Assignment 1:

Chapter 1: Introduction to MATLAB

Q1: For the following program: Compound interest:

```
balance = 1000;
rate = 0.09;
interest = rate * balance;
balance = balance + interest;
disp( 'New balance:' );
disp( balance );
```

1. Run the compound interest program as it stands.
2. Change the first statement in the program to read balance = 2000;
Make sure that you understand what happens when the program runs.
3. Leave out the line
balance = balance + interest;
and rerun. Can you explain what happens?
4. Rewrite the program so that the original value of balance is *not* lost.

Q2:
2.1. Give values to variables a and b on the command line—for example, a=3 and b=5. Write statements to find the sum, difference, product, and quotient of a and b.

2.2. Given a script for animating the Mexican hat problem.

```
[x y ] = meshgrid( -8 : 0.5 : 8);
r = sqrt(x.^2 + y.^2) + eps;
z = sin(r)./ r;
mesh(z);
```

Type this into the editor, save it, and execute it.

Q3:
3.1. Enter a statement like x = [1 3 0 –1 5]
Can you see that you have created a vector (list) with five *elements*?
Enter the command disp(x) to see how MATLAB displays a vector.
Enter the command whos (or look in the Workspace browser). Under the heading Size you will see that x is 1 by 5, which means 1 row and 5 columns. You will also see that the total number of elements is 5.
3.2 You can use commas instead of spaces between vector elements if you like.
Try this: a = [5,6,7]

3.3 You can use one vector in a list for another one. Type in the following:
a = [1 2 3];
b = [4 5];
c = [a –b];
Can you work out what c will look like before displaying it?
And what about this?
a = [1 3 7];
a = [a 0 –1];

3.4 Enter the following
x = [ ]
Note in the Workspace browser that the size of x is given as 0 by 0 because x is *empty*. This means x is defined and can be used where an array is appropriate without causing an error; however, it has no size or value.

Remember the following important rules:
■ Elements in the list must be enclosed in square brackets, not parentheses.
■ Elements in the list must be separated *either by spaces or by commas*.
■ A subscript is indicated by parentheses.
■ A subscript may be a scalar or a vector.
■ In MATLAB subscripts always start at 1.
■ Fractional subscripts are not allowed.

Chapter 2:

Operators, Expressions, and Statements - Arrays, vectors and matrices

Table 2.1 Arithmetic Operations between Two Scalars

Operation	Algebraic form	MATLAB
Addition	$a+b$	a + b
Subtraction	$a-b$	a - b
Multiplication	$a \times b$	a * b
Right division	a/b	a / b
Left division	b/a	a \ b
Power	a^b	a ^ b

Table 2.2 Precedence of Arithmetic Operator

Precedence	Operator
1	Parentheses
2	Power, left to right
3	Multiplication and division, left to right
4	Addition and subtraction, left to right

Real decimal numbers (double)

1.234×10^5, -8.765×10^{-4}, 10^{-15}, -10^{12}

```
(1.234e5,   -8.765e-4,   1e-15,   -1e12)
```

Table 2.3 Arithmetic Operators for Element-by-Element Arrays

Operator	Description
.*	Multiplication
./	Right division
.\	Left division
.^	Power

To deal with arrays on an element-by-element level we need to use the following array or dot-operators:

.* , **./** and **.^**

Assignment 2:

Q1: Use MATLAB array operations to do the following:

1.1. Add 1 to each element of the vector [2 3 –1].

1.2. Multiply each element of the vector [1 4 8] by 3.
1.3. Find the array product of the two vectors [1 2 3] and [0 –1 1]. (Answer: [0 –2 3])
1.4. Square each element of the vector [2 3 1].

Q2:

2.1. Evaluate the following expressions yourself (before you use MATLAB to check). The numerical answers are in parentheses.
(a) 2 / 2 * 3 (3)
(b) 2 / 3 ^ 2 (2/9)
(c) (2 / 3) ^ 2 (4/9)
(d) 2 + 3 * 4 – 4 (10)
(e) 2 ^ 2 * 3 / 4 + 3 (6)
(f) 2 ^ (2 * 3) / (4 + 3) (64/7)
(g) 2 * 3 + 4 (10)
(h) 2 ^ 3 ^ 2 (64)
(i) –4 ^ 2 (–16; ^ has higher precedence than –)

2.2. Use MATLAB to evaluate the following expressions. The answers are in parentheses.
(a) $\sqrt{2}$ (1.4142; use sqrt or ^0.5)
(b) $\frac{3+4}{5+6}$ (0.6364; use brackets)
(c) Find the sum of 5 and 3 divided by their product (0.5333)
(d) 2^{3^2} (512)
(e) Find the square of 2π (39.4784; use pi)
(f) $2\pi^2$ (19.7392)

2.3. Try to avoid using unnecessary brackets in an expression. Can you spot the errors in the following expression? (Test your corrected version with MATLAB.)
(2(3+4)/(5*(6+1))^2

Note that the MATLAB Editor has two useful ways of dealing with the problem of "unbalanced delimiters" (which you should know about if you have been working through Help!):
■ When you type a closing delimiter, that is, a right),], or }, its matching opening delimiter is briefly highlighted. So if you don't see the highlighting when you type a right delimiter, you immediately know you've got one too many.
■ When you position the cursor anywhere inside a pair of delimiters and select **Text→Balance Delimiters** (or press **Ctrl**+**B**), the characters inside the delimiters are highlighted.

