Mathematical Computing with MATLAB

By

Dr. Rajesh Kumar

PhD, PDF (NUS, Singapore)
SMIEEE, FIETE, MIE (I), LMCSI, SMIACSIT, LMISTE, MIAENG
Associate Professor, Department of Electrical Engineering
Adjunct Associate Professor, Centre of Energy
Malaviya National Institute of Technology, Jaipur, India, 302017
Tel: (91) 9549654481
rkumar@ieee.org, rkumar.ee@gmail.com, rkumar.ee@mnit.ac.in
http://drrajeshkumar.wordpress.com
Talk Outline

Topics:
1. Introduction
2. Matlab Desktop
3. Matlab Programming
4. Matrix Operations
5. Basic Mathematics with MATLAB
6. Special purposes matrices
7. Basic Functions
8. Solving linear equations
9. Solving differential equations
10. Symbolic Math
Development Philosophy

Major Software Characteristics

• Matrix-Based Numeric Computation
• High-Level Programming Language
• Graphics and Visualization
• Application Specific Toolboxes

Open & Extensible System Architecture

System Interfaces:

• Extensive Data I/O Facility
• Custom C, Fortran (MATLAB Calls)

PC/ UNIX Support
MATLAB is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran.
MATLAB

Dr. Rajesh kumar, MNIT, Jaipur
Starting Matlab

- **Windows:** Start menu → Matlab → Matlab
- **Unix:** Terminal window → type `matlab`
Getting Started with MATLAB

Enter MATLAB functions at the Command Window prompt.

To get started, select MATLAB Help or Demos

The Command History maintains a record of the MATLAB functions you ran.
Desktop Overview

Menus change, depending on the tool you are currently using.

View or change the current directory.

Click to move Command Window outside of desktop (undock).

To get started, select MATLAB Help or Demos

>>

Dr. Rajesh kumar, MNIT, Jaipur
1. Using **HELP** menu → MATLAB Help
   
   HELP → Using Help Browser

2. >> helpdesk      Opens the Help browser.

3. >> help commandname/toolboxname/functionname
   
   Ex: >> help sin

4. >> doc commandname/toolboxname/functionname
   
   displays the detailed info in the Help browser.
   
   Ex: >> doc sin

Other commands:

5. >> lookfor = helpdesk -> search
Help Browser

Tabs in the Help Navigator pane provide different ways to find information.

- Use the close box to hide the pane.
- Drag the separator bar to adjust the width of the panes.

View documentation in the display pane.

This section discusses techniques for plotting data and provides examples showing how to plot, annotate, and print graphs.

MATLAB Plotting Tools
- MATLAB Plotting Commands
- Annotating Graphs
- Creating Specialized Plots
- Displaying

Creating plots and setting graphic object properties
Plotting vector and matrix data in 2-D representations
Adding annotations, axis labels, titles, and legends to graphs
Creating bar graphs, histograms, contour plots and other specialized plots
Displaying and modifying
Workspace Browser and Array Editor

View and change values of array elements.

Arrange the display of array documents.

Use the document bar to access other documents open in the Editor/Debugger.

Comment selected lines and specify indenting style using the Text menu.

Find and replace text.

Set breakpoints where you want execution to pause so you can examine variables.

Use the document bar to view other variables you have open in the Array Editor.

Dr. Rajesh kumar, MNIT, Jaipur
Matlab Programming

- Matlab Variables
- Operators
- Functions

......
Matlab Variables

- A MATLAB variable is essentially **a tag that you assign to a value in memory**.
- MATLAB does not require any type declarations or dimension statements.
- When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage.
- If the variable already exists, MATLAB changes its contents.
- **Variable names** consist of **a letter**, followed by any number of **letters, digits, or underscores**.
- MATLAB uses only **the first 31 characters** of a variable name.
- MATLAB is **case sensitive**; it distinguishes between uppercase and lowercase letters.
- MATLAB stores variables in a part of memory called **workspace**.
- To view what is stored in a variable type its name.

**Types of Variables:**
MATLAB provides three basic types of variables:
- Local Variables
- Global Variables
- Persistent Variables
Matlab Variables

Rules for variable names:
- Make Sure Variable Names Are Valid
- Don't Use Function Names for Variables
- Check for Reserved Keywords
- Avoid Using i and j for Variables

Syntax:
variableName=Value;

Example:
>> a=5;
>> b=7;
>> c=a+b
>> method='linear'

How to remove a variable from workspace:

>> clear variableName
>> clear
removes all variables from the workspace (!!!!)

ans = default variable, when the result is not assign to a variable

Exercise: 1. Define a1=8 and b2=8, c1=a1+b2 variable=input('prompt') (>>help input)
2. Other commands: >> a3=input('a3=')
## Special Values

