Engineering and Scientific Applications: Using MatLab® for Data Processing and Visualization

Syamal K. Sen

Department of Mathematical Sciences, Florida Institute of Technology, 150 West University Boulevard, Melbourne, FL 32901-6975, United States

sksen@fit.edu

and

Gholam Ali Shaykhian

National Aeronautics and Space Administration (NASA), Information Technology (IT) Directorate, Technical Integration Office (IT-G) Kennedy Space Center, FL 32899, United States

ali.shaykhian@nasa.gov

Abstract: MatLab® (MATrix LABoratory) is a numerical computation and simulation tool that is used by thousands Scientists and Engineers in many countries. MatLab does purely numerical calculations, which can be used as a glorified calculator or interpreter programming language; its real strength is in matrix manipulations. Computer algebra functionalities are achieved within the MatLab environment using "symbolic" toolbox. This feature is similar to computer algebra programs, provided by Maple or Mathematica to calculate with mathematical equations using symbolic operations.

MatLab in its interpreter programming language form (command interface) is similar with well known programming languages such as C/C++, support data structures and cell arrays to define classes in object oriented programming. As such, MatLab is equipped with most of the essential constructs of a higher programming language. MatLab is packaged with an editor and debugging functionality useful to perform analysis of large MatLab programs and find errors.

We believe there are many ways to approach real-world problems; prescribed methods to ensure foregoing solutions are incorporated in design and analysis of data processing and visualization can benefit engineers and scientist in gaining wider insight in actual implementation of their perspective experiments. This presentation will focus on data processing and visualizations aspects of engineering and scientific applications. Specifically, it will discuss methods and techniques to perform intermediate-level data processing covering engineering and scientific problems. MatLab programming techniques including reading various data files formats to produce customized publication-quality graphics, importing engineering and/or scientific data, organizing data in tabular format, exporting data to be used by other software programs such as Microsoft Excel, data presentation and visualization will be discussed.
The presentation will emphasize creating practical scripts (programs) that extend the basic features of MatLab. Topics include

- Matrix and vector analysis and manipulations
- Mathematical functions
- Symbolic calculations & functions
- Import/export data files
- Program logic and flow control
- Writing function and passing parameters
- Test application programs
Sixth International Conference on Dynamic Systems and Applications!

Engineering and Scientific Applications: Using MatLab® for Data Processing and Visualization

Sixth International Conference on Dynamic Systems and Applications

May, 25-28, 2011
www.dynamicpublishers.com/icdsa6.htm
Morehouse College, Atlanta, GA, 30314, USA.

Syamal K. Sen
Department of Mathematical Sciences
Florida Institute of Technology
150 West University Boulevard
Melbourne, FL 32901-6975
sksen@fit.edu

Gholam Ali Shaykhian
Information Technology (IT) Directorate
Technical Integration Office (IT-G)
NASA-KSC
Kennedy Space Center, FL 32899
ali.shaykhian@nasa.gov
Agenda:

- Abstract
- Matrix and vector analysis and manipulations
- Mathematical functions
- Symbolic calculations & functions
- Import/export data files
- Writing function and passing parameters
- Test application programs
Abstract

MatLab® (MATrix LABoratory) is a numerical computation and simulation tool that is used by thousands of Scientists and Engineers in many countries. MatLab does purely numerical calculations, which can be used as a glorified calculator or interpreter programming language; its real strength is in matrix manipulations. Computer algebra functionalities are achieved within the MatLab environment using "symbolic" toolbox. This feature is similar to computer algebra programs, provided by Maple or Mathematica to calculate with mathematical equations using symbolic operations.

MatLab in its interpreter programming language form (command interface) is similar to well known programming languages such as C/C++, support data structures and cell arrays to define classes in object oriented programming. As such, MatLab is equipped with most of the essential constructs of a higher programming language. MatLab is packaged with an editor and debugging functionality useful to perform analysis of large MatLab programs and find errors.

We believe there are many ways to approach real-world problems; prescribed methods to ensure foregoing solutions are incorporated in design and analysis of data processing and visualization can benefit engineers and scientist in gaining wider insight in actual implementation of their perspective experiments/design. This presentation will focus on data processing and visualization aspects of engineering and scientific applications. Specifically, it will discuss methods and techniques to perform intermediate-level data processing covering engineering and scientific problems. MatLab programming techniques including reading various data files formats to produce customized publication-quality graphics, importing engineering and/or scientific data, organizing data in tabular format, exporting data to be used by other software programs such as Microsoft Excel, data presentation and visualization will be discussed.

The presentation will emphasize creating practical scripts (programs) that extend the basic features of MatLab. Topics include:

- Matrix and vector analysis and manipulations
- Symbolic calculations & functions
- Program logic and flow control
- Test application programs
- Mathematical functions
- Import/export data files
- Writing function and passing parameters
Vector

Row vector
There are different ways to declare a row vector

A row vector with 5 different elements declared and initialized below:

```plaintext
>> a = [1, 2, 3, 4, 5]
```

```plaintext
>> a = [1 2 3 4 5]
```

```plaintext
>> a = 1:5
```

Column vector
Column vector are defined by using a ; between each element of vector

Column vector can also be created by transposing a row vector

Single quote (') is used to transpose a matrix

```plaintext
>> b = [1; 2; 3; 4; 5]
```

```plaintext
>> b = a'
```

```plaintext
>> a = [1:5]
```
Matrix

Matrix (two dimensional)

$$m = \begin{bmatrix} 1.2, 3, 4; & -3.7, -2, 5; & 1, 2, 3 \end{bmatrix}$$

m =

\[
\begin{array}{ccc}
1.2000 & 3.0000 & 4.0000 \\
-3.7000 & -2.0000 & 5.0000 \\
1.0000 & 2.0000 & 3.0000 \\
\end{array}
\]

