$$

Where D represents the duty cycle.

The converter is coupled with resistive load from a 50 V source in this example, and the PWM frequency is fixed to 10 kHz.

In this exemple assignment, you will build a boost converter circuit in SimPowerSystems. Figure bellow depicts the circuit. A input DC source, a diode, inductor,transistor,capacitor and a resistive (R) load comprise the circuit. The source's input voltage is 50v, and the output volatge is 100 V.

Open the SimPowerSystems libraries, chose the component and drag it to simulink window

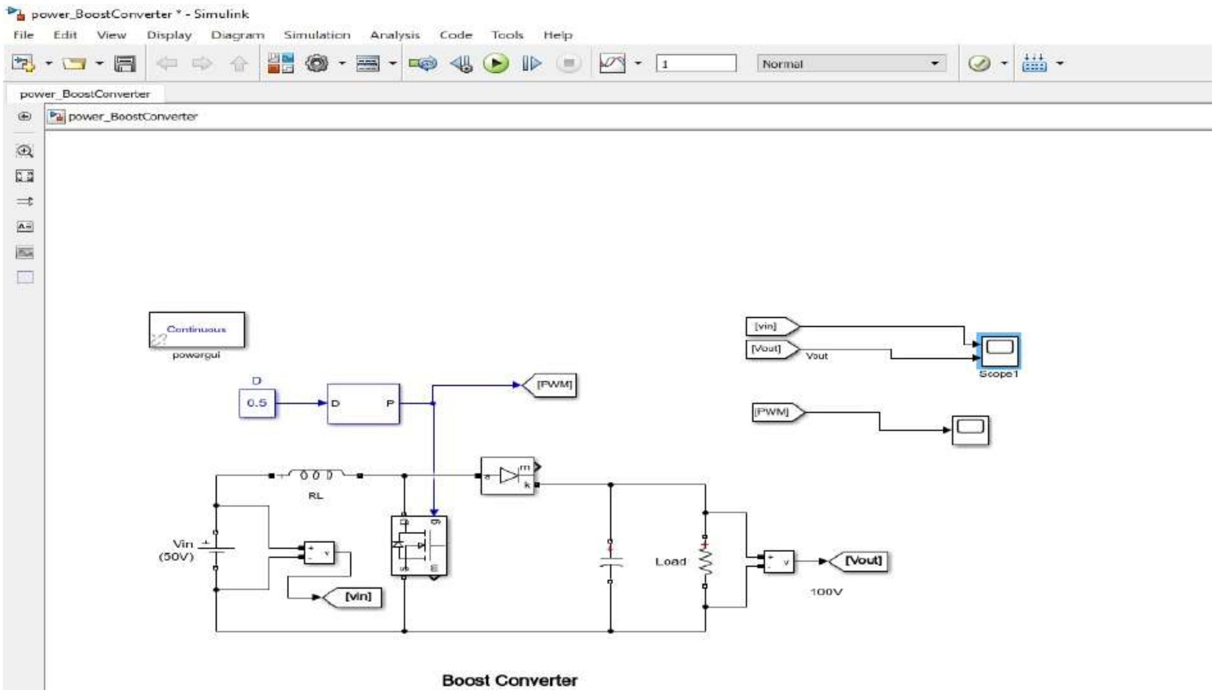
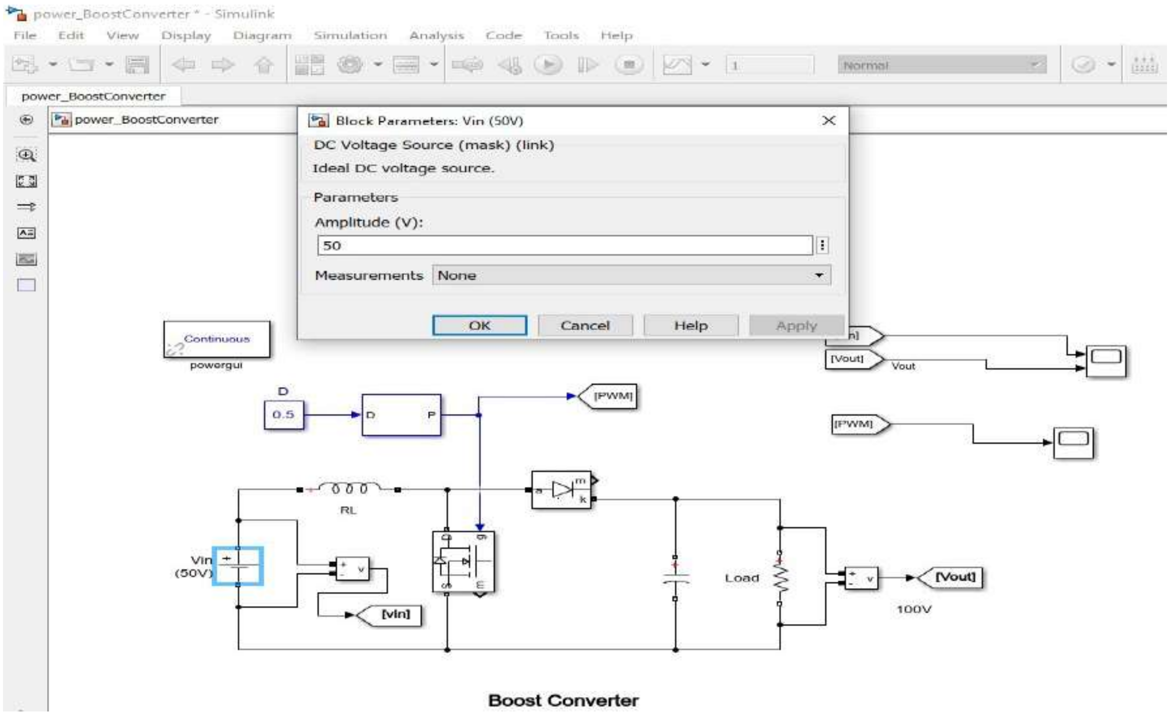


Figure 0-10 Exemple

Before the circuit can be simulated, the operational settings for the various blocks must be established.

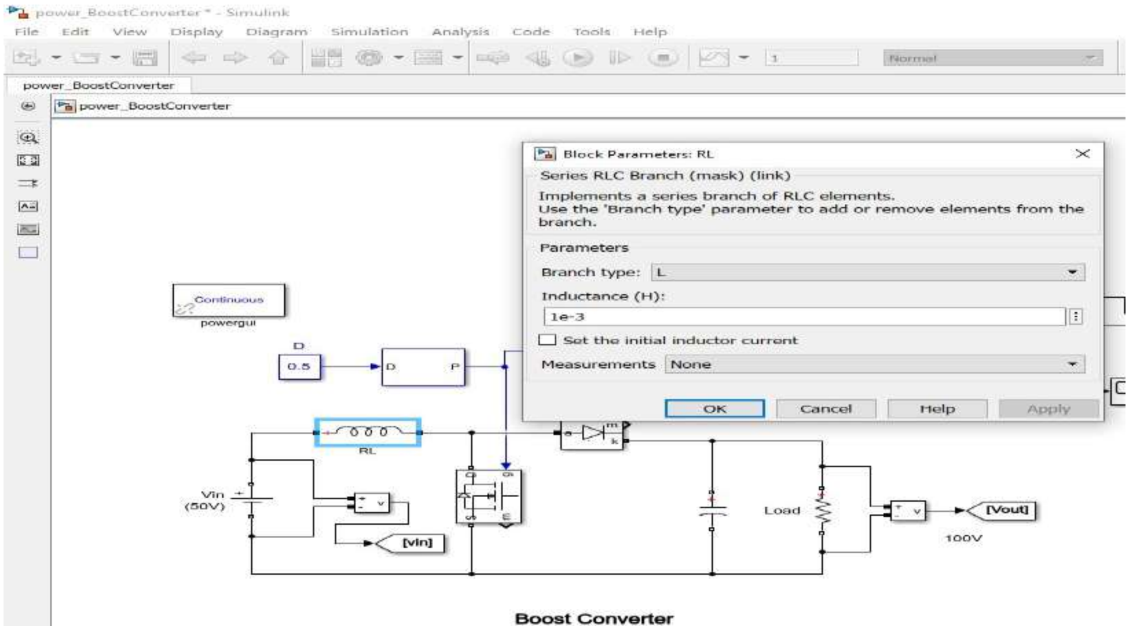
1. **The input volateg block :** provide the subsequent settings in the dialog window that appears after clicking on the component: Amplitude (v)=50 v.


Block Parameters: Vin (50V)
DC Voltage Source (mask) (link)
Ideal DC voltage source.
Parameters
Amplitude (V):
50
Measurements None

OK
Cancel
Help
Apply
power_BoostConverter * - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
Continuous
powergui
Vin
(50V)
Load
[Vout]
100V
Boost Converter

Figure 0-11 DC block edit

2. **Inductor, Diode, capacitor and resistive load** . Enter the subsequent settings in the dialog box which displays when clicking on the component : inductor L= 1mH, , Capacitance = 200e-6 and Resistance = 2 Ohms.


power_BoostConverter * - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
Normal
power_BoostConverter
Continuous
powergui
D
0.5
RL
Vin
(50V)
[Vin]
Load
[Vout]
100V
Boost Converter

Block Parameters: RL

Series RLC Branch (mask) (link)

Implements a series branch of RLC elements.
Use the 'Branch type' parameter to add or remove elements from the branch.

Parameters

Branch type: L

Inductance (H):

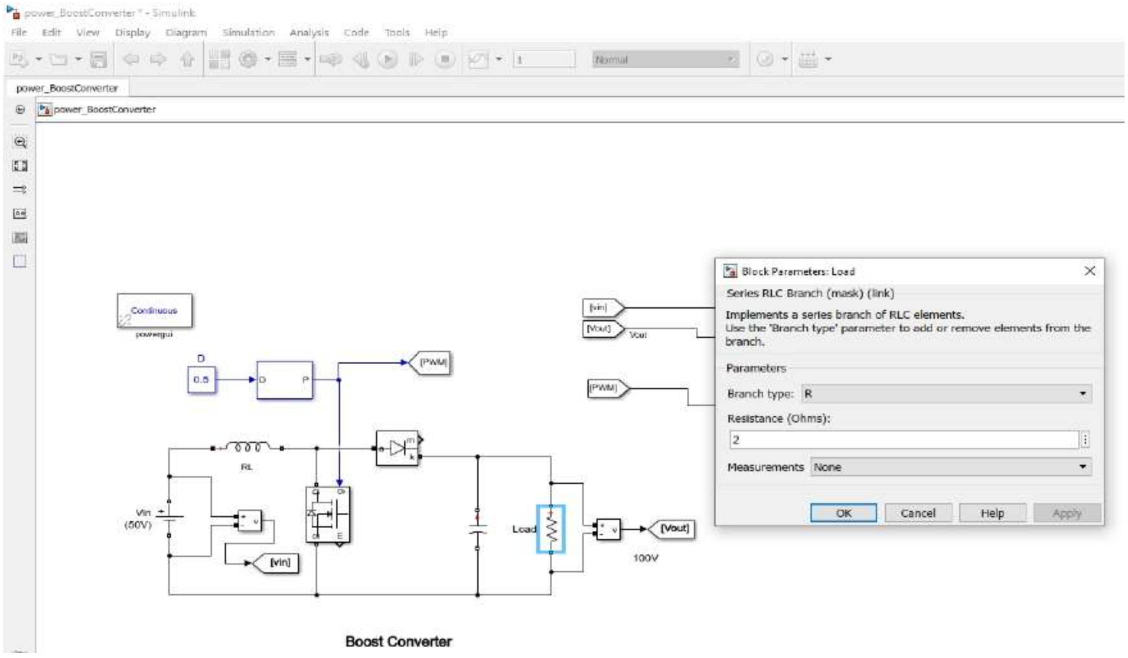
1e-3

☐ Set the initial inductor current

Measurements None

OK Cancel Help Apply

Figure 0-12 RL block Edit



Block Parameters: Load

Series RLC Branch (mask) (link)

Implements a series branch of RLC elements.
Use the 'Branch type' parameter to add or remove elements from the branch.

Parameters

Branch type: R

Resistance (Ohms):

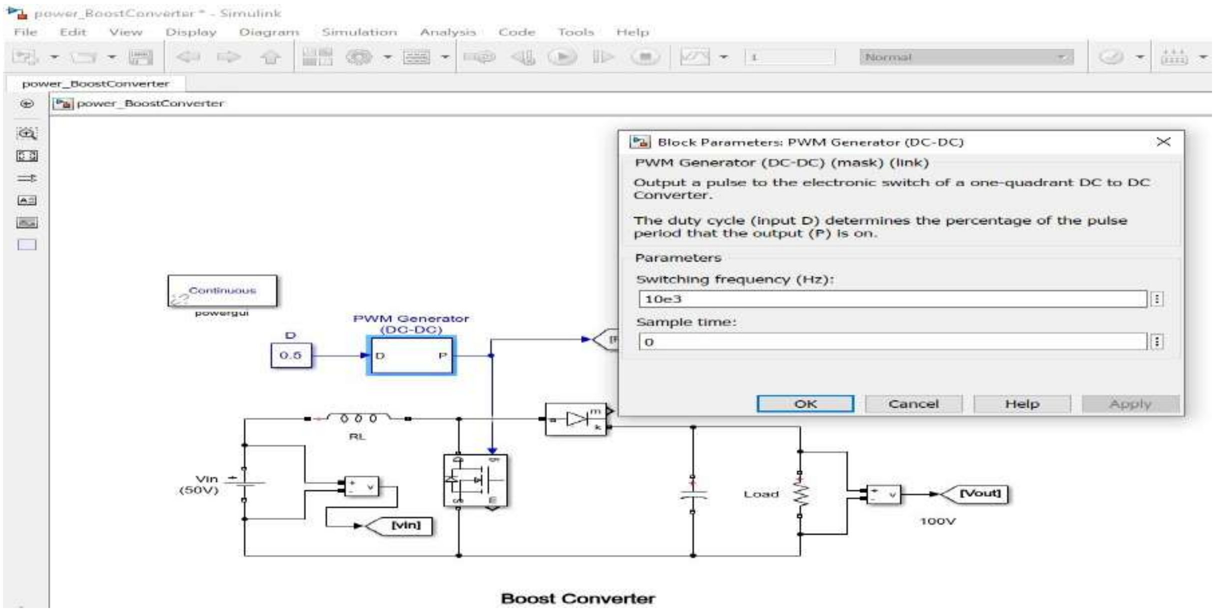
2

Measurements None

OK Cancel Help Apply

Figure 0-13 RLC block Edit

3. **PWM block**. Double-click on this block to determine the switching frequancy.



Block Parameters: PWM Generator (DC-DC)

PWM Generator (DC-DC) (mask) (link)

Output a pulse to the electronic switch of a one-quadrant DC to DC Converter.

The duty cycle (input D) determines the percentage of the pulse period that the output (P) is on.

Parameters

Switching frequency (Hz): 10e3

Sample time: 0

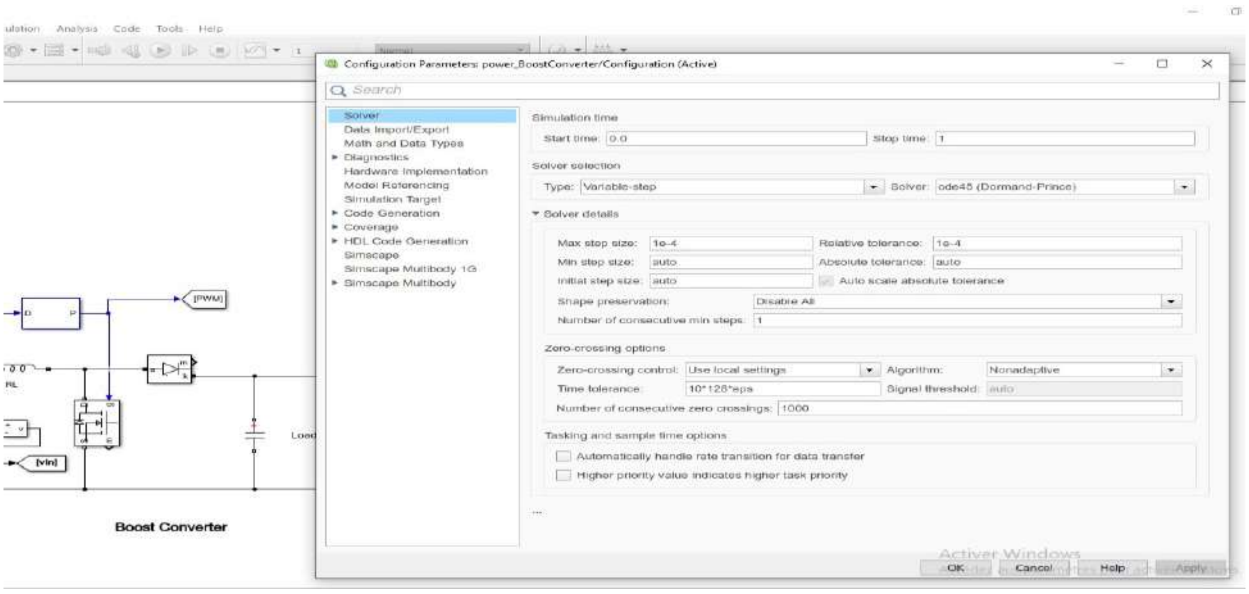
Figure 0-14 PWM block Edit

4. **Set the simulation parameter.** Setting the simulation parameters in Simulink is essential for controlling various aspects of the simulation, such as its duration, solver type, and step size.

- Click the "Simulation" menu at the top of the Simulink model window.
- Dropdown menu: "Model Configuration Parameters" This will open a new window with a list of simulation settings to which you may make changes.

Set the following basic parameters:

- **Solver:** In the "Solver" pane, you may choose the appropriate solver (e.g., 'ode45', 'ode23', etc.) for your model.
- **Start and Stop Time:** Set the beginning and ending times of your simulation.
- **Optional solvers:** You can choose between "Fixed-step" and "Variable-step" depending on your needs. Variable-step solvers vary the time step based on model dynamics, whereas fixed-step solutions employ a constant time step.


