

Ahsanullah University of Science and Technology (AUST)
Department of Mechanical and Production Engineering (MPE)
IPE 4206: Industrial Simulation Sessional
Credit Hour: 1.5

General Guidelines:

1. Students must be prepared for the experiment prior to the class.
2. Report of an experiment must be submitted in the next class.
3. A quiz will be taken on the experiments at the end of the semester.
4. Marks distribution:

Total Marks: 100		
Attendance	Assignment	Quiz
10	20	70

Introduction:

A simulation is the imitation of the operation of a real-world process or system over time. Whether done by hand or on a Computer, simulation involves the generation of an artificial history of a system.

The behavior of a system as it evolves over time is studied by developing a simulation model. This model usually takes the form of a set of assumptions concerning the operation of the system.

Objective

Objective of this sessional is to observe how simulation modeling can be used both as an analysis tool for predicting the effect of changes to existing systems (Material handling, Scheduling, assembly, inventory, queuing etc.) & as a design tool to predict the performance of a new systems under varying sets of circumstances.

Briefing-1: Introduction to simulation systems, Models & Simulation

Consider a manufacturing firm that is contemplating building a large extension onto one of its plants but is not sure if the potential gain in productivity would justify the construction cost. It certainly would not be cost-effective to build the extension and then remove it later if it does not work out. However, a careful simulation study could shed some light on the question by simulating the operation of the plant as it currently exists and as it would be if the plant were expanded.

Application areas for simulation are numerous and diverse. Below is a list of some particular kinds of problems for which simulation has been found to be a useful and powerful tool:

- * Designing and analyzing manufacturing systems
- * Evaluating hardware and software requirements for a computer system
- * Evaluating a new military weapons system or tactic
- * Determining ordering policies for an inventory system
- * Designing communications systems and message protocols for them
- * Designing and operating transportation facilities such as freeways, airports, subways, or Ports
- * Evaluating designs for service organizations such as hospitals, post offices, or fast-food restaurants
- * Analyzing financial or economic systems

A system is defined to be a collection of entities, e.g., people or machines, that act and interact together toward the accomplishment of some logical end. In practice, what is meant by "the system" depends on the objectives of a particular study. The collection of entities that compose a system for one study might be only a subset of the overall system for another. For example, if one wants to study a bank to determine the number of tellers needed to provide adequate service for customers who want just to cash a check or make a savings deposit, the system can be defined to be that portion of the bank consisting of the tellers and the customers waiting in line or being served. If, on the other hand, the loan officer and the safety deposit boxes are to be included, the definition of the system must be expanded in an obvious way. We define the state of a system to be that collection of variables necessary to describe a system at a particular time, relative to the objectives of a study. In a study of a bank, examples of possible state variables are the number of busy tellers, the number of customers in the bank, and the time of arrival of each customer in the bank.

Discrete-event simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change instantaneously at separate points in time. (In more mathematical terms, we might say that the system can change at only a countable number of points in time.) These points in time are the ones at which an event occurs, where an event is defined as an instantaneous occurrence that may change the state of the system. Although discrete-event simulation could conceptually be done by hand calculations the amount of data that must be stored and manipulated for most real world systems dictates that discrete-event simulations be done on a digital computer.

Briefing-2: Simulation of Queuing Systems

Example: A small grocery store has only one checkout counter. Customers arrive at this checkout counter at random from 1 to 8 minutes apart. Each possible value of inter arrival time is the same probability of occurrence, as shown in Table 1. The service times vary from 1 to 6 minutes with the probabilities shown in Table 2. The problem is to analyze the system by simulation for the arrival and service of 20 customers.

Table 1

Time between Arrivals (Minutes)	Probability	Cumulative Probability	Random Assignment Digit
1	0.125	0.125	001-125
2	0.125	0.250	126-250
3	0.125	0.375	251-375
4	0.125	0.500	376-500
5	0.125	0.625	501-625
6	0.125	0.750	626-750
7	0.125	0.875	751-875
8	0.125	1.000	876-000

Table 2

Service Time (Minutes)	Probability	Cumulative Probability	Random Assignment Digit
1	0.10	0.10	01-10
2	0.20	0.30	11-30
3	0.30	0.60	31-60
4	0.25	0.85	61-85
5	0.10	0.95	86-95
6	0.05	1.00	96-00

Find out the following:

- ☐ The average waiting time of a customer
- ☐ The probability that a customer has to wait in the queue
- ☐ The proportion of idle time of the server
- ☐ The average service time of the server
- ☐ The average time between arrival
- ☐ The average waiting time of those who wait
- ☐ The average time a customer spends in the system

Assignment:

In the above exercise let the service distribution be changed to the following

Service Time (Minutes)	1	2	3	4	5	6
Probability	0.05	0.10	0.20	0.30	0.25	0.10

Develop the simulation table and the analysis for 20 customers. What is the effect of changing the service-time distribution?

Briefing-3: Simulation of Inventory System

Example: Suppose that the maximum inventory level M is 11 units and the review period N is 5 days. The problem is to estimate by simulation (for 5 Cycle) the average ending units in inventory and the number of days when a shortage condition existed. The distribution of the number of units demanded per day is shown in table. Lead time is random variable also shown in table. Assume that the orders are placed at the close of business. Beginning inventory is 3 units and an order of 8 units is scheduled to arrive in 2 days' time.

