**Image Processing**

**(01CE0507)**

**Department of Computer Engineering**

**5th Semester**

**Lab Manual**

**(July-Dec 2022)**

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Lab** | **Programs** | **Date** | **Signature** |
| 1 | Study of matlab image processing toolkit and various commands on matlab. |  |  |
| 2 | Point processing in spatial domain   1. Negation of an image 2. Thresholding of an image 3. Contrast Stretching of an image |  |  |
| 3 | Write a program for histogram equalization. |  |  |
| 4 | Write a program to apply various filtering techniques in matlab.   1. Low pass filtering 2. High pass filtering 3. Median filtering |  |  |
| 5 | Write a program for image segmentation   1. Local thresholding 2. Global thresholding |  |  |
| 6 | Write a program for color image processing   1. Color approximation 2. Quantization |  |  |
| 7 | Write a program, for Image restoration   1. Facial Images 2. Texture Images |  |  |
| 8 | Write a program for edge detection. |  |  |
| 9 | Write a program for smoothening and sharpening for 8-bit color image. |  |  |
| 10 | Write a program to implement morphological operations. |  |  |

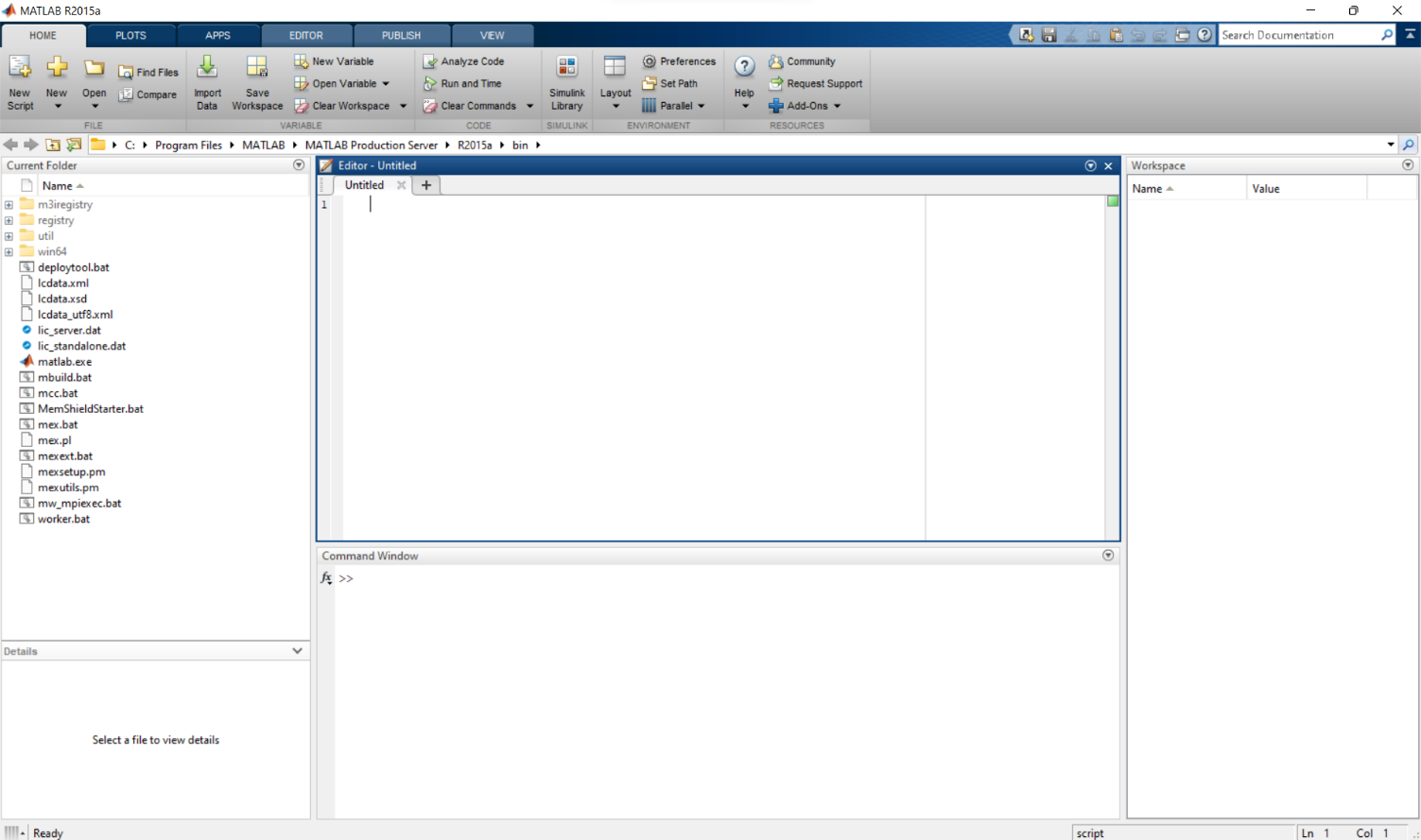
**Practical 1**

**Aim:** Study of matlab image processing toolkit and various commands on matlab.

* **Introduction to Various component of MATLAB Tool like editor, command window, workspace.**

MATLAB is a programming language developed by MathWorks. It started out as a matrix programming language where linear algebra programming was simple. It can be run both under interactive sessions and as a batch job. This tutorial gives you aggressively a gentle introduction of MATLAB programming language. It is designed to give students fluency in MATLAB programming language. Problem-based MATLAB examples have been given in simple and easy way to make your learning fast and effective.MATLAB development IDE can be launched from the icon created on the desktop. The main working window in MATLAB is called the desktop.

The desktop has the following panels −

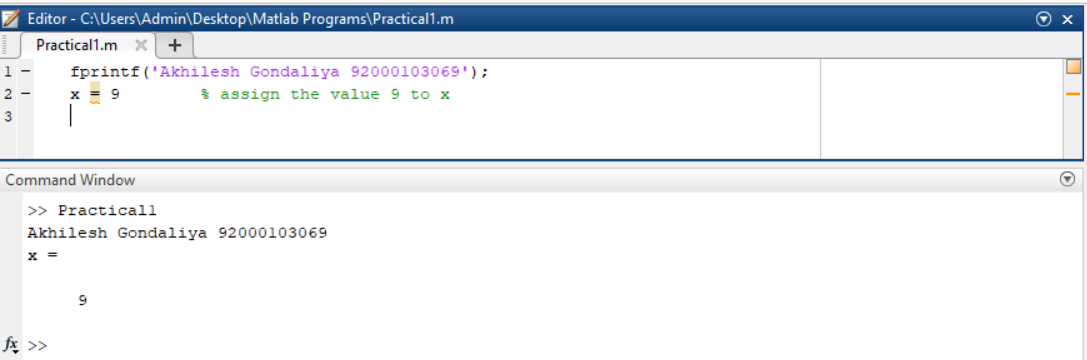
* **Current Folder** − This panel allows you to access the project folders and files.
* **Command Window** − This is the main area where commands can be entered at the command line. It is indicated by the command prompt (>>).
* **Workspace** − The workspace shows all the variables created and/or imported from files.
* **Command History** − This panel shows or return commands that are entered at the command line.

* **Basic Syntax, Variables, Commands**
  + **Use of Semicolon (;) in MATLAB**

Semicolon (;) indicates end of statement. However, if you want to suppress and hide the MATLAB output for an expression, add a semicolon after the expression.

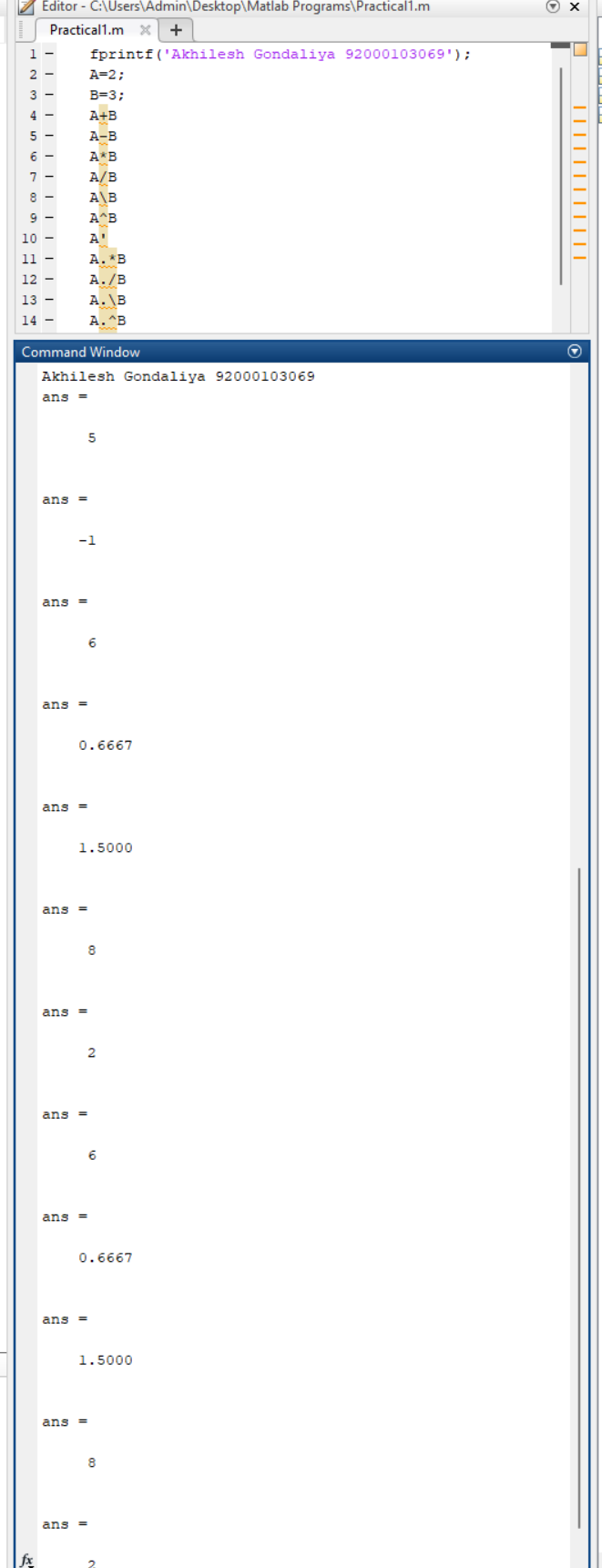
* + **Adding Comments**

The percent symbol (%) is used for indicating a comment line. You can also write a block of comments using the block comment operators % { and % }.

****

* + **Commonly used Operators and Special Characters**

|  |  |
| --- | --- |
| **Operator** | **Purpose** |
| **+** | Plus; addition operator. |
| **-** | Minus; subtraction operator. |
| **\*** | Scalar and matrix multiplication operator. |
| **.\*** | Array multiplication operator. |
| **^** | Scalar and matrix exponentiation operator. |
| **.^** | Array exponentiation operator. |
| **\** | Left-division operator. |
| **/** | Right-division operator. |
| **.\** | Array left-division operator. |
| **./** | Array right-division operator. |
| **:** | Colon; generates regularly spaced elements and represents an entire row or column. |
| **( )** | Parentheses; encloses function arguments and array indices; overrides precedence. |
| **[ ]** | Brackets; enclosures array elements. |
| **.** | Decimal point. |
| **…** | Ellipsis; line-continuation operator |
| **,** | Comma; separates statements and elements in a row |
| **;** | Semicolon; separates columns and suppresses display. |
| **%** | Percent sign; designates a comment and specifies formatting. |
| **\_** | Quote sign and transpose operator. |
| **.\_** | Nonconjugated transpose operator. |
| **=** | Assignment operator |

****

* + **Special Variables and Constants**
  + **save and load Command**

The **save** command is used for saving all the variables in the workspace, as a file with .mat extension, in the current directory.

You can reload the file anytime later using the **load** command.

