

Basic usage of MATLAB and SIMULINK*

Toru Namerikawa[†] Kanazawa University, JAPAN
Yasuhide Kobayashi Nagaoka University of Technology, JAPAN

13th June 2003 ver.1.2
8th March 2009 English version
9th Sep. 2010 fixed typos

Contents

1	Basic usage of MATLAB	1
1.1	What is MATLAB ?	1
1.2	Starting up and Shutting down of MATLAB	1
1.3	Simple example to use MATLAB	1
1.4	How to work with m-file	3
1.5	How to save figures as files	4
2	Matrix manipulation and linear algebra with MATLAB	4
2.1	Defining data	4
2.1.1	Substitution of values	4
2.1.2	How to use functions in MATLAB	5
2.2	Matrix manipulation and linear algebra	7
3	Basic usage of Simulink	12
3.1	What is Simulink ?	12
3.2	Preparing a simple model	12
3.2.1	Starting up Simulink	12
3.2.2	Preparing new model window	13
3.2.3	Copying blocks	13
3.2.4	Connecting blocks	14
3.2.5	Execution of simulation	15

*For detail, see manuals 'Getting Started with MATLAB' and 'Simulink 7 Getting Started Guide'

[†]He was supposed to come Timor-Leste, however, he couldn't because of the project delay due to the riot

1 Basic usage of MATLAB

1.1 What is MATLAB ?

MATLAB is a high-performance language for technical computing which includes numerical calculations in various scientific fields. The main features and application fields are as follows:

Feature:

1. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. Making program with MATLAB is easier than with non-interactive language such as C or FORTRAN.
2. MATLAB provides an easy-to-use environment with GUI(Graphical User Interface) to integrate computation, visualization, and programming.
3. MATLAB features a family of add-on application-specific solutions called *Toolboxes*. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

Application fields:

1. Math, computation, and engineering (especially, control engineering)
2. Algorithm development
3. Modeling and simulation
4. Data analysis, exploration, and visualization
5. Scientific and engineering graphics
6. Application development, including GUI building

The name “MATLAB” stands for **MAT**rix **LAB**oratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects, and have been developed.

1.2 Starting up and Shutting down of MATLAB

On the desktop of your PC, there is a icon of MATLAB. You can simply double click this icon to start up MATLAB. When you start MATLAB, the MATLAB desktop appears, containing GUI tools for managing files, variables, and applications associated with MATLAB. The right-hand-side window is command window, where you can type commands at the prompt `>>`.

```
>>
```

To end your MATLAB session, type “exit” at the prompt:

```
>> exit
```

1.3 Simple example to use MATLAB

Now let us consider to define a row vector t as

$$t = [1 \ 2 \ 3]. \quad (1)$$

This can be done in MATLAB by simply typing:

```
>> t=[1 2 3]
```

Then, the result will be the following:

```
t =  
    1    2    3  
>>
```

Next, consider to define a column vector u as follows:

$$u = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}. \quad (2)$$

In this case, each row needs a semicolon at the end of each row to indicate the row is closed:

```
>> u=[1;2;3]
```

```
u =  
    1  
    2  
    3
```

```
>>
```

Similarly, matrices can be defined. For example, how to define the following matrix V ?

$$V = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad (3)$$

This can be done by expressing three row vectors separated by semicolons:

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

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

```
>>
```

So far, you can define row and column vectors, and matrices. Next, let us consider manipulations and calculations with defined vectors. The simplest manipulation is to transpose. For example, the row vector t defined above is transposed to a column vector as:

$$t^T = [1 \ 2 \ 3]^T = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad (4)$$

This can be done as follows:

```
>> t'
```

```
ans =  
    1  
    2  
    3
```

```
>>
```

In the above, “ans” stands for “ANSwer”, which corresponds to the execution just before it.

Now, let us see how matrices are easily multiplied by evaluating a multiplication t^T and t . The operator for multiplication is $*$. So one type as:

```
>> t'*t
```

```
ans =  
    1    2    3
```

```
2 4 6
3 6 9
```

```
>>
```

So far, the answer always appears just after the execution, however, you can suppress it by using a semicolon. If you type

```
>> t=[1 2 3];
```

Then, no answer is appeared. (But t is memorized in MATLAB.)