Table 1: Random Digit Assignment for Daily Demand

Demand	Probability	Cumulative Probability	Random Digit Assignment
0	0.10	0.10	01-10
1	0.25	0.35	11-35
2	0.35	0.70	36-70
3	0.21	0.91	71-91
4	0.09	1.00	92-00

Table 2: Random Digit Assignments for Lead Time

Lead Time (Days)	Probability	Cumulative Probability	Random Digit Assignment
1	0.6	0.6	1-6
2	0.3	0.9	7-9
3	0.1	1.0	0

Find the following:

- Average ending inventory
- Number of days' shortage condition existed

Assignment: rework the above example for 10 cycles with $M=10$

Briefing-4: Simulation on reliability problem

Example: A large milling machine has three different bearings that fall in service. The cumulative distribution function of the life of each bearing is identical as shown in the table. When a bearing fails, the mills stops, a repairperson is called, and a new bearing is installed. The delay time of the repairperson's arriving at the milling machine is also a random variable, with the distribution given in table 2. Downtime for the mill is estimated at \$5 per minute. The direct on site cost of the repairperson is \$15 per hour. It takes 20 minutes to change one bearing, 30 minutes to change two bearings, and 40 minutes to change three bearings. The bearings cost \$16 each. A proposal has been made to replace all three bearings whenever a bearing fails. Management needs an evaluation of this proposal.

Table 1: Random Digit Assignment for Bearing life

Bearing life (Hours)	Probability	Cumulative Probability	Random Digit Assignment
1000	0.10	0.10	01-10
1100	0.13	0.23	11-23
1200	0.25	0.48	24-48
1300	0.13	0.61	49-61
1400	0.09	0.70	62-70
1500	0.12	0.82	71-82
1600	0.02	0.84	83-84
1700	0.06	0.90	85-90
1800	0.05	0.95	91-95
1900	0.05	1.00	96-00

Table 2: Random Digit Assignments for Delay Time

Delay Time (Minutes)	Probability	Cumulative Probability	Random Digit Assignment
5	0.6	0.6	1-6
10	0.3	0.9	7-9
15	0.1	1.0	0

Simulate the problem for 20000 hours of operation. It will be assumed in this problem that the times are never exactly the same and thus no more than 1 bearing is changed at any breakdown. 16 bearings changes were made for bearings 1 and 2 but only 14 bearings were required for bearing 3.

Assignment: Rework the simulation of the proposed method using new random digits and 30000 hours.

Briefing-5: Simulation of Newspaper seller problem

Example: A paper seller buys the papers for 33 cents each and sells them for 50 cents each. Newspapers not sold at the end of the day are sold as scrap for 5 cents each. Newspaper can be purchased in the bundles of 10. Thus the paper seller can buy 50, 60, and so on. There are three types of news days, “good”, “fair” and “poor” of 0.35, 0.45 and 0.20. The distribution of papers demanded on each of these days is given below:

Table 1: Distribution of paper demand and the respective probability distribution

Demand	Demand probability distribution		
	Good	Fair	poor
40	0.13	0.23	11-23
50	0.25	0.48	24-48
60	0.13	0.61	49-61
70	0.09	0.70	62-70
80	0.12	0.82	71-82
90	0.02	0.84	83-84
100	0.06	0.90	85-90

The profits are given by the following relationships

Profit = revenue from sales- cost of newspapers- loss profit from excess demand+ salvage from sales of scarp papers

Determine the optimum number of papers the seller should purchase. This will be accomplished by simulating demands for 20 day and recording profits from sales each day. (Simulate for 40 newspaper)

Table 2: Random Digit Assignments for Type of Newsday

Types of News days	Probability	Cumulative Probability	Random Digit Assignment
Good	0.35	0.35	01-35
Fair	0.45	0.80	36-80
Poor	0.20	1.00	81-00

Table 3: Random digit assignment of Newspaper demanded

Demand	Cumulative distribution			Random Digit Assignment		
40	0.03	0.1	0.44	01-03	01-10	01-44
50	0.08	0.28	0.66	04-08	11-28	45-66
60	0.23	0.68	0.82	09-23	29-68	67-82
70	0.43	0.88	0.94	24-43	69-88	83-94
80	0.78	0.96	1	44-78	89-96	95-00
90	0.93	1	1	79-93	97-00	
100	1	1	1	94-00		

Assignment: Simulate the above example for 50 & 60 newspaper.

Lab Manual of Computer Simulation

Briefing 1: Variables, script and operations

MATLAB can be thought of as a super-powerful graphing calculator: Remember the TI-83 from calculus? With many more buttons (built-in functions). In addition it is a programming language MATLAB is an interpreted language, like Java Commands executed line by line