* + **who and whos command**

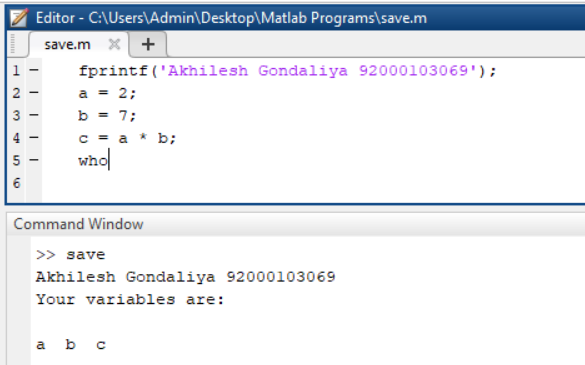
fprintf('Akhilesh Gondaliya 92000103069');

a = 2;

b = 7;

c = a \* b;

who

****

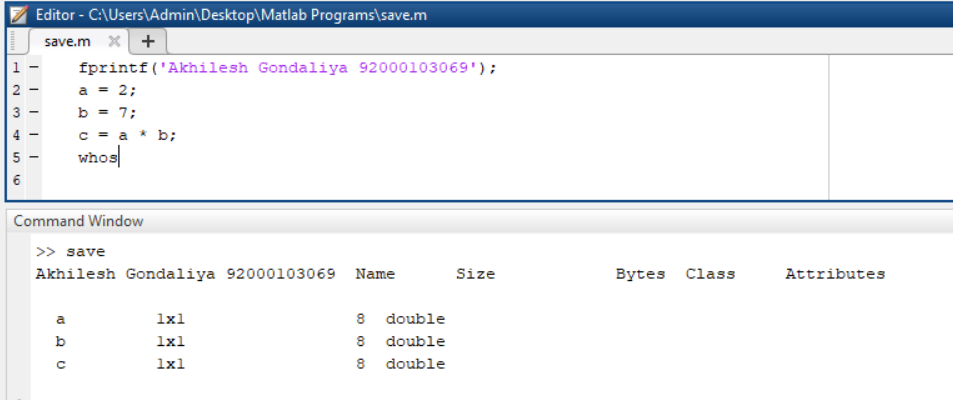
fprintf('Akhilesh Gondaliya 92000103069');

a = 2;

b = 7;

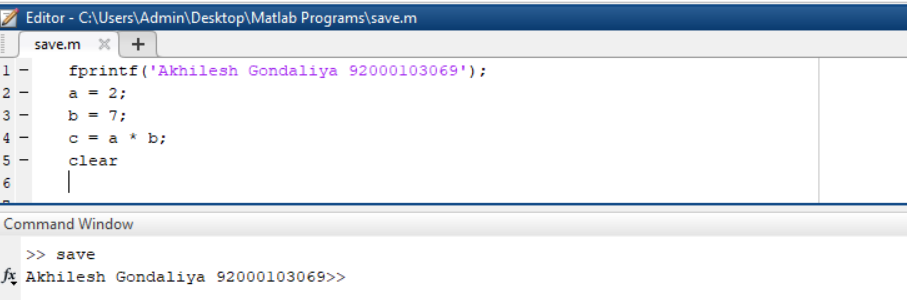
c = a \* b;

whos

****

* + **clear and clc Command**

The **clear** command deletes all (or the specified) variable(s) from the memory.

****

The **CLC** command clears the command window.

* + **disp and fprintf command**

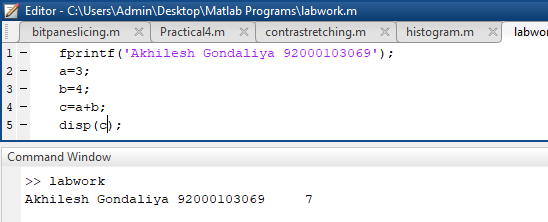
fprintf('Akhilesh Gondaliya 92000103069');

a=3;

b=4;

c=a+b;

disp(c);

****

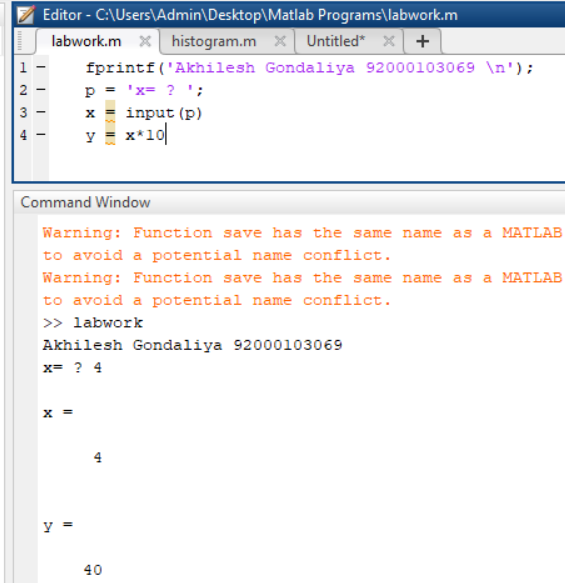
* + **input command**

fprintf('Akhilesh Gondaliya 92000103069 \n');

p = ‘x = ?';

x = input(p)

y = x\*10



* + **M-Files**

MATLAB allows writing two kinds of program files −

* **Scripts** − script files are program files with **.m extension**. In these files, you write series of commands, which you want to execute together. Scripts do not accept inputs and do not return any outputs. They operate on data in the workspace.
* **Functions** − functions files are also program files with **.m extension**. Functions can accept inputs and return outputs. Internal variables are local to the function.
  + **Plotting Commands**
* **Data Types in MATLAB**
  + **List of Data Types**

|  |  |
| --- | --- |
| **Sr.No.** | **Data Type & Description** |
| 1 | **int8** |
| 2 | **uint8** |
| 3 | **int16** |
| 4 | **uint16** |
| 5 | **int32** |
| 6 | **uint32** |
| 7 | **int64** |
| 8 | **uint64** |
| 9 | **single** |
| 10 | **double** |
| 11 | **logical** |
| 12 | **char** |
| 13 | **cell array** |
| 14 | **structure** |
| 15 | **function handle** |
| 16 | **user classes** |
| 17 | **java classes** |

* + **Data Type Conversion**

|  |  |
| --- | --- |
| **Function** | **Purpose** |
| char | Convert to character array (string) |
| int2str | Convert integer data to string |
| mat2str | Convert matrix to string |
| num2str | Convert number to string |
| str2double | Convert string to double-precision value |
| str2num | Convert string to number |
| native2unicode | Convert numeric bytes to Unicode characters |
| unicode2native | Convert Unicode characters to numeric bytes |
| base2dec | Convert base N number string to decimal number |
| bin2dec | Convert binary number string to decimal number |
| dec2base | Convert decimal to base N number in string |
| dec2bin | Convert decimal to binary number in string |
| dec2hex | Convert decimal to hexadecimal number in string |
| hex2dec | Convert hexadecimal number string to decimal number |
| hex2num | Convert hexadecimal number string to double-precision number |
| num2hex | Convert singles and doubles to IEEE hexadecimal strings |
| cell2mat | Convert cell array to numeric array |
| cell2struct | Convert cell array to structure array |
| cellstr | Create cell array of strings from character array |
| mat2cell | Convert array to cell array with potentially different sized cells |
| num2cell | Convert array to cell array with consistently sized cells |
| struct2cell | Convert structure to cell array |

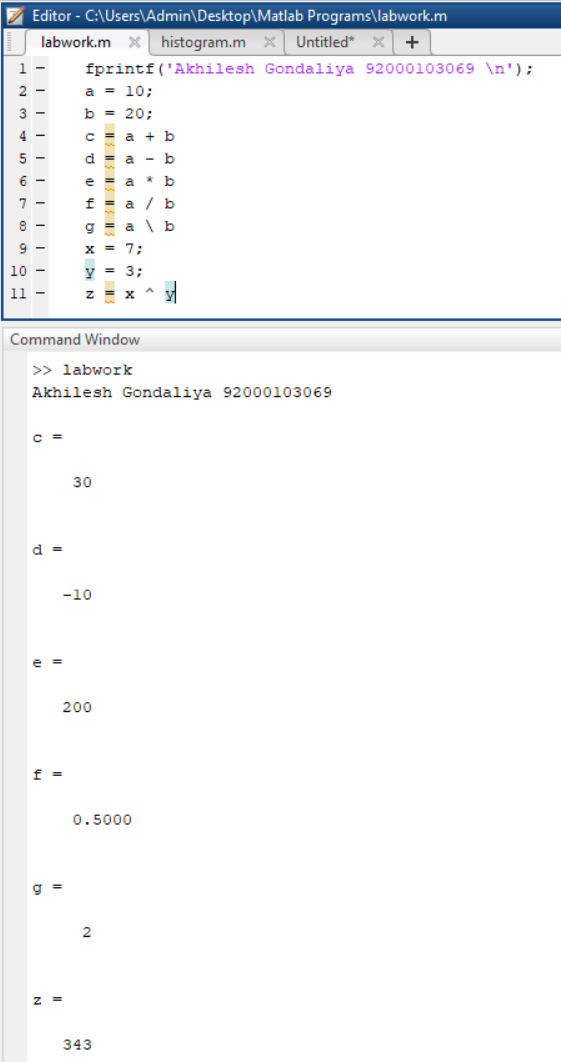
* + **Determination of Data Types**

|  |  |
| --- | --- |
| **Function** | **Purpose** |
| is | Detect state |
| isa | Determine if input is object of specified class |
| iscell | Determine whether input is cell array |
| iscellstr | Determine whether input is cell array of strings |
| ischar | Determine whether item is character array |
| isfield | Determine whether input is structure array field |
| isfloat | Determine if input is floating-point array |
| ishghandle | True for Handle Graphics object handles |
| isinteger | Determine if input is integer array |
| isjava | Determine if input is Java object |
| islogical | Determine if input is logical array |
| isnumeric | Determine if input is numeric array |
| isobject | Determine if input is MATLAB object |
| isreal | Check if input is real array |
| isscalar | Determine whether input is scalar |
| isstr | Determine whether input is character array |
| isstruct | Determine whether input is structure array |
| isvector | Determine whether input is vector |
| class | Determine class of object |
| validateattributes | Check validity of array |
| whos | List variables in workspace, with sizes and types |

* **Operators in MATLAB**
  + **Types of Operators**

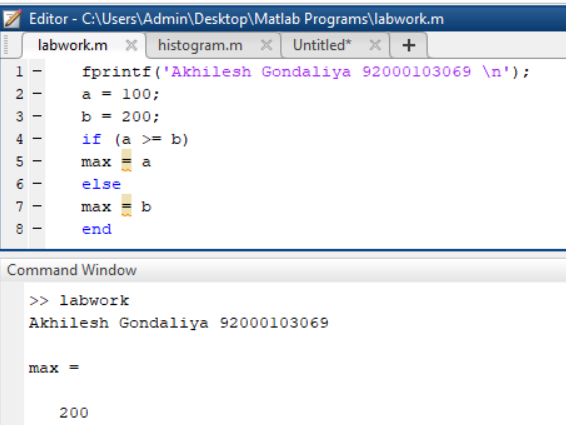
1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operations
5. Set Operations
   * **Arithmetic Operators**

