An introduction to MATLAB and SIMULINK

Maarten Vergauwen, Alexander Bertrand, and Toon van Waterschoot[1]

March 2010

[1]For questions or remarks: toon.vanwaterschoot@esat.kuleuven.be

Contents

1 Matlab **5**

1.1 What is Matlab ? . . . 5
1.2 Getting started, getting stopped . . . 5
1.2.1 Starting a Matlab session . . . 5
1.2.2 Stopping a Matlab session . . . 5
1.3 Setting your Matlab-path . . . 6
1.4 Getting help . . . 6
1.5 Use of special keys . . . 6
1.5.1 Stopping a command . . . 6
1.5.2 Using arrow-keys . . . 6
1.6 Entering data . . . 7
1.6.1 Constants . . . 7
1.6.2 Creating a vector . . . 7
1.6.3 Creating a matrix . . . 7
1.6.4 The size of a matrix . . . 8
1.7 Selecting data . . . 8
1.8 Matrix arithmetics and functions . . . 8
1.8.1 Basic matrix arithmetics . . . 8
1.8.2 Elementwise functions . . . 9
1.8.3 Matrix functions, based on svd . . . 9
1.8.4 Elementary math functions . . . 9
1.9 Operations on the data stack . . . 10
1.9.1 Viewing the current variables . . . 10
1.9.2 Saving variables into a data file . . . 10
1.9.3 Loading a data file . . . 10
1.9.4 Clearing variables . . . 10
1.10 Graphics . . . 10
1.10.1 Making a plot . . . 10
1.10.2 Zooming . . . 11
1.10.3 Handling figures . . . 11
1.10.4 Using subplots . . . 11
1.10.5 How to print a screen plot . . . 11
1.11 M-files . . . 12
1.11.1 Creating an M-file . . . 12
1.11.2 Asking for input . . . 13
1.12 Functions . . . 13
1.13 Control structures . . . 13
1.13.1 The “for” loop . . . 13

1.13.2 The “while” loop 14
1.13.3 The “if” statement 14
1.13.4 Pre-allocation 14
1.14 Useful commands 15

2 Simulink **17**
2.1 What is Simulink ? 17
2.2 Getting started 17
2.3 The library 17
2.4 The worksheet 18
2.4.1 Entering new blocks 18
2.4.2 Resizing and moving blocks 18
2.4.3 Copying, removing, rotating and flipping blocks 18
2.4.4 Connecting blocks 18
2.4.5 Grouping blocks 19
2.4.6 Saving and loading block diagrams 19
2.4.7 Simulation parameters 19
2.4.8 Simulating 20
2.4.9 The Scope block 20

Chapter 1

An overview of Matlab

1.1 What is Matlab ?

Matlab is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

Matlab is an interactive system whose basic data element is an array that does not require dimensioning. This allows to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

Matlab has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, Matlab is the tool of choice for high-productivity research, development, and analysis.

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

In these exercise sessions, an introduction will be given to Matlab and the Simulink-toolbox. In this text only the Matlab-part is described.

1.2 Getting started, getting stopped

1.2.1 Starting a Matlab session

- In Linux: Type `matlab` in a terminal to start Matlab.
- In Windows: should be in the programs list.

1.2.2 Stopping a Matlab session

- type `quit` or `exit`.

1.3 Setting your Matlab-path

- Matlab knows about all files in the current directory. You can query this directory with the command `pwd` and change it with the `cd` command, like in a UNIX terminal. If you want Matlab to search not only the current directory for files, you can set the **Matlab-path**. This is a variable in Matlab containing a list of directories where Matlab will search for files or commands.

- Alternative: above the Matlab terminal, you see 'current directory'. Here you can type the path you need or you can press the '...' button on the right to select it from a list.

- Create a new directory for this exercise session, and set the path of Matlab to this directory.

1.4 Getting help

- To get an overview of all packages, type `help`.

- To get help on a specific topic, type `help topic`. This is very important and you should do this very frequently. A certain command often has a lot of possible uses and outputs, and can have a different meaning when applied to a vector or a matrix. For instance, the `min` command returns the minimum of a vector, or the minimum of every *column* (not row!) of a matrix. It can also be used to find the indices in the vector/matrix where this minimum occurs. Therefore, the only way to know the right use of `min` for your case, just type `help min`.