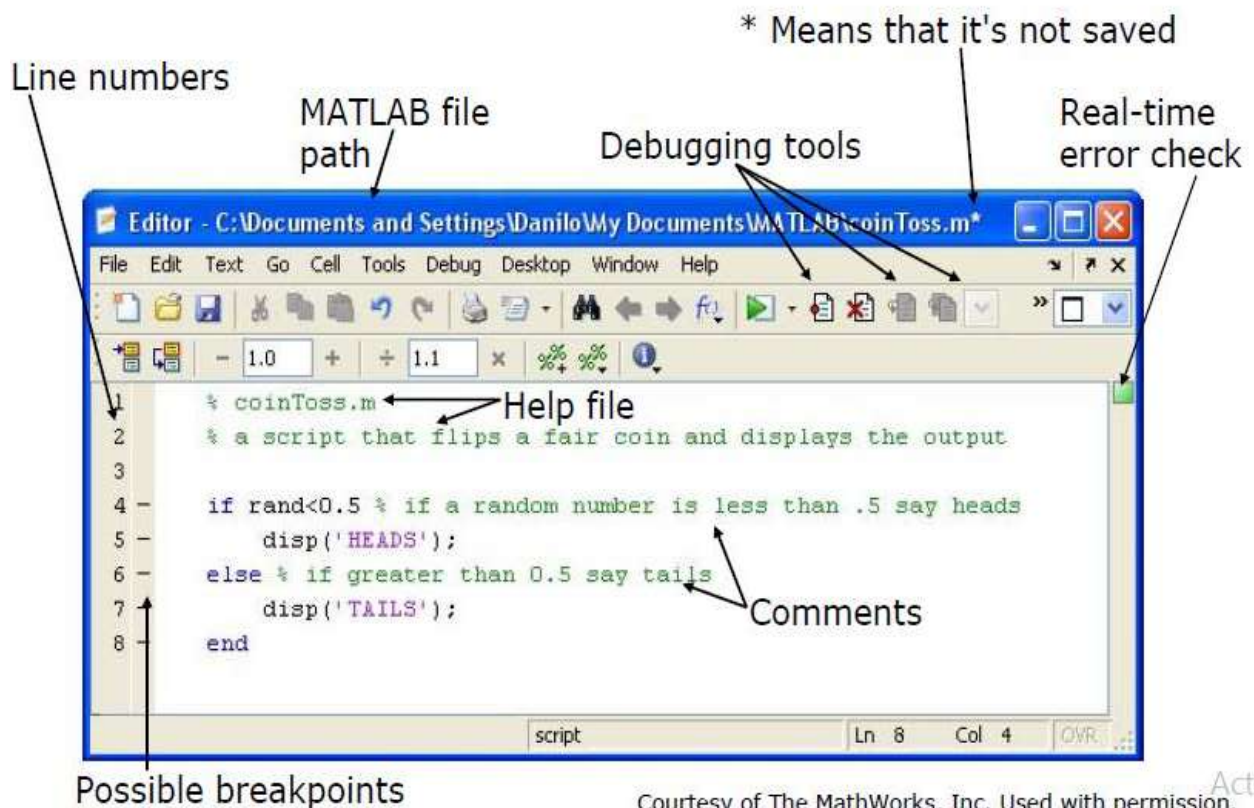
Help/Docs: help: The most important function for learning MATLAB on your own and to get info on how to use a function: »help sin>>Help lists related functions at the bottom and links to the doc: To get a nicer version of help with examples and easy-to-read descriptions: doc sin

To search for a function by specifying keywords: »doc + Search tab

Scripts: Overview

- Scripts are collection of commands executed in sequence written in the MATLAB editor saved as MATLAB files (.m extension)
- To create an MATLAB file from command-line
- »edit helloWorld.m
- COMMENT: Anything following a % is seen as a comment. The first contiguous comment becomes the script's help file. Comment thoroughly to avoid wasting time later note that scripts are somewhat static, since there is no input and no explicit output. All variables created and modified in a script exist in the workspace even after it has stopped running.
- Exercise: Scripts
- Make a helloWorld script
 - When run, the script should display the following text:
- Hello World!
- I am going to learn MATLAB!
- Hint: use disp to display strings. Strings are written between single quotes, like 'This is a string'.

Scripts: the Editor



Courtesy of The MathWorks, Inc. Used with permission.

Acti
Go tr

Variable Types

- MATLAB is a weakly typed language. No need to initialize variables! MATLAB supports various types, the most often used are
 - »3.84=64-bit double (default)
 - »'a'=16-bit char
- Most variables you'll deal with will be vectors or matrices of doubles or chars
- Other types are also supported: complex, symbolic, 16-bit and 8 bit integers, etc. You will be exposed to all these types through the homework.
- Naming variables
- To create a variable, simply assign a value to a name: »var1=3.1 »myString='hello world'

- Variable names: first character must be a LETTER, after that, any combination of letters, numbers and CASE SENSITIVE! (var1 is different from Var1)
- Built-in variables. Don't use these names!
- i and j can be used to indicate complex numbers
- pi has the value 3.1415926...
- ans stores the last unassigned value (like on a calculator)
- Inf and -Inf are positive and negative infinity
- NaN represents 'Not a Number'.

Row Vectors

- Row vector: comma or space separated values between brackets

```
» row = [1 2 5.4 -6.6]
```

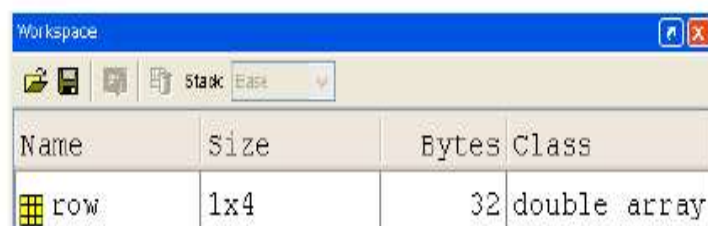
```
» row = [1, 2, 5.4, -6.6];
```

- Command window: `>> row=[1 2 5.4 -6.6]`

```
row =
```

```
1.0000    2.0000    5.4000   -6.6000
```

- Workspace:



The screenshot shows the MATLAB Workspace window. At the top, there are icons for saving, refreshing, and a 'Stack' dropdown menu set to 'Base'. Below this is a table with the following data:

Name	Size	Bytes	Class
row	1x4	32	double array

Column Vectors

- Column vector: semicolon separated values between brackets

```
» column = [4;2;7;4]
```

- Command window: `>> column=[4;2;7;4]`

```
column =
```

```
4
2
7
4
```

- Workspace:

Workspace			
Name	Size	Bytes	Class
column	4x1	32	double array

Matrices

- Make matrices like vectors

- Element by element

```
» a = [1 2;3 4];
```

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- By concatenating vectors or matrices (dimension matters)

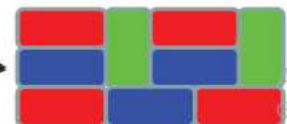
```
» a = [1 2];
» b = [3 4];
» c = [5;6];
```

```
» d = [a;b];
```

```
» e = [d c];
```

```
» f = [[e e];[a b a]];
» str = ['Hello, I am ' 'John'];
```

➤ Strings are character vectors



Act
Got

Basic Scalar Operations:

- Arithmetic operations (+, -, *, /) » 7/45 » (1+i)*(2+i) » 1 / 0 » 0 / 0
- Exponentiation (^) » 4^2 » (3+4*j)^2
- Complicated expressions, use parentheses » ((2+3)*3)^0.1
- Multiplication is NOT implicit given parentheses » 3(1+0.7) gives an error
- To clear command window » clc

Built-in Functions:

- MATLAB has an enormous library of built-in functions
- Call using parentheses – passing parameter to function » sqrt(2) » log(2), log10(0.23)
» cos(1.2), atan(-.8) » exp(2+4*i) » round(1.4), floor(3.3), ceil(4.23) » angle(i); abs(1+i);

Transpose:

- The transpose operators turns a column vector into a row vector and vice versa
- » a = [1 2 3 4+i]
- » transpose(a)
- » a'
- » a.'

