

MATLAB TUTORIAL



Necmettin Yildirim

Division of Natural Sciences

New College of Florida

5800 BayShore Road, Sarasota FL 34243

August, 2016

Contents

Chapter 1. WHAT IS MATLAB?	4
1.1. MATLAB at New College	4
1.2. MATLAB Screen	5
1.3. Command Window	5
1.4. Using the Help Browser	5
1.5. Reserved MATLAB Variables	6
1.6. Vectors	6
1.7. What is in the memory of MATLAB	7
1.8. Matrices	8
1.9. Algebra with Matrices in MATLAB	9
1.10. Problems	10
Chapter 2. OPERATIONS with MATRICES in MATLAB	11
2.1. Solving a Linear System of Equations ($Bx = t$)	13
2.2. Randomly Generated Matrices	14
2.3. floor, round and ceil	15
2.4. Output formats	15
2.5. MATLAB Operations: Summary	17
2.6. Some Useful Built-In Mathematical Functions in Matlab	17
2.7. MATLAB Built-in Vector Functions	17
2.8. A few more useful built-in functions in Matlab	18
2.9. Concatenation of Matrices and Arrays in MATLAB	19
2.10. Problems	22
Chapter 3. FUNCTIONS and SCRIPTS in MATLAB	24
3.1. Inline functions	24
3.2. Saving and Loading your work in an “.m” file	25
3.3. Plot functions	25
3.4. Plot multiple curves in 2D with <code>plot</code> : Vector vs Matrix	27
3.5. Plotting curves in 3D with <code>plot3</code>	28
3.6. Plotting in 3D with <code>plot3</code>	29
3.7. Plotting in 3D with <code>surf</code>	29

3.8. Plotting in 3D with <code>surf</code>	30
3.9. <code>image</code>	30
3.10. Problems	31
Chapter 4. WRITING YOUR OWN FUNCTIONS and SCRIPTS in MATLAB	32
4.1. Driver codes and functions in MATLAB	32
4.2. <code>if...elseif...else...end</code> conditional statements	34
MATLAB's Relational and Logical Operators	35
4.3. Loops in MATLAB	37
4.4. Structure of <code>for...end</code> loop	37
4.5. Structure of <code>while...end</code> loop	40
4.6. Problems	44
Chapter 5. SOME BUILT-IN FUNCTIONS in MATLAB	46
5.1. <code>roots</code> : Polynomial roots	46
5.2. <code>fsolve</code> : Solving system of nonlinear equations with several variables	47
5.3. <code>fminbnd</code> Single-variable bounded nonlinear function minimization	48
5.4. <code>fminsearch</code> Multidimensional unconstrained nonlinear minimization	49
5.5. <code>lsqcurvefit</code> : solves non-linear least squares problems	51
5.6. MATLAB: Using the Debugger	51
5.7. MATLAB Hints (Source: Tobin A. Driscoll 2009)	52
5.8. Problems	53
Chapter 6. SOLVING ORDINARY DIFFERENTIAL EQUATIONS in MATLAB	56
6.1. Solving an ODE with an inline function	56
6.2. Solving an ODE with “.m” script and a function	57
6.3. Solving a System ODEs	58

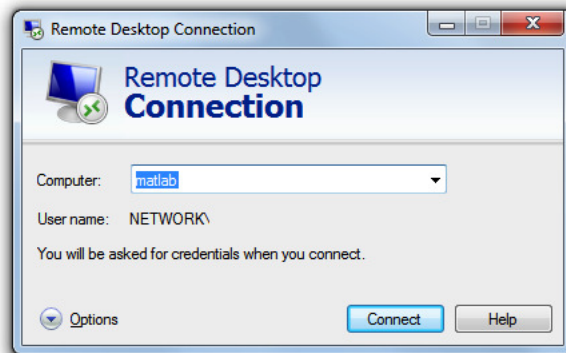
CHAPTER 1

WHAT IS MATLAB?

- MATLAB is a high level language with many specialized functions and toolboxes to ease simulation
- MATLAB stands for **Matrix Laboratory**
- Everything in MATLAB is encoded in a matrix
- This tutorial covers the basics of MATLAB, including matrices and functions, as well as simulation of ordinary differential equations (ODEs)

1.1. MATLAB at New College

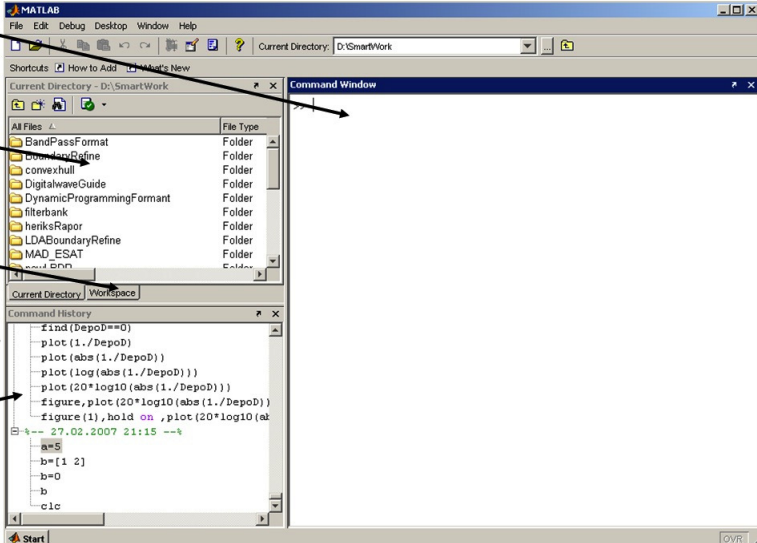
- New College has a MATLAB server. You must login to the server to run MATLAB.
- You need a user account in order to be able to connect to the server.
- To connect to the server, use Windows Remote Desktop Connection(RDC). If you have a Mac, you can download Microsoft Remote Desktop for free
- In Windows, you can access RDC using Start=>Accessories=>Remote Desktop Connection. Both PC and Mac users should then type matlab and enter your NCF email credential.



1.2. MATLAB Screen

Matlab Screen

- **Command Window**
 - type commands
- **Current Directory**
 - View folders and m-files
- **Workspace**
 - View program variables
 - Double click on a variable to see it in the Array Editor
- **Command History**
 - view past commands
 - save a whole session using diary



1.3. Command Window

- You can use MATLAB's command window as a fancy calculator. In the command window type the followings:

```
>> 2+2 % notice that MATLAB can simply be a calculator
```

```
ans =
```

```
4
```

```
>> x=10 % sets x to 10
```

```
x =
```

```
10
```

```
>> x=10;% the semicolon suppresses the output
```

```
>>
```

- MATLAB ignores anything after the comment sign “%”
- Use % to add comments in your code

1.4. Using the Help Browser

You can always consult the help browser

- MATLAB has thousands of built-in functions. You can search and get some help if you know the name of a MATLAB function

```
>> help 'name of a function'
```

EXAMPLE 1. To get some help on MATLAB's square root function "sqrt", just type `»help sqrt`.

```
>>help sqrt
SQRT   Square root.
SQRT(X) is the square root of the elements of X.
Complex results are produced if X is not positive.
See also sqrtm, realsqrt, hypot.
Reference page in Help browser doc sqrt
```

1.5. Reserved MATLAB Variables

Special variable names:

- `ans` : the answer of the last unassigned expression
- `pi` : 3.1415.....
- `eps` : the smallest possible number on this computer

1.6. Vectors

(A) Row Vectors

```
>> x=[2, 3, 6]
x =
     2     3     6
>> 2*x      % you can multiply a vector by a constant
ans =
     4     6    12
>> 5+x      % you can add a constant to a vector
ans =
     7     8    11
>> x=[2 3 6] % commas are optional
x =
     2     3     6
```

(B) Column Vectors

```

>> x=[2; 3; 6] % you have to put “;” between numbers
x =
2
3
6
>> x' % transpose of x
ans =
     2     3     6
>>x=2:2:10 %this notation is the same as x=[2 4 6 8 10]
x =
     2     4     6     8    10
>> y=sin(x) % creates a new vector of values from x.
y =
    0.9093   -0.7568   -0.2794    0.9894   -0.5440

```

```
>>x=linspace(X1, X2, N)
```

% generates a row vector of N linearly equally spaced points between X1 and X2.

```

x=linspace(1,3,5) % generates 5 numbers between 1&3
x=linspace(1,3)   % generates 100 numbers between 1& 3