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ans</strong></td>
<td>Most recent answer (variable). If you do not assign an output variable to an expression, MATLAB automatically stores the result in ans.</td>
</tr>
<tr>
<td><strong>pi</strong></td>
<td>3.1415926535897...</td>
</tr>
<tr>
<td><strong>inf</strong></td>
<td>Infinity. Calculations like n/0, where n is any nonzero real value, result in inf.</td>
</tr>
<tr>
<td><strong>NaN</strong></td>
<td>Not-a-Number, an invalid numeric value. Expressions like 0/0 and inf/inf result in a NaN, as do arithmetic operations involving a NaN. n/0, where n is complex, also returns NaN.</td>
</tr>
</tbody>
</table>
## Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>.*</td>
<td>Multiplication (element wise)</td>
</tr>
<tr>
<td>./</td>
<td>Right division (element wise)</td>
</tr>
<tr>
<td>.\</td>
<td>Left division (element wise)</td>
</tr>
<tr>
<td>+</td>
<td>Unary plus</td>
</tr>
<tr>
<td>-</td>
<td>Unary minus</td>
</tr>
<tr>
<td>:</td>
<td>Colon operator</td>
</tr>
<tr>
<td>.^</td>
<td>Power (element wise)</td>
</tr>
<tr>
<td>'</td>
<td>Transpose</td>
</tr>
<tr>
<td>*</td>
<td>Matrix multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Matrix right division</td>
</tr>
<tr>
<td>\</td>
<td>Matrix left division</td>
</tr>
<tr>
<td>^</td>
<td>Matrix power</td>
</tr>
</tbody>
</table>
### Relational and Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Returns 1 for every element location that is true (nonzero) in both arrays, and 0 for all other elements.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Returns 1 for every element location that is true (nonzero) in either one or the other, or both, arrays and 0 for all other elements.</td>
</tr>
<tr>
<td>~</td>
<td>Complements each element of input array, A.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>~=</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>

Dr. Rajesh kumar, MNIT, Jaipur
Matlab Functions

1. Standard elementary mathematical functions

```matlab
>> help elfun

Trigonometric (sin, cos)
Exponential (exp, log)
Complex (abs, angle)
Rounding and remainder (round)
```

2. Elementary matrices and matrix manipulation.

```matlab
>> help elmat
```

3. Specialized math functions.

```matlab
>> help specfun
```

<table>
<thead>
<tr>
<th>EPS</th>
<th>3.14159265...</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Imaginary unit, $\sqrt{-1}$</td>
</tr>
<tr>
<td>1</td>
<td>Same as i</td>
</tr>
<tr>
<td>eps</td>
<td>Floating-point relative precision, $\varepsilon = 2^{-52}$</td>
</tr>
<tr>
<td>Inf</td>
<td>Infinity</td>
</tr>
<tr>
<td>NaN</td>
<td>Not-a-number</td>
</tr>
</tbody>
</table>
M-files

Files that contain code in the MATLAB language are called M-files.
You create M-files using a text editor.
Use a M-file as any other MATLAB function or command.
A M-file is a plain text file.

Two kinds of M-files:

Scripts

do not accept input arguments or return output arguments
operate on data in the workspace.

Functions

can accept input arguments and return output arguments
internal variables are local to the function.

>> edit fileName
>> edit exSwitch
Flow Control

- **If**
  
  ```
  if n < 0            % If n negative, display error message.
      disp('Input must be positive');
  elseif rem(n,2) == 0 % If n positive and even, divide by 2.
      A = n/2;
  else
      A = (n+1)/2;     % If n positive and odd, increment and divide.
  end
  ```

- **switch**
  
  ```
  switch var
  case 1
      disp('1')
  case {2,3,4}
      disp('2 or 3 or 4')
  case 5
      disp('5')
  otherwise
      disp('something else')
  end
  ```
Flow Control

- **while**
  
  ```
  n = 1;
  while prod(1:n) < 1e100
    n = n + 1;
  end
  ```

- **for**
  
  ```
  for i = 2:6
    x(i) = 2*x(i-1)
  %end
  ```

- **break** terminates execution of a for or while loop.

- **return** causes execution to return to the invoking function
- Matrices
- Operators
- Functions

...
Using Matlab

Working with Matrices

- Matlab works with essentially only one kind of object, a rectangular numerical matrix.
- A matrix is a collection of numerical values that are organized into a specific configuration of rows and columns.
- The number of rows and columns can be any number.
- Example
  - 3 rows and 4 columns define a 3 x 4 matrix having 12 elements.
- A scalar is a single number and is represented by a 1 x 1 matrix in matlab.
- A vector is a one-dimensional array of numbers and is represented by an n x 1 column vector or a 1 x n row vector of n elements.
Using Matlab

Working with Matrices

c = 5.66  or  c = [5.66]      c is a scalar or a 1 x 1 matrix
Using Matlab

Working with Matrices

c = 5.66 or c = [5.66]  c is a scalar or a 1 x 1 matrix
x = [ 3.5, 33.22, 24.5 ]  x is a row vector or a 1 x 3 matrix
Using Matlab

Working with Matrices

c = 5.66 or c = [5.66]  
c is a scalar or a 1 x 1 matrix
x = [ 3.5, 33.22, 24.5 ]  
x is a row vector or a 1 x 3 matrix
x1 = [ 2
      5
      3
      -1]  
x1 is column vector or a 4 x 1 matrix
Using Matlab