Addition and Subtraction

- Addition and subtraction are element-wise; sizes must match (unless one is a scalar):

$$\begin{array}{r} \begin{bmatrix} 12 & 3 & 32 & -11 \\ 2 & 11 & -30 & 32 \end{bmatrix} \\ + \begin{bmatrix} 2 & 11 & -30 & 32 \end{bmatrix} \\ \hline \begin{bmatrix} 14 & 14 & 2 & 21 \end{bmatrix} \end{array} \qquad \begin{bmatrix} 12 \\ 1 \\ -10 \\ 0 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \\ 13 \\ 33 \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \\ -23 \\ -33 \end{bmatrix}$$

- The following would give an error
» c = row + column
- Use the transpose to make sizes compatible
» c = row' + column
» c = row + column'
- Can sum up or multiply elements of vector
» s=sum(row);
» p=prod(row);

Operators: element-wise

To do element-wise operations, use the dot: . (*, ./, .^). BOTH dimensions must match (unless one is scalar)!

- »a=[1 2 3];b=[4;2;1];
- »a.*b, a./b, a.^b Æ all errors
- »a.*b', a./b', a.^(b') Æ all valid

Operators: standard

- Multiplication can be done in a standard way or element-wise
- Standard multiplication (*) is either a dot-product or an outer-product
 - Remember from linear algebra: inner dimensions must MATCH!!
- Standard exponentiation (^) can only be done on square matrices or scalars
- Left and right division (/ \) is same as multiplying by inverse
 - Our recommendation: just multiply by inverse (more on this later)

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} * \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = 11$$

$1 \times 3 * 3 \times 1 = 1 \times 1$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

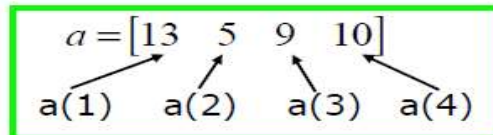
Must be square to do powers

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 6 & 12 & 18 \\ 9 & 18 & 27 \end{bmatrix}$$

$3 \times 3 * 3 \times 3 = 3 \times 3$

Vector Indexing

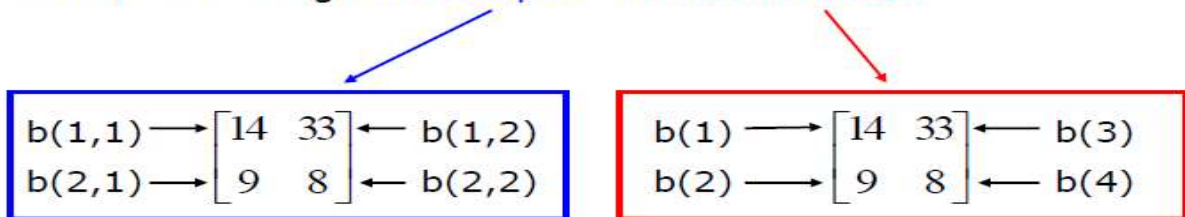
- MATLAB indexing starts with **1**, not **0**
 - We will not respond to any emails where this is the problem.
- `a(n)` returns the n^{th} element



- The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.
 - » `x = [12 13 5 8];`
 - » `a = x(2:3);` → `a = [13 5];`
 - » `b = x(1:end-1);` → `b = [12 13 5];`

Matrix Indexing

- Matrices can be indexed in two ways
 - using **subscripts** (row and column)
 - using linear **indices** (as if matrix is a vector)
- Matrix indexing: **subscripts** or **linear indices**



- Picking submatrices
 - » `A = rand(5)` % shorthand for 5x5 matrix
 - » `A(1:3,1:2)` % specify contiguous submatrix
 - » `A([1 5 3], [1 4])` % specify rows and columns

Advanced Indexing 1

- To select rows or columns of a matrix, use the **:**

$$c = \begin{bmatrix} 12 & 5 \\ -2 & 13 \end{bmatrix}$$

```
» d=c(1,:); → d=[12 5];  
» e=c(:,2); → e=[5;13];  
» c(2,:)= [3 6]; %replaces second row of c
```

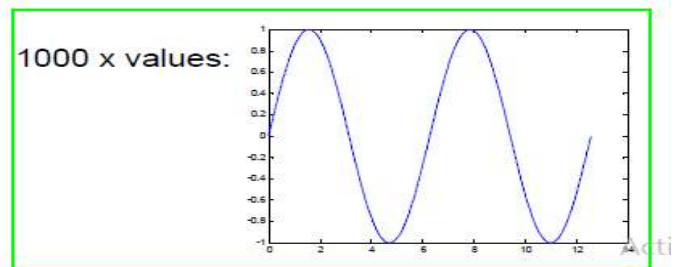
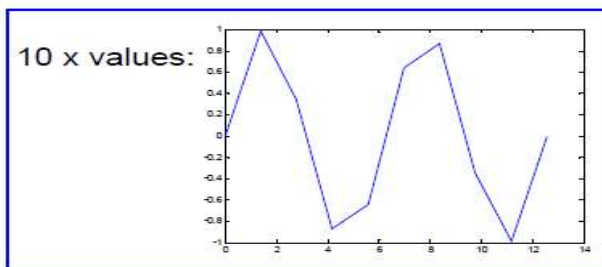
Advanced Indexing 2

- MATLAB contains functions to help you find desired values within a vector or matrix
» `vec = [5 3 1 9 7]`
- To get the minimum value and its index:
» `[minVal,minInd] = min(vec);`
➤ `max` works the same way
- To find any the indices of specific values or ranges
» `ind = find(vec == 9);`
» `ind = find(vec > 2 & vec < 6);`
➤ **find** expressions can be very complex, more on this later
- To convert between subscripts and indices, use **ind2sub**, and **sub2ind**. Look up **help** to see how to use them.

