EXPERIMENT 1

Aim : To study basic operations on matrices and images in GNU Octave.

❖ Some Basic Functions

- **Log :-**

  Compute the natural logarithm, 'ln (X)', for each element of X.

  Ex. :
  ```
  log(10)
  ans =  2.3026
  ```

- **Log2 :-**

  Compute the base-2 logarithm of each element of X.

  If called with two output arguments, split X into binary mantissa and exponent so that '1/2 <= abs(f) < 1' and E is an integer.

  If 'x = 0', 'f = e = 0'.

  Ex. :
  ```
  log2(10)
  ans =  3.3219
  ```

- **Log10 :-**

  Compute the base-10 logarithm of each element of X.

  Ex. :
  ```
  log10(10)
  ans =  1
  ```

- **Sqrt :-**

  Compute the square root of each element of X.

  If X is negative, a complex result is returned.

  Ex. :
  ```
  sqrt(25)
  ans =  5
  ```

- **Exp :-**

Compute 'e^x' for each element of X.

Ex. :
```
exp(2)
ans =  7.3891
```

- **Ones :-**

Return a matrix or N-dimensional array whose elements are all 1.

If invoked with a single scalar integer argument N, return a square NxN matrix.

If invoked with two or more scalar integer arguments, or a vector of integer values, return an array with the given dimensions.

Ex. :
```
ones(2,3)
ans =
        1   1   1
        1   1   1
```

- **Zeros :-**

Return a matrix or N-dimensional array whose elements are all 0.

If invoked with a single scalar integer argument, return a square NxN matrix.

If invoked with two or more scalar integer arguments, or a vector of integer values, return an array with the given dimensions.

Ex. :
```
zeros(3,3)
ans =
        0   0   0
        0   0   0
        0   0   0
```

- **Eye :-**

Return an identity matrix.

If invoked with a single scalar argument N, return a square NxN identity matrix.

If supplied two scalar arguments (M, N), 'eye' takes them to be the number of rows and columns. If given a vector with two elements, 'eye' uses the values of the elements as the number of rows and columns, respectively.

Ex. : eye(3)

```
ans =
        Diagonal Matrix
        1   0   0
        0   1   0
        0   0   1
```

- **Imshow :-**

  Display the image IM, where IM can be a 2-dimensional (grayscale image) or a 3-dimensional (RGB image) matrix.

  If LIMITS is a 2-element vector '[LOW, HIGH]', the image is shown using a display range between LOW and HIGH. If an empty matrix is passed for LIMITS, the display range is computed as the range between the minimal and the maximal value in the image.

  Ex. : `imshow(a)`

- **Imread :-**

  Read images from various file formats.

  Read an image as a matrix from the file FILENAME or from the online resource URL. If neither is given, but EXT was specified, look for a file with the extension EXT.

  Ex. : `imread (filename, "PixelRegion", {[200 600], [300 700]});`

- **Imwrite :-**

  Write images in various file formats.

  The image IMG can be a binary, grayscale, RGB, or multi-dimensional image. The size and class of IMG should be the same as what should be expected when reading it with 'imread': the 3rd and 4th dimensions reserved for color space, and multiple pages respectively. If it's an indexed image, the colormap MAP must also be specified.

  Ex. : `imwrite(c,'E:\SEM 7\IP\lab\img.jpg')`

- **Imfinfo :-**

  Read image information from a file.

  'imfinfo' returns a structure containing information about the image stored in the file FILENAME. If there is no file FILENAME, and EXT was specified, it will look for a file named FILENAME and extension EXT, i.e., a file named FILENAME.EXT.

  Ex. : imfinfo(filename)

- **Subplot :-**

  Set up a plot grid with ROWS by COLS subwindows and set the current axes for plotting ('gca') to the location given by INDEX.

  If an axes handle HAX is provided after the (ROWS, COLS, INDEX) arguments, the corresponding axes is turned into a subplot.

  If only one numeric argument is supplied, then it must be a three digit value specifying the number of rows in digit 1, the number of columns in digit 2, and the plot index in digit 3.

  The plot index runs row-wise; First, all columns in a row are numbered and then the next row is filled.

