Introduction to Matlab

Insert of a Dürer painting. The image is filled with mathematical symbolism. This matrix is known as a magic square and was believed by many in Dürer's time to have genuinely magical properties. It does turn out to have some fascinating characteristics worth exploring.
# Introduction to Matlab

## Content:

1. **What is MATLAB?** 3
2. **Getting Started** 3
   - 2.1 Lock-in CLAM V 4
   - 2.2 Remote Lock in CLAM V 4
   - 2.3 Matlab on local PCs 4
3. **Getting Help** 4
4. **MATLAB Variables, Scalars, Vectors and Matrices** 6
   - 4.1 Complex numbers 6
   - 4.2 Generating a vector 6
   - 4.3 Accessing vector elements 7
5. **Matrix operations** 7
   - 5.1 Arithmetic matrix operations 7
   - 5.2 Relational operations 11
   - 5.3 Flow control operations 12
6. **Mathematical functions** 12
   - 6.1 Basic mathematical functions 12
   - 6.2 Special mathematical functions 12
7. **MATLAB Files** 20
   - 7.1 M-Files 20
     - 7.1.1 Scripts 20
     - 7.1.2 Functions 21
   - 7.2 Mat-Files 21
8. **Plotting** 22
   - 8.1 Basic Plotting Functions 22
     - 8.1.1 Auto Scaling 23
     - 8.1.2 Adding Plots to existing Graphs 23
     - 8.1.3 Plotting only data points 23
     - 8.1.4 Line Plots of Matrix Data 23
     - 8.1.5 Plotting Imaginary and Complex Data 24
     - 8.1.6 Display multiple plots per figure 24
  - 8.2 Discrete Data Graphs 25
    - 8.2.1 Stem 25
    - 8.2.2 Stairs 25
  - 8.3 Basic Printing and Exporting 26
    - 8.3.1 Linux 26
    - 8.3.2 Windows 27
1. What Is MATLAB?

MATLAB is a programming language and data visualization software package, which is very effective and powerful in signal and image processing, communications, and control systems. We will in particular focus on the application of Matlab in the area of signals and systems. The document is a short introduction to MATLAB with respect to the Signals and Systems Lab Course (Advanced Electrical Engineering Lab Course I) at Jacobs University Bremen.

The following primer will offer the first technical instructions to use MATLAB. No prior knowledge of MATLAB is necessary. In addition to this primer you will find other information like the extensive MATLAB documentation or several websites (e.g. http://www.csb.yale.edu/userguides/datamanip/matlab/help/techdoc/basics/gettingtoc.html). Furthermore, MATLAB books addressing the special topic of signal and signals will be available in the Library soon.

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you 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 non-interactive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB uses software developed by the LAPACK and ARPACK projects, which together represent the state-of-the-art in software for matrix computation.

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.

2. Getting Started

There are three ways how you can access Matlab, Release 13 at the International University Bremen. Matlab Release 13 is installed on all CLAMV PCs (30 licenses). All students have direct access to Clam V. All students can use their Notebook as a remote terminal to access Matlab on the CLAM V PCs. The third option is only available for the students, who participate in the Advanced Electrical Engineering Lab Course. All students will be provided with a classroom license of Matlab Release 13.
2.1 Lock in on CLAM V

As part of the computer teaching Lab (CALM V) 26 computers are equipped with MATLAB. Students can lock in on each of the machines. Matlab is available on each machine.

2.2 Remote Lock in CLAM V

In general it is possible and allowed for all students to lock in on the PCs in CLAM V from their Notebooks.

However, Jacobs University will not provide you with SSH and/or putty. It is your responsibility to install software on your machine, which is licensed or free/shareware. Jacobs University will take no responsibility for illegal software installed on the Notebooks of the students.

2.3 Matlab on local PCs

All students, who participate in the Advanced Electrical Engineering Lab Course will be provided with a classroom version of MATLAB, Release 13, Simulink, the Symbolic Tool box and the Toolboxes for Signal Processing, Image Processing and Control systems.

In order to start MATLAB on a local PC a LAN/internet connection is necessary. A floating license server will check whether all of the 30 licenses are in use or not.

The PLP (Product License Password), which is necessary to install the classroom version on the local PCs, will not be distributed.

3 Getting Help

If you need help and you know the name of the function you are looking for you can use the command ‘help’ in the command window.

>> help functionname

This command displays a description of the function and generally also includes a list of related functions. If you cannot remember the name of the function, you can use the ‘lookfor’ command and the name of some keywords associated to the keyword you are looking for

>> lookfor keyword

However, often it is better to directly go to the ‘question mark’ or to the ‘Help icon’ in the Launch Pad. From there you can easily go to the Matlab function reference and the alphabetical list of the available functions. However, the list includes only the functions, which are provided by the standard Matlab package. The functions, which
are available by the toolboxes, are not listed there. All toolboxes have their own help functions including an alphabetic reference list.

Further help can be found by typing ‘info’ or ‘helpwin’ in the command window.