```

1.7. What is in the memory of MATLAB

- **whos** : List current variables in the memory
- **clear** : clear variables and functions memory
- **clear all** : removes all variables from memory

```

>> whos % List current variables
Name      Size      Bytes  Class      Attributes
Example  1x1         976  struct
ans       1x2          16  double
x         1x5          40  double
y         1x5          40  double
>> clear x
>> whos
Name      Size      Bytes  Class      Attributes
Example  1x1         976  struct
ans       1x2          16  double
y         1x5          40  double
>>

```

1.8. Matrices

```
>> A= [ 1 2; 3 4] % defining a matrix
A =
     1     2
     3     4
>> B= [ 1 1; 1 1];
>> A*B % the symbol "*" is matrix multiplication
ans =
     3     3
     7     7
>> A.*B % term by term multiplication
ans =
     1     2
     3     4
>> A^2 % taking square of A, which is A*A
ans =
     7    10
    15    22
>> A.^2 % taking term by term square
ans =
     1     4
     9    16
>> A.^4 % fourth power of each term
ans =
     1    16
    81   256
```

1.8.1. Special Matrices.

```
>> eye(5) % create the "identity matrix" of size 5
ans =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
>> ones(4) % create a square matrix of ones
ans =
```



```

1     1     1     1
1     1     1     1
1     1     1     1
1     1     1     1

>> 5*eye(4)
ans =
5     0     0     0
0     5     0     0
0     0     5     0
0     0     0     5

>> ones(2,4) % create a 2x4 matrix with ones in every entry
ans =
1     1     1     1
1     1     1     1

>> zeros(2,4) % create a 2x4 matrix of zeros
ans =
0     0     0     0
0     0     0     0

>>

```

1.9. Algebra with Matrices in MATLAB

```

>> x=[2 4 6]
average = x*ones(3,1)/3
x =
2     4     6
average =
4

>>

```

EXERCISE 2. Try the same line as above with `ones(1,3)` replacing `ones(3,1)`. What happens?

- Punctuation: If you want to continue a line, use `...` at the end

```

>> longone = 1+2+3+4+5+ ...
6+7+8+9+10
longone =
55

```

- Most mathematical functions work as you'd expect:

```

x=[0 1]
sin(x)
asin(x)
exp(x)
log(x) % ?? Log of zero? What's that?

```

EXAMPLE 3. Apply *sqrt* to a simple 2×2 array of numbers such as $\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$.

1.10. Problems

- (1) Compute $\frac{35.7 \times 64 - 7^3}{45 + 5^2}$ and $\frac{5}{4} - 7 \times 6^2 + \frac{3^7}{9^3 - 652}$
- (2) Evaluate $a + \frac{ab}{c} + \frac{(a + 0.5(ab - c))^2}{\sqrt{|ab|}}$ if $a = 15.62$; $b = -7.08$; $c = 62.5$.
- (3) Use MATLAB's `linspace` to create the following matrix,

$$A = \begin{bmatrix} 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 \\ 69 & 68 & 67 & 66 & 65 & 64 & 63 & 62 \\ 1.4 & 1.1 & 0.8 & 0.5 & 0.2 & -0.1 & -0.4 & -0.7 \end{bmatrix}$$

CHAPTER 2

OPERATIONS with MATRICES in MATLAB

```
>> A = [1 2 3; 3 4 5; 6 7 8]
A =
     1     2     3
     3     4     5
     6     7     8
>> A(2,3) % entry located in 2nd row and 3rd column of A
ans =
     5
>> A(4,1)
??? Index exceeds matrix dimensions.
```

Submatrices can be “extracted”

```
>> A(1:2,2:3) % extracts first two rows and 2nd and 3rd columns
ans =
     2     3
     4     5
>> A(:,2) % extracts 2nd column
ans =
     2
     4
     7
>> A(3,:) % extracts 3rd row
ans =
     6     7     8
>> A([1 3],[2 3]) % extracts 1st&3rd rows,2nd&3rd columns
ans =
     2     3
     7     8
>> K=[1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15; 16 17 18 19 20]
K =
```

```
    1     2     3     4     5
    6     7     8     9    10
   11    12    13    14    15
   16    17    18    19    20

>> K(1:2:3,2:3:end) % extract 1st,3rd rows and 2nd&5th columns (until last)
ans =
     2     5
    12    15

>> B=[-1 3 10;-9 5 25;0 14 2];
>> s=[-1 8 5];
>> t=[7;0;11];
>> B-2          % subtracts 2 from each entry in B
ans =
    -3     1     8
   -11     3    23
    -2    12     0

>> A+B % sum of A and B
ans =
     0     5    13
    -6     9    30
     6    21    10

>> A*B % product of A and B
ans =
   -19    55    66
   -39    99   140
   -69   165   251

>> s-t
??? Error using ==> minus
Matrix dimensions must agree.

>> s-t' % subtracts transpose of t from s
ans =
    -8     8    -6

>> B*t % multiplies B and t
ans =
   103
   212
    22
```

```

>> B*s % tries to multiply B and s
??? Error using ==> mtimes
Inner matrix dimensions must agree.
>> B*s' % multiplies B and transpose of s
ans =
    75
   174
   122

```

2.1. Solving a Linear System of Equations ($Bx = t$)

$x = B^{-1}t$ is the solution of the system of linear equation $Bx = t$, where x and t are column vectors, and B is a matrix. Note the important differences between row vectors and column vectors!

```

>> B
B =
   -1     3    10
   -9     5    25
    0    14     2

>> t
t =
     7
     0
    11

>> x=B\t % solution of Bx=t
x =
    2.4307    0.6801    0.7390

>> B
B =
   -1     3    10
   -9     5    25
    0    14     2

>> s
s =
   -1     8     5

>> y=s/B % solution of yB=s

```

```
x =
0.1686 0.0924 0.5023
```

2.2. Randomly Generated Matrices

- `rand` : Generates UNIFORMLY distributed pseudorandom numbers on the unit interval $(0, 1)$
- `randn`: Generates NORMALLY distributed pseudorandom numbers with mean $\mu = 0$ and standard deviation $\sigma = 1$

```
>> rand(4) % 4x4 uniform random matrix from (0,1)
```

```
ans =
    0.8909    0.1493    0.8143    0.1966
    0.9593    0.2575    0.2435    0.2511
    0.5472    0.8407    0.9293    0.6160
    0.1386    0.2543    0.3500    0.4733
```

```
>> 2*rand(4) % 4x4 uniform random matrix from (0,2)
```

```
ans =
    0.7033    1.8344    0.7609    1.0616
    1.6617    0.5717    1.1356    1.5583
    1.1705    1.5144    0.1517    1.8680
    1.0994    1.5075    0.1079    0.2598
```

```
>> 1+2*rand(4) % 4x4 uniform random matrix from (1,3)
```

```
ans =
    2.1376    1.3244    1.3313    2.3784
    1.9388    2.5886    2.2040    2.4963
    1.0238    1.6224    1.5259    1.9011
    1.6742    2.0571    2.3082    1.1676
```

```
>> randn(5) % from normal distribution with 0 mean and 1 std
```

```
ans =
   -0.8396    1.4367    0.8252    1.0984   -0.8236
    1.3546   -1.9609    1.3790   -0.2779   -1.5771
   -1.0722   -0.1977   -1.0582    0.7015    0.5080
    0.9610   -1.2078   -0.4686   -2.0518    0.2820
    0.1240    2.9080   -0.2725   -0.3538    0.0335
```

EXAMPLE 4. How can you generate 100 random numbers from the uniform distribution on the interval $[a, b]$?

Answer: `r = a + (b-a).*rand(100,1);`

EXAMPLE 5. How can you generate a 6x6 random matrix from the uniform distribution on the interval $[15, 25]$?

