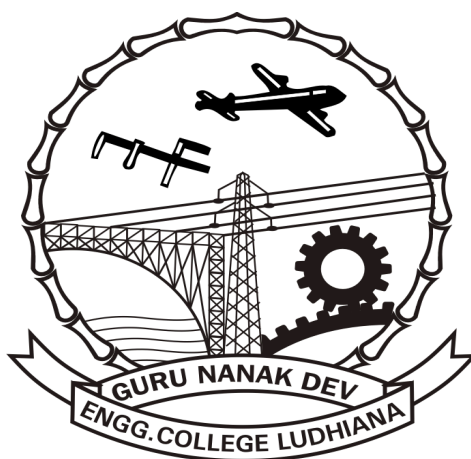


INSTRUCTION MANUAL

SIMULATION AND MODELLING LAB (BTCS-607)



Prepared by

**Er. Diana
Assistant Professor (CSE)**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**GURU NANAK DEV ENGINEERING COLLEGE
LUDHIANA – 141006**

DECLARATION

This Manual of Simulation And Modeling (BTCS-607) has been prepared by me as per syllabus of Simulation And Modeling (BTCS-607).

Signature

Syllabus

1. **Programming in MATLAB:** Introduction, Branching statements, loops, functions, additional data types, plots, arrays, inputs/outputs etc.
2. Introduction regarding usage of any Network Simulator.
3. Practical Implementation of Queuing Models using C/C++.

List of Practical

| Sr. No. | Topic | PAGE NUMBER |
|----------------|--|--------------------|
| 1. | Introduction to MATLAB | 1 |
| 2. | Programming in MATLAB: Introduction, Branching statements, loops, functions, additional data types, plots, arrays, inputs/outputs etc. | 4 |
| 3. | Introduction regarding usage of any Network Simulator. | 8 |
| 4. | Practical Implementation of Queuing Models using C/C++. | 10 |
| 5. | *Applications of MATLAB | 11 |

*Learning Beyond Syllabus Applications of MATLAB

Experiment 1

AIM : Introduction to MATLAB

MATLAB

MATLAB is widely used in all areas of applied mathematics, in education and research at universities, and in the industry. MATLAB stands for MATrix LABoratory and the software is built up around vectors and matrices. This makes the software particularly useful for linear algebra but MATLAB is also a great tool for solving algebraic and differential equations and for numerical integration. MATLAB has powerful graphic tools and can produce nice pictures in both 2D and 3D. It is also a programming language, and is one of the easiest programming languages for writing mathematical programs. MATLAB also has some tool boxes useful for signal processing, image processing, optimization, etc.

How to start MATLAB

Mac: Double-click on the icon for MATLAB.

PC: Choose the submenu "Programs" from the "Start" menu. From the "Programs" menu, open the "MATLAB" submenu. From the "MATLAB" submenu, choose "MATLAB".

Unix: At the prompt, type matlab.

You can quit MATLAB by typing exit in the command window.

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including Graphical User Interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

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

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to *learn* and *apply* specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

The MATLAB System

The MATLAB system consists of five main parts:

The MATLAB language.

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

The MATLAB working environment.

This is the set of tools and facilities that you work with as the MATLAB user or programmer. It includes facilities for managing the variables in your workspace and importing and exporting data. It also includes tools for developing, managing, debugging, and profiling M-files, MATLAB's applications.

Handle Graphics.

This is the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that allow you to fully customize the appearance of graphics as well as to build complete Graphical User Interfaces on your MATLAB applications.

The MATLAB mathematical function library.

This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix

inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

The MATLAB Application Program Interface (API).

This is a library that allows you to write C and Fortran programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

Experiment 2

AIM : Programming in MATLAB

Branching

When an "Algorithm" makes a choice to do one of two (or more things) this is called branching. The most common programming "statement" used to branch is the "IF" statement.

The If Statement

In a computer program, the algorithm often must choose to do one of two things depending on the "state" of the program. If the grade is greater than 90, then give the student an A, otherwise if the grade is greater than 80, give the student a B,... etc.