Q3:

3.1 Set up a vector n with elements 1, 2, 3, 4, 5. Use MATLAB array operations on it to set up the following four vectors, each with five elements:

(a) 2, 4, 6, 8, 10
(b) 1/2, 1, 3/2, 2, 5/2
(c) 1, 1/2, 1/3, 1/4, 1/5
(d) 1, 1/22, 1/32, 1/42, 1/52

3.2. Suppose vectors a and b are defined as follows:

a = [2 –1 5 0];
b = [3 2 –1 4];

Evaluate by hand the vector c in the following statements. Check your answers with MATLAB.

(a) c = a – b;
(b) c = b + a – 3;
(c) c = 2 * a + a .^ b;
(d) c = b ./ a;
(e) c = b . a;
(f) c = a .^ b;
(g) c = 2.^b+a;
(h) c = 2*b/3.*a;
(i) c = b*2.*a;

3.3. Water freezes at 32° and boils at 212° on the Fahrenheit scale. If *C* and *F* are Celsius and Fahrenheit temperatures, the formula

$$F = 9/5C + 32$$

converts from one to the other. Use the MATLAB command line to convert Celsius 37° (normal human temperature) to Fahrenheit (98.6°).

```
C = 37;
F = C*9/5 + 32;
disp(F);
```

Assignment 3:

Chapter 2: Arrays: vectors and matrices - Repeating with for loop

Q1: Vertical motion under gravity problem:

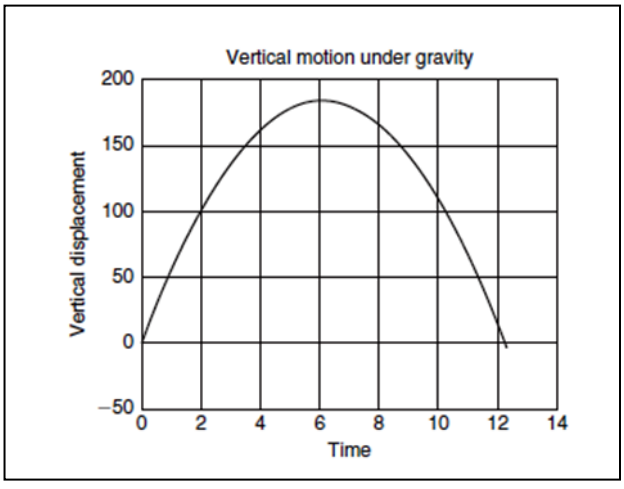
If a stone is thrown vertically upward with an initial speed u, its vertical displacement s after an elapsed time t is given by the formula $s = ut - gt^2/2$, where g is the acceleration due to gravity. Air resistance is ignored. We would like to compute the value of s over a period of about 12.3 seconds at intervals of 0.1 seconds, and plot the distance–time graph over this period.

Solution:

The structure plan for this problem is as follows:

1. Assign the data (g, u, and t) to MATLAB variables
2. Calculate the value of s according to the formula
3. Plot the graph of s against t
4. Stop

```
% Vertical motion under gravity
g = 9.81; % acceleration due to gravity
u = 60; % initial velocity in m/sec
t = 0 : 0.1 : 12.3; % time in seconds
s = u * t – g / 2 * t .^ 2; % vertical displacement in meters
plot(t, s), title( ‘Vertical motion under gravity’ )
xlabel( ‘time’ ), ylabel( ‘vertical displacement’ )
grid
disp( [t’ s’] ) % display a table
```



Q2- Linear equations

Solve a system of linear equations:

$$x + 2y = 4$$
$$2x - y = 3$$

1- Using matrix notation:

```
a = [1 2; 2 –1];
b = [4; 3];
x = a\b
```

2- Using solve command:

```
[x,y] = solve('x+2*y=4','2*x–y=3')
whos
x = double(x), y=double(y)
whos
```

Q3: OUTPUT

3.1. The disp statement:

General form of *disp* for a numerical variable is:

disp (variable)

To display a message and a value on the same line, use the following trick:

```
x = 2;
disp( ['The answer is ', num2str(x)] );
```

The output should be

The answer is 2

3.2. The format command:

Enter the following commands (MATLAB's response is also shown).

```
≫ format compact
≫ x = [1e3 1 1e-4]
 x =
  1.0e+003 *
    1.0000    0.0010    0.0000
```

```
≫format bank
≫x
x =
         1000.00            1.00            0.00
≫format short e
≫x
x =
  1.0000e+003    1.0000e+000    1.0000e-004
```

3.3 fprintf statement:

Used to exactly control the output format. For example:

* fprintf('%s %4.0f\n', name, mark)

The basic for construct

for *index* = *j*:*k* *statements* end	for *index* = *j*:*m*:*k* *statements* end

If the *for* construct has the form:
for k = first : increment : last
The number of times the loop is executed may be calculated from the following equation:

$$\texttt{floor}\left(\frac{\texttt{last} - \texttt{first}}{\texttt{increment}}\right) + 1$$

where the MATLAB function floor(x) rounds x *down* toward $-\infty$.
This value is called the *iteration* or *trip count*.

Q4: Factorials!

Run the following program to generate a list of *n* and *n*! ("*n* factorial," or "*n* shriek"), where

$$n! = 1 \times 2 \times 3 \times \ldots \times (n-1) \times n$$

```
n = 10;
fact = 1;
for k = 1:n
fact = k * fact;
disp( [k fact] )
end
```

Experiment to find the largest value of *n* for which MATLAB can find the *n* factorial.

Q5: Series with alternating sign:

Find the sum of the first 99 terms of the following series with a for loop:

$$\ln(2) = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \cdots$$

```
sign = -1;
s = 0;
for n = 1:99
sign = -sign;
s = s + sign / n;
end
```

Homework:

Write MATLAB programs to find the following sums with for loops and by vectorization.

