This document gives you a brief introduction and a reference section for getting started with Matlab. The first section describes the extensive on-line help that is provided within the Matlab environment. The remaining sections describe some of the more commonly used Matlab commands, how data is stored in matrices and how to create your own Matlab functions.

**HELP**

There are an enormous number of Matlab commands that can be used. Using the Command Window or the Help Window, one can access most of the information about using Matlab. Access to the Help Window is though typing `helpdesk` or selecting the Help menu or typing `help` in the Command Window. A good way to get more familiar with using Matlab is to know how to use the help provided effectively.

The Help has a hierarchical structure, for example:

```
help (help topics) → help elfun (elementary math functions) → help atan2
```

(four quadrant inverse tangent).

The help entries can be searched for keywords using the `lookfor` command. For example, searching for the keyword gives numerous matches:

```
lookfor 'inverse' →
```

```
  nvhilb  inverse Hilbert matrix.
  ipermute inverse permute array dimensions.
  acos     inverse cosine.
  acosh    inverse hyperbolic cosine.
  acot     inverse cotangent.
  acoth    inverse hyperbolic cotangent.
  ...
```

Typing `help` in the Command Window gives a list of the available help topics.
<table>
<thead>
<tr>
<th>Directory/Totopic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>matlab\general</code></td>
<td>General purpose commands</td>
</tr>
<tr>
<td><code>matlab\ops</code></td>
<td>Operators and special characters</td>
</tr>
<tr>
<td><code>matlab\lang</code></td>
<td>Programming language constructs</td>
</tr>
<tr>
<td><code>matlab\elmat</code></td>
<td>Elementary matrices and matrix manipulation</td>
</tr>
<tr>
<td><code>matlab\elfun</code></td>
<td>Elementary math functions</td>
</tr>
<tr>
<td><code>matlab\specfun</code></td>
<td>Specialized math functions</td>
</tr>
<tr>
<td><code>matlab\matfun</code></td>
<td>Matrix functions and numerical linear algebra</td>
</tr>
<tr>
<td><code>matlab\datafun</code></td>
<td>Data analysis and Fourier transforms</td>
</tr>
<tr>
<td><code>matlab\audio</code></td>
<td>Audio support</td>
</tr>
<tr>
<td><code>matlab\polyfun</code></td>
<td>Interpolation and polynomials</td>
</tr>
<tr>
<td><code>matlab\funfun</code></td>
<td>Function functions and ODE solvers</td>
</tr>
<tr>
<td><code>matlab\sparfun</code></td>
<td>Sparse matrices</td>
</tr>
<tr>
<td><code>matlab\graph2d</code></td>
<td>Two dimensional graphs</td>
</tr>
<tr>
<td><code>matlab\graph3d</code></td>
<td>Three dimensional graphs</td>
</tr>
<tr>
<td><code>matlab\specgraph</code></td>
<td>Specialized graphs</td>
</tr>
<tr>
<td><code>matlab\graphics</code></td>
<td>Handle Graphics</td>
</tr>
<tr>
<td><code>matlab\uitools</code></td>
<td>Graphical user interface tools</td>
</tr>
<tr>
<td><code>matlab\strfun</code></td>
<td>Character strings</td>
</tr>
<tr>
<td><code>matlab\iofun</code></td>
<td>File input/output</td>
</tr>
<tr>
<td><code>matlab\timefun</code></td>
<td>Time and dates</td>
</tr>
<tr>
<td><code>matlab\datatypes</code></td>
<td>Data types and structures</td>
</tr>
<tr>
<td><code>matlab\verctrl</code></td>
<td>Version control</td>
</tr>
<tr>
<td><code>matlab\winfun</code></td>
<td>Windows Operating System Interface Files (DDE/COM)</td>
</tr>
<tr>
<td><code>matlab\demos</code></td>
<td>Examples and demonstrations</td>
</tr>
<tr>
<td><code>toolbox\local</code></td>
<td>Preferences</td>
</tr>
<tr>
<td><code>images\images</code></td>
<td>Image Processing Toolbox</td>
</tr>
<tr>
<td><code>images\imdemos</code></td>
<td>Image Processing Toolbox --- demos and sample images</td>
</tr>
<tr>
<td><code>signal\signal</code></td>
<td>Signal Processing Toolbox</td>
</tr>
<tr>
<td><code>signal\signal</code></td>
<td>Signal Processing Toolbox</td>
</tr>
<tr>
<td><code>signal\sigtools</code></td>
<td>Design &amp; Analysis Tool (GUI)</td>
</tr>
<tr>
<td><code>signal\sptoolgui</code></td>
<td>Signal Processing Toolbox GUI</td>
</tr>
<tr>
<td><code>signal\sigdemos</code></td>
<td>Signal Processing Toolbox Demonstrations.</td>
</tr>
</tbody>
</table>

For more help on directory/topic, type "help topic".
For command syntax information, type "help syntax".
**Help elfun**

**Elementary math functions**

<table>
<thead>
<tr>
<th>Trigonometric</th>
<th></th>
<th>Hyperbolic sine</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin</td>
<td>Sine</td>
<td>sinh</td>
</tr>
<tr>
<td>asin</td>
<td>Inverse sine</td>
<td>asinh</td>
</tr>
<tr>
<td>cos</td>
<td>Cosine</td>
<td>cosh</td>
</tr>
<tr>
<td>acos</td>
<td>Inverse cosine</td>
<td>acosh</td>
</tr>
<tr>
<td>tan</td>
<td>Tangent.</td>
<td>tanh</td>
</tr>
<tr>
<td>atan</td>
<td>- Inverse tangent</td>
<td>atan2</td>
</tr>
<tr>
<td>atanh</td>
<td>Inverse hyperbolic tangent</td>
<td></td>
</tr>
<tr>
<td>sec</td>
<td>Secant</td>
<td>sech</td>
</tr>
<tr>
<td>asec</td>
<td>Inverse secant</td>
<td>asech</td>
</tr>
<tr>
<td>csc</td>
<td>Cosecant</td>
<td>csch</td>
</tr>
<tr>
<td>acsc</td>
<td>Inverse cosecant</td>
<td>acsch</td>
</tr>
<tr>
<td>cot</td>
<td>Cotangent</td>
<td>coth</td>
</tr>
<tr>
<td>acot</td>
<td>Inverse cotangent</td>
<td>acoth</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exponential</th>
<th></th>
<th>Natural logarithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp</td>
<td>Exponential</td>
<td>log</td>
</tr>
<tr>
<td>log10</td>
<td>Common (base 10) logarithm</td>
<td>log2</td>
</tr>
<tr>
<td>pow2</td>
<td>Base 2 power and scale floating point number</td>
<td>realpow</td>
</tr>
<tr>
<td>reallog</td>
<td>Natural logarithm of real number</td>
<td>realsqrt</td>
</tr>
<tr>
<td>sqrt</td>
<td>Square root</td>
<td>nextpow2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Complex</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>Absolute value</td>
<td>angle</td>
</tr>
<tr>
<td>complex</td>
<td>Construct complex data from real and imaginary parts</td>
<td>conj</td>
</tr>
<tr>
<td>imag</td>
<td>Complex imaginary part</td>
<td>real</td>
</tr>
<tr>
<td>unwrap</td>
<td>Unwrap phase angle</td>
<td>isreal</td>
</tr>
<tr>
<td>cplxpair</td>
<td>Sort numbers into complex conjugate pairs</td>
<td></td>
</tr>
</tbody>
</table>
### Rounding and remainder