- The `lookfor` command allows you to search for functions based on a keyword. It searches through the first line of help text for each matlab function. For example: if you try `help inverse`, matlab will tell you that there is no file called "inverse.m". If you type in `lookfor inverse`, you will get over a dozen matches.

- If you have some time left at the end of this session and you want to see some matlab examples, type `demo`. From the menu displayed, run the demos that interest you, and follow the instructions on the screen.

- The help pages in Matlab are very good. If you have problems with the syntax or the behavior of a certain function, check the help page of this function first, *before asking*.

1.5 Use of special keys

1.5.1 Stopping a command

- To interrupt the execution of a command, type <CONTROL>–C. Example: type `i=1:0.001:1000` and press <CONTROL>–C to interrupt.

1.5.2 Using arrow-keys

- To get former commands back, use the up-arrow. This is very interesting for editing long commands. Example: Type `a=[1:0.1:10;5:0.1:14)`. This will raise an error. Then press the up-arrow and change the '`)`' into '`]`'.

- If it has been a while since you entered the command you want to review, type in the first characters of that command. Pressing the up-arrow will then only browse through the commands which start with those characters.

1.6 Entering data

1.6.1 Constants

- Some important constants exist in Matlab.
  - `pi` is 3.14159265... Example: check that `cos(pi)` $=-1$.
  - `i` or `j` is the imaginary unit $\sqrt{-1}$. It is a good idea to never use `i` or `j` as variables. However, Matlab allows this, so pay attention.
  - `Inf` is infinity.
  - `NaN` is Not-a-number. Example: check that Inf/Inf equals NaN.

1.6.2 Creating a vector

Vectors are one-dimensional matrices and are often used in Matlab. They are created as follows.

- Type `v=[1 2 3]` or `v=[1,2,3]` to create a horizontal vector.
- Type `v=[1;2;3]` to create a vertical vector.
- Type `v=[1:10]` to create a vector with elements 1 to 10.
- Type `v=[1:0.5:10]` to create a vector with elements from 1 to 10 with a step of 0.5.
- Typing a semicolon (;) after a command will suppress the print-out of the result to the screen. See the difference between the following statements by trying them out: `v=[1:5]` and `v=[1:5];`

1.6.3 Creating a matrix

Matrices are the basic features in matlab (in fact everything, even numbers or vectors, is stored as a matrix. That's where the name "Matlab" comes from: MATrix LABoratory).

- Type `m=[0 1 2;1 2 3]` to create a matrix with two rows and three columns.
- Type `m=[0:2:20;1:0.1:1]` to see that all rows (and columns) must have the same number of elements. Type `m=[0:2:20;1:0.1:2]` to get the correct matrix (don't forget the arrow-keys ;-))
- To create a matrix with 10 rows and 2 columns which is filled with zeros, type `m=zeros(10,2)`.
- To create a matrix with 10 rows and 2 columns which is filled with ones, type `m=ones(10,2)`.
- To create a unity matrix of rank 5, type `m=eye(5)`.

- Everything together: Create the following matrix:

$$m = \begin{pmatrix} 5 & 1 & 0 & 1 & 0 \\ 3 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

  A short expression for this is
  `m=[[5 1 0;3 1 0] eye(2);ones(1,5);zeros(1,5)]`.

1.6.4 The size of a matrix

- Type `size(m)` to see the numbers of rows and columns of the matrix m.
- Type `length(v)` to see the number of elements in the vector v.

1.7 Selecting data

- Type `m` to see the current value of the variable m.
- Type `m(i,j)` to select the element on the i-th row and the j-th column of m.
- Type `m(i,:)` to select the i-th row of m.
- Type `m(:,j)` to select the j-th column of m.
- Type `m([1 3],:)` to get a new matrix containing the first and third row of m.
- Type `diag(m)` to get a vector that holds the diagonal of m.

1.8 Matrix arithmetics and functions

1.8.1 Basic matrix arithmetics

Always keep in mind that most operators are matrix operators, e.g. multiplication performs a matrix multiplication.

- Addition: `x=a+b`
- Subtraction: `x=a-b`
- Multiplication: `x=a*b`
- Inversion: `x=inv(a)`
- Division: `x=a\b` solves $ax = b$. Notice that this statement is equivalent to `x=inv(a)*b`. If a is not invertible or ill-conditioned, a Warning message will appear. If a has more rows than columns, the solution is given in least-squares sense (do you remember the formula for the least-squares solution?).