Configuration Parameters: power_BoostConverter/Configuration (Active)

Simulation time
Start time: 0.0   Stop time: 1

Solver selection
Type: Variable-step   Solver: ode45 (Dormand-Prince)

Solver details
Max step size: 1e-4   Relative tolerance: 1e-4
Min step size: auto   Absolute tolerance: auto
Initial step size: auto   Auto scale absolute tolerance
Shape preservation: Disable All
Number of consecutive min steps: 1

Zero-crossing options
Zero-crossing control: Use local settings   Algorithm: Nonadaptive
Time tolerance: 10*128*eps   Signal threshold: auto
Number of consecutive zero crossings: 1000

Tasking and sample time options
Automatically handle rate transition for data transfer
Higher priority value indicates higher task priority

Figure 6. 15 simulation parameter

5. **Star the simulation.** Set the time of simulation to 1 seconde and clique the star button. After the simulation is finished, the results are available by 1.) Reading the results provided by the Scopes components and 2.) Using the GUI output tools. To view the output, click on the Scope blocks.

The first scope on can show the output voltage in blue and the input voltage in yelow by analysing this figure we observe that the output voltage is boosted twice comparing with the input voltage becaus D=0.5

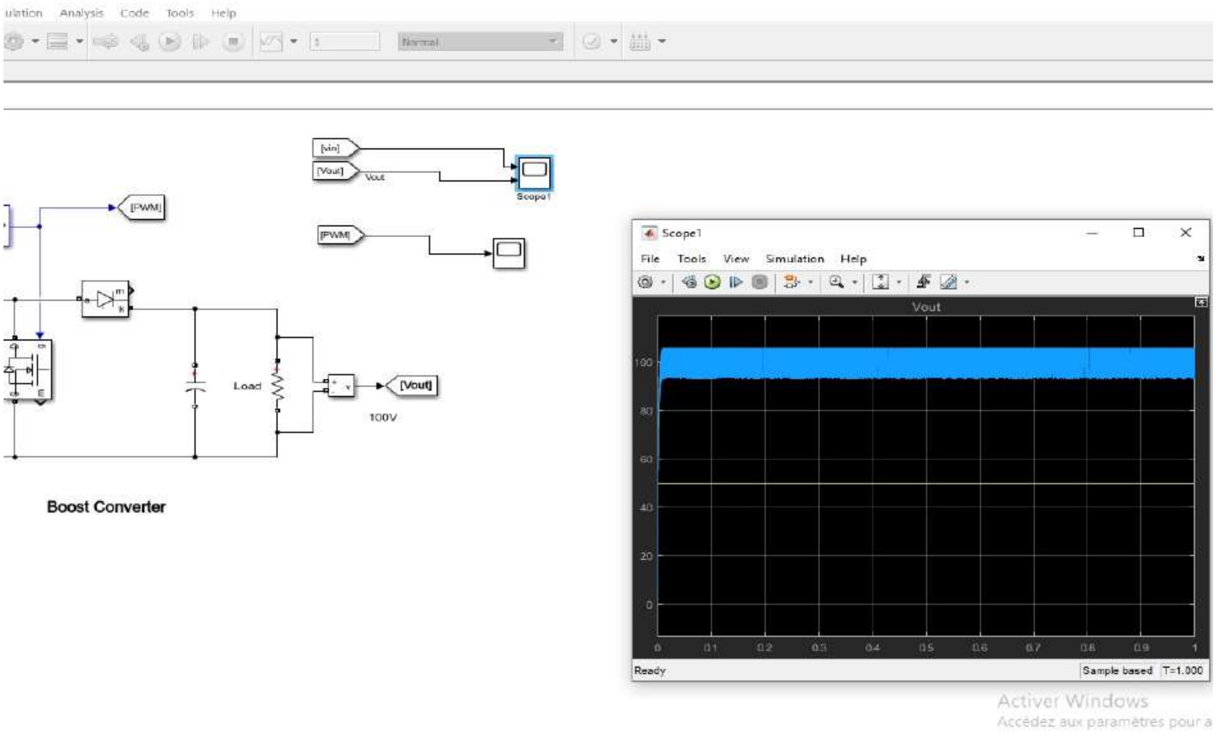


Figure 6. 16 Scope 1

The seconde scope the represent the PWM signal

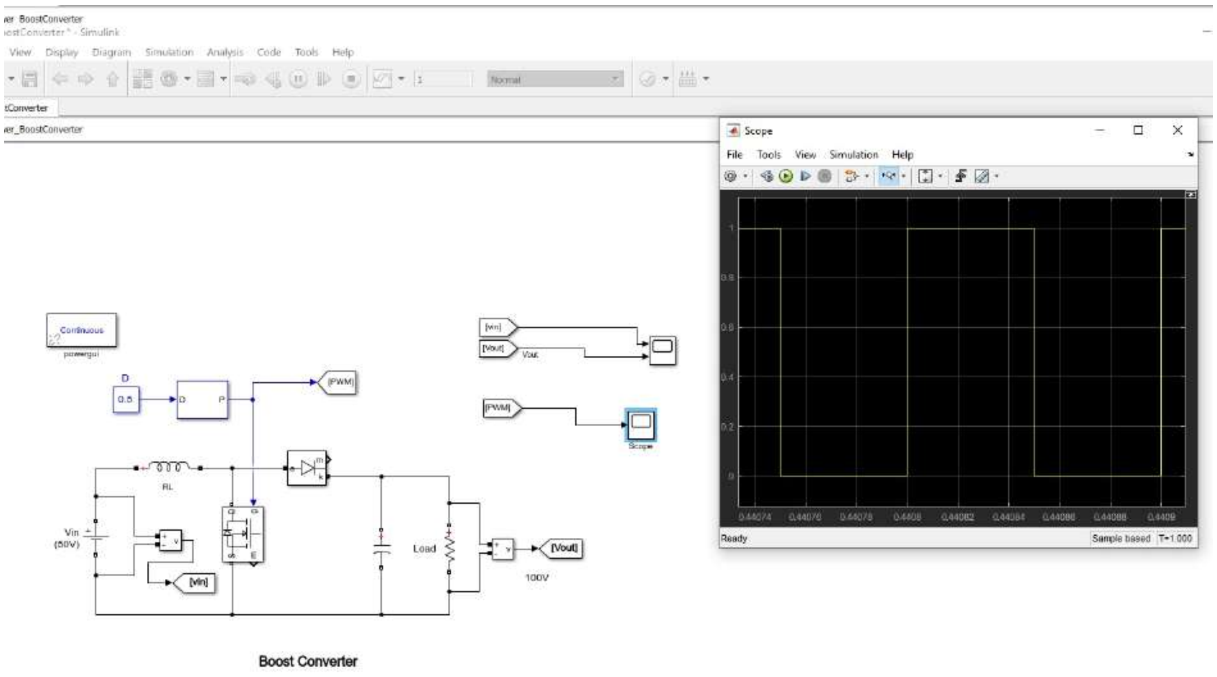


Figure 6. 16 Scope 2

6. **Printing and storing the result of the simulation.** There are numerous ways to save simulation findings and include them as part of a written document. The immediate printing of waveform traces and plots from the Multimeter and Scope blocks is possible. Result can also be saved using to workspace Block and trace them using the plot instruction.

Chapter 7
Simulation and co-simulation with other software

There is a range of simulators that can simulate different types of electrical circuits. This offering mainly covers the simulation of electronic circuits, both analog and digital or mixed. There are also software whose purpose is more educational, easy to use, but not of obvious scientific interest

Here are some of the most commonly used electrical circuit simulators:

- PSIM is a simulation program created primarily for power electronics and motor control. Power electronics experts and academics like it because of its powerful performance and simple to use layout.
- Linear Technology's LTSpice is a high-performance SPICE simulator software. It is open source and extensively adopted by academics and industry.

- PSpice: Among the most widely utilized electrical simulation tools, PSpice is provided by Cadence Design Systems. It offers several circuit simulation possibilities.

- Multisim: Multisim is a complete circuit design and evaluation program developed by National Instruments that combines an intuitive GUI with SPICE simulation.

- HOMER stands for Hybrid Optimization of Multiple Energy Resources in the framework of renewable energy. It is a computer-based tool for optimizing microgrid design in various industries, ranging from local power systems to bigger, multi-megawatt industrial power systems.

- Electronic Workbench: Also known as Electronics Workbench or simply Workbench, it served as a popular software tool for designing and evaluating electronic circuits. Its capabilities have grown through time, and it is now known as Multisim, a component of the NI (National Instruments) suite.

- OrCAD: It is a powerful program that can do whatever from schematic drafting to circuit analysis and PCB design.

- HSPICE: One of the industry standards for SPICE simulation, it is mostly used for semiconductor device simulations.

- ISIS Proteus is a software application used for the simulation of microcontrollers, microprocessors, and various other electronic components. The software is a product of Labcenter Electronics.

7.1 Simulation using PSim and co-simulation Simulink-PSim

PSIM (Power Simulation) is an application for simulation which is widely used in the disciplines of power electronics, motor control, and renewable energy. PSIM offers various benefits such :

- Efficiency: PSIM has the ability of quick simulation. Power electronics systems might be complicated, but PSIM ensures that simulations are completed on time.

- PSIM provides a wide range of modules to meet specific demands, including the Motor Drive Module for motor control applications, the Digital Control Module for digital control design, the Renewable Energy Module for solar and wind energy systems, and many more.

- PSIM can additionally be used in combination with well-known control design tools such as MATLAB/Simulink. This implies that designers may incorporate their MATLAB/Simulink control algorithms into PSIM for a more comprehensive simulation.

- Extensive Component Library: The program has a vast component library that includes a variety of power electronic devices, magnetic components, control blocks, sources, and loads.

- PSIM is often employed for real-time simulation and hardware-in-the-loop (HIL) validation with the necessary modules and tools, making it useful for practical implementation of control techniques without putting actual hardware at risk.

The nexet figure represent the PSIM interface

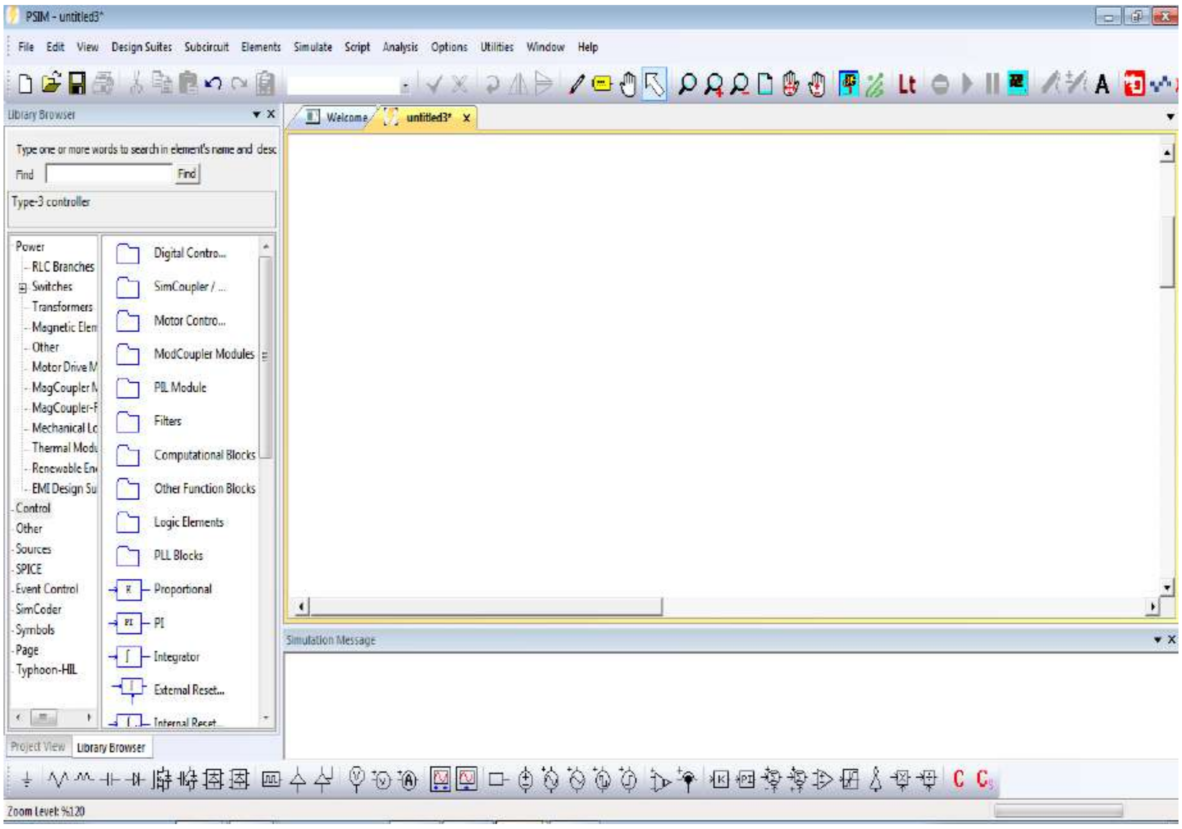


Figure 7. 1 PSIM interface

To oppen new schematic select file / New

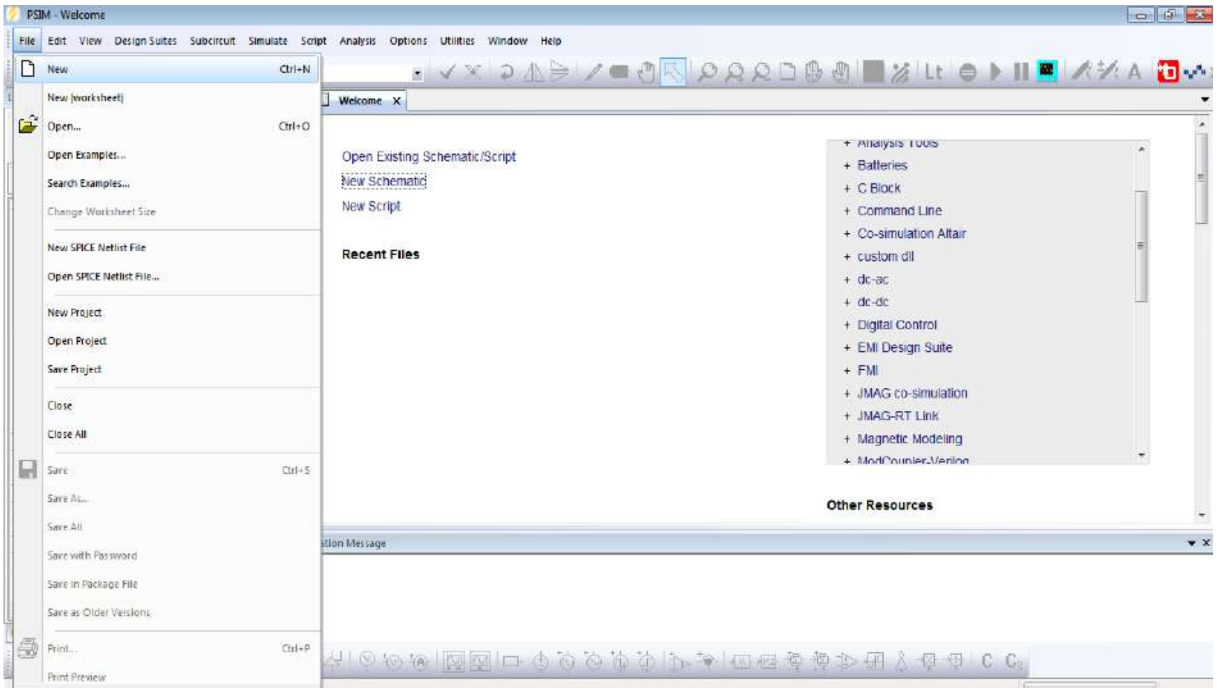


Figure 7. 2 PSIM new schematic

The PSIM menu bar may have categories such as:

- File: This menu has options for creating a new project, opening an existing one, saving, saving as, printing, and exiting the software.

- Undo, redo, cut, copy, paste, and preferences/settings are examples of common operations.

- Zooming in/out, revealing the grid, toggling toolbars, and other visualization settings are examples of view options.

- Insert: This function in design software may allow you to add certain features or components to your workspace.

- Tools: Software-provided tools or utilities for simulation, analysis, or other specialized functions.

- Simulation: This includes instructions for starting, pausing, and stopping simulations, as well as accessing simulation settings and parameters.

- Toolbars: menus with the main shortcuts for a specific activity.

- Workspace: area for the user to insert the project circuits.

- Status Bar: project status bar.

- Scroll Bar: vertical/horizontal scroll bar of the workspace.

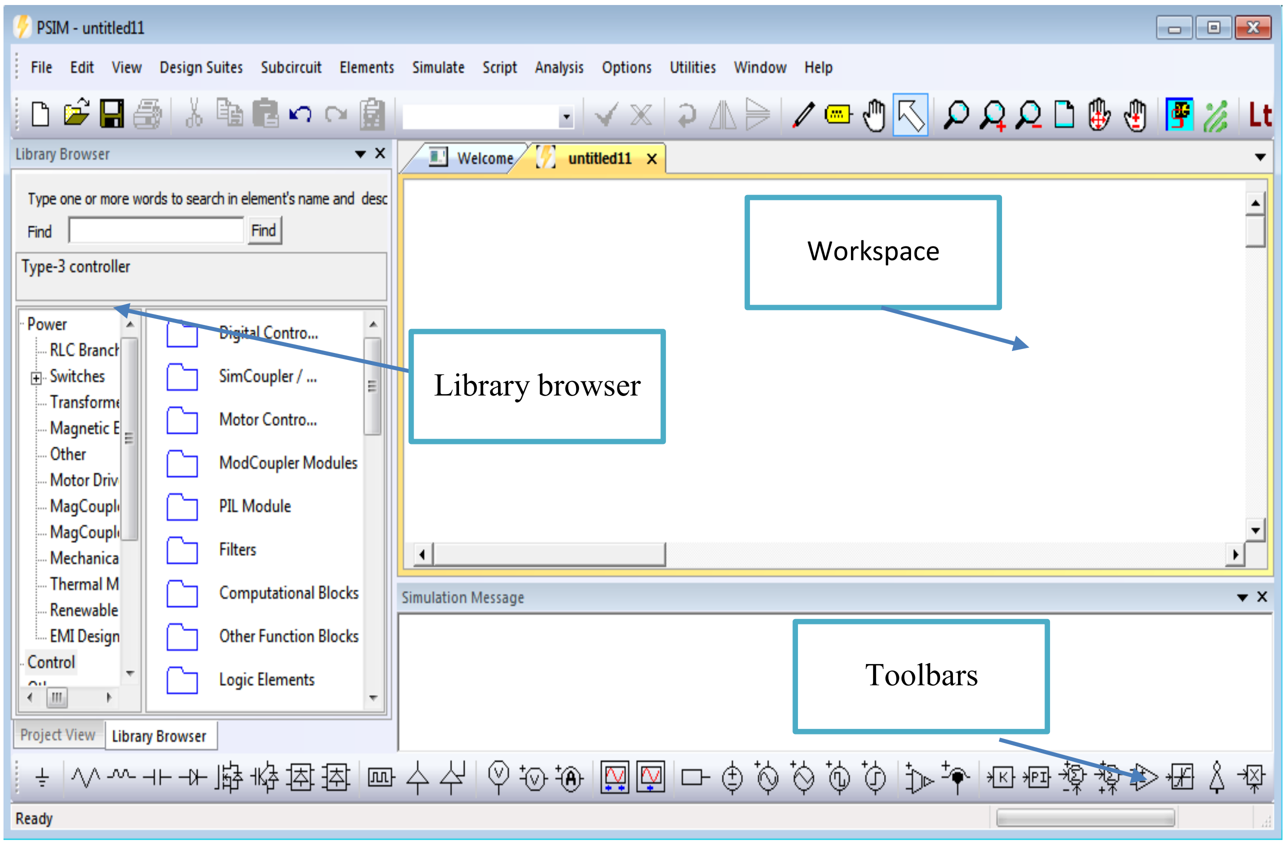


Figure 7. 3 PSIM principale workspace

The elements menu contains all of the components. Investigate it for yourself.