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fix</td>
<td>Round towards zero</td>
</tr>
<tr>
<td>floor</td>
<td>Round towards minus infinity</td>
</tr>
<tr>
<td>ceil</td>
<td>Round towards plus infinity</td>
</tr>
<tr>
<td>round</td>
<td>Round towards nearest integer</td>
</tr>
<tr>
<td>mod</td>
<td>Modulus (signed remainder after division)</td>
</tr>
<tr>
<td>rem</td>
<td>Remainder after division</td>
</tr>
<tr>
<td>sign</td>
<td>Signum</td>
</tr>
</tbody>
</table>

#### help atan2

**ATAN2** Four quadrant inverse tangent.  
**ATAN2(Y,X)** is the four quadrant arctangent of the real parts of the elements of **X** and **Y**.  
\[-\pi < \text{ATAN2}(Y,X) \leq \pi.\]

See also **ATAN**.

The Matlab help accessed through the Help Window contains more information than the information displayed in the Command Window. For example, searching in the Help Window for the function **atan2** gives:

**atan2** Four-quadrant inverse tangent

**Syntax**

\[ P = \text{atan2}(Y,X) \]

**Description**

**P** = atan2**(Y,X)** returns an array **P** the same size as **X** and **Y** containing the element-by-element, four-quadrant inverse tangent (arctangent) of the real parts of **Y** and **X**. Any imaginary parts are ignored.

Elements of **P** lie in the closed interval \([-\pi, \pi]\), where \(\pi\) is the MATLAB floating-point representation of \(\pi\).

atan uses sign**(Y)** and sign**(X)** to determine the specific quadrant.

**atan2**(Y,X) contrasts with atan**(Y/X)**, whose results are limited to the interval \([-\pi/2, \pi/2]\) , or the right side of this diagram.

**Examples**

Any complex number \(z = x + iy\) is converted to polar coordinates with

\[ r = \text{abs}(z) \]
\[ \text{theta} = \text{atan2}(\text{imag}(z), \text{real}(z)) \]

For example, \(z = 4 + 3i\):

\[ r = \text{abs}(z) \]
\[ \text{theta} = \text{atan2}(\text{imag}(z), \text{real}(z)) \]
\[ r = 5 \quad \text{theta} = 0.6435 \]
help FILEFORMATS

Readable file formats.

<table>
<thead>
<tr>
<th>Data Formats</th>
<th>Command</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAT - MATLAB workspace</td>
<td>Load</td>
<td>Variables in file</td>
</tr>
<tr>
<td>CSV - Comma separated numbers</td>
<td>csvread</td>
<td>Double array.</td>
</tr>
<tr>
<td>DAT - Formatted text</td>
<td>importdata</td>
<td>Double array.</td>
</tr>
<tr>
<td>DLM - Delimited text</td>
<td>dlmread</td>
<td>Double array.</td>
</tr>
<tr>
<td>TAB - Tab separated text</td>
<td>dlmread</td>
<td>Double array.</td>
</tr>
</tbody>
</table>

Spreadsheet formats

<table>
<thead>
<tr>
<th>Spreadsheet formats</th>
<th>Command</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLS - Excel worksheet</td>
<td>xlsread</td>
<td>Double &amp; cell array</td>
</tr>
</tbody>
</table>

Scientific data formats

<table>
<thead>
<tr>
<th>Scientific data formats</th>
<th>Command</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDF - Common Data Format</td>
<td>cdfread</td>
<td>Cell array of CDF records</td>
</tr>
<tr>
<td>FITS - Flexible Image Transport System</td>
<td>fitsread</td>
<td>Primary or extension table data</td>
</tr>
<tr>
<td>HDF - Hierarchical Data Format</td>
<td>hdfread</td>
<td>HDF or HDF-EOS data set</td>
</tr>
</tbody>
</table>

Movie formats

<table>
<thead>
<tr>
<th>Movie formats</th>
<th>Command</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVI - Movie</td>
<td>aviread</td>
<td>MATLAB movie</td>
</tr>
</tbody>
</table>

Image formats

<table>
<thead>
<tr>
<th>Image formats</th>
<th>Command</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIFF - TIFF image</td>
<td>imread</td>
<td>Truecolor, grayscale or indexed image(s).</td>
</tr>
<tr>
<td>PNG - PNG image</td>
<td>imread</td>
<td>Truecolor, grayscale or indexed image</td>
</tr>
<tr>
<td>HDF - HDF image</td>
<td>imread</td>
<td>Truecolor or indexed image(s)</td>
</tr>
<tr>
<td>BMP - BMP image</td>
<td>imread</td>
<td>Truecolor or indexed image</td>
</tr>
<tr>
<td>JPEG - JPEG image</td>
<td>imread</td>
<td>Truecolor or grayscale image</td>
</tr>
<tr>
<td>GIF - GIF image</td>
<td>imread</td>
<td>Indexed image</td>
</tr>
<tr>
<td>PCX - PCX image</td>
<td>imread</td>
<td>Indexed image</td>
</tr>
<tr>
<td>XWD - XWD image</td>
<td>imread</td>
<td>Indexed image</td>
</tr>
<tr>
<td>CUR - Cursor image</td>
<td>imread</td>
<td>Indexed image</td>
</tr>
<tr>
<td>ICO - Icon image</td>
<td>imread</td>
<td>Indexed image</td>
</tr>
<tr>
<td>RAS - Sun raster image</td>
<td>imread</td>
<td>Truecolor or indexed</td>
</tr>
<tr>
<td>PBM - PBM image</td>
<td>imread</td>
<td>Grayscale image</td>
</tr>
<tr>
<td>PGM - PGM image</td>
<td>imread</td>
<td>Grayscale image</td>
</tr>
<tr>
<td>PPM - PPM image</td>
<td>imread</td>
<td>Truecolor image</td>
</tr>
</tbody>
</table>