The most used way to make a decision in a program is the "IF" statement. The statement looks like:

```
if ( something is true ) do
    this code;
do all code before the end or else;
    do not do anything in the else "block" of code else
% if the something is false (NOTE: we don't have to test this) do
other code;
end
```

The ELSE statement

The else statement shown above is optional... if "NOTHING" is to be done when the condition is false, then the else statement should not be used! For example, below we only congratulate our good students, we do **not** chastise our bad students.

```
grade = % some_number;
```



```

        if ( grade > 75 )
fprintf('congrats, your grade %d is passing\n', grade);
        end

```

The ELSEIF statement

Because we often break problems into several sub paths, Matlab provides an "elseif" control statement. For example, if we have 5 dollars go to the dollar theater, "else if" we have 10 dollars go to the regular theater, "else if" you have 100 dollars, go to a Broadway play, else if you have 1000000 dollars, buy a theater...

The code in Matlab would look like:

```

if (money < 5)
    do this;
elseif (money < 10)
    do that;
elseif (money < 1000)
    do a lot of stuff;;
else
    do a default action when nothing above is true;
end

```

Loop Control Statements

With loop control statements, you can repeatedly execute a block of code. There are two types of loops:

for statements loop a specific number of times, and keep track of each iteration with an incrementing index variable.

For example, preallocate a 10-element vector, and calculate five values:

```

x = ones(1,10);
for n = 2:6
    x(n) = 2 * x(n - 1);
end

```

while statements loop as long as a condition remains true.

For example, find the first integer n for which $\text{factorial}(n)$ is a 100-digit number:

```
n = 1;
nFactorial = 1;
while nFactorial < 1e100
    n = n + 1;
    nFactorial = nFactorial * n;
end
```

Each loop requires the end keyword.

It is a good idea to indent the loops for readability, especially when they are nested (that is, when one loop contains another loop):

```
A = zeros(5,100);
for m = 1:5
    for n = 1:100
        A(m, n) = 1/(m + n - 1);
    end
end
```

You can programmatically exit a loop using a `break` statement, or skip to the next iteration of a loop using a `continue` statement. For example, count the number of lines in the help for the magic function (that is, all comment lines until a blank line):

```
fid = fopen('magic.m','r');
count = 0;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line)
        break
    elseif ~strncmp(line,'% ',1)
        continue
    end
    count = count + 1;
end
fprintf('%d lines in MAGIC help\n',count);
fclose(fid);
```

Useful functions and operations in MATLAB

Using MATLAB as a calculator is easy.

Example: Compute $5 \sin(2.5^{3-\pi}) + 1/75$. In MATLAB this is done by simply typing $5*\sin(2.5^{(3-\pi)})+1/75$ at the prompt. Be careful with parantheses and don't forget to type * whenever you multiply!

Note that MATLAB is *case sensitive*. This means that MATLAB knows a difference between letters written as lower and upper case letters. For example, MATLAB will understand $\sin(2)$ but will not understand $\text{Sin}(2)$.

Here is a table of useful operations, functions and constants in MATLAB.

| Operation, function or constant | MATLAB command |
|------------------------------------|----------------|
| + (addition) | + |
| - (subtraction) | - |
| \times (multiplication) | * |
| / (division) | / |
| x (absolute value of x) | abs(x) |
| square root of x | sqrt(x) |
| e^x | exp(x) |
| $\ln x$ (natural log) | log(x) |
| $\log_{10} x$ (base 10 log) | log10(x) |
| $\sin x$ | sin(x) |
| $\cos x$ | cos(x) |
| $\tan x$ | tan(x) |
| $\cot x$ | cot(x) |
| $\arcsin x$ | asin(x) |
| $\arccos x$ | acos(x) |
| $\arctan x$ | atan(x) |
| $\text{arccot } x$ | acot(x) |
| $n!$ (n factorial) | gamma(n+1) |
| e (2.71828...) | exp(1) |
| (3.14159265...) | pi |
| i (imaginary unit, $\sqrt{-1}$) | i |

Experiment 3

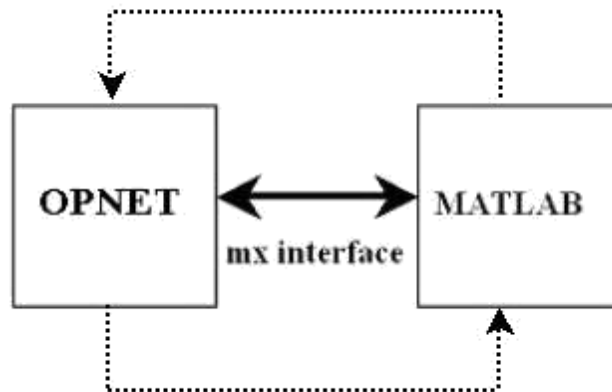
AIM : Introduction regarding usage of any Network Simulator.