- $1^2 + 2^2 + 3^2 + \cdots + 1000^2$
- $1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots - \frac{1}{1003}$
- Sum the left-hand side of the series:

  $\frac{1}{1^2-3^2} + \frac{1}{3^2-5^2} + \frac{1}{5^2-7^2} + \cdots = \frac{\pi^2-8}{16}$ (sum is 0.1169 - with 500 terms)

Solution:

- $1^2 + 2^2 + 3^2 + \cdots + 1000^2$ (sum is 333 833 500)

```
s = 0;                          % with for loop
for i = 1:1000
s = s + i^2;
end
disp(s);

a = 1:1000;                     % by vectorization
s = sum(a.^2);
```

- $1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots - \frac{1}{1003}$ (0.7849)

```
sign = -1;                      % with for loop
s = 0;
for i = 1:2:1003
 sign = -sign;
 s = s + sign/i;
end
disp(s);

a = 1:4:1001;                   % by vectorization
b = 3:4:1003;
s = sum(a.\1) - sum(b.\1);
```

- $\frac{1}{1^2-3^2} + \frac{1}{3^2-5^2} + \frac{1}{5^2-7^2} + \cdots = \frac{\pi^2-8}{16}$ (-0.1169 with 500 terms)

```
s = 0;                          % with for loop
for i = 1:2:501
 s = s + 1/(i^2-(i+2)^2);
end
disp(s);
```

Assignment 4:

Chapter 2

Decisions – Structure plan

If statement

if *condition statement*, end

if *condition statementA*, else *statementB*, end

if *condition1*
statementsA
elseif *condition2*
statementsB
elseif *condition3*
statementsC
...
else
statementsE
end

Table 2.4 Relational Operators

Operator	Meaning
<	Less than
<=	Less than or equal
==	Equal
~=	Not equal to
>	Greater than
>=	Greater than or equal

Logical operators
More complicated logical expressions can be constructed using the three *logical operators*:
& (and), | (or), and ~ (not).

Q1: The quadratic equation

$$ax^2 + bx + c = 0$$

```
if (b ^ 2 - 4*a*c == 0) & (a ~= 0)
x = –b / (2*a);
end

d = b^ 2 – 4*a*c;
if a ~= 0
if d < 0
disp( 'Complex roots' )
else
x1 = (–b + sqrt( d )) / (2*a);
x2 = (–b – sqrt( d )) / (2*a);
end % first end «««««««
end
```

Q2: switch – case statement:

```
d = floor(3*rand) + 1
switch d
case 1
disp( 'That''s a 1!' );
case 2
disp( 'That''s a 2!' );
otherwise
disp( 'Must be 3!' );
end
```

Multiple expressions can be handled in a single case statement by enclosing the case expression in a cell array

```
d = floor(10*rand);
switch d
case {2, 4, 6, 8}
disp( 'Even' );
case {1, 3, 5, 7, 9}
disp( 'Odd' );
otherwise
disp( 'Zero' );
end
```

Chapter 3: Program Design and Algorithm Development: Review of basic elements:

- Input
  Initialize, define, *request*, or assign numerical values to variables.
- Set of commands
  Operations applied to input variables that lead to the desired result.
- Output
  Display (graphically or numerically) result.

Structure plan examples:

1- Write a script to convert a temperature on the Fahrenheit scale (where water freezes and boils at 32° and 212°, respectively) to the Celsius scale.

1. Initialize Fahrenheit temperature
2. Calculate and display Celsius temperature
3. Stop

```
%
F = input('Temperature in degrees F: ');
C = (F-32)*5/9;
disp([ 'Temperature in degrees C = ',num2str(C)])
% STOP
```

2- Quadratic Equation: $ax^2 + bx + c = 0$

A structure plan of the *complete* algorithm for finding the solution(s) x, given any values of a, b, and c

1. Start
2. Input data (a, b, c)
3. If $a = 0$ then
   If $b = 0$ then
   If $c = 0$ then
   Display 'Solution indeterminate'
   else
   Display 'There is no solution'
   else
   $x = -c/b$
   Display x (only one root: equation is linear)
   else if $b^2 < 4ac$ then
   Display 'Complex roots'
   else if $b^2 = 4ac$ then
   $x = -b/(2a)$
   Display x (equal roots)
   else
   $x_1 = (-b + \sqrt{b^2 - 4ac})/(2a)$
   $x_2 = (-b - \sqrt{b^2 - 4ac})/(2a)$
   Display x_1, x_2
4. Stop

1. Input the data
2. Find and display the solution(s)
3. Stop

```
% Step 1: Start
format compact
disp(' ')
disp(' Quadratic roots finder ')
disp(' ')
% Step 2: Input data
A = input(' Specify coefficients as follows: [a, b, c] = ');
a = A(1);b = A(2);c = A(3);
% Step 3: Evaluation of roots
if a==0 & b==0 & c==0
   disp(' Solution is indeterminate ')
elseif a==0 & b==0
   disp(' There is no solution ')
elseif a==0
   x = -c/b
   disp(' Only one root: equation is linear.')
elseif b^2 - 4*a*c < 0
   disp(' Complex roots')
elseif b^2 - 4*a*c == 0
   x = -b/(2*a)
   disp(' Equal roots')
elseif b^2 - 4*a*c > 0
    x1 = (-b + sqrt(b.^2-4.*a.*c) )./(2.*a);
    x2 = (-b - sqrt(b.^2-4.*a.*c) )./(2.*a);
    disp(' x1,x2 the two distinct roots')
    disp([x1 x2])
end
format
% Step 4: Stop
```

Chapter 4: MATLAB Functions and Data Import–Export Utilities:

1- Common functions:

`cos(x)`	cosine of `x`.
`cosh(x)`	hyperbolic cosine of `x`, or $\frac{e^x + e^{-x}}{2}$ (see Figure 4.1).
`cot(x)`	cotangent of `x`.
`csc(x)`	cosecant of `x`.
`cumsum(x)`	cumulative sum of the elements of `x` (e.g., `cumsum(1:4)` returns `[1 3 6 10]`).
`date`	date in a string in dd-mmm-yyyy format (e.g., `02-Feb-2001`).
`exp(x)`	value of the exponential function e^x (see Figure 4.1).
`fix(x)`	rounds to the nearest integer toward zero (e.g., `fix(-3.9)` returns `-3`; `fix(3.9)` returns `3`).
`floor(x)`	largest integer not exceeding `x`—that is, rounds down to nearest integer (e.g., `floor(-3.9)` returns `-4`; `floor(3.9)` returns `3`).
`length(x)`	number of elements in vector `x`.
`log(x)`	natural logarithm of `x`.
`log10(x)`	base 10 logarithm of `x`.
`max(x)`	maximum element of vector `x`.
`mean(x)`	mean value of elements in vector `x`.
`min(x)`	minimum element in vector `x`.
`pow2(x)`	2^x.
`prod(x)`	product of the elements of `x`.
`rand`	pseudo-random number in the interval [0, 1).

Note that if the argument of a function is an array, the function is applied element by element to all the values in the array.
For example, sqrt([1 2 3 4])
returns 1.0000 1.4142 1.7321 2.0000

To examine these functions plot them.
x = –1:.1:1; <Enter> plot(x,abs(x),'o') <Enter> % abs(x) absolute value of x.

2- Importing and Exporting data: The **load** and **save** commands

To export (save) the array in "delimited" ASCII format in the file myData.txt,

```
A = 1   2   3
    4   5   6
```

use the command

save myData.txt A –ascii

If you view myData.txt in a text editor (or type it in the Command Window) it looks like this:

```
1.0000000e+000 2.0000000e+000 3.0000000e+000
4.0000000e+000 5.0000000e+000 6.0000000e+000
```

Importing text (ASCII) data

A = load('myData.txt')

Chapter 5: Logical Vectors:

Examples:
- If you enter

```
r = 1:5;
r <= 3
```

the logical expressionr <= 3(where r is a vector) returns a *vector*
1 1 1 0 0

- To compare vectors with vectors in logical expressions:

```
a = 1:5;
b = [0 2 3 5 6];
a == b        % no semi-colon!
```

The logical expression a == b returns the logical vector
0 1 1 0 0
because it is evaluated element by element; that is, a(1) is compared with b(1), a(2) with b(2), and so forth.

1- Logical Operators:

Table 5.1 Logical Operators

Operator	Meaning
~	NOT
&	AND
|	OR

Table 5.2 Truth Table

lex1	lex2	~lex1	lex1 & lex2	lex1 | lex2	xor(lex1, lex2)
F	F	T	F	F	F
F	T	T	F	T	T
T	F	F	F	T	T
T	T	F	T	T	F

T = true; F = false

Example,
~0 & 0 returns 0 (false), whereas
~(0 & 0) returns 1 (true).

Some more examples:
(b * (b == 4) * a * c) & (a ~= 0)
(final >= 60) & (final < 70)
(a ~= 0) | (b ~= 0) | (c != 0)
~((a == 0) & (b == 0) & (c == 0))

Note: The expression 0 < r < 1 is coded as (0 < r) & (r < 1)

Table 5.3 Operator Precedence

Precedence	Operator
1	( )
2	^ .^ ' .' (pure transpose)
3	+ (unary plus) – (unary minus) ~ (NOT)
4	* / \ .* ./ .\
5	+ (addition) – (subtraction)
6	:
7	> < >= <= == ~=
8	& (AND)
9	| (OR)

Exercise:

Work out the results of the following expressions before checking them at the command line:

a = [–1 0 3];
b = [0 3 1];

~a	ans=	0	1	0
a & b	ans=	0	0	1
a | b	ans=	1	1	1
xor(a, b)	ans=	1	1	0
a > 0 & b > 0	ans=	0	0	1
a > 0 | b > 0	ans=	0	1	1
~a > 0	ans=	0	1	0
a + (~b)	ans=	0	0	3
a > ~b	ans=	0	0	1
~ a > b	ans=	0	0	0
~ (a > b)	ans=	1	1	0

2- Subscripting with logical vectors

Examples:

* a = [–2 0 1 5 9];
  a([5 1 3]) returns 9 –2 1

* a(logical([0 1 0 1 0])) returns 0 5

* a = a(a > 0) removes all the nonpositive elements from a because a > 0 returns the logical vector [0 0 1 1 1].
* islogical(a > 0) returns 1, because a>0 is a logical vector

3- Logical Functions:

any(x) returns the scalar 1 (true) if *any* element of x is nonzero (true).

all(x) returns the scalar 1 if *all* elements of x are nonzero.

* Because any and all with vector arguments return *scalar*s, they are particularly useful in if statements.
* if all(a >= 1)
  do something
  end

  if any(a ~= b)
  do something
  end

exist('a') returns 1 if a is a workspace variable.

find(x) returns a vector containing the subscripts of the *nonzero* (true) elements of x.