Audio formats

<table>
<thead>
<tr>
<th>Audio formats</th>
<th>Command</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>AU - NeXT/Sun sound</td>
<td>auread</td>
<td>Sound data and sample rate</td>
</tr>
<tr>
<td>SND - NeXT/Sun sound</td>
<td>auread</td>
<td>Sound data and sample rate</td>
</tr>
<tr>
<td>WAV - Microsoft Wave sound</td>
<td>wavread</td>
<td>Sound data and sample rate</td>
</tr>
</tbody>
</table>

See also IOFUN

Matlab help is very useful but extensive and so the purpose of this Chapter is to review many of the common Matlab features and commands through illustrative examples.
GENERAL PURPOSE COMMANDS

The following table lists just a few of the Matlab commands that are used for managing the Matlab environment. The Matlab command is typed into the Command Window. For more information on any of the commands listed use help, e.g., help ver.

<table>
<thead>
<tr>
<th>Matlab Command</th>
<th>Function / Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>helpdesk</td>
<td>Opens Help Window</td>
</tr>
<tr>
<td>demo</td>
<td>Can view and run available Matlab demonstrations.</td>
</tr>
<tr>
<td>info</td>
<td>Provides contact information for getting extra assistance with Matlab.</td>
</tr>
<tr>
<td>ver</td>
<td>Displays the current Matlab, Simulink and toolbox version information.</td>
</tr>
<tr>
<td>dir</td>
<td>Lists the files in a directory. Pathnames and wildcards may be used. For example, dir <em>.</em> lists all the M-files in the current directory.</td>
</tr>
<tr>
<td>cd</td>
<td>cd by itself, prints out the current directory. Change current working directory. cd directory-spec: sets the current directory to the one specified cd \a03\mat\graphics</td>
</tr>
<tr>
<td>path</td>
<td>Controls Matlab's search path. For example, the following statements add another directory to Matlab's search path Windows: path(path,'c:\a03\mat\graphics')</td>
</tr>
<tr>
<td>pdf</td>
<td>Shows current working directory.</td>
</tr>
<tr>
<td>what</td>
<td>List MATLAB specific files in the current directory.</td>
</tr>
<tr>
<td>which</td>
<td>Locates functions and files which result → result not found. which sinc → C:\a03\mat\mg\scripts\sinc.m</td>
</tr>
<tr>
<td>save</td>
<td>save test.mat → saves all workspace variables to the file test.mat in the current directory.</td>
</tr>
<tr>
<td>load</td>
<td>load test → loads the variables saved in the file test.mat. save xData → saves only the variable xData.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>load Xdata</code></td>
<td>loads the variable <code>xData</code> into the Matlab workspace.</td>
</tr>
<tr>
<td><code>save test.mat xData yData zData</code></td>
<td>saves the variables <code>Xdata</code>, <code>yData</code> and <code>zData</code> in the file <code>test.mat</code> in the current directory.</td>
</tr>
<tr>
<td><code>delete</code></td>
<td>deletes the file <code>test.mat</code> from the current directory.</td>
</tr>
<tr>
<td><code>pack</code></td>
<td>Consolidate workspace memory: performs memory garbage collection. Extended Matlab sessions may cause memory to become fragmented, preventing large variables from being stored. <code>pack</code> saves all variables on disk, clears the memory, and then reloads the variables.</td>
</tr>
<tr>
<td><code>diary</code></td>
<td>Save text of MATLAB session. <code>diary filename</code> causes a copy of all subsequent command window input and most of the resulting command window output to be appended to the named file. If no file is specified, the file 'diary' is used. <code>diary off</code> suspends it. <code>diary on</code> turns it back on. <code>diary</code>, by itself, toggles the diary state.</td>
</tr>
<tr>
<td><code>clear</code></td>
<td>clears all variables and functions from workspace. <code>clear all</code> removes all variables, globals, functions and MEX links. <code>clear Xdata yData</code> clears the variables <code>xData</code> and <code>yData</code> from the workspace.</td>
</tr>
<tr>
<td><code>home</code></td>
<td>Moves the cursor to the upper left corner of the Command Window and clears the visible portion of the window. You can use the scroll bar to see what was on the screen previously.</td>
</tr>
<tr>
<td><code>clc</code></td>
<td>Clears the command window and homes the cursor.</td>
</tr>
<tr>
<td><code>echo on / off</code></td>
<td>Toggles the printing of instructions from m-script in Command Window.</td>
</tr>
</tbody>
</table>
### Miscellaneous m-script commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pause</strong></td>
<td>wait for user response (press any key to continue)</td>
</tr>
<tr>
<td><strong>pause(10)</strong></td>
<td>halts execution of m-script for 10 seconds</td>
</tr>
<tr>
<td><strong>pause(0.1)</strong></td>
<td>halts execution of m-script for 0.1 seconds</td>
</tr>
<tr>
<td><strong>pause off</strong></td>
<td>subsequent pause ignored</td>
</tr>
<tr>
<td><strong>pause on</strong></td>
<td>subsequent pause commands should pause</td>
</tr>
</tbody>
</table>

**keyboard** stops execution of the m-script and gives control to keyboard.

- K appears before the prompt.
- Variables may be examined or changed – all commands are valid.
- Keyboard mode terminated by hitting Enter

**input** prompt user for input

```
num = input('How many particles? ')
```

**menu**

```
choice = menu(header, item1, item2, ...)
```

Generate a menu of choices for user input.

```
K = menu({'Choose a color','Red','Blue','Green'})
```

----- Choose a color ------

1) Red
2) Blue
3) Green

Select a menu number:

The number entered by the user in response to the prompt is returned as K (i.e. \( K = 2 \) implies that the user selected Blue).
Operators and special functions

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>plus</td>
<td>minus</td>
<td>matrix power</td>
</tr>
<tr>
<td>.</td>
<td>array power</td>
<td>backslash or left division</td>
<td>slash or right division</td>
</tr>
<tr>
<td>/</td>
<td>array division</td>
<td>colon (subscripting, array manipulation)</td>
<td>parentheses (contains arguments)</td>
</tr>
<tr>
<td>.</td>
<td>decimal point</td>
<td>parent directory</td>
<td>continuation</td>
</tr>
<tr>
<td>,</td>
<td>comma (argument / statement separator)</td>
<td>semicolon (suppress statement output)</td>
<td>matrix multiplication</td>
</tr>
<tr>
<td>.*</td>
<td>array multiplication</td>
<td>%</td>
<td>comment</td>
</tr>
<tr>
<td>.</td>
<td>nonconjugated transpose</td>
<td>=</td>
<td>assignment</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>greater than</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than or equal to</td>
<td>&amp;</td>
<td>logical AND</td>
</tr>
<tr>
<td>~</td>
<td>logical NOT</td>
<td>xor</td>
<td>logical EXCLUSIVE OR</td>
</tr>
</tbody>
</table>