Plotting:

What does plot do?

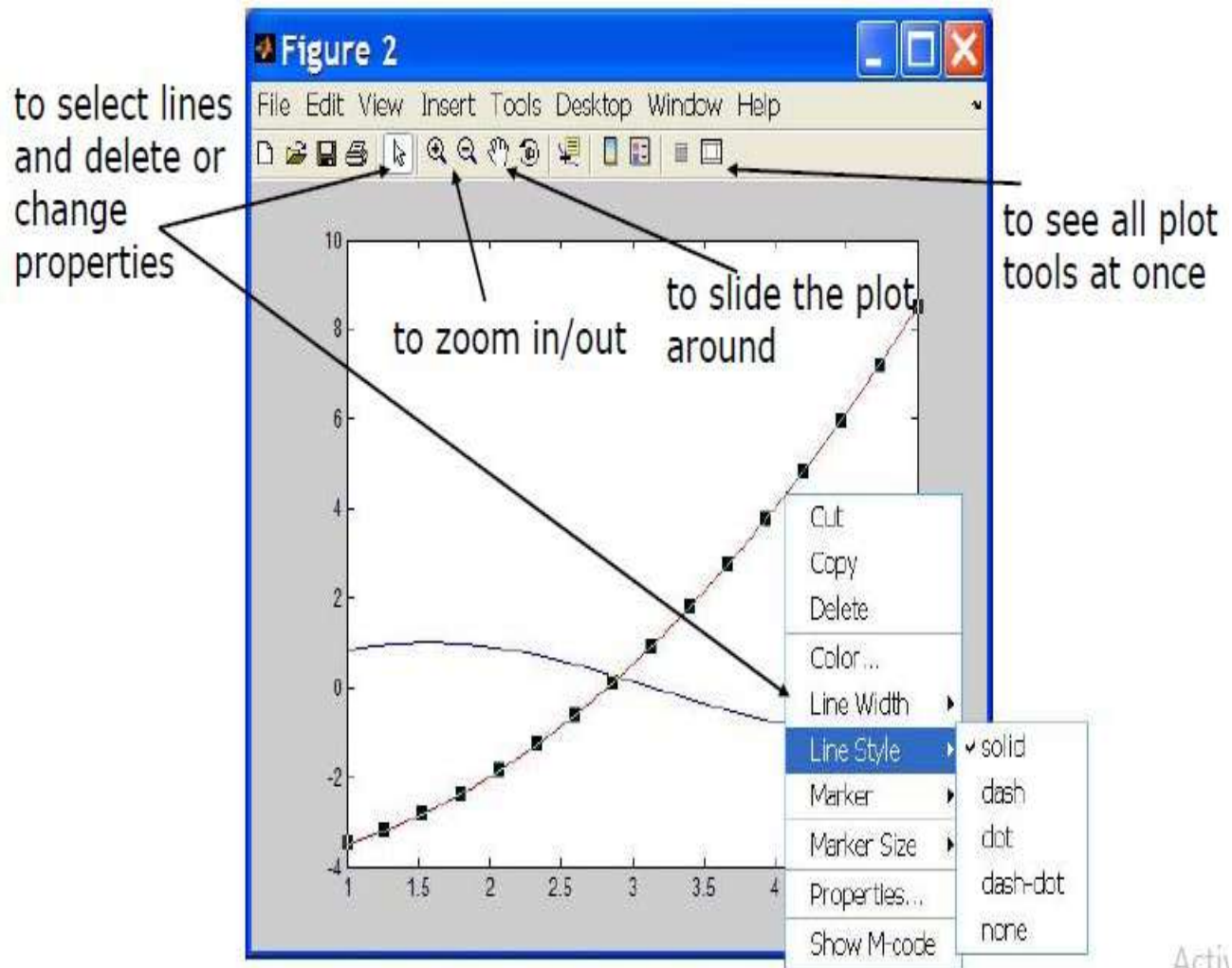
- **plot** generates dots at each (x,y) pair and then connects the dots with a line
- To make plot of a function look smoother, evaluate at more points
 - » `x=linspace(0,4*pi,1000);`
 - » `plot(x,sin(x));`
- x and y vectors must be same size or else you'll get an error
 - » `plot([1 2], [1 2 3])`
 - error!!