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | **+**  Addition or unary plus. A+B adds the values stored in variables A and B. A and B must have the same size, unless one is a scalar. A scalar can be added to a matrix of any size. |
| 2 | **-**  Subtraction or unary minus. A-B subtracts the value of B from A. A and B must have the same size, unless one is a scalar. A scalar can be subtracted from a matrix of any size. |
| 3 | **\***  Matrix multiplication. C = A\*B is the linear algebraic product of the matrices A and B. More precisely,  Matrix Multiplication  For non-scalar A and B, the number of columns of A must be equal to the number of rows of B. A scalar can multiply a matrix of any size. |
| 4 | **.\***  Array multiplication. A.\*B is the element-by-element product of the arrays A and B. A and B must have the same size, unless one of them is a scalar. |
| 5 | **/**  Slash or matrix right division. B/A is roughly the same as B\*inv(A). More precisely, B/A = (A'\B')'. |
| 6 | **./**  Array right division. A./B is the matrix with elements A(i,j)/B(i,j). A and B must have the same size, unless one of them is a scalar. |
| 7 | **\**  Backslash or matrix left division. If A is a square matrix, A\B is roughly the same as inv(A)\*B, except it is computed in a different way. If A is an n-by-n matrix and B is a column vector with n components, or a matrix with several such columns, then X = A\B is the solution to the equation *AX = B*. A warning message is displayed if A is badly scaled or nearly singular. |
| 8 | **.\**  Array left division. A.\B is the matrix with elements B(i,j)/A(i,j). A and B must have the same size, unless one of them is a scalar. |
| 9 | **^**  Matrix power. X^p is X to the power p, if p is a scalar. If p is an integer, the power is computed by repeated squaring. If the integer is negative, X is inverted first. For other values of p, the calculation involves eigenvalues and eigenvectors, such that if [V,D] = eig(X), then X^p = V\*D.^p/V. |
| 10 | **.^**  Array power. A.^B is the matrix with elements A(i,j) to the B(i,j) power. A and B must have the same size, unless one of them is a scalar. |
| 11 | **'**  Matrix transpose. A' is the linear algebraic transpose of A. For complex matrices, this is the complex conjugate transpose. |
| 12 | **.'**  Array transpose. A.' is the array transpose of A. For complex matrices, this does not involve conjugation. |

****

* + **Relational Operators**

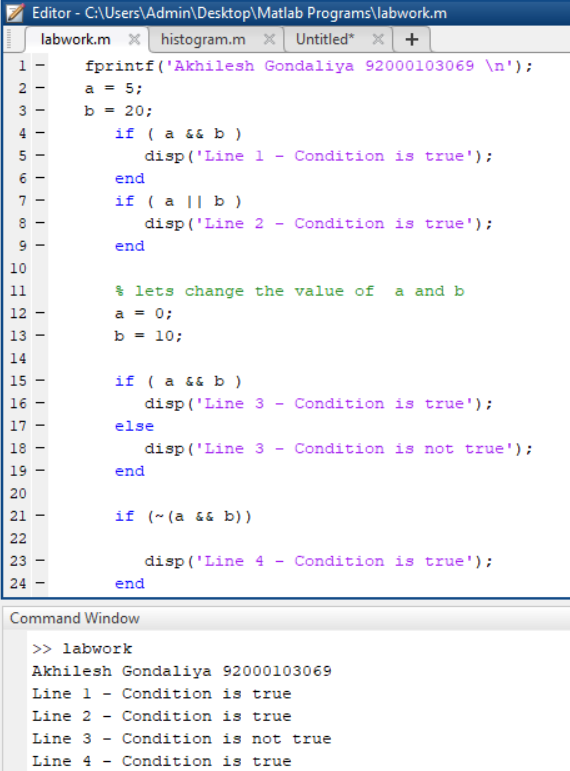
|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | **<**  Less than |
| 2 | **<=**  Less than or equal to |
| 3 | **>**  Greater than |
| 4 | **>=**  Greater than or equal to |
| 5 | **==**  Equal to |
| 6 | **~=**  Not equal to |

****

* + **Logical Operators**

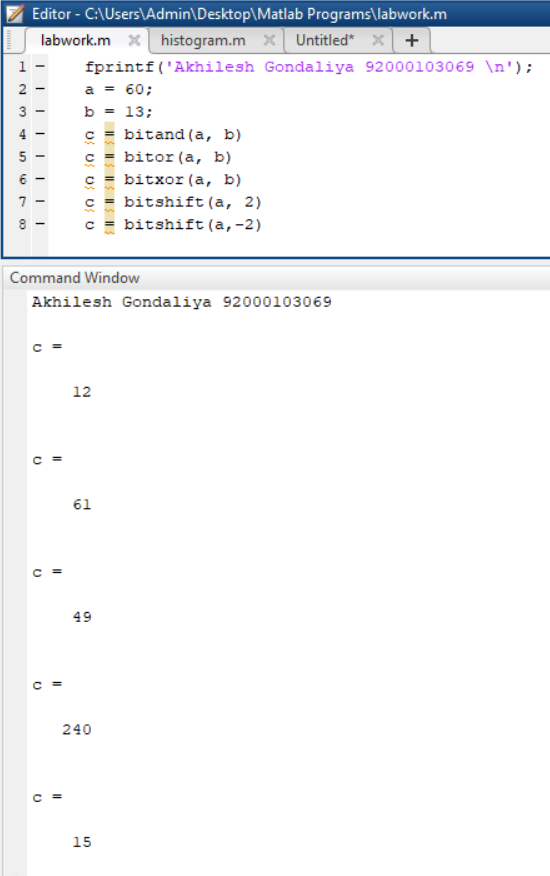
MATLAB offers two types of logical operators and functions −

* Element-wise − These operators operate on corresponding elements of logical arrays.
* Short-circuit − These operators operate on scalar and, logical expressions.



* + **Bitwise Operations**

|  |  |
| --- | --- |
| **Function** | **Purpose** |
| bitand(a, b) | Bit-wise AND of integers *a* and *b* |
| bitcmp(a) | Bit-wise complement of *a* |
| bitget(a,pos) | Get bit at specified position *pos*, in the integer array *a* |
| bitor(a, b) | Bit-wise OR of integers *a* and *b* |
| bitset(a, pos) | Set bit at specific location *pos* of *a* |
| bitshift(a, k) | Returns *a* shifted to the left by *k* bits, equivalent to multiplying by 2k. Negative values of k correspond to shifting bits right or dividing by 2|k| and rounding to the nearest integer towards negative infinite. Any overflow bits are truncated. |
| bitxor(a, b) | Bit-wise XOR of integers *a* and *b* |
| swapbytes | Swap byte ordering |

****

* **Decision Making in MATLAB**
  + **if ... end statement**

fprintf('Akhilesh Gondaliya 92000103069 \n');

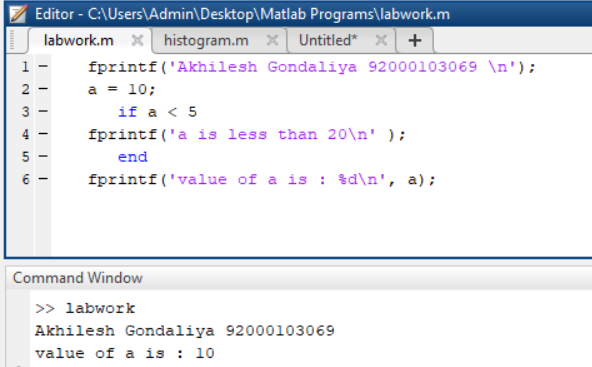
a = 10;

if a < 5

fprintf('a is less than 20\n' );

end

fprintf('value of a is : %d\n', a);

****

* + **if...else...end statement**

fprintf('Akhilesh Gondaliya 92000103069 \n');

a = 10;

if a < 50

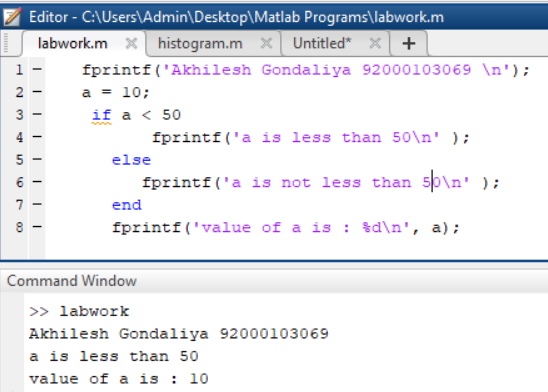
fprintf('a is less than 50\n' );

else

fprintf('a is not less than 50\n' );

end

fprintf('value of a is : %d\n', a);

****

* + **If... elseif...elseif...else...end statements**

fprintf('Akhilesh Gondaliya 92000103069 \n');

a = 10;

if a == 60

fprintf('Value of a is 60\n' );

elseif( a == 40 )

fprintf('Value of a is 40\n' );

elseif a == 70

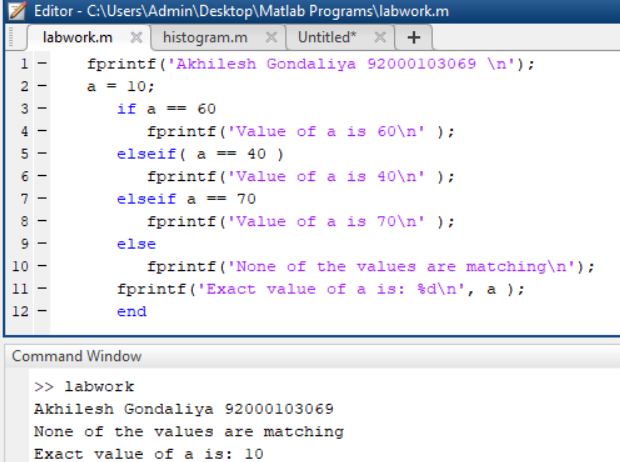
fprintf('Value of a is 70\n' );

else

fprintf('None of the values are matching\n');

fprintf('Exact value of a is: %d\n', a );

end

****

* + **nested if statements**

fprintf('Akhilesh Gondaliya 92000103069 \n');

a = 60;

b = 40;

if( a == 60 )

if( b == 40 )

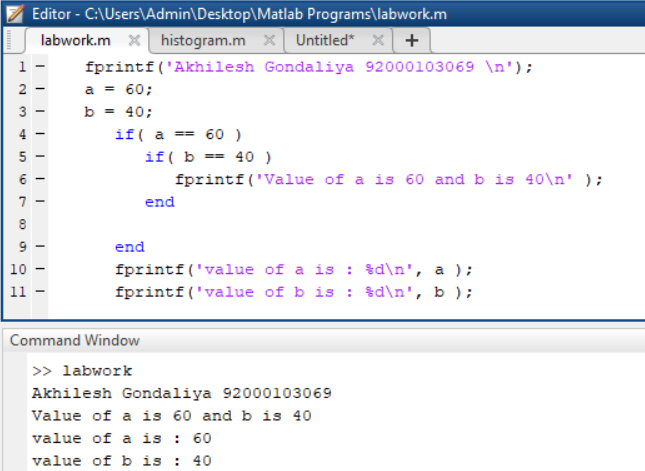
fprintf('Value of a is 60 and b is 40\n' );

end

end

fprintf('value of a is : %d\n', a );

fprintf('value of b is : %d\n', b );

****

* + **switch statement**

fprintf('Akhilesh Gondaliya 92000103069 \n');

grade = 'B';

switch(grade)

case 'A'

fprintf('Excellent!\n' );

case 'B'

fprintf('Well done\n' );

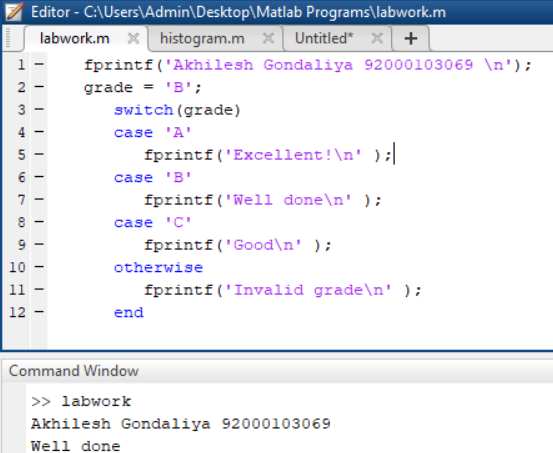
case 'C'

fprintf('Good\n' );

otherwise

fprintf('Invalid grade\n' );

end

****

* **Loop Types in MATLAB**
  + **while loop**

fprintf('Akhilesh Gondaliya 92000103069 \n');