Rational and logical operations

<table>
<thead>
<tr>
<th>=</th>
<th>&lt;=</th>
<th>&gt;=</th>
<th>==</th>
<th>~=</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1; y = 20; if x == 2, a = 0, end; if y &gt;= 15, a = 1, end → a = 1 x == 4 → 0 x == 1 → 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&amp;</th>
<th>logical and</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1; y = 20; if x == 1 &amp; y &lt;= 2, a = 0, end; if x &gt; 0 &amp; y &lt; 100, a = 1, end → a = 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>logical or</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1; y = 20; a = 99; b = 1; if x &lt; 1</td>
<td>y &gt;= 2, a = 0, end; if x &gt; 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>~</th>
<th>logical not</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1; y = 20; a = 0; ~x → 0 ~y → 0 <del>a → 1 x =</del> 4 → 1</td>
<td></td>
</tr>
</tbody>
</table>
ARRAYS AND MATRICES

Matlab works with arrays or matrices and the elements may be strings, real or complex numbers and functions can have real or complex arguments. Matlab functions and arithmetic operations can be performed directly on matrices. A matrix of a single element can be through of a single constant or variable (A = 3). A matrix can be a row vector or a column vector or a multi-dimensional array. Unlike most programming languages, commands can act simultaneously on all elements of an array. For example the set of numbers 1, 4, 9, 16, 25, 36, 49 can be entered into a row vector by a statement in the Command, for example:

\[ x_R = [1 \ 4 \ 9 \ 16 \ 25 \ 36 \ 49] \]

The command \texttt{sqrt(xR)} will act on each element of the array \( x_R \) by taking the square root of each number

\[ \text{sqrt}(xR) \rightarrow 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \]

The name of a matrix must be start with any letter, followed by any combination of letters (upper or lower case) and numbers, for example,

\[ A, a, \text{ScreenWidth}, xR, xC, \text{Slit\_separation} \quad (A \text{ and } a \text{ refer to different matrices}) \]

The tables below show how data can be entered into a matrix; how to perform some of the operations and use functions that can act on matrices; and lists some of the special Matlab matrices. The appearance and number of significant figures of a matrix displayed in the Command Window can be changed using the \texttt{format} command.

<table>
<thead>
<tr>
<th>Format</th>
<th>x = 51.12345678987654321</th>
<th>y = 5.1123456 \times 10^{23}</th>
</tr>
</thead>
</table>
| \texttt{format} or \texttt{format short} | x \rightarrow 51.1235 \ 
y \rightarrow 5.1123e+023 | \texttt{format long} | x \rightarrow 51.12345678987654 \ 
y \rightarrow 5.1123456000000000e+023 |
| \texttt{format short e} | x \rightarrow 5.1123e+001 \ 
y \rightarrow 5.1123e+023 | \texttt{format long e} | x \rightarrow 5.112345678987654e+001 \ 
y \rightarrow 5.1123456000000000e+023 |
| \texttt{disp} (display array) | \texttt{disp(x) \rightarrow 51.1235} \ 
\texttt{tm = ' time t (s)'} | \texttt{disp(tm) \rightarrow time t (s)} |
There is a very extensive set of Matlab mathematical functions. Some of the functions which are most commonly used are given in the table below. It is a good idea to practice using these functions in the Command Window.

### Miscellaneous functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs(x)</code></td>
<td>abs(x) → 51</td>
<td><code>abs(-51) → 51</code></td>
</tr>
<tr>
<td><code>sqrt(x)</code></td>
<td><code>sqrt(51) → 7.1414</code> <code>sqrt(-51) → 0 + 7.1414i</code></td>
<td></td>
</tr>
<tr>
<td><code>round(x)</code></td>
<td><code>round to nearest integer</code></td>
<td><code>round(51) → 51</code></td>
</tr>
<tr>
<td><code>fix(x)</code></td>
<td><code>round towards zero</code></td>
<td><code>fix(51) → 51</code></td>
</tr>
<tr>
<td><code>floor(x)</code></td>
<td><code>round toward -∞</code></td>
<td><code>floor(51) → 51</code></td>
</tr>
<tr>
<td><code>ceil(x)</code></td>
<td><code>round toward +∞</code></td>
<td><code>ceil(51) → 51</code></td>
</tr>
<tr>
<td><code>sign(x)</code></td>
<td><code>sign</code></td>
<td><code>sign(51.145) → 1</code></td>
</tr>
<tr>
<td><code>mod(x,y)</code></td>
<td><code>modulus</code></td>
<td><code>mod(30,5) → 0</code></td>
</tr>
<tr>
<td><code>rem(x,y)</code></td>
<td><code>remainder</code></td>
<td><code>rem(30,5) → 0</code></td>
</tr>
<tr>
<td><code>log(x)</code></td>
<td><code>log base e</code></td>
<td><code>log(exp(1)) → 1</code></td>
</tr>
<tr>
<td><code>factorial(x)</code></td>
<td><code>factorial x!</code></td>
<td><code>factorial(4) → 24</code></td>
</tr>
<tr>
<td><code>rand</code></td>
<td><code>random number 0 to 1</code></td>
<td><code>rand → 0.9318</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>rand(2,3) →</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.4660 0.8462 0.2026</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.4186 0.5252 0.6721</td>
</tr>
<tr>
<td></td>
<td><code>a = 1; b = 10; floor(a + b*rand) → random integer from 1 to 10</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>`rand(‘state’, sum(100*clock))</td>
<td>reset random number generator, so different results are obtained</td>
</tr>
<tr>
<td><strong>date</strong></td>
<td><strong>clock</strong></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>( s = \text{date}; \quad s \rightarrow 04-\text{Oct}-2003 )</td>
<td>( \text{clock} = [\text{year} \ \text{month} \ \text{day} \ \text{hour} \ \text{minute} \ \text{seconds}] )</td>
<td></td>
</tr>
<tr>
<td>( \text{fix}(\text{clock}) \rightarrow 2003 \ 10 \ 4 \ 16 \ 23 \ 35 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>etime</strong></th>
<th><strong>tic</strong> - <strong>toc</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Difference in seconds between two dates</td>
<td>Starts a seconds counter</td>
</tr>
<tr>
<td>( T_0 = [2000 \ 10 \ 15 \ 0 \ 0 \ 0] )</td>
<td>Stops the seconds counter</td>
</tr>
<tr>
<td>( T_1 = [2004 \ 10 \ 10 \ 12 \ 0 \ 0] )</td>
<td></td>
</tr>
<tr>
<td>( \text{etime}(T_1, T_0) \rightarrow 125841600 )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>cputime</strong></th>
<th><strong>feval</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed CPU time in seconds</td>
<td>Executes functions specified by strings</td>
</tr>
<tr>
<td>( a = \text{cputime} \rightarrow )</td>
<td></td>
</tr>
<tr>
<td>( a = 1.811010000000000e+003 )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>eval</strong></th>
<th><strong>global</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluates a string expression</td>
<td>Defines global variables</td>
</tr>
<tr>
<td>( s = '123' )</td>
<td>... ( \text{(press Enter)} )</td>
</tr>
<tr>
<td>( a = \text{eval}(s) \rightarrow a = 123 )</td>
<td>Long lines converted into short lines</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Complex numbers</strong></th>
<th>( z = x + iy )</th>
<th>( i = \sqrt{-1} ) or ( j = \sqrt{-1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>angle(z)</strong></td>
<td>( z = 4.5600 + 1.2300i )</td>
<td>( \text{real}(z) \rightarrow 4.5600 )</td>
</tr>
<tr>
<td>( \text{angle}(z) \rightarrow 0.2635 )</td>
<td><strong>imag(z)</strong></td>
<td>( z = 4.5600 + 1.2300i )</td>
</tr>
<tr>
<td>( \text{imag}(z) \rightarrow 1.2300 )</td>
<td><strong>conj(z)</strong></td>
<td>( z = 4.5600 + 1.2300i )</td>
</tr>
<tr>
<td>( \text{conj}(z) \rightarrow 4.5600 - 1.2300i )</td>
<td><strong>abs(z)</strong></td>
<td>( z = 4.5600 + 1.2300i )</td>
</tr>
<tr>
<td>( \text{abs}(z) \rightarrow 4.7230 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Trigonometric functions (angles must be radians)

