Tutorial 1[§]:

Introduction to MATLAB

Daniela Raicu
draicu@cs.depaul.edu
School of Computer Science, Telecommunications, and Information Systems
DePaul University, Chicago, IL 60604

The purpose of this tutorial is to present basics of MATLAB. We do not assume any prior knowledge of this package; this tutorial is intended for users running a professional version of MATLAB 6.5, Release 13.

Topics discussed in this tutorial include:

1. Introduction to MATLAB and its toolboxes
2. Development environment
3. MATLAB help and basics
4. MATLAB Toolboxes demos
5. MATLAB Programming
6. MATLAB resources on the Internet
7. MATLAB toolboxes descriptions

1.0 MATLAB and its toolboxes

MATLAB (Matrix Algebra laboratory), distributed by The MathWorks, is a technical computing environment for high performance numeric computation and visualization. It integrates numerical analysis, matrix computation, signal processing, and graphics in an easy-to-use environment.

MATLAB also features a family of application-specific solutions called toolboxes. Toolboxes are comprehensive collections of MATLAB functions that extend its environment in order to solve particular classes of problems. The table below includes the toolboxes that are available in the last version of MATLAB:

Communications	**Image Processing**	**System Identification**
Control System	**Instrument Control**	**Wavelet**
Data Acquisition	**Mapping**	**MATLAB Compiler**
Database	**Neural Network**	**MATLAB C/C++ Graphics Library**
Datafeed	**Optimization**	**MATLAB C/C++ Math Library**
Filter Design	**Partial Differential Equation**	**MATLAB Report Generator**
Financial	**Robust Control**	**MATLAB Runtime Server**
Frequency Domain System Identification	**Signal Processing**	**MATLAB Web Server**
Fuzzy Logic	**Statistics**	**Simulink**
Higher-Order Spectral Analysis	**Spline**	**Symbolic/Extended Math**

[§] Event Sponsor: Visual Computing Area Curriculum Quality of Instruction Council (QIC) grant

2.0 Development Environment: Command Window

You can start MATLAB by double clicking on the MATLAB icon that should be on the desktop of your computer. This brings up the window called the Command Window. This window allows a user to enter simple commands. To perform a simple computations type a command and next press the Enter or Return key. For instance,

```
>> s = 1 + 2
s =
3
```

Note that the results of these computations are saved in variables whose names are chosen by the user. If they will be needed during your current MATLAB session, then you can obtain their values typing their names and pressing the Enter or Return key.

For instance,

```
>> s
s =
3
```

Variable name begins with a letter, followed by letters, numbers or underscores. MATLAB recognizes only the first 31 characters of a variable name.