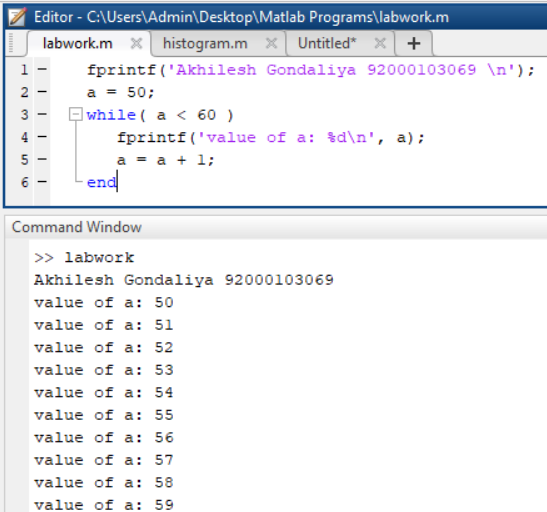
a = 50;

while( a < 60 )

fprintf('value of a: %d\n', a);

a = a + 1;

end



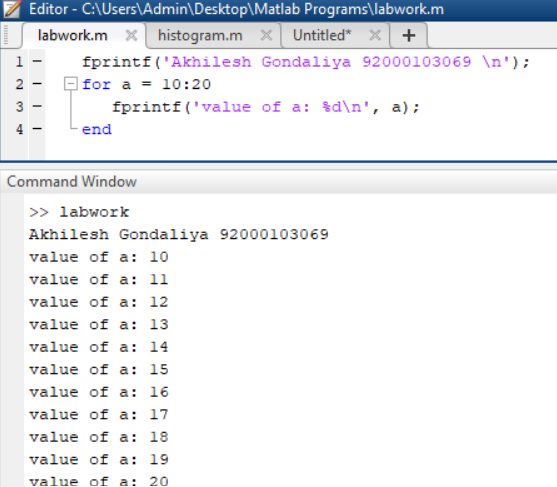
* + **for loop**

fprintf('Akhilesh Gondaliya 92000103069 \n');

for a = 10:20

fprintf('value of a: %d\n', a);

end

****

* + **nested loops**

fprintf('Akhilesh Gondaliya 92000103069 \n');

for i = 50:100

for j = 51:100

if(~mod(i,j))

break;

end

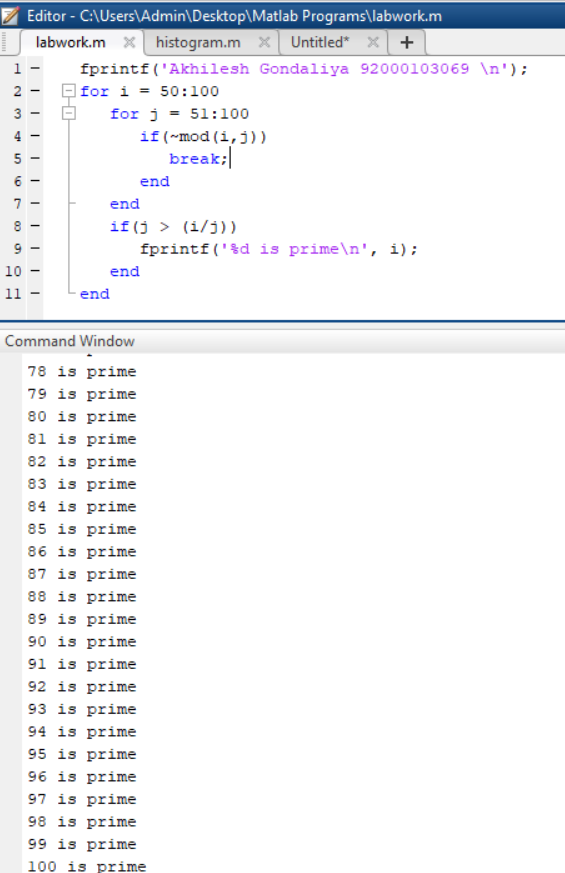
end

if(j > (i/j))

fprintf('%d is prime\n', i);

end

end

****

* + **break statement**

fprintf('Akhilesh Gondaliya 92000103069 \n');

a = 5;

while (a < 20 )

fprintf('value of a: %d\n', a);

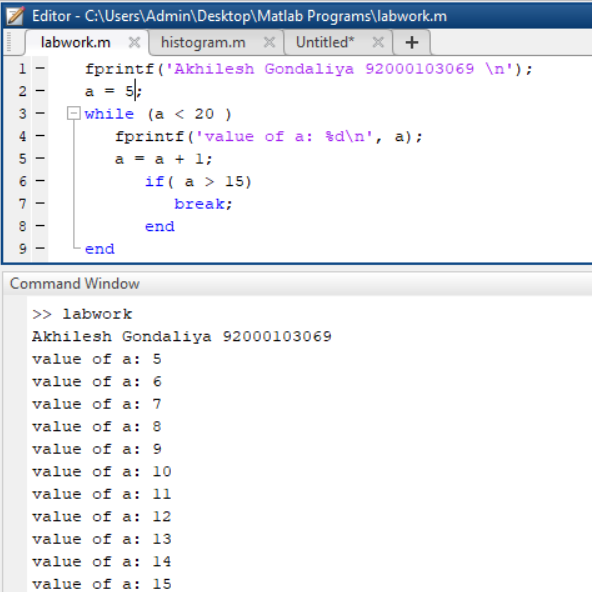
a = a + 1;

if( a > 15)

break;

end

end

****

* + **continue statement**

fprintf('Akhilesh Gondaliya 92000103069 \n');

a = 9;

while a < 20

a = a + 1;

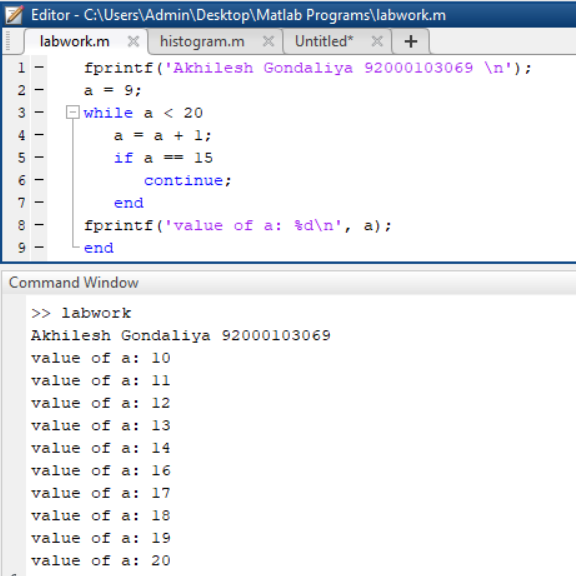
if a == 15

continue;

end

fprintf('value of a: %d\n', a);

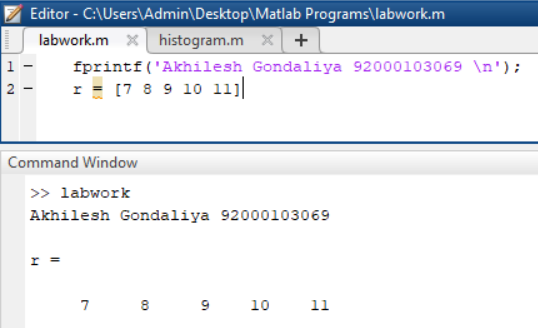
end

****

* **Vectors in MATLAB**
  + **Row Vectors**

fprintf('Akhilesh Gondaliya 92000103069 \n');

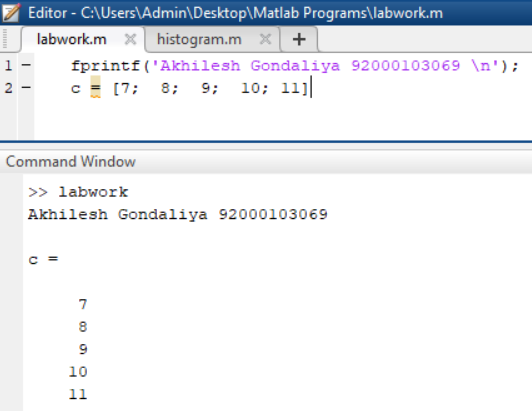
r = [7 8 9 10 11]

****

* + **Column Vectors**

fprintf('Akhilesh Gondaliya 92000103069 \n');

c = [7; 8; 9; 10; 11]

****

* + **Referencing the Elements of a Vector**

fprintf('Akhilesh Gondaliya 92000103069 \n');

v = [ 1; 2; 3; 4; 5; 6];

v(3)

****

* + **Vector Operations**
    - **Addition and Subtraction of Vectors**

fprintf('Akhilesh Gondaliya 92000103069 \n');

A = [7, 11, 15, 23, 9];

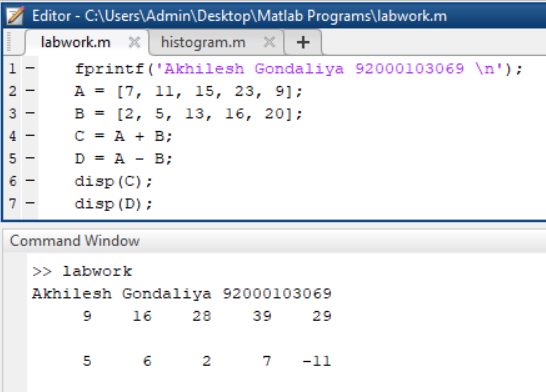
B = [2, 5, 13, 16, 20];

C = A + B;

D = A - B;

disp(C);

disp(D);

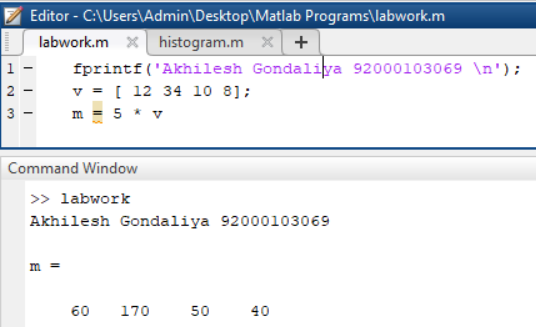
****

* + - **Scalar Multiplication of Vectors**

fprintf('Akhilesh Gondaliya 92000103069 \n');

v = [ 12 34 10 8];

m = 5 \* v

****

* + - **Transpose of a Vector**

fprintf('Akhilesh Gondaliya 92000103069 \n');

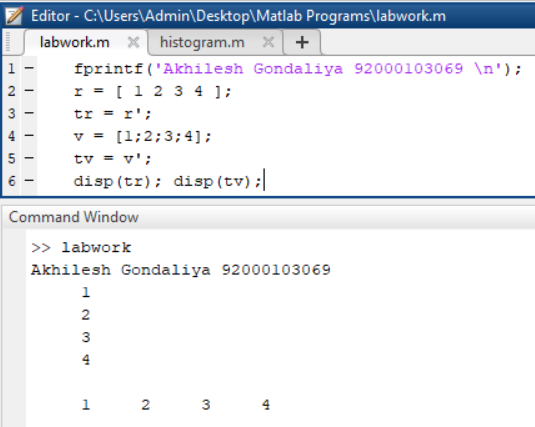
r = [ 1 2 3 4 ];

tr = r';

v = [1;2;3;4];

tv = v';

disp(tr); disp(tv);



* + - **Appending Vectors**

fprintf('Akhilesh Gondaliya 92000103069 \n');

r1 = [ 1 2 3 4 ];

r2 = [5 6 7 8 ];

r = [r1,r2]

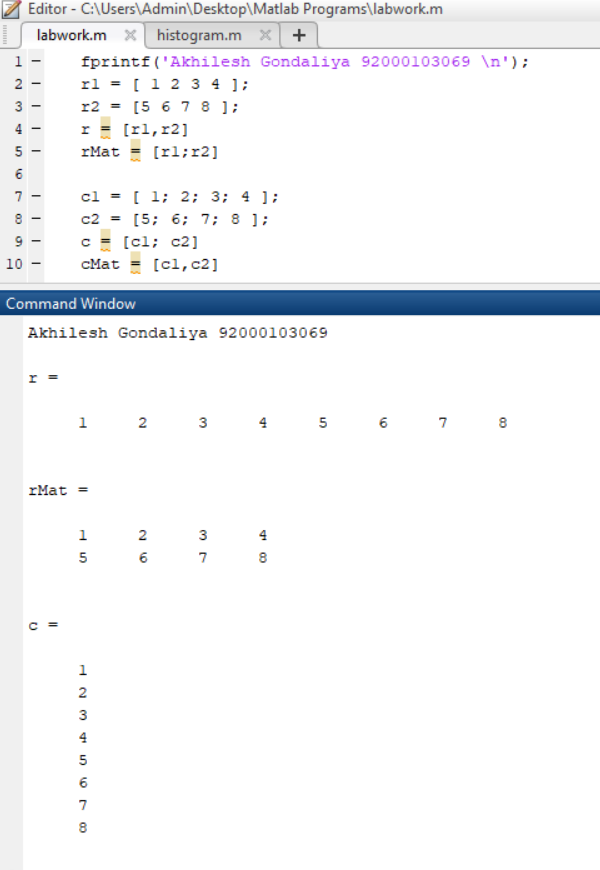
rMat = [r1;r2]

c1 = [ 1; 2; 3; 4 ];

c2 = [5; 6; 7; 8 ];

c = [c1; c2]

cMat = [c1,c2]