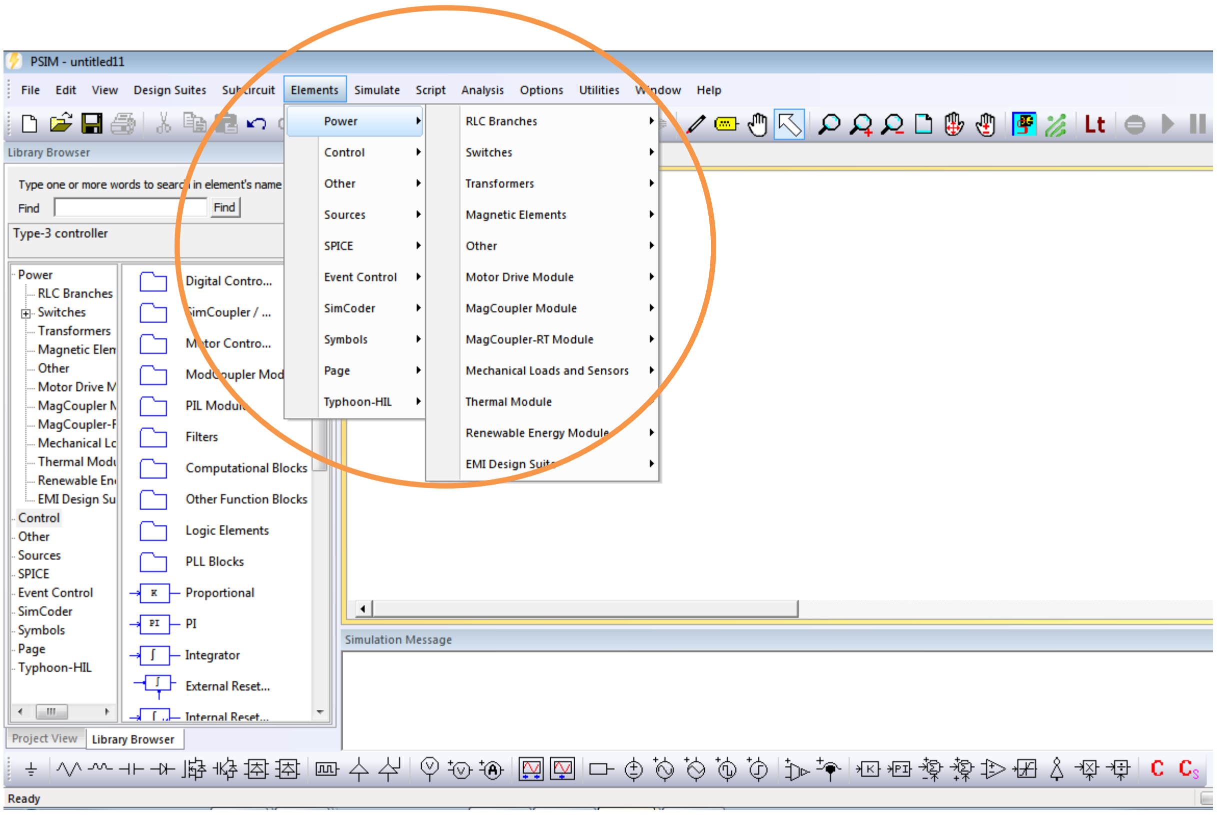


Figure 7. 4 PSIM element menu

A toolbar at the bottom of the PSIM screen displays an array of the majority of frequently utilized components in circuit schematics, making it easier to add them to the workspace. Figure bellow shows a list of the components.

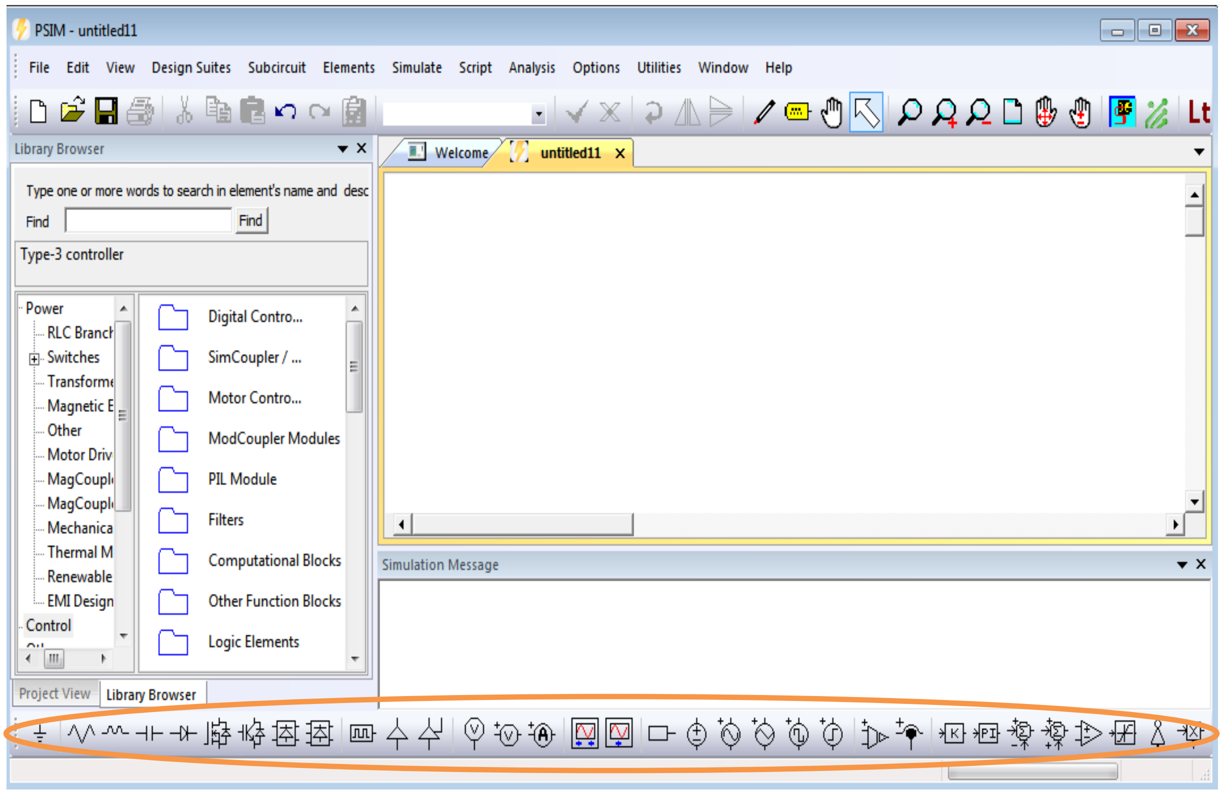


Figure 7. 5 PSIM component toolbar

7.1.1 Exemple of simulation with PSIM

Figure depicts a two-phase interleaved boost converter. The structure contains two parallel converter circuits. The first leg is made up of inductor L, Transistor T, and diode D, whereas the second leg is made up of the same configuration as the fisrt leg. The two converter circuits are practically in parallel, however they function in an interleaved method. At the output, they shared a common output capacitor C. The specifications of the two legs are considered to be identical. In an interleaving arrangement, the gating signals T1 and T2 for the two transistor identical but shifted by 360°/2=180°, where 2 is the leg number coupled in parallel.

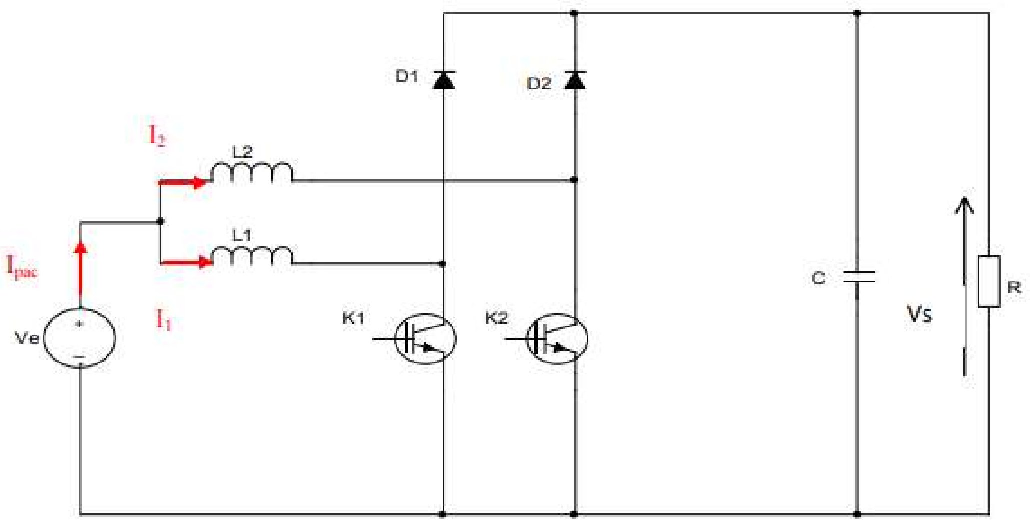


Figure 7. 6 interleaved boost

In this exemple assignment, you will build a two phases interleaved boost converter circuit in PSIM software. Figure bellow depicts the circuit. A input DC source, a diode, inductor, transistor, capacitor and a resistive (R) load comprise the circuit. The source's input voltage is 24v, and the output volatge is 40V (this model is available in the exemples directory in PSIM).

The step-by-step process for simulating a circuit can be briefly described as:

- Create the schematic in a program;
- Conduct the simulation;
- Visulate and analyze the results.

We now consider the simulation of a circuit.

Create the schematic in a program

The schematic circuit of the intereaved boost in PSIM is as follow :

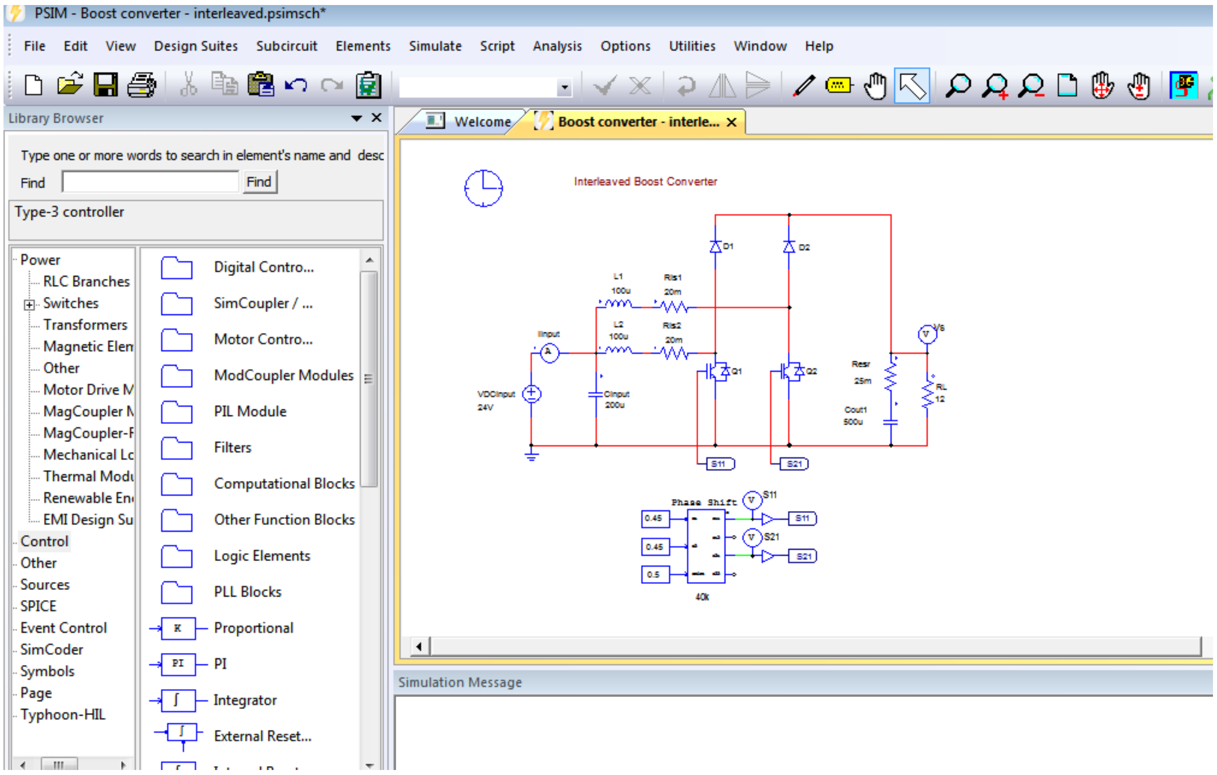
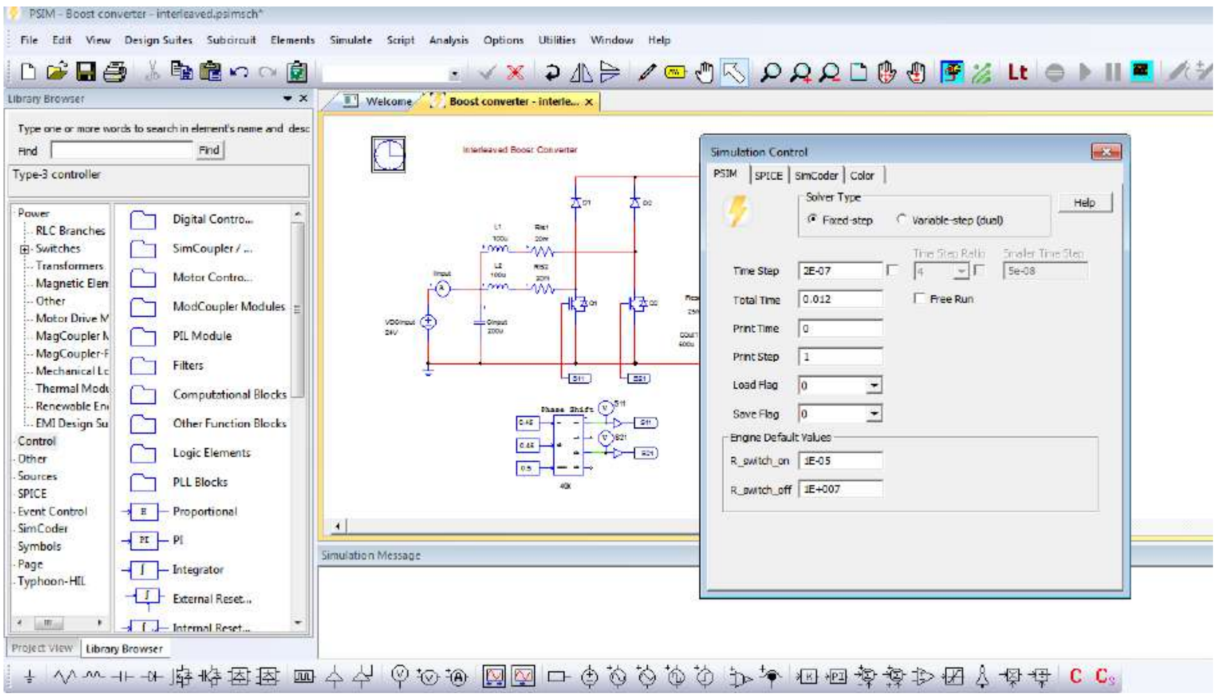


Figure 7. 7 interleaved boost in PSIM

It is important to provide the simulation settings once the file has been saved. The simulation control, which can be accessed via the Simulate Simulation Control menu, must be used in this situation.


PSIM - Boost converter - interleaved.psimsch*
File Edit View Design Suites Subcircuit Elements Simulate Script Analysis Options Utilities Window Help
Library Browser
Welcome
Boost converter - interle...
Simulation Message
Project View
Library Browser
Simulation Control
PSIM
SPICE
SimCoder
Color
Help

Solver Type: [✓ Fixed-step] Variable-step (dual)

Time Step: 2E-07
Time Step Ratio: 4
Smaller Time Step: 5e-08
Total Time: 0.012
Free Run
Print Time: 0
Print Step: 1
Load Flag: 0
Save Flag: 0

Engine Default Values
R_switch_on: 1E-05
R_switch_off: 1E+007

Figure 7. 8 Exemple simulation control

Conduct the simulation

You can begin the model simulation after configuring the simulation settings. Chose run PSIM simulation in the simulate menu at the top toolbar or press F8 Button on the keyboard. When the Auto-run option is chosen, the Simview application will launch directly after the simulation finishing.