* a = a(find(a)) removes all zero elements from a.
* x = [8 1 –4 8 6];
  find(x >= max(x)) returns the vector [1 4], which comprises the subscripts of the largest element.

isempty(x) returns 1 if x is an empty array and 0 otherwise

isinf(x) returns 1s for the elements of x that are +Inf or –Inf, and 0's otherwise.

isnan(x) returns 1s where the elements of x are NaN, and 0's otherwise.

ischar() returns 1 if () contains character data

isnumeric() returns 1 if () contains numeric data

Exercise:

Table 5.4 Income Tax Calculation

Taxable Income	Tax Payable
$10,000 or less	10% of taxable income
$10,000 to $20,000	$1000 + 20% of amount by which taxable income exceeds $10,000
More than $20,000	$3000 + 50% of amount by which taxable income exceeds $20,000

```
% Income tax the old-fashioned way
inc = [5000 10000 15000 30000 50000];
for ti = inc
  if ti < 10000
      tax = 0.1 * ti;
  elseif ti < 20000
      tax = 1000 + 0.2 * (ti – 10000);
  else
      tax = 3000 + 0.5 * (ti – 20000);
  end;
 disp( [ti tax] )
end;
```

Now, using the logical way:

```
% Income tax the logical way
inc = [5000 10000 15000 30000 50000];
tax = 0.1 * inc .* (inc <= 10000);
tax = tax + (inc > 10000 & inc <= 20000) .* (0.2 * (inc–10000) + 1000);
tax = tax + (inc > 20000) .* (0.5 * (inc–20000) + 3000);
disp( [inc' tax'] );
```

```
Taxable income      Income tax
5000.00             500.00
10000.00            1000.00
15000.00            2000.00
30000.00            8000.00
50000.00            18000.00
```

Examples:

1- Discontinuous graphs

One very useful application of logical vectors is in plotting discontinuities. The following script plots the graph, defined by:

$$y(x) = \begin{cases} \sin(x) & (\sin(x) > 0) \\ 0 & (\sin(x) \leq 0) \end{cases}$$

over the range 0 to 3π:

```
x = 0 : pi/20 : 3 * pi;
y = sin(x);
y = y .* (y > 0); % set negative values of sin(x) to zero
plot(x, y)
```

2- Avoiding division by zero

Suppose you want to plot the graph of sin*(x)/x* over the range −4π to 4π. The most convenient way to set up a vector of the *x* coordinates is

```
x = –4*pi : pi / 20 : 4*pi;
```

But then, when you try

```
y = sin(x) ./ x;
```

you get the Divide by zero warning because one of the elements of x is exactly zero. The following script plots the graph correctly—without a missing segment at *x* = 0.

```
x = –4*pi : pi/20 : 4*pi;
x = x + (x == 0)*eps;          % adjust x = 0 to x = eps
y = sin(x) ./ x;
plot(x, y)
```

When x has the value eps, the value of sin(eps)/eps has the correct limiting value of 1 (check it) instead of NaN (Not-a-Number) resulting from a division by zero.

3- Counting random numbers

The function ***rand*** returns a (pseudo-) random number in the interval [0, 1); rand(1, n) returns a row vector of n such numbers.
Work out the following problem on the command line:

1. Set up a vector r with seven random elements

```
r = rand(1,7) % no semi-colon
```

2. Check that the logical expression r < 0.5 gives the correct logical vector. Using the function sum on the logical expression r < 0.5 will effectively count how many elements of r are less than 0.5. Try it and check your answer against the values displayed for r:

sum(r < 0.5)

4- Rolling dice

When a fair die is rolled, the number uppermost is equally likely to be any integer from 1 to 6. Thus, if rand is a random number in the range [0, 1), 6 * rand will be in the range 0, 6) and 6 * rand + 1 will be in the range [1, 7), that is, between 1 and 6.9999. Discarding the decimal part of this expression with floor gives an integer in the required range.

Try the following:
Generate a vector d of 20 random integers in the range 1 to 6: Count the number of "sixes" thrown. Verify your result by displaying d. Estimate the probability of throwing a six by dividing the number of sixes thrown by 20.
Repeat with more random numbers in the vector d. The more you have, the closer the proportion of sixes gets to the theoretical expected value of 0.1667 (i.e., 1/6).

```
d = floor(6 * rand(1, 2000)) + 1  % Generate a random vector d in the range 1 to 6
n = sum( d == 6)  % Count the number of "sixes" thrown
p = n/2000  % Estimate the probability of throwing a six
```

Homework

5.1. Determine the values of the following expressions yourself before checking your answers using MATLAB. You may need to consult Table 5.3.
(a) 1 & –1
(b) 13 & ˜(–6)
(c) 0 < –2|0
(d) ˜[1 0 2] * 3
(e) 0 <= 0.2 <= 0.4
(f) 5 > 4 > 3
(g) 2 > 3 & 1

5.2. Given that a = [1 0 2] and b = [0 2 2], determine the values of the following expressions. Check your answers with MATLAB.
(a) a ˜= b
(b) a < b
(c) a < b < a
(d) a < b < b
(e) a | (˜a)
(f) b & (˜b)
(g) a|(˜(˜b))

Chapter 6: Matrices of numbers and arrays of strings:

Learning objectives:

- **Creating and manipulating matrices**
- **Introduction to matrix operations**
- **Introduction to character strings and facilities for handling them**

6.1 MATRICES

Example: Transportation problem:

Transportation costs

	D1	D2	D3
S1	3	12	10
S2	17	18	35
S3	7	10	24

Supply-Demand

	D1	D2	D3
S1	4	0	0
S2	6	6	0
S3	0	3	5

To calculate the total transportation in MATLAB, enter **C** and **X** as matrices from the commands:

```
C = [3 12 10; 17 18 35; 7 10 24];
X = [4 0 0; 6 6 0; 0 3 5];
```

and then find the *array* product of **C** and **X**:

```
total = C .* X
```

which gives

```
12   0    0
102  108  0
 0   30   120
```

The command

```
sum(total)
```

then returns a vector where each element is the sum of each column of total:

```
114  138  120
```

Summing this in turn—that is,

```
sum(sum( total ))
```

—gives the final answer of 372.