How can we know the all memorized variables in MATLAB ? To do this, you can `who` command such as:

```
>> who
```

```
Your variables are:
```

```
V      ans      t      u
>>
```

Before proceeding next, let us clear the all variables in MATLAB. The command “clear” is available:

```
>> clear
```

After clearing variables, you can confirm by `who` command that all of the variables are erased.

Next, an array variable can be defined e.g. by the following:

```
>> k=0:0.1:10;
```

As the result, k is made as a row vector whose number of elements is 101. In the above statement `0:0.1:10` means a sequence of numbers from 0 to 10 with a incremental step of 0.1.

Now let us use a function of sin by:

```
>> y=sin(k);
```

In this case, the result of sin is stored in the variable y which is a 1×101 matrix, i.e. a row vector. This matrix size can be confirmed by `whos` command as:

```
>> whos
Name      Size      Bytes Class Attributes

k         1x101     808  double

y         1x101     808  double

>>
```

Finally, let us draw a figure. `plot` command is available. You can simply indicate target variables as arguments of the command. For example,

```
>> plot(k,y)
```

This draws a figure by using the variable k and y as x-axis and y-axis, respectively.

1.4 How to work with m-file

“m-file” is a one of convenient functions of MATLAB. m-file is text files containing MATLAB code, whose file name is ends with “.m”. Instead of typing in the MATLAB command window, you can create m-file which contains the same statements you would type at the MATLAB command line. To execute the stored statements in the created m-file, you can just type the file name excluding “.m”.

Exercise: Execute the statements `a=5`, `b=3`, and `c=a+b`.

Caution

- Don't use equations and/or preserved command names. (e.g. `1-2.m`, `sin.m`)
- Create m-file at the same directory to the working directory.

1.5 How to save figures as files

So far, you have one figure of sinusoidal wave. After drawing figures, they can be saved as various type of files e.g. GIF, JPEG, PostScript and so on.

Title, x-axis and y-axis labels can be set as follows:

```
>> title('sinosoidal wave')
>> xlabel('k')
>> ylabel('sin(k)')
```

Finally, if you choose JPEG format to save figure, then type as follows:

```
>> print -djpeg sin.jpg
```

In the above, the figure is saved as a file whose name is "sin.jpg".

2 Matrix manipulation and linear algebra with MATLAB

2.1 Defining data

In MATLAB, you can use various type of variables and data easily since

- All data is treated as array including scalars and vectors.
- No declaration is needed to prepare variables.
- Numerical data is treated as double precision floating point in calculation.

2.1.1 Substitution of values

Basic conventions for definition:

- Separate the elements of a row with blanks or commas.
- Use a semicolon, ; , to indicate the end of each row.
- Surround the entire list of elements with square brackets, [].
- If you want to suppress the output of the execution, use a semicolon at the end of the line.

Example 2.1 Define $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$

Answer There are several ways to define the matrix as follows:

(a) by using blanks

```
>> A=[1 2 3 ; 4 5 6 ; 7 8 0]
A=

     1     2     3
     4     5     6
     7     8     0
>>
```

(b) by using commas

```
>> A=[1, 2, 3 ; 4, 5, 6 ; 7, 8, 0]
A=

     1     2     3
     4     5     6
     7     8     0
>>
```

Caution

- (1) Upper and lower characters are distinguished in MATLAB to represent variable names. For example, A and a are different variables.
- (2) Maximum length of variable name is 31 characters.
- (3) Variable name must not begin with numbers nor operators.
- (4) Don't use preserved variables, functions, command such as imaginary unit i , the ratio of the circumference π , `sin`, `pwd`.

2.1.2 How to use functions in MATLAB

MATLAB provides basic functions as listed in Table 1 in order to generate various matrices. In addition, expansion of matrices and extraction of elements from matrices can be done easily.