Note: The default representation of numeric values can be changed through the File>Preferences>Numeric Format
Work with Matrix

Add extra row to a Matrix

```
>> a = [1 2 4; 2 4 6]
a =
     1  2  4
     2  4  6
>> a = [a; 7 7 7]
a =
     1  2  4
     2  4  6
     7  7  7
```

Add extra column to a Matrix

```
>> a = [1 2 4; 2 4 6]
a =
     1  2  4
     2  4  6
>> a = [a, [9; 9]]
a =
     1  2  4  9
     2  4  6  9
```
Work with Matrix

Colon Operator:
• Used to define new matrices
• Modify existing matrices
• Extract data from existing matrices

% delete second column
>> a(:,2) = []
a =
2 6
3 7
1 9

% delete third row
>> a(3,:) = []
a =
2 4 6
3 5 7
Work with Matrix

Select a row/ column from a Matrix and assign it to a vector

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

```matlab
a =
    2     4     6
    3     5     7
    1     8     9
```

% select the second column of Matrix a and assign it to vector b

```matlab
>> b = a(:,2)
```

```matlab
b =
    4
    5
    8
```
Work with Matrix

Select a row/ column from a Matrix and assign it to a vector

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

% select the second column of Matrix a and assign it to vector b

\[ b = a(:,2) \]

\[ b = \begin{bmatrix} 4 \\ 5 \\ 8 \end{bmatrix} \]
Work with Matrix

Convert a matrix to a column vector use the MatLab colon operator (:)  

>> v=b(:)

> b=[2 3;1 6;7 8]

b =
  2   3
  1   6
  7   8

v =
  2
  1
  7
  3
  6
  8
Elementary Matrix Manipulations

The command `help elmat` provides a complete list of elementary matrices and matrix manipulation commands

```plaintext
>> help elmat
```

- **zeros** - Zeros array, Creates a matrix of all zeros
- **ones** - Ones array, Creates a matrix of all ones
- **eye** - Identity matrix.
- **repmat** - Replicate and tile array.
- **rand** - Uniformly distributed random numbers.
- **randn** - Normally distributed random numbers.
- **linspace** - Linearly spaced vector.
- **logspace** - Logarithmically spaced vector.
- **freqspace** - Frequency spacing for frequency response.
- **meshgrid** - X and Y arrays for 3-D plots.
- **:** - Regularly spaced vector and index into matrix.
Elementary Matrix Manipulations

• reshape - Change size.
• diag - Diagonal matrices and diagonals of matrix, Extracts a diagonal or creates an identity matrix
• blkdiag - Block diagonal concatenation.
• tril - Extract lower triangular part.
• triu - Extract upper triangular part.
• fliplr - Flip matrix in left/right direction.

• flipud - Flip matrix in up/down direction.
• flipdim - Flip matrix along specified dimension.
• rot90 - Rotate matrix 90 degrees.
• find - Find indices of nonzero elements.
• end - Last index.
• sub2ind - Linear index from multiple subscripts.
• ind2sub - Multiple subscripts from linear index.
Specialized matrices

- compan - Companion matrix.
- gallery - Higham test matrices.
- hadamard - Hadamard matrix.
- hankel - Hankel matrix.
- hilb - Hilbert matrix.
- invhilb - Inverse Hilbert matrix.
- magic - Magic square, Creates a "magic" matrix
- pascal - Pascal matrix.
- rosser - Classic symmetric eigenvalue test problem.
- toeplitz - Toeplitz matrix.
- vander - Vandermonde matrix.
- wilkinson - Wilkinson's eigenvalue test matrix.
Work with Matrix

MatLab ones() function
ones(N) function returns a N-by-N matrix of ones
ones(M,N) returns a M-by-N matrix of ones

Example: Initialize 2 x 3 matrix with ones
>> b = ones(2,3)

b =
1 1 1
1 1 1

MatLab size() function
size(X) returns the size of a matrix (number of rows and columns in the matrix)
b =
1 1 1
1 1 1

>> size(b)
ans = 2 3

MatLab length() function
length(X) returns the length of (row or column) vector X
>> length(b)
ans = 3
Matrix Arithmetic Operations

MatLab matrix arithmetic operations are:

- A+B
- A-B
- A*B A.*B
- A\B A./B
- A^B A.^B
- A' A.'

+ **matrix addition** is the operation of adding two matrices by adding the corresponding entries together.

A + B adds A and B
A and B must have the same dimensions, unless one is scalar.

```matlab
>> A = [1 2; 3 4]; B = [3 5; 6 7];
>> A+B
ans =
    4    7
    9   11
```
Matrix Arithmetic Operations

- **matrix subtraction** is the operation of subtracting two matrices by subtracting the corresponding entries together.

A - B subtracts B from A
A and B must have the same dimensions, unless one is scalar.

```matlab
>> A = [1 2; 3 4]; B = [3 5; 6 7];
A - B
ans =
    -2   -3
    -3   -3
```

* **matrix multiplication** is the operation of multiplying two matrices

>> K = M * L
the number of columns in the matrix on the left (M) must equal the number of rows in the matrix on the right (L)

Note: M * L \neq L * M

```matlab
>> M = [1 2 3; 4 5 6 ]
>> N = [2 3; 5 6; 2 7]
>> K = M*N
K =
     18  36
     45  84
```
Matrix Arithmetic Operations

.* term-by-term multiplication (array multiplication)

A.*B is the entry-by-entry product of A and B

A and B must have the same dimensions

>> M= [1 2; 3 4], N=[3 5; 6 7]
>> M.*N
ans =
    3    10
   18    28
Matrix Arithmetic Operations

**matrix division**
MatLab supports two division operators, namely right division / and left division \\

/ Matrix right division
X = B/A solves the symbolic linear equation X*A = B
Note that B/A is the same as (A.'\B.')