Answer: `15+round(10*rand(6))`

```
>> A= 15+round(10*rand(6))
A =
    24    22    24    15    20    16
    24    20    23    20    16    23
    18    20    21    17    22    23
    22    24    17    25    15    22
    17    21    17    22    16    16
    15    21    24    20    20    22
```

2.3. floor, round and ceil

- `floor` : Round towards negative infinity.
- `ceil` : Round towards positive infinity.
- `round` : Round towards nearest integer.

```
>> round(-1.8)
ans =
    -2
>> ceil(-1.8)
ans =
    -1
>> floor(-1.8)
ans =
    -2
```

2.4. Output formats

- `format short/long/short e/long e`: Set output formats.

```
>> pi
ans =
    3.1416
>> format long
>> pi
ans =
    3.141592653589793
```

```
>> format short e
>> pi
ans =
    3.1416e+000
```


2.5. MATLAB Operations: Summary

+ : addition
 - : subtraction
 * : multiplication
 ^ : power
 ' : transpose
 \ : left division
 / : right division
 .* : term by term multiplication
 .^ : term by term power
 .\ : term by term left division
 ./ : right division

2.6. Some Useful Built-In Mathematical Functions in Matlab

sin : trigonometric sine
 cos : trigonometric cosine
 tan : trigonometric tangent
 asin : trigonometric inverse sine (arcsine)
 acos : trigonometric inverse cosine (arccosine)
 atan : trigonometric inverse tangent (arctangent)
 exp : exponential log natural logarithm
 abs : absolute value
 sqrt : square root
 rem : remainder

2.7. MATLAB Built-in Vector Functions

max : largest component
 min : smallest component
 length : length of a vector
 sort : sort in ascending order
 sum : sum of elements
 prod : product of elements
 median : median value (middle value when sorted out)
 mean : mean value ($\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$)
 std : standard deviation ($\bar{x} = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$)

```

>> u=[0.0155  0.9841  0.1672  0.1062  0.3724];
>> max(u)
ans =
    0.9841
>> sort(u)
ans =
    0.0155  0.1062  0.1672  0.3724  0.9841

```

EXAMPLE 6. max/min/sort... works for matrices column wise

```

>> A=rand(3,4) % generate a random 3x4 matrix from uniform distribution
A =
    0.1981    0.9203    0.4228    0.9831
    0.4897    0.0527    0.5479    0.3015
    0.3395    0.7379    0.9427    0.7011

>> max(A)
ans =
    0.4897    0.9203    0.9427    0.9831

```

2.8. A few more useful built-in functions in Matlab

```

size : size of a matrix
det  : determinant of a square matrix
inv  : inverse of a matrix
rank : rank of a matrix
rref : row reduced echelon form
eig  : eigenvalues and eigenvectors
poly : characteristic polynomial
lu   : LU factorization
qr   : QR factorization
>>A=[-1 2 5;6 -1 4; 1 0 1];
>> det(A)
ans =
    2.0000
>> inv(A)
ans =
   -0.5000   -1.0000    6.5000

```

```

-1.0000    -3.0000    17.0000
 0.5000     1.0000    -5.5000
>> A*inv(A) % should give the identity matrix
ans =
 1.0000         0     0.0000
 0.0000     1.0000    -0.0000
 0.0000     0.0000     1.0000
>> [V,D]=eig(A) % computes eigenvalues and eigenvectors of A
V =
 0.5718    -0.5363    -0.3319
-0.8143    -0.8225    -0.8947
-0.0999    -0.1893     0.2989
D =
-4.7221         0         0
         0     3.8326         0
         0         0    -0.1105

```

EXERCISE 7. Use MATLAB to check if the columns of V are eigenvectors and the diagonal entries of D are corresponding eigenvalues of U

```

>> A*V(:,1)-D(1,1)*V(:,1)
ans =
 1.0e-014 *
 0.1332    -0.0444    -0.0222

```

2.9. Concatenation of Matrices and Arrays in MATLAB

You can concatenate matrices and arrays in MATLAB. See the examples below.

Let \mathbf{a} , \mathbf{b} , \mathbf{A} and \mathbf{B} be the following matrices

```

>> a=rand(2,2)
a =
 0.8407    0.8143
 0.2543    0.2435
>> b=rand(3,2)
b =
 0.9293    0.2511
 0.3500    0.6160
 0.1966    0.4733

```

```
>> A=[1:5;6:10]% Define a 2x5 matrix
A =
     1     2     3     4     5
     6     7     8     9    10
>> B=3*[1:5;6:10]% Define a 2x5 matrix
B =
     3     6     9    12    15
    18    21    24    27    30
>> C=[A B] % Create 2x10 matrix
C =
     1     2     3     4     5     3     6     9    12    15
     6     7     8     9    10    18    21    24    27    30
>> D=[b' a] % 2x5 matrix
D =
    0.9293    0.3500    0.1966    0.8407    0.8143
    0.2511    0.6160    0.4733    0.2543    0.2435
>> E=[A;B]% 4x5 matrix
E =
     1     2     3     4     5
     6     7     8     9    10
     3     6     9    12    15
    18    21    24    27    30
>> F=[A;[b' a]] % 4x5 matrix
F =
    1.0000    2.0000    3.0000    4.0000    5.0000
    6.0000    7.0000    8.0000    9.0000   10.0000
    0.9293    0.3500    0.1966    0.8407    0.8143
    0.2511    0.6160    0.4733    0.2543    0.2435
>> Z(:,:,1)=A; Create a 3D matrix
>> Z(:,:,2)=B;
>> Z
```

```

Z(:,:,1) =
     1     2     3     4     5
     6     7     8     9    10
Z(:,:,2) =
     3     6     9    12    15
    18    21    24    27    30

>> F=Z(1,,:) % first front slide
F(:,:,1) =
     1     2     3     4     5
F(:,:,2) =
     3     6     9    12    15
>> F2=Z(2,,:) % second front slide
F2(:,:,1) =
     6     7     8     9    10
F2(:,:,2) =
    18    21    24    27    30

>> G=Z(:,1,)% first xx slide
G(:,:,1) =
     1
     6
G(:,:,2) =
     3
    18
>> G4=Z(:,4,)% 4th xx slide
G4(:,:,1) =
     4
     9
G4(:,:,2) =
    12
    27

>> A(:,2)=[ ] % Eliminate the second column
A =
     1     3     4     5
     6     8     9    10
>> A(:,[3 4])=[ ] % Eliminate the third and fourth columns

```

```

A =
     1     3
     6     8
>> A(1,:)=[] % Eliminate the first row
A =
     6     8

```

2.10. Problems

(1) Define $A = \begin{bmatrix} 2 & 9 & 0 & 0 \\ 0 & 4 & 1 & 4 \\ 7 & 5 & 5 & 1 \\ 7 & 8 & 7 & 4 \end{bmatrix}$, $b = \begin{bmatrix} -1 \\ 6 \\ 0 \\ 9 \end{bmatrix}$ and $a = [3 \ -2 \ 4 \ -5]$ and calculate

the following matrices (when defined)

- $A \cdot b$
 - $a + 4$
 - $b \cdot a$
 - $a \cdot b^T$
 - $A \cdot a^T$
- Explain any difference between the answers that MATLAB returns when you type in $A*A$, A^2 and $A.^2$
 - What is the command that isolates the submatrix that consists of the 2nd to 3rd rows of the matrix A ?
 - Consider

$$A = \begin{bmatrix} 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 \\ 69 & 68 & 67 & 66 & 65 & 64 & 63 & 62 \\ 1.4 & 1.1 & 0.8 & 0.5 & 0.2 & -0.1 & -0.4 & -0.7 \end{bmatrix}$$

What is the Matlab command that creates a 3x4 submatrix B from the 1st, 3rd, and 4th rows, and the 1st, 3rd, 5th, and 7th columns of the matrix A .

- The depth of a well in meters, d , can be calculated from the time it takes for a stone that is dropped into the well (with zero initial velocity) to hit the bottom by the equation

$$d = \frac{1}{2}gt^2$$

where t is time in seconds and $g = 9.81 \text{ (m/sec}^2\text{)}$, which is the gravitational acceleration. Determine d for the impact times $t = 1, 2, 3, \dots, 10$

- For $x = [2:2:10]$ and $y = [3:3:15]$, calculate z using element-by-element calculation if $z = \frac{xy + \frac{y}{x}}{(x+y)(y-x)} + \frac{12x}{y}$

- (7) Estimate numerically that $\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n = e$. First create a vector n as $n = [1 \ 10 \ 100 \ 500 \ 1000 \ 2000 \ 4000 \ 8000]$. Then, create a new vector y in which each entry of the vector is calculated by $y = \left(1 + \frac{1}{n}\right)^n$ from each n . Compare the elements of y with the known value of $e \approx 2.71828$. Show numerically that as n increases, y approaches the limit of $\exp(1)$, or e .