Famous ‘Mexican hat’. The ‘Mexican hat’ based on a 2D sinc(x)=\sin(x)/x function.
4. MATLAB Variables, Scalars, Vectors and Matrices

MATLAB stores variables in the form of matrices, which are M x N, where M is the number of the rows and N the number of columns. A variable is therefore always a matrix of a two-dimensional array of real or complex numbers. A 1 x 1 matrix is a scalar; a 1 x N matrix is a row vector, and an M x 1 matrix is a column vector. All elements of a matrix can be real or complex numbers: sqrt(-1) can be written as either ‘i’ or ‘j’ assuming the user does not redefine them. A matrix is written with a square bracket ‘[ ]’ with spaces separating adjacent columns and semicolons separating adjacent rows. For examples, consider the following assignments of the variable x:

Real scalar >> x = 5
Complex scalar >> x = 5+10j (or x = 5+10i)
Row vector >> x = [1 2 3] (or [1, 2, 3])
Column vector >> x = [1; 2; 3]
3 x 3 matrix >> x = [1 2 3; 4 5 6; 7 8 9]

There are a few notes for caution. Complex elements of a matrix should not be typed with spaces, i.e. ‘-1+2j’ is fine as a matrix element, ‘-1 + 2j’ is not. Also, ‘-1+2j’ is interpreted correctly whereas ‘-1+j2’ is not (MATLAB interprets the ‘j2’ as the name of the variable. You can also write ‘1+j*2’.

Furthermore, it is important to know that informally (documentation of MATLAB), the terms matrix and array are often used interchangeably.

4.1 Complex Numbers

Some of the important operations are listed here, which are used to operate or modify complex numbers.

Complex scalar >> x = 3+4j
Real part of a complex scalar >> real(x) ⇒ 3
Imaginary part of a complex scalar >> imag(x) ⇒ 4
Magnitude or Amplitude of a complex scalar >> abs(x) ⇒ 5
Angle or phase of a complex scalar >> angle(x) ⇒ 0.9273
Complex conjugation of the complex scalar >> Conj(x) ⇒ 3-4j

4.2 Generating a vector

Vectors can be created by using the ‘:’ command. For example, to generate a vector x that takes on the values 0 to 10 in increments of 0.5, type the following, which generates a 1 x 21 matrix.

>> x = [0:0.5:10]

In general terms it can be described as:
Other ways to generate vectors include the commands: ‘linspace’ which generates a vector by specifying the first and the last number of equally spaced entries between the first and the last number.

\[ t = \text{linspace}[0, 5, 100] \]

which would correspond to a 1 x 100 matrix, with the values of 0 to 5 with an equally spacing of 0.02. In general terms

\[ \text{vector} = \text{linspace}'\text{start value}', 'stop value', 'number of linear points in between'\]

The command ‘logspace’ is available for the creation of matrices with a logarithmic spacing between the points.

### 4.3 Accessing vector elements

Elements of a matrix are accessed by specifying the row and the column. For example, in the matrix specified by \( A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \), the element in the first row and the third column can be accessed by writing

\[ x = A(1,3) \]

which yields 3

The entire second row can be accessed with

\[ y = A(2,:) \]

which yields \([4 \ 5 \ 6]\)

where the ‘:’ here means “take all the entries in the column”. A submatrix of \( A \) consisting of row 1 and 2 and all three columns is specified by

\[ A(1:2) \]

which yields \([1 \ 2 \ 3; 4 \ 5 \ 6]\)

### 5 Matrix operations

In following a few important matrix operations are listed.

#### 5.1 Arithmetic matrix operations

Addition and subtraction of matrices is defined just as it is for arrays, element-by-element. Adding \( A \) to \( B \) and then subtracting \( A \) from the result recovers \( B \). We start with

\[ A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{bmatrix} \]

\[ A = \]

\[ A = \]
\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 2 & 3 \\
1 & 3 & 6 \\
\end{bmatrix}
\]

and

\[
\begin{bmatrix}
8 & 1 & 6 \\
3 & 5 & 7 \\
4 & 9 & 2 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 2 & 3 \\
1 & 3 & 6 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
8 & 1 & 6 \\
3 & 5 & 7 \\
4 & 9 & 2 \\
\end{bmatrix}
\]

\[
\begin{align*}
\text{>> } & B = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix} \\
B & = \\
& \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix} \\
\text{>> } & X = A + B \\
yields \\
X & = \\
& \begin{bmatrix} 9 & 2 & 7 \\ 4 & 7 & 10 \\ 5 & 12 & 8 \end{bmatrix} \\
\text{and} \\
\text{>> } & Y = X - A \\
yields \\
Y & = \\
& \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}
\]

Addition and subtraction requires both matrices to have the same dimension, or one of them be a scalar. If the dimensions are incompatible, an error results.

\[
\begin{align*}
\text{>> } & X = A + [1; 2; 3] \\
\text{Error using } & => + \\
\text{Matrix dimensions must agree.}
\end{align*}
\]

Addition and subtraction involves element-by-element arithmetic operations; matrix multiplication and division does not involve element-by-element arithmetic operations.

\[
\begin{align*}
\text{>> } & C = A*B \\
yields \\
C & = \\
& \begin{bmatrix} 15 & 15 & 15 \\ 26 & 38 & 26 \end{bmatrix}
\]
As a consequence it is obviously clear that

\[ C = A \cdot B \neq B \cdot A \]

\[ \text{>>} C = B \cdot A \]

yields

\[
\begin{bmatrix}
15 & 28 & 47 \\
15 & 34 & 60 \\
15 & 28 & 43 \\
\end{bmatrix}
\]

Matrix multiplication: \( C = A \cdot B \) is the linear algebraic product of the matrices \( A \) and \( B \). More precisely,

\[
C(i, j) = \sum_{k=1}^{n} A(i, k) \cdot B(k, j)
\]

For non-scalar \( A \) and \( B \), the number of columns of \( A \) must equal the number of rows of \( B \). A scalar can multiply a matrix of any size.