\ Matrix left division
X = A\B solves the symbolic linear equations A*X = B
Note that A\B is roughly equivalent to inv(A)*B.

\ Array left division
A./B is the matrix with entries A(i,j)/B(i,j)
A and B must have the same dimensions, unless one is scalar.

." Array right division
A./B is the matrix with entries A(i,j)/B(i,j)
A and B must have the same dimensions, unless one is scalar.
Matrix Arithmetic Operations

\[ \begin{align*}
X_2 + X_3 &= 5 \\
3X_1 + X_3 &= 6 \\
-X_1 + X_2 &= 1
\end{align*} \]

\[ \begin{align*}
A &= \begin{bmatrix} 0 & 1 & 1 \\ 3 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix} \\
b &= [5; 6; 1]
\]

\[ x = A \backslash b \quad \text{% left division} \]

\[ x = \begin{bmatrix} 1.0000 \\ 2.0000 \\ 3.0000 \end{bmatrix} \]
MatLab Mathematical Functions

MatLab has several standard (preprogrammed) mathematical functions.

These preprogrammed functions are grouped as:
- Trigonometric Functions
- Exponential Functions
- Complex Functions
- Rounding and remainder functions
MatLab Mathematical Functions (Trigonometric)

acos  Inverse cosine.
acosh Inverse hyperbolic cosine.
acot  Inverse cotangent.
acoth Inverse hyperbolic cotangent.
acsc  Inverse cosecant.
acsch Inverse hyperbolic cosecant.
asec  Inverse secant.
asech Inverse hyperbolic secant.
asin  Inverse sine.
asinh Inverse hyperbolic sine.
atan  Inverse tangent.
atan2  Four quadrant inverse tangent.
atanh Inverse hyperbolic tangent.
cos   Cosine.
cosh  Hyperbolic cosine.
cot   Cotangent.
coth  Hyperbolic cotangent.
csc   Cosecant.
csch  Hyperbolic cosecant.
sec   Secant.
sech  Hyperbolic secant.
sin   Sine.
sinh  Hyperbolic sine.
tan   Tangent.
tanh  Hyperbolic tangent.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp</td>
<td>Exponential ((e^x)).</td>
</tr>
<tr>
<td>Log</td>
<td>Natural logarithm.</td>
</tr>
<tr>
<td>Log10</td>
<td>Common (base 10) logarithm.</td>
</tr>
<tr>
<td>Log2</td>
<td>Base 2 logarithm and dissect floating-point numbers.</td>
</tr>
<tr>
<td>nextpow2</td>
<td>Next higher power of 2.</td>
</tr>
<tr>
<td>Pow2</td>
<td>Base 2 power and scale floating-point numbers.</td>
</tr>
<tr>
<td>Reallog</td>
<td>Guarantee output from log is a noncomplex matrix.</td>
</tr>
<tr>
<td>reallog10</td>
<td>Guarantee output from log10 is a noncomplex matrix.</td>
</tr>
<tr>
<td>realpow</td>
<td>Guarantee output from power is a noncomplex matrix.</td>
</tr>
<tr>
<td>Realsqrt</td>
<td>Guarantee output from sqrt is a noncomplex matrix.</td>
</tr>
<tr>
<td>Sqrt</td>
<td>Square root.</td>
</tr>
<tr>
<td>Abs</td>
<td>Absolute value.</td>
</tr>
<tr>
<td>Angle</td>
<td>Phase angle.</td>
</tr>
<tr>
<td>Conj</td>
<td>Complex conjugate.</td>
</tr>
<tr>
<td>cplxpair</td>
<td>Sort numbers into complex conjugate pairs.</td>
</tr>
<tr>
<td>Imag</td>
<td>Complex imaginary part.</td>
</tr>
<tr>
<td>Isreal</td>
<td>True for noncomplex arrays.</td>
</tr>
<tr>
<td>Real</td>
<td>Real part of complex array.</td>
</tr>
<tr>
<td>Unwrap</td>
<td>Remove phase angle jumps across 360° boundaries.</td>
</tr>
</tbody>
</table>
MatLab Mathematical Functions (Statistical /Discrete Mathematics Functions)

- **mean**: arithmetic mean or average value of elements
- **median**: median value of elements
- **min**: smallest component
- **max**: largest component
- **var**: variance of the elements in a vector
- **std**: standard deviation from the mean of elements
- **sum**: sum of elements
- **prod**: product of elements
- **sort**: sorting elements within a vector
- **sortrows**: sorting rows within a matrix by values in a column
- **cov**: variance of a vector or covariance of a matrix
- **corrcoef**: correlation coefficient

**Discrete Mathematics**

- **factor(x)**: returns a vector containing the prime factors of x
- **gcd(x,y)**: greatest common denominator
- **lcm(x)**: lowest common multiple
- **rats(x)**: represent x as a fraction
- **factorial(x)**: returns factorial of x
- **primes(x)**: generates a list of prime numbers less than or equal to x
- **isprime(x)**: returns 1 if the elements of x which are prime, 0 otherwise
MatLab plot() function

plot() function is used to plot a two dimensional plot
plot3() function is used to plot a three dimensional plot
A plot can be made using various symbols, colors and line types
Line types, plot symbols and colors for plot() or plot3() functions are represented through a character string