OPNET provides a discrete event simulation engine. MATLAB is software for numerical calculations and provides communication engineers with a vast library for implementing communication systems such as channel models and beam forming algorithms. By integrating MATLAB simulation with OPNET we are able to reuse the beam forming algorithms developed in MATLAB and analyze their performance and effect on the upper layers

(specifically, data link and network layers) of the communication system. This would be difficult to realize without a discrete event simulator. In this thesis, we have interfaced MATLAB and OPNET so as to reuse the antenna beam steering algorithms, which were developed in MATLAB, and use the graphics library of MATLAB to provide the capability to observe the dynamic changes in the antenna patterns during simulation execution. For interfacing OPNET and MATLAB, we made use of the MX interface provided by MATLAB, which allows C programs to call functions developed in MATLAB. This is illustrated in Figure 3.1. For calling MATLAB functions the user needs to include following files in the *bind_shobj_libs* environment attribute.

1. libmat.lib
2. libeng.lib
3. libmex.lib
4. libmx.lib

Weights of Antenna Elements



Antenna and Channel Attributes

The directory where the above files are present is included in *bind_shobj_flags*. After including the necessary files into the *include* path, the MATLAB engine is started by OPNET simulation at the beginning of the simulation by using the function `engOpen()`. This provides the OPNET

simulation with a pointer to a memory location that can be used to pass MATLAB commands to the MATLAB engine. The engine pointer can be shared among different processes by declaring the engine pointer in a header file common to all process models. Variables can be exchanged between OPNET and MATLAB using functions `engPutArray()` and `engGetArray()`.

Network Modeling Using OPNET

OPNET is among the leading discrete event network simulators used both by the commercial and research communities. It provides a comprehensive framework for modeling wired as well as wireless network scenarios.

Simulation models are organized in a hierarchy consisting of three main levels: the simulation network, node models and process models. The top level refers to the simulation scenario or simulation network. It defines the network layout, the nodes and the configuration of attributes of the nodes comprising the scenario. OPNET Modeler™ uses an object-oriented approach for the development of models and simulation scenarios. The models can be identified as a *CLASS*, which can be reused any number of times in the simulation by creating its different instantiations, just like the creation of objects in any object-oriented programming language. Besides allowing the creation of multiple instances, OPNET allows the user to extend the functionality of the basic models already available as part of the model library. Thus, by defining the value of the attributes of the basic model the user can develop customized models following particular standards or vendor specifications.

Experiment 4

AIM: Practical implementation of Queuing Models using C/C++

Program1

```
#ifndef SIMULATION

#include <iostream>

#include <queue>

#include <list>

#include <fstream>
using namespace std;
class SimulateClass
{
private:

typedef list<int> eventList;
typedef queue<int> bankQueue;

void processArrival(int, ifstream& , eventList&, bankQueue& );

void processDeparture(int, eventList&, bankQueue& );
int arrivalCount;

void displaySimulation();

int waitingTime;

    int waitingTime;
int departureCalls;

    int newEvent;

SimulateClass(); void
simulateBank();

};

#endif
```

Experiment 5

AIM : Applications of Matlab

You can use MATLAB for a range of applications, including signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology. More than a million engineers and scientists in industry and academia use MATLAB, the language of technical computing.