The division is described by

\[
/ \quad \text{right division} \\
\backslash \quad \text{left division}
\]

where

\[
/
\]

Slash or matrix right division. \( B/A \) is roughly the same as \( B \cdot \text{inv}(A) \). More precisely, \( B/A = (A' \cdot B')' \).

\[
\backslash
\]

Backslash or matrix left division. If \( A \) is a square matrix, \( A \backslash B \) is roughly the same as \( \text{inv}(A) \cdot B \), except it is computed in a different way. If \( A \) is an \( n \)-by-\( n \) matrix and \( B \) is a column vector with \( n \) components, or a matrix with several such columns, then \( X = A \backslash B \) is the solution to the equation \( AX = B \) computed by Gaussian elimination (see "Algorithm" for details). A warning message prints if \( A \) is badly scaled or nearly singular.

If \( A \) is an \( m \)-by-\( n \) matrix with \( m \neq n \) and \( B \) is a column vector with \( m \) components, or a matrix with several such columns, then \( X = A \backslash B \) is the solution in the least squares sense to the under- or over-determined system of equations \( AX = B \). The effective rank, \( k \), of \( A \), is determined from the QR decomposition with pivoting (see "Algorithm" for details). A solution \( X \) is computed which has at most \( k \) nonzero components per
column. If \( k < n \), this is usually not the same solution as \( \text{inv}(A)\cdot B \), which is the least squares solution with the smallest norm, \( ||X|| \).

\[^{\wedge}\]

Matrix power. \( X^p \) is \( X \) to the power \( p \), if \( p \) is a scalar. If \( p \) is an integer, the power is computed by repeated multiplication. If the integer is negative, \( X \) is inverted first. For other values of \( p \), the calculation involves eigenvalues and eigenvectors, such that if \( [V,D] = \text{eig}(X) \), then \( X^p = V^pD^pV \).

If \( x \) is a scalar and \( P \) is a matrix, \( x^P \) is \( x \) raised to the matrix power \( P \) using eigenvalues and eigenvectors. \( X^P \), where \( X \) and \( P \) are both matrices, is an error.

Matrix transpose: \( A' \) is the linear algebraic transpose of \( A \). For complex matrices, this is the complex conjugate transpose.

Besides the addition and subtraction MATLAB provides the following functions for element-by-element:

\(^{.*}\)

Array multiplication: \( A.*B \) is the element-by-element product of the arrays \( A \) and \( B \). \( A \) and \( B \) must have the same size, unless one of them is a scalar.

\[
A = \\
\begin{bmatrix}
1 & 1 & 1 \\
1 & 2 & 3 \\
1 & 3 & 6
\end{bmatrix}
\]

and

\[
B = [8 \ 1 \ 6; \ 3 \ 5 \ 7; \ 4 \ 9 \ 2]
\]

\[
X = A.*B = B.*A
\]

yields

\[
X = \\
\begin{bmatrix}
8 & 1 & 6
\end{bmatrix}
\]
Array right division: \( A./B \) is the matrix with elements \( A(i,j)/B(i,j) \). \( A \) and \( B \) must have the same size, unless one of them is a scalar.

Array left division. \( A./B \) is the matrix with elements \( B(i,j)/A(i,j) \). \( A \) and \( B \) must have the same size, unless one of them is a scalar.

Array power. \( A.^B \) is the matrix with elements \( A(i,j) \) to the \( B(i,j) \) power. \( A \) and \( B \) must have the same size, unless one of them is a scalar.

Array transpose. \( A.' \) is the array transpose of \( A \). For complex matrices, this does not involve conjugation.

5.2 Relational operations

The following relational operations are defined:

- `<`  
  less than
- `<=`  
  less than or equal to
- `>`  
  greater than
- `>=`  
  greater than or equal to
- `==`  
  equal to
- `~=`  
  not equal to

Relational operators perform element-by-element comparisons between two arrays. They return an array of the same size, with elements set to logical true (1) where the relation is true, and elements set to logical false (0) where it is not.

The operators `<` , `>` and use only the real part of their operands for the comparison. The operators `==` and `~=` test real and imaginary parts.

**Examples:**

If one of the operands is a scalar and the other a matrix, the scalar expands to the size of the matrix. For example, the two pairs of statements:

\[
X = 5; X >= [1 2 3; 4 5 6; 7 8 10] \\
X = 5*ones(3,3); X >= [1 2 3; 4 5 6; 7 8 10]
\]

produce the same result:
ans =
  1  1  1
  1  1  0
  0  0  0

5.3 Flow control operations

MATLAB contains the usual set of flow control structures, e.g., for, while, and if, plus
the logical operators, e.g., & (and), | (or), and ~ (not).

6 Mathematical functions

MATLAB comes with a large number of built-in functions that operate on matrices on
an element-by-element basis. These include:

6.1 Basic mathematical functions

Alphabetic list of important functions, which are available as part of the standard
MATLAB package:

sin    sine
cos    cosine
tan    tangent
asin   inverse sine
acos   inverse cosine
atan   inverse tangent
exp    exponential
log    natural logarithm
log10  common logarithm
sqrt   square root
sign   signum

6.2 Special mathematical functions

Alphabetic list of important functions, which are available as part of the signal
processing toolbox.