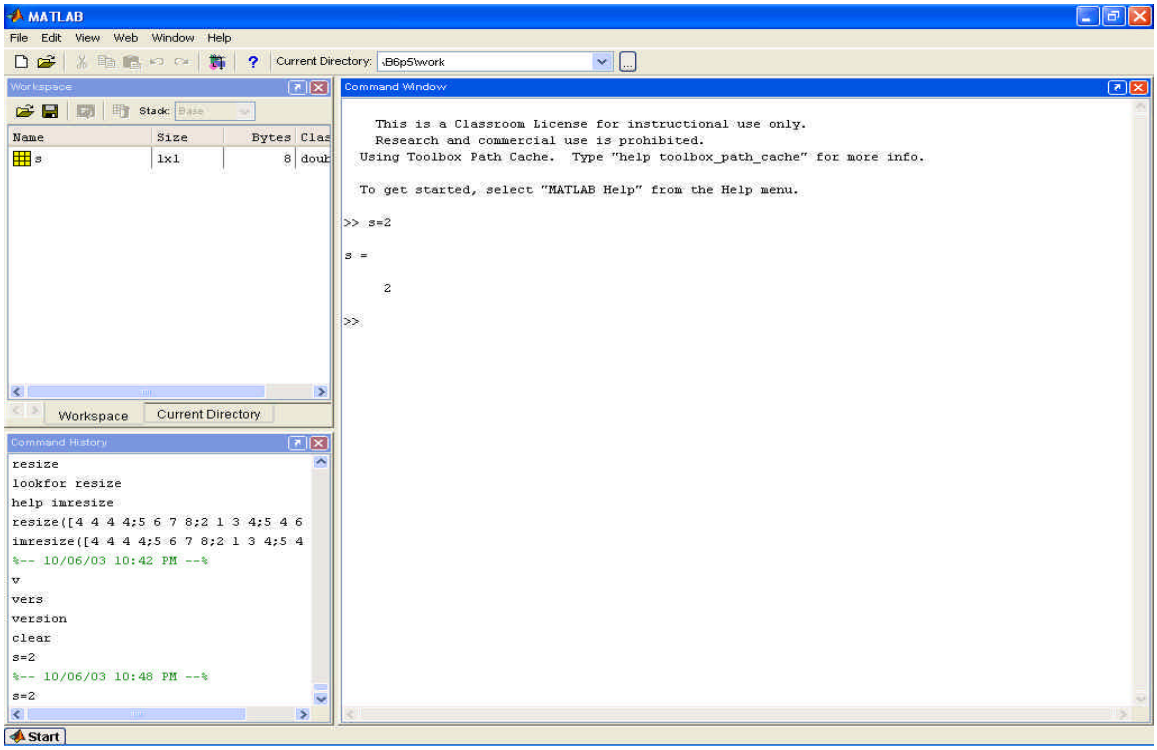

Figure 1: A screenshot of the development environment, command windows

For a demo for the MATLAB Desktop, you can visit http://www.mathworks.com/products/demos/# (there is also a demo **for the Command History and Workspace Browser)**.

To close MATLAB type exit in the Command Window and next press Enter or Return key. A second way to close your current MATLAB session is to select File in the MATLAB's toolbar and next click on Exit MATLAB option. All unsaved information residing in the MATLAB Workspace will be lost.

3.0 MATLAB Help and Basics

To get help type “help” (will give you a list of help topics) or “help topic”.

If you don't know the exact name of the topic or command you are looking for, type "lookfor keyword" (e.g., "lookfor regression")

When writing a long MATLAB statement that exceeds a single row use “...” to continue statement to next row.

When using the command line, a ";" at the end means MATLAB will not display the result. If ";" is omitted then MATLAB will display result.

Use the up-arrow to recall commands without retyping them (and down arrow to go forward in commands).

The symbol "%" is used in front of a comment.