Creating matrices

Bigger matrices can be constructed from smaller ones; for example, the statements

```
a = [1 2; 3 4];
x = [5 6];
a = [a; x]
```

```
a =   1   2
      3   4
      5   6
```

Subscripting:

For the above example:

```
a(3,2)    returns  6          a(5)    returns  4
a(3,3) = 7;
```

will add a third column to a with 0s everywhere except at a(3,3).

The transpose operator (')

```
a = [1:3; 4:6]
b = a'
```

result in

```
a =
     1   2   3
     4   5   6
b =
     1   4
     2   5
     3   6
```

The colon operator (:)

For example, if a is the matrix

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

the statement

```
a(2:3,1:2)
```

results in

```
     4   5
     7   8
```

The statement

```
a(3,:)
```

results in

```
     7   8   9
```

Finally, the statement

```
a(1:2,2:3) = ones(2)
```

results in

```
a =
     1   1   1
     4   1   1
     7   8   9
```

You can use vector subscripts to get more complicated effects.
For example,

```
a(:,[1 3]) = b(:,[4 2])
```

replaces the first and third columns of a by the fourth and second columns of b

Example: Elementary row operations in Gauss reduction.

For example, if a is the matrix

```
a =
     1   –1    2
     2    1   –1
     3    0    1
```

the statement

```
a(2,:) = a(2,:) – a(2,1)*a(1,:)
```

subtracts the first row multiplied by the first element in the second row from the second row, resulting in

```
a =
     1   –1    2
     0    3   –5
     3    0    1
```

The keyword *end* refers to the *last* row or column of an array. For example, if r is a row vector, the statement

```
sum(r(3:end))
```

returns the sum of all the elements of r from the third one to the last one.

The colon operator may also be used as a single subscript, on the right-hand side, a(:) gives all the elements in one long column vector. Thus, if

```
a =
     1   2
     3   4
```

the statement

```
b = a(:)
```

results in

```
b =
     1
     3
     2
     4
```

However, on the left-hand side of an assignment, a(:) *reshapes* a matrix that already exists, i.e., the matrix on the right-hand side is reshaped into the shape of a on the left-hand side. If

```
b =
     1   2   3
     4   5   6
```

and

```
  a =
       0   0
       0   0
       0   0
```

the statement

```
a(:) = b
```

results in

```
   a =
       1   5
       4   3
       2   6
```

```
a(:) = 1:6
```

(with a as above) results in

```
   a =
       1   4
       2   5
       3   6
```

```
a(:) = –1
```

replaces *all* the elements of a matrix with a scalar:

Deleting rows and columns

Use the colon operator and the empty array to delete entire rows or columns.

For example,

```
a(:,2) = [ ]
```

deletes the second column of a.

However, using single-subscript notation you can delete a sequence of elements and reshape the remaining elements into a row vector.

You can use logical vectors to extract a selection of rows or columns from a matrix, if

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

the statement

```
a(:, logical([1 0 1]))
```

results in

```
    ans =
        1   3
        4   6
        7   9
```

(i.e., first and third columns extracted).

The same effect is achieved with a(:, [1 3])

Using MATLAB functions with matrices

Some MATLAB functions operate on matrices *column by column*, so if

```
a =
     1   0   1
     1   1   1
     0   0   1
```

the statement

```
all(a)
```

results in

```
ans =
     0   0   1
```

However, the statement

```
any(a)
```

returns

```
ans =
     1   1   1
```

Manipulating matrices

Here are some functions for manipulating matrices (See Help for details):

- diag extracts or creates a diagonal.
- fliplr flips from left to right.
- flipud flips from top to bottom.
- rot90 rotates.
- tril extracts the lower triangular part
- triu extracts the upper triangular part.

Array (element-by-element) operations on matrices

All of the array operations discussed in Chapter 2 apply to matrices as well as vectors. For example, if a is a matrix, a * 2 multiplies each of its elements by 2, and if

```
a =
     1   2   3
     4   5   6
```

the statement `a .^ 2` results in

```
ans =
     1    4    9
    16   25   36
```

Matrices and for

In the most general form of the for statement, if

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

the statements

```
for v = a
disp(v')
end
```

result in

```
     1    4    7
     2    5    8
     3    6    9
```

The index v takes on the value of each *column* of the matrix a in turn. This provides a way of processing all the columns of a matrix. You can do the same with the rows if you transpose a, so, for example, the statements

```
for v = a'
disp(v')
end
```

display the rows of a one at a time.

6.2 MATRIX OPERATIONS

Multiplication

The * operator is used for matrix multiplication. For example, if

```
a =
     1     2
     3     4
```

and

```
b =
     5     6
     0    –1
```

the statement

```
c = a * b
```

results in

```
c =
     5     4
    15    14
```

Exponentiation

The matrix operation $\mathbf{A}^2$ means $\mathbf{A} \times \mathbf{A}$, where **A** must be a square matrix. The operator ^ is used for matrix exponentiation, so, for example, if

```
a =
     1     2
     3     4
```

the statement

```
a ^ 2
```

results in

```
ans =
     7    10
    15    22
```

6.3 OTHER MATRIX FUNCTIONS

det determinant.

eig eigenvalue decomposition.

expm matrix exponential (i.e., $e^{\mathbf{A}}$, where **A** is a matrix). The matrix exponential may be used to evaluate analytical solutions of linear ordinary differential equations with constant coefficients.

inv inverse.