Abs
Absolute value (magnitude).

Syntax
y = abs(x)
Description: \( y = \text{abs}(x) \) returns the absolute value of the elements of \( x \). If \( x \) is complex, \( \text{abs} \) returns the complex modulus (magnitude).

\[
\text{abs}(x) = \sqrt{\text{real}(x)^2 + \text{imag}(x)^2}
\]

The abs function is part of the standard MATLAB language and the signal processing toolbox.

**Example:**
Calculate the magnitude of the FFT of a sequence.

\[
t = (0:99)/100; \\
x = \sin(2\pi*15*t) + \sin(2\pi*40*t); \\
y = \text{fft}(x); \\
m = \text{abs}(y);
\]

---

\textbf{angle}

Phase angle

**Syntax**

\[
p = \text{angle}(h)
\]

**Description:** \( p = \text{angle}(h) \) returns the phase angles, in radians, of the elements of complex vector or array \( h \). The phase angles lie between \(-\pi\) and \(\pi\).

For complex sequence \( h = x + iy \), the magnitude and phase are given by

\[
m = \text{abs}(h) \\
p = \text{angle}(h)
\]

To convert to the original \( h \) from its magnitude and phase, type

\[
i = \sqrt{-1} \\
h = m.*\exp(i*p)
\]

The angle function is part of the standard MATLAB language and the signal processing toolbox.

**Example:**
Calculate the phase of the FFT of a sequence.

\[
t = (0:99)/100; \\
x = \sin(2\pi*15*t) + \sin(2\pi*40*t); \\
y = \text{fft}(x); \\
p = \text{unwrap}(%
\]

---

\textbf{gauspuls}

Generate a Gaussian-modulated sinusoidal pulse.

**Syntax**

\[
yi = \text{gauspuls}(t,fc,bw)
\]
yi = gauspuls(t,fc,bw,bwr)
[yi,yq] = gauspuls(...)
[yi,yq,ye] = gauspuls(...)
tc = gauspuls('cutoff',fc,bw,bwr,tpe)

Description: gauspuls generates Gaussian-modulated sinusoidal pulses.

yi = gauspuls(t,fc,bw) returns a unity-amplitude Gaussian RF pulse at the times indicated in array t, with a center frequency fc in hertz and a fractional bandwidth bw, which must be greater than 0. The default value for fc is 1000 Hz and for bw is 0.5.

yi = gauspuls(t,fc,bw,bwr) returns a unity-amplitude Gaussian RF pulse with a fractional bandwidth of bw as measured at a level of bwr dB with respect to the normalized signal peak. The fractional bandwidth reference level bwr must be less than 0, because it indicates a reference level less than the peak (unity) envelope amplitude. The default value for bwr is -6 dB.

[yi,yq] = gauspuls(...) returns both the in-phase and quadrature pulses.

[yi,yq,ye] = gauspuls(...) returns the RF signal envelope.

tc = gauspuls('cutoff',fc,bw,bwr,tpe) returns the cutoff time tc (greater than or equal to 0) at which the trailing pulse envelope falls below tpe dB with respect to the peak envelope amplitude. The trailing pulse envelope level tpe must be less than 0, because it indicates a reference level less than the peak (unity) envelope amplitude. The default value for tpe is -60 dB.

Examples:
Plot a 50 kHz Gaussian RF pulse with 60% bandwidth, sampled at a rate of 1 MHz. Truncate the pulse where the envelope falls 40 dB below the peak.

tc = gauspuls('cutoff',50e3,0.6,[],-40);
t = -tc : 1e-6 : tc;
yi = gauspuls(t,50e3,0.6);
plot(t,yi)

pulstran
Generate a pulse train.

Syntax
y = pulstran(t,d,'func')
y = pulstran(t,d,'func',p1,p2,...)
y = pulstran(t,d,p,fs)
y = pulstran(t,d,p)

Description: pulstran generates pulse trains from continuous functions or sampled prototype pulses.
y = pulstran(t,d,'func') generates a pulse train based on samples of a continuous
function, 'func', where 'func' is:

'gauspuls', for generating a Gaussian-modulated sinusoidal pulse
'recpuls', for generating a sampled aperiodic rectangle
'tripuls', for generating a sampled aperiodic triangle

pulstran is evaluated length(d) times and returns the sum of the evaluations
y = func(t-d(1)) + func(t-d(2)) + ...

The function is evaluated over the range of argument values specified in array t, after
removing a scalar argument offset taken from the vector d. Note that func must be a
vectorized function that can take an array t as an argument.

An optional gain factor may be applied to each delayed evaluation by specifying d as
a two-column matrix, with the offset defined in column 1 and associated gain in
column 2 of d. Note that a row vector will be interpreted as specifying delays only.

pulstran(t,d,'func',p1,p2,...) allows additional parameters to be passed to 'func' as
necessary.
For example,
func(t-d(1),p1,p2,...) + func(t-d(2),p1,p2,...) + ... pulstran(t,d,p,fs) generates a pulse
train that is the sum of multiple delayed interpolations of the prototype pulse in vector
p, sampled at the rate fs, where p spans the time interval [0,(length(p)-1)/fs], and its
samples are identically 0 outside this interval. By default, linear interpolation is used
for generating delays.

pulstran(t,d,p) assumes that the sampling rate fs is equal to 1 Hz.

pulstran(...,'func') specifies alternative interpolation methods. See interp1 for
a list of available methods.