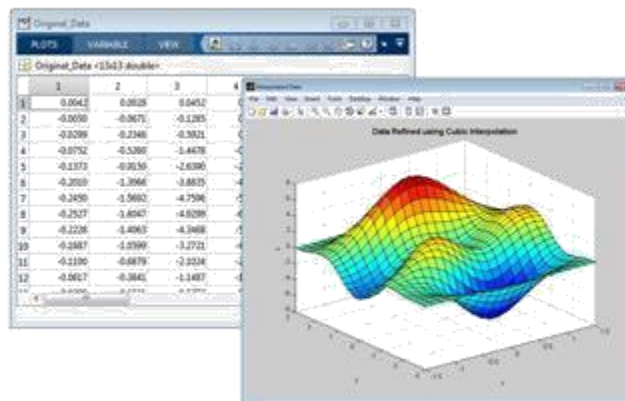
Numeric Computation

MATLAB provides a range of numerical computation methods for analyzing data, developing algorithms, and creating models. The MATLAB language includes mathematical functions that support common engineering and science operations. Core math functions use processor-optimized libraries to provide fast execution of vector and matrix calculations.

Available methods include:

Interpolation and regression
Differentiation and integration
Linear systems of equations
Fourier analysis
Eigenvalues and singular values
Ordinary differential equations (ODEs)
Sparse matrices

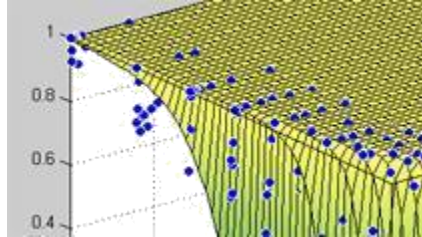
MATLAB add-on products provide functions in specialized areas such as statistics, optimization, signal analysis, and machine learning.



Refinement of gridded data using 2-D cubic interpolation.

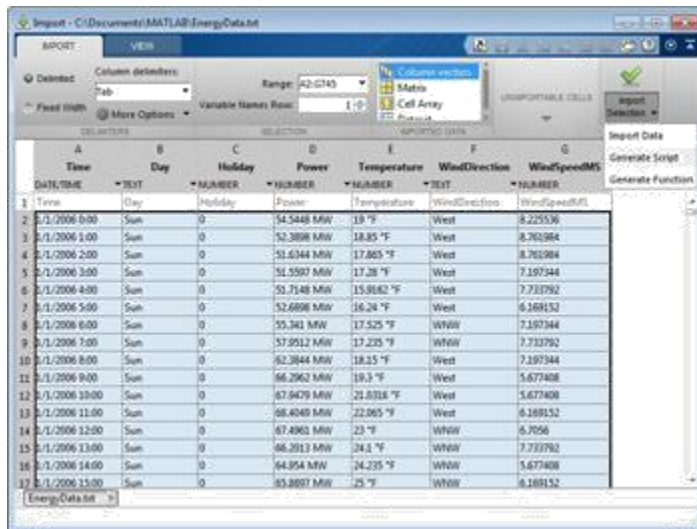
Data Analysis and Visualization

MATLAB provides tools to acquire, analyze, and visualize data, enabling you to gain insight into your data in a fraction of the time it would take using spreadsheets or traditional programming languages. You can also document and share your results through plots and reports or as published MATLAB code.



Acquiring Data

MATLAB lets you access data from files, other applications, databases, and external devices. You can read data from popular file formats such as Microsoft Excel; text or binary files; image, sound, and video files; and scientific files such as netCDF and HDF. File I/O functions let you work with data files in any format.