- Transposition: `x=a'`. Always keep in mind that for complex numbers, this will also apply a complex conjugate together with the transposition. In that way, this command is equivalent to taking the conjugate transpose of a matrix. A Hermitian matrix is invariant under this operator. For real numbers, a Hermitian matrix is a symmetric matrix. To transpose a complex matrix, without applying the conjugate operator, use `x=a.'`.

- Exponentiation: `x=a^n`

1.8.2 Elementwise functions

Adding a dot in front of the operator makes it element-wise:

- Multiplication: `x=a.*b` yields $x_{ij} = a_{ij} * b_{ij}$.

- Division: `x=a./b` yields $x_{ij} = \frac{a_{ij}}{b_{ij}}$.

- Exponentiation: `x=a.^n` yields $x_{ij} = a_{ij}^n$.

- Absolute value (or modulus for complex numbers): `x=abs(a)` yields $x_{ij} = |a_{ij}|$.

- Minimum:

  `x=min(a)` yields a vector with the minimum of each column.

  `x=min(min(a))` yields $x = min(a_{ij})$.

- Maximum:

  `x=min(a)` yields a vector with the maximum of each column.

  `x=max(max(a))` yields $x = max(a_{ij})$.

1.8.3 Matrix functions, based on svd

- Type `[V,D]=eig(m)` to get the eigenvalues of matrix m (stored in D) and the corresponding eigenvectors (stored in V).

- Type `[U,S,V]=svd(m)` to get the singular values of matrix m (stored in S) and the corresponding left and right singular vectors (respectively stored in U and V).

- Type `rank(m)` to compute the rank of matrix m. Remember, however, that this is not a very robust way to compute the rank. It is better to check the singular values.

- Type `det(m)` to compute the determinant of matrix m.

1.8.4 Elementary math functions

- Type `cos(m)` to get the cosine of all elements of matrix m.

- In the same manner, `sin, tan, exp, asin, acos, atan, log, sqrt, ...` can be used. Type `help elfun` to see all possible elementary functions.

1.9 Operations on the data stack

1.9.1 Viewing the current variables

- Type `who` to see the names of all defined variables.
- Type `whos` to see the names and sizes of all the variables.

1.9.2 Saving variables into a data file

- Type `save data` to save all the variables in the file *data.mat*.
- Type `save data m v` to save the variables m and v in the file *data.mat*.

1.9.3 Loading a data file

- Type `load data` to load the file *data.mat*.

1.9.4 Clearing variables

- Type `clear m v` to clear variables m and v.
- Type `clear` to clear all variables.

1.10 Graphics

1.10.1 Making a plot

- First make a vector x and a vector y with the same length. For example, type `x=[0:0.1:10];` and `y=sin(x);`
  Remark the typical Matlab procedure here: one can not define "continuous functions" in matlab but one has to create a discrete x-vector with a certain number of values. For each of these values one can compute the value of the function.
- Type `plot(x,y)` to plot the y-values against the x-values.
- Type `hold on` after a plot if you want the next plot be plotted together with the current plot.
  Type `hold off` to turn this off. Example: plot $\sin(x)$ and $\sin(2x)$ on the same plot by typing the following commands: `plot(x,sin(x)); hold on ; plot(x,sin(2*x))`.
- Type `hold off; plot(x,sin(x),'y'); hold on; plot(x,sin(2*x),'r');` to plot $\sin(x)$ in yellow and $\sin(2x)$ in red.
- Type `title('drawing')` to put the title *drawing* above your plot.
- Type `xlabel('position')` to indicate that each x-value corresponds to a *position*.
- Type `ylabel('sine-wave')` to indicate that the y-values correspond to a *sine–wave*.
- Type `grid` to make a grid on your plot.
- Type `axis([xmin,xmax,ymin,ymax])` to set the x-axis limits to $[xmin, xmax]$ and the y-axis limits to $[ymin, ymax]$.