****

* + - **Magnitude of a Vector**

fprintf('Akhilesh Gondaliya 92000103069 \n');

v = [1: 2: 20];

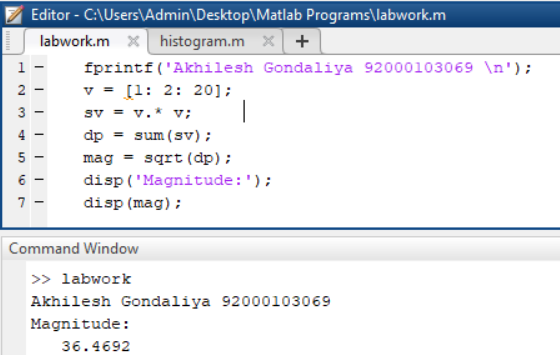
sv = v.\* v;

dp = sum(sv);

mag = sqrt(dp);

disp('Magnitude:');

disp(mag);



* + - **Vector Dot Product**

fprintf('Akhilesh Gondaliya 92000103069 \n');

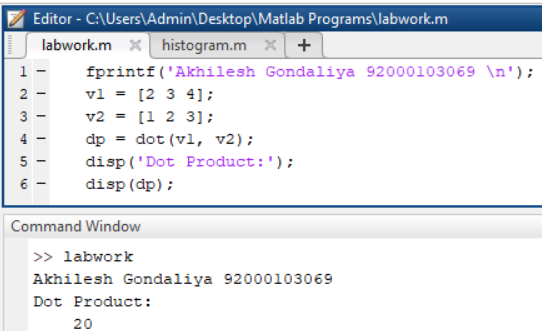
v1 = [2 3 4];

v2 = [1 2 3];

dp = dot(v1, v2);

disp('Dot Product:');

disp(dp);

****

* + - **Vectors with Uniformly Spaced Elements**

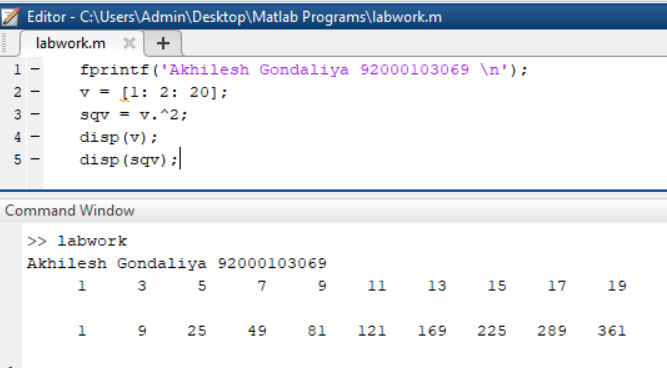
fprintf('Akhilesh Gondaliya 92000103069 \n');

v = [1: 2: 20];

sqv = v.^2;

disp(v);

disp(sqv);

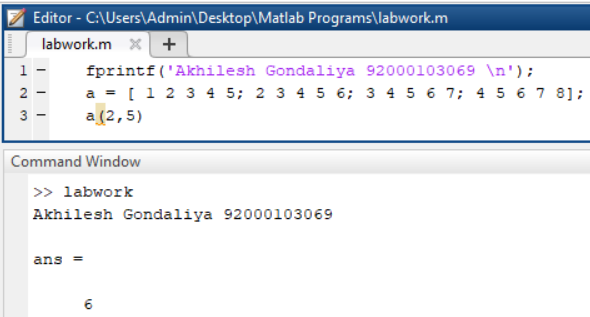
****

* **Matrix in MATLAB**
  + **Referencing the Elements of a Matrix**

fprintf('Akhilesh Gondaliya 92000103069 \n');

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

a(2,5)



* + **Matrix Operations**
    - **Addition and Subtraction of Matrices**

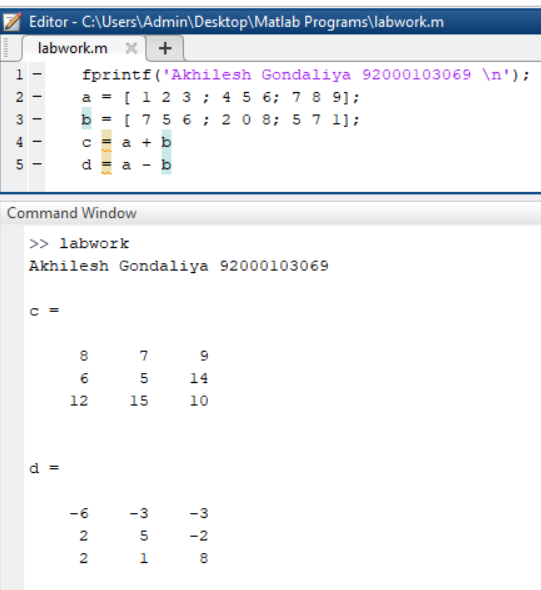
fprintf('Akhilesh Gondaliya 92000103069 \n');

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

b = [ 7 5 6 ; 2 0 8; 5 7 1];

c = a + b

d = a – b



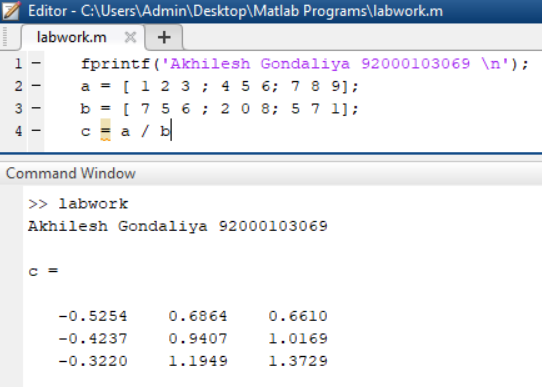
* + - **Division of Matrices**

fprintf('Akhilesh Gondaliya 92000103069 \n');

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

b = [ 7 5 6 ; 2 0 8; 5 7 1];

c = a / b

****

* + - **Scalar Operations of Matrices**

fprintf('Akhilesh Gondaliya 92000103069 \n');

a = [ 10 12 23 ; 14 8 6; 27 8 9];

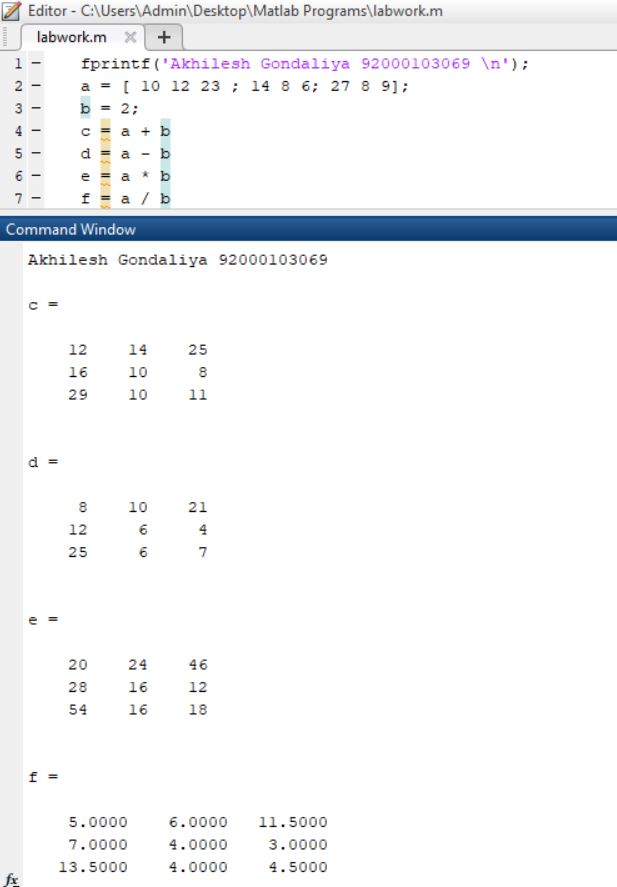
b = 2;

c = a + b

d = a - b

e = a \* b

f = a / b

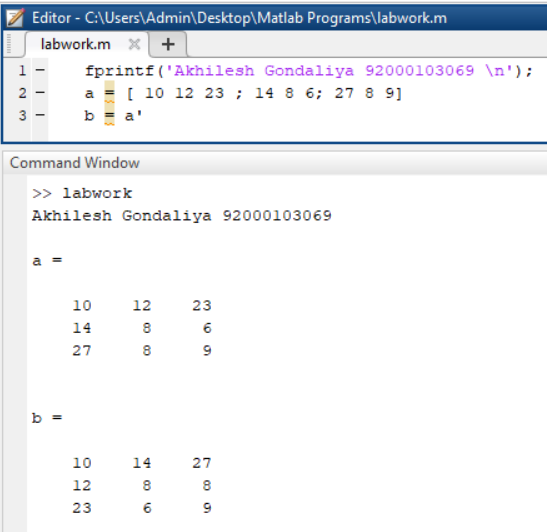


* + - **Transpose of a Matrix**

fprintf('Akhilesh Gondaliya 92000103069 \n');

a = [ 10 12 23 ; 14 8 6; 27 8 9]

b = a'



* + - **Concatenating Matrices**

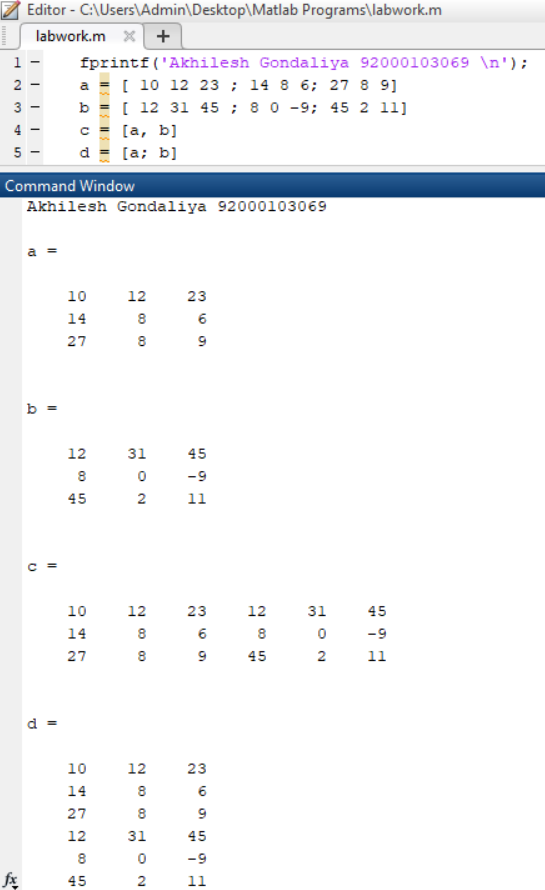
fprintf('Akhilesh Gondaliya 92000103069 \n');

a = [ 10 12 23 ; 14 8 6; 27 8 9]

b = [ 12 31 45 ; 8 0 -9; 45 2 11]

c = [a, b]

d = [a; b]