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
<th>Calculation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sin(x))</td>
<td>(x = 30) % x in degrees</td>
<td>(\sin(x\cdot(\pi/180)))</td>
<td>0.5000</td>
</tr>
<tr>
<td>(\cos(x))</td>
<td>(x = 30) % x in degrees</td>
<td>(\cos(x\cdot(\pi/180)))</td>
<td>0.8660</td>
</tr>
<tr>
<td>(\tan(x))</td>
<td>(x = 30) % x in degrees</td>
<td>(\tan(x\cdot(\pi/180)))</td>
<td>0.5774</td>
</tr>
<tr>
<td>(\arcsin(x))</td>
<td>(x = 0.7071)</td>
<td>(\arcsin(x))</td>
<td>0.7854</td>
</tr>
<tr>
<td>(\arccos(x))</td>
<td>(x = 0.7071)</td>
<td>(\arccos(x))</td>
<td>0.7854</td>
</tr>
<tr>
<td>(\arctan(x))</td>
<td>(x = 0.7071)</td>
<td>(\arctan(x))</td>
<td>0.0706</td>
</tr>
</tbody>
</table>

\(180/\pi)\) * Arc functions (angle in degrees)

\(\arcsin(x)\) | \(x = 0.7071\) | \(\arcsin(x)\) | 44.995 |
| \(\arccos(x)\) | \(x = 0.7071\) | \(\arccos(x)\) | 45.0005 |
| \(\arctan(x)\) | \(x = 0.7071\) | \(\arctan(x)\) | 4.0447 |

\(\tan2(y,x)\)

\(y = 1; x = 1\) | \(\tan2(y,x)/\pi\) | 0.2500 |
| \(y = 1; x = -1\) | \(\tan2(y,x)/\pi\) | 0.7500 |

\(\arctan2(y,x)\)

\(y = 1; x = 1\) | \(\arctan2(y,x)/\pi\) | -0.2500 |
| \(y = 1; x = -1\) | \(\arctan2(y,x)/\pi\) | 0.7500 |
### Setting up matrices

#### Simple variables

\[ u = 6; \ v = -12; \]

#### Complex variable

\[ z = 12 - 3i \]

#### Row vector

\[
\begin{align*}
xR &= [1 \ 2 \ 3 \ 4] \rightarrow 1 \ 2 \ 3 \ 4 \\
yR &= [9, \ 6, \ 3] \rightarrow 9 \ 6 \ 3 \\
X_{\min} &= 0; \ X_{\max} = 3; \ dX = 0.5; \\
X &= X_{\min} : dX : X_{\max} \\
&\rightarrow 0 \ 0.5 \ 1.0 \ 1.5 \ 2.0 \ 2.5 \ 3.0 \\
\text{length}(X) &= 7 \\
\text{length}(X_R) &= 4 \\
\text{length}(y_R) &= 3
\end{align*}
\]

Removing an element

\[ X(6) = [] \rightarrow 0 \ 0.5 \ 1.0 \ 1.5 \ 2.0 \ 3.0 \]

#### Column Vector

\[
\begin{align*}
xC &= [1; \ 2; \ 3; \ 4] \rightarrow \\
&1 \\
&2 \\
&3 \\
&4 \\
\text{length}(xC) &= 4 \\
\text{Removing element} \\
xC(3) &= [] \rightarrow \\
&1 \\
&2 \\
&4 \\
\text{Adding a column} \\
xC(:,2) &= [6; \ 7; \ 8; \ 9] \rightarrow \\
&1 \ 6 \\
&2 \ 7 \\
&3 \ 8 \\
&4 \ 9
\end{align*}
\]

#### Matrix \(3 \text{ rows } \times 4 \text{ columns}\)

\[
\begin{align*}
M &= [1 \ 2 \ 3 \ 4; \ 5 \ 6 \ 7 \ 8; \ -1 \ -5 \ -8 \ -7] \rightarrow \\
&1 \ 2 \ 3 \ 4 \\
&5 \ 6 \ 7 \ 8 \\
&-1 \ -5 \ -8 \ -7 \\
M(3,2) &= -5 \\
M(1,4) &= 4 \\
M(:,\ 2) &= [6; \ -5] \\
M(1, :) &= 1 \ 2 \ 3 \ 4 \\
M(2, [1 \ 3]) &= 5 \ 7 \\
M(3, 2:3) &= -5 \ -8 \\
\text{Removing a column} \\
M(:, 2) &= [] \rightarrow 1 \ 3 \ 4 \\
&5 \ 7 \ 8 \\
&-1 \ -8 \ -7
\end{align*}
\]
Matricies