For example, a character string 'go:' means a green dotted line with a circle at each data point
plot(X,Y,'go:') plots a green dotted line with a circle at each data point

plot3(X,Y,Z,'ks-') plots a black dashdot line with square at each point

<table>
<thead>
<tr>
<th>Color</th>
<th>Symbols</th>
<th>Line types</th>
</tr>
</thead>
<tbody>
<tr>
<td>b blue</td>
<td>. point</td>
<td>- solid</td>
</tr>
<tr>
<td>g green</td>
<td>o circle</td>
<td>: dotted</td>
</tr>
<tr>
<td>r red</td>
<td>x x-mark</td>
<td></td>
</tr>
<tr>
<td>c cyan</td>
<td>+ plus</td>
<td>-- dashed</td>
</tr>
<tr>
<td>m magenta</td>
<td>* star</td>
<td></td>
</tr>
<tr>
<td>y yellow</td>
<td>s square</td>
<td></td>
</tr>
<tr>
<td>k black</td>
<td>d diamond</td>
<td></td>
</tr>
<tr>
<td>p pentagram</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h hexagram</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Line types</th>
</tr>
</thead>
<tbody>
<tr>
<td>v triangle (down)</td>
<td></td>
</tr>
<tr>
<td>^ triangle (up)</td>
<td></td>
</tr>
<tr>
<td>&lt; triangle (left)</td>
<td></td>
</tr>
<tr>
<td>&gt; triangle (right)</td>
<td></td>
</tr>
</tbody>
</table>
MatLab plot() function

```matlab
>> angle = 0:pi/30:2*pi; Y = sin(angle)
>> plot(angle, Y)
```

plot(angle, Y, 'r:') plots a red dotted line with a circle at each data point

```matlab
>> angle = 0:pi/30:2*pi; Y = sin(angle)
>> plot(angle, Y, 'ro:')
```
MatLab plot() function

```matlab
>> angle=0:0.1:1
>> plot(cos(2*pi*angle),
      sin(2*pi*angle), 'ro-'); axis square
```

```matlab
>> angle=0:0.05:1
>> plot(cos(2*pi*angle),
      sin(2*pi*angle), 'ro-'); axis square
```
MatLab Multiple function plots

```matlab
>> angle = 0:pi/30:2*pi
>> f1 = exp(-2*angle)
>> f2 = sin(angle*3)
>> f3 = cos(angle*4)
>> plot(angle,[f1; f2; f3])
>> % or plot(angle, f1, angle, f2, angle, f3)
```

```matlab
>> angle = 0:pi/30:2*pi
>> f1 = exp(-2*angle)
>> f2 = sin(angle*3)
>> f3 = cos(angle*4)
>> plot(angle,f1,'ro:', angle, f2, 'g*-',angle, f3, 'bh--')
```
MatLab Multiple function plots

```matlab
>> angle = linspace(0,10*pi,100)
>> % generates 100 points between 0 and 10* pi
>> y = cos(angle); z = sin(angle)
>> plot3(angle, y, z); grid ...
>> xlabel('Angle'); ylabel('cos(x)'); zlabel('sin(x)')
```

```matlab
>> angle=-5:0.01:5; plot3(cos(2*pi*angle), sin(2*pi*angle),angle)
```
MatLab three-dimensional plots

```matlab
>> % A three-dimensional plot using the mesh command
>> x = (-3:0.1:3); % grid frame in x direction
>> y = (-3:0.1:3)'; % grid frame in y direction
>> v = ones(length(x),1); % auxiliary vector
>> X = v*x; % grid matrix of the x values
>> Y = y*v'; % grid matrix of the y values
>> f = sin(X.^2+Y.^2).*exp(-0.2*(X.^2+Y.^2)); % function value
>> mesh(X, Y, f) % mesh plot with mesh
>> zlabel('f = sin(X.^2+Y.^2).*exp(-0.2*(X.^2+Y.^2))')
>> xlabel('X')
>> ylabel('Y')
```

![Example of a three-dimensional plot using MatLab](image)
MatLab three-dimensional plots

```matlab
>> % A three-dimensional plot using the mesh command
>> x = [-2:0.2:2]; y = [-2:0.2:2];
>> [X,Y] = meshgrid(x,y);
>> Z = X.*exp(-X.^2-Y.^2);
>> mesh(X,Y,Z);
>> xlabel('X')
>> ylabel('Y')
>> zlabel('Z = X.*exp(-X.^2-Y.^2)')
```
MatLab three-dimensional plots

```matlab
>> % A three-dimensional plot using the surf command
>> x = [-2:0.2:2]; y = [-2:0.2:2];
>> [X,Y] = meshgrid(x,y);
>> Z = ((8*X+Y).*(cos(X)-cos(2*Y)).^2)/(4*sqrt(0.8+(X-4.2).^2+2*(Y-7).^2))+Y;
>> surf(X,Y,Z);
>> xlabel('X')
>> ylabel('Y')
>> zlabel('Z ')
```

![3D plot](image)

\[ Z = \frac{(8X+Y)\cdot \left(\cos(X)-\cos(2Y)\right)^2}{4\sqrt{0.8+(X-4.2)^2+2(Y-7)^2}} + Y \]
MatLab three-dimensional plots

```matlab
>> % A three-dimensional plot using the surf command
>> x = [-2:0.2:2]; y = [-2:0.2:2];
>> [X,Y] = meshgrid(x,y);
>> Z = Z = (1-X).^2+100*(X.^2-Y).^2
>> surf(X,Y,Z);
>> xlabel('X')
>> ylabel('Y')
>> zlabel('Z ')
```

\[ Z = (1-X)^2 + 100(X^2-Y)^2 \]
Symbolic Calculations

MATLAB is a numerical simulation tool (not a symbolic algebraic)

MatLab provides a Symbolic Math toolbox to perform symbolic calculations

Type the command help symbolic to get the MatLab symbolic capabilities

The Symbolic Math Toolbox uses "symbolic objects" produced by the "sym" function. For example, the statement

x = sym('x') produces a symbolic variable named x

The statements x = sym('x'); y = sym('y'); can be combined into one statement involving the "syms" function.

syms x y

Symbolic variables can be use in expressions or as arguments to many different functions.
Symbolic Calculations - Simplify a function: simple(f)