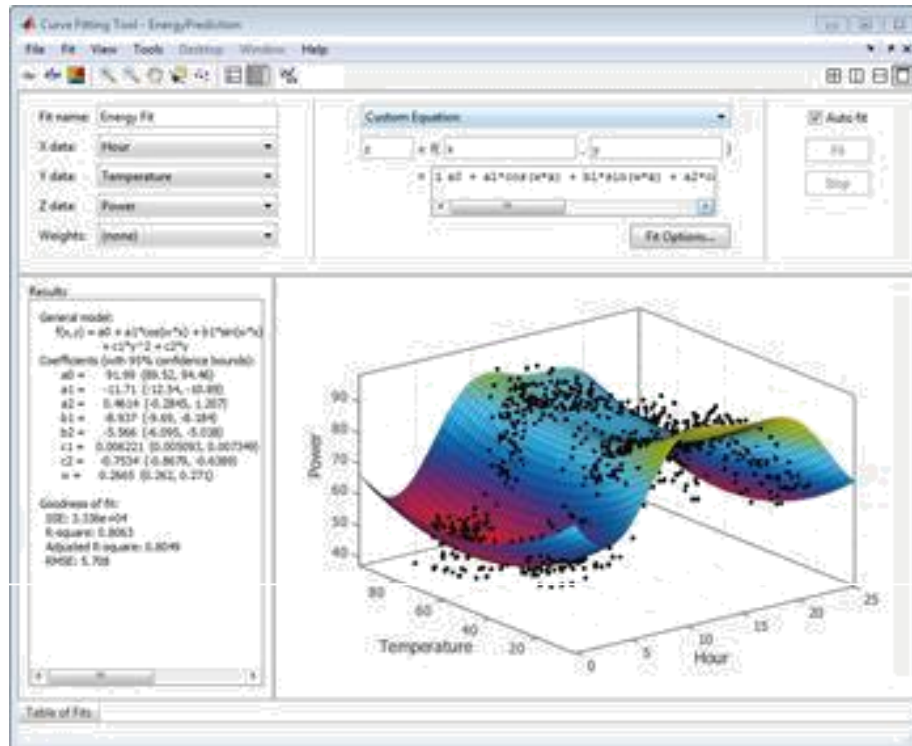
| A | B | C | D | E | F | G |
|-----------|----------------|--------|---------|------------|-------------|---------------|
| DATE/TIME | TEXT | NUMBER | NUMBER | NUMBER | TEXT | NUMBER |
| 1 | Time | Day | Holiday | Power | Temperature | WindDirection |
| 2 | 2/1/2006 0:00 | Sun | 0 | 54.5448 MW | 19 °F | West |
| 3 | 2/1/2006 1:00 | Sun | 0 | 52.3808 MW | 18.85 °F | West |
| 4 | 2/1/2006 2:00 | Sun | 0 | 51.6344 MW | 17.865 °F | West |
| 5 | 2/1/2006 3:00 | Sun | 0 | 51.5587 MW | 17.38 °F | West |
| 6 | 2/1/2006 4:00 | Sun | 0 | 51.7548 MW | 15.8682 °F | West |
| 7 | 2/1/2006 5:00 | Sun | 0 | 52.6698 MW | 16.24 °F | West |
| 8 | 2/1/2006 6:00 | Sun | 0 | 55.341 MW | 17.525 °F | WNW |
| 9 | 2/1/2006 7:00 | Sun | 0 | 57.9512 MW | 17.235 °F | WNW |
| 10 | 2/1/2006 8:00 | Sun | 0 | 62.3844 MW | 18.15 °F | West |
| 11 | 2/1/2006 9:00 | Sun | 0 | 66.2962 MW | 19.3 °F | West |
| 12 | 2/1/2006 10:00 | Sun | 0 | 67.8479 MW | 21.8318 °F | West |
| 13 | 2/1/2006 11:00 | Sun | 0 | 68.4049 MW | 22.965 °F | West |
| 14 | 2/1/2006 12:00 | Sun | 0 | 67.4963 MW | 23 °F | WNW |
| 15 | 2/1/2006 13:00 | Sun | 0 | 66.3913 MW | 24.3 °F | WNW |
| 16 | 2/1/2006 14:00 | Sun | 0 | 64.954 MW | 24.235 °F | WNW |
| 17 | 2/1/2006 15:00 | Sun | 0 | 63.8897 MW | 25 °F | WNW |

A mixed numeric and text file for import into MATLAB using the Import Tool. MATLAB automatically generates a script or function to import the file programmatically.

Using MATLAB with add-on products, you can [acquire data](#) from hardware devices, such as your computer's serial port or sound card, as well as stream live, measured data directly into MATLAB for analysis and visualization. You can also communicate with instruments such as oscilloscopes, function generators, and signal analyzers.