Plot Options

- Can change the line color, marker style, and line style by adding a string argument
 - » `plot(x,y,'k.-');`
 - color marker line-style
- Can plot without connecting the dots by omitting line style argument
 - » `plot(x,y,'.')`
- Look at **help plot** for a full list of colors, markers, and linestyles

Playing with the Plot

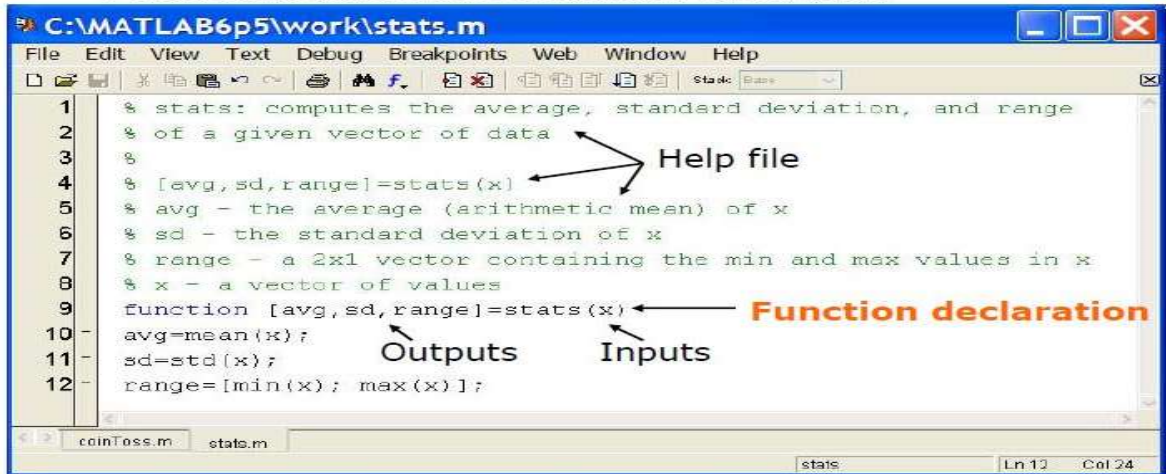


Courtesy of The MathWorks, Inc. Used with permission.

Activ
Go to

User-defined Functions

- Functions look exactly like scripts, but for **ONE** difference
 - Functions must have a function declaration



User-defined Functions

- Some comments about the function declaration

function [x, y, z] = funName(in1, in2)

Annotations:

- Must have the reserved word: function**: Points to the word `function`.
- If more than one output, must be in brackets**: Points to the output variables `[x, y, z]`.
- Function name should match MATLAB file name**: Points to the function name `funName`.
- Inputs must be specified**: Points to the input variables `in1, in2`.

- No need for return**: MATLAB 'returns' the variables whose names match those in the function declaration
- Variable scope**: Any variables created within the function but not returned disappear after the function stops running

Relational Operators

- MATLAB uses *mostly* standard relational operators
 - equal ==
 - not** equal ~=
 - greater than >
 - less than <
 - greater or equal >=
 - less or equal <=
- Logical operators
 - And & (elementwise) && (short-circuit (scalars))
 - Or | (elementwise) || (short-circuit (scalars))
 - Not** ~
 - Xor xor
 - All true all
 - Any true any
- Boolean values: zero is false, nonzero is true
- See **help** . for a detailed list of operators

if/else/elseif

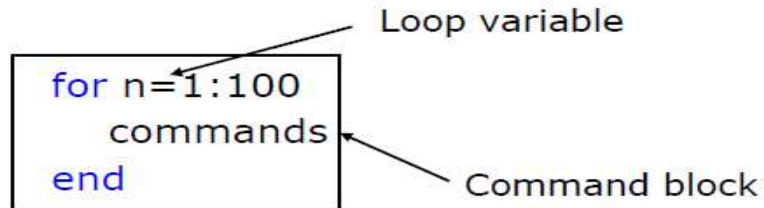
- Basic flow-control, common to all languages
- MATLAB syntax is somewhat unique



- No need for parentheses:** command blocks are between reserved words

for

- **for** loops: use for a known number of iterations
- MATLAB syntax:



- The loop variable
 - Is defined as a vector
 - Is a scalar within the command block
 - Does not have to have consecutive values (but it's usually cleaner if they're consecutive)
- The command block
 - Anything between the **for** line and the **end**

while

- The while is like a more general for loop:
 - Don't need to know number of iterations



- The command block will execute while the conditional expression is true
- Beware of infinite loops!

Systems of Linear Equations

- Given a system of linear equations
 - $x+2y-3z=5$
 - $-3x-y+z=-8$
 - $x-y+z=0$
- Construct matrices so the system is described by $Ax=b$
 - » `A=[1 2 -3;-3 -1 1;1 -1 1];`
 - » `b=[5;-8;0];`
- And solve with a single line of code!
 - » `x=A\b;`
 - x is a 3x1 vector containing the values of x , y , and z
- The `\` will work with square or rectangular systems.
- Gives least squares solution for rectangular systems. Solution depends on whether the system is over or underdetermined.

MATLAB makes linear algebra fun!



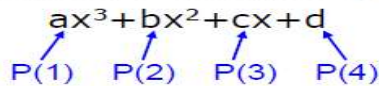
Exercise: Linear Algebra

- Solve the following systems of equations:
 - System 1:
 - $x+4y=34$
 - $-3x+y=2$
 - » `A=[1 4;-3 1];`
 - » `b=[34;2];`
 - » `rank(A)`
 - » `x=inv(A)*b;`
 - System 2:
 - $2x-2y=4$
 - $-x+y=3$
 - $3x+4y=2$
 - » `A=[2 -2;-1 1;3 4];`
 - » `b=[4;3;2];`
 - » `rank(A)`
 - rectangular matrix
 - » `x1=A\b;`
 - gives least squares solution
 - » `error=abs(A*x1-b)`

Polynomials

- Many functions can be well described by a high-order polynomial
- MATLAB represents a polynomials by a vector of coefficients
 - if vector P describes a polynomial

$$ax^3 + bx^2 + cx + d$$


P(1) P(2) P(3) P(4)

- $P=[1 \ 0 \ -2]$ represents the polynomial x^2-2
- $P=[2 \ 0 \ 0 \ 0]$ represents the polynomial $2x^3$

Polynomial Operations

- P is a vector of length N+1 describing an N-th order polynomial
- To get the roots of a polynomial