Working with Matrices

c = 5.66 or \( c = [5.66] \)  \( c \) is a scalar or a 1 x 1 matrix

x = [ 3.5, 33.22, 24.5 ]  \( x \) is a row vector or a 1 x 3 matrix

x1 = [ 2
      5
      3
      -1]  \( x1 \) is column vector or a 4 x 1 matrix

A = [ 1  2  4
      2  -2  2
      0  3  5
      5  4  9 ]  \( A \) is a 4 x 3 matrix
Working with Matrices

- Spaces, commas, and semicolons are used to separate elements of a matrix
Using Matlab

Working with Matrices

- Spaces, commas, and semicolons are used to separate elements of a matrix

- Spaces or commas separate elements of a row

- [1 2 3 4] or [1,2,3,4]
Using Matlab

Working with Matrices

- Spaces, commas, and semicolons are used to separate elements of a matrix

- Spaces or commas separate elements of a row
  - \([1\ 2\ 3\ 4] \) or \([1,2,3,4]\)

- Semicolons separate columns
  - \([1,2,3,4;5,6,7,8;9,8,7,6] = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 8\ 7\ 6]\)
Using Matlab

Indexing Matrices

- A $m \times n$ matrix is defined by the number of $m$ rows and number of $n$ columns.
- An individual element of a matrix can be specified with the notation $A(i,j)$ or $A_{i,j}$ for the generalized element, or by $A(4,1)=5$ for a specific element.
Using Matlab

Indexing Matrices

- A $m \times n$ matrix is defined by the number of $m$ rows and number of $n$ columns.
- An individual element of a matrix can be specified with the notation $A(i,j)$ or $A_{i,j}$ for the generalized element, or by $A(4,1)=5$ for a specific element.

Example:

```
>> A = [1 2 4 5;6 3 8 2]            \ A is a 4 x 2 matrix
>> A(2,1)
Ans 6
```
Using Matlab

Indexing Matrices

- Specific elements of any matrix can be overwritten using the matrix index

Example:

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

\[
>> A(2,1) = 9
\]

Ans

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

- The ones and zeros functions can be used to create any $m \times n$ matrices composed entirely of ones or zeros.

Example

```matlab
a = ones(3,2)
b = zeros(1,5)
```

```
a = [1 1
     1 1
     1 1]
b = [0 0 0 0 0]
```
Subscripting and Indexing

- **Accessing Single Elements of a Matrix**
  \[ A(i,j) \]

- **Accessing Multiple Elements of a Matrix**
  \[ A(1,4) + A(2,4) + A(3,4) + A(4,4) \Rightarrow \text{sum}(A(1:4,4)) \text{ or } \text{sum}(A(:,\text{end})) \]
  The keyword end refers to the *last* row or column.

- **Expanding the Size of a Matrix**
  If you store a value in an element outside of the matrix, the size of the matrix increases to accommodate the new element, but If you try to access an element outside of the matrix, it is an error.

- **Deleting Rows and Columns**
  To delete the second column of \( X \), use
  \[ X(:,2) = [] \]

- **Concatenating Matrices \( A \) and \( B \)**
  \[ C=[A;B] \]
Matrix Operations
# Matrix: Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>A + B</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>A - B</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>A * B</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>A / B</td>
</tr>
<tr>
<td>\</td>
<td>Left division (described in &quot;Matrix&quot;)</td>
<td>A \ B</td>
</tr>
<tr>
<td>^</td>
<td>Power</td>
<td>A ^ B</td>
</tr>
<tr>
<td>'</td>
<td>Complex conjugate transpose</td>
<td>A'</td>
</tr>
<tr>
<td>( )</td>
<td>Specify evaluation order</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element-by-element multiplication</td>
<td>A .* B</td>
</tr>
<tr>
<td>./</td>
<td>Element-by-element division</td>
<td>A ./ B</td>
</tr>
<tr>
<td>.\</td>
<td>Element-by-element left division</td>
<td>A .\ B</td>
</tr>
<tr>
<td>.^</td>
<td>Element-by-element power</td>
<td>A .^ B</td>
</tr>
<tr>
<td>.''</td>
<td>Unconjugated array transpose</td>
<td>A .''</td>
</tr>
</tbody>
</table>

https://obrajeshkumar.wordpress.com/
Basic Matrix Operations

- First, let's create a simple vector with 9 elements called a. 
  \[ a = [1 \ 2 \ 3 \ 4 \ 6 \ 4 \ 3 \ 4 \ 5] \]

- Now let's add 2 to each element of our vector, a, and store the result in a new vector. Notice how MATLAB requires no special handling of vector or matrix math.
  
  \[ b = a + 2 \]

  \[
  b =
  \begin{bmatrix}
  3 & 4 & 5 & 6 & 8 & 6 & 5 & 6 & 7 
  \end{bmatrix}
  \]
Creating a matrix

Creating a matrix is as easy as making a vector, using semicolons (;) to separate the rows of a matrix.

```matlab
>> A = [1 2 0; 2 5 -1; 4 10 -1]
A =
     1     2     0
     2     5    -1
     4    10    -1
```
We can easily find the transpose of the matrix A.

\[
B = A' = \begin{bmatrix}
1 & 2 & 4 \\
2 & 5 & 10 \\
0 & -1 & -1
\end{bmatrix}
\]
BASIC MATHEMATICS

with MATLAB
Adding and Subtracting Matrices

```matlab
>> a=magic(3);
>> b=ones(3);
>> ans=a-b
```

```
ans =
   7     0     5
   2     4     6
   3     8     1
```