<table>
<thead>
<tr>
<th>size</th>
<th>size of an array</th>
</tr>
</thead>
<tbody>
<tr>
<td>size(u)   → 1, 1</td>
<td></td>
</tr>
<tr>
<td>size(xR)  → 1, 4</td>
<td></td>
</tr>
<tr>
<td>size(xC)  → 4, 1</td>
<td></td>
</tr>
<tr>
<td>size(M)   → 3, 4</td>
<td></td>
</tr>
</tbody>
</table>

who
lists the current variables

whos
lists all the variables in the current workspace, together with information about their size, bytes, class, etc.

lin space(Xmin; Xmax, N)
linear spaced row vector with N elements from Xmin to Xmax values with a spacing between elements of dX = (X(N)-X(1))/(N-1)

Xmin = 0; Xmax = 10; N = 10;
X = linspace(Xmin,Xmax,N) →
0, 1.111, 2.222, 3.333, 4.444, 5.556, 6.667, 7.778, 8.889, 1.000

Xmin = 0; Xmax = 10; N = 11;
X = linspace(Xmin,Xmax,N) →
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

log space(a, b, N)
generates a row vector of N logarithmically equally spaced points between decades 10^a and 10^b

a = 0; b = 3; N = 6;
X = logspace(a,b,N) →
1.0e+003 *
0.0010 0.0040 0.0158
0.0631 0.2512 1.0000

eye(N)
N×N identity matrix

eye(4) →
\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

zeros(M,N)
M×N zero matrix

zeros(3,5) →
\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

ones(M,N)
M×N unit matrix

ones(2,6) →
\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
\]

sort(X)

X = [9 5 7 3 1 2 8]
sort(X) → 1 2 3 5 7 8 9
### sum(X)

<table>
<thead>
<tr>
<th>X = [9 5 7 3 1 2 8]</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum(X) → 35</td>
</tr>
</tbody>
</table>

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

\[
\text{sum(M)} \text{ or sum(M,1)} \rightarrow 5 \ 3 \ 2 \ 5
\]

sums columns

\[
\text{sum(M(:,1))} \rightarrow 5
\]

sums 1st column

\[
\text{sum(M,2)} \rightarrow 10 \\
\text{26} \\
\text{-21}
\]

sums rows

\[
\text{sum(sum(M))} \rightarrow 15
\]

sums all elements

### max(X) \ min(X)

<table>
<thead>
<tr>
<th>X = [9 5 7 3 1 2 8]</th>
</tr>
</thead>
<tbody>
<tr>
<td>max(X) → 9</td>
</tr>
<tr>
<td>min(X) → 1</td>
</tr>
</tbody>
</table>

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

\[
\text{min(M)} \rightarrow -1 \ -5 \ -8 \ -7
\]

min columns

\[
\text{max(M)} \rightarrow 5 \ 6 \ 7 \ 8
\]

max columns

\[
\text{min(min(M))} \rightarrow -8
\]

\[
\text{max(max(M))} \rightarrow 8
\]

max or min of all elements

\[
\text{max(M')} \rightarrow 4 \ 8 \ -1
\]

max rows

### num2str

converts a number to a string

\[
\text{pi} = 3.14159265358979
\]

\[
\text{h} = 6.626076 \times 10^{-34}
\]

\[
\text{S} = \begin{bmatrix}
1.126 & 2.123 \\
3.123 & 4.123
\end{bmatrix}
\]

\[
\text{num2str(pi)} \rightarrow '3.1416'
\]

\[
\text{num2str(pi, 0)} \rightarrow '3'
\]

\[
\text{num2str(pi,8)} \rightarrow '3.1415927'
\]

\[
\text{num2str(h)} \rightarrow '6.6261e-034'
\]

\[
\text{num2str(h, 0)} \rightarrow '7e-034'
\]

\[
\text{num2str(h,8)} \rightarrow '6.63e-034'
\]

\[
\text{num2str(S,2)} \rightarrow
\begin{bmatrix}
'1.1' & '2.1' \\
'3.1' & '4.1'
\end{bmatrix}
\]

### str2num

converts a character array representation of a matrix of numbers to a numeric matrix

\[
\text{str2num('123')} \rightarrow 123
\]

\[
\text{str2num('abc123')} \rightarrow []
\]

### disp

displays the array, without printing the array name, same as leaving the semi-colon off an expression except that empty arrays don't display.

\[
\text{disp(pi)} \rightarrow 3.1416
\]

\[
\text{disp('Speed')} \rightarrow \text{Speed}
\]

\[
\text{max_speed} = 25.45
\]

\[
\text{disp('The maximum speed is ',num2str(max_speed), ' m/s')} \rightarrow
\]

The maximum speed is 25.45 m/s
**Format** is a format control string containing conversion specifications or any optional text

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%P.Qe</td>
<td>for exponential</td>
</tr>
<tr>
<td>%P.Qf</td>
<td>for fixed point</td>
</tr>
<tr>
<td>%P.Qg</td>
<td>select shorter of %P.Qe or %P.Qf</td>
</tr>
</tbody>
</table>

- P integer specifying field width
- Q integer specifying number of decimal places

\n produces a new line

**fprintf**

Write formatted data to file.

```matlab
x = 0:.1:1; y = [x; exp(x)];
fid = fopen('exp.txt','w');
fprintf(fid,'%6.2f  %12.8f
',y);
fclose(fid);
```

**sprintf**

Write formatted data to string.

```matlab
sprintf('%0.5g',(1+sqrt(5))/2) → 1.618
sprintf('%0.5g',1/eps) → 4.5036e+15
sprintf('%15.5f',1/eps) → 503599627370496.00000
sprintf('%d',round(pi)) → 3
sprintf('%s','Speed') → Speed
sprintf('The array is %dx%d.',2,3) → The array is 2x3.
sprintf('\n') → line termination character
```

**csvrad** read a file of comma-separated values

**cswrite** write a file of comma-separated values

**fclose** close file

**fopen** open file

**fread** read binary data from file

**fwrite** write binary data to file

**fprintf** write formatted data to file

**fscanf** read formatted data from file
Matrix operations

Matrices that have identical dimensions can be **added** or **subtracted**.

\[
\begin{align*}
A &= \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} & B &= \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \end{bmatrix} \\
A + B &= \begin{bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \end{bmatrix} & A - B &= \begin{bmatrix} 0 & 2 & 0 \\ -2 & 0 & 2 \end{bmatrix} \\
C &= [A ; 10 11 12] & D &= A + C \\
&= \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 10 & 11 & 12 \end{bmatrix} & & \text{??? Error using } == > + \text{ Matrix dimensions must agree.}
\end{align*}
\]