» `r=roots(P)`

➤ r is a vector of length N

- Can also get the polynomial from the roots

» `P=poly(r)`

➤ r is a vector length N

- To evaluate a polynomial at a point

» `y0=polyval(P,x0)`

➤ x0 is a single value; y0 is a single value

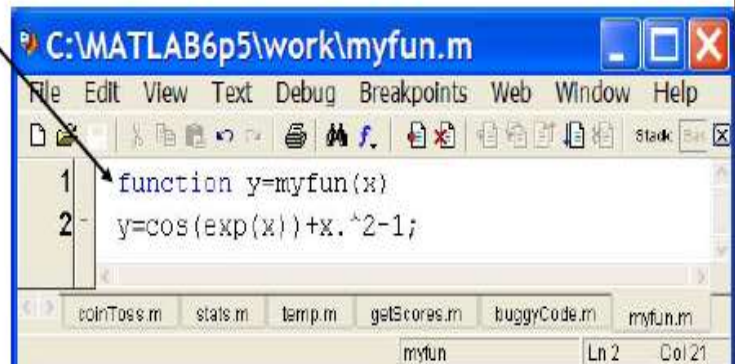
- To evaluate a polynomial at many points

» `y=polyval(P,x)`

➤ x is a vector; y is a vector of the same size

Nonlinear Root Finding

- Many real-world problems require us to solve $f(x)=0$
- Can use **fzero** to calculate roots for *any* arbitrary function
- **fzero** needs a function passed to it.
- We will see this more and more as we delve into solving equations.
- Make a separate function file
 - » `x=fzero('myfun',1)`
 - » `x=fzero(@myfun,1)`
 - 1 specifies a point close to where you think the root is



Courtesy of The MathWorks, Inc. Used with permission. Activ

Go to

Minimizing a Function

- **fminbnd**: minimizing a function over a bounded interval
 - » `x=fminbnd('myfun', -1, 2);`
 - myfun takes a scalar input and returns a scalar output
 - myfun(x) will be the minimum of myfun for $-1 \leq x \leq 2$
- **fminsearch**: unconstrained interval
 - » `x=fminsearch('myfun', .5)`
 - finds the local minimum of myfun starting at $x=0.5$

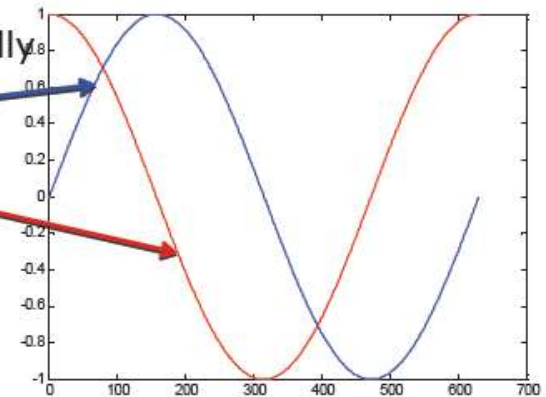
Optimization Toolbox

- If you are familiar with optimization methods, use the optimization toolbox
- Useful for larger, more structured optimization problems
- Sample functions (see [help](#) for more info)
 - » `linprog`
 - linear programming using interior point methods
 - » `quadprog`
 - quadratic programming solver
 - » `fmincon`
 - constrained nonlinear optimization

Numerical Differentiation

- MATLAB can 'differentiate' numerically

```
» x=0:0.01:2*pi;  
» y=sin(x);  
» dydx=diff(y)./diff(x);  
➤ diff computes the first difference
```



- Can also operate on matrices

```
» mat=[1 3 5;4 8 6];  
» dm=diff(mat,1,2)  
➤ first difference of mat along the 2nd dimension, dm=[2 2;4 -2]  
➤ see help for more details  
➤ The opposite of diff is the cumulative sum cumsum
```

- 2D gradient

```
» [dx,dy]=gradient(mat);
```

A

Numerical Integration

- MATLAB contains common integration methods
- Adaptive Simpson's quadrature (input is a function)
 - » `q=quad('myFun',0,10);`
➤ q is the integral of the function **myFun** from 0 to 10
 - » `q2=quad(@(x) sin(x)*x,0,pi)`
➤ q2 is the integral of **sin(x)*x** from 0 to pi
- Trapezoidal rule (input is a vector)
 - » `x=0:0.01:pi;`
 - » `z=trapz(x,sin(x));`
➤ z is the integral of **sin(x)** from 0 to pi
 - » `z2=trapz(x,sqrt(exp(x))./x)`
➤ z2 is the integral of $\sqrt{e^x}/x$ from 0 to pi

ODE Solvers: MATLAB

- MATLAB contains implementations of common ODE solvers
- Using the correct ODE solver can save you lots of time and give more accurate results
 - » `ode23`
 - Low-order solver. Use when integrating over small intervals or when accuracy is less important than speed
 - » `ode45`
 - High order (Runge-Kutta) solver. High accuracy and reasonable speed. Most commonly used.
 - » `ode15s`
 - Stiff ODE solver (Gear's algorithm), use when the diff eq's have time constants that vary by orders of magnitude

ODE Solvers: Standard Syntax

- To use standard options and variable time step

```
» [t,y]=ode45('myODE',[0,10],[1;0])
```

ODE integrator:
23, 45, 15s

ODE function

Time range

Initial conditions

- Inputs:
 - ODE function name (or anonymous function). This function takes inputs (t,y), and returns dy/dt
 - Time interval: 2-element vector specifying initial and final time
 - Initial conditions: column vector with an initial condition for each ODE. This is the first input to the ODE function
- Outputs:
 - t contains the time points
 - y contains the corresponding values of the integrated variables.