- Type `semilogx(x,y)` to plot y against x. The only difference with `plot` is that a logarithmic scale (base 10) is used for the X-axis.
- Type `semilogy(x,y)` for a plot with a logarithmic scale for the Y-axis.
- Type `loglog(x,y)` for a plot with logarithmic scales for both the X- and Y-axes.
- `clf` clears the current figure.
- `[x,y]=ginput` gathers the x- and y-coordinates of data points that are entered by pressing a mouse button in the current plot until the return key is pressed.
- For more information and other plot commands, type `help plot`.

1.10.2 Zooming

- You can zoom in and out in the current figure. Type `zoom on` to enable zooming. Now, go to the current figure and press the left mouse button to zoom. The right mouse button is used to zoom out again.
- You can also zoom in on a rectangular part of the figure. To accomplish this, move the mouse while pressing the left mouse button.
- To turn off the zoom-feature, just type `zoom off`.

1.10.3 Handling figures

- If you want a new figure, type `figure`.
- When more than one figure is present and you want to change one of them, you have to tell Matlab which figure you want to work on. You can do so by typing `figure(`n`)` with the n the number of the figure you want to work on.
- If you want to delete a figure, type `close(`n`)` with n the number of the figure you want to delete.
- To delete all figures, type `close all`.

1.10.4 Using subplots

- You can put different drawings in one plot with the command `subplot`.
- Type `subplot(2,1,1)` to indicate that you want two plots vertically, one plot horizontally and select the first of the two. Type `plot(x,sin(x))` to plot $\sin(x)$ there.
- Type `subplot(2,1,2)` to indicate that you want to select the second of the two. Type `plot(x,sin(2*x))` to plot $\sin(2x)$ there. The result should look like figure 1.1.

1.10.5 How to print a screen plot

- Draw a plot as described above.
- If you have more than one figure on your screen, type `figure(n)` to select the n-th figure.
- Type `print -dps plotname.ps` to make a postscript file of your plot.

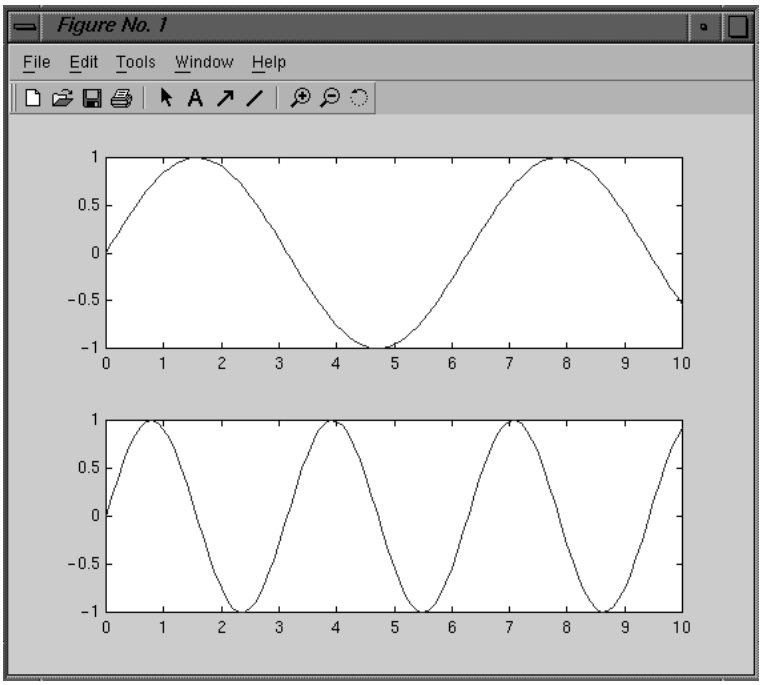

Figure 1.1: Two plots in one window, using `subplot`

1.11 M-files

A very important aspect of Matlab is the ability to create **M-files**. These are files, written by the user, enabling him to execute a number of commands sequentially without the need of typing them in at the Matlab-prompt. It is also possible to write functions. More details about this feature can be found in paragraph 1.12.

1.11.1 Creating an M-file

- Copy the file *ftp://ftp.esat.kuleuven.be/pub/sista/vanwaterschoot/alari/exercise_session0_MSc/example.m* to your directory for this session and open it. Or create a new m-file *example.m* and copy the following code:
```
x=[0:0.01:10];
y=sin(5*x);
plot(x,y);
title('sine wave');
```

- start the M-file by typing `example` at the Matlab prompt.