Examples:
Example 1
This example generates an asymmetric sawtooth waveform with a repetition
frequency of 3 Hz and a sawtooth width of 0.1s. It has a signal length of 1s and a 1
kHz sample rate.

t = 0 : 1/1e3 : 1;            % 1 kHz sample freq for 1 sec
d = 0 : 1/3 : 1;               % 3 Hz repetition freq
y = pulstran(t,d,'tripuls',0.1,-1);
plot(t,y)

Example 2
This example generates a periodic Gaussian pulse signal at 10 kHz, with 50%
bandwidth. The pulse repetition frequency is 1 kHz, sample rate is 50 kHz, and pulse
train length is 10 msec. The repetition amplitude should attenuate by 0.8 each time.

t = 0 : 1/50E3 : 10e-3;
d = [0 : 1/1E3 : 10e-3 ; 0.8.^((0:10))];
y = pulstran(t,d,'gauspuls',10e3,0.5);
plot(t,y)

rectpuls
Generate a sampled aperiodic rectangle.

Syntax
y = rectpuls(t)
y = rectpuls(t,w)

Description: y = rectpuls(t) returns a continuous, aperiodic, unity-height rectangular pulse at the sample times indicated in array t, centered about t = 0 and with a default width of 1. Note that the interval of non-zero amplitude is defined to be open on the right, that is, rectpuls(-0.5) = 1 while rectpuls(0.5) = 0.

y = rectpuls(t,w) generates a rectangle of width w.

rectpuls is typically used in conjunction with the pulse train generating function pulstran.

sawtooth
Generate a sawtooth or triangle wave.

Syntax
x = sawtooth(t)
x = sawtooth(t,width)

Description: sawtooth(t) generates a sawtooth wave with period 2 for the elements of time vector t. sawtooth(t) is similar to sin(t), but creates a sawtooth wave with peaks of -1 and 1 instead of a sine wave. The sawtooth wave is defined to be -1 at multiples of 2 and to increase linearly with time with a slope of 1/ at all other times.

sawtooth(t,width) generates a modified triangle wave where width, a scalar parameter between 0 and 1, determines the point between 0 and 2 at which the maximum occurs. The function increases from -1 to 1 on the interval 0 to 2*width, then decreases linearly from 1 to -1 on the interval 2*width to 2. Thus a parameter of 0.5 specifies a standard triangle wave, symmetric about time instant with peak-to-peak amplitude of 1. sawtooth(t,1) is equivalent to sawtooth(t).

sinc
Sinc function.

Syntax
y = sinc(x)

Description: sinc computes the sinc function of an input vector or array, where the sinc function is
\[
\sin c(t) = \begin{cases} 
1, & t = 0 \\
\frac{\sin(\pi t)}{\pi t}, & t \neq 0
\end{cases}
\]

This function is the continuous inverse Fourier transform of the rectangular pulse of width \(2\pi\) and height 1.

\[
\sin c(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \exp(j\omega t) d\omega
\]

\(y = \text{sinc}(x)\) returns an array \(y\) the same size as \(x\), whose elements are the sinc function of the elements of \(x\).

Example
Perform ideal bandlimited interpolation by assuming that the signal to be interpolated is 0 outside of the given time interval and that it has been sampled at exactly the Nyquist frequency.

\[
t = (1:10)^\prime; \\
\text{randn('state',0);} \\
x = \text{randn(size(t));} \\
ts = \text{linspace(-5,15,600)^'}; \\
y = \text{sinc}(ts(:,\text{ones(size(t)))) - t(:,\text{ones(size(ts))))')*x;}
\]

\textbf{square}
Generate a square wave.

Syntax
\[
x = \text{square}(t) \\
x = \text{square}(t,duty)
\]

Description: \(x = \text{square}(t)\) generates a square wave with period 2 for the elements of time vector \(t\). \(\text{square}(t)\) is similar to \(\sin(t)\), but creates a square wave with peaks of \(\pm 1\) instead of a sine wave.

\(x = \text{square}(t,duty)\) generates a square wave with specified duty cycle, \(duty\). The duty cycle is the percent of the period in which the signal is positive.

\textbf{tripuls}
Generate a sampled aperiodic triangle.

Syntax
\[
y = \text{tripuls}(T) \\
y = \text{tripuls}(T,w) \\
y = \text{tripuls}(T,w,s)
\]
Description: \( y = \text{tripuls}(T) \) returns a continuous, aperiodic, symmetric, unity-height triangular pulse at the times indicated in array \( T \), centered about \( T=0 \) and with a default width of 1.

\( y = \text{tripuls}(T,w) \) generates a triangular pulse of width \( w \).

\( y = \text{tripuls}(T,w,s) \) generates a triangular pulse with skew \( s \), where \(-1 < s < 1\). When \( s \) is 0, a symmetric triangular pulse is generated.

\textbf{conj}
Symbolic conjugate.

\textbf{Syntax}
\( \text{conj}(X) \)

\textbf{Description}
\( \text{conj}(X) \) is the complex conjugate of \( X \).

For a complex \( X \), \( \text{conj}(X) = \text{real}(X) - i \cdot \text{imag}(X) \).