<code>zeros</code>	Zero matrices
<code>ones</code>	All ones
<code>eye</code>	Identity matrices
<code>diag</code>	Diagonal matrices
<code>magic</code>	Magic squares
<code>rand</code>	Random matrices
<code>:</code>	Vectors with linearly changing elements

Table 1: Operators and functions for matrix manipulation

Example 2.2

- (1) Definition for basic matrices

- (a) Define 2×3 zero matrix $X1$

```
>> X1=zeros(2,3)
```

```
X1=
```

```
0 0 0
0 0 0
```

- (b) Define 3×3 identity matrix $X2$

```
>> X2=eye(3)
```

```
X2=
```

```
1 0 0
0 1 0
0 0 1
```

```
>>
```

- (c) Define a diagonal matrix $X3$ whose diagonal elements are 2, 5, and 7

```
>> X3=diag([2 5 7])
```

```
X3=
```

```
2 0 0
0 5 0
0 0 7
```

```
>>
```

(d) Extract diagonal elements from diagonal matrix

```
>> X4=diag(X2)

X4=

     1
     1
     1
>>
```

(2) Define vectors with linearly changing elements

(a) Define a sequence Y1 which starts from 25 to 0 with constant step of -5

```
>> Y1=25:-5:0

Y1=

    25    20    15    10     5     0
>>
```

(b) Define a sequence Y2 which starts from 1 to 5 with unit spacing

```
>> Y2=1:5

Y2=

     1     2     3     4     5
>> Y2=1:1:5

Y2=

     1     2     3     4     5
```

In the above example, unit step 1 can be omitted

(3) Examine matrix size

```
>> d=size(X1)

d=

     2     3
```

In the above, it is informed that the size of matrix $X1$ is 2×3 .

(4) Extract elements from matrices, substitute elements to matrices, and expand matrices with additional elements

First, define a matrix composed of random numbers as an example:

```
>> Z1=rand(3)

Z1=

    0.2190    0.6793    0.5194
    0.0470    0.9347    0.8310
    0.6789    0.3835    0.0346
>>
```

Note that the above matrix is made with random numbers ranging from 0 to 1. Therefore, different result is obtained for every execution.

(a) Substitute 1 to the (3, 2)-element of Z1

```
>> Z1(3,2)=1

Z1=

    0.2190    0.6793    0.5194
    0.0470    0.9347    0.8310
    0.6789    1.0000    0.0346
>>
```

(b) Extract (3, 2)-element of Z1

```
>> a=Z1(3,2)

a=

    1
```

(c) Extract the 1st column of Z1

```
>> b=Z1(:,1)

b=

    0.2190
    0.0470
    0.6789
>>
```

In the above, colon means all rows (or columns).

(c) Expand Y2 from 1×5 to 1×10 matrix by specifying the (1, 10)-element as 10.

```
>> Y2(10)=10

Y2=

    1    2    3    4    5    0    0    0    0    10
```

(d) Combine X3 and Z1

```
>> Z2=[X3 Z1]

Z2=

    2.0000    0         0         0.2190    0.6793    0.5194
    0         5.0000    0         0.0470    0.9347    0.8310
    0         0         7.0000    0.6789    1.0000    0.0346
>>
```

(e) Wrong combination of two matrices

```
>> [Y2 X3]
???: All matrices on a row in the bracketed expression must have
the same number of rows.
```

2.2 Matrix manipulation and linear algebra

MATLAB provides functions as shown in Table 2 and 3 for matrix calculation.

Example 3.1

+	Addition of arrays
-	Subtraction of arrays
*	Multiplication of matrices
/	(Right) division of matrices
\	left division of matrices
^	Power of matrices
.*	Multiplication of arrays
./	Right division of arrays
.\	Left division of array
.^	Power of arrays
'	Complex conjugate transpose of matrices
.'	Complex conjugate transpose of arrays

Table 2: Operators and functions for matrix manipulation

sin	Sine value
cos	Cosine value
abs	Absolute value
log	Natural logarithms
log10	Common logarithms
sqrt	Square roots
inv	Inverse matrices
eig	Eigenvalue and eigenvector
lu	LU decomposition
chol	Cholesky decomposition
qr	QR decomposition
svd	Singular value decomposition
trace	Trace (summation of diagonal elements)

Table 3: Scalar functions and matrix functions

(1) Execute the following addition:

$$\begin{bmatrix} 1 & 6 & 3 \\ 7 & 0 & 2 \\ 4 & 5 & 9 \end{bmatrix} + \begin{bmatrix} 7 & 2 & -1 \\ 6 & 8 & 3 \\ 9 & 2 & 2 \end{bmatrix} \quad (5)$$

by defining the 1st and 2nd matrices as X and Y , respectively.