- At this point, if you want to change the scale of x, the frequency of the sine-wave, the title, ..., you only have to change it in the M-file, save it and run it again. Example: Change x into `x=[0:0.01:3]`. Save and run it.

- Remark the difference between a **M-file** and a **Mat-file**. The former is a file which holds a "program" which can be executed in Matlab. The latter contains "data" which can be loaded and saved.

1.11.2 Asking for input

- You can ask the user for input with the command `input`;
- Type `n = input('Give in an odd number');` in an m-file. This will ask for an odd number and assign the result to n.

1.12 Functions

The biggest drawback in using M-files is the fact that you can't parametrize your code except by asking for input which stops the process until the user responds. To alleviate this problem, you can use **functions**.

- Copy the file *ftp://ftp.esat.kuleuven.be/pub/sista/vanwaterschoot/alari/exercise_session0_MSc/examplefunc.m* to your directory.
- Open the file.
- This is an example of a function file for Matlab. It contains the following code:

```
function [t,y] = examplefunc(freq)
freq_sample=10*freq;
t=[0:1/freq_sample:0.1];
y=sin(freq*2*pi*t);
plot(t,y);
title('sine wave');
```

- Call the function by typing `examplefunc(100)`. The function will then execute. First it computes the sampling frequency as 5 times the Nyquist frequency to get a nice plotting result. Then t and y are generated and the plot is made.
- The function also gives a return value. This can be used to call the function in a slightly different way: Type `[T Y] = examplefunc(40)`. The t and y variables are then stored in the (global) variables T and Y.

1.13 Control structures

Matlab knows about different control structures. You can compare them to their counterparts in any programming language (like C, C++, Pascal, Basic, ...).

1.13.1 The "for" loop

- This control structure is ideal to execute a certain command or set of commands a fixed number of times.
- If we want to make the following vector for instance

$$v = \begin{pmatrix} 1 \\ 1/2 \\ 1/4 \\ 1/8 \end{pmatrix}$$

we type

```
for i=0:3,
v(i+1) = (1/2)^i;
end
```

1.13.2 The "while" loop

- This control structure is ideal to execute a certain command or set of commands until a condition is fulfilled.
- If we want to create the vector v of the previous example but keep adding elements until they are smaller than 10^{-4} we type

```
a = 1;
i = 0; while(a > 1e-4),
a = (1/2)^i;
v(i+1) = a;
i = i+1;
end
```

1.13.3 The "if" statement

- There are times when you want your code to make a decision. For example you want to accept only an odd number and reject an even number. For this the **if** statement is used.
- The syntax is easy:

```
if(statement),
command to be executed if statement is non-nil;
else
command to be executed if statement is nil;
end
```

- For the above mentioned example the code would be:

```
n = input('Give in an odd number:  ');
if (rem(n,2))
disp('Thank you');
else
disp('This is not an odd number !!!');
end
```

- use `==`, `~=`, `>=`, `<=`, `>`, `<`, `&&`, `||` to denote the logic operators $=, \neq, \geq, \leq, >, <$, and, or.

1.13.4 Pre-allocation

If you use a matrix or vector variable, and do not know its content beforehand, it is often a good idea to pre-allocate this matrix as an all-zero matrix (using the command `zeros`). Often it is tempting to fill a matrix like in this example:

```
received_bits=[];
```

```
for k=1:number_of_segments
received_bits=[received_bits bitsegment(k,:)];
end
```

This is easy since you don't have to think about what size `received_bits` will have after all bitsegments are added, and the code is simple. However, this kind of programming significantly slows down the execution time of the code, because Matlab must copy the current version of `received_bits` to a new version of `received_bits` at another place in the memory. The following code does the same and has faster execution time:

```
segmentlength=size(bitsegment,2);
received_bits=zeros(1,number_of_segments*segmentlength);
for k=1:number_of_segments
received_bits((k-1)*segmentlength+1:k*segmentlength)=bitsegment(k,:);
end
```

1.14 Useful commands

Here is a list of some useful, not yet mentioned commands that you might need during the project. Use the `help` command to know how they work. Re-read this list regularly to fresh up your memory. It can save you a lot of implementation time!