\textbf{rand}
Uniformly distributed random numbers and arrays

\textbf{Syntax}
\( \text{Y} = \text{rand}(n) \)
\( \text{Y} = \text{rand}(m,n) \)
\( \text{Y} = \text{rand}([m \ n]) \)
\( \text{Y} = \text{rand}(m,n,p,...) \)
\( \text{Y} = \text{rand}([m \ n \ p...]) \)
\( \text{Y} = \text{rand(size(A))} \)
\( s = \text{rand('state')} \)

\textbf{Description}
The \text{rand} function generates arrays of random numbers whose elements are uniformly distributed in the interval \((0,1)\).

\( \text{Y} = \text{rand}(n) \) returns an \( n \)-by-\( n \) matrix of random entries. An error message appears if \( n \) is not a scalar.

\( \text{Y} = \text{rand}(m,n) \) or \( \text{Y} = \text{rand}([m \ n]) \) returns an \( m \)-by-\( n \) matrix of random entries.

\( \text{Y} = \text{rand}(m,n,p,...) \) or \( \text{Y} = \text{rand}([m \ n \ p...]) \) generates random arrays.

\( \text{Y} = \text{rand(size(A))} \) returns an array of random entries that is the same size as \( A \).

\text{rand}, by itself, returns a scalar whose value changes each time it's referenced.

\( s = \text{rand('state')} \) returns a 35-element vector containing the current state of the
uniform generator. To change the state of the generator:

rand('state',s)

Resets the state to s.

rand('state',0)

Resets the generator to its initial state.

rand('state',j)

For integer j, resets the generator to its j-th state.

rand('state',sum(100*clock))

Resets it to a different state each time.

Examples
Example 1: R = rand(3,4) may produce

R =
    0.2190  0.6793  0.5194  0.0535
    0.0470  0.9347  0.8310  0.5297
    0.6789  0.3835  0.0346  0.6711

This code makes a random choice between two equally probable alternatives.

    if rand < .5
      'heads' else
      'tails'
    end

Example 2: Generate a uniform distribution of random numbers on a specified interval [a,b]. To do this, multiply the output of rand by (b-a) then add a. For example, to generate a 5-by-5 array of uniformly distributed random numbers on the interval [10,50]

    a = 10; b = 50;
    x = a + (b-a) * rand(5)
    x =
      18.1106  10.6110  26.7460  43.5247  30.1125
      17.9489  39.8714  43.8489  10.7856  38.3789
      34.1517  27.8039  31.0061  37.2511  27.1557
      20.8875  47.2726  18.1059  25.1792  22.1847
      17.9526  28.6398  36.8855  43.2718  17.5861
7 MATLAB Files

MATLAB is a powerful programming language as well as an interactive computational environment. Files that contain code in the MATLAB language are called M-files.

7.1 M-Files

MATLAB is an interpretive language, i.e., commands typed at the MATLAB prompt are interpreted within the scope of the current MATLAB session. However, it is tedious to type in long sequences of commands each time MATLAB is used to perform a task. There are two means of extending MATLAB’s power — scripts and functions. Both make use of m-files (named because they have a .m extension and they are therefore also called dot-m files). The advantage of m-files is that commands are saved and can be easily modified without retyping the entire list of commands.

Scripts, which do not accept input arguments or return output arguments. They operate on data in the workspace.

Functions, which can accept input arguments and return output arguments. Internal variables are local to the function.

If you're a new MATLAB programmer, just create the M-files that you want to try out in the current directory. As you develop more of your own M-files, you will want to organize them into other directories and personal toolboxes that you can add to MATLAB’s search path.

If you duplicate function names, MATLAB executes the one that occurs first in the search path.

To view the contents of an M-file, for example, myfunction.m, use

```
type myfunction
```

7.1.1 Scripts

MATLAB script files are sequences of commands typed with an editor and saved in an m-file. A m-file can be created by using emacs (Linux) or notepad (Windows).

When you start a script, MATLAB simply executes the commands found in the file. Scripts can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, any variables that they create remain in the workspace, to be used in subsequent computations. In addition, scripts can produce graphical output using functions like plot.

For example, create a file called magicrank.m that contains these MATLAB commands.

```
% Investigate the rank of magic squares
r = zeros(1,32);
```
for n = 3:32
    r(n) = rank(magic(n));
end
r
bar(r)

Typing the statement *Magicrank* causes MATLAB to execute the commands, compute the rank of the first 30 magic squares, and plot a bar graph of the result. After execution of the file is complete, the variables n and r remain in the workspace.

Execution of the m-file is equivalent to typing the entire list of commands in the command window at the MATLAB prompt. All the variables used in the m-file are placed in MATLAB’s workspace. The workspace, which is empty when MATLAB is initiated, contains all the variables defined in the MATLAB session.