CHAPTER 3

FUNCTIONS and SCRIPTS in MATLAB

3.1. Inline functions

An easy way of defining functions in MATLAB is inline functions:

Functions with one variable.

```
>>f=@(x) x^2+1
f =
@(x)x^2+1
>> f(2)
ans =
5
```

Functions with two variables.

```
>>f=@(x,y) x^2+y/5
f =
@(x,y)x^2+y/5
>>f(1,5)
ans =
2
```

Function with three variables.

```
>>f=@(x,y,z) 1/z+x^2+y/5;
>>f(1,2,5)
ans =
1.6000
```


Function with two variables (alternative).

```
>>f=@(x) x(1)*x(2)+x(1);
>>f([3 4])
ans =
15
>>f=@(x) x(1)+x(2)/7+x(3)^4;
>>f([1 1 1])
ans =
2.1429
```

3.2. Saving and Loading your work in an “.m” file

- If you type

```
>>save myfile
```

in the command window, all defined variables will be saved to a file called `myfile.mat` in the current directory. This file format is specific to MATLAB. You can also select a subset of variables to be saved by typing their names after the file name such as

```
>>save myfile a,b,C
```
- If you later enter `load myfile`, the saved variables are returned to the workspace, overwriting any presently defined values assigned to the same names.

3.3. Plot functions

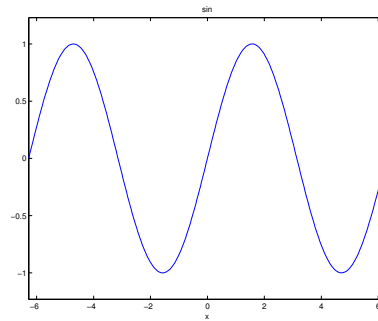
```
ezplot : Easy to use 2D-plot
plot    : 2D plotter
plot3d  : 3D plotter
image   : Displays image of a matrix
surf    : 3-D colored surface.
surfc   : 3-D colored surface.
```

Plotting with ezplot. ezplot makes two-dimensional (2D) plots of explicit, implicit or parametric functions.

- Explicit functions: $y = f(x)$
- Implicit functions: $F(x, y) = 0$
- Parametric functions: $\begin{cases} x = f(t) \\ y = g(t) \end{cases}$

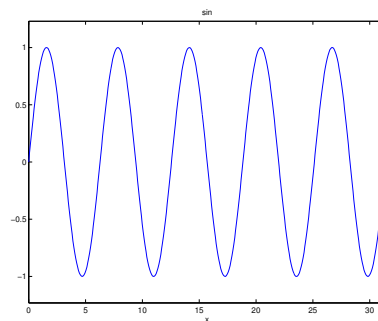
ezplot for explicit functions. `ezplot(fun)` plots the function $\text{fun}(x)$ over the default domain $-2\pi < x < 2\pi$, where $y = \text{fun}(x)$ is an explicitly defined function of x .

```
>> ezplot(@sin)
```



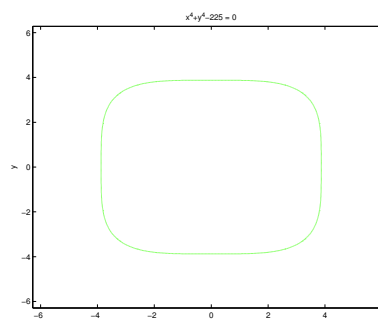
ezplot for explicit functions: defining a custom domain.

```
ezplot(@sin, [0 10*pi])
```



ezplot for implicit functions. `ezplot(fun)` plots the implicitly defined function $\text{fun}(x,y) = 0$ over the default domain $-2\pi < x < 2\pi$ and $-2\pi < y < 2\pi$.

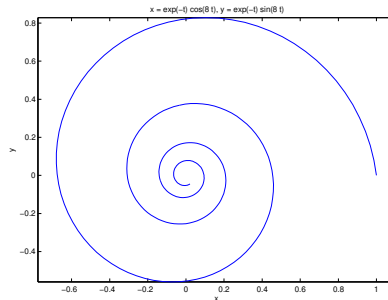
```
>> ezplot(@(x,y) x.^4+y.^4-225)
```



ezplot for parametric function.

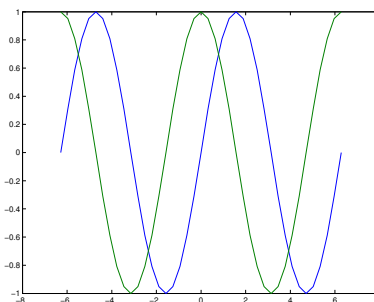
`ezplot(fx,fy,[tmin ,tmax])` plots $fx(t)$ and $fy(t)$ over $t_{min} < t < t_{max}$.

```
>> x= @(t) exp(-t).*cos(8*t);
>> y= @(t) exp(-t).*sin(8*t);
>> ezplot(x,y,[0 3])
```



3.3.1. Plot curves in 2D with ‘plot’ function. `plot(X,Y)` plots vector Y versus vector X. If X or Y is a matrix, then the vector is plotted versus the rows or columns of the matrix, whichever line up.

```
>> x = -2*pi:(pi/10):2*pi;
>> y1 =sin(x);
>> y2 =cos(x);
>> plot(x,y1,x,y2)
```

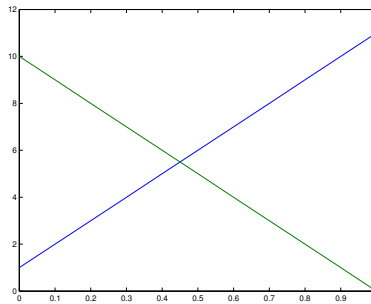


3.4. Plot multiple curves in 2D with plot: Vector vs Matrix

EXAMPLE 8. X and Y are both matrices, then they must have equal size. `plot(X,Y)` plots columns of Y versus columns of X. If X is a vector, then the plot function plots each matrix column Y versus X.

```
>> x=[0:.1:1];
>> A=[1:11;10:-1:0]
```

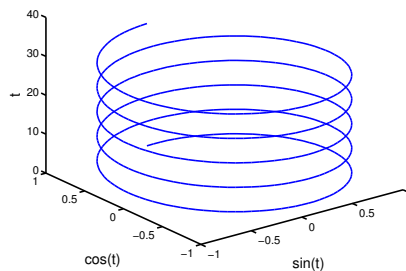
```
>> plot(x,A)
```



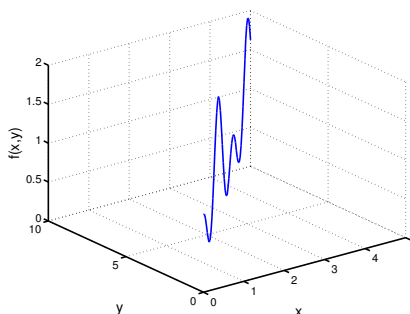
3.5. Plotting curves in 3D with plot3

- `plot3(x,y,z)`, where `x`, `y` and `z` are three vectors of the same length, plots a line in 3-space through the points whose coordinates are the elements of `x`, `y` and `z`.

```
>> t = 0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t); %parametric functions
```



```
>> x=[0:.1:5];
>> y=2*[0:.1:5];
>> f=@(x,y) sin(x).^2+cos(y).^2; % explicit functions
>> plot3(x,y,f(x,y))
```

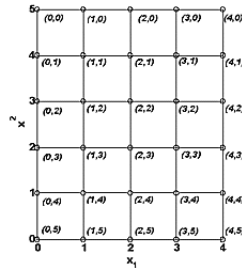


3.6. Plotting in 3D with plot3

`plot3(X,Y,Z)`, where X , Y , Z are vectors or matrices, plots one or more lines in three-dimensional space through the points whose coordinates are the elements of X , Y , and Z .

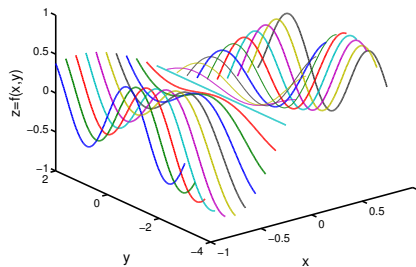
EXAMPLE 9. Creating a meshgrid

```
>> x1=0:4;
>> x2=0:5;
>> [x,y]=meshgrid(x1,x2) % creating a meshgrid
```



EXAMPLE 10. Plot $z = \sin(xy) \cos(y)$ when $-1 \leq x \leq 1$ and $-3 \leq y \leq 2$

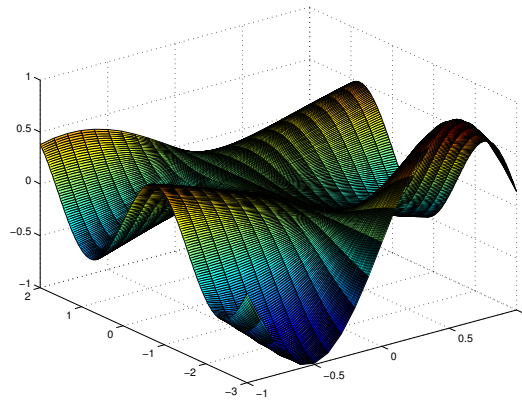
```
>> [x,y]=meshgrid(-1:.1:1,-3:.02:2);
>> z=sin(x.*y).*cos(y);
>> plot3(x,y,z)
```



3.7. Plotting in 3D with surf

`surf(X,Y,Z)` plots the colored parametric surface.