Answer The matrix X can be defined as:

```
>> X=[1 6 3 ; 7 0 2 ; 4 5 9]

X =
     1     6     3
     7     0     2
     4     5     9
>>
```

Similarly, the matrix Y can be defined as:

```
>> Y=[7 2 -1 ; 6 8 3 ; 9 2 2]

Y =
     7     2    -1
     6     8     3
     9     2     2
>>
```

Finally, X and Y are added by using operator '+' as:

```
>> X+Y

ans =
     8     8     2
    13     8     5
    13     7    11
>>
```

(2) Execute the following multiplication:

$$\begin{bmatrix} 2 & 6 \\ 8 & 3 \\ 7 & 1 \end{bmatrix} \begin{bmatrix} 2 & 8 & 7 \\ 6 & 3 & 1 \end{bmatrix} \quad (6)$$

Answer First of all, define a matrix (say A for example) as:

```
>> A=[2 6 ; 8 3 ; 7 1]
```

```
A =
     2     6
     8     3
     7     1
>>
```

It can be seen that the 2nd matrix is the transpose of A , i.e. A^T . Thus, the multiplication can be executed as:

```
>> A*A'
```

```
ans =
     40     34     20
     34     73     59
     20     59     50
>>
```

(3) Execute multiplication of matrices $A = \begin{pmatrix} 3 & 6 \\ -2 & 1 \end{pmatrix}$ and $B = \begin{pmatrix} 8 & 0 \\ 5 & 7 \end{pmatrix}$. In addition, execute multiplication for each element.

Remark The difference from (2) is the multiplication in (3) is done for each element. For example, if matrices A and B are given as:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, \quad (7)$$

the matrix multiplication is defined as

$$AB = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}, \quad (8)$$

while multiplication for each element means

$$\begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} \\ A_{21}B_{21} & A_{22}B_{22} \end{bmatrix}. \quad (9)$$

Answer Two matrices can be defined as:

```
>> A=[3 6 ; -2 1];
>> B=[8 0 ; 5 7];
```

Multiplication for each element can be done by using the operator `.*` as:

```
>> A.*B
```

```
ans =
     24     0
    -10     7
>>
```

(4) Find eigenvalue and eigenvector of the following matrix A

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$$

```
>> A = [1, 2; 0, 3];
```

For given matrix A , eigenvalues and eigenvectors are defined as values λ and vectors x such that the following equation holds:

$$Ax = \lambda x. \tag{10}$$

The concept of eigenvalues and eigenvectors is very important for understanding and development in various scientific field of subjects including engineering, since the effect of matrix multiplication (left hand side of Eq.(10)) can be simply interpreted as scalar multiplication (right hand side) for the direction specified by eigenvectors.

To find the solution, function `eig` is used. If you type

```
>> eig(A)
```

then eigenvalues of matrix A will be obtained. If you want to find eigenvectors as well, then type

```
>> [V,D] = eig(A)
```

where V is a full rank matrix which is composed of eigenvectors, and D is a diagonal matrix whose diagonal elements are composed of eigenvalues, so that the following equation holds:

$$AV = VD \tag{11}$$

which is matrix version of Eq. (10).

```
>> [V,D]=eig(A)
```

```
V =
```

```
1.0000    0.7071
      0    0.7071
```

```
D =
```

```
1    0
0    3
>>
```

Example 3.2 Solve the following equation (find x and y):

$$\begin{cases} x + 5y = 7 \\ 2x + 4y = 8 \end{cases} \tag{12}$$

Answer :

First, transform the equation above into a liner matrix equation:

$$\begin{bmatrix} 1 & 5 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 7 \\ 8 \end{bmatrix} \tag{13}$$