### 7.1.2 Functions

A second type of m-file is a function file which is generated with an editor exactly as the script file but it has the following general form:

```matlab
function [output 1, output 2] = functionname(input1, input2)
    %
    %[output 1, output 2] = functionname(input1, input2) Functionname
    %
    % Some comments that explain what the function does go here.
    %
    MATLAB command 1;
    MATLAB command 2;
    MATLAB command 3;

    The first line of a function M-file starts with the keyword function. It gives the function name and order of arguments. In this case, there are two input arguments and two output arguments.

    The next several lines, up to the first blank or executable line, are comment lines that provide the help text. These lines are printed when you type

    The first line of the help text is the H1 line, which MATLAB displays when you use the *lookfor* command or request help on a directory.

    The rest of the file is the executable MATLAB code defining the function.

    The name of the m-file for this function is *functionname.m* and it is called from the MATLAB command line or from another m-file by the following command

    ```matlab
    >> [output1, output2] = functionname(input1, input2)
    ```
```

### 7.2 Mat-Files

Mat-files (named because they have a .mat extension and they are therefore also called dotmat files) are compressed binary files used to store numerical results.
These files can be used to save results that have been generated by a sequence of MATLAB instructions. For example, to save the values of the two variables, variable1 and variable2 in the file named filename.mat, type

```matlab
>> save filename.mat variable1 variable2
```

Saving all the current variables in that file is achieved by typing

```matlab
>> save filename.mat
```

A mat-file can be loaded into MATLAB at some later time by typing

```matlab
>> load filename (or load filename.mat)
```

The documentation contains a lot of information how to create a mat-file by C++ or Fortran.

## 8 Plotting

### 8.1 Basic Plotting Commands

MATLAB provides a variety of functions for displaying vector data as line plots in 2D and 3D, as well as functions for annotating and printing these graphs. The following table summarizes the functions that produce basic line plots. These functions differ in the way they scale the plot's axes. Each accepts input in the form of vectors or matrices and automatically scales the axes to accommodate the data.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>plot</code></td>
<td>Graph 2-D data with linear scales for both axes</td>
</tr>
<tr>
<td><code>plot3</code></td>
<td>Graph 3-D data with linear scales for both axes</td>
</tr>
<tr>
<td><code>loglog</code></td>
<td>Graph with logarithmic scales for both axes</td>
</tr>
<tr>
<td><code>semilogx</code></td>
<td>Graph with a logarithmic scale for the x-axis and a linear scale for the y-axis</td>
</tr>
<tr>
<td><code>semilogy</code></td>
<td>Graph with a logarithmic scale for the y-axis and a linear scale for the x-axis</td>
</tr>
<tr>
<td><code>plotyy</code></td>
<td>Graph with y-tick labels on the left and right side</td>
</tr>
</tbody>
</table>

The `plot` function has different forms depending on the input arguments. For example, if `y` is a vector, `plot(y)` produces a linear graph of the elements of `y` versus the index of the elements of `y`. If you specify two vectors as arguments, `plot(x,y)` produces a graph of `y` versus `x`.

For example, these statements create a vector of values in the range \([0, 2]\) in increments of \(1/100\) and then use this vector to evaluate the sine function over that range. MATLAB plots the vector on the x-axis and the value of the sine function on the y-axis.
8.1.1 Auto scaling:

If it is not further specified MATLAB automatically selects appropriate axis ranges and tick mark locations.

The commands how to change the format of the graph is extensively described in the MATLAB documentation.

8.1.2 Adding Plots to an Existing Graph:

You can add plots to an existing graph using the `hold` command. When you set hold to on, MATLAB does not remove the existing graph; it adds the new data to the current graph, rescaling if the new data falls outside the range of the previous axis limits.

For example, these statements first create a semilogarithmic plot, then add a linear plot.

```matlab
semilogx(1:100,'+')
hold on
plot(1:3:300,1:100,'--')
hold off
```

While MATLAB resets the x-axis limits to accommodate the new data, it does not change the scaling from logarithmic to linear.

8.1.3 Plotting Only the Data Points

To plot a marker at each data point without connecting the markers with lines, use a specification that does not contain a line style. For example, given two vectors,

```matlab
x = 0:pi/15:4*pi;
y = exp(2*cos(x));
calling plot with only a color and marker specifier

plot(x,y,'r+')
```

plots a red plus sign at each data point.

8.1.4 Line Plots of Matrix Data

When you call the plot function with a single matrix argument

```matlab
plot(Y)
```

MATLAB draws one line for each column of the matrix. The x-axis is labeled with the row index vector, 1:m, where m is the number of rows in Y. For example,

```matlab
Z = peaks;
```
returns a 49-by-49 matrix obtained by evaluating a function of two variables. Plotting this matrix produces a graph with 49 lines.

8.1.5 Plotting Imaginary and Complex Data

When the arguments to plot are complex (i.e., the imaginary part is nonzero), MATLAB ignores the imaginary part except when plot is given a single complex argument. For this special case, the command is a shortcut for a plot of the real part versus the imaginary part. Therefore,

\[
\text{plot}(Z)
\]

where \( Z \) is a complex vector or matrix, is equivalent to

\[
\text{plot}(\text{real}(Z),\text{imag}(Z))
\]

For example, this statement plots the distribution of the eigenvalues of a random matrix using circular markers to indicate the data points.

\[
\text{plot(eig(randn(20,20)),'o','MarkerSize',6)}
\]

8.1.6 Displaying Multiple Plots per Figure

You can display multiple plots in the same figure window and print them on the same piece of paper with the subplot function.

\[
\text{subplot}(m,n,i)
\]

breaks the figure window into an \( m \)-by-\( n \) matrix of small subplots and selects the \( i \)th subplot for the current plot. The plots are numbered along the top row of the figure window, then the second row, and so forth.