Analyzing Data

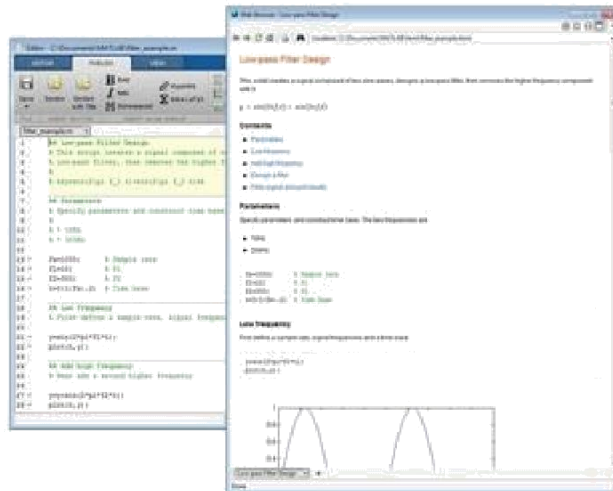
MATLAB lets you manage, filter, and preprocess your data. You can perform exploratory data analysis to uncover trends, test assumptions, and build descriptive models. MATLAB provides functions for filtering and smoothing, interpolation, convolution, and fast Fourier transforms (FFTs). Add-on products provide capabilities for curve and surface fitting, multivariate statistics, spectral analysis, image analysis, system identification, and other analysis tasks. Fitting a surface to data with a custom model using MATLAB and Curve Fitting Toolbox.



Visualizing Data

MATLAB provides built-in 2-D and 3-D plotting functions, as well as volume visualization functions. You can use these functions to visualize and understand data and communicate results. Plots can be customized either interactively or programmatically.

The MATLAB plot gallery provides examples of many ways to display data graphically in



MATLAB. For each example, you can view and download source code to use in your MATLAB application.

Editing the title of a surface contour plot using the MATLAB interactive plotting environment.

Documenting and Sharing Results

You can share results as plots or complete reports. MATLAB plots can be customized to meet publication specifications and saved to common graphical and data file formats.

You can automatically generate a report when you execute a MATLAB program. The report contains your code, comments, and program results, including plots. Reports can be published in a variety of formats, such as HTML, PDF, Word, or LaTeX.

MATLAB program (left) published as HTML (right) using the MATLAB Editor. Results that display in the Command Window or as plots are captured and included, and the code comments are turned into section