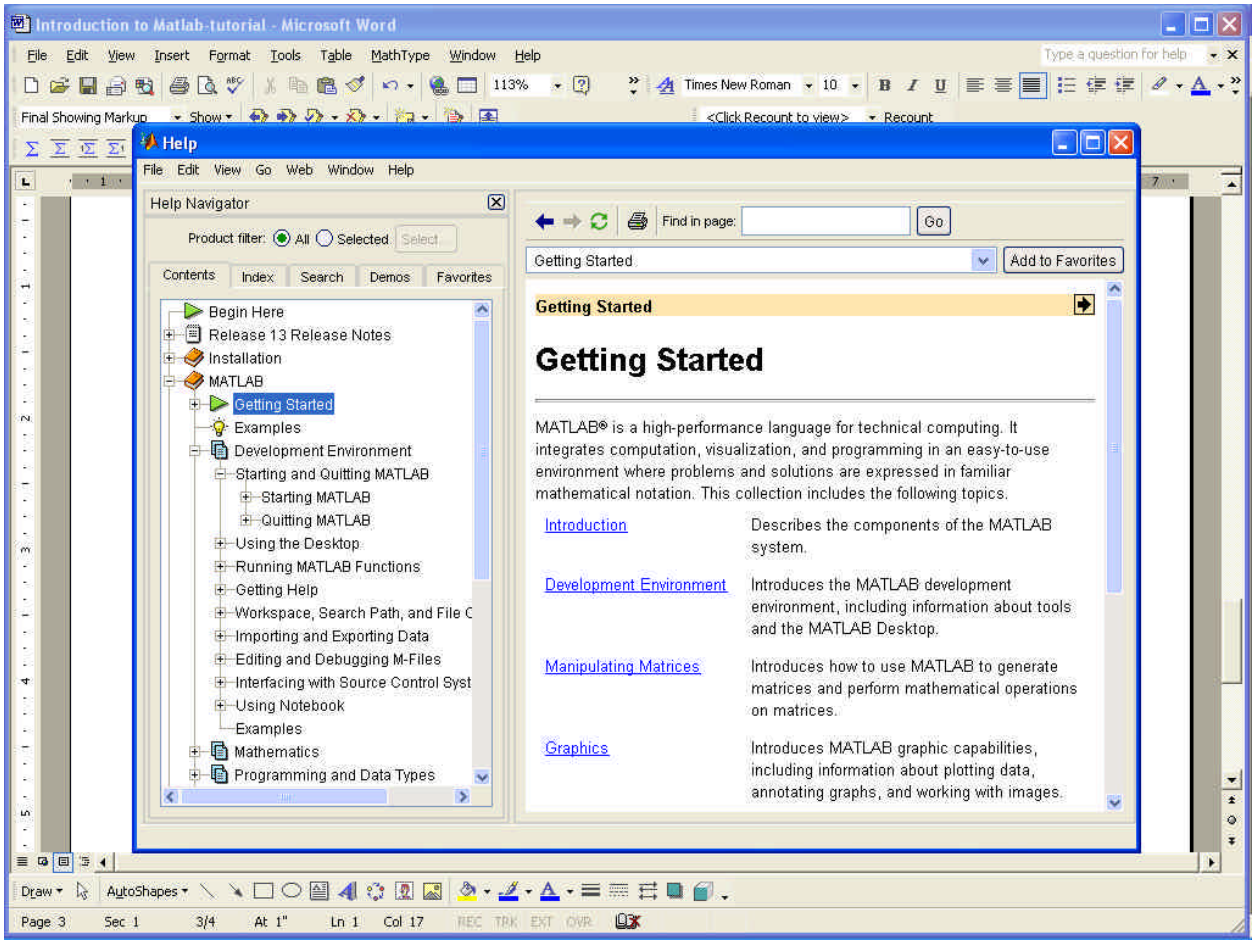

Figure 2: A screenshot of Help

4.0 MATLAB Toolboxes Demos

To learn more about MATLAB capabilities you can execute the demo command in the Command Window or click on Help and next select **Demos** from the pull-down menu. Some of the MATLAB demos use both the Command and the Figure windows.