If we define

$$A = \begin{bmatrix} 1 & 5 \\ 2 & 4 \end{bmatrix}, \quad X = \begin{bmatrix} x \\ y \end{bmatrix}, \quad b = \begin{bmatrix} 7 \\ 8 \end{bmatrix}, \tag{14}$$

then the equation is represented as:

$$AX = b. \tag{15}$$

Multiplying A^{-1} from left side to the above equation, the following equation holds:

$$X = A^{-1}b \tag{16}$$

In MATLAB, matrices A and b can be defined as:

```
>> A=[1 5 ; 2 4];
>> b=[7 ; 8];
```

X can be obtained as follows:

```
>> X=A\b
```

```
X=2.000
    1.000
```

Caution In order for matrix A to have inverse, it is necessary that matrix A has full rank. (In the present case, 2.) It can be confirmed by the function ‘rank’ as follows:

```
>> n=rank(A)
n=
    2
```

Because the matrix A here is 2×2 square matrix, the result above means the matrix A has full rank.

There is another way to find X as follows:

```
>> X=inv(A)*b
```

However, the accuracy of the calculation with ‘inv’ function might be worse than the previous one.

Exercise

(1) Define the following matrices:

1-1)

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{bmatrix} \tag{17}$$

1-2)

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 4 & 0 & 3 \end{bmatrix} \tag{18}$$

1-3)

$$\begin{bmatrix} 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 \\ 6 & 6 & 6 & 6 \\ 8 & 8 & 8 & 8 \\ 10 & 10 & 10 & 10 \end{bmatrix} \tag{19}$$

(2) Solve the following equation:

$$\begin{cases} 2x + y + 5z = 5 \\ 2x + 2y + 3z = 7 \\ x + 3y + 3z = 6 \end{cases} \tag{20}$$

(3) Solve the following equation:

$$x^2 + 6x + 10 = 0 \tag{21}$$

3 Basic usage of Simulink

This section simply describes basic usage of Simulink which is one of MATLAB software. Specifically, in section 3.1 outline of Simulink is introduced, then in Section 3.2 a simple model is created followed by the simulation.

3.1 What is Simulink ?

Simulink is a software package of MATLAB for modeling, simulation, and analysis of dynamic systems. Simulink provides a graphical user interface (GUI) for building models as block diagrams, allowing you to draw models as you would with pencil and paper.

First of all, in order to know how Simulink works, let's execute a demo model. Type following in MATLAB prompt:

```
>> thermo
```

Then a demo model will be appeared. In the model, double click the block named 'Thermo Plots'. This will open a Scope window. To start simulation, pull down **Simulation** menu and choose the **Start** command. When the simulation starts, both indoor and outdoor temperatures are plotted in the Scope block named 'Indoor vs. Outdoor Temp', and accumulated heat cost is plotted in the Scope block named 'Heat Cost(\$)'.

To stop the simulation while it is running, choose **Stop** command in the **Simulation** menu. (Since the default simulation period of the demo is 10 seconds, it is automatically stop after the period even if **Stop** command is not given.)

Close the model after the simulation is finished by selecting **Close** in **File** menu.

3.2 Preparing a simple model

This section describes how to create a simple model using Simulink as shown in Fig. 1.

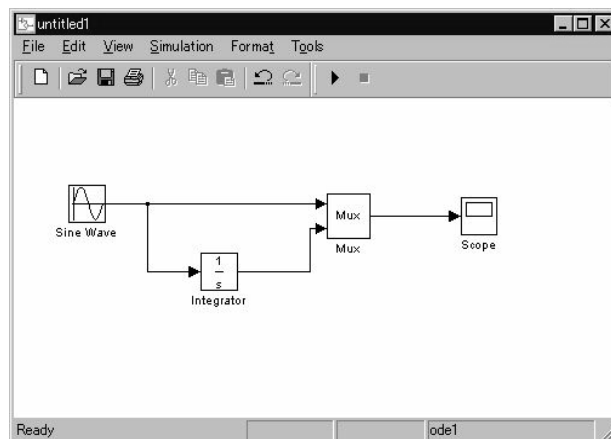


Figure 1: Simulink model

3.2.1 Starting up Simulink

Type as following in MATLAB command window to start up Simulink:

```
>> simulink
```

Then, you have Simulink Library Browser shown as Fig.2.