Matrices can be **multiplied** together. For example, \( C = A \cdot B \) where the matrix \( A \) has elements \( a_{ik} \), \( i \) row and \( k \) column, \( B \) has elements \( b_{kj} \) and \( C \) has elements \( c_{ij} \)

\[
c_{ij} = \sum_k a_{ik} b_{kj}
\]

\[
A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} & B = \begin{bmatrix} 1 & 4 \\ 2 & 6 \\ 3 & 8 \end{bmatrix} \\
A \cdot B &= \begin{bmatrix} 14 & 40 \\ 32 & 94 \end{bmatrix} & B \cdot A &= \begin{bmatrix} 17 & 22 & 27 \\ 26 & 34 & 42 \\ 35 & 46 & 57 \end{bmatrix}
\]

**Element by element multiplication** can be done using the **dot \( \cdot \)** operator, for example,

\[
\begin{align*}
C &= [2 8; 4 12; 6 16] \\
B \cdot C &= C \cdot B \\
&= \begin{bmatrix} 2 & 32 \\ 8 & 72 \\ 18 & 128 \end{bmatrix}
\end{align*}
\]

For element by element multiplication, the two matrices must have matching dimensions. For example, error messages are returned for \( A \cdot B \) or \( B \cdot A \)

\[
A \cdot B \rightarrow \text{??? Error using } == > \cdot \text{ Matrix dimensions must agree.} \\
B \cdot A \rightarrow \text{??? Error using } == > \cdot \text{ Matrix dimensions must agree.}
\]
The **transpose** of a matrix is given by the command `transpose` or `'`. For example,

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

\[
\text{transpose}(A) \rightarrow \begin{bmatrix}
2 & 4 \\
5 & 6 \\
6 & 3
\end{bmatrix}
\]

\[
B' \rightarrow \begin{bmatrix}
1 & 2 & 3 \\
4 & 6 & 8
\end{bmatrix}
\]

\[
xR = [2 \ 4 \ 6 \ 8] \rightarrow 2 \ 4 \ 6 \ 8 \\
yR = [-1 \ 1 \ 1 \ -1] \rightarrow -1 \ 1 \ 1 \ -1
\]

\[
xR \ast yR \rightarrow ??? \text{ Error using ==> * Inner matrix dimensions must agree.}
\]

\[
xR \ast yR' \rightarrow 0
\]

\[
xR \ast yR \rightarrow -2 \ 4 \ 6 \ -8
\]

\[
xR \ast yR' \rightarrow ??? \text{ Error using ==> .* Matrix dimensions must agree.}
\]

If the matrix that is to be transposed has complex elements, then the `'` operator gives the complex conjugate transpose. To give the transpose without conjugation, use the `.'` operation

\[
\begin{bmatrix}
1 + 4i & 2 - i & 3 + 6i & 6 - 3i
\end{bmatrix}
\]

\[
C' \rightarrow \begin{bmatrix}
1.0000 & 2.0000 + 1.0000i \\
4.0000 - 8.0000i & 5.0000 \\
6.0000 - 3.0000i
\end{bmatrix}
\]

\[
C.' \rightarrow \begin{bmatrix}
1.0000 & 2.0000 - 1.0000i \\
4.0000 + 8.0000i & 5.0000 \\
6.0000 - 3.0000i
\end{bmatrix}
\]

**Matrix division**

\[
A / B \rightarrow \begin{bmatrix}
2.3333 & -3.3333 \\
3.3333 & -4.3333
\end{bmatrix} \\
A \backslash B \rightarrow \begin{bmatrix}
-6.0000 & -5.5000 & -5.0000 \\
0 & 0 & 0 \\
5.0000 & 4.5000 & 4.0000
\end{bmatrix}
\]

\[
A ./ B \rightarrow \begin{bmatrix}
0.1111 & 0.2500 & 0.4286 \\
0.6667 & 1.0000 & 1.5000
\end{bmatrix}
\]
## Manipulating matrices

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| `rot90` | rotate matrix by 90 degrees | `rot90(A)` → \[
\begin{bmatrix}
3 & 6 & 9 \\
2 & 5 & 8 \\
1 & 4 & 7 \\
\end{bmatrix}
\] |
| `diag` | create or extract diagonals | `diag(A)` → \[
\begin{bmatrix}
1 \\
5 \\
9 \\
\end{bmatrix}
\] |
| `trace` | sum of diagonal elements | `trace(A)` → 15 |
| `det` | determinant | `det(A)` → 0 |
| `tril` | extract lower triangular part | `tril(A)` → \[
\begin{bmatrix}
1 & 0 & 0 \\
4 & 5 & 0 \\
7 & 8 & 9 \\
\end{bmatrix}
\] |
| `triu` | extract upper triangular part | `triu(A)` → \[
\begin{bmatrix}
1 & 2 & 3 \\
0 & 5 & 6 \\
0 & 0 & 9 \\
\end{bmatrix}
\] |
| `fliplr` | flip matrix in the left–right direction | `fliplr(A)` → \[
\begin{bmatrix}
6 & 5 & 4 \\
9 & 8 & 7 \\
\end{bmatrix}
\] |
| `flipup` | flip matrix in the up–down direction | `flipup(A)` → \[
\begin{bmatrix}
7 & 8 & 9 \\
4 & 5 & 6 \\
1 & 2 & 3 \\
\end{bmatrix}
\] |
| `flipdim` | flip matrix along specified dimension | `flipdim(A,1)` → \[
\begin{bmatrix}
7 & 8 & 9 \\
4 & 5 & 6 \\
1 & 2 & 3 \\
\end{bmatrix}
\] | `flipdim(A,2)` → \[
\begin{bmatrix}
3 & 2 & 1 \\
6 & 5 & 4 \\
9 & 8 & 7 \\
\end{bmatrix}
\] |
| `norm` | matrix or vector norm | `norm(x)` gives Euclidean length | `x = [0 1 2 3]` | `norm(x)` → \[
\sqrt{0+1+4+9} \\
= 3.7417
\] |
**Find**

Find indices of nonzero elements.

\[ I = \text{FIND}(X) \rightarrow \text{the indices of the vector } X \text{ that are non-zero.} \]

\[ I = \text{FIND}(A > 100) \rightarrow \text{the indices of } A \text{ where } A \text{ is greater than } 100. \]

\[ [I, J] = \text{FIND}(X) \rightarrow \text{row and column indices of the nonzero entries in matrix } X. \]

\[ [I, J, V] = \text{FIND}(X) \rightarrow \text{vector containing the nonzero entries in } X. \]

Note that `find(X)` and `find(X~=0)` will produce the same I and J, but the latter will produce a V with all 1's.