6.4 STRINGS

Strings may be assigned to variables by enclosing them in apostrophes:

```
s = 'Hi there';
```

If an apostrophe is part of the string, it must be repeated:

```
s = 'o''clock';
```

Input

Strings may be entered in response to the input statement in two ways:

- Enclose the string in apostrophes when you enter it.
- Use an additional argument 's' with input

```
name = input( 'Enter your surname: ', 's' );
```

Strings as arrays

A MATLAB string is actually an array, with each element representing one character. For example, if

```
s = 'Napoleon'
```

whos reveals that s is 1 by 8. The statement

```
s(8:-1:1)
```

will therefore display the string Napoleon backwards.

The following code will remove all the blanks from the string s.

```
s = 'Twas brillig and the slithy toves';
nonblanks = s ~= ' ';
s(nonblanks)
```

nonblanks is a logical vector with 0s corresponding to the positions of the blanks. Using nonblanks as a subscript removes the blanks.

String concatenation

Because strings are vectors, they may be *concatenated* (joined) with square brackets:

```
king = 'Henry';
king = [king, ' VIII']
```

ASCII codes: double and char

A MATLAB character is represented internally by a 16-bit numerical value. For example, the ASCII codes for the letters A through Z are the consecutive integers from 65 to 90, while the codes for a through z run from 97 to 122.

You can see that the ASCII codes for a string with double:

```
double('Napoleon')
```

return

78 97 112 111 108 101 111 110

Conversely, using char:

```
char(65:70)
```

returns a string from the ASCII codes

ABCDEF

String display with fprintf

Strings may be displayed with fprintf using the %s format specifier. They can be right- or left-justified and truncated. The statements

```
fprintf( '%8sVIII\n', 'Henry' )
fprintf( '%-8sVIII\n', 'Henry' )
fprintf( '%.3sVIII\n', 'Henry' )
```

result in

HenryVIII
HenryVIII
HenVIII

To display single characters with fprintf, use the %c format specifier.

Comparing strings

Try out the following statements:

s1 = 'ann';

s2 = 'ban';

s1 < s2

You should get the logical vector

1 0 0

because the logical expression s1 < s2 compares the ASCII codes of the two strings element by element (the strings must have the same length). This is the basis of alphabetical sorting.

The function *strcmp(s1, s2)* compares the two strings s1 and s2, returning 1 if they are identical and 0 otherwise. (The strings do not have to be the same length.)

Other string functions

blanks generates a string of blanks.

int2str, num2str convert their numeric arguments to strings.

ischar returns 1 if its argument is a string and 0 otherwise.

lower**,** upper convert strings to lowercase and uppercase, respectively.

sprintf works like fprintf, except that it returns a string according to the format specification:

s = sprintf('Value of K: %g', K)

returns the string Value of K: 0.015 in s if K has the value 0.015.

6.5 TWO-DIMENSIONAL STRINGS

For example,

nameAndAddress = ['Adam B Carr '; '21 Barkly Ave'; 'Discovery ']

results in

nameAndAddress =

Adam B Carr

21 Barkly Ave

Discovery

An easier way to create two-dimensional strings is to use the char function:

nameAndAddress = char('Adam B Carr', '21 Barkly Ave', 'Discovery')

Exercises:

6.1. Set up any 3 × 3 matrix a. Write some command-line statements to perform the following operations:

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

results in

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

(a) Interchange columns 2 and 3.

```
a(:,[2 3]) = a(:,[3 2])
```

results in

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

(b) Add a fourth column (of 0s).

```
a(3,4) = 0
```

results in

```
a =
     1   3   2   0
     4   6   5   0
     7   9   8   0
```

(c) Insert a row of 1s as the new second row (i.e., move the current second and third rows down).

```
a([3 4],:) = a([2 3],:);
a(2,:) = ones
```

results in

```
a =
     1   3   2   0
     1   1   1   1
     4   6   5   0
     7   9   8   0
```

(d) Remove the second column.

```
a(:,2) = [ ]
```

results in

```
a =
     1   2   0
     1   1   1
     4   5   0
     7   8   0
```

6.2. Write a function deblank(s) that removes all blanks from the string s.

```
>> s = 'I am a student in King Saud University';
>> deblanks = s ~= ' ';
>> s(deblanks)
```

6.3. Write a function toupper(s) that translates lowercase letters in the string s to uppercase.

```
>> upper(s)
```

Simulation:

RANDOM NUMBER GENERATION

Random events are easily simulated in MATLAB with the function *rand*.

Example: FLIPPING COINS

When a fair (unbiased) coin is flipped, the probability of getting heads or tails is 0.5 (50%). Since a value returned by rand is equally likely to be anywhere in the interval [0, 1), we can represent heads with a value less than 0.5 and tails otherwise.

Suppose an experiment calls for a coin to be flipped 50 times and the results to be recorded. In real life you may need to repeat such an experiment a number of times; this is where computer simulation is handy.

The following script simulates flipping a coin 50 times:

```
for i = 1:50
   r = rand;
   if r < 0.5
     fprintf( 'H' )
   else
    fprintf( 'T' )
   end
end
fprintf( '\n' ) % newline
```

Here is the output from two sample runs:
THHTTHHHHTTTTTHTHTTTHHTHTTTHHTTTTHTTHHTHHHHHHTTHTT
THTHHHTHTHHTTHTHTTTHHTTTTTTTHHHTTTHTHHTHHHHTTHTHTT

ROLLING DICE

When a fair die is rolled, the number uppermost is equally likely to be any integer from 1 to 6. The following statement generates a vector with 10 random integers in the range 1–6:
d = floor(6 * rand(1,10) + 1)
Here are the results of two such simulations:
2 1 5 5 6 3 4 5 1 1
4 5 1 3 1 3 5 4 6 6

We can do statistics on our simulated experiment, just as if it were a real one. For example, we can estimate the mean of the number obtained when the die is rolled 100 times and the probability of getting a 6.