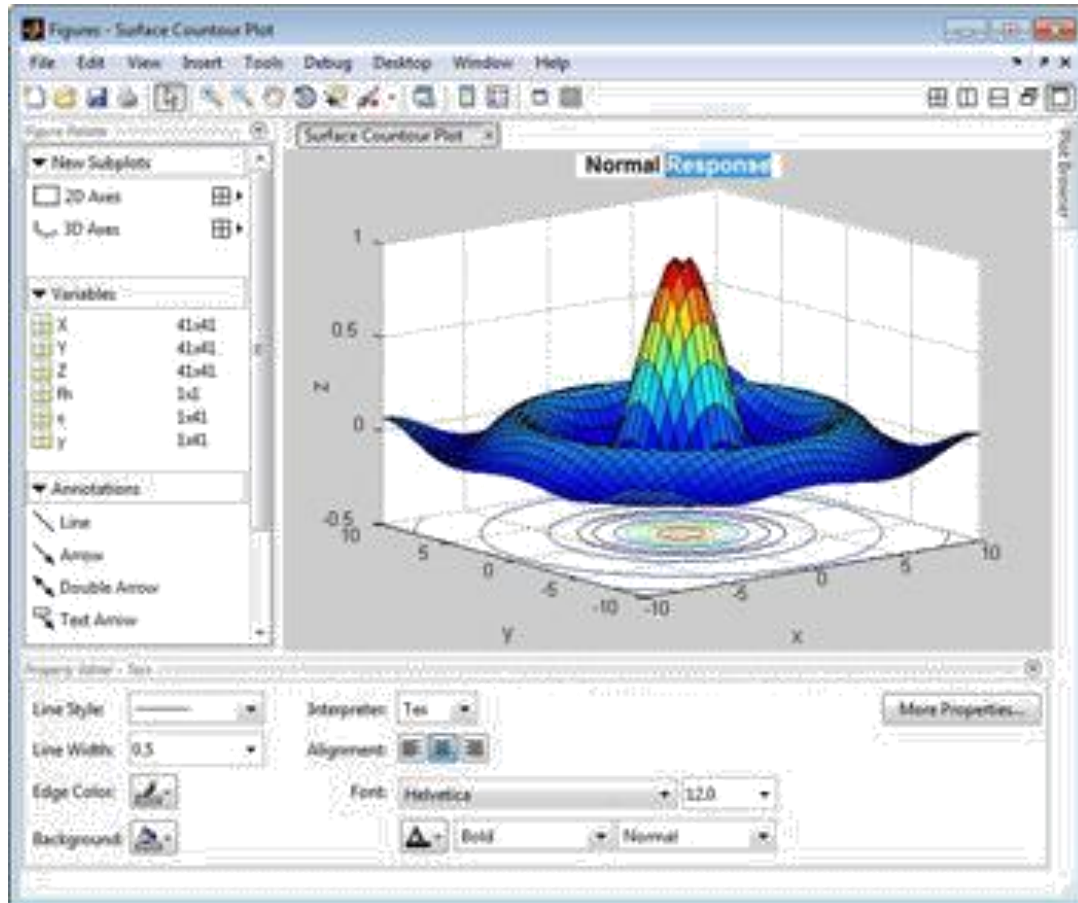
Programming and Algorithm Development

MATLAB provides a high-level language and development tools that let you quickly develop and analyze algorithms and applications.

- Key Features
- Numeric Computation
- Data Analysis and Visualization
- Programming and Algorithm Development

Application Development and Deployment *Programming and Algorithm Development*

MATLAB provides a high-level language and development tools that let you quickly develop and analyze algorithms and applications.



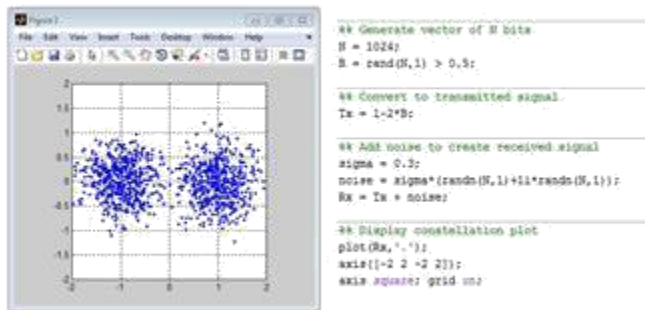
The MATLAB Language

The MATLAB language provides native support for the vector and matrix operations that are fundamental to solving engineering and scientific problems, enabling fast development and execution.

With the MATLAB language, you can write programs and develop algorithms faster than with traditional languages because you do not need to perform low-level administrative tasks such as declaring variables, specifying data types, and allocating memory. In many cases, the support for

vector and matrix operations eliminates the need for for-loops. As a result, one line of MATLAB code can often replace several lines of C or C++ code.

MATLAB provides features of traditional programming languages, including flow control, error handling, and object-oriented programming (OOP). You can use fundamental data types or advanced data structures, or you can define custom data types.



A communications algorithm that generates 1024 random bits, converts the vector to a transmitted signal, adds complex Gaussian noise, and plots the result in nine lines of MATLAB code.

You can produce immediate results by interactively executing commands one at a time. This approach lets you quickly explore multiple options and iterate to an optimal solution. You can capture interactive steps as scripts and functions to reuse and automate your work.

MATLAB add-on products provide built-in algorithms for signal processing and communications, image and video processing, control systems, and many other domains. By combining these algorithms with your own, you can build complex programs and applications.

Development Tools

MATLAB includes a variety of tools for efficient algorithm development, including:

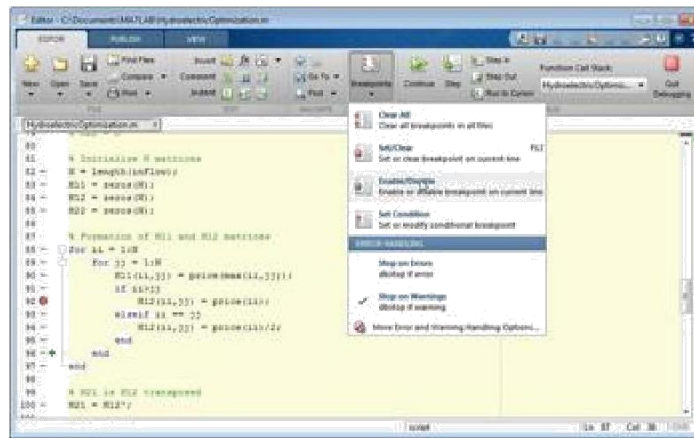
Command Window - Lets you interactively enter data, execute commands and programs, and display results