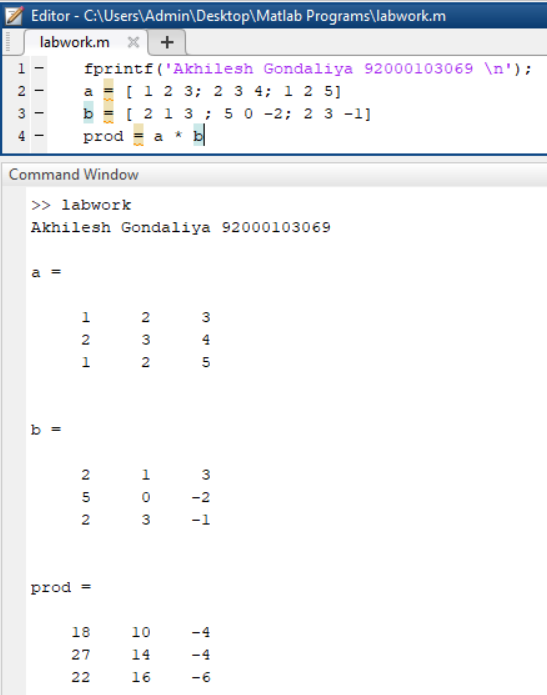
* + - **Matrix Multiplication**

fprintf('Akhilesh Gondaliya 92000103069 \n');

a = [ 1 2 3; 2 3 4; 1 2 5]

b = [ 2 1 3 ; 5 0 -2; 2 3 -1]

prod = a \* b

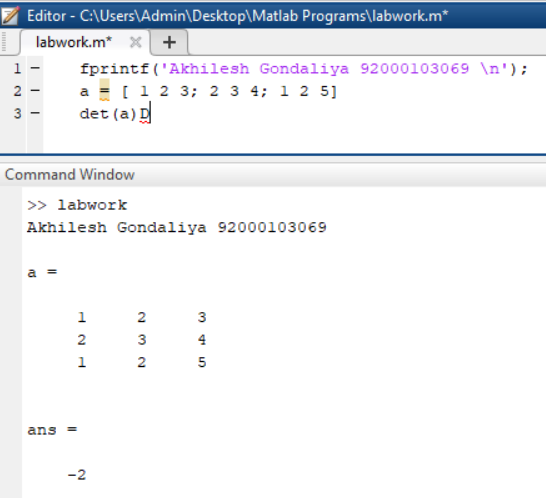


* + - **Determinant of a Matrix**

fprintf('Akhilesh Gondaliya 92000103069 \n');

a = [ 1 2 3; 2 3 4; 1 2 5]

det(a)

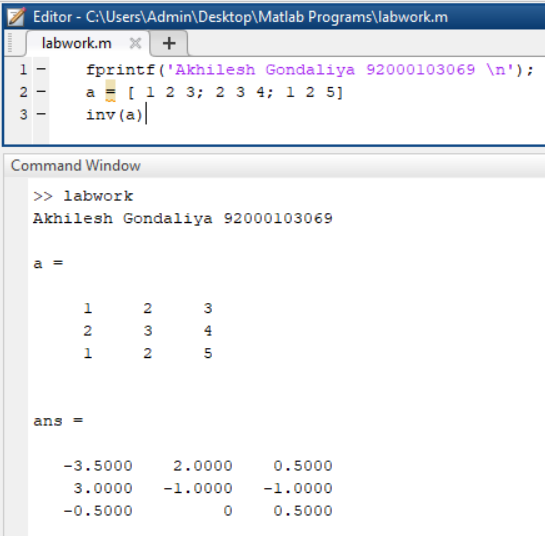


* + - **Inverse of a Matrix**

fprintf('Akhilesh Gondaliya 92000103069 \n');

a = [ 1 2 3; 2 3 4; 1 2 5]

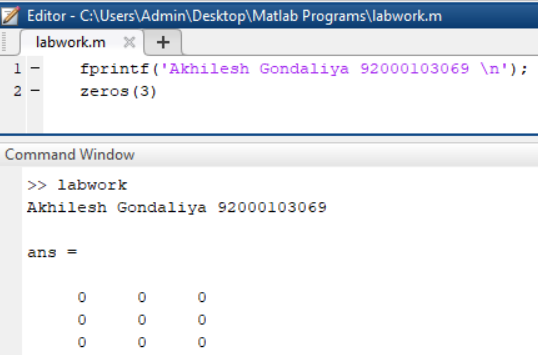
inv(a)



* **Arrays in MATLAB**
  + **Special Arrays in MATLAB**
    - **zeros( ) function**

fprintf('Akhilesh Gondaliya 92000103069 \n');

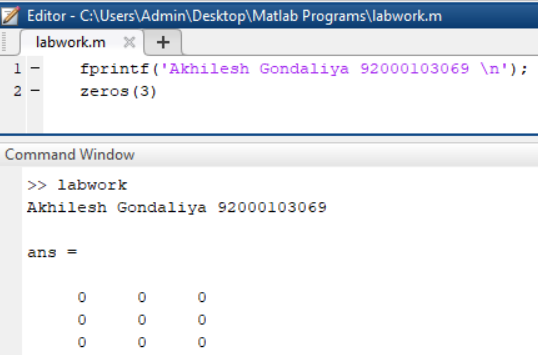
zeros(3)



* + - **ones( ) function**

fprintf('Akhilesh Gondaliya 92000103069 \n');

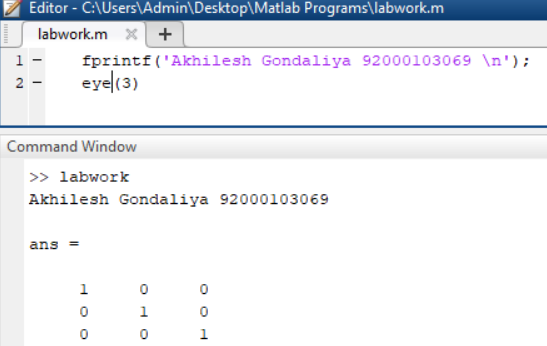
ones(3,2)

****

* + - **eye( ) function**

fprintf('Akhilesh Gondaliya 92000103069 \n');

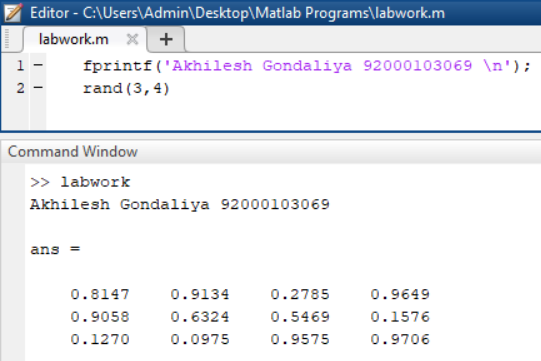
eye(3)

****

* + - **rand( ) function**

fprintf('Akhilesh Gondaliya 92000103069 \n');

rand(3,4)

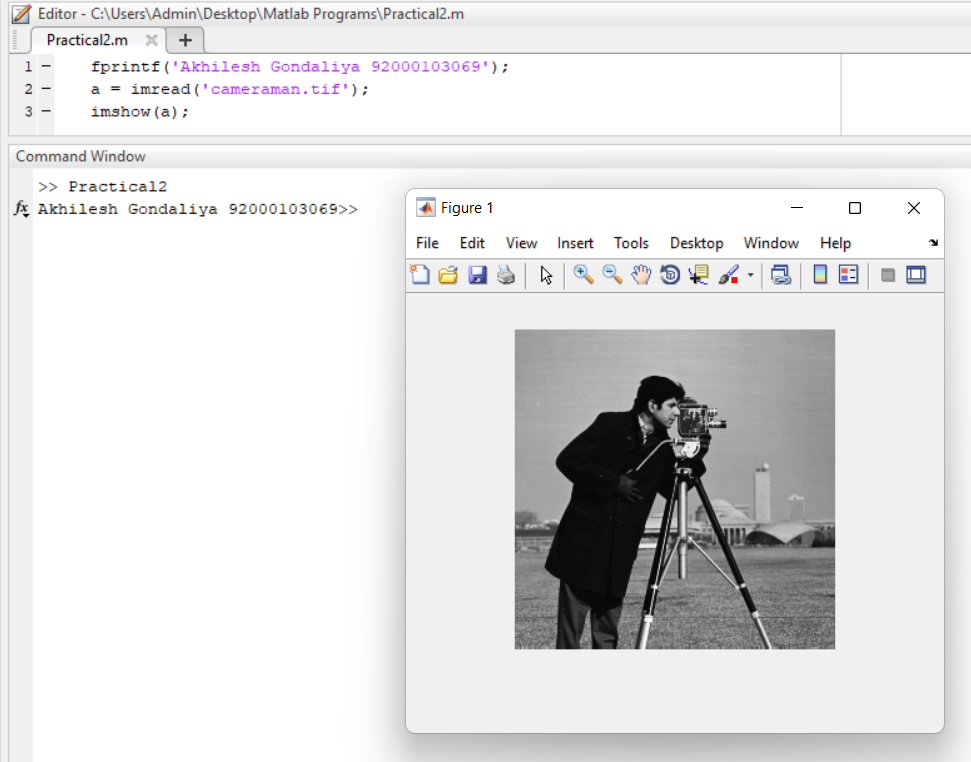


* **Image Processing Toolkit**
  + **imread( ) and imshow( ) command**

fprintf('Akhilesh Gondaliya 92000103069');

a = imread('cameraman.tif');

imshow(a);

****

* + **imwrite( ) command**
  + **size( ) command**
  + **imfinfo( ) command**
  + **impixel( ) command**
  + **subplot( ) command**
  + **imagesc( ) command**
  + **imresize( ) command**
  + **imcrop( ) command**
  + **im2bw( ) command**
  + **rgb2gray( ) command**
  + **grayslice( ) command**
  + **imadd( ) command**
  + **imsubtract( ) command**
  + **imdivide( ) command**
  + **immultiply( ) command**
  + **imcomplement( ) command**
  + **uigetfile( ) command**

**Practical 2**

**Aim:** Point processing in spatial domain

1. Negation of an image
2. Thresholding of an image
3. Contrast Stretching of an image
4. **Negation of an image**

**Code:**

fprintf('Akhilesh Gondaliya 92000103069')

A = imread('cameraman.tif');

B=A;

subplot(1,3,1);

imshow(A);

title('Original');

for row=1:size(A,1)

for col=1:size(A,2)

B(row,col)=255-A(row,col);

end

end

subplot(1,3,2);

imshow(B);

title('Negative Tansformation');

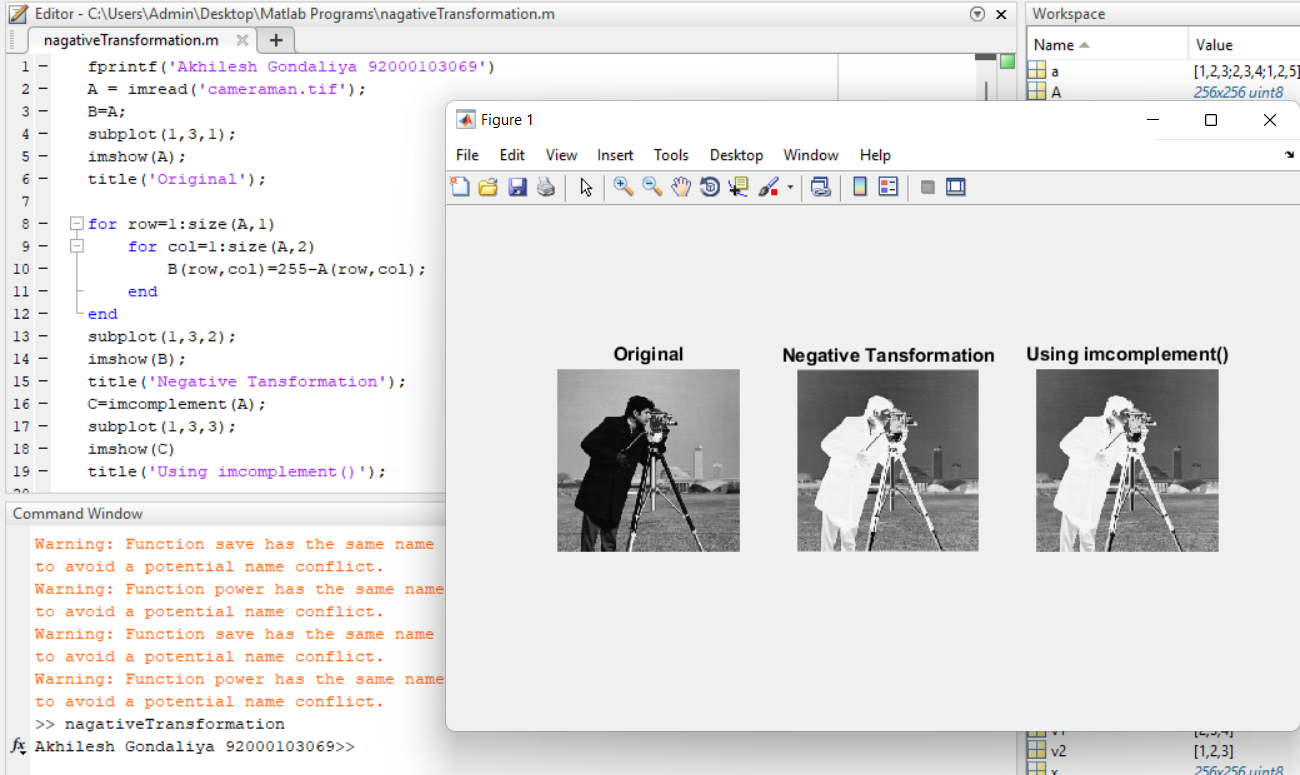
C=imcomplement(A);

subplot(1,3,3);

imshow(C)

title('Using imcomplement()');