The following libraries are used in order to create the model shown in Fig.1:

- Sources library (Sine Wave block)
- Sinks library (Scope block)
- Continuous library (Integrator block)
- Signal Routing library (Mux block)

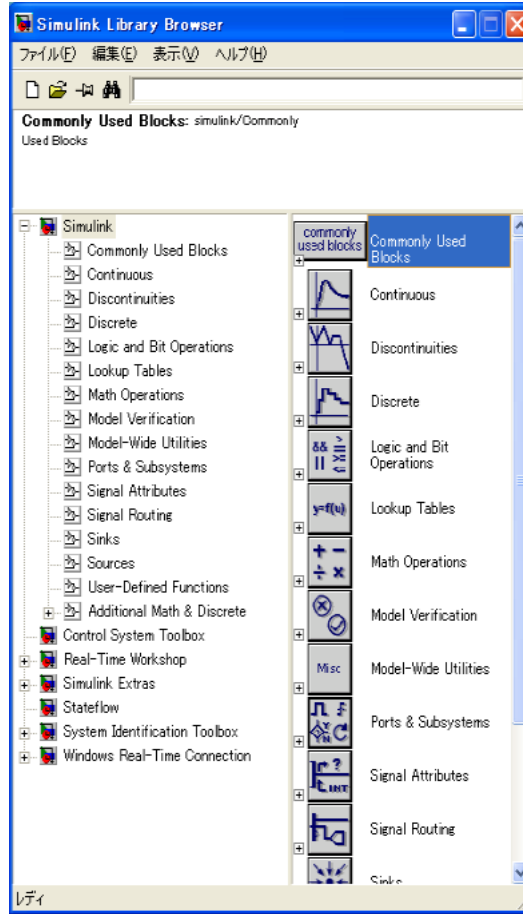


Figure 2: Simulink Library Browser

3.2.2 Preparing new model window

First, a new model window has to be opened, where blocks are located. To do that, select **File->New->Model** in the Simulink Library Browser. Then, the empty model window as shown in Fig.3 is opened.

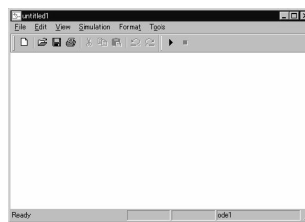


Figure 3: New model window

3.2.3 Copying blocks

Now copy the necessary blocks from the Simulink Library Browser to the model window.

For **Sine Wave** block, open **Sources** library by double-clicking the icon. In **Sources** library as shown in Fig.4, all of the blocks are sources of signals.

Select the **Sine Wave** block in the Simulink Library Browser, then drag it to the model window. A copy of the Sine Wave block appears in the model window as shown in Fig.5.

Similarly, copy the remaining blocks to the model window. Then your model window becomes as shown in Fig.6.