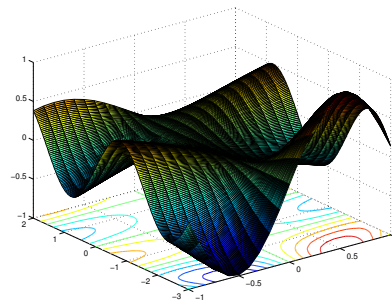
```
>> [x,y]=meshgrid(-1:.1:1,-3:.02:2);
>> z=sin(x.*y).*cos(y);
>> surf(x,y,z)
```



3.8. Plotting in 3D with surfc

surfc: : Combination surf/contour plot.

```
>> [x,y]=meshgrid(-1:.1:1,-3:.02:2);  
>> z=sin(x.*y).*cos(y);  
>> surfc(x,y,z)
```



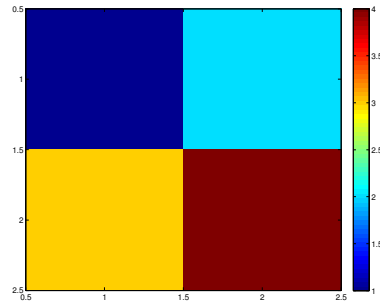
3.9. image

The **image** command is useful for large matrices which would take too much computing time to plot

image(C): : displays matrix C as an image.

imagesc(C): : Scale data and display as image.

```
>> A=[1,2;3,4];  
>> imagesc(A),colorbar
```



3.10. Problems

- (1) Compute $2 + 3$, 2×3 , $2/3$, 2^3 , $2^{\frac{1}{3}}$, $\log(3)$, $\cos(3)$, $\sin(5)$ in the command line.
- (2) Plot $f(x) = e^{-x}$ on $[-2, 2]$.
- (3) Plot $g(t) = |t|$ on $[-1, 1]$.
- (4) Use `linspace` to generate a row vector of 100 linearly evenly spaced points between -1 and 1 and evaluate $f(x) = x^2$ at each point. Plot the graph.
- (5) Make surface plots of the following functions over the given ranges:
 - (a) $(x^2 + 3y^2)e^{-x^2-y^2}$, $-3 \leq x \leq 3$, $-3 \leq y \leq 3$.
 - (b) $-3y/(x^2 + y^2 + 1)$, $|x| \leq 2$, $|y| \leq 4$.
 - (c) $|x| + |y|$, $|x| \leq 1$, $|y| \leq 1$.

CHAPTER 4

WRITING YOUR OWN FUNCTIONS and SCRIPTS in MATLAB

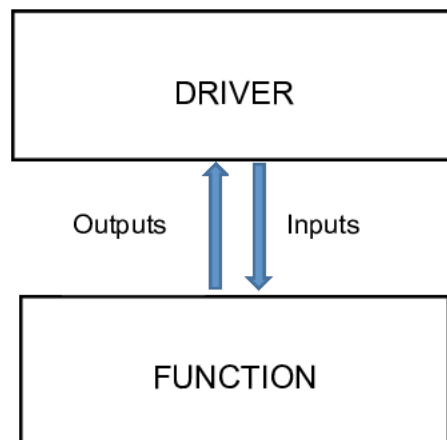
An M-file is a plain text file containing MATLAB commands and saved with the file name extension .m. There are two types:

- Scripts
- Functions

4.1. Driver codes and functions in MATLAB

Driver Code (MATLAB Script): sends inputs to the function and gets back output

Function Code (MATLAB function): takes in inputs from driver and runs an algorithm and sends back outputs to the driver



Reminder: an extremely important type of statement in any M-file is the comment, which is indicated by a percent sign `%`. Any text on the same line after a percent sign is ignored. Furthermore, the **first block** of comments in an M-file serves as documentation for the file and will be typed out in the command window if the `help` command is used on the file.

Structure of a MATLAB Script.

```
% Some comments go here
% .....
% .....
Input

    Body of Algorithm

Output
```

EXAMPLE 11. Open a new MATLAB editor page and type the following MATLAB commands

```
% This MATLAB scripts multiplies a 3x3 random matrix
% by a 3x1 random vector
clc
clear all
A=rand(3)
b=rand(1,3)
c=A*b'
```

and save it under a name Driver.m. Then go to the command window type Driver and push ENTER. What happens?

Structure of a MATLAB Function.

```
function Output=FunctionName(Input)

    Body of Algorithm

Output=...;
```

Typical MATLAB function with starts with a line such as

```
function [output1,output2] = myfun(input1,input2,input3)
```

EXAMPLE 12. Take the script in Example 11 and convert it into a MATLAB function. Your function will have two inputs: (1) 3x3 random matrix, (2) 3x1 random vector. It will have one output, c , which is defined as $c = A \cdot b^T$

```
function c=fcn(A,b)
% This MATLAB scripts multiplies a 3x3 random matrix
% by a 3x1 random vector
c=A*b';
```

EXAMPLE 13. Write a set of a function and a driver that implements the quadratic formula for finding the roots of $ax^2 + bx + c = 0$

$$\begin{aligned} ax^2 + bx + c &= 0 \\ x_{1,2} &= \frac{-b \pm d}{2a} \\ d &= \sqrt{b^2 - 4ac} \end{aligned}$$

```
% Driver
a=1;
b=4;
c=3;
[x1,x2] = quadform(a,b,c)
```

In a separate file:

```
function [x1,x2] = quadform(a,b,c)
d = sqrt(b^2 - 4*a*c);
x1 = (-b + d)/(2*a);
x2 = (-b - d)/(2*a);
```

EXAMPLE 14. Write a MATLAB function that computes base 3 logarithm of any given number by using the change-of-base formula

```
function a=log3(x)
% This code computes the base 3 logarithm of x
a=log(x)/log(3);
```

4.2. if...elseif...else...end conditional statements

These statements are executed or not depending on whether a relation holds true.

Structure of “if elseif...elseif...else...end” statement.

```
if (relation)
    statement(s)
elseif (relation)
    statement(s)
elseif (relation)
    statement(s)
    :
else
    statement(s)
end
```

The simplest if...end statement looks like

```
if (relation)
    statement(s)
end
```

MATLAB's Relational and Logical Operators

- $<$: less than
- $>$: greater than
- \leq : less than or equal to
- \geq : greater than or equal to
- $==$: equal to
- $\sim =$: not equal to

EXAMPLE 15. An example

```
if (5>3)
    disp('5 is bigger than 3')
end
```

EXAMPLE 16. MATLAB function with an if..else..if statement that computes $\log_3(x)$ for a given number $x > 0$

```
function a=log3(x)
% This code computes the base 3 logarithm of x
a = NaN;
if sum(-x>0)==0
    a=log(x)/log(3);
else
    disp('x has negative value(s)')
end
```

EXAMPLE 17. The Newton-Raphson method (also known as Newton's Method) for finding roots of the equation $f(x) = 0$ is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

To run this method, an initial guess x_0 has to be provided. Now, write a MATLAB function that implements this method to approximate $\sqrt{5}$ in 4 steps

If $f(x) = x^2 - 5 \Rightarrow f'(x) = 2x$. Hence the formula becomes

$$x_{n+1} = \frac{x_n}{2} + \frac{5}{2x_n}$$

Given x_0 , we want to compute x_4 .

```
% Driver part (Filename: Driver.m)
x4=NewtonAlg(x0)

% function part(Filename: NewtonAlg.m)
function [x4]=NewtonAlg(x0)
x1=x0/2+5/(2*x0);
x2=x1/2+5/(2*x1);
x3=x2/2+5/(2*x2);
x4=x3/2+5/(2*x3);
```

EXAMPLE 18. MATLAB function for $f(x) = |x|$

```
function y = AbsVal(x)
y=x;
  if x<0
    y=-x;
  end
```

4.3. Loops in MATLAB

There are two basic kinds of loops

- (1) for...end loops
- (2) while...end loops

4.4. Structure of for...end loop

Statements (indexed by j) are executed step by step first through last.

```
for j=first:last
  (statements)
end
```