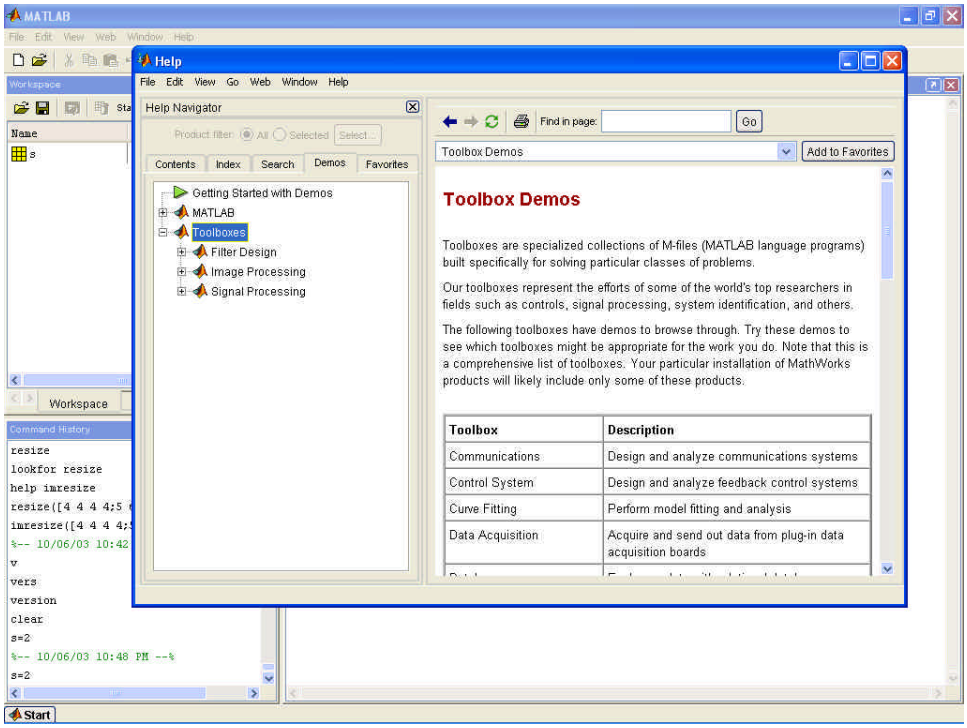

Figure 3: A screenshot of Demos' Help

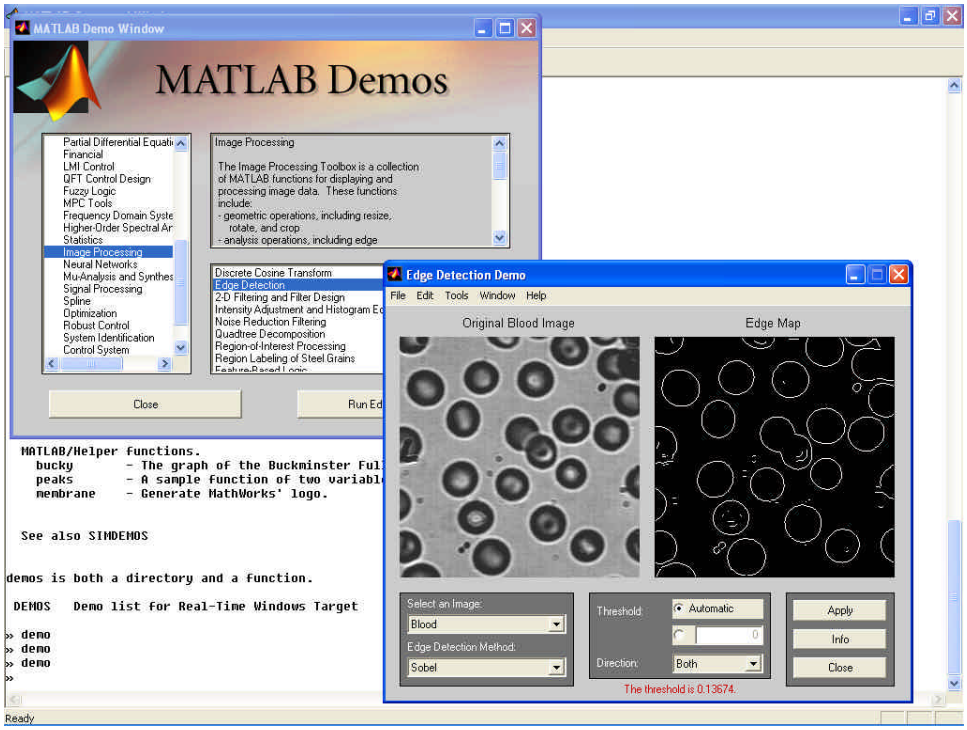

Figure 4: A screenshot of the image processing demo

5.0 MATLAB Programming

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
% Introducing MATLAB (adapted from http://www.cns.nyu.edu/~eero and
% http://www.cs.dartmouth.edu/~farid/teaching/cs88/MATLAB.intro.html)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
```

% (**1) Objects in MATLAB -- the basic objects in MATLAB are scalars, vectors, and matrices...**

```
N       = 5                 % a scalar
v       = [1 0 0]           % a row vector
v       = [1;2;3]           % a column vector
v       = v'                % transpose a vector
v       = [1:.5:3]          % a vector in a specified range:
v       = pi*[-4:4]/4       % [start: stepsize: end]
v       = []                % empty vector

m       = [1 2 3; 4 5 6]    % a matrix: 1ST parameter is ROWS
                            % 2ND parameter is COLS
m       = zeros(2,3)        % a matrix of zeros
v       = ones(1,3)         % a matrix of ones
m       = eye(3)            % identity matrix
v       = rand(3,1)         % rand matrix (see also randn)

v       = [1 2 3];          % access a vector element
v(3)                        %vector(number)

m       = [1 2 3; 4 5 6]
m(1,3)                      % access a matrix element
                            % matrix(rownumber, columnnumber)

m(2,:)                      % access a matrix row (2nd row)
m(:,1)                      % access a matrix column (1st row)

size(m)                     % size of a matrix
size(m,1)                   % number rows
size(m,2)                   % number of columns

m1      = zeros(size(m))    % create a new matrix with size of m

who                         % list of variables
whos                        % list/size/type of variables