- Note that dimensions of matrices must match otherwise an error is reported.
- Right now assume magic and ones are special functions that return a 3 X 3 matrix each.
- Their meaning shall be made clear later.
Multiplication of Matrices

\[ \begin{array}{ccc}
5 & 12 & 24 \\
12 & 30 & 59 \\
24 & 59 & 117 \\
\end{array} \]

- We can also do element by element multiplication.

\[ \begin{array}{c}
\text{Notice the } \cdot \star \text{ operator.} \\
\text{This is just like dot product if u use two vectors!!!!} \\
\end{array} \]
Matrix: Operations

\[
A = \begin{bmatrix}
1 & 2 & 3 \\
2 & 3 & 1 \\
2 & 2 & 2
\end{bmatrix}, \quad B = \begin{bmatrix}
10 & 20 & 30 \\
11 & 21 & 31 \\
1 & 2 & 3
\end{bmatrix}, \quad \text{ans} = \begin{bmatrix}
35 & 68 & 101 \\
54 & 105 & 156 \\
44 & 86 & 128
\end{bmatrix}
\]

\[
A = \begin{bmatrix}
a_{11} & a_{12} & a_{13} \\
a_{21} & a_{22} & a_{23} \\
a_{31} & a_{32} & a_{33}
\end{bmatrix}, \quad B = \begin{bmatrix}
b_{11} & b_{12} & b_{13} \\
b_{21} & b_{22} & b_{23} \\
b_{31} & b_{32} & b_{33}
\end{bmatrix}, \quad \text{ans} = \begin{bmatrix}
10 & 40 & 90 \\
22 & 63 & 31 \\
2 & 4 & 6
\end{bmatrix}
\]
Determinant of a matrix

- A = [1 2 0; 2 5 -1; 4 10 -1]

- A =

- 1  2  0
- 2  5  -1
- 4 10 -1

- >> det(A)

- ans =

- 1
Inverse of a matrix

- It was never so easy

```matlab
>> X = inv(A)
X =
   5  2 -2
  -2 -1  1
  -2 -1  1
```

Dr. Rajesh kumar, MNIT, Jaipur
Functions to obtain eigen-values

```matlab
>> eig(A)
ans =
   3.7321
   0.2679
   1.0000
```
Diagonal Matrix

>> diag([1 2 3])
ans =
     1     0     0
     0     2     0
     0     0     3

- Note that input to diagonal function is a vector containing the diagonal matrices enclosed inside big brackets([ ])
Special purposes matrices

- Zeros.
- Ones.
- Identity matrix.
- Magic square.
Zeros

>> zeros(n)
   creates a n X n matrix of all zeros.
>> zeros(2)
   ans =
   0  0
   0  0

>> zeros(m,n)
   creates a m X n matrix of all zeros.
>> zeros(2,3)
   ans =
   0  0  0
   0  0  0
Ones

ones(n)

creates an \( n \times n \) matrix of all ones.

\[ 
\begin{bmatrix}
1 & 1 \\
1 & 1 \\
\end{bmatrix}
\]

>> ones(2)
ans =
   1   1
   1   1

>> ones(m,n)
creates a \( m \times n \) matrix of all ones.

>> ones(2,3)
ans =
   1   1   1
   1   1   1
Identity matrix

>>eye(n)

creates an identity matrix of order n.

>>eye(3)

ans =
    1     0     0
    0     1     0
    0     0     1
Magic Square

>> magic(n)

returns an n-by-n matrix constructed from the integers 1 through \( n^2 \) with equal row, column and diagonal sums.

>> magic(3)

ans =

8 1 6
3 5 7
4 9 2
The Colon Operator

- The colon, :, is one of the most important and most powerful MATLAB operators.

- It occurs in several different forms. The expression
  \[1:10\]
  is a row vector containing the integers from 1 to 10,
  \[1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10\]

- To obtain nonunit spacing, specify an increment.
  \[100:-7:50\]
  is \[100 \quad 93 \quad 86 \quad 79 \quad 72 \quad 65 \quad 58 \quad 51\]
Positive Integer Powers

• If $A$ is a square matrix and $p$ is a positive integer, then $A^p$ effectively multiplies $A$ by itself $p-1$ times. For example,

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

• $A =$

• $1$ $1$ $1$
• $1$ $2$ $3$
• $1$ $3$ $6$

• $X = A^2$

• $X =$

• $3$ $6$ $10$
• $6$ $14$ $25$
• $10$ $25$ $46$
Inverse and Fractional Powers

- If \( A \) is square and nonsingular, then \( A^{-p} \) effectively multiplies \( \text{inv} (A) \) by itself \( p-1 \) times.
- \( Y = A^{-3} \)

\[
\begin{align*}
Y &= \\
&= \begin{bmatrix}
145.0000 & -207.0000 & 81.0000 \\
-207.0000 & 298.0000 & -117.0000 \\
81.0000 & -117.0000 & 46.0000 \\
\end{bmatrix}
\end{align*}
\]

- Fractional powers, like \( A^{2/3} \), are also permitted; the results depend upon the distribution of the eigen values of the matrix.
Element-by-Element Powers

• The .^ operator produces element-by-element powers. For example,

• \( X = A.^2 \)

• \( A = \)

• 1 1 1
• 1 4 9
• 1 9 36
Partition of Matrix

- Lower triangular part of a matrix
- \( L = \text{tril}(X) \)
- \( L = \text{tril}(X,k) \)

\( L = \text{tril}(X,k) \) returns the elements on and below the \( k \)th diagonal of \( X \). \( k = 0 \) is the main diagonal, \( k > 0 \) is above the main diagonal, and \( k < 0 \) is below the main diagonal.