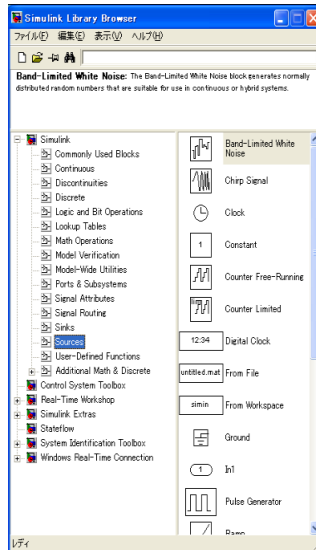


Figure 4: Sources library

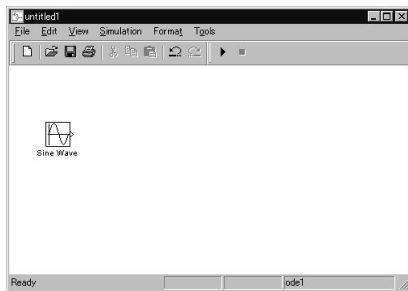


Figure 5: Sine Wave

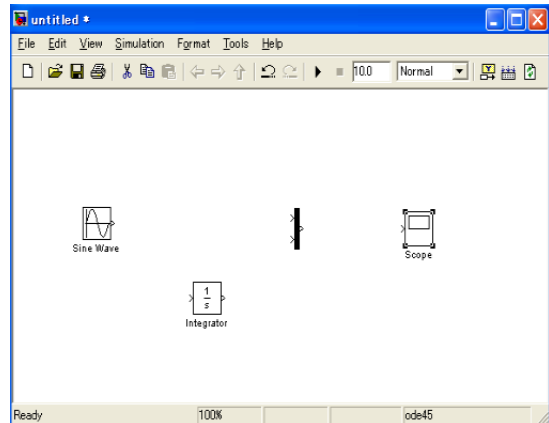


Figure 6: Model

You can see block icons have symbol \triangleright , where the symbol \triangleright pointing into the block is an **input port**, while the symbol \triangleright pointing out of the block is an **output port**. All of signals get out from blocks by output ports, travel by lines, and get into blocks by input ports. After connecting blocks by lines, symbols \triangleright are disappeared.

3.2.4 Connecting blocks

To draw a line between the Sine Wave block and the Mux block, position the mouse pointer over the output port on the Sine Wave block. Note that the pointer changes to a cross hairs (+) shape while over the port. Now drag a line from the output port to the top input port of the Mux block. Note that the line is dashed while you hold the mouse button down, and that the pointer changes to a double-lined cross hairs as it approaches the input port of the Mux block. Then, release the mouse button over the output port. Then the blocks are connected as shown in Fig.7.

In our goal as in Fig.1, there is a branch of lines. The branch is made by the following procedure:

- 1 Position the mouse pointer on the line between the Sine Wave and Mux block.
- 2 Press and hold the **Ctrl** key, then drag a line to the Integrator block's input port.

Finally the goal of the model as in Fig.1 has been created.

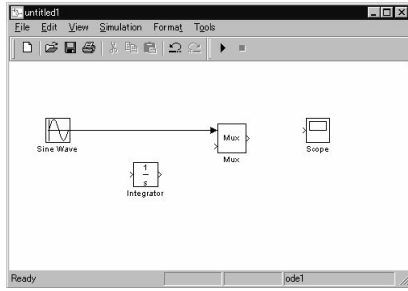


Figure 7: Connecting blocks(1)

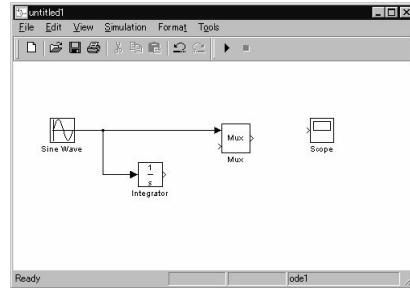


Figure 8: Connecting blocks(2)

3.2.5 Execution of simulation

Before executing the simulation, prepare the scope figure as shown in Fig.9 by double-clicking the Scope block.

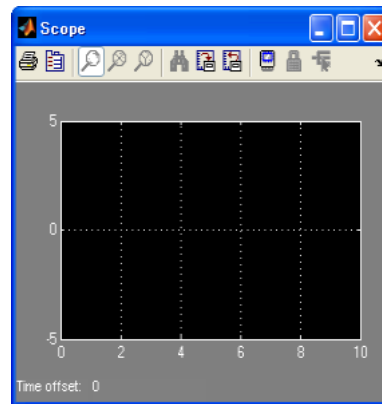


Figure 9: Scope

Select **Simulation->Configuration Parameters** to open the Configuration Parameters dialog box (Fig.10). If you need to change the parameter, e.g. the simulation period from 10 seconds to 20 seconds, give suitable values. Click **OK** to close Configuration Parameters dialog box.

Now you are ready to simulate your example model and observe the simulation results. To run the simulation, select **Simulation->Start** in the model window. The simulation will be automatically stopped after the specified simulation period. In the scope, two waveforms are appeared as shown in Fig.11.

If you want to stop the simulation while the simulation running, select **Simulation->Stop**.

To save the model file you created, select **File->Save** in the model window followed by specifying its file name.

Exercise

(1) Draw the step response of the 1-order system as shown in Fig.12.