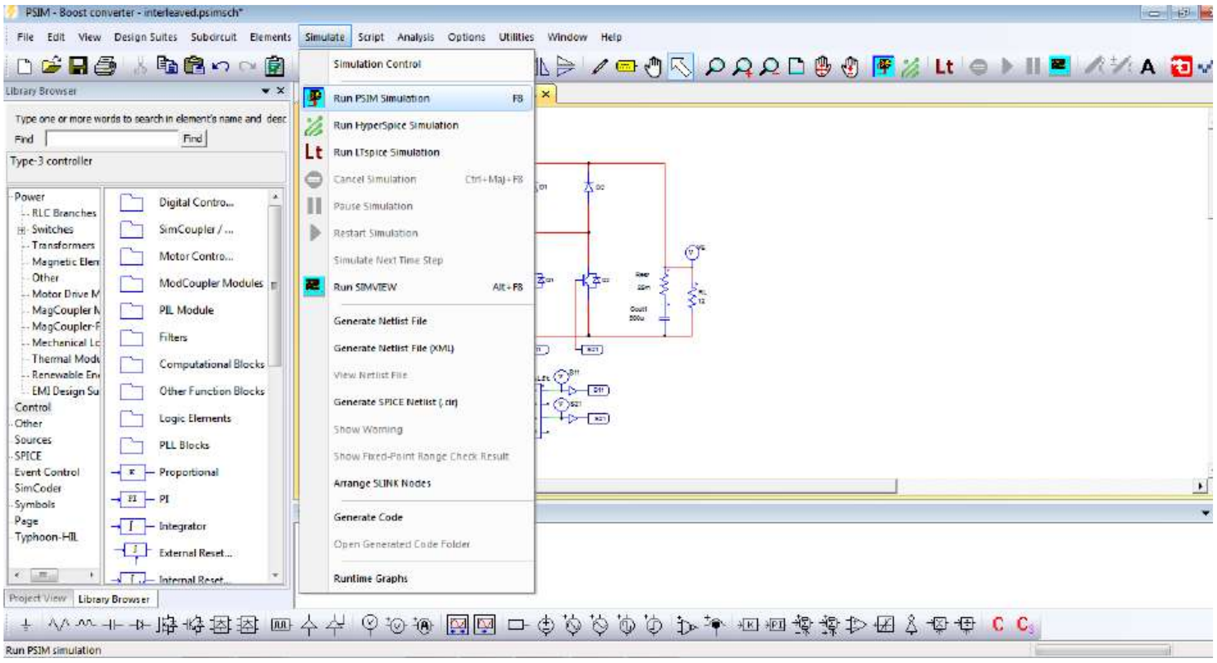

PSIM - Boost converter - interleaved.psimsch*
File Edit View Design Suites Subcircuit Elements Simulate Script Analysis Options Utilities Window Help
Simulation Control
Run PSIM Simulation F8
Run HyperSpice Simulation
Run LTspice Simulation
Cancel Simulation Ctrl+Maj+F8
Pause Simulation
Restart Simulation
Simulate Next Time Step
Run SIMVIEW Alt+F8
Generate Netlist File
Generate Netlist File (XML)
View Netlist File
Generate SPICE Netlist (.cir)
Show Warning
Show Fixed-Point Range Check Result
Arrange SLINK Nodes
Generate Code
Open Generated Code Folder
Runtime Graphs
Library Browser
Run PSIM simulation

Figure 7. 9 Run simulation

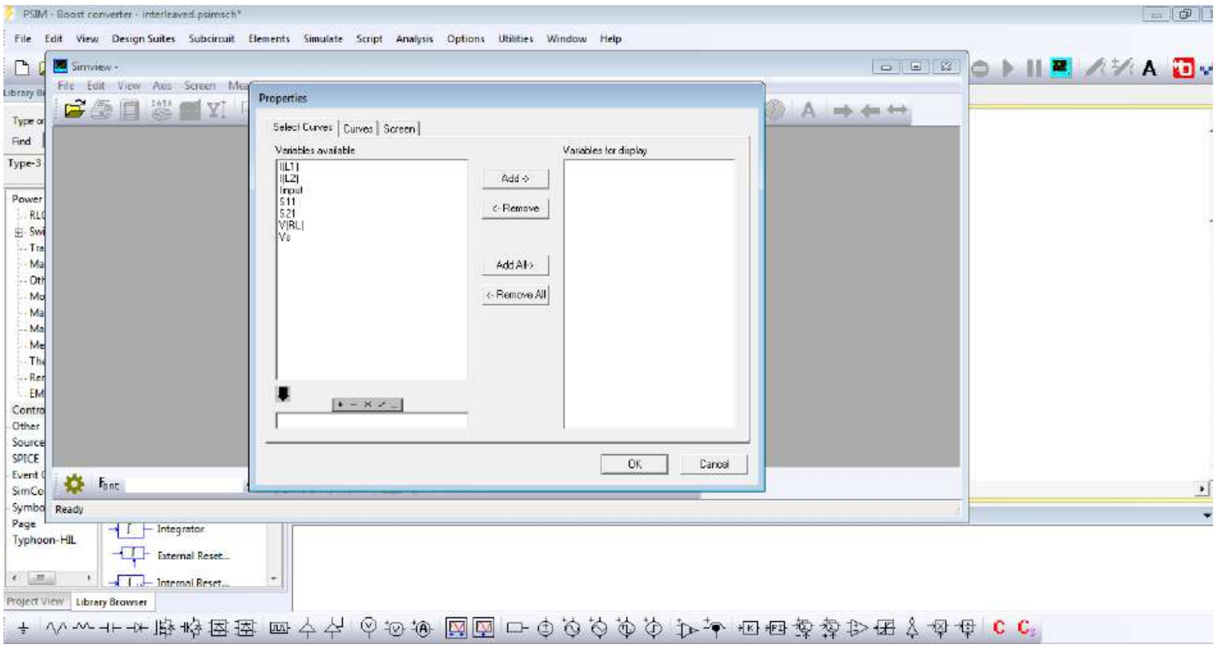


Figure 7. 10 PSIM Simview

Visulate the simulation results

After finishing the simulation, the Simview program will appear which depicts all the waveform measurements that can be performed. We can select the measurements to be plotted.

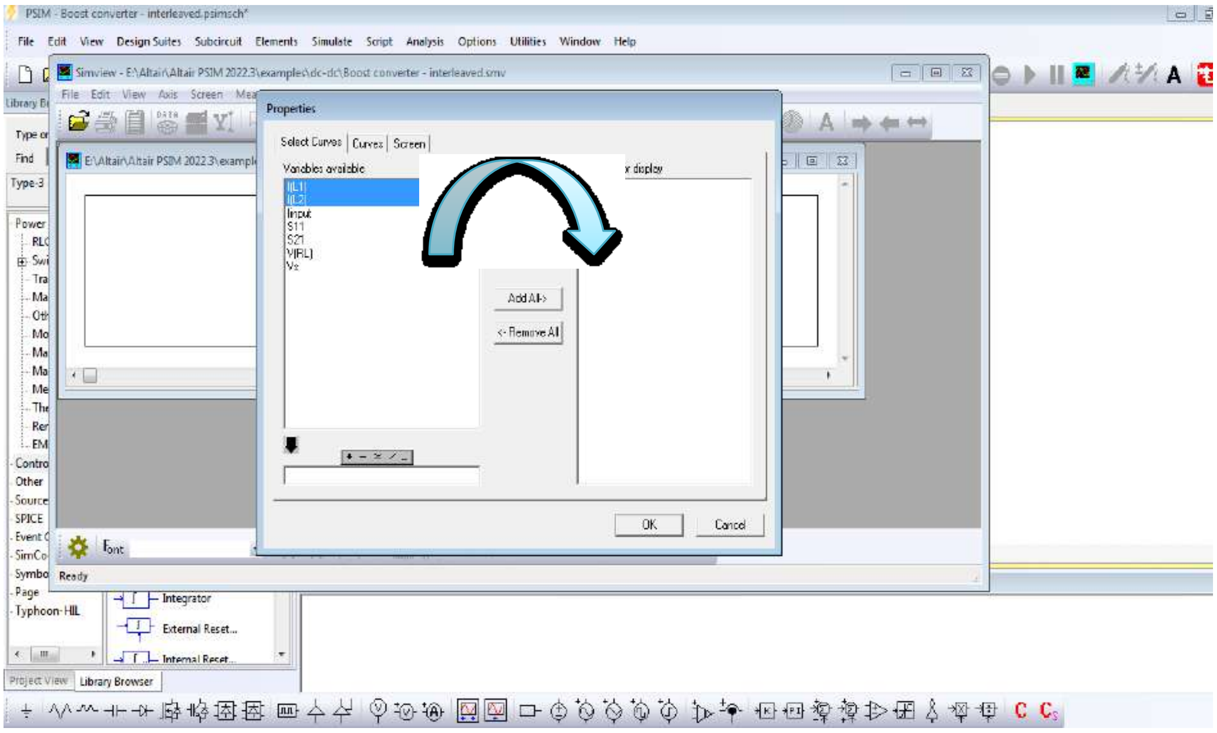

PSIM - Boost converter - interleaved.psimsch*
Simview - E:\Altair\Altair PSIM 2022.3\examples\dc-dc\Boost converter - interleaved.smv
Properties
Select Curves
Curves
Screen
Variables available
I(L1)
I(L2)
Input
S11
S21
V(RL)
Vo
Add All->
<- Remove All
OK
Cancel

Figure 7. 11 PSIM Simview properties

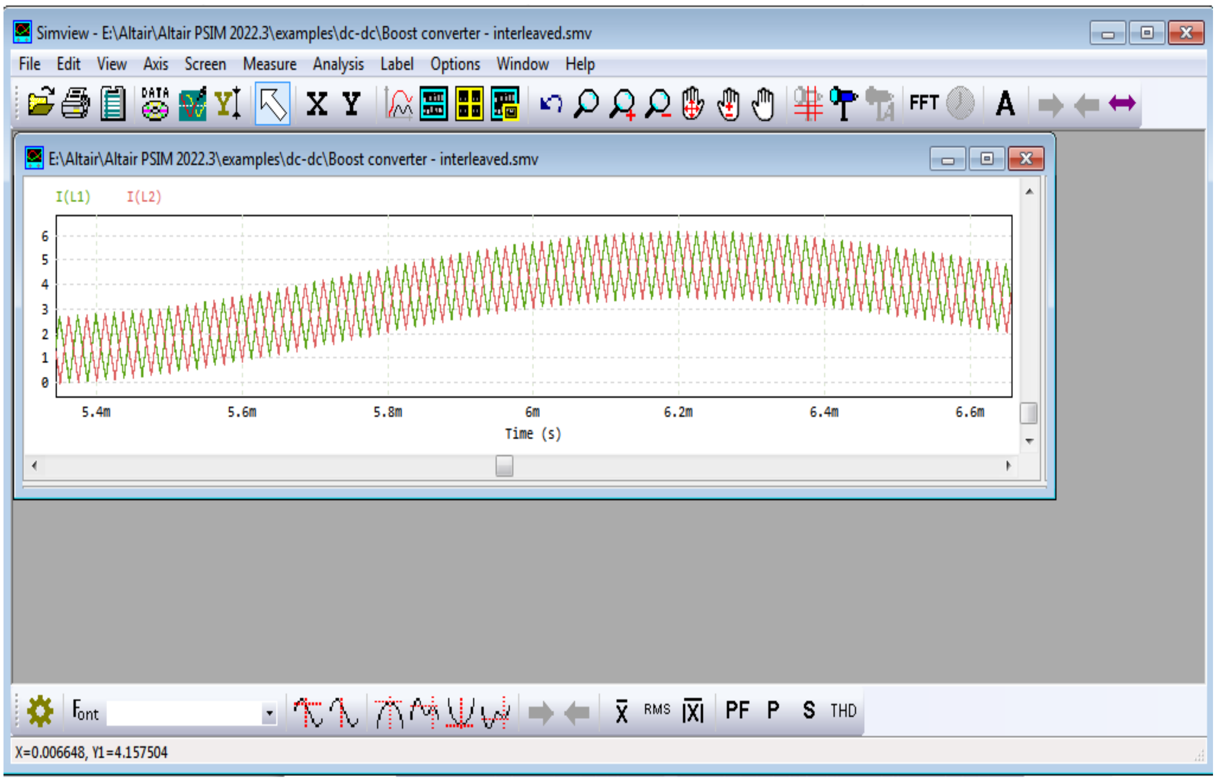

Simview - E:\Altair\Altair PSIM 2022.3\examples\dc-dc\Boost converter - interleaved.smv
File Edit View Axis Screen Measure Analysis Label Options Window Help
I(L1) I(L2)
Time (s)
X=0.006648, Y1=4.157504

Figure 7. 12 Current plot

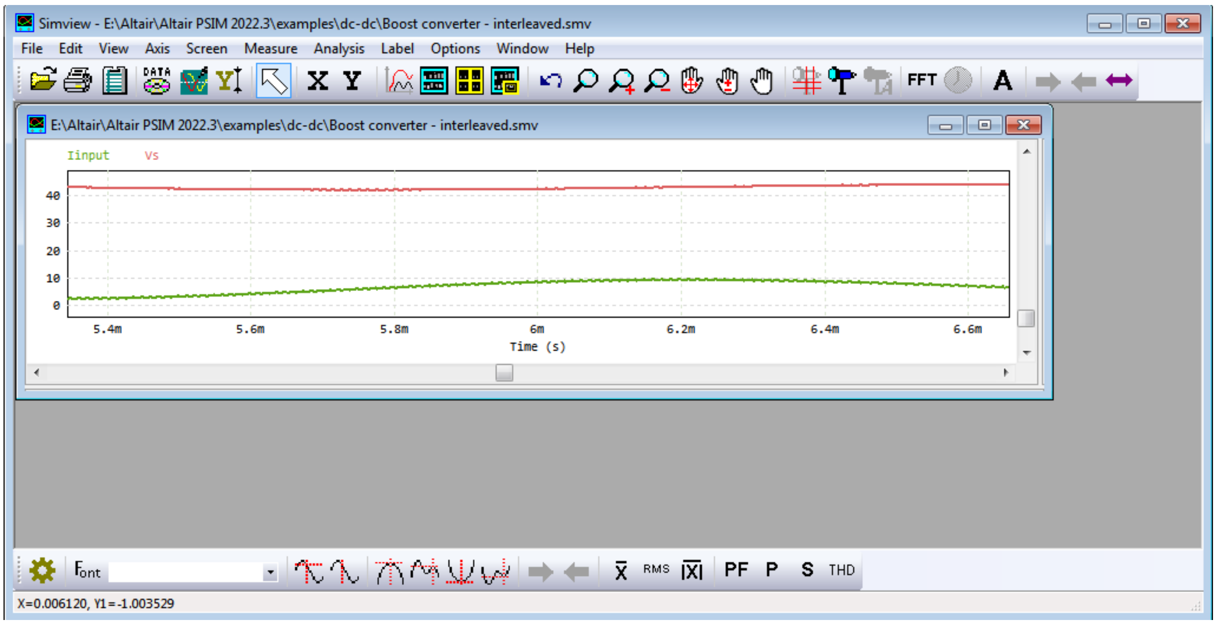

Simview - E:\Altair\Altair PSIM 2022.3\examples\dc-dc\Boost converter - interleaved.smv
File Edit View Axis Screen Measure Analysis Label Options Window Help
E:\Altair\Altair PSIM 2022.3\examples\dc-dc\Boost converter - interleaved.smv
Iinput Vs
Time (s)
Font
X=0.006120, Y1=-1.003529

Figure 7. 13 Voltage plot

To improve the visualization of the obtained results, there is the option toolbar in the up to modify the waveform.

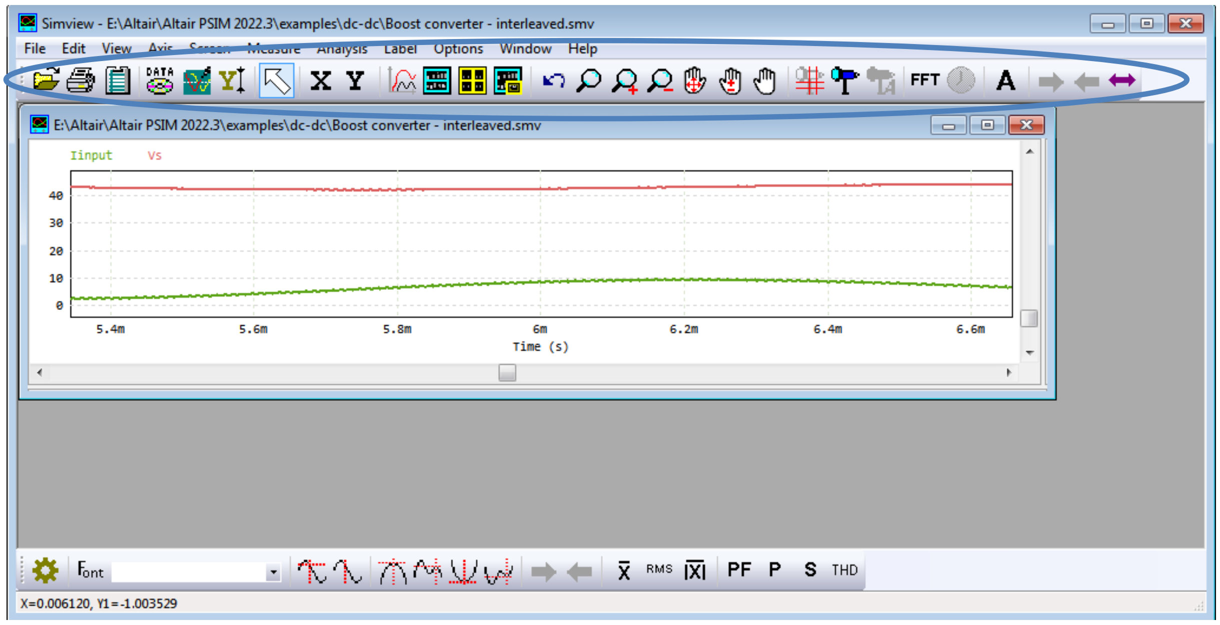

Simview - E:\Altair\Altair PSIM 2022.3\examples\dc-dc\Boost converter - interleaved.smv
File Edit View Axis Screen Measure Analysis Label Options Window Help
E:\Altair\Altair PSIM 2022.3\examples\dc-dc\Boost converter - interleaved.smv
Iinput Vs
Time (s)
Font
X=0.006120, Y1=-1.003529

Figure 7. 14 option toolbar

7.1.2 Co-Simulation with Matlab/Simulink

Co-simulation is the concurrent simulation of multiple tools in order to take use of each tool's advantages. This implies that although one component of a complex system may be simulated in one piece of software (such as PSIM), another component may be simulated in a different piece of software (such as Simulink).

Using co-simulation, one may create a control algorithm in Simulink and then examine how it impacts a power circuit in PSIM, or vice versa. This close connection guarantees that design are comprehensive and take into account interactions between system components.