- \( \text{tril(ones(4,4),-1)} \)

\[ \text{ans} = \begin{array}{cccc}
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
\end{array} \]
Partition of Matrix

• Upper triangular part of a matrix

• Syntax
  • U = triu(X)
  • U = triu(X,k)

• Description
  • U = triu(X) returns the upper triangular part of X.
  • U = triu(X,k) returns the element on and above the kth diagonal of X. k = 0 is the main diagonal, k > 0 is above the main diagonal, and k < 0 is below the main diagonal.

• triu(ones(4,4),-1)
• ans =

• 1 1 1 1
• 1 1 1 1
• 0 1 1 1
• 0 0 1 1
CURVE FITTING with MATLAB
Curve fitting with tools

- >> x=[5 10 20 50 100]
- >> y=[15 33 53 140 301]
- >> plot(x,y,'o')
- Go to figure Tools
- Select Basic Fitting
Curve fitting with polynomial functions

- **polyfit**: `a=polyfit (x, y, n)`
  - `x, y` = data points, `n` = order of polynomial

- `x=[5 10 20 50 100]`
- `y=[15 33 53 140 301]`
- `p = polyfit(x,y,1)`
- `p = 2.9953 -2.4264` There are two coefficients and the polynomial is

\[ 2.9953x - 2.4264 \]
Interpolation

• Interp1 = one dimensional interpolation
  \[ y_{\text{new}} = \text{interp1}(x, y, x_{\text{new}}, 'method') \]
  • nearest
  • linear
  • cubic
  • Spline
• interp2 for 2D and interp3 for 3D
Data Analysis with MATLAB
Data Analysis with tools

• $x=\text{linspace}(0, \pi, 30)$
• $y=\sin(x)$
• $\text{plot}(x, y, 'o')$
• Go to figure Tools
• Select Data Statistics
Data Analysis with Commands

• $X=[1 \ 2 \ 3 \ 4 \ 5]$  
• $\text{mean}(x)=3$  
• $\text{median}(x)=3$  
• $\text{std}(x)=1.5811$  
• $\text{max}(x)=5$  
• $\text{min}(x)=1$  
• $\text{sum}(x)=15$  
• $\text{cumsum}(x)=[1 \ 3 \ 5 \ 6 \ 10 \ 15]$  
• $\text{prod}(x)=120$
BASIC FUNCTIONS
MATLAB provides a large set of basic functions to enhance your productivity speeds.

These include basic algebraic, trigonometric, hyperbolic, inverse trigonometric, support for complex algebra, signal processing, equation solving and differential equation solving functions.

These functions are easy to use and extensive help is also available about their usage.

We have a look at some of these common facilities now…
Trigonometric functions

- These functions are very flexible ie. They can operate on a scalar, a vector or a complete matrix.

```matlab
>> sin(pi/2)
ans = 1.0000
>> a = [-pi/2 -pi/4 pi/4 pi/2];
>> y = sin(a)
y =
    -1.0000   -0.7071    0.7071    1.0000
>> b = [-pi/2 -pi/4;pi/4 pi/2];
>> x = sin(b)
x =
    -1.0000   -0.7071
    0.7071    1.0000
```

Dr. Rajesh kumar, MNIT, Jaipur
Similarly we can use other trigonometric and inverse trigonometric functions like

>>cos
>>tan
>>sec
>>asin (sine inverse)
>>acos (cosine inverse)
>>atan (tangent inverse) and so on…
Hyperbolic functions

Their usage is similar to trigonometric functions. Various hyperbolic functions are:

- sinh
- cosh
- tanh
- asinh (hyperbolic sine inverse)
- acosh (hyperbolic cosine inverse)
- atanh (hyperbolic tangent inverse)
Exponential and logarithmic functions

```matlab
>> a=[-1 -0.5 -0.25; 0.25 0.5 1];
>> y=exp(a)
y =
  0.3679    0.6065    0.7788
  1.2840    1.6487    2.7183

a=[ 0.25 0.5 1];
>> y=log(a)
y =
  -1.3863   -0.6931         0

➢ The usage is quite simple and self-explanatory.
```
Complex numbers

• Unlike many other programming environments, MATLAB provides built-in support for complex numbers.
• This is so much beneficial for fast development of engineering applications.
• Two in-built constants i and j are provided whose value is equal to iota (the square root of -1).
Complex numbers

➢ If you type at command window:

\[
\text{>>sqrt(-1)}
\]

You get:

\[
\text{ans = 0 + 1.0000i}
\]

➢ On other programming environments, you are bound to get errors.
Complex numbers

- Some common complex number functions are:
  - `>>abs(z)` returns the magnitude of the complex number.
  - `>>real(z)` or `imag(z)` return the real/imaginary part of `z`.