- By double-clicking the Step block, you can modify the parameter of the step signal, e.g. raising time ('Step time') as shown in Fig.13.
- Similarly, you can change parameters of **Transfer Fcn** block. For example, if you specify

Numerator: [1]

Denominator: [2 3]

then, the corresponding transfer function, say $G(s)$, is

$$G(s) = \frac{1}{2s + 3}. \quad (22)$$

Alternatively, if you specify

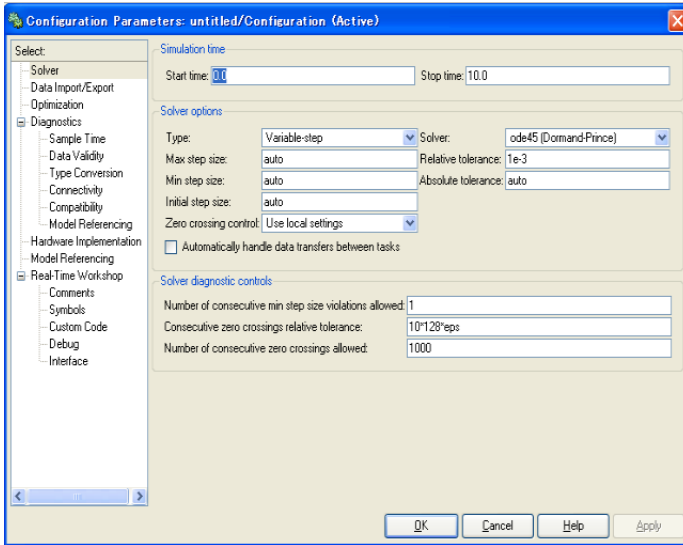


Figure 10: Configuration Parameters dialog box

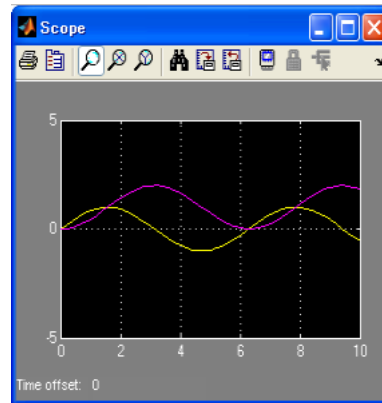


Figure 11: Scope

Numerator: [4]

Denominator: [1 2 3]

then, $G(s)$ corresponds to following:

$$G(s) = \frac{4}{s^2 + 2s + 3} \quad (23)$$

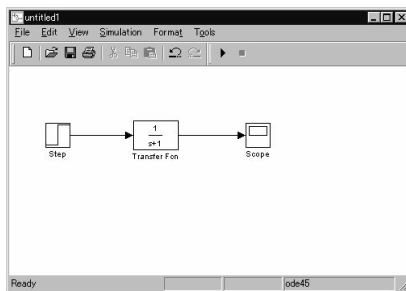


Figure 12: Model 1-order system

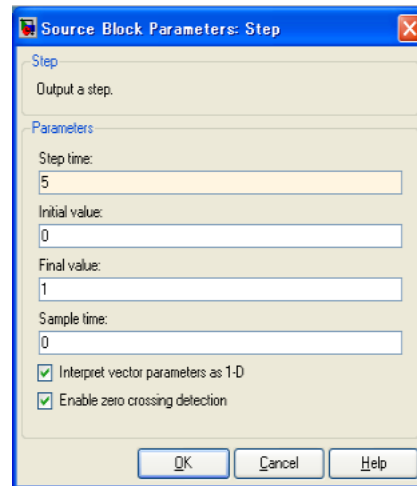


Figure 13: Parameters of Step block

(2) Draw the response of 2-order system.

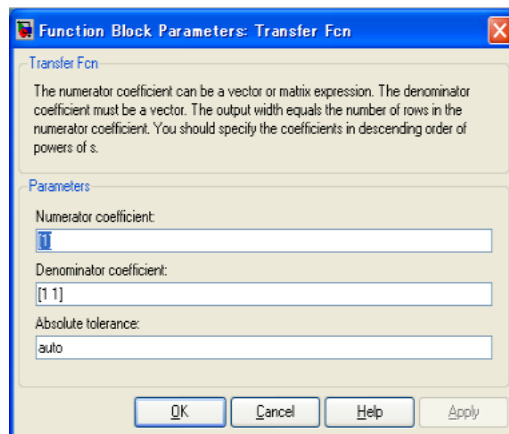


Figure 14: Parameters of Transfer Fcn block