PSIM can co-simulate with Matlab/Simulink. If the SimCoupler Module is activated in the licensing when PSIM is installed, the installer will automatically configure co-simulation between PSIM and Matlab/Simulink. If you wish to modify the PSIM folder name after installing PSIM, you must re-enable co-simulation by executing Utilities >> SimCoupler Setup. If you are running various versions of PSIM or Matlab/Simulink, you may associate a specific version of PSIM with a specific version of Matlab/Simulink by executing Utilities >> SimCoupler Setup from that PSIM version and selecting the relevant Matlab/Simulink version.

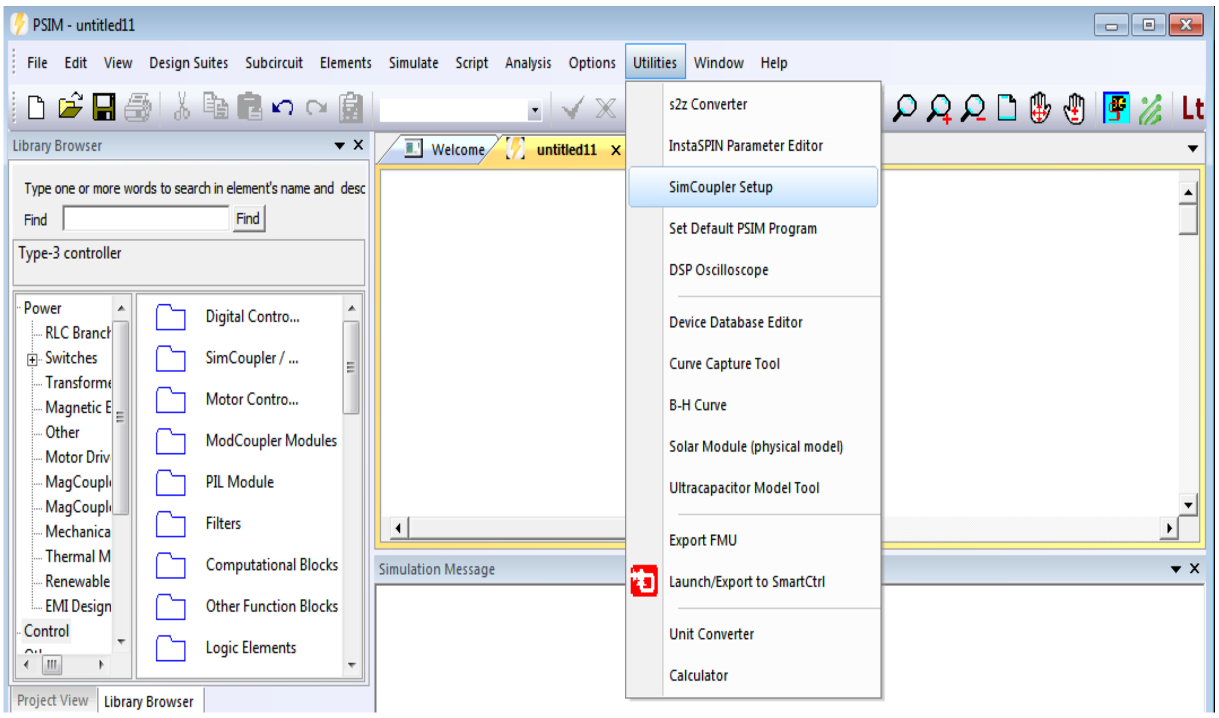
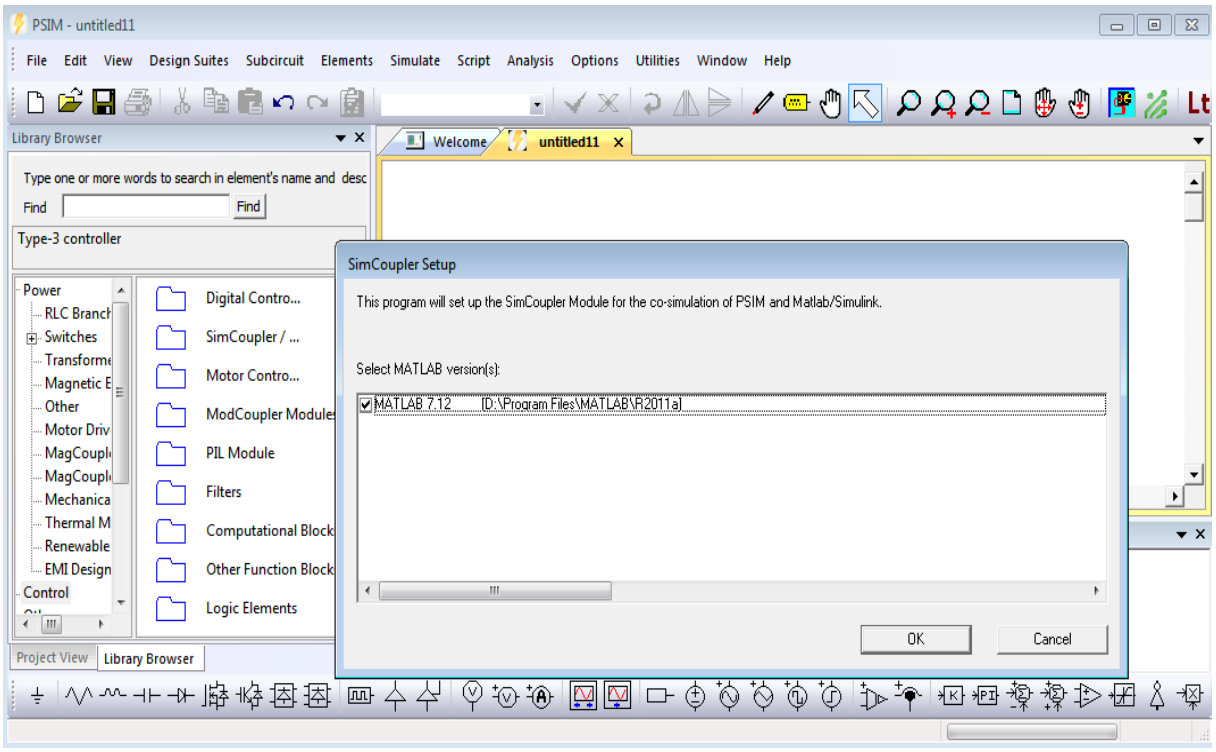


Figure 7. 15 simcoupler setup



This program will set up the SimCoupler Module for the co-simulation of PSIM and Matlab/Simulink.

Select MATLAB version(s):

☑ MATLAB 7.12 [D:\Program Files\MATLAB\R2011a]

Figure 7. 16 simcoupler setup 2

To illustrate how co-simulation works, as follows:

1. Create a simple circuit model.

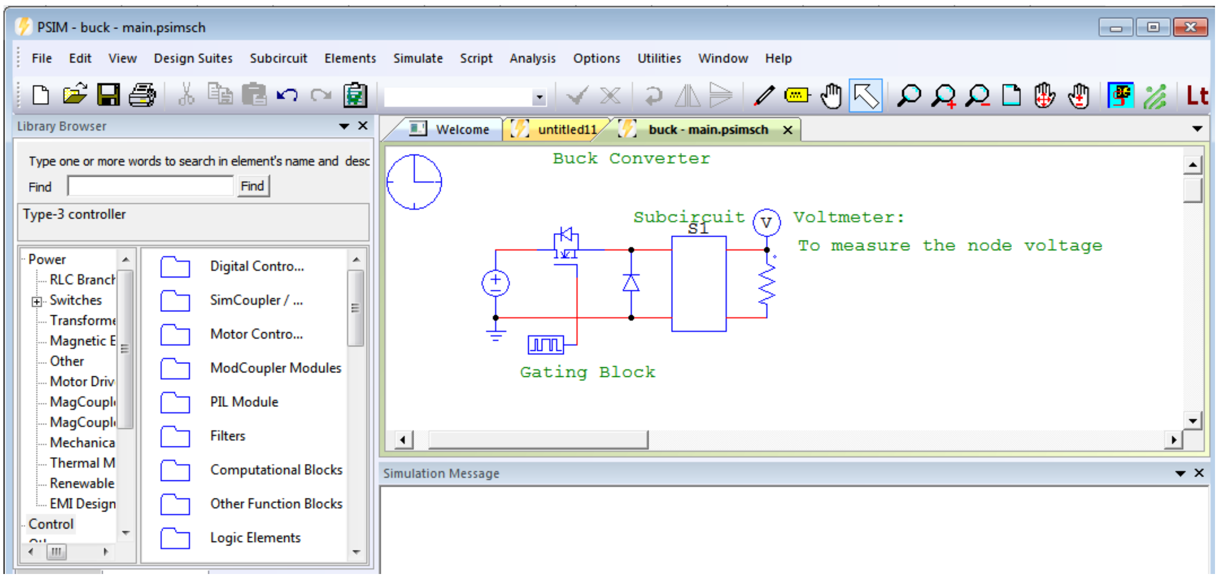

PSIM - buck - main.psimsch
File Edit View Design Suites Subcircuit Elements Simulate Script Analysis Options Utilities Window Help
Library Browser
Welcome
untitled11
buck - main.psimsch
Buck Converter
Subcircuit
S1
Voltmeter:
To measure the node voltage
Gating Block
Simulation Message

Figure 7. 17 simcoupler setup 3

2. Run Utilities >> SimCoupler Setup from PSIM. The following will set up PSIM for co-simulation. It should be noted that this only has to be done once.
3. Navigate into the "Elements/Control/SimCoupler Module/" menu.

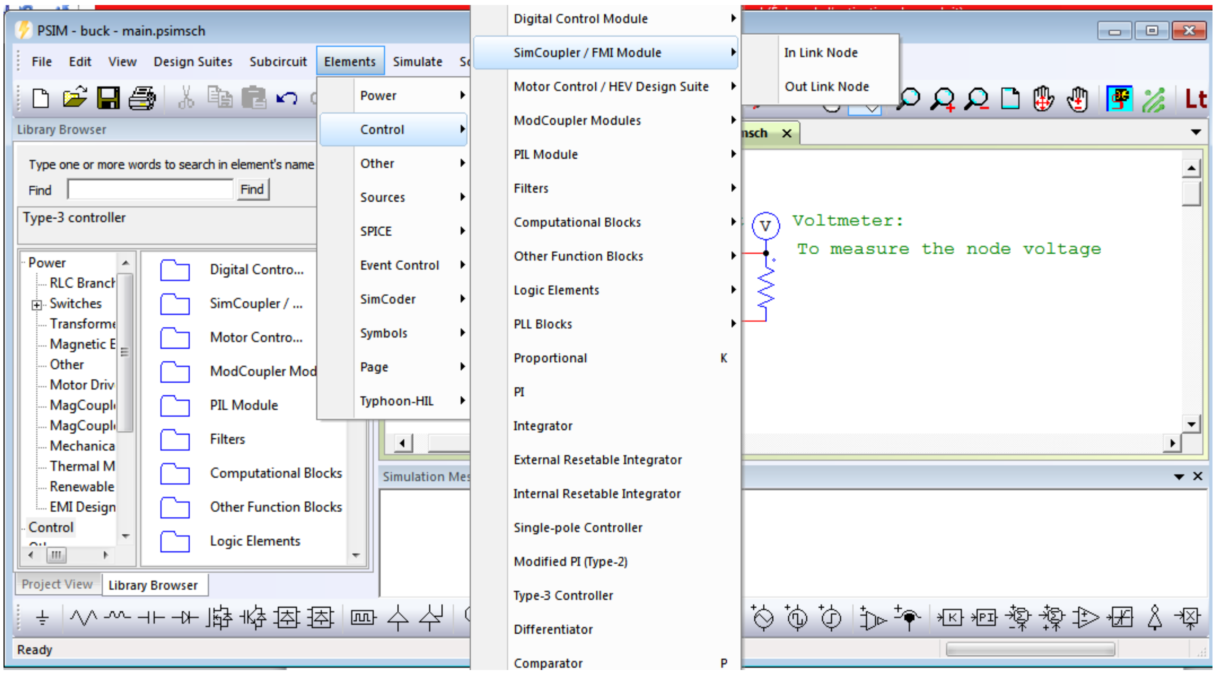


Figure 7. 18 simcoupler node menu

4. Then connect OUT SLINK and IN SLINK node to the circuit model. Each In Link Node gets a value coming from Simulink, as well as an Out Link Node transmits the value to Simulink.

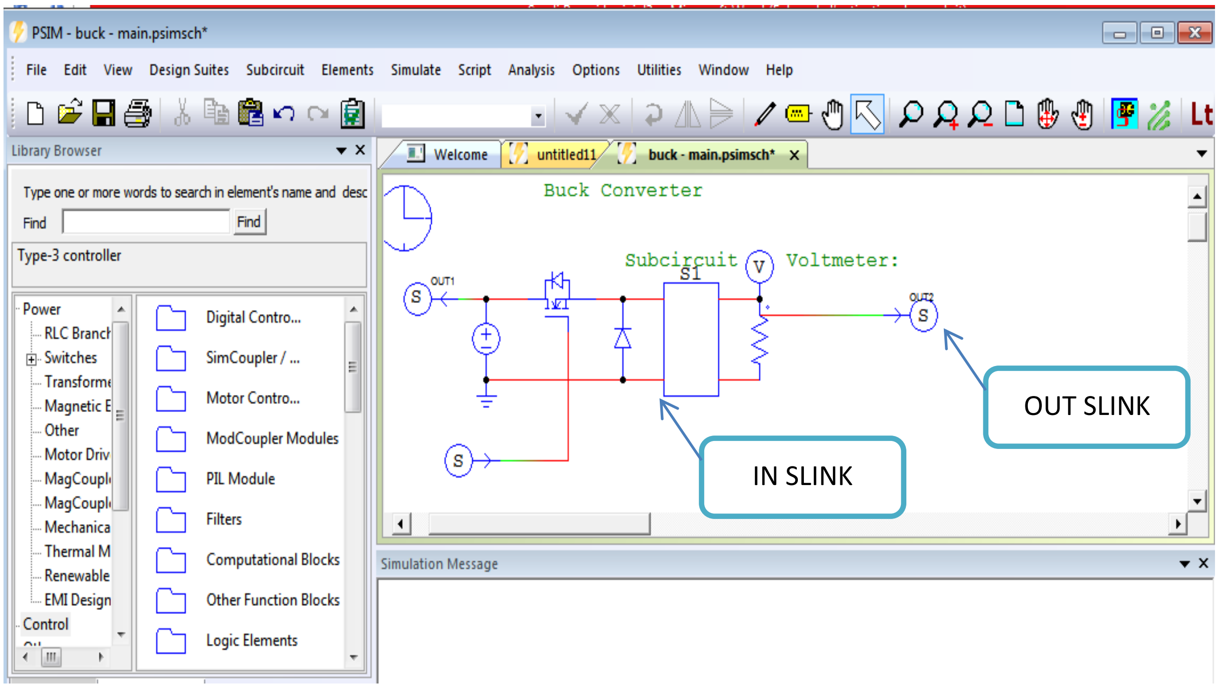


Figure 7. 19 simcoupler node placement

5. To organize the arrangement of the connecting nodes. Choose Arrange SLINK Nodes from the Simulate Menu.

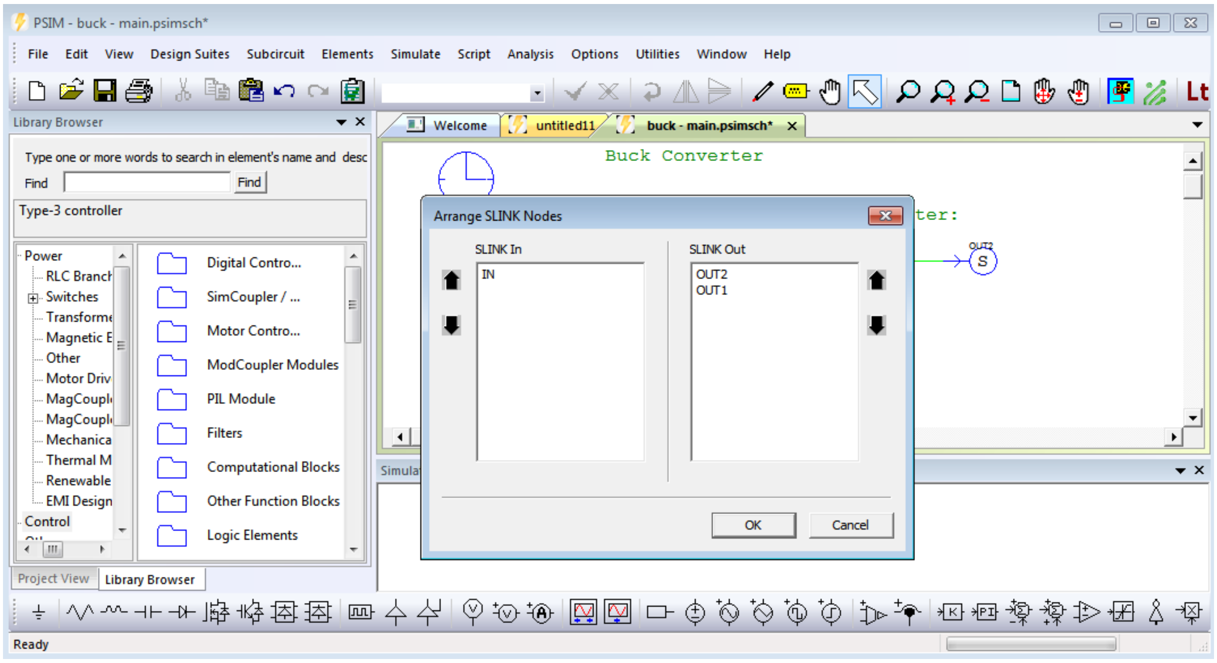


Figure 7. 20 simcoupler node placement 2

6. Choose Generate Netlist File from the Simulate Menu. A netlist file that contains the extension.cct is going to be created and archived in the same location folder that holds the circuit file.