Ac
Go

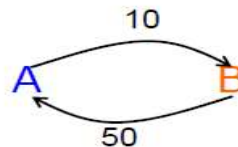
ODE Function

- The ODE function must return the value of the derivative at a given time and function value
- Example: chemical reaction

- Two equations

$$\frac{dA}{dt} = -10A + 50B$$

$$\frac{dB}{dt} = 10A - 50B$$



- ODE file:
 - y has [A;B]
 - dydt has [dA/dt;dB/dt]

Courtesy of The MathWorks, Inc. Used with permission.

Go to

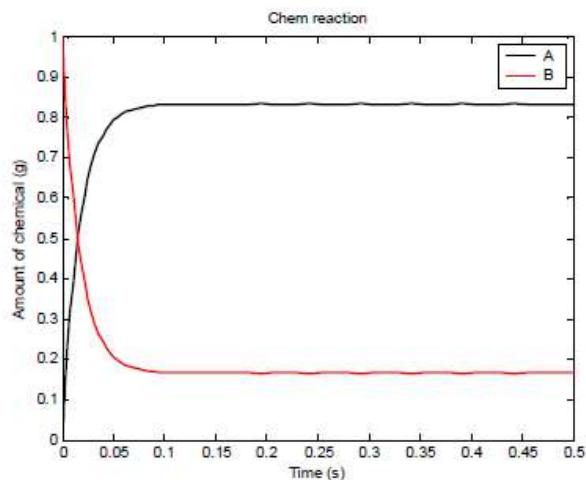
ODE Function: viewing results

- To solve and plot the ODEs on the previous slide:

```
» [t,y]=ode45('chem',[0 0.5],[0 1]);  
    ➤ assumes that only chemical B exists initially  
» plot(t,y(:,1),'k','LineWidth',1.5);  
» hold on;  
» plot(t,y(:,2),'r','LineWidth',1.5);  
» legend('A','B');  
» xlabel('Time (s)');  
» ylabel('Amount of chemical (g)');  
» title('Chem reaction');
```

ODE Function: viewing results

- The code on the previous slide produces this figure



Statistics

- Whenever analyzing data, you have to compute statistics
 - » `scores = 100*rand(1,100);`
- Built-in functions
 - mean, median, mode
- To group data into a histogram
 - » `hist(scores,5:10:95);`
 - makes a histogram with bins centered at 5, 15, 25...95
 - » `N=histc(scores,0:10:100);`
 - returns the number of occurrences between the specified bin edges 0 to <10, 10 to <20...90 to <100. you can plot these manually:
 - » `bar(0:10:100,N,'r')`

Random Numbers

- Many probabilistic processes rely on random numbers
- MATLAB contains the common distributions built in
 - » `rand`
 - draws from the uniform distribution from 0 to 1
 - » `randn`
 - draws from the standard normal distribution (Gaussian)
 - » `random`
 - can give random numbers from many more distributions
 - see **doc random** for help
 - the docs also list other specific functions
- You can also seed the random number generators
 - » `rand('state',0); rand(1); rand(1);`
`rand('state',0); rand(1);`

4
6

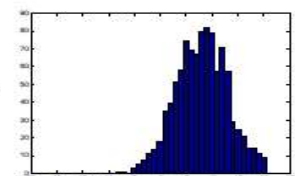
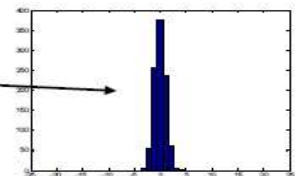
Changing Mean and Variance

- We can alter the given distributions
 - » `y=rand(1,100)*10+5;`
 - gives 100 uniformly distributed numbers between 5 and 15
 - » `y=floor(rand(1,100)*10+6);`
 - gives 100 uniformly distributed integers between 10 and 15. **floor** or **ceil** is better to use here than **round**

» `y=randn(1,1000)`

» `y2=y*5+8`

➤ increases std to 5 and makes the mean 8



Ac
Go

Symbolic Variables

- Symbolic variables are a type, like **double** or **char**
 - To make symbolic variables, use `sym`
 - » `a=sym('1/3');`
 - » `b=sym('4/5');`
 - » `mat=sym([1 2;3 4]);`
 - fractions remain as fractions
 - » `c=sym('c','positive');`
 - can add tags to narrow down scope
 - see **help sym** for a list of tags
 - Or use `syms`
 - » `syms x y real`
 - shorthand for `x=sym('x','real');` `y=sym('y','real');`
-

Cleaning up Symbolic Statements

- » `pretty(ans)` →
$$\frac{1}{9} - \frac{2}{3}c + c^2$$
 - makes it look nicer
- » `collect(3*x+4*y-1/3*x^2-x+3/2*y)`
 - collects terms →
$$\text{ans} = 2x + 11/2y - 1/3x^2$$
- » `simplify(cos(x)^2+sin(x)^2)`
 - simplifies expressions →
$$\text{ans} = 1$$
- » `subs('c^2',c,5)` →
$$\text{ans} = 25$$
 - Replaces variables with numbers or expressions. To do multiple substitutions pass a cell of variable names followed by a cell of values
- » `subs('c^2',c,x/7)` →
$$\text{ans} = \frac{1}{49}x^2$$

Ac
Go

Symbolic Expressions

- Multiply, add, divide expressions

- » `d=a*b` →
$$d = \frac{4}{15}$$
 - does $\frac{1}{3} * \frac{4}{5} = \frac{4}{15}$;
- » `expand((a-c)^2);`
 - multiplies out →
$$\text{ans} = \frac{1}{9} - \frac{2}{3}c + c^2$$
- » `factor(ans)` →
$$\text{ans} = \frac{1}{9}(3c-1)^2$$
 - factors the expression
- » `matInv=inv(mat)` →
$$\text{ans} = \begin{bmatrix} -2 & 1 \\ 3/2 & -1/2 \end{bmatrix}$$
 - Computes inverse symbolically