For example, the following statements plot data in four different subregions of the figure window.

\[
t = 0:pi/20:2*pi;
[x,y] = \text{meshgrid}(t);
\text{subplot}(2,2,1)
\text{plot}(\sin(t),\cos(t))
\text{axis equal}
\text{subplot}(2,2,2)
z = \sin(x)+\cos(y);
\text{plot}(t,z)
\text{axis}([0 2*pi -2 2])
\text{subplot}(2,2,3)
z = \sin(x).*\cos(y);
\text{plot}(t,z)
\text{axis}([0 2*pi -1 1])
\text{subplot}(2,2,4)
\]

24
8.2 Discrete Data Graphs

MATLAB has a number of specialized functions that are appropriate for displaying discrete data. This section describes how to use `stem` plots and `stairs` step plots to display this type of data. (Bar charts, discussed earlier in this section, are also suitable for displaying discrete data.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stem</td>
<td>Displays a discrete sequence of y-data as stems from x-axis</td>
</tr>
<tr>
<td>stem3</td>
<td>Displays a discrete sequence of z-data as stems from xy-plane</td>
</tr>
<tr>
<td>Stairs</td>
<td>Displays a discrete sequence of y-data as steps from x-axis</td>
</tr>
</tbody>
</table>

8.2.1 stem

Plot discrete sequence data

Syntax

```
stem(Y)
stem(X,Y)
stem(...,'fill')
stem(...,LineSpec)
```

Description: A two-dimensional stem plot displays data as lines extending from the x-axis. A circle (the default) or other marker whose y-position represents the data value terminates each stem.

`stem(Y)` plots the data sequence Y as stems that extend from equally spaced and automatically generated values along the x-axis. When Y is a matrix, stem plots all elements in a row against the same x value.

`stem(X,Y)` plots X versus the columns of Y. X and Y are vectors or matrices of the same size. Additionally, X can be a row or a column vector and Y a matrix with `length(X)` rows.

`stem(...,'fill')` specifies whether to color the circle at the end of the stem.

`stem(...,LineSpec)` specifies the line style, marker symbol, and color for the stem plot. See `LineSpec` for more information.

```
h = stem(...)
```

h = stem(...) returns handles to line graphics objects.

8.2.2 stairs

Stairstep plot

Syntax
stairs(Y)
stairs(X,Y)
stairs(...,LineSpec)
[xb,yb] = stairs(Y)
[xb,yb] = stairs(X,Y)

Description: Stairstep plots are useful for drawing time-history plots of digitally sampled data systems.

stairs(Y) draws a stairstep plot of the elements of Y. When Y is a vector, the x-axis scale ranges from 1 to size(Y). When Y is a matrix, the x-axis scale ranges from 1 to the number of rows in Y.

stairs(X,Y) plots X versus the columns of Y. X and Y are vectors of the same size or matrices of the same size. Additionally, X can be a row or a column vector, and Y a matrix with length(X) rows.

stairs(...,LineSpec) specifies a line style, marker symbol, and color for the plot (see LineSpec for more information).

[xb,yb] = stairs(Y) and [xb,yb] = stairs(x,Y) do not draw graphs, but return vectors xb and yb such that plot(xb,yb) plots the stairstep graph.

Examples:
Create a stairstep plot of a sine wave.

x = 0:.25:10;
stairs(x,sin(x))

8.3 Basic Printing and Exporting

MATLAB enables you to do basic printing and exporting of graphs. This section describes some frequently used ways to print or export a MATLAB figure, and explains the basic printing and exporting interfaces.

MATLAB enables you to print a figure from the screen directly to a printer, or to a file for later printing. It also enables you to export a figure in graphics format to a file, or to the clipboard, so that you can import, or paste, it into an application such as a word processor.

Before you print or export the figure, MATLAB enables you to change many settings that control the look of the figure. For example you can change the size and position of the figure, the paper type and orientation, as well as line, text and background attributes.

8.3.1 Linux

Plots generated in MATLAB can be saved to a postscript file so that they can be printed at a later time (for example, by the standard UNIX ‘lpr’ command). For example, to save the current plot type
The plot can also be printed directly from within MATLAB by typing

```matlab
>> print -Pprintername
```

## 8.3.2 Windows

Export a figure in a graphics format to a file if you want to import it into another application, such as a word processor. You can also export it to the Windows clipboard, and paste it from there into an application. Before deciding on a graphics format, check what formats are supported by your target application and platform. See the print reference page for a complete list of supported graphics formats.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Command Line</th>
<th>-device Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMP 8-bit color, bitmap</td>
<td>Export a figure to the clipboard (Windows only).</td>
<td></td>
<td>-dbitmap</td>
</tr>
<tr>
<td>EMF color vector format</td>
<td>Export a figure to the clipboard (Windows only).</td>
<td></td>
<td>-dmeta</td>
</tr>
<tr>
<td>EPS color and black-and-white</td>
<td>Export line plots or simple graphs to a file. Note: An EPS file does not display within some applications unless you add a TIFF preview image to it. See the example Exporting a Figure in EPS Format with a TIFF Preview.</td>
<td></td>
<td>-deps (black and white) -ddepsc (color) -ddepsc -tiff (TIFF preview)</td>
</tr>
<tr>
<td>JPEG 24-bit</td>
<td>Export plots with surface lighting or transparency to a file. This format can be displayed by most Web browsers.</td>
<td></td>
<td>-djpeg -djpegnumber where number is the compression.</td>
</tr>
<tr>
<td>TIFF 24-bit bitmap color</td>
<td>Export plots with surface lighting or transparency to a file. Widely available. A good format to choose if you are not sure what formats your application supports.</td>
<td></td>
<td>-dtiff</td>
</tr>
</tbody>
</table>