**Output:**



1. **Thresholding of an image**

**Code:**

fprintf('Akhilesh Gondaliya 92000103069')

A = imread('cameraman.tif');

subplot(1,2,1);

imshow(A);

title('Original');

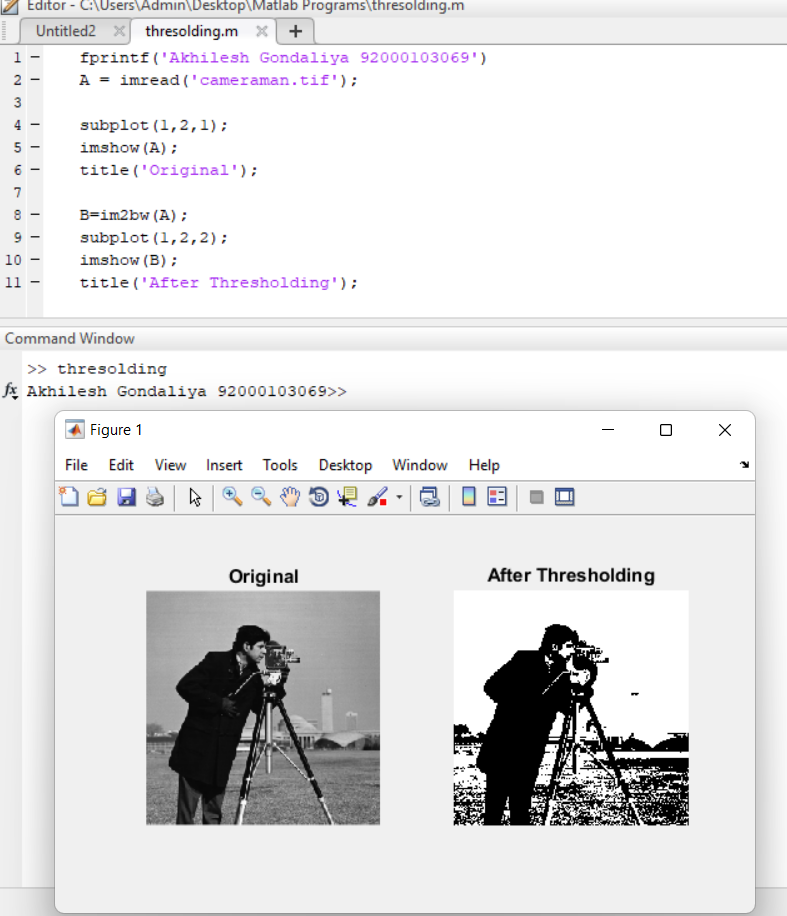
B=im2bw(A);

subplot(1,2,2);

imshow(B);

title('After Thresholding');

**Output:**



1. **Contrast Stretching of an image**

**Code:**

fprintf('Akhilesh Gondaliya 92000103069')

A = imread('kids.tif');

subplot(2,2,1);

imshow(A);

title('Original Image');

B=1./(50./(double(A)+eps).^20);

subplot(2,2,2);

imshow(B);

title('Contrast Stretching 50');

B=1./(100./(double(A)+eps).^20);

subplot(2,2,3);

imshow(B);

title('Contrast Stretching 100');

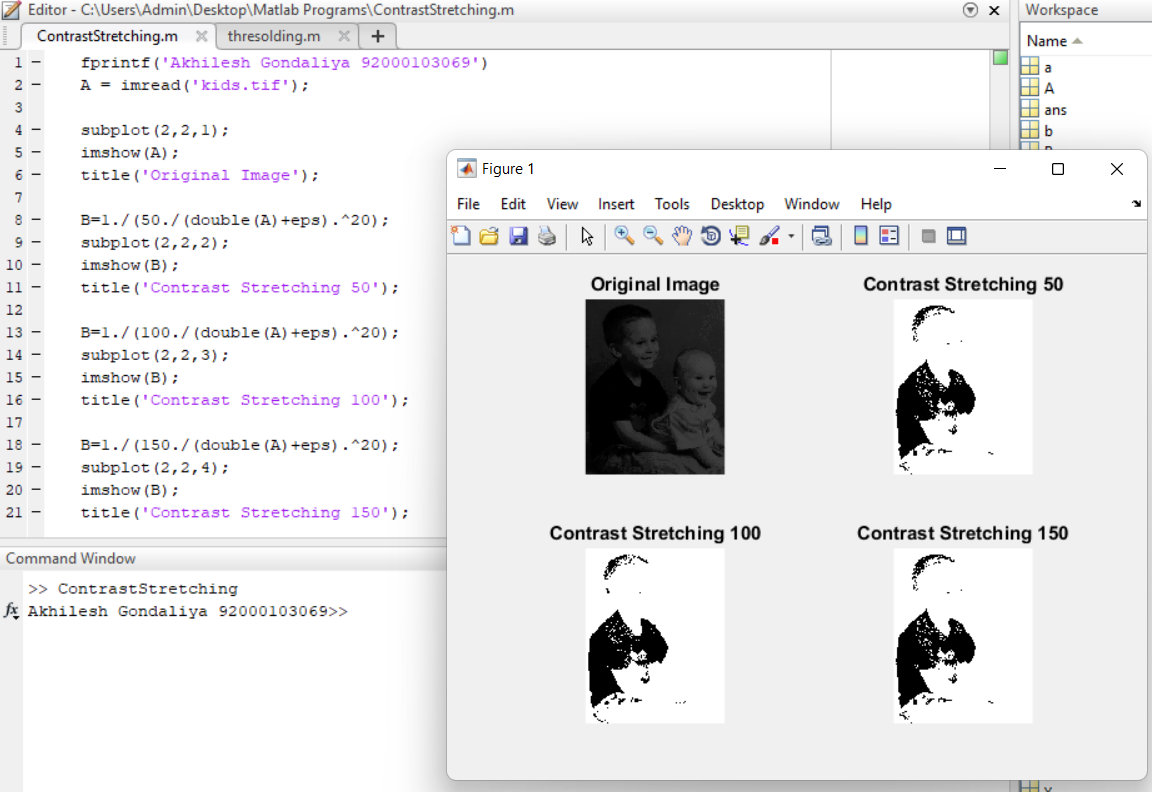
B=1./(150./(double(A)+eps).^20);

subplot(2,2,4);

imshow(B);

title('Contrast Stretching 150');

**Output:**

****

**Extra:**

1. Log Transformation
2. Power-Law Functions
3. Gray-level slicing
4. Bit-plane slicing
5. **Log Transformation**

**Code:**

fprintf('Akhilesh Gondaliya 92000103069')

A = imread('cameraman.tif');

subplot(1,3,1);

imshow(A);

title('Original Image');

B=log(1+double(A));

subplot(1,3,2);

imshow(B);

title('Intermediate Image');

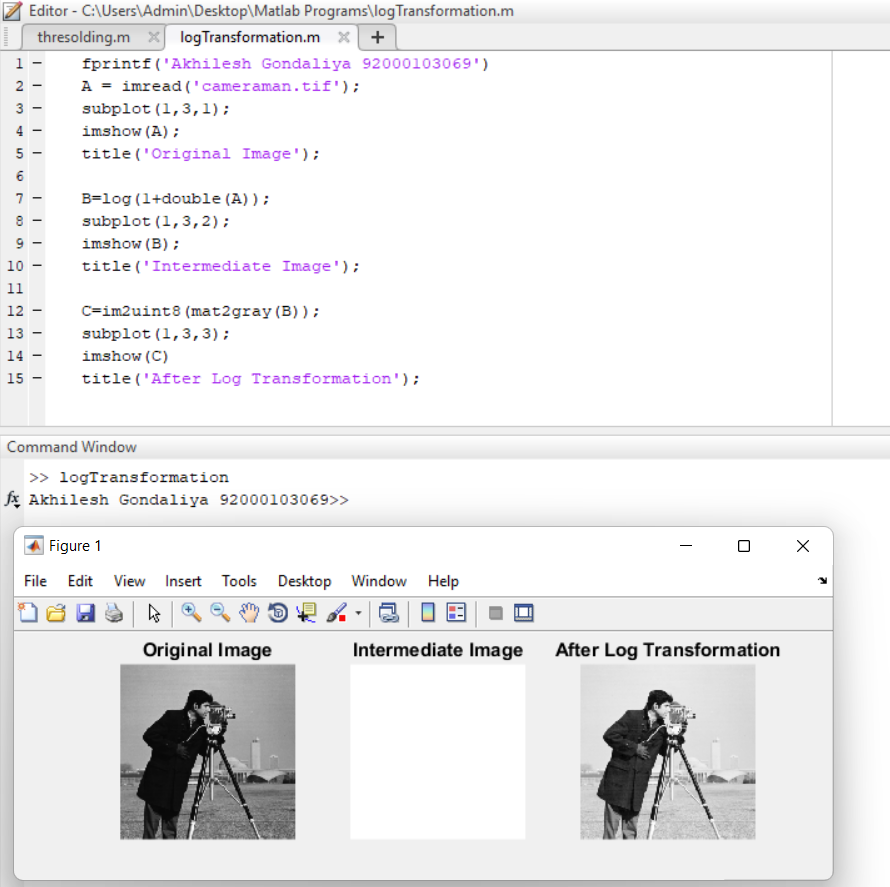
C=im2uint8(mat2gray(B));

subplot(1,3,3);

imshow(C)

title('After Log Transformation');

**Output:**

****

1. **Power-Law Functions**

**Code:**

fprintf('Akhilesh Gondaliya 92000103069');

subplot(2,3,1)

a=imread('cameraman.tif');

imshow(a);

title('Actual Image');

subplot(2,3,2)

g = imadjust(a,[0 1],[1 0]);

imshow(g);

title('R=1');

subplot(2,3,3)

f = imadjust(a,[0.5 0.75],[0 1],0.5);

imshow(f);

title('R<1');

subplot(2,3,4)

d = imadjust(a,[0.5 0.75],[0.6 1],0.5);

imshow(d);

title('R<1');