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
```

% **(2) Simple operations on vectors and matrices**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% (A) Pointwise (element by element) Operations:

% addition of vectors/matrices and multiplication by a scalar
% are done "element by element"
a= [1 2 3 4];               % vector
2 * a                       % scalar multiplication
a / 4                       % scalar multiplication
```

```
b        = [5 6 7 8];          % vector
a + b                          % pointwise vector addition
a - b                          % pointwise vector addition
a .^ 2                         % pointise vector squaring (note .)
a .* b                         % pointwise vector multiply (note .)
a ./ b                         % pointwise vector multiply (note .)

log( [1 2 3 4] )               % pointwise arithmetic operation
round( [1.5 2; 2.2 3.1] )      % pointwise arithmetic operation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% (B) Vector Operations (no for loops needed)
% Built-in MATLAB functions operate on vectors, if a matrix is given,
% then the function operates on each column of the matrix

a        = [1 4 6 3]           % vector
sum(a)                         % sum of vector elements
mean(a)                        % mean of vector elements
var(a)                         % variance
std(a)                         % standard deviation
max(a)                         % maximum

a        = [1 2 3; 4 5 6]      % matrix
mean(a)                        % mean of each column
max(a)                         % max of each column
max(max(a))                    % to obtain max of matrix
max(a(:))                      %         or...

%%%%%%%%%%%%%%%%%%%%%%%%
% (C) Matrix Operations:

[1 2 3] * [4 5 6]'             % row vector 1x3 times column vector 3x1
                               % results in single number, also
                               % known as dot product or inner product

[1 2 3]' * [4 5 6]             % column vector 3x1 times row vector 1x3
                               % results in 3x3 matrix, also
                               % known as outer product

a        = rand(3,2)           % 3x2 matrix
b        = rand(2,4)           % 2x4 matrix
c        = a * b               % 3x4 matrix

a        = [1 2; 3 4; 5 6]     % 3 x 2 matrix
b        = [5 6 7];            % 3 x 1 vector
b * a                          % matrix multiply
a' * b'                        % matrix multiply
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

%**(3) Saving your work**

```
save mysession                 % creates session.mat with all variables
save mysession a b             % save only variables a and b

clear all                      % clear all variables
clear a b                      % clear variables a and b
```

```
load mysession                                  % load session
a
b
```

%%
% (4) Relations and control statements

% Example: given a vector v, create a new vector with values equal to v if they are greater than 0, and equal to 0 if they less than or equal to 0.

```
v       = [3 5 -2 5 -1 0]           % 1: FOR LOOPS
u       = zeros( size(v) );         % initialize
for i = 1:size(v,2)
        if( v(i) > 0 )
                u(i) = v(i);
        end
end
u

v       = [3 5 -2 5 -1 0]           % 2: NO FOR LOOPS
u2      = zeros( size(v) );         % initialize
ind     = find( v>0 )                       % index into >0 elements
u2(ind) = v( ind )
```

%%
%**(5) Creating functions using m-files:**
% Functions in MATLAB are written in m-files. Create a file called 'thres.m' In this file put the following:

```
function res = thres( v )

u       = zeros( size(v) );         % initialize
ind     = find( v>0 )                       % index into >0 elements
u(ind)  = v( ind )

v       = [3 5 -2 5 -1 0]
thres( v )                          % call from command line
```

%%
%**(6) Plotting**

```
x       = [0 1 2 3 4];                      % basic plotting
plot( x );
plot( x, 2*x );
axis( [0 8 0 8] );

x       = pi*[-24:24]/24;
plot( x, sin(x) );
xlabel( 'radians' );
ylabel( 'sin value' );
title( 'dummy' );
gtext( 'put cursor where you want text and press mouse' );
```