Simplify \( f = \cos(x)^2 + \sin(x)^2 \)

\[
\gg f = \cos(x)^2 + \sin(x)^2 \\
\gg f = \text{simple}(f) \\
\gg \text{simple}(f)
\]

simplify:
1
radsimp:
\( \sin(x)^2 + \cos(x)^2 \)
combine(trig):
1
factor: \( \sin(x)^2 + \cos(x)^2 \)
expand: \( \sin(x)^2 + \cos(x)^2 \)
convert(exp): \(-1/4*(\exp(i*x)-1/\exp(i*x))^2+(1/2*\exp(i*x)+1/2/\exp(i*x))^2\)
convert(sincos): \( \sin(x)^2 + \cos(x)^2 \)
convert(tan):
\[
4*\tan(1/2*x)^2/(1+\tan(1/2*x)^2)^2+(1-\tan(1/2*x)^2)^2/(1+\tan(1/2*x)^2)^2 \\
\]
collect(x):
\( \sin(x)^2 + \cos(x)^2 \)
ans =
1
\[
\gg
\]
Symbolic Calculations – diff - Differentiate

Steps to execute a symbolic calculation

1. Use command symbols to declare the variables necessary to perform a symbolic calculation
   >> syms x y
   % or we can use x = sym('x'); y = sym('y');

2. Use a MatLab symbolic command
   >> f 1 = sin(x*y)*cos(2*y)
   f 1 = sin(x*y)*cos(2*y)
   >> diff(f1)
   >> % differentiate with respect to symbol x
   ans =
   cos(x*y)*y*cos(2*y)
   >> pretty (ans)
   \[ cos(x \cdot y) \cdot y \cdot cos(2 \cdot y) \]
Symbolic Calculations – diff - Differentiate

```matlab
% or we can use x = sym('x'); y = sym('y);
>> f1 = sin(x*y)*cos(2*y)
f1 = sin(x*y)*cos(2*y)

% differentiate with respect to symbol x
>> diff(f1)
ans =
cos(x*y)*y*cos(2*y)

% pretty print
>> pretty(ans)
cos(x * y) y cos(2 * y)
```

```matlab
% or we can use x = sym('x'); y = sym('y);
>> f2 = sin(x*y)*cos(2*y)
f2 = sin(x*y)*cos(2*y)

% differentiate with respect to symbol y
>> diff(f2,y)
ans = cos(x*y)*x*cos(2*y) - 2*sin(x*y)*sin(2*y)

% pretty print
>> pretty(ans)
cos(x * y) * cos(2 * y) - 2 sin(x * y) sin(2 * y)
```
Symbolic Calculations — int - Integrate

Perform the following integrals symbolically, and for the indefinite integrals

\[ \int_{0}^{\pi/2} \cos x \sin x \, dx \]
\[ \text{>> int(cos(x)*sin(x),x, 0,pi/2)} \]
\[ \text{ans} = \frac{1}{2} \]

\[ \int \cos x \sin x \, dx \]
\[ \text{>> int(cos(x)*sin(x),x)} \]
\[ \text{ans} = \frac{1}{2} \sin(x)^2 \]

\[ \int_{-\infty}^{\infty} 3e^{-x^2} \, dx \]
\[ \text{>> f=3*exp(-x^2);} \]
\[ \text{>> int(f,-lnf,lnf)} \]
\[ \text{ans} = 3\pi^{1/2} \]
Symbolic Calculations - Summation

Compute the following sums:

$$\sum_{k=1}^{n} k^3$$

```matlab
>> syms k n
>> symsum(k^3,k,1,n)
ans = 1/4*(n+1)^4 - 1/2*(n+1)^3 + 1/4*(n+1)^2
```

```matlab
>> pretty(ans)
\[
\frac{1}{4} (n + 1)^4 - \frac{1}{2} (n + 1)^3 + \frac{1}{4} (n + 1)^2
\]
```

```matlab
>> simplify(ans)
ans = 1/4*n^4 + 1/2*n^3 + 1/4*n^2
```
Symbolic Calculations – MatLab Symbolic functions

```plaintext
>> help symbolic

Calculus
  diff   - Differentiate.
  int    - Integrate.
  limit  - Limit.
  taylor - Taylor series.
  jacobian - Jacobian matrix.
  symsum - Summation of series.

Linear Algebra.
  diag    - Create or extract diagonals.
  triu    - Upper triangle.
  tril    - Lower triangle.
  inv     - Matrix inverse.
  det     - Determinant.
  rank    - Rank.
  rref    - Reduced row echelon form.
  null    - Basis for null space.
  colspace - Basis for column space.
  eig     - Eigenvalues and eigenvectors.
  svd     - Singular values and singular vectors.
  jordan  - Jordan canonical (normal) form.
  poly    - Characteristic polynomial.
  expm    - Matrix exponential.
```
MatLab Programming

MatLab is a programming language in itself; it includes programming language constructs such as loops, branches and writing of functions or scripts

- M-File
- Scripts
- MatLab Functions
- Function Handle
M-files

*M-File can be either scripts or functions*

*Scripts are simply files containing a sequence of MatLab statements/commands*

*Functions make use of their own local variables and accept input arguments*
MATLAB Script

Steps in writing and running a MatLab script program:

Use an editor; to create a file (from the menu) File > New > m-file

Type in the MatLab command sequences

Save the file (e.g. dynamicSys1.m); make sure the file path is accessible to MatLab

Run the file (e.g. dynamicSys1.m); in Command Window, type the file name (e.g. dynamicSys1) M-File name

View the file (e.g. dynamicSys1.m); type LS in Command Window

Note: The name of an M-file begins with an alphabetic character and has a filename extension of .m.
Import/export data files