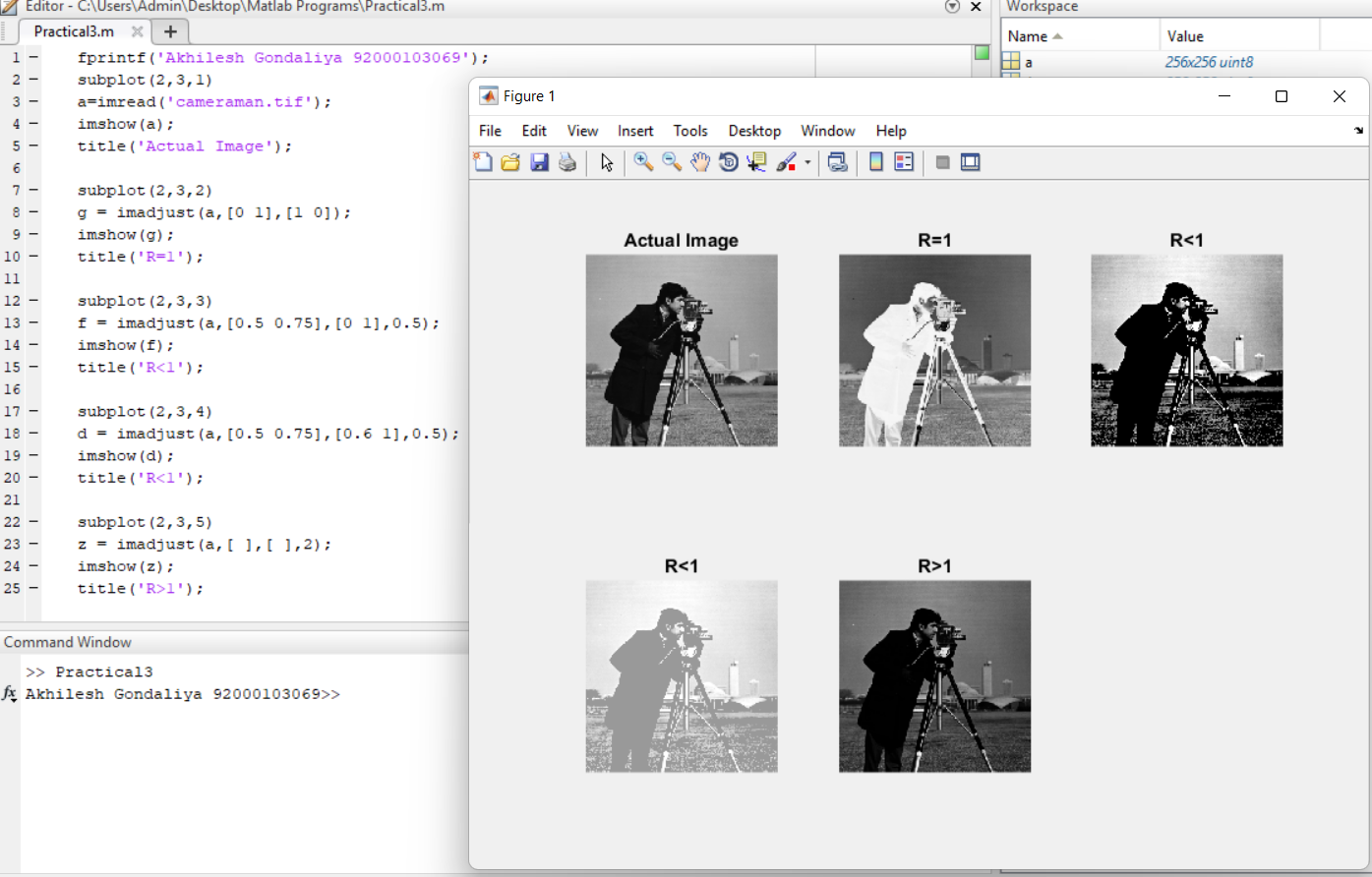
subplot(2,3,5)

z = imadjust(a,[ ],[ ],2);

imshow(z);

title('R>1');

**Output:**

****

1. **Gray-level slicing**

**Code:**

fprintf('Akhilesh Gondaliya 92000103069');

A = imread('cameraman.tif');

B=A;

C=A;

subplot(1,3,1);

imshow(A);

title('Original Image');

for row=1:size(A,1)

for col=1:size(A,2)

if A(row,col)>100 && A(row,col)<200

B(row,col)=255;

C(row,col)=255;

else

B(row,col)=0;

C(row,col)=A(row,col);

end

end

end

subplot(1,3,2);

imshow(B);

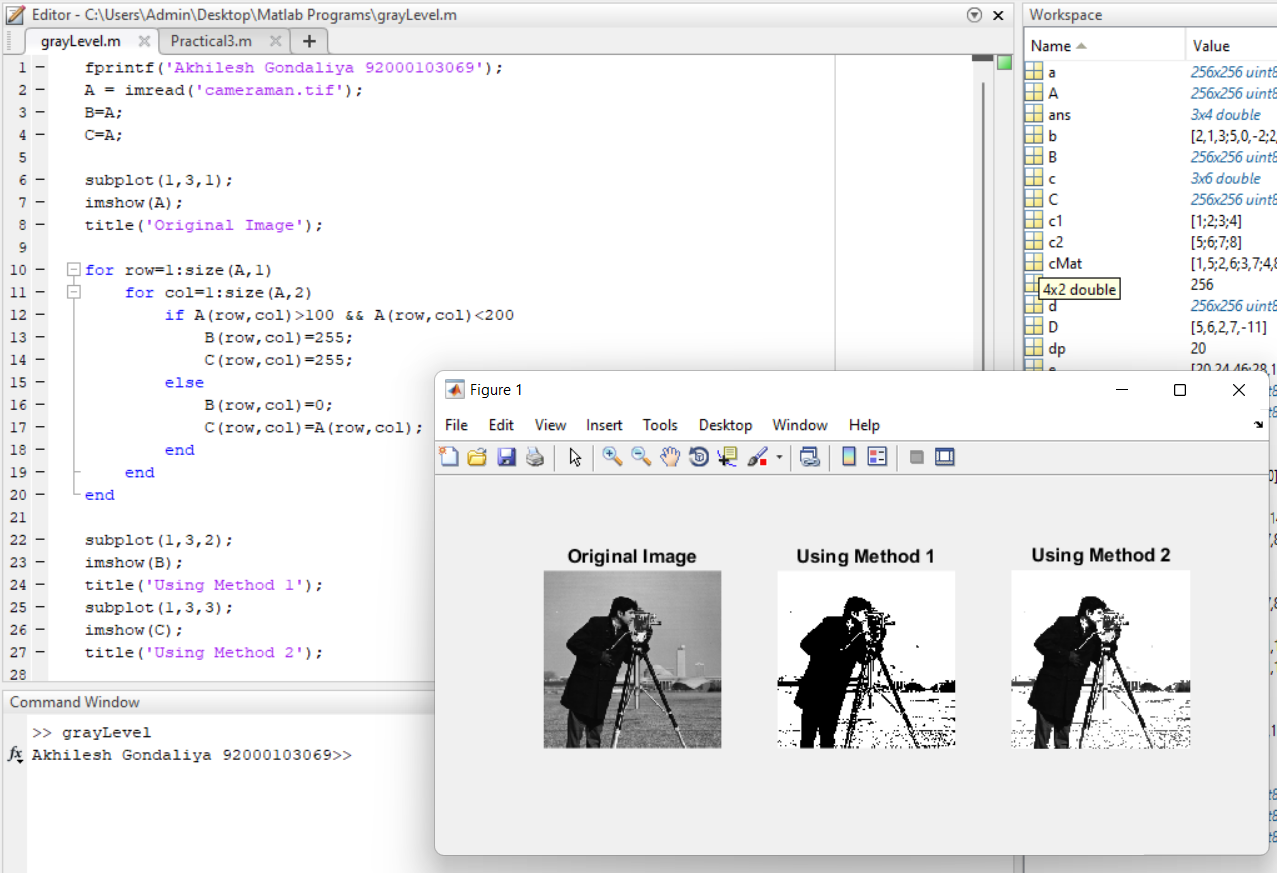
title('Using Method 1');

subplot(1,3,3);

imshow(C);

title('Using Method 2');

**Output:**

****

1. **Bit-plane slicing**

**Code:**

fprintf('Akhilesh Gondaliya 92000103069');

a = imread('cameraman.tif');

subplot(3,2,1);

imshow(a);

title('Original Image');

subplot(3,2,2);

imhist(a);

title('Histogram');

I=numel(a);

subplot(3,2,3);

imshow(a./I);

title('Normalized Image');

p=imhist(a)/I;

subplot(3,2,4);

imhist(p);

title('Normalized Histogram');

j=histeq(a);

subplot(3,2,5);

imshow(j);

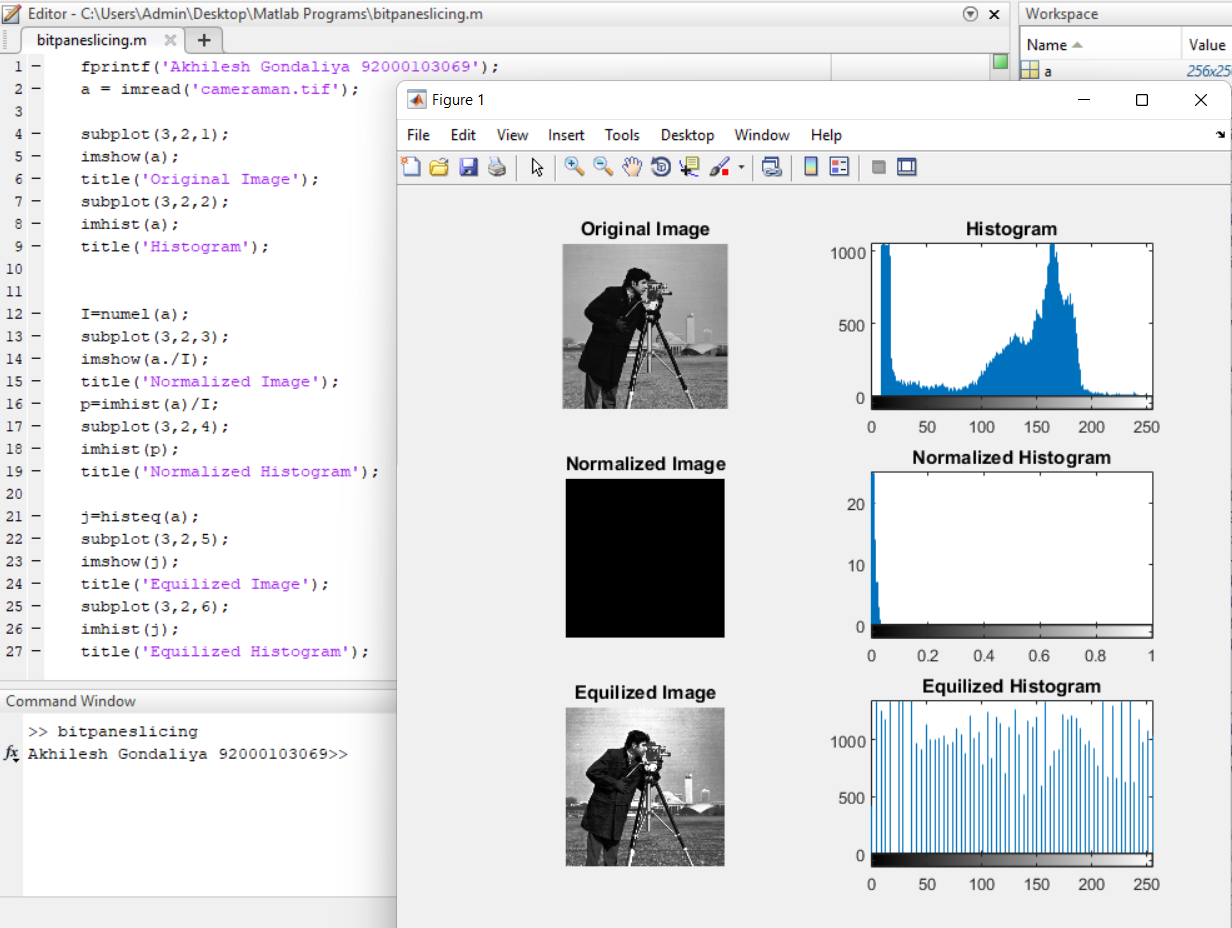
title('Equilized Image');

subplot(3,2,6);

imhist(j);

title('Equilized Histogram');

**Output:**

****