```
figure;                                    % multiple functions in separate graphs
subplot( 1,2,1 );
plot( x, sin(x) );
axis square;
subplot( 1,2,2 );
plot( x, 2.*cos(x) );
axis square;

figure;                                    % multiple functions in single graph
plot( x,sin(x) );
hold on;
plot (x, 2.*cos(x), '--' );
legend( 'sin', 'cos' );
hold off;

figure;                                    % matrices as images
m       = rand(64,64);
imagesc(m)
colormap gray;
axis image
axis off;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
%(7) Working with Images

[I,map]=imread('trees.tif');      % read a TIFF image

figure, imshow(I,map)             % display it as indexed image

I2=ind2gray(I,map);               % convert it to grayscale

figure
imagesc(I2,[0 1])                 % scale data to use full colormap
                              %  for values between 0 and 1
colormap('gray')                  % use gray colormap
axis('image')                     % make displayed aspect ratio %proportional
                              %  to image dimensions

I=imread('photo.jpg');            % read a JPEG image into 3D %array

figure
imshow(I)
rect=getrect;                     % select rectangle
I2=imcrop(I,rect);                % crop
I2=rgb2gray(I2);                  % convert cropped image to grayscale
imagesc(I2)                       % scale data to use full colormap
                              %  between min and max values in I2
colormap('gray')
colorbar                          % turn on color bar
pixval                            % display pixel values interactively
truesize                          % display at resolution of one %screen pixel
                              %  per image pixel
truesize(2*size(I2))              % display at resolution of two %screen pixels
                              %  per image pixel
```

```
I3=imresize(I2,0.5,'bil');            % resize by 50% using bilinear
                                      % interpolation
I3=imrotate(I2,45,'bil','same');      % rotate 45 degrees and crop to
                                      % original size
I3=double(I2);                        % convert from uint8 to double, to %allow
                                      % math operations
imagesc(I3.^2)                        % display squared image (pixel-wise)
imagesc(log(I3))                      % display log of image

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

6.0 MATLAB Resources on the Internet

MATLAB demos: http://www.mathworks.com/products/demos/#

MATLAB tutorials: http://www.math.siu.edu/MATLAB/tutorials.html

MATLAB Primer: http://math.ucsd.edu/~driver/21d-s99/MATLAB-primer.html

Tutorial code snippets: http://www-cse.ucsd.edu/~sjb/classes/MATLAB/MATLAB.intro.html

MATLAB FAQ: http://www.mit.edu/~pwb/cssm/

MathWorks, INC.: http://www.mathworks.com

7.0 MATLAB Toolboxes Descriptions

Communications Toolbox provides a comprehensive set of tools for the design, analysis, and simulation of digital and analog communication systems. The toolbox contains an extensive collection of MATLAB/Simulink blocks for developing and simulating algorithms and system designs in applications such as wireless devices, modems and storage systems. It also serves as an excellent basis for research and education in communications engineering.

Control System Toolbox - is a collection of MATLAB functions for modeling, analyzing, and designing automatic control systems. The functions in this Toolbox implement mainstream classical and modern control techniques. With the Control System Toolbox, you can analyze and simulate both continuous-time and discrete-time linear dynamic systems. The graphical user interfaces allow you to quickly compute and graph time responses, frequency responses, and root-locus diagrams.

Data Acquisition Toolbox provides a complete set of tools for controlling and communicating with a variety of off-the-shelf, PC-compatible data acquisition hardware. The toolbox lets you configure your external hardware devices, read data into MATLAB for analysis, or send data out.

Database Toolbox allows you to connect to and interact with most ODBC/JDBC databases from within MATLAB. The Database Toolbox allows you to use the powerful data analysis and visualization tools of MATLAB for sophisticated analysis of data stored in databases. From within the MATLAB environment, you can use Structured Query Language (SQL) commands to: Read and write data to and from a database; Apply simple and advanced conditions to your database queries.

Datafeed Toolbox integrates the numerical, computational, and graphical capabilities of MATLAB® with financial data providers. Datafeed Toolbox provides a direct connection between MATLAB and data provided by the Bloomberg data service. Once the data is in MATLAB, it can then be analyzed using other tools in the MATLAB product family such as GARCH, Statistics, Financial Time Series, and Neural Networks.