- You can add, subtract, multiply and divide complex numbers like you operate with ordinary matrices.
Discrete mathematics

- **Factor**
  Prime factors.

- **Factorial**
  Factorial function.

- **Gcd**
  Greatest common divisor.

- **isprime**
  True for prime numbers.

- **lcm**
  Least common multiple.

- **Nchoosek**
  All combinations of $N$ elements taken $K$ at a time.

- **Perms**
  All possible permutations.

- **primes**
  Generate list of prime numbers.

- **rat, rats**
  Rational fraction approximation.
examples

$$\text{>> } f = \text{factor}(123)$$

$$f =$$

\[
\begin{array}{cc}
3 & 41 \\
\end{array}
\]

$$\text{>> } \text{lcm}(8, 40)$$

$$\text{ans } = 40$$
MATLAB at Engineering tasks
One of the most common tasks engineers face is to solve a variety of linear and differential equations.

In the following slides we show various options available with MATLAB to solve such equations.

In the end you will appreciate the easiness and versatility of MATLAB in solving such equations.
Solving linear equations

➢ A linear equation is any equation of the form.

Ax + By = C

➢ A system of linear equations is a set of linear equations that we usually want to solve at the same time; i.e., simultaneously.

For example, to solve for x and y in the system of linear equations

\[2x + y = 13\] and \[X - 3y = -18\]
For example, to solve for x and y in the system of linear equations

\[ 2x + y = 13 \text{ and } x - 3y = -18 \]

Matrix algebra represents this system as an equation involving three matrices: A for the left-side constants, X for the variables, and B for the right-side constants.

\[ A = \begin{bmatrix} 2 & 1 \\ 1 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 13 \\ -18 \end{bmatrix}, \quad X = \begin{bmatrix} x \\ y \end{bmatrix} \]

Thus \( AX = B \);
Solution by MATLAB

- \( A = \begin{bmatrix} 2 & 1 \\ 1 & -3 \end{bmatrix} \)
- \( B = \begin{bmatrix} 13 \\ -18 \end{bmatrix} \)
- \( X = A \backslash B \)

solves for \( X \) in \( A \times X = B \).

- \( X = \begin{bmatrix} 3 \\ 7 \end{bmatrix} \)
- Thus \( x=3 \) and \( y=7 \);
- Note the use of \( \backslash \) operator which is different from divide (\(/\)) symbol.
- Similarly we can solve for a system of \( n \) equations.
Solving Linear System

Linear system of algebraic equations:

\[-x_1 + x_2 + 2x_3 = 2\]
\[3x_1 - x_2 + x_3 = 6\]
\[-x_1 + 3x_2 + 4x_3 = \angle\]

\[Ax = b\]

\[>> A = \begin{bmatrix} -1 & 1 & 2 \\ 3 & -1 & 1 \\ -1 & 3 & 4 \end{bmatrix}\]

\[A = \begin{bmatrix} -1 & 1 & 2 \\ 3 & -1 & 1 \\ -1 & 3 & 4 \end{bmatrix}\]

\[>> b = [2 6 4]‘\]

\[b = \begin{bmatrix} 2 \\ 6 \\ 4 \end{bmatrix}\]

\[>> \text{rank}(A)\]

\[\text{ans} = 3\]

\[>> x = \text{inv}(A)*b\]

\[x = \begin{bmatrix} 1.0000 \\ -1.0000 \\ 2.0000 \end{bmatrix}\]

Dr. Rajesh kumar, MNIT, Jaipur
Differential equations

• Solving differential equations is as easy as linear equations.
• One can solve ordinary differential equations of any order.
• One can also solve simultaneous differential equations also.
Solving differential equations

- Consider the equation
- $\frac{dy}{dt} = 1 + y^2$
- The following call to `dsolve`
  - `dsolve('Dy=1+y^2')`
- uses $y$ as the dependent variable and $t$ as the default independent variable. The output of this command is
- $\text{ans} = \tan(t + C1)$
Higher order equations

• These can be solved as easily as first order equations.
• The computation time may be a bit long.
• Some practice is required as to understand the syntax.
A second order equation

Here is a second order differential equation with two initial conditions. The commands

```matlab
y = dsolve('D2y=cos(2*x)-y','y(0)=1','Dy(0) =0', 'x');
simplify(y)
```

Produce

```matlab
ans =4/3*cos(x)-2/3*cos(x)^2+1/3
```
Nonlinear equations

- They may have multiple solutions, even when initial conditions are given

\[ x = \text{dsolve}('\left(Dx\right)^2+x^2=1','x(0)=0') \]

- results in \[ x = \left[ \sin(t), -\sin(t) \right] \]

- It may take you some time initially to learn the syntax. Once done you are ready with a powerful mathematical tool.