Chapter 7: Introduction to Graphics:

BASIC TWO-DIMENSIONAL GRAPHS

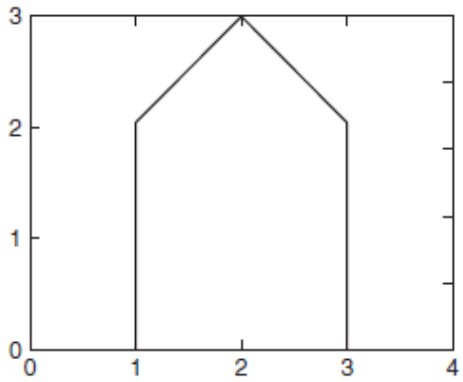
The most common form of plot is **plot(x, y)**, where x and y are vectors of the same length.
Another common form of plot function us:
x = 0:pi/40:4*pi;
plot(x, sin(x))

The easy-to-use form of plot is ezplot:
ezplot('tan(x)')

EXERCISES

1. Draw lines joining the following points: (0, 1), (4, 3), (2, 0), and *(*5,−2*)*.
plot([0 4 2 5], [1 3 0 -2])

2. Draw a "house" similar to the one depicted in Figure.
plot([0 1 1 2 3 3 4], [0 0 2 3 2 0 0])



Labels:

grid adds/removes grid lines to/from the current graph.
text(x, y, 'text') writes text in the graphics window at the point specified by x and y.
title('text') writes the text as a title at the top of the graph.
xlabel('horizontal') labels the *x*-axis.
ylabel('vertical') labels the *y*-axis.

Multiple plots on the same axes

Use plot with multiple arguments. For example, **plot(x1, y1, x2, y2, x3, y3, ...)** plots the (vector) pairs (x1, y1), (x2, y2), and so on. The advantage of this method is that the vector pairs may have different lengths. MATLAB automatically selects a different color for each pair.

If you are plotting two graphs on the same axes, you may find plotyy useful—it allows you to have independent *y*-axis labels on the left and the right:
plotyy(x,sin(x), x, 10*cos(x))

Line styles, markers, and color

For example,
plot(x, y, '--') joins the plotted points with dashed lines, whereas **plot(x, y, 'o')** draws circles at the data points with no lines joining them.
You can specify all three properties:
plot(x,sin(x), x, cos(x), 'om--')
which plots sin*(x)* in the default style and color and cos*(x)* with circles joined by dashes in magenta. The available colors are denoted by the symbols c, m, y, k, r, g, b, w.

Axis limits

Whenever you draw a graph, MATLAB automatically scales the axis limits to fit the data. You can override this with
axis([xmin, xmax, ymin, ymax]) which sets the scaling on the *current* plot.

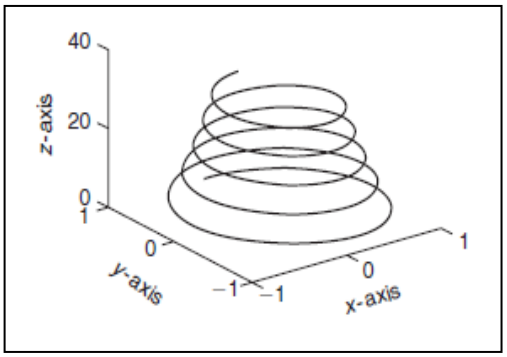
THREE-DIMENSIONAL PLOTS

The plot3 function

The function plot3 is the 3D version of plot. The command
plot3(x, y, z)
draws a 2D projection of a line in 3D through the points whose coordinates are the elements of the vectors x, y, and z

Example:

```
t = 0:pi/50:10*pi;
plot3(exp(–0.02*t).*sin(t), exp(–0.02*t).*cos(t),t)
xlabel('x-axis'), ylabel('y-axis'), zlabel('z-axis')
```



Chapter 8: LOOPS:

DETERMINATE REPETITION WITH for

Example1: Binomial coefficient

```
ncr = 1;
n = 10;
r = 3;
for k=1:r
ncr = ncr*(n-k+1)/k;
end
disp (ncr)
```

120

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

$$\binom{n}{r} = \frac{n(n-1)(n-2)\cdots(n-r+1)}{r!}$$

$$\text{e.g., } \binom{10}{3} = \frac{10!}{3! \times 7!} = \frac{10 \times 9 \times 8}{1 \times 2 \times 3}$$

INDETERMINATE REPETITION WITH while

The while statement

In general the while statement looks like this:

while *condition*
statements
end

Example: A guessing game

MATLAB generates one at random an integer between 1 and 10. You have to guess it. If your guess is too high or too low, the script must say so. If your guess is correct, a message of congratulations must be displayed.

A structure plan might be helpful:

1. Generate random integer
2. Ask user for guess
3. While guess is wrong:
   If guess is too low Tell her it is too low
   Otherwise Tell her it is too high
   Ask user for new guess
4. Polite congratulations
5. Stop

Here is the script:

```
num = floor(10 * rand + 1);
guess = input( 'Your guess please: ' );
load splat
while guess ~= num
sound(y, Fs)
if guess > num
```

```
disp( 'Too high' )
else
disp( 'Too low' )
end;
guess = input( 'Your next guess please: ' );
end
disp( 'At last!' )
load handel
sound(y, Fs) % hallelujah!
```

Try it out a few times. Note that the while loop repeats as long as num is not equal to guess.