- `rem` (rest after division, i.e. modulo operator)
- `real`, `imag` (real part and imaginary part of complex numbers)
- `conj` (takes conjugate of complex numbers)
- `angle` (return phase angles of complex numbers in radians)
- `conv` (convolution)
- `toeplitz` (create a toeplitz-matrix, handy to represent convolutions as matrix multiplications)
- `fft`, `ifft` (FFT and inverse FFT)
- `fftfilt` (filter a signal with a given filter in an efficient way)
- `rand`,`randn` (create uniform random noise, create Gaussian random noise). Note: this function always uses the same seed when Matlab starts up. This means that the same noise string will be produced when you restart Matlab. If you want a time dependant seed, use the `help` command for more information.
- `find` (find all elements in a matrix that satisfy a certain logic expression, e.g. $x > 0$)
- `round, ceil, floor` (make integer value out of non-integer value)
- `reshape` (change the shape of a matrix or vector)
- `spectrogram` (create a spectrogram of a given input signal. A spectrogram shows information on the frequency content of the signal over time. Carefully consider the parameters you use in this command!)
- `break` (go out of a loop immediately)

- `norm` (calculate any norm of a vector or matrix)
- `image` (display a matrix as an image, where the matrix values define the color of each pixel)
- `imagesc` (ditto, but now the matrix elements are scaled first to use the full color map. See also `help colormap`)
- `colorbar` (add a colorbar to the image to visualize the mapping between colors and numerical values)
- `soundsc` (plays an audio signal, and scales it to use full dynamic range)
- `tic,toc` (determining the execution time of Matlab-code inbetween tic ... toc)

Chapter 2

Simulink

2.1 What is Simulink ?

Simulink, a companion program to Matlab, is an interactive system for simulating linear and nonlinear dynamic systems. It is a graphical mouse-driven program that allows you to model and simulate a system by drawing a block diagram on the screen and manipulating it dynamically. It can work with linear, nonlinear, continuous-time, discrete-time, multivariable, and multirate systems.
During the exercise sessions for this course, you will probably only make use of a small part of the possibilities of Simulink, namely the real-time audio playing and recording. Therefore, this tutorial is very short. If you want to learn more about all features of Simulink, we refer to tutorials on the WWW.

2.2 Getting started

- Before you start, close all applications that allocate a lot of colors (e.g., web browsers).
- Start a Matlab-session. Type `simulink` at the Matlab prompt to start the Simulink toolbox.
- A window will appear on your screen: the **library** window.
- To start a worksheet select **New** → **Model** in the **File** menu or press <CONTROL>-n.

2.3 The library

- The default library, which is always opened when Simulink is started, consists of eight sub-libraries: **Sources, Sinks, Continuous, Discrete, Math, Functions and Tables, Nonlinear, Signals and Systems** and **Blocksets & Toolboxes**.
- By double-clicking on any sub-library, you open it and gain access to the blocks of that sub-library. For example, in the **Sources** sub-library, you can find often-used blocks like **Constant, Signal Generator, Pulse Generator** and **Clock**.
- It's also possible to load other (self-made) libraries.

2.4 The worksheet

2.4.1 Entering new blocks

- To put a block from the library into your worksheet, put the pointer on the block in the library window. Push the left mouse button and keep it down. Drag the block to your worksheet and release the mouse button.
- To change the parameters of a block, double-click on it with the left mouse button. A window pops up in which you can fill out the parameter fields. Click **OK** when ready. It is also possible to specify a variable-name instead of a value. The variable must be specified in your Matlab workspace before running the simulation. This can be very useful to try different initial conditions and parameters.
- If you're not sure how the block works or what all parameters do, click on the **Help** button. This will open a Netscape window with help for this specific block.

2.4.2 Resizing and moving blocks

- First select the block by clicking with the left mouse button. In every corner of the block a little filled square appears. Put the pointer on one of the corners , press the left mouse button and keep it down. Move the mouse and release the mouse button when the block has the desired size.
- To move a block, put the pointer inside it, click the left mouse button, hold it down, drag the block to its new position and release the mouse button.

2.4.3 Copying, removing, rotating and flipping blocks

- Position the pointer on the block to be copied. Click the right mouse button and keep it down. Move the mouse to the desired position of the copy of the block and release the mouse button. The new block will have the same properties as the original block. The name of the block is copied too but a "1" is added at the end.
- To remove a block, click on it once with the left mouse button to select it and press <CONTROL>-X or the <BACKSPACE> key. If this doesn't work, use **Cut** button in the **Edit** menu instead.
- To rotate a block, select it and press <CONTROL>-R. If this doesn't work, use the **Rotate Block** button in the **Format** menu.
- To flip a block (i.e. rotate it 180 degrees), select it and press <CONTROL>-F. If this doesn't work, use the **Flip Block** button in the **Format** menu.