Dr. Rajesh kumar, MNIT, Jaipur
Ordinary Differential Equations

- \([\text{time}, \text{solution}]=\text{ode23}(\text{ function}', \text{tspan}, \text{xo})\)
  - solution = solution matrix
  - function = function you want to solve
  - tspan = time interval
  - xo = initial condition

\[
\frac{dx}{dt} = x + t, \quad x(0) = 0
\]
Nonlinear Pendulum Eqn

\[ \ddot{\theta} + \omega^2 \sin \theta = 0; \]
\[ \theta(0) = 1, \quad \dot{\theta}(0) = 1 \]
\[ z_1 = \theta, \quad z_2 = \dot{\theta} \]
\[ \dot{z}_1 = z_2 \]
\[ \dot{z}_2 = -\omega^2 \sin(z_1) \]
First order Eqn

Write a function

\[
\text{function } zdot = \text{pend}(t,z);
\]

\[
ww = 1.56;
\]

\[
zdot = [z(2); -ww*\sin(z(1))]
\]
Execution

- $tspan=[0,20]$;
- $z0=[1;0]$;
- $[t,z]=ode23('pend',tspan,z0)$;
- $x=z(:,1);y=z(:,2)$;
- plot$(t,x,t,y)$
- figure
- plot$(x,y)$
Roots of polynomials

\[ x^5 - 3x^3 + x^2 - 9 = 0 \]

c=[1 0 -3 1 0 -9]

Roots(c)
Numerical Integration

• Integral=quad (‘function’, a, b)
  a and b are the limits of integration

\[ \int_{1/2}^{3/2} e^{-x^2} \, dx \]

• integral = dbguard(‘fun’, xmin, xmax, ymin, ymax)
Calculates the discrete fourier transform.

\[
>> Y = \text{fft}(X)
\]

returns the discrete Fourier transform (DFT) of vector X, computed with a fast Fourier transform (FFT) algorithm.
FFT example

- \( t = 0:0.001:0.6; \)
- \( x = \sin(2\pi 50t) + \sin(2\pi 120t); \)
- \( y = x + 2\text{randn(size(t))}; \)
- \( \text{plot}(1000\times t(1:50),y(1:50)) \)
- \( \text{title('Signal Corrupted with Zero-Mean Random Noise')} \)
- \( \text{xlabel('time (milliseconds)')} \)
It is difficult to identify the frequency components by looking at the original signal. Converting to the frequency domain, the discrete Fourier transform of the noisy signal $y$ is found by taking the 512-point fast Fourier transform (FFT)

$$Y = \text{fft}(y, 512);$$

The power spectrum, a measurement of the power at various frequencies, is $P_{yy} = \frac{Y \cdot \text{conj}(Y)}{512};$
Graph the first 257 points (the other 255 points are redundant) on a meaningful frequency axis:

- \( f = 1000 \times (0:256)/512; \)
- \( \text{plot}(f,Pyy(1:257)) \)
- \( \text{title}('Frequency content of y') \)
- \( \text{xlabel}('frequency (Hz)') \)
This represents the frequency content of \( y \) in the range from DC up to and including the Nyquist frequency. (The signal produces the strong peaks.)
Symbolic Math

- This is purely a MATLAB innovation.
- You will need a Symbolic Math toolbox along with the basic MATLAB product.
- By symbolic maths means that you shall be able to perform calculations in terms of symbols rather along with numbers.
Getting started with symbolic math

- First of all you will have to define your symbols as follows:

  >> syms x
  >> syms y

This creates two symbols x and y. Note that x and y now do not hold any number but MATLAB treats them as symbols.
Symbolic math

➢ Now you can do symbolic math with the recently declared symbol
➢ Example

```plaintext
>> x^2/x
ans = x
```

➢ Now you realize how easy is to perform all kinds of symbolic algebra with the toolbox.
➢ You can have symbolic complex numbers and symbolic equations too.
Symbolic Calculus

- Most of the digital computers provide support to numerical integration and differentiation.
- With Symbolic Toolbox of MATLAB you can have symbolic differentiation and integration.
- We jump start with an example
Example

>> syms x
>> f = sin(5*x)
➢ The command
>> diff(f)
differentiates f with respect to x:
ans = 5*cos(5*x)
Similarly integration

• Lets make it a bit complex

```matlab
>>syms x
>>f=exp(x)*cos(x)+exp(-x)*sin(x)
>>int(f)
>>ans =
    1/2*exp(x)*cos(x)+1/2*exp(x)*sin(x)-
    1/2*exp(-x)*cos(x)-1/2*exp(-x)*sin(x)
```

➢ Try experimenting with these functions.. You will find this tool very handy.
Graphs and plots
Creating graphs in MATLAB is as easy as one command. Let's plot the result of our vector addition with grid lines.

Let \( b = [3 \ 4 \ 5 \ 6 \ 8 \ 6 \ 5 \ 6 \ 7] \)

- \( \text{plot}(b) \)
- \( \text{grid on} \)
X versus Y plots

>> plot(x,y)

➢ X and Y must be vectors of same size
➢ For example consider the following set of commands

>> t=-2*pi:pi/32:2*pi;
>> phi=pi/8;
>> omega=1;
>> y=sin(omega*t+phi);
>> plot(t,y);
>> grid on;
Note:

- $\pi$ is a inbuilt constant in MATLAB.
- $\sin$ is a inbuilt MATLAB function.
- A large number of similar inbuilt functions are available for your help.