---

**MATLAB AS A PROGRAMMING LANGUAGE**

**Control of the execution of a program**

Matlab is a program language where the code is stored in text files as m-script or as functions. An important set of commands are used to control the flow of the program by testing when some condition is satisfied using `if-else-end` or `switch-case` commands and by using `for` and `while` loops to repeat a set of statements.

**Examples: if-end, if-else-end, if-elseif-end commands**

```matlab
N = input(' Enter a number '); text = 'The number is not an integer';
if mod(N,2) == 0, text = 'The number is even integer'; end
if mod(N,2) == 1, text = 'The number is odd integer'; end
disp(text)
```

```matlab
N = input(' Enter a number '); if mod(N,2) == 0
    text = 'The number is even integer';
else
    text = 'The number is not an even integer';
end
disp(text)
```

```matlab
N = input(' Enter a number '); if mod(N,2) == 0
    text = 'The number is even integer';
elseif mod(N,2) == 1
    text = 'The number is an odd integer';
else
    text = 'The number is not an integer';
end
disp(text)
```
Loops
To maximize speed of execution, matrices should be pre-allocated before a For or While Loop.

for ... end break
Using the for ... end loop commands, statements can be repeated a number of times. Long loops are more memory efficient when the colon expression appears in the for command since the index vector is never created. The break statement can be used to terminate the loop prematurely. If the initial value is xMin, the increment is dx (can be positive or negative) and xMax is the final value of the loop variable

```
for c = xMin : dx : xMax
    x = 20;
    y = 10;
    for cx = 1 : x;
        for cy = x: -2 : y;
            psi(cx, cy) = cx^2 + cy^2;
            sin(2*pi*cx/25)*sin(2*pi*cy/55);
        end
    end
end
```

while ... end
The while statement is used to repeat a statement a number of times until a conditions is not satisfied. For example, to calculate a function a given number of steps

```
maxSteps = input('Enter the number of steps for calculations ');
Steps = 1;
while Steps <= maxSteps
    x(Steps+1) =
```

switch ... case ... end
The selection of a block of code to be executed can be done with the switch - cases statements. For example to evaluate different functions

```
a = 2; b = 0.5;
x = 0 : 2 : 10;
flag = input('Select type of equation: 1, 2, ..., ')
switch flag
    case 1
        y = a .* x + b;
    case 2
        y = a .* x;
    case 3
        y = a .* exp(-b.*x)
    otherwise
        y = [];
end
```

Running this code with flag = 2 → y = 0 4 8 12 16 20
FUNCTIONS

Functions in Matlab are a very powerful tool for evaluating a sequence of commands and / or evaluating mathematical functions. The function is a text file similar to an m-script and has a .m extension. Input variables can be passed to the function and output variables are returned, any intermediate variable values within the function are not passed on to the Matlab workspace or to other functions. The function can be executed from the Command Window or from an m-script. To illustrate the how to create and use Matlab functions, a number of examples will be considered.

Example: Distance between two points
If the coordinates of two points \( P(x_P, y_P, z_P) \) and \( Q(x_Q, y_Q, z_Q) \) are known than the distance, \( d \) between the points is

\[
d = \sqrt{(x_P - x_Q)^2 + (y_P - y_Q)^2 + (z_P - z_Q)^2}
\]

The input variables passed to the function are the six coordinates of the two points \( P \) and \( Q \). The output variable returned from the function is the distance \( d \). The text for the function distance.m is

```matlab
function d = distance(xP,yP,zP,xQ,yQ,zQ)
    % Function to calculate the distance between two points P and Q
    d = sqrt((xP-xQ)^2 + (yP-yQ)^2 + (zP-zQ)^2);
end
```

The following statement when entered into the Command Window

```matlab
d = distance(0,0,0,1,1,1)
```

gives

\[
d = 1.7321
\]
Example: Converting between Cartesian and polar coordinates

A point \( P \) in a plane can be specified in Cartesian \((x_P, y_P)\) or in polar coordinates \((\rho_P, \theta_P)\). The relationships between the two coordinate systems are

\[
\begin{align*}
\rho_P &= \sqrt{x_P^2 + y_P^2} \\
x_P &= \rho_P \cos \theta_P \\
y_P &= \rho_P \sin \theta_P
\end{align*}
\]

The two functions to convert Cartesian to polar and polar to Cartesian coordinates are

```matlab
function [rho, theta] = CartesianToPolar(x,y)

% Function to convert Cartesian coordinates (x, y)
%   to polar coordinates (rho, theta)
% The Matlab function atan2 returns an angle in radians
% If y >=0 then 0 <= theta <= pi
% if y < 0 then -pi < theta < 0
rho = sqrt(x^2 + y^2);
theta = atan2(y,x);
end

function [x, y] = PolarToCartesian(rho,theta)

% Function to convert Polar coordinates (rho, theta)
%   to Cartesian coordinates (x, y)
% The angle theta must be in radians
x = rho * cos(theta);
y = rho * sin(theta);
end
```

The functions can be used in the Command Window or executed from an m-script, for example,

```matlab
[xP, yP] = PolarToCartesian(1, pi/4)
gives
xP = 0.7071 and yP = 0.7071
```

```matlab
[rho, theta] = CartesianToPolar(3, 4)
gives
rho = 5 and theta = 0.9273
```
Example: Evaluating an expression
Functions are very useful for evaluating a mathematical expression from the Command Window or from an m-script. We will consider evaluating the sinc function that is widely used in physics and engineering with the function sinc.m. The sinc function can be expressed as a function of the single variable \( \theta \) where \( \theta \) is an angle in radians by
\[
sinc(\theta) = \frac{\sin(\theta)}{\theta}
\]
The sinc function approaches 1 as \( \theta \) approaches 1, but this causes a problem in Matlab when you try to divide by zero. This can problem can be overcome by using the Matlab function \( \text{eps} \) which is the smallest difference between two numbers.

function result = sinc(theta)
% Function to evaluate the sinc function
result = sin(theta + eps) ./ (theta + eps);

For example, \( \text{sinc}(0) \) gives then answer 1. For the array input for \( \theta \)
\[
\begin{array}{cccccc}
\text{theta} & 0.0000 & 0.2500 & 0.5000 & 0.7500 & 1.0000 \\
\text{sinc(theta)} & 1.0000 & 0.9896 & 0.9589 & 0.9089 & 0.8415 \\
\end{array}
\]