2.4.4 Connecting blocks

- Put the mouse pointer on one of the outputs of the source block (indicated by a small triangle). Press the left mouse button and keep it down. Move the pointer towards the target block. A straight, dashed line will be drawn. If you release the mouse button an arrow will appear at the end of the line. From here you can start a new line. If you reach the input of the target block and release the mouse button, a bold arrow will appear to indicate that the connection is complete.

- For example: add a Sine-Wave (Sources library) and a Scope block Ssinks library) and connect them. The worksheet window should look like figure 2.1.

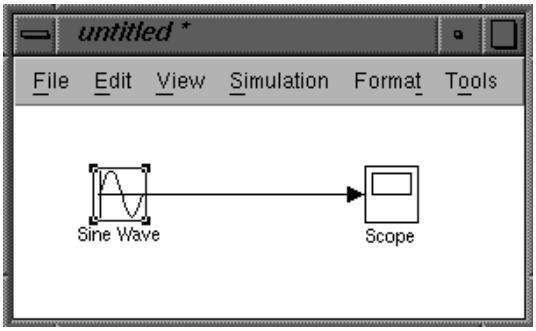

Figure 2.1: Sine wave + scope

- To branch from existing connections (for example to make multiple connections from one source block) use the right mouse button.

2.4.5 Grouping blocks

- You can group a set of blocks that belong together to make your worksheet more surveyable. Select the blocks you want to group. You can do so by pressing the left mouse button and keeping it down; then move the mouse: a rectangle will appear. When you release the mouse button all blocks inside the rectangle are selected. To group the selected blocks, press <CONTROL>-G or use **Create Subsystem** in the **Edit** menu. A subsystem is now created. If you want to see what's inside it, double-click on it. A new window will appear with the contents of the subsystem. Try this out on the system of figure 2.1.

2.4.6 Saving and loading block diagrams

- Use the **Save As** topic in the **File** menu to save your worksheet. A window will pop up. In the **Selection** information field you can specify the name of the file. Since the block diagram is saved as an "mdl" file, the extension `.mdl` will automatically be added if you do not specify it yourself. Click on **Done** to save your block diagram.
- From now on you can save this worksheet under the same name by using **Save** in the **File** menu. We urge you to do this very frequently.
- Use **Close** in the **File** menu to close the worksheet window.
- Use **Open** in the **File** menu to load a previously saved worksheet.

2.4.7 Simulation parameters

- A very important menu item is the **Parameters** item in the **Simulation** menu. If you click on it, a new window will appear in which you can specify all important simulation parameters.
- Among the parameters which you can alter are **Start-** and **Stop time**, **Maximum** and **Initial step size** and **Relative** and **Absolute tolerance**.

- Setting the correct parameters is not an easy task, and mainly depends on the application you want to simulate. The correct values will be given for each exercise. If not all parameters are specified, it is assumed that the non-mentioned parameters take their default value, which is the value they get when Simulink is first started.

2.4.8 Simulating

- Now we are ready to really start simulating a system. Let's take the example of the Sine-Wave connected to the Scope.
- Click twice on the Scope to open it.
- Set the stop-time of the simulation to 40 and the Max step size to 1e-2 in the **Simulation parameters** window.
- Start the simulation in the **Simulation** menu or click <CONTROL>-t.
- When the simulation is finished, you'll hear a beep.
- You can pause the simulation and later continue or stop it. Just click on the appropriate item in the **Simulation** menu.

2.4.9 The Scope block

- This block creates a plot of its input. If you want to plot two or more different signals in the same scope you have to combine the signals with a **Mux** block (Signal and Systems library) and connect the output of this block to the input of the scope.
- With the first three buttons on the left (showing a looking-glass) you can perform different zooming actions on the current plotted figure.
- The button with the binoculars "autoscales" the figure
- The fifth button on the left allows you to save the current axis settings.
- The second button from the right can be used to change some properties of the scope. The right button is for printing.
- If you right-click on the y-axis you can explicitly set its properties.