Eg. $\sin$, $\cos$, $\tan$, $\exp$, $\log$, $\sinh$, $\atan$ and so on
The plot

\[ y = \sin(\omega t + \phi) \]
Matlab - Plotting

```matlab
plot

Syntax:
plot(y);    plot(x,y);    plot(x,y,s)
```

The plot function has different forms, depending on the input arguments.

If \( y \) is a vector, \texttt{plot(y)} produces a piecewise linear graph of the elements of \( y \) versus the index of the elements of \( y \).

If you specify two vectors as arguments, \texttt{plot(x,y)} produces a graph of \( y \) versus \( x \).
Matlab - Plotting

```matlab
plot(x, y, s);
```

`s` allows to plot: colors, symbols, different lines

<table>
<thead>
<tr>
<th>Color</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>blue</td>
<td>. point</td>
</tr>
<tr>
<td>g</td>
<td>green</td>
<td>o circle</td>
</tr>
<tr>
<td>r</td>
<td>red</td>
<td>x x-mark</td>
</tr>
<tr>
<td>c</td>
<td>cyan</td>
<td>+ plus</td>
</tr>
<tr>
<td>m</td>
<td>magenta</td>
<td>star</td>
</tr>
<tr>
<td>y</td>
<td>yellow</td>
<td>s square</td>
</tr>
<tr>
<td>k</td>
<td>black</td>
<td>d diamond</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

```matlab
plot(x, y, 'c+:')
```
plots a cyan dotted line with a plus at each data point.
Matlab - Plotting

clear

t=0:0.01:10; % time seconds
signalSin=sin(2*pi*t); % signal1 - frequency =1 Hz
signalCos=0.5*cos(2*pi*t); % signal2 - frequency =1 Hz

figure
plot(t,signalSin);

hold on
plot(t,signalCos, '-*r');

xlabel('time'); ylabel('signal');
legend('Sin', 'Cos');
title('Two Signals'; 'FontSize',12)

Other commands: figure xlabel ylabel legend, title
Visualization (optional)

plot, plotyy, stem, subplot

plot(Y)
plot(X1,Y1,...)
plot(X1,Y1,LineSpec,PropertyName',PropertyValue,...)

plotyy (X1,Y1,X2,Y2) plots Y1 versus X1 with y-axis labeling on the left and plots Y2 versus X2 with y-axis labeling on the right.

stem (X,Y) plots the data sequence Y at the values specified in X.

subplot (m,n,p), breaks the Figure window into an m-by-n matrix of small axes, selects the p-th axes for the current plot
Visualization

%%%% plot commands %%%%%

t=1:1:20; x=sin(t/5);
y=0.01*x.*exp(-t);

figure
subplot(2,2,1)
plot(t,x,'--rs','LineWidth',2,'MarkerEdgeColor','k',... 
'MarkerFaceColor','g','MarkerSize',6);
title('Plot 1');

subplot(2,2,2)
stem(t,x,'--rs','LineWidth',2);
title('Plot 2')
subplot(2,2,3);
bar(t,x); title('Plot 3')

subplot(2,2,4)
plotyy(t,x, t,y); title('plotyy')

see: testPlot.m
Visualization (optional)

Line Plots

Bar Plots

3D Surface

Vector Fields

Dr. Rajesh kumar, MNIT, Jaipur
MATLAB has versatile plotting features.
Even in 2-D plots there are a lot of options.
Various types of 3-D plots are available like

1. Plot3 (A function for simple 3-d plot)
2. Mesh and Surface Plots.
Plot3

➢ For example consider following code

```matlab
>> t = 0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t)
>> axis square;
grid on
```
Mesh and Surface Plots

- \( [X,Y] = \text{meshgrid}(-8:.5:8); \)
- \( R = \sqrt{X.^2 + Y.^2} + \text{eps}; \)
- \( Z = \sin(R)/R; \)
- \( \text{mesh}(X,Y,Z,'\text{EdgeColor}', \text{'red'}); \)
Surface plots

```matlab
>>k = 5;
>>n = 2^k-1;
>>[x,y,z] = sphere(n);
>>c = hadamard(2^k);
>>surf(x,y,z,c);
>>colormap([1 1 0; 0 1 1])
>>axis equal
```
3-D Contour Plots

>>[X,Y] = meshgrid([-2:.25:2]);
>>Z = X.*exp(-X.^2-Y.^2);
>>contour3(X,Y,Z,30)
>>surface(X,Y,Z,'EdgeColor',[.8 .8 .8],'FaceColor','none')
>>grid off
>>view(-15,25)
>>colormap cool
Contour Surface Plots

```matlab
>>[C,h] = contourf(peaks(20),10);
>>colormap autumn
```
This is just to illustrate the plotting powers of MATLAB.

- A variety of options is available to specify the 3D plots which may be useful for different analyses purposes.
- You are highly encouraged to refer to the help available with MATLAB and learn about various plots.
- 3D plots your work easy to comprehend and none the less attractive.
A very special note

- The number of options available in MATLAB are beyond the scope of a particular person to remember.
- So a user of MATLAB has to constantly refer to help for obtaining information about various functions, syntaxes and function options.
- Feel free to constantly refer to help for any such purpose.