Filter Design Toolbox - A collection of tools built on top of the MATLAB computing environment and the Signal Processing Toolbox. The Filter Design Toolbox provides advanced techniques for designing, simulating, and analyzing digital filters. It extends the capabilities of the Signal Processing Toolbox, adding architectures and design methods for demanding real-time DSP applications and it also provides functions that simplify the design of fixed-point filters and analysis of quantization effects.

Financial Toolbox is used for a wide array of applications including fixed income pricing, yield, and sensitivity analysis; advanced term structure analysis; coupon cash flow date and accrued interest analysis; and derivative pricing and sensitivity analysis.

Frequency Domain System Identification Toolbox provides specialized tools for identifying linear dynamic systems from time responses or measurements of the system's frequency response. Frequency domain methods support continuous-time modeling, which can be a powerful and highly accurate complement to the more commonly used discrete-time methods. The methods in the Toolbox can be applied to problems such as the modeling of electronic, mechanical, and acoustical systems.

Fuzzy Logic Toolbox features a simple point-and-click interface that guides you effortlessly through the steps of fuzzy design, from setup to diagnosis. It provides built-in support for the latest fuzzy logic methods, such as fuzzy clustering and adaptive neuro-fuzzy learning. The Toolbox's interactive graphics let you instantly visualize and fine tune system behavior.

Higher-Order Spectral Analysis Toolbox contains specialized tools for analyzing signals using the cumulants, or higher-order spectra, of a signal. The Toolbox features a wide range of higher-order spectral analysis techniques, providing access to algorithms at the forefront of signal processing technology.

Image Processing Toolbox provides engineers and scientists with an extensive suite of robust digital image processing and analysis functions. Seamlessly integrated within the MATLAB development environment, the Image Processing Toolbox is designed to free technical professionals from the time consuming tasks of coding and debugging fundamental image processing and analysis operations from scratch. This translates into significant time saving and cost reduction benefits, enabling you to spend less time coding algorithms and more time exploring and discovering solutions to your problems.

Instrument Control Toolbox provides features for communicating with data acquisition devices and instruments, such as spectrum analyzers, oscilloscopes, and function generators. Support is provided for GPIB (IEEE-488, HPIB) and VISA communication protocols. You can generate data in MATLAB to send out to an instrument or read data into MATLAB for analysis and visualization.

Mapping Toolbox provides a comprehensive set of functions and graphical user interfaces for performing interactive geographic computations, data fusion, map projection display, generation of presentation graphics, and accessing external geographic data. In addition, the toolbox ships with several, widely used atlas data sets for global and regional displays. Its flexibility, broad functionality, ease-of-use, and built-in visualization tools serve a wide range of engineering and scientific users who work with geographically based information.

MATLAB Compiler serves for two primary user groups: - Developers looking to deploy MATLAB applications to standalone C/C++ applications and - users who want to compile their MATLAB algorithms to improve code performance by converting them to C. The MATLAB Compiler automatically converts M-files into C and C++ source code.

MATLAB C/C++ Graphics Library is a collection of approximately 100 graphics routines that works with the MATLAB C/C++ Math Library. By using the Graphics Library with the MATLAB Compiler and MATLAB C/C++ Math Library, you can automatically convert MATLAB GUIs, graphics, and images to C and C++ code.

MATLAB C/C++ Math Library is a compiled version of the math functions that resides within MATLAB. The library contains advanced math functionality ranging from fast Fourier transforms and singular value decompositions to random number generators that are callable from C or C++. The C/C++ Math Library serves two user groups: - MATLAB programmers who have developed an application or algorithm and want to convert their M-file to C/C++; - Programmers working in C and C++ who need a fast, easy-to-use matrix math library.

MATLAB Report Generator and Simulink Report Generators let you easily create standard and customized reports from your MATLAB, Simulink, and Stateflow models and data in multiple output formats, including HTML, RTF, XML, and SGML. You can automatically document your large-scale systems, as you create a set of reusable and extensible templates that facilitate the transfer of information across departments. Your reports can contain any information available from the MATLAB workspace, including data, variables, functions, MATLAB programs, models, and diagrams. You can even include snapshots of all system graphics or figures generated by your M-files or models.