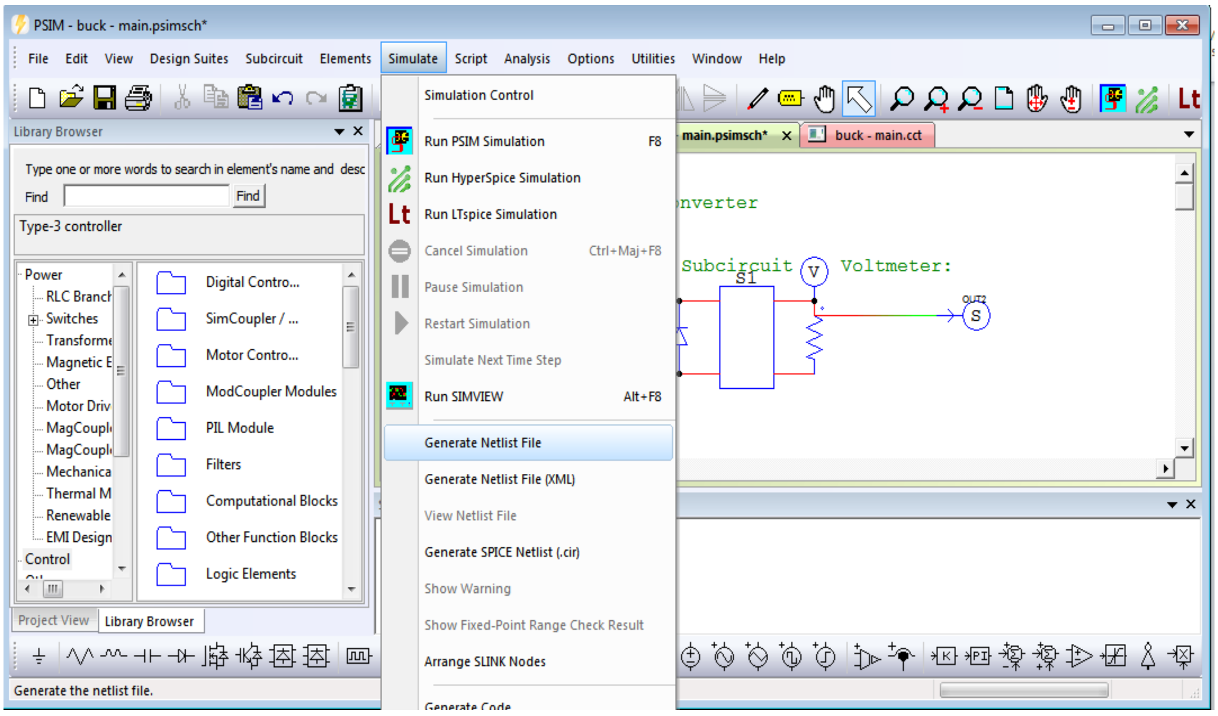

PSIM - buck - main.psimsch*
File Edit View Design Suites Subcircuit Elements Simulate Script Analysis Options Utilities Window Help
Simulation Control
Run PSIM Simulation F8
Run HyperSpice Simulation
Run LTspice Simulation
Cancel Simulation Ctrl+Maj+F8
Pause Simulation
Restart Simulation
Simulate Next Time Step
Run SIMVIEW Alt+F8
Generate Netlist File
Generate Netlist File (XML)
View Netlist File
Generate SPICE Netlist (.cir)
Show Warning
Show Fixed-Point Range Check Result
Arrange SLINK Nodes
buck - main.cct
Generate the netlist file.

Figure 7. 21 Generate Netlist file

7. Go to matlab and select the working directory the same location folder that holds the circuit file.
8. Go to Simulink and create a new file and build the control schematic that will coupled with PSIM

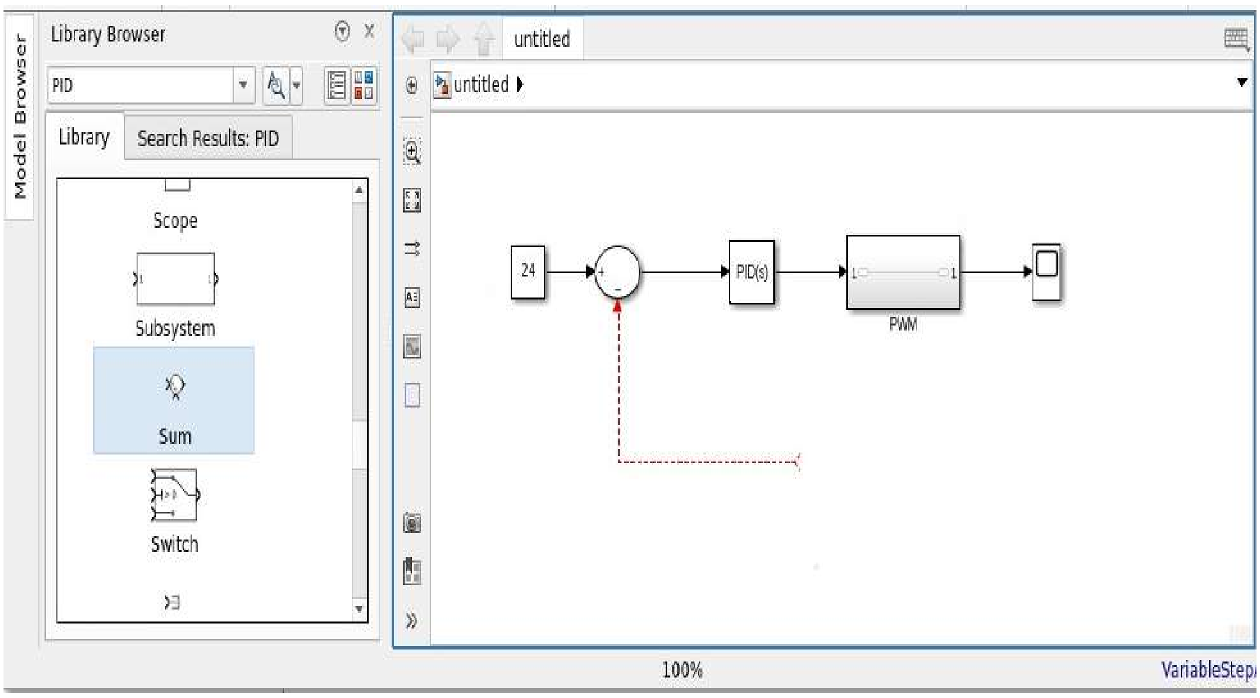


Figure 7. 22 Simulink new file

9. In Simulink, search the block "SimCoupler " in the HDL coder directory, containing the SimCoupler model block. And copier paste the SimCoupler model block into the control schematic.

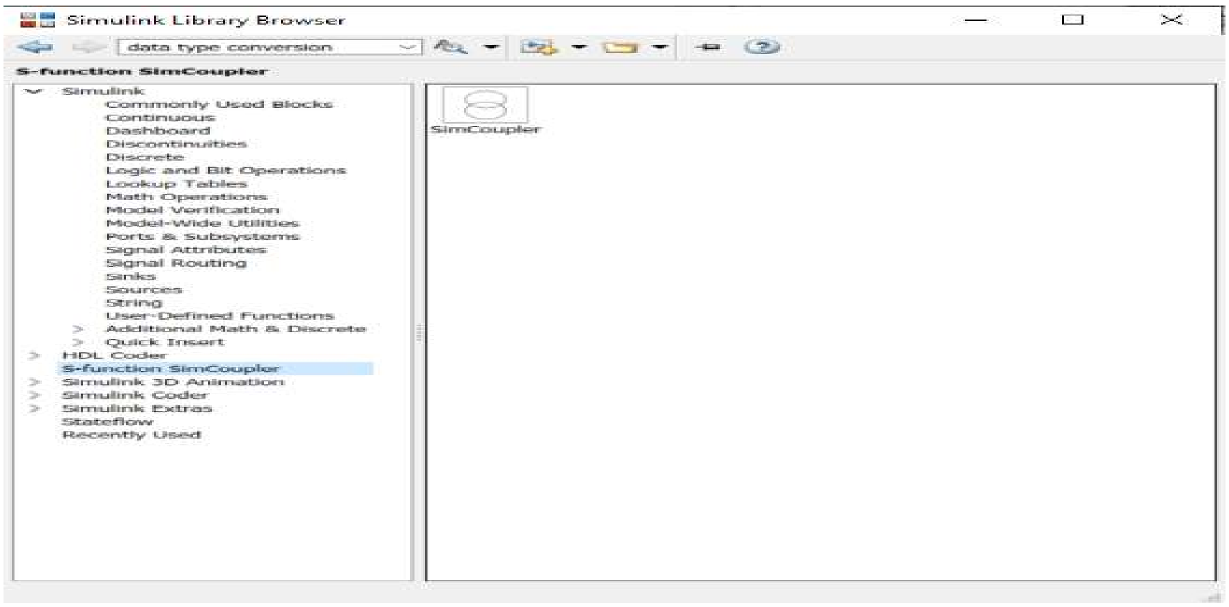


Figure 7. 23 Simulink simcoupler block browser

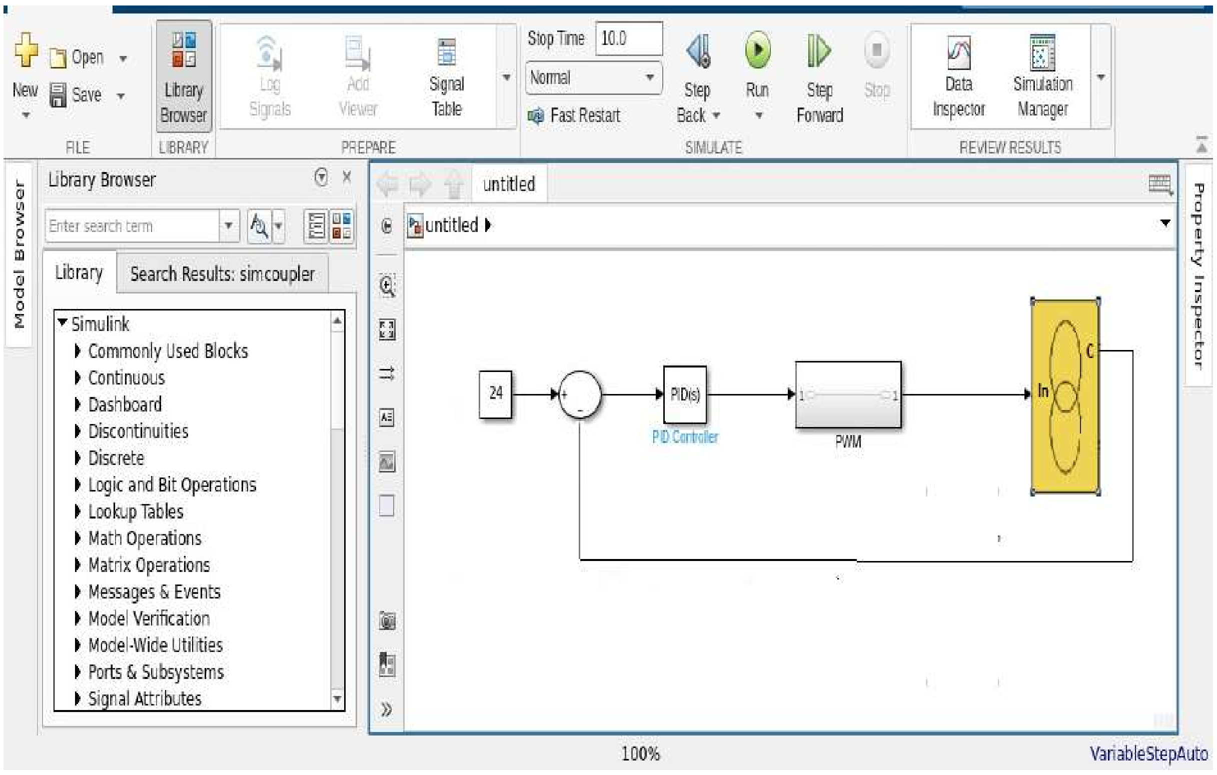


Figure 7. 24 Simulink simcoupler block

10. In the SimCoupler block, type the name and location of the PSIM netlist (file name.cct), and then select Ok.

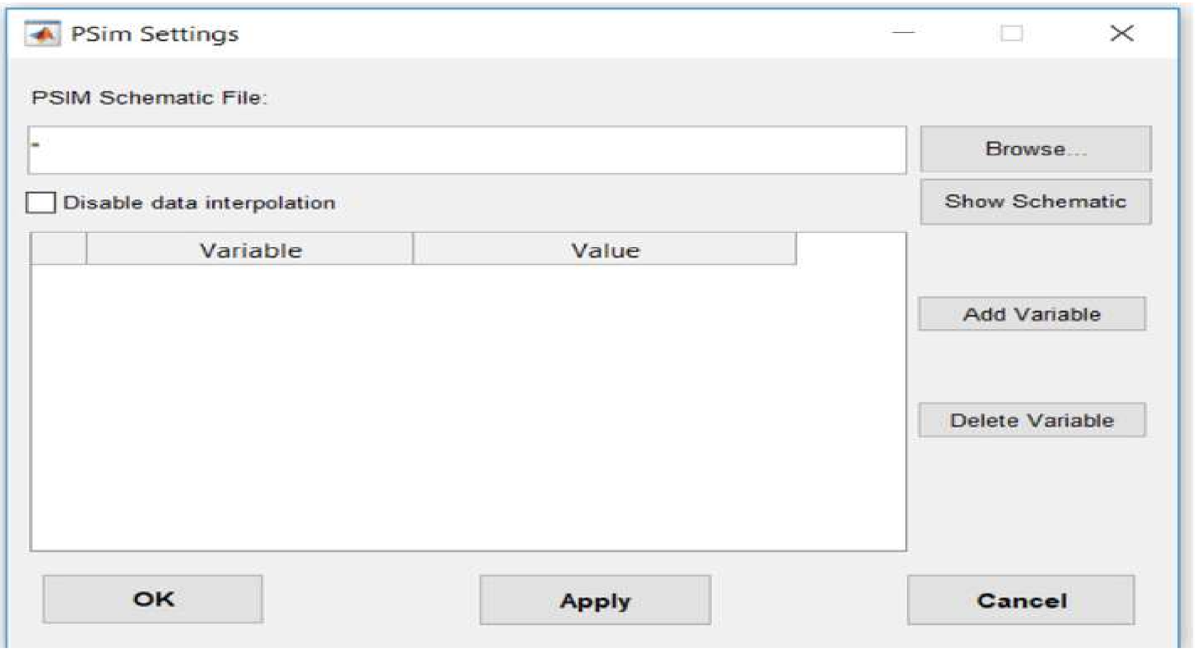
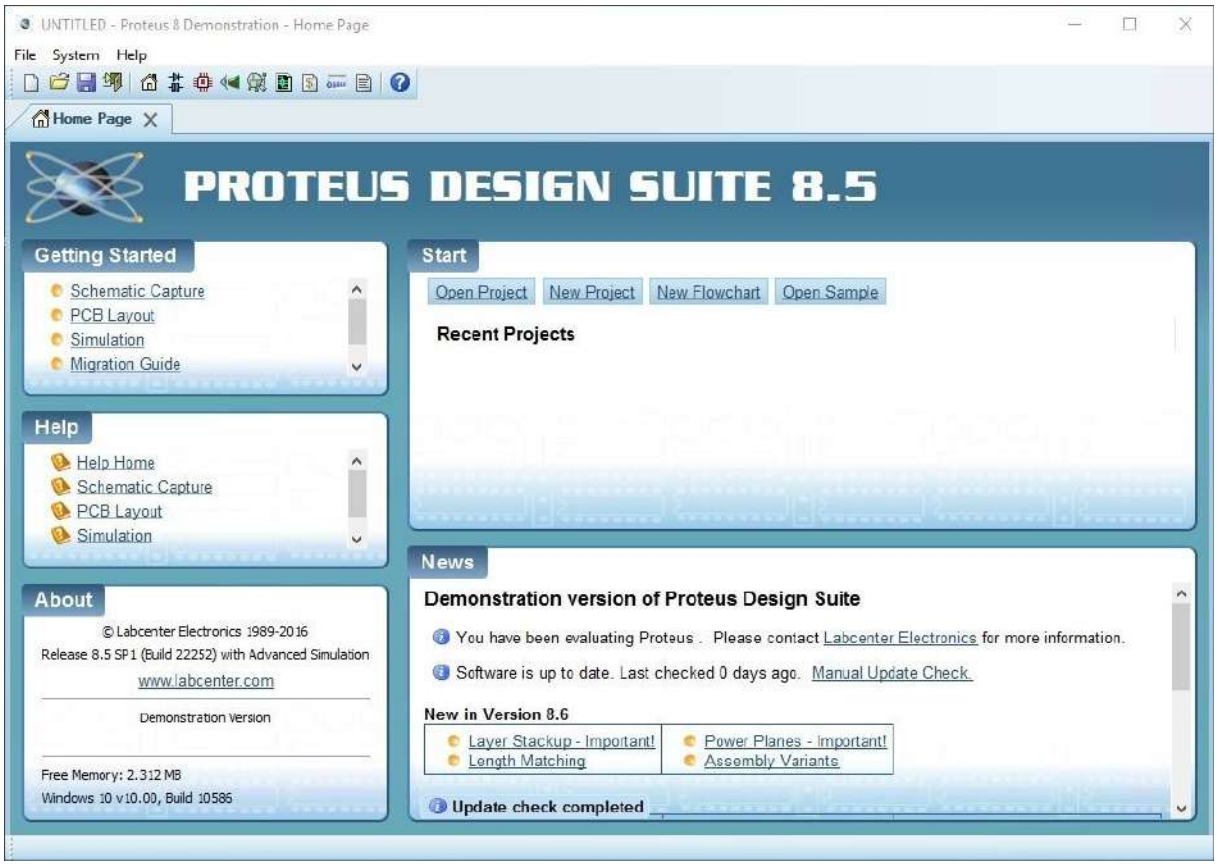


Figure 7. 25 Simulink simcoupler settings

11. The cosimulation configuration in Simulink is currently finished. Begin the simulation by going to Run in Simulink menu.

7.2 Simulation with other software

7.2.1 Proteus



© Labcenter Electronics 1989-2016
Release 8.5 SP1 (Build 22252) with Advanced Simulation
www.labcenter.com
Demonstration Version
Free Memory: 2,312 MB
Windows 10 v10.00, Build 10586

Demonstration version of Proteus Design Suite
You have been evaluating Proteus . Please contact Labcenter Electronics for more information.
Software is up to date. Last checked 0 days ago. Manual Update Check
Update check completed

Figure 7. 26 Proteus interface

The PROTEUS software is composed of three modules:

• The ISIS schematic editor

• The LISA simulator

• The ARES printed circuit board design tool.

ISIS produces, on one hand, a list of equipotentials that can be used by the LISA simulator and the ARES printed circuit board design tool. On the other hand, it produces a materials list and reports for checking electrical rules.