EXAMPLE 19. The following script prints the first four positive integers

```
for i=1:4
  i
end
```

EXAMPLE 20. The following script squares each entry in a given vector v with length 10

```
v=1:10 % define v here
for i=1:10
  v2(i) = v(i)^2;
end
```

EXAMPLE 21. The Newton-Raphson method for the same function $f(x) = x^2 - 5$ with a for..end loop in 4-steps is given below

```

% Driver part
x0=1;
RootOut=NewtonAlg(x0)

% function part
function [x_out]=NewtonAlg(x_in)
for i=1:4
    x_out = x_in/2+5/(2*x_in) ;
    x_in = x_out;
end

```

4.4.1. Nested for..end loops.

```

for j=j0:jn
    (statements)
    for k=k0:kn
        (statements)
    end
end

```

EXAMPLE 22. The code

```

for i=1:2
    for j=1:3
        [i,j]
    end
end

```

will produce the following output

```

i = 1 1 1 2 2 2
j = 1 2 3 1 2 3

```

EXAMPLE 23. A function to compute the transpose of a given $n \times n$ square matrix A

```

function B=transposeA(A)
%This code computes transpose of a matrix
[m,n] = size(A);
for i=1:n
    for j=1:n
        B(i,j)=A(j,i);
    end
end

```

```

end
end

```

EXAMPLE 24. Write a function that adds up all the entries in a given $m \times n$ matrix A

```

function [s_sum]=Sum_of_A(A)
% This function computes sum of all entries in a matrix
[m,n] = size(A);
s_sum=0
for i=1:m
    for j=1:n
        s_sum=s_sum+A(i,j);
    end
end
end

```

EXAMPLE 25. A function that produces an upper triangular $m \times n$ matrix B from an $m \times n$ matrix A . In other words,

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix} \Rightarrow B = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{mn} \end{bmatrix}$$

Answer:

```

function [B]=Upper(A)
[m,n]=size(A);
B=zeros(m,n);
for i=1:m
    for j=i:n
        B(i,j)=A(i,j);
    end
end
end

```

EXAMPLE 26. The dot product of two vectors with same sizes $v = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$ and $w = \begin{bmatrix} w_1 & w_2 & \dots & w_n \end{bmatrix}$ are defined as

$$v \cdot w = \sum_{i=1}^n v_i w_i = v_1 w_1 + v_2 w_2 + \dots + v_n w_n$$

The l_2 -norm of v is defined as

$$|v| = \sqrt{\sum_{i=1}^n v_i^2} = \sqrt{v_1^2 + v_2^2 + \dots + v_n^2}$$

EXERCISE. Write a function that takes in v and w and sends out their dot product and l_2 -norms.

4.5. Structure of `while...end` loop

Statement is executed while the boolean expression remains true.

```
while (expression)
    (statement)
end
```

EXAMPLE 27. The following code prints the first four positive integers and employs a `while...end` loop

```
j=1;
while j<=4
    j
    j=j+1;
end
```

EXAMPLE 28. The following code computes roots of $x^2 - 5 = 0$, until numerical error becomes less than given a preset error tolerance Er and uses `while...end`

```
% Driver part
clc
Er      = 1e-8;
x_out   = 0;
x_in    = 1;
[x_out] = NewtonAlg(x_in,Er)
% function part
function [x_out]=NewtonAlg(x_in,Er)
while abs(x_in-sqrt(5))>Er
```



```

x_out = x_in/2+5/(2*x_in) ;
x_in  = x_out;
end

```

EXAMPLE 29. Determine what the following MATLAB function does. Then explain the output of `[a] = exple(10)`

```

function [a] = exple(n)
a = 0;
while 2^a < n
    a = a + 1;
end % End of function

```

EXAMPLE 30. The code given below computes $S = \sum_{n=1}^{100} n$

```

% Sum of first 100 positive integers
tsum=0;
for i=1:100
    tsum=tsum+i;
end
tsum

```

EXAMPLE 31. Generate 2000 random numbers between 0 and 2000 and count how many of them are in the following intervals. Then use MATLAB's `bar` built-in function to plot the counts you computed to produce a histogram.

- (a) [0.00,0.25)
- (b) [0.25,0.50)
- (c) [0.50,0.75)
- (d) [0.75,1.00]

Answer:

```

A      = rand(1,2000);
count  = zeros(1,4);
for i   = 1:2000
    if (A(i)>=0.0 && A(i)<0.25)
        count(1)=count(1)+1;
    elseif (A(i)>=0.25 && A(i)<0.50)

```

```

        count(2)=count(2)+1;
elseif (A(i)>=0.50 && A(i)<0.75)
        count(3)=count(3)+1;
else
        count(4)=count(4)+1;
end
end
end

```

EXAMPLE 32. Write a function that approximates $\sqrt{5}$ using Newton's Method in n steps in two different ways:

- (a) without using an array
- (b) using an array to store all approximations to $\sqrt{5}$

Newton Algorithm 2 (using for loop)

```

function [x_out] = NewtonAlg2(x_in,n)
    for i = 1:n
        x_out = x_in/2 + 5/(2*x_in);
        x_in = x_out;
    end
end

```

Newton Algorithm 3 (using array)

```

function [xfinal] = NewtonAlg3(x_in,n)
x=zeros(1,numIter+1);
x(1) = x_in;
for i = 1:n
    x(i+1) = x(i)/2 + 5/(2*x(i));
end
xfinal = x(i+1);

```

EXAMPLE 33. Write a MATLAB script with a for...end loop and if...else...end statements that plots the graph of the following piecewise function

$$f(x) = \begin{cases} x^2 & \text{if } -1 \leq x < 0.5 \\ 0.25 & \text{if } 0.5 \leq x \leq 1 \end{cases}$$

Answer:....

```
x=[-1:.1:1];
for i=1:length(x)
if x(i)<0.5
    f(i)=x(i)^2;
else
    f(i)=0.25;
end
end
plot(x,f)
xlabel('x')
ylabel('f')
```

EXAMPLE 34. Develop a MATLAB function that solves the linear system $Ax = b$ for a given $n \times n$ square matrix A and a $n \times 1$ vector b (Use `if...end` statements to check if the system has a solution first)

4.6. Problems

Answer the following questions

- (1) Write a MATLAB function that computes $n! = n \times (n - 1) \times \dots \times 3 \times 2 \times 1$ (Use `if...end` and `for...end` statements)
- (2) Write a MATLAB function that computes $\sum_{k=1}^n \frac{1}{k^2}$.
- (3) The Fibonacci sequence $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$, where $F_0 = 0$; $F_1 = 1$. Write a MATLAB function that computes the n^{th} Fibonacci number.
- (4) Develop a driver called *Driver_Quad.m* and two other MatLab functions called *R_Quad.m* and *Disc_Quad.m* that solves a quadratic equation

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

Organize your codes so that the driver code *Driver_Quad.m* supplies the coefficients a, b and c into the function file *R_Quad.m*, that computes the roots x_1 and x_2 from the quadratic formula and sends them back to the driver *Driver_Quad.m*. While computing the roots in *R_Quad.m*, have it call another function file *Disc_Quad.m* to get the discriminant value calculated.

- (5) Plot the graphs of $f(x) = x^2$, $g(x) = x^3$ for $x = -1, \dots, 1$ on the same axis. Label the x and y axes and create a legend indicating which graph is which.
- (6) Let $x = [3, 2, -1, 5, 8]$, determine outputs of each of the following MATLAB command
 - (a) `x(end)`
 - (b) `x(3)`
 - (c) `length(x)`
 - (d) `x(2:4)`
- (7) A magic square is an $n \times n$ matrix in which each integer $1, 2, \dots, n^2$ appears once and for which all the row, column, and diagonal sums are identical. MatLab has a command `magic` that returns magic squares. Check its output when $n=5$ and $n=20$ and use MatLab to verify the summation property. (The antidiagonal sum will be the trickiest. Look for help on “`fliplr`” .)
- (8) Let P1, P2, and P3 be the vertices of an equilateral triangle. Start with a point anywhere inside the triangle. At random, pick one of the three vertices and move halfway toward it. Repeat indefinitely. If you plot all the points obtained, a very clear pattern will emerge. (Hint: This is particularly easy to do if you use complex numbers. If z is complex, then `plot(z)` is equivalent to `plot(real(z), imag(z))`.)