% Simple program script – Example 1
clear all % clear all variables from memory

% S=load(filename) load data from the specified file into a matrix
% load filename - it uses the file name as the matrix variable name
% 
% load student.dat reads all data from the student.dat file into S matrix variable
S=load('student.dat');
height = S(:,1);
weight = S(:,2);
% Calculate Mean and standard deviations
average_weight = mean(weight);
std_weight = std(weight);
maximum_weight = max(weight);
minimum_weight = min(weight);
disp(sprintf('Average Weight: %6.2f', average_weight));
disp(sprintf('Standard Deviation: %6.2f', std_weight));
disp(sprintf('Maximum Weight: %6.2f', maximum_weight));
disp(sprintf('Minimum Weight: %6.2f', minimum_weight));
disp(sprintf('n
'));
%divide figure in one row, two columns, then select pane 1
subplot(1,2,1); % subplot: row=1, column=2, pane=1
% plot (linear 2-D plot) of weight data
plot(weight)
title('Weight of Freshman Class Men')
xlabel('Count(students)')
ylabel('Weight, lbs')
grid on
hold on
\begin{verbatim}
% plot average line
plot(1:size(weight),average_weight,'r:');
% subplot: row=1, column=2, pane=2
subplot(1,2,2)
plot(height)
title('Height of Freshman Class Men')
xlabel('Count(students)')
ylabel('Height, inches')
average_height = mean(height);
std_height = std(height);
maximum_height = max(height);
minimum_height = min(height);
disp(sprintf('\nAverage Height: %6.2f', average_height));
disp(sprintf('\nStandard Deviation: %6.2f', std_weight));
disp(sprintf('\nMaximum Height: %6.2f', maximum_height));
disp(sprintf('\nMinimum Height: %6.2f', minimum_height));
disp(sprintf('\nDone'));
\end{verbatim}
Figure No. 1

Weight of Freshman Class Men

Height of Freshman Class Men

Count (students)

Weight, lbs

Height, inches

Count (students)
% Simple program script – Example 2
% clear removes all variables from the workspace; to free up system
% memory.
clear all

% clc clears all input and output from the Command Window display,
% giving you a "clean screen."
clc

% The linspace function generates linearly spaced vectors. It is
% similar to the colon operator ":", but gives direct control over
% the number of points.
% y = linspace(a,b,n) generates a row vector y of n points linearly
% spaced between and including a and b.
% 
% x=linspace(0,10*pi,1000);
% The Colon Operator (:) is used to create a vector containing:
% -2.0000 -1.8000 -1.6000 ..... 1.6000 1.8000 2.0000
% with 21 cells
x=[-2:0.2:2];
y=[-2:0.2:2];

% meshgrid - Generates X and Y matrices for three-dimensional plots
% [X,Y] = meshgrid(x,y) transforms the domain specified by vectors x
% and y into arrays X and Y, which can be used to evaluate functions
% of two variables and three-dimensional mesh/surface plots. The rows
% of the output array X are copies of the vector x; columns of the
% output array Y are copies of the vector y.
%
% For example, the [X,Y] = meshgrid(1:3,10:14) are:
%
Sixth International Conference on Dynamic Systems and Applications!

\% \textbf{X} = \\
\% \\
\% \textcolor{white}{1} 2 3 \\
\% \textcolor{white}{1} 2 3 \\
\% \textcolor{white}{1} 2 3 \\
\% \textcolor{white}{1} 2 3 \\
\% \textcolor{white}{1} 2 3 \\
\% \textcolor{white}{1} 2 3 \\
\%

\% \textbf{Y} = \\
\% \\
\% \textcolor{white}{10} 10 10 \\
\% \textcolor{white}{10} 10 10 \\
\% \textcolor{white}{11} 11 11 \\
\% \textcolor{white}{11} 11 11 \\
\% \textcolor{white}{12} 12 12 \\
\% \textcolor{white}{12} 12 12 \\
\% \textcolor{white}{13} 13 13 \\
\% \textcolor{white}{13} 13 13 \\
\% \textcolor{white}{14} 14 14
\[ [X,Y] = \text{meshgrid}(x,y); \]

\%

* operator is used to perform an element by element multiplication

\[ Z = X \ast \exp(-X \ast \text{power}2 - Y \ast \text{power}2); \]

\textit{subplot}(2,2,1)

\%

mesh(X,Y,Z) draws a wireframe mesh with color determined by Z so color
%

is proportional to surface height.

\textit{mesh}(X,Y,Z)

\textit{title}('Mesh Plot')

\textit{xlabel}('x-axis')

\textit{ylabel}('y-axis')

\textit{zlabel}('z-axis')
subplot(2,2,2)
% surf(X,Y,Z) creates a shaded surface using Z for the color data as well
% as surface height.
surf(X,Y,Z)
title('Surface Plot')
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')

subplot(2,2,3)
% contour(X,Y,Z) draws contour plot of Z
contour(X,Y,Z)
title('Contour Plot')
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
subplot(2,2,4)
% surfc(...) draws a contour plot beneath the surface.
surfc(X,Y,Z)
title('Combination Surface and Contour Plot')
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
M-file function

Syntax

function [out1, out2, ...] = funname(in1, in2, ...)

function [out1, out2, ...] = funname(in1, in2, ...) defines function funname that accepts inputs in1, in2, etc. and returns outputs out1, out2, etc.
Multiple functions within an M-File

MatLab allows multiple subfunctions within a M-File

These subfunctions are not visible to other functions in the other M-Files

A function is terminated using an end statement or use a return statement to force an early return

Local variables

The variables within the body of the function are all local variables
When MATLAB does not recognize a function by name, it searches for a file of the same name on disk. If the function is found, MATLAB compiles it into memory for subsequent use.