LISA is a set of simulation modules linked to ISIS. The PROSPICE simulation core is based on version 3F5 of the SPICE engine released by the University of Berkeley.

ARES is a Windows-compatible printed circuit board design module. It allows for the placement of components in automatic, manual, or semi-automatic modes and the routing of connections across multiple layers in automatic, manual, or semi-automatic modes.

7.2.1.1 The ISIS schematic editor

Isis is a schematic editor that integrates an analog, logic, or mixed simulator. All operations take place within this environment, from configuring the various sources to placing probes and plotting curves.

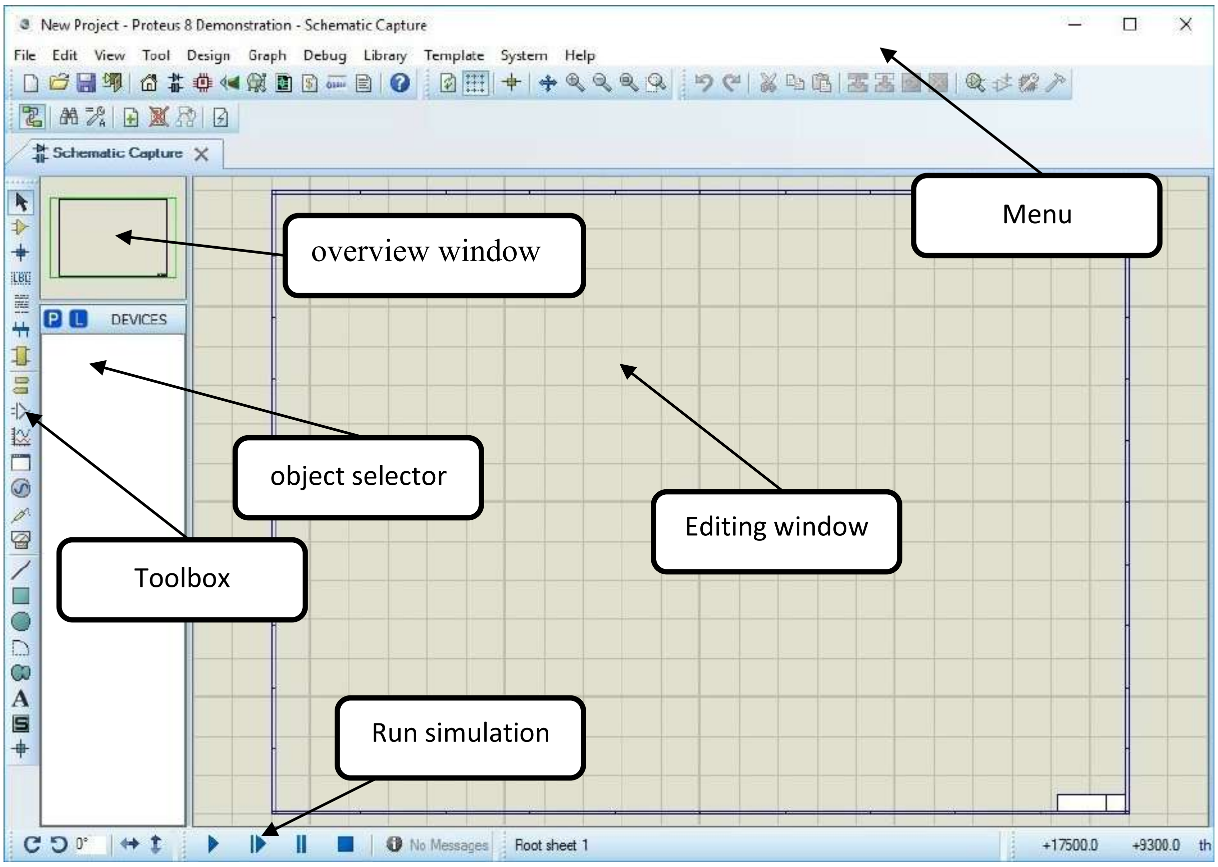


Figure 7. 27 ISIS Proteus interface

Overview window

The workspace is shown by the blue frame thanks to the 'Define sheet size' command from the'system' menu.

The working area, or the portion of the schematic that is displayed in the main window, is defined by the green frame.

- ➢ You can move this working area by pointing the mouse to the desired area of the overview window and clicking the left button.
- ➢ You can redefine the working area in the overview window by pressing the 'shift' key on the keyboard, combined with moving the mouse while holding down the left button

Editing window

Your circuit will be edited in this window. Only the portion of the circuit shown by the green frame in the overview window is represented by it. Using the overview window or the 'Zoom' command from the 'Display' menu, you may modify the work area.

Toolbox

It is composed of a set of icons whose functions will be detailed later and an object selector used to choose packages, pad styles, traces, vias, etc.

Cursor coordinates

The cursor's location in relation to the origin, which is by default at the middle of the editing window, is determined by the coordinates.
The unit of measurement for coordinates is 1/1000 inch (th).

Organization of the toolbox isis proteus

Various tools are often included in ISIS's toolbox to help users design and simulate electrical circuits.

The general structure of toolboxes in such software is presented as follow:

- Drawing tools help users to position components, trace wires and add labels or text annotations.
- Component Library: the location where users are able to search for and select electronic components to add to their personal schematic.
- Simulation Controls: Tools for starting, pausing and finishing simulations, additionally to establishing simulation parameters.
- Measurement Tools: it is used for measuring voltages, currents, and other parameters during simulations
- Editing Tools: To move, remove, rotate, and copy existing components or wires.
- Options for showing or hiding particular levels or categories of elements in the schematic are called "layers" and "views."

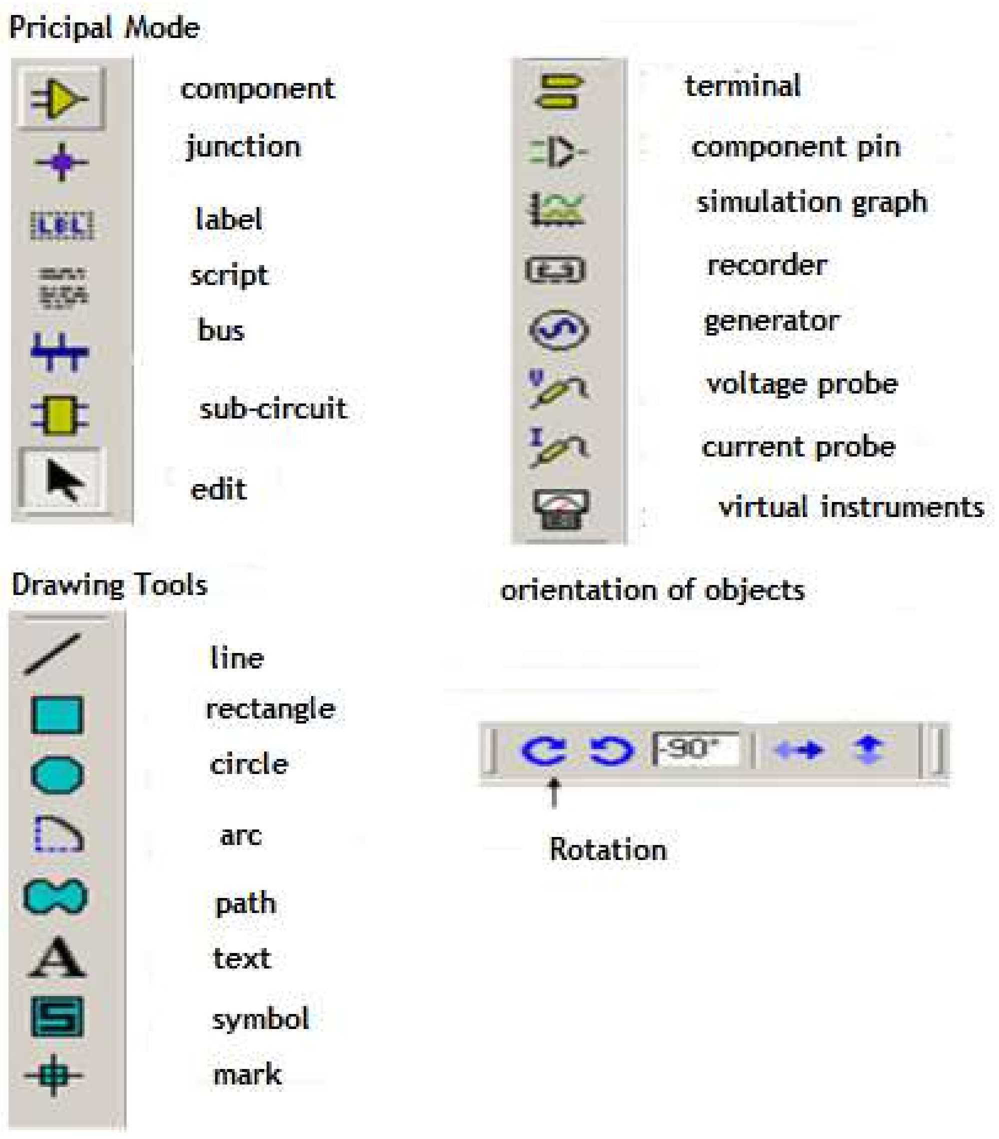


Figure 7. 28 ISIS Proteus Toolbars

7.2.1.2 Steps for entering a schematic

In the next basic outline for the steps in entering a schematic in ISIS Proteus, presented in an organizational or flowchart-like structure:

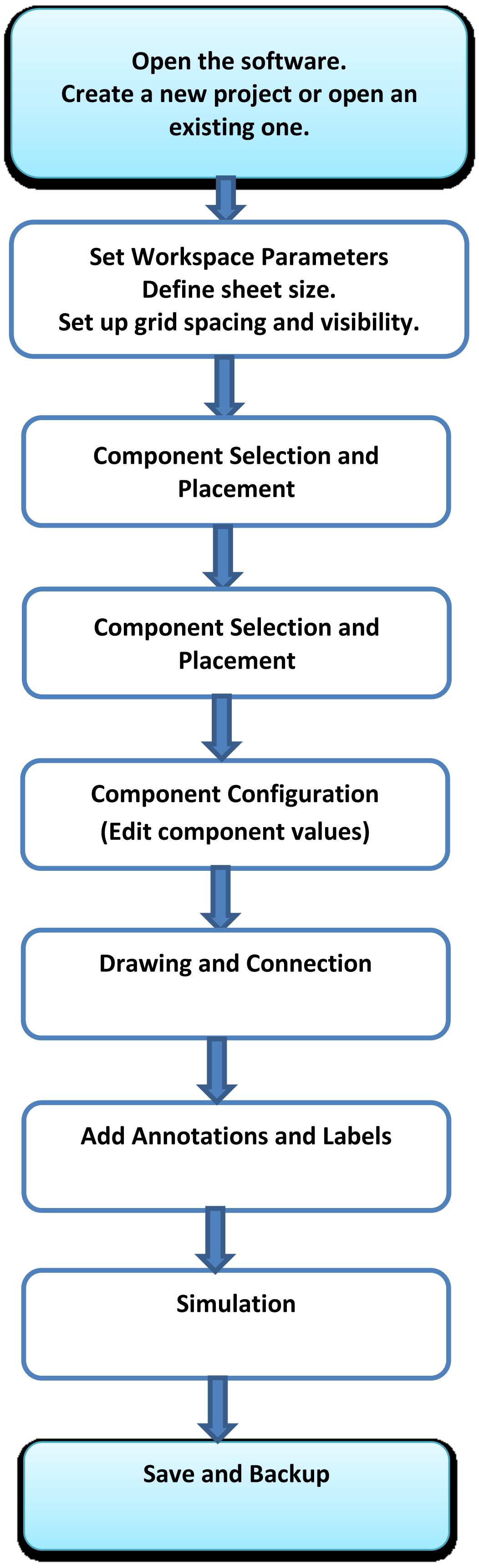

Open the software.
Create a new project or open an existing one.
Set Workspace Parameters
Define sheet size.
Set up grid spacing and visibility.
Component Selection and Placement
Component Selection and Placement
Component Configuration
(Edit component values)
Drawing and Connection
Add Annotations and Labels
Simulation
Save and Backup

Figure 7. 29 flowchart of step by step simulation with ISIS Proteus Toolbars

7.2.1.3 The ARES printed circuit board design tool

ARES is an element of software that enables for the automated or manual routing of electronic cards.

It is feasible to utilize ARES without first creating an ISIS graphic. This functionality enables you to easily design low-complexity circuits by simply putting components and sketching traces in ARES.

After the connections are made, the rails may be routed automatically.

You may also create new packages and save them in a library using this program. When combined with ISIS, you have a full solution that allows you to do all card design phases with a single schematic.

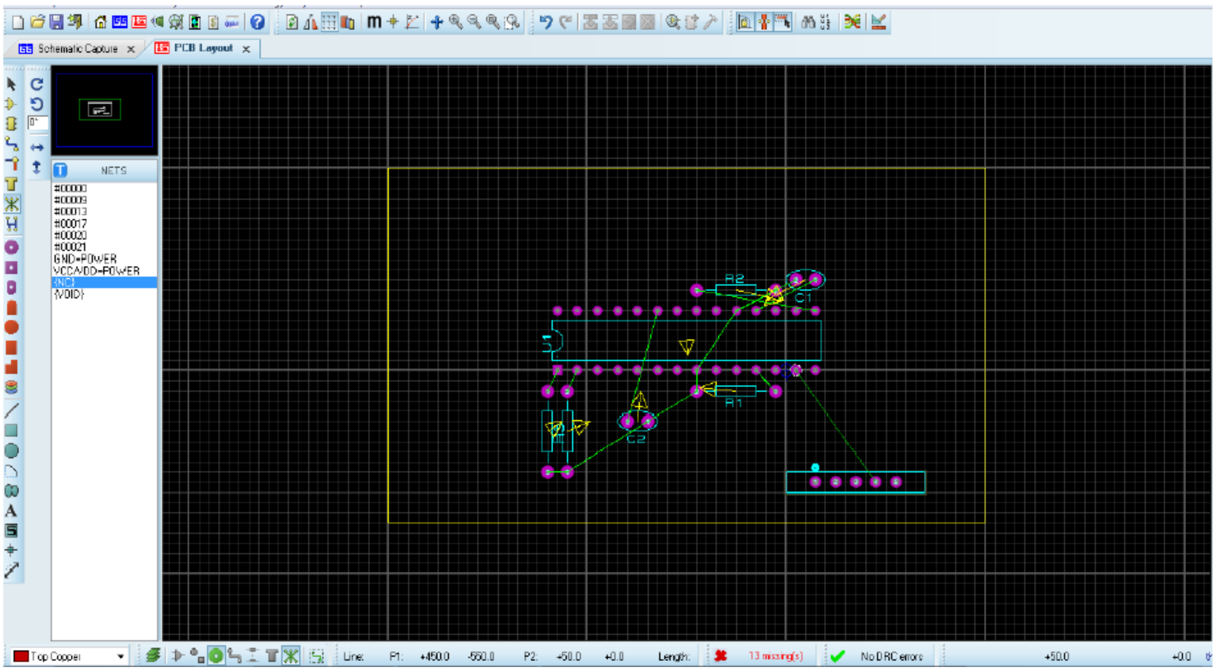


Figure 7. 30 ARES Proteus interface

The general appearance of ARES is very similar to that of ISIS.

- A menu bar.
- A main window in which you will create your routing but also your new components.
- A preview window in the top right corner.
- A palette below and a package selector.
- An active area selector, at the bottom left

The other functions are done identically to ISIS

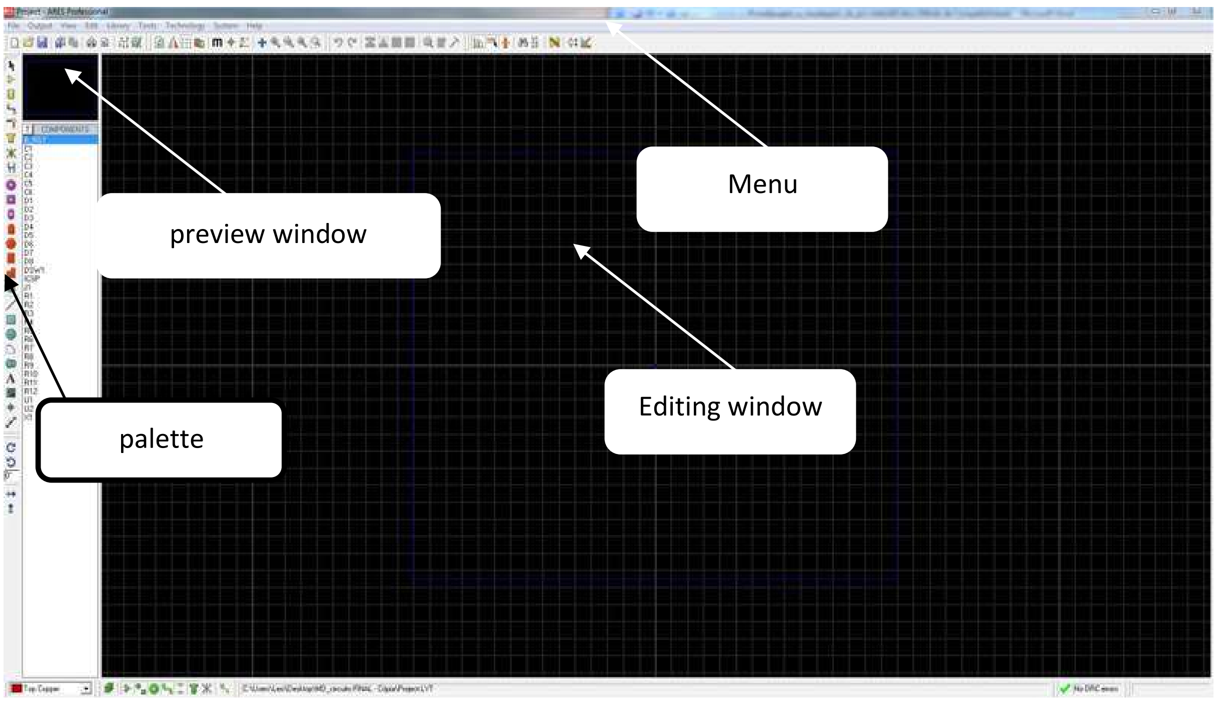
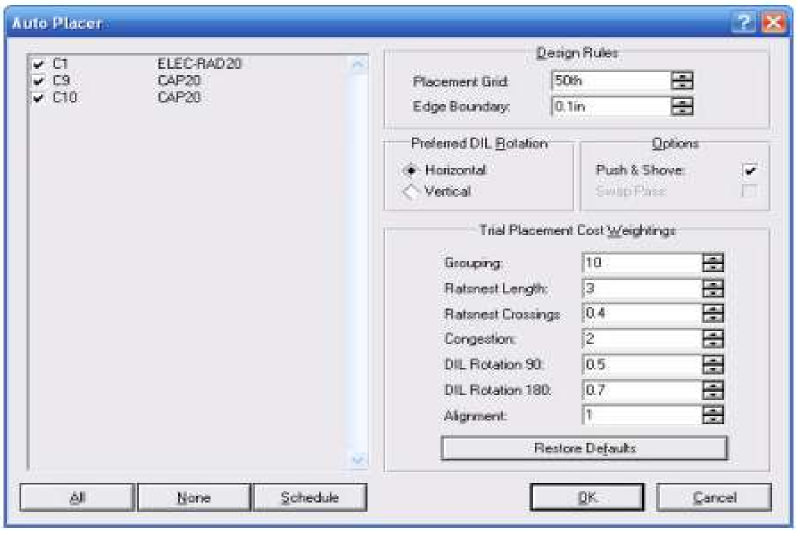


Figure 7. 31 ARES Proteus interface description

Creating board and placement the component

The first step before beginning routing is to create a board with design dimensions. If feasible, it is strongly encouraged to complete the project in standard formats. Go to graphical mode and choose "Board Edge" via the surface choice. Select the rectangle icon and create a rectangle that corresponds to the board's border. You've defined the surface of your electrical board once you've created the contour.