- (9) Generate 100 random matrices using `randn(100)`, and plot all of their eigenvalues as dots in the complex plane on one graph. (Thus, you should see 10,000 dots.) Use `axis equal` to make the aspect ratio one-to-one. You should see a fairly striking result.

CHAPTER 5

SOME BUILT-IN FUNCTIONS in MATLAB

5.1. roots: Polynomial roots

roots computes the roots of the polynomial whose coefficients are the elements of the vector C . If C has $N + 1$ components, then the polynomial is $C_1X^N + \dots + C_NX + C_{N+1}$

EXAMPLE 35. Solve for x

$$x^2 + 3x + 2 = 0,$$

```
roots([1 3 2])
ans=
-2
-1
```

EXAMPLE 36. Solve for x

$$x^5 + x^4 + x^3 + x^2 + x + 1 = 0,$$

```
>> S=roots([1 1 1 1 1 1 ])
S =
 0.5000 + 0.8660i
 0.5000 - 0.8660i
-1.0000
-0.5000 + 0.8660i
-0.5000 - 0.8660i
```

EXAMPLE 37. Lets solve

$$x^2 + 3x + 5 = 0,$$

```
>>S= roots([1 3 5])
S =
```

```

-1.5000 + 1.6583i
-1.5000 - 1.6583i
>> real(S)
ans =
-1.5000    -1.5000
>> imag(S)
ans =
1.6583    -1.6583

```

EXAMPLE 38. Look at another example

$$2x^5 - 13x^2 + 5x + 1 = 0$$

```

>>roots([2 0 0 -13 5 1])
ans =
-1.0437 + 1.6365i
-1.0437 - 1.6365i
1.6925
0.5401
-0.1452

```

5.2. fsolve: Solving system of nonlinear equations with several variables

fsolve attempts to solve equations like
 $F(X) = 0$
 where F and X may be vectors or matrices.

EXAMPLE 39. Solve $x - e^x = 0$, $x_0 = 2$

```

>> f=@(x) x-exp(x);
>> fsolve(f,2)
ans =
3.0136e-07
% Alternative way:
function Driver
x =fsolve(@fcn,2)
function f = fcn(x)
f = x-exp(x);

```

EXAMPLE 40. Solve the following system numerically using fsolve

$$\begin{cases} x^2 - xy + y^2 = 1 \\ 3xy - y^3 = 2 \end{cases}$$

Solve the following system numerically using fsolve

$$\begin{cases} 2x - y = e^{-x} \\ -x + 2y = e^{-y} \end{cases}$$

5.3. fminbnd Single-variable bounded nonlinear function minimization

`X = fminbnd(FUN,x1,x2)` attempts to find a local minimizer `X` of the function `FUN` in the interval `x1 < X < x2`.

EXAMPLE 41. Minimize the following function over $-5 \leq x \leq 5$

$$f(x) = \sin x$$

```
clear all
clc
clf
%% List of functions
fcu=@(x) sin(x);
[xout fval] =fminbnd(fcu,-5*0,1)
plot(xout,fval,'ro','markersize',20),
hold on
u=[-3:.01:3];
plot(u,fcu(u))
```

EXERCISE 42. Use the code above to minimize the following functions

- $f(x) = \sin x - 0.6$
- $f(x) = x^2$
- $f(x) = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{((x-0.9)^2+.04)-6}$

5.4. `fminsearch` Multidimensional unconstrained nonlinear minimization

`X = fminsearch(FUN,X0)` starts at `X0` and attempts to find a local minimizer `X` of the function `FUN`.

EXAMPLE 43. Minimize the following function (called the Rosenbrock Valley Function) starting from $x_0 = (2, 3)$

$$f(x_1, x_2) = 100(x_2 - x_1)^2 + (1 - x_1)$$

```
%% Rosenbrock Valley
f =@(x) 100*(x(2)-x(1)^2).^2+(1-x(1)).^2;
[xout fval] =fminsearch(f,[2 2])
%% Plotting Rosenbrock Valley
[X,Y]=meshgrid(-3:0.2:3,-3:0.2:3);
surf(X,Y,f(X,Y))
```

EXAMPLE 44. Minimize the following function (Rastring Function) starting from $(2, 3)$

$$f(x_1, x_2) = 20 + 10 \cos(2\pi x_1) + 10 \cos(2\pi x_2)$$

```
%% Rastrigin function
[xout fval] =fminsearch(f,[2 2])
f =@(x) 20+10*cos(2*pi*x(1))+10*cos(2*pi*x(2));
%% Plotting Rastrigin function
[X,Y]=meshgrid(-2:0.02:2,-2:0.02:2);
Z = 20 +10*cos(2*pi*X)+10*cos(2*pi*Y);
surf(X,Y,Z)
```

EXAMPLE 45. (Data fitting)Codes that minimize the sum of square differences between the data and a given function

```
%% ----- Driver function -----
function fminsearchApplication
%% Generating artificial data
clc
a = 1;
```

```
b    = 1;
c    = 1;
xdata = -5:.5:5;
per  = 30/100;
ydata = fcn(xdata,a,b,c).*(1+per*randn(1,length(xdata)));
%% Estimate a,b,c
a0  = rand;
b0  = rand;
c0  = rand;
x0  = [a0 b0 c0];
[xout fval] = fminsearch(@sum_square,x0,[],xdata,ydata)
%% ----- Objective function -----
function SSE= sum_square(p,xdata,ydata)
a  = p(1);
b  = p(2);
c  = p(3);
%% Definition of the objective function
M  = fcn(xdata,a,b,c);
SSE = sum((ydata-M).^2);
% Plot every step plot(xdata,ydata,'ro',xdata,M,'b-s'),
axis([2*min(xdata) 2*max(xdata) 4*min(ydata)-2 2*max(ydata)])
pause(.01)
%% ----- Model -----
function fout = fcn(xdata,a,b,c)
fout    = a*sin(b*xdata)+c;
```

5.5. lsqcurvefit: solves non-linear least squares problems

```
X = lsqcurvefit(FUN,X0,XDATA,YDATA)
starts at X0 and finds
coefficients X to best fit
the nonlinear functions
```

EXAMPLE 46. Following code minimize the difference between data and the function

$$f = a \cos(bx) + c$$

```
% Driver
clc
% Input data set
xdata=[-5.0 -4.0 -3.0 -2.0 -1.0 0.0 1.0 2.0 3.0 4.0 5.0];
ydata =[8.5 10.8 11.9 11.5 9.7 7.0 4.3 2.4 2.0 3.2 5.5];
% Initial estimate
X0=2+rand(1,3);
% Upper and lower bounds
LB=[];
UB=[];
% call lsqcurvefit
[xout,resnorm] = lsqcurvefit(@fcfn,X0,xdata,ydata,LB,UB, [],ydata)
% the optimization function
function f=fcfn(X0,xdata,ydata)
a=X0(1);
b=X0(2);
c=X0(3);
f=a*cos(b*xdata)+c;
% plotting for every estimate
plot(xdata,ydata,'ro',xdata,f,'b')
pause(.01)
```

5.6. MATLAB: Using the Debugger

The debugger helps us understand what a program actually does (rather than what we want it to do)

- Setup breakpoints in an m-file by pressing F12, clicking on the file icon with the red dot, or clicking on the "-" in the margin of the editor.
- Execute the m-file at the command line and note the prompt change to K>>.
- `dbstep` executes the next command in the program (after the breakpoint or previous command)
- `dbcont` executes all remaining steps of the program until the next breakpoint or termination of the program.
- `dbquit` exits debugging mode

5.7. MATLAB Hints (Source: Tobin A. Driscoll 2009)

A few things things about MATLAB that can come in handy, but which often do not come to the attention of beginners:

- Use the up-arrow key to cycle through previous commands. If you type specific characters first, only commands matching the typed characters will be recalled.
- If a computation is taking too long, interrupt it by pressing Ctrl-C (after making sure the Command Window is active in the operating system).
- Even when MATLAB displays only 4–5 digits of a result, it's storing about 15 significant digits. (You can see them all by typing `format long`). By copying or retyping a displayed result, you throw away a lot of information. Wait until the end of the calculation to round off your results.
- MATLAB has great debugging tools. Run your code step by step to uncover errors. Run someone else's code step by step to understand it thoroughly.
- The code checker (`checkcode`) makes some good suggestions
- The previous two items alone are sufficient reasons to use the built-in MATLAB Editor for writing code. Open it by entering `edit`.
- If the execution of your code is too slow to suit, use the Profiler to find the slowest steps.
- Don't use a screen or window capture function to paste figures into a document or presentation. The results look cheesy and amateurish, MATLAB's use `print` command (look at the help file to get more about usage of this command)
- After you have properly exported a figure to a graphics file, save that figure again in the native FIG format. You may want to make changes to it someday.


```

clf
clc
%% ===== Part(A) =====
R = 2.9;
t = [0:.1:1];
N = length(t);
x(1)= 0.6;
for i=1:length(t)
    x(i+1) = R*x(i)*(1-x(i));%
end
subplot(2,1,1),plot(t,t,'b'), hold on;
subplot(2,1,1),plot(t,R*t.*(1-t),'r'), hold on;
axis('square');
axis([0 1 0 1]);
set(gca,'XTick',(0:.1:1),'YTick',(0:.1:1))
grid on;
xlabel('xt')
ylabel('xt+1')
%% ===== Part(B) =====
subplot(2,1,1),line([x(1) x(1)],[0 x(2)],'Color','g')
subplot(2,1,1),plot(x(1), x(1),'ko');
for ic=1:length(t)-2
subplot(2,1,1), line([x(ic) x(ic+1)],[x(ic+1) x(ic+1)],'Color','g')
subplot(2,1,1), line([x(ic+1) x(ic+1)],[x(ic+1) x(ic+2)],'Color','g')
subplot(2,1,1), plot(x(ic+1), x(ic+1),'ko');
    pause
end
line([x(N) x(N+1)],[x(N+1) x(N+1)],'Color','g')

%% ===== Part(C) =====
At = text(.1,.6,['R=',num2str(R)]); set(at,'FontSize',12);
pause(20)
subplot(2,1,2),plot(t,x(1:end-1),'b-o')
axis('square');
axis([0 1 0 1]);
xlabel('t')
ylabel('xt')

```

CHAPTER 6

SOLVING ORDINARY DIFFERENTIAL EQUATIONS in MATLAB

A MATLAB code for solving a differential equation with an initial condition (IVP) usually consists of two parts: (1) driver part and (2) differential equation part. The driver part constitutes initial values for dependent variables, parameter values, one of MATLAB's differential equation solver and a common for plotting the solution, whereas the function part consists of the differential equation. The general form of an initial value problem (IVP) is

$$\begin{aligned}\frac{dx}{dt} &= f(t, x) \\ x(t_0) &= x_0\end{aligned}$$

To solve this problem with the right hand side of the differential equation $f = f(t, x)$, the initial starting point (t_0, x_0) and a range $[t_0, t_{end}]$ for the independent variable t have to be provided. There are a number of ODE solvers in MATLAB, each using a different numerical method. Here, our focus is on the solver **ode15s**.

6.1. Solving an ODE with an inline function

This is the simplest way of solving an ODE with an initial condition is to use inline function. Consider the following example

EXAMPLE 47. Numerically approximate the solution of the first order differential equation given below

$$\frac{dx}{dt} = 0.5x, \text{ and } x(0) = 1.0$$

on the interval $x \in [0, 5]$. We begin by defining the function $f(t, x)$ in $\frac{dx}{dt} = f(t, x)$:

```
fcn=@(t,x) -.5*x;
```

The basic usage for MATLAB's solver **ode15s** is **ode15s(function, domain, initial condition)**.

That is, we use


```
[Tout,Xout]=ode15s(f,[0 .5],1);
plot(Tout,Xout)
xlabel('t')
ylabel('x(t)')
```

and MATLAB returns two column vectors, the first with values of Tout and the second with values of Xout(solution). Since the MATLAB output is long and it is omitted here. Now we can plot the results

```
plot(Tout,Xout)
xlabel('t')
ylabel('x(t)')
```

6.2. Solving an ODE with “.m” script and a function

EXAMPLE 48. Solve the following IVP for $0 \leq t \leq 10$

$$\begin{aligned}\frac{dx}{dt} &= k_1 \ln(x) + x - 1 \\ x(0) &= 0.8\end{aligned}$$

if $k_1 = -0.4$.

MATLAB code for this problem is given below.

```
%% ----- Driver part -----
clc % Clears Command Window
clf % Clears figure Window
%% Enter parameters
k1= -.4;
%% Input initial and final time points ( $T_{int}$  and  $T_{end}$ )
T_int = 0;
T_end = 10;
%% Enter Initial Values for the dependent variables
x1_0 = 0.8;
%% Call MATLAB's solver
[Tout,Xout] = ode15s(@fcn,[T_int T_end],[x1_0],[],k1);
%% Plot the solution
plot(Tout,Xout,'r')
```

```

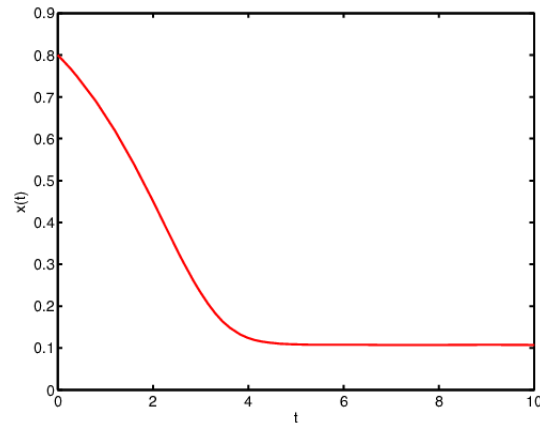
%% Specify labels for x and y axis
xlabel('t')
ylabel('x(t)')

%%----- Differential Equation part -----
function dxout=fcn(t,x,k1)
% Assign variables
x1 = x(1) % assigns x(1)to the variable x1
%% Enter the equation
dx1 = k1*log(x1)+x1-1;
dxout = [dx1]';

```

The result is shown in the plot below

FIGURE 6.2.1. The solution of the problem in Example 48



6.3. Solving a System ODEs

Below is a template you can modify to develop your code to solve a system of differential equation involving a number of parameters:

```

%% ----- Driver part -----
function Driver_fcn
clc % clears Command Window
clf % clears figure Window
%% Enter parameter values for k1,k2,k3,...
k1= ?;
k2= ?;
k3= ?;
:
%% Input initial and final time points (T_int and T_end)
T_int = ?;
T_end = ?;
%% Enter Initial Values for the dependent variables
x1_0 = ?;
x2_0 = ?;
:
%% Call MATLAB's ode15s solver
[Tout,Xout]=ode15s(@fcn,[T_int T_end],[x1_0 x2_0 ...],[],k1,k2,k3,...);
%% Plot the solution
plot(Tout,Xout,'r')
%% Specifiy labels for x and y axis
xlabel('t')
ylabel('x(t)')

%%----- Differential Equation part -----
function dxout=fcn(t,x,k1,k2,k3,...)
%% Assign variables
x1 = x(1) % assigns x(1)to the variable x
x2 = x(2) % assigns x(2)to the variable x
:
%% Enter the equations
dx1 = f1(x1,x2,...);
dx2 = f2(x1,x2,...);
:
dxout = [dx1 dx2 ... ]';

```

EXAMPLE 49. Solve the following IVP for $0 \leq t \leq 10$

$$\begin{aligned}\frac{dx_1}{dt} &= ax_1(1 - x_1) - \frac{x_1x_2}{k + x_1} \\ \frac{dx_2}{dt} &= b\frac{x_1x_2}{k + x_1} - x_2 \\ x_1(0) &= 1 \text{ and } x_2(0) = 1\end{aligned}$$

The MATLAB script and function solving this system of differential equations are given below. The figure is shown below the code.

```
%% ----- Driver part -----
clc % Clears Command Window
clf % Clears figure Window
%% Enter parameters
a = 1;
b = 3;
k = 1/2;
%% Input initial and final time points (T_{int} and T_{end} )
T_int = 0;
T_end = 100;
%% Enter Initial Values for the dependent variables
x1_0 = 1;
x2_0 = 1;
%% Call MATLAB's solver
[Tout,Xout] = ode15s(@fcn,[T_int T_end],[x1_0 x2_0],[],a,b,k);
%% Plot the solution
plot(Tout,Xout)
%% Specify labels for x and y axis
xlabel('time')
ylabel('x_1(t) and x_2(t)')

%%----- Differential Equation part -----
function dout=fcn(t,x,a,b,k)
% Assign variables
x1 = x(1);
x2 = x(2);
```

```
% Enter the equation
dx1 = a*x1*(1-x1)-x1*x2/(k+x1);
dx2 = b*x1*x2/(k+x1)-x2;
dout = [dx1 dx2]';
```

FIGURE 6.3.1. The Solution of the problem in Example 49