When you call an M-file function from the command line or from within another M-file, MATLAB parses the function and stores it in memory. The parsed function remains in memory until cleared with the clear command or you quit MATLAB.
Example
M-File Name: stat.m
Input: vector x
Output: mean and stdev

function [mean, stdev] = stat(x)
    n = length(x);
    mean = sum(x)/n;
    stdev = sqrt(sum((x-mean).^2/n));
end
function_handle (@)

Function handle is used to call functions indirectly

Syntax
handle = @functionname

handle = @functionname returns a handle to the specified MATLAB function
Function handle

A function handle is a MATLAB value that provides a means of calling a function indirectly.
M-Files

rosenbrock.m function

test.m script

minimize.m function

Test calls minimize function, function Handle to rosenbrock function is passed as parameter to minimize function
% Partial example

function y = rosenbrock()
    % Rosenbrock banana function
    % The minimum is at (1,1), (y=8.1777e-010).
    % The traditional starting point is (-1.2,1).

    y = 100*(x(2)-x(1)^2)^2+(1-x(1))^2;

end
% MatLab Script
% File name: Test
x=[-1.2 1];
[x, y, history] = minimize(@rosenbrock, x);
x
y
history
m1=min(history(:,1));
m2=max(history(:,1));
m3=min(history(:,2));
m4=max(history(:,2));
m1, m2, m3, m4

.......
% File Name: minimize.m

function [x, y, history] = minimize(functionHandle, x_init)
    history = [];
    options = optimset('OutputFcn', @myoutput);
    [x y] = fminsearch(functionHandle, x_init, options);
    function stop = myoutput(x, optimvalues, state);
        stop = [];
        if state == 'iter'
            history = [history; x];
        end
    end
end
end
% MatLab Script – Example

x=[-2:0.2:2];
y=[-2:0.2:2];
[X,Y]=meshgrid(x,y);
Z =100*(Y-X.^2).^2+(1-X).^2;
subplot(2,2,1)

mesh(X,Y,Z) draws a wireframe mesh with color determined by Z so color
% is proportional to surface height.

mesh(X,Y,Z)
title('Mesh Plot')
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
subplot(2,2,2)
% MatLab Script
% surf(X,Y,Z) creates a shaded surface using Z for the color data as well
% as surface height.
surf(X,Y,Z)
title('Surface Plot')
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')

subplot(2,2,3)
% contour(X,Y,Z) draws contour plot of Z
contour(X,Y,Z)
title('Contour Plot')
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
% MatLab Script
% File name: minimizeTest

subplot(2,2,4)
% surfc(...) draws a contour plot beneath the surface.
surfc(X,Y,Z)
title('Combination Surface and Contour Plot')
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')

close all; % close all previously drawn graphs
figure; % create a new figure/plot
grid on;
title('History points')
xlabel('x(1)');
ylabel('x(2)');
plot(history(:,1),history(:,2), '--rs', 'LineWidth',2, 'MarkerEdgeColor', 'k', 'MarkerFaceColor','g', 'MarkerSize',2)
The first 300,000 digits of pi

Pi with million digits:

\[ \pi = 3.14159265358979323846264338327950288419716939937510582097494459223078164062862089896280348253421170679
821480865133282306647093984469059550852231725359408128481117450702890107200881838511550237528963841
428819756559334461284756642376878316572011920194568456992346603486104543266682123936307620249414729
72548700606315588174881520920662829524091715364367892590360011330535058840665218341469519415116094
33057720165675959159503286171138236619701051158470349677926597351885757272489127293818301194912
983367336244066643086021234946935924273719702179860943707270539211716293176752386478418647669405132
00056812714526356080277857134275778960917367317872114648409101224953430146549585371050792279692589235
420199561121290219608640344118519831629747713109960518702113499999832797804995105973173281690631859
5024459455346908302642232052344865053526193118817101000313783758528865783320388342061717769147303
5982534904287954687311595628638823573875937195778185778035217122668613001922768671119590901164201989
38095257201065485863278865936153381872968230301952033018529689577362259941389124972177528347913151
55748572424514056959508295331168617278558890750988318756437464939319250560049927071671173009688420412
858361306536730760610471018194295559619894667837449448255379774726847110407543564620804066842590696912
9331367702898991520475162015966602405803815019325125338243003558764024794647326939149192726024692279
6782354618670939314721641211992485631503028619254775570674983850549458858692699569092710797590320955
3211653449872027559602364806654991198818347977535663698074265452578625518184175476728909777727938000
81647606010145249192172317214772350141441917356841613615735752533123475718498643852323290739414333
45477624168625189835684985562099219222184272550254256887671790494601653466804988627232791788658578438
8279679766814510995388378636095060806422512505117392848986081428488626945604219652850222106611863
0674427895899149450471237187669563643791917276877466575739624138908683264595813390478025795009
94657640789512694838935259597098258226052248904772671974287682486260147669909264013693447345530068203
4965254712459396514321490819065290972216964151507498387410597885959772957498930161573928461832
686838689427471559104885592524595399541049972524680845987723646958486538363763226262099124608501524388
4390451244136549767280797715691435977001296160894164968655854840635342022722582848864861584628056
01684273992124022358254954660772823986565961636548862305754648908355936345661743241125
150786069479451065965094025228879710893145669136687287284905601015303086179286092087476091782493858
900914190679589261361954798193127984821692898847252685084575640142704775515312379641451523476243364
5428584447395265867821051143547375395231134721661021359653623144295248493871781101457654035902799344
0374200731057853906219838747487804874896833214457138687519453506430218453190484100537061648067491927
% Utility program to group digits of Pi
% (group of 1, 10 and 15 digits) –
% Use the first 300000 digits of Pi
fid = fopen('PI.txt');
a = fscanf(fid,'%s'); % It has two rows now.

b=[];
for d=1:300000
    if(a(1,d)>='0' && a(1,d)<='9')
        a(1,d);
        b=[b,a(1,d)];
    end
end

size(b);
k=10; l=1; m=15;
fid1 = fopen('PI10Out-150000.txt','w');
fid2 = fopen('PI15Out-150000.txt','w');
fid3 = fopen('PI1Out-150000.txt','w');