You may arrange the components in automated mode after designing an electronic board.
The above window displays when you employ the 'Automatic Placement' function. One may select which components you want to employ. The components are placed on the board once they have been validated.



Auto Placer

✓ C1	ELEC-RAD20
✓ C9	CAP20
✓ C10	CAP20

Design Rules
Placement Grid: 50th
Edge Boundary: 0.1in

Preferred DIL Rotation: ● Horizontal ○ Vertical

Options: Push & Shove: ✓ Swap Pass: ☐

Trial Placement Cost Weightings
Grouping: 10
Ratsnest Length: 3
Ratsnest Crossings: 0.4
Congestion: 2
DIL Rotation 90: 0.5
DIL Rotation 180: 0.7
Alignment: 1

Figure 7. 32 ARES Proteus auto placer window

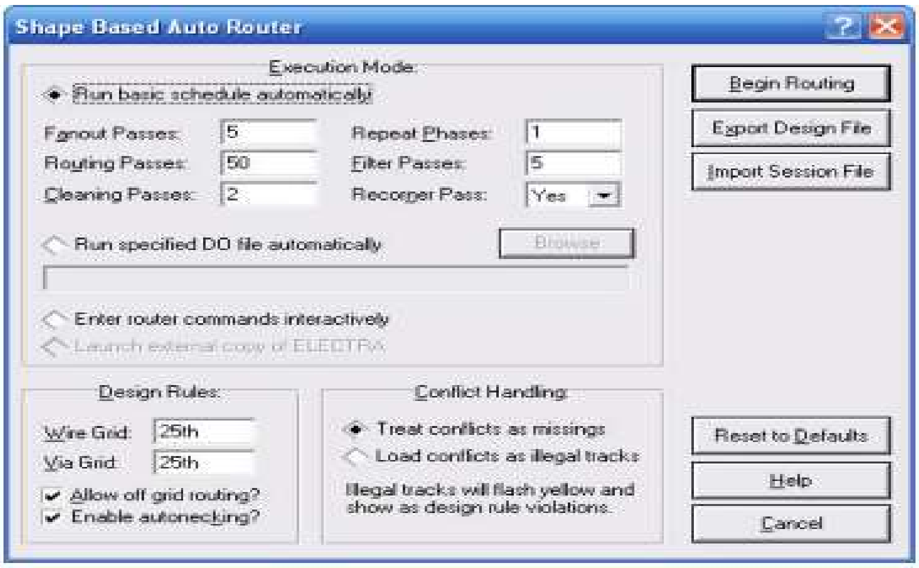
Manually putting components in ARES PCB design tools necessitate precise and deliberate judgments to achieve optimal circuit functioning, low noise, and efficient routing.

The routing of components

There are two kinds also for components routing in ARES software: the automatic and manual routing.

"Auto-routing" is a powerful function featured in ARES PCB design, that allow the paths linking components to be routed up automatically based on a set of established criteria. The goal is to reduce time over hand routing, particularly for complicated boards.

You can find this function in configuration menu by choose Shape based auto router.



Shape Based Auto Router

Execution Mode:
● Run basic schedule automatically
Fanout Passes: 5
Routing Passes: 50
Cleaning Passes: 2
Repeat Phases: 1
Filter Passes: 5
Recorner Pass: Yes
○ Run specified DO file automatically
○ Enter router commands interactively
○ Launch external copy of ELECTRA

Design Rules:
Wire Grid: 25th
Via Grid: 25th
✓ Allow off grid routing?
✓ Enable autonecking?

Conflict Handling:
● Treat conflicts as missings
○ Load conflicts as illegal tracks
Illegal tracks will flash yellow and show as design rule violations.

Figure 7. 33 ARES Proteus auto router window

It is quite unusual for an automated routing to be adequate without adjustment. As a result, being able to manually change or create tracks is really beneficial.
In ARES, choose the mode for manual routing. This might be represented by an icon that looks like a trace or wire. Select on a pin or pad where you want to begin tracing. A trace will show up while you drag the mouse.

7.2.2 PSpice

PSpice is a useful software application for the simulation and evaluation of electrical circuits. It enables the development of a schematic of any assembly in which the components are defined in a way that as closely as possible replicates genuine components. Following that, it mimics the operation of this assembly in order to study it from all sides, employing tools as diverse as they are complex.

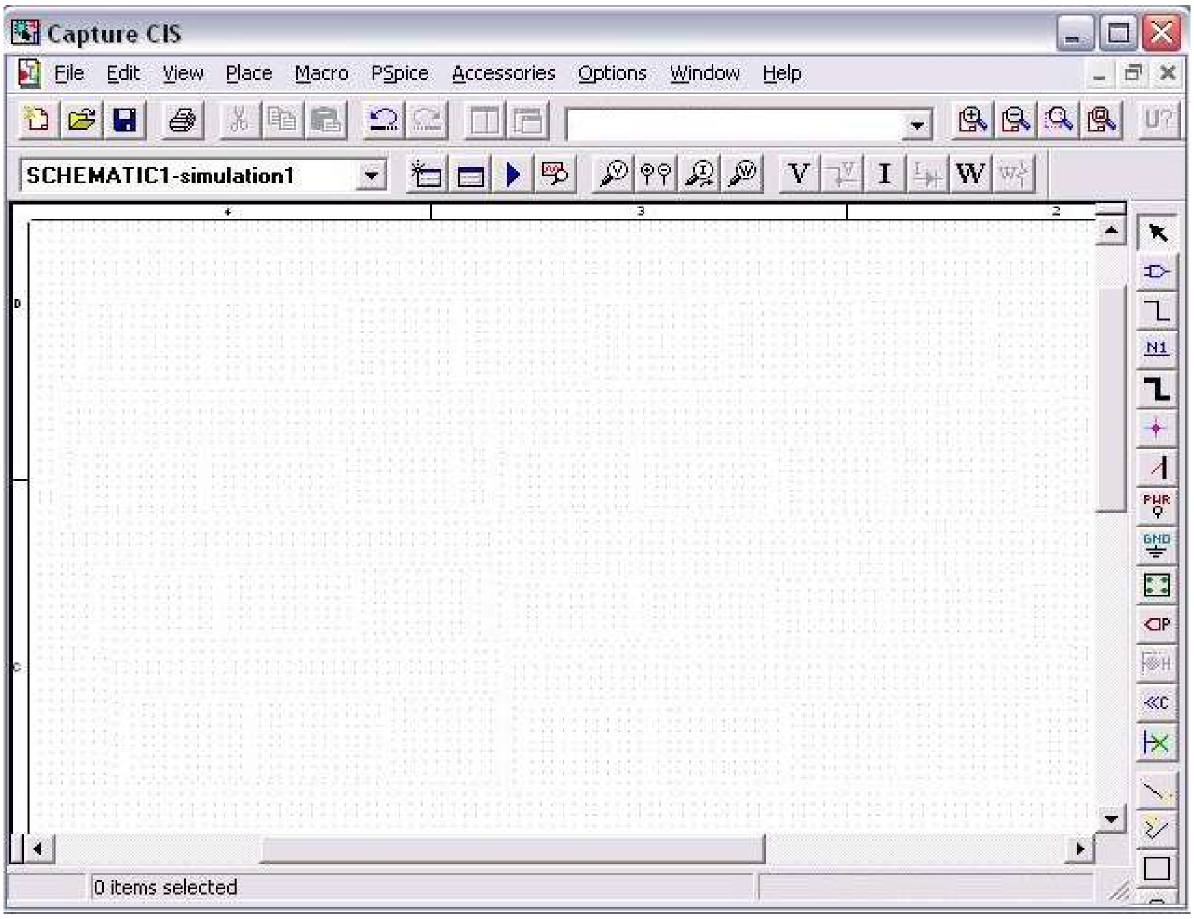

Capture CIS
File Edit View Place Macro PSpice Accessories Options Window Help
SCHEMATIC1-simulation1
V I W
N1
PWR
GND
0 items selected

Figure 7. 34 PSpice interface

PSpice is capable of running a variety of analyses, including:

- DC Analysis: The process of determining the behavior of a circuit at a certain point.

- AC Analysis: The examination of a circuit's frequency response.
- Transient Analysis: To examine the temporal response of the circuit to diverse inputs.
- Parametric analysis is used to determine how altering

Users can develop and build circuits through a visual interface, which can subsequently be simulated using software.

PSpice includes a large library of components that you can take advantage of to design an electrical circuit. These components vary from simple resistors and capacitors to complicated circuits with integrated circuits.

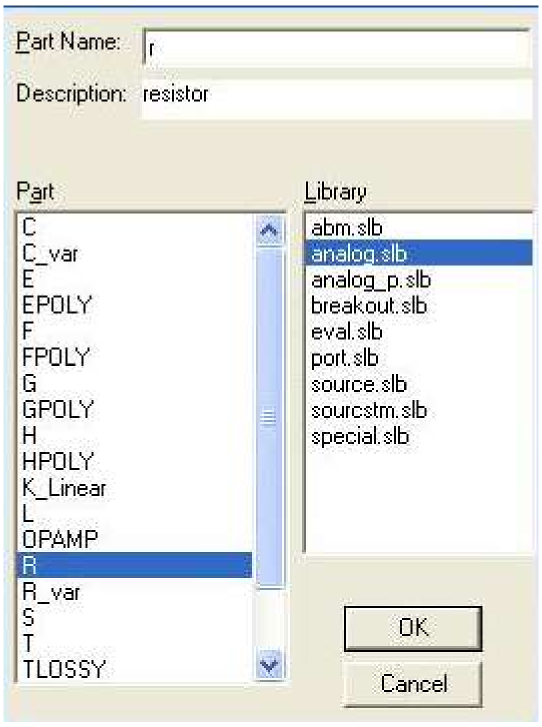


Figure 7. 35 PSpice component interface

7.2.3 Scilab

Scila is a cross-platform numerical computing program that is free to use and provides a computer environment for scientific purposes. It has a high-level programming language that is focused on numerical calculation. Its applications include signal processing, statistical analysis, image processing, fluid dynamics simulations, numerical optimization, and modeling and simulation of both explicit and implicit dynamic systems.

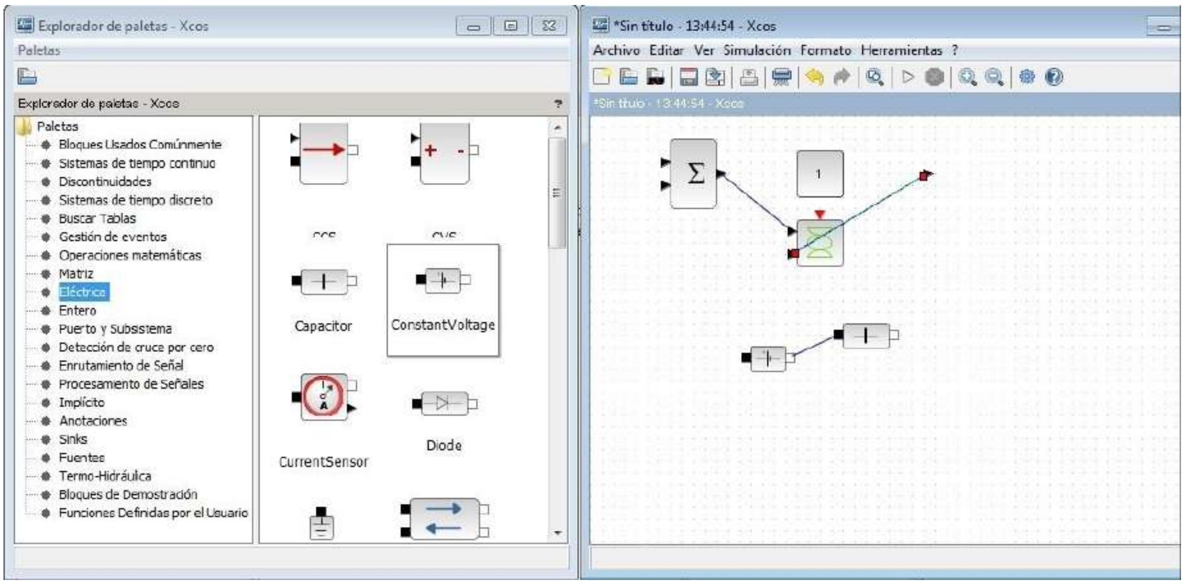

Explorador de paletas - Xcos
Paletas
Explorador de paletas - Xcos
Paletas
Bloques Usados Comúnmente
Sistemas de tiempo continuo
Discontinuidades
Sistemas de tiempo discreto
Buscar Tablas
Gestión de eventos
Operaciones matemáticas
Matriz
Eléctrica
Entero
Puerto y Subsistema
Detección de cruce por cero
Enrutamiento de Señal
Procesamiento de Señales
Implícito
Anotaciones
Sinks
Fuentes
Termo-Hidráulica
Bloques de Demostración
Funciones Definidas por el Usuario
CCS
CVS
Capacitor
ConstantVoltage
CurrentSensor
Diode
*Sin título - 13:44:54 - Xcos
Archivo Editar Ver Simulación Formato Herramientas ?
*Sin título - 13:44:54 - Xcos
Σ
1

Figure 7. 36 Scilab interface

Following are some interesting facts concerning Scilab

- One of the Scilab's primary benefits is that it is open source, which signifies that anyone can access to it. This renders it an appealing alternative for low-budget academics and industrial developers.
- Scilab provides its own programming language designed for numerical calculation.It is similar to matlab syntax.
- Scilab can handle a variety of rang of application.
- Scilab can co-simulate with other popular software such as MATLAB and C….
- Can Plot 2D and 3D graphics, which are mendatory for data analysis and result presentation.

Bibliographic references

Bibliographic references

[1] Course materials, « Logiciels de simulation, » by Dr. AFIF BENAMEUR, University of MASCARA.

[2] Course materials, « Applied Scientific Computing in Civil Engineering Using MATLAB, » by BABA HAMED FATIMA ZOHRA, USTO.

[3] « Control Tutorials for MATLAB and Simulink - Simulink Basics Tutorial » by Dawn Tilbury and Bill Messner.

[4] « Introduction à Matlab, » by Dr. F. Mudry, Higher School of Engineering of the Canton of Vaud

[5] « Matlab : Flow Control » by F. Nicol

[6] " Introduction à Matlab," by John Chaussard, University of Paris 13.

[7] " INTRODUCTION TO MATLAB FOR ENGINEERING STUDENTS," by David Houcque, Northwestern University.

[8] Course materials, « Logiciels de simulation, » by Dr. Kiyyour Ibrahim, University of BISKRA.

[9] Course materials, « Outils de Programmation pour les Mathématiques» by Dr. MOHAMMED OMARI, University of Adrar.

[10] « MATLAB : une introduction, »by François Langot, MAINE University.

[11] « MATLAB : LOGICIEL DE CALCUL SCIENTIFIQUE ET LANGAGE DE PROGRAMMATION, » by M.Y. BOUROUBI - UdeM.

[12] " Outils Mathematiques et utilisation de Matlab," by Quentin Glorieux, Pierre et Marie Curie - Paris VI University.

[13] « MATLAB : LOGICIEL DE CALCUL SCIENTIFIQUE ET LANGAGE DE PROGRAMMATION, » by M.Y. BOUROUBI - UdeM.

[14] « solving differential equations using simulink » by Russell Herman

[15] « MATLAB : prise en main » by F. Delebecque et J.C. Pesquet

[16] « Débuter avec MATLAB » by Stéphane Balac INSA of Lyon

[17] « 'MATLAB Manual' version 0.1 », by Yassine Ariba and Jérome Cadieux , Departments of GEI & Mechanics, Icam - Toulouse

[18] « Bases de la programmation » by Łukasz Starzak

[19] « : Introduction à la programmation avec Matlab » by Z.Mansouri, University of SKIKDA

[20] « LE LOGICIEL DE SIMULATION PSIM » by ELIPSE EDITION

[21] « PSIM User's Guide » by Powersim Inc

[22] « Simulation - sous ISIS » by Carlos Valente

[23] « Quick Guide for PSIM » by Powersim Inc

[24] « PSIM A Beginner's Guide » by Melvin Koshy

[25] « TUTORIAL How to use the Simcoupler » by Powersys SARL

[26] Training offer L.M.D 'Academic License' National Program 2018-2019.

WEBSITE

[1] https://matlab.mathworks.com/

[2] http://outilsrecherche.over-blog.com/
[3] http://www-syscom.univ-mlv.fr/~pesquet/optim/tpinit.pdf
[4] http://www.ordinateur.cc/Logiciel/Software-Engineering/128680.html
[5] https://www.filetypeadvisor.com/fr/extension/m
[6] https://www.javapoint.com/matlab-data-types
[7] https://ctms.engin.umich.edu/CTMS/index.php?aux=Basics_Simulink
[8] http://fabrice.sincere.pagesperso-orange.fr/cm_electronique/pspice_accueil.htm
[9] https://www.scilab.org/scilab-pour-debutants-tutoriel-french
[10] https://www.labcenter.com/