MATLAB Editor - Provides editing and debugging features, such as setting breakpoints and stepping through individual lines of code

Code Analyzer - Automatically checks code for problems and recommends modifications to maximize performance and maintainability

MATLAB Profiler - Measures performance of MATLAB programs and identifies areas of code to modify for improvement

Additional tools compare code and data files, and provide reports showing file dependencies, annotated reminders, and code coverage.



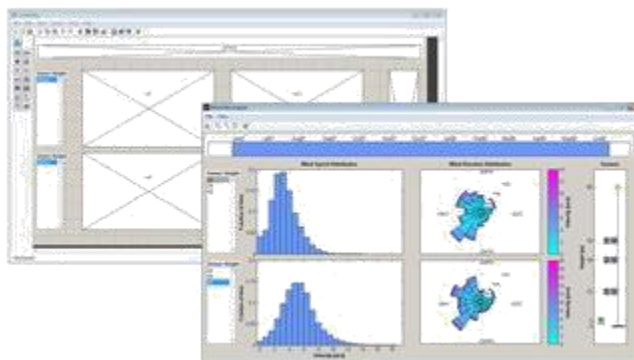
MATLAB program running in debug mode to diagnose problems.

Application Development and Deployment

MATLAB tools and add-on products provide a range of options to develop and deploy applications. You can share individual algorithms and applications with other MATLAB users or deploy them royalty-free to others who do not have MATLAB.

Designing Graphical User Interfaces

Using GUIDE (Graphical User Interface Development Environment), you can lay out, design, and edit custom graphical user interfaces. You can include common controls such as list boxes, pull-down menus, and push buttons, as well as MATLAB plots. Graphical user interfaces can also be created programmatically using MATLAB functions.



GUIDE layout of a wind analysis user interface (top) and the completed interface (bottom).

Deploying Applications

To distribute an application directly to other MATLAB users, you can package it as a MATLAB app, which provides a single file for distribution. Apps automatically install in the MATLAB apps gallery for easy access.

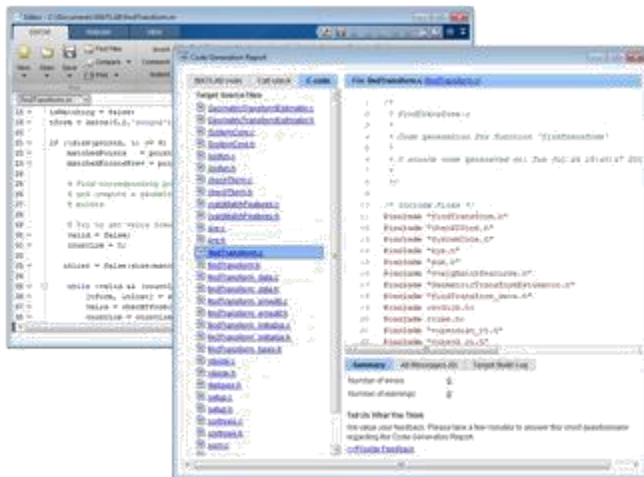
To share applications with others who do not have MATLAB, you can use application

deployment products. These add-on products automatically generate standalone applications, shared libraries, and software components for integration in C, C++, Java, .NET, and Excel environments. The executables and components can be distributed royalty-free.

MATLAB Production Server™ lets you run MATLAB programs packaged with MATLAB Compiler™ within your production systems, enabling you to incorporate numerical analytics in web, database, and enterprise applications.

Generating C Code

You can use MATLAB Coder™ to generate standalone C code from MATLAB code. MATLAB Coder supports a subset of the MATLAB language typically used by design engineers for developing algorithms as components of larger systems. This code can be used for standalone execution, for integration with other software applications, or as part of an embedded application.



MATLAB code (left) and code generation report (right) showing generated C code.