% group of 15, 5 groups per row
for d=1:20000
    ac1=b(l:m); l=l+15; m=m+15;
    ac2=b(l:m); l=l+15; m=m+15;
    ac3=b(l:m); l=l+15; m=m+15;
    ac4=b(l:m); l=l+15; m=m+15;
    ac5=b(l:m); l=l+15; m=m+15;
    fprintf(fid2,'%15s %15s %15s %15s ... %15s\n',ac1,ac2,ac3,ac4,ac5);
end
% group of 10, 5 groups per row
for d=1:30000
    ab1=b(j:k); j=j+10; k=k+10;
    ab2=b(j:k); j=j+10; k=k+10;
    ab3=b(j:k); j=j+10; k=k+10;
    ab4=b(j:k); j=j+10; k=k+10;
    ab5=b(j:k); j=j+10; k=k+10;
    fprintf(fid1,'%10s %10s %10s %10s %10s
',ab1, ab2, ab3,ab4,ab5);
End

% group of 1, 75 groups per row
j=1; k=1;
for d=1:2000
    b1=[ ];
    for b2=1:75
        b1=[b1,b(j:k)];
        b1=[b1,' '];
        j=j+1; k=k+1;
    end
    fprintf(fid3,'%s
',b1);
End
fclose(fid); fclose(fid1); fclose(fid2); fclose(fid3);
Sixth International Conference on Dynamic Systems and Applications!

% The first 300,000 digits of pi

Pi with million digits:

\[ \pi = 3.1415926535897932384626433832795028841971693993751058209749445923078164062862089888628034832453423891747663384941717373817107056055786775432922393906877517786054774845566584098849699506600672779647903428755428531903295766781609433680597748214292680703045727330355286813418458388699690064720587560057494333000762878736975939972168460247988190895551278083179340825343728173007988945287248498608720065826365852388850757187275171636299562271666389012565100708074194573273383875208750947954104633312396908540084748364909353105839867881748847841837943975507248458588002643773251173452742285035565904560364482813774209548660223081608632374078033967728018527801324820920369899001963475339586440245935637474761051008399294040643123345121627207445901524800501421231117319243899200011260044375598243607903243337588319449508441510816003280458817165410845038141858939989691612785317829349887956415671080702300162795223247821109383133428274251411272955160601619016989618776804995859571427565562063441341006513875329627829315331233687838362944168983590100180065052759231861857898787071410069701626999784589660671139067550181079082902033365002829221297308859496492949120786759940850341628531314634021861231660811411359413066297537597091751878899624092098281876100150777475427423982810339380487731906959002799707664534772708153044418946229959636388811223212926985908013104252315846137529357290332301137274881970587919496234296601177765148256458676232277736376060479535019320090230927485369192244048289539487589896830352106784719652500524175943928181323296423207804814789298620538528186265472014538476766857866789494634852463526382931133491444286828965398537088154358897823527923009611300006664055237302494906769279003420333598127167159386088201144783837837984344849187634220677807823491472828376213626712845356034394914582828330599511840363362225690777050019772641371587451928939377249885629092094991620157806036241146710042524283345207836801605671266504713813324262123203006177562741450820969360809289900309280607772129586360960605798717774682367030910288736898888751769780959822217283053101616949713913620803899724120390886227776989655618812476455953124077187783101473610176791795773868224163849423919212161022584742057341714350777123478345982498660682907503661181115458744529667580239630590504005683885469725229141651148997488186927253194454675474319719405154079778751710302454972950579480259283102281711525412577129084794356102964096044002387676114612705836909680945846302335631006288241135414511107370929476751265007053075264359965869427934580291464221638137761794972105906525681751644616732671047059760061062227592667062564214337685060836312088160590618406839448006945980147539186753397946769280812529121813067277852337607315234134737762143898536170297502146428190712042881559445020807718785314580700118051100942192477472817163019221707749578392424...
% Utility Program to count digits 0 – 9 in the first 300000 digits of Pi
%
fid = fopen('Pi.txt');
b = fscanf(fid,'%s'); % It has two rows now.

d1=0; d2=0; d3=0; d4=0; d5=0; d6=0; d7=0; d8=0; d9=0; d0=0;

a=[];

for d=1:300000
    if(b(1,d)>'0' && b(1,d)<'9')
        a=[a,b(1,d)];
    end
end
size(b)
% Change 300000 to a desire count
%
% Program utility to count 0 – 9 in the first 300000 digits of Pi
% inside 'file name'

fid = fopen('Pi.txt');
b = fscanf(fid,'%s'); % It has two rows now.

d1=0; d2=0; d3=0; d4=0; d5=0; d6=0; d7=0; d8=0; d9=0; d0=0;

a=[];

for d=1:300000
    if(b(1,d)>'0' && b(1,d)<'9')
        a=[a,b(1,d)];
    end
end

size(b)
% Change 300000 to a desire count
%
for d=1:300000
    if a(1,d) == '1'
        d1=d1+1;
    elseif a(1,d) == '2'
        d2=d2+1;
    elseif a(1,d) == '3'
        d3=d3+1;
    elseif a(1,d) == '4'
        d4=d4+1;
    elseif a(1,d) == '5'
        d5=d5+1;
    elseif a(1,d) == '6'
        d6=d6+1;
    elseif a(1,d) == '7'
        d7=d7+1;
    elseif a(1,d) == '8'
        d8=d8+1;
    elseif a(1,d) == '9'
        d9=d9+1;
    else
        a(1,d)
    end
end
fclose(fid);
% show counts of digits 0-9
d0, d1, d2, d3, d4, d5, d6, d7, d8, d9
Thank you!