MATLAB Runtime Server lets you turn any MATLAB application into a standalone product that is easy and cost-effective to distribute. By creating your own application-specific graphical user interfaces (GUIs) using the MATLAB GUI development tools, you can deliver MATLAB based products that do not require your end users to learn MATLAB.

MATLAB Web Server lets you easily deploy any MATLAB or Simulink based applications via the Web. The Web Server is an ideal deployment tool for developers who want a quick, inexpensive, and secure way to share their applications. MATLAB applications running on the MATLAB Web Server can be run on any machine with Internet access using a Netscape or Microsoft Web browser.

Neural Network Toolbox provides comprehensive support for the design, implementation, and simulation of many proven network paradigms and is completely extensible and customizable. Its consistent methodology and modular organization facilitate research, provide a flexible framework for experimentation, and simplify customization.

Optimization Toolbox extends the MATLAB environment to provide tools for general and large-scale optimization of nonlinear problems. Additional tools are provided for linear programming, quadratic programming, nonlinear least-squares, and solving nonlinear equations.

Partial Differential Equation (PDE) Toolbox contains tools for the study and solution of PDEs in two space dimensions (2-D) and time, using the finite element method (FEM). Its command line functions and graphical user interface can be used for mathematical modeling of PDEs in a broad range of engineering and science applications, including structural mechanics, electromagnetics, heat transfer, and diffusion.

Robust Control Toolbox provides tools for the design and analysis of multivariable control systems where robustness is a concern. This includes systems where there may be modeling errors, dynamics that are not completely known, or parameters that can vary during the lifespan of the product. The powerful algorithms in this toolbox allow you to perform complex calculations while considering a number of parameter variations.

Signal Processing Toolbox provides a rich, customizable framework for digital signal processing (DSP). Built on a solid foundation of filter design and spectral analysis techniques, the toolbox contains powerful tools for algorithm development, signal and linear system analysis, and time-series data modeling.

Simulink is an interactive tool for modeling, simulating, and analyzing dynamic systems. It enables you to build graphical block diagrams, simulate dynamic systems, evaluate system performance, and refine your designs. Simulink integrates seamlessly with MATLAB, providing you with immediate access to an extensive range of analysis and design tools. Simulink is tightly integrated with Stateflow for modeling event-driven behavior. These benefits make Simulink the tool of choice for control system design, DSP design, communications system design, and other simulation applications.

The Spline Toolbox is a collection of MATLAB functions for data fitting, interpolation, extrapolation, and visualization.

Stateflow is an interactive design tool for modeling and simulating complex event-driven systems. Tightly integrated with Simulink and MATLAB, Stateflow provides an elegant solution for designing embedded systems that contain supervisory logic. Its combination of graphical modeling and animated simulation brings system specification and design closer together.

Statistics Toolbox is an easy-to-use environment for analyzing historical data, modeling systems to predict their behavior, developing statistical algorithms, and learning and teaching statistics. Interactive GUI tools let you apply statistical methods easily and consistently, while the MATLAB language lets you easily create custom statistical methods and analyses. This combination gives you the freedom to access functions such as probability and ANOVA directly from the command line, or to use the interactive interfaces to learn and experiment with the Toolbox's built-in visualization and analysis tools.

Symbolic/Extended Math Toolbox integrates symbolic mathematics and variable precision computation into MATLAB. The toolbox incorporates the computational kernel of Maple V release 5, developed by Waterloo Maple Software. Extended Symbolic Math adds support for full Maple programming and Maple's specialized libraries. With the Symbolic Math Toolboxes, MATLAB users can easily combine numeric and symbolic computation into a single environment without sacrificing speed or accuracy.

System Identification Toolbox is a collection of MATLAB® functions for creating mathematical models of dynamic systems. Both novices and experts can build accurate, simplified models of complex systems based on observed input/output data. Applications include control system design, model-based and adaptive signal processing, time-series analysis, and vibration analysis.

Wavelet Toolbox provides a comprehensive collection of routines for examining local, multiscale, and nonstationary phenomena. Wavelet methods offer additional insight and performance in any application where Fourier techniques have been used. The Wavelet Toolbox is useful in many signal processing applications, including speech and audio processing, communications, geophysics, finance, and medicine.