  Ex. : a plot with 2x3 grid will have plot indices running as follows:

```
+-----+-----+-----+
|  1  |  2  |  3  |
+-----+-----+-----+
|  4  |  5  |  6  |
+-----+-----+-----+
```

❖ **Create the following matrix A:**

$$A = \begin{bmatrix} 6 & 43 & 2 & 11 & 87 \\ 12 & 6 & 34 & 0 & 5 \\ 34 & 18 & 7 & 41 & 9 \end{bmatrix}$$

Use the matrix A to :

a) Create a live-element row vector named va that contains the elements of the second row of A.

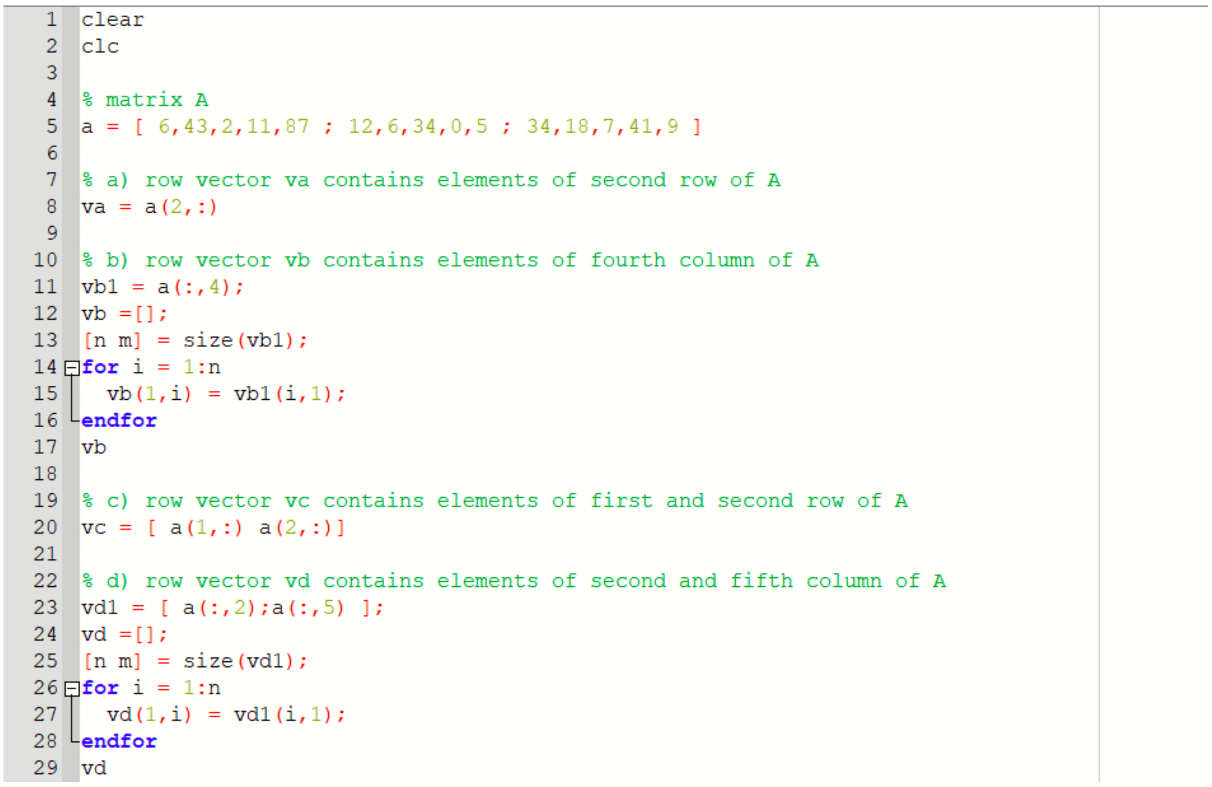
b) Creates three-element row vector named vb that contains the elements of the fourth column of A.

c) Create a ten-element row vector named vc that contains the elements of the first and second rows of A.

d) Create a six-element row vector named vd that contains the elements of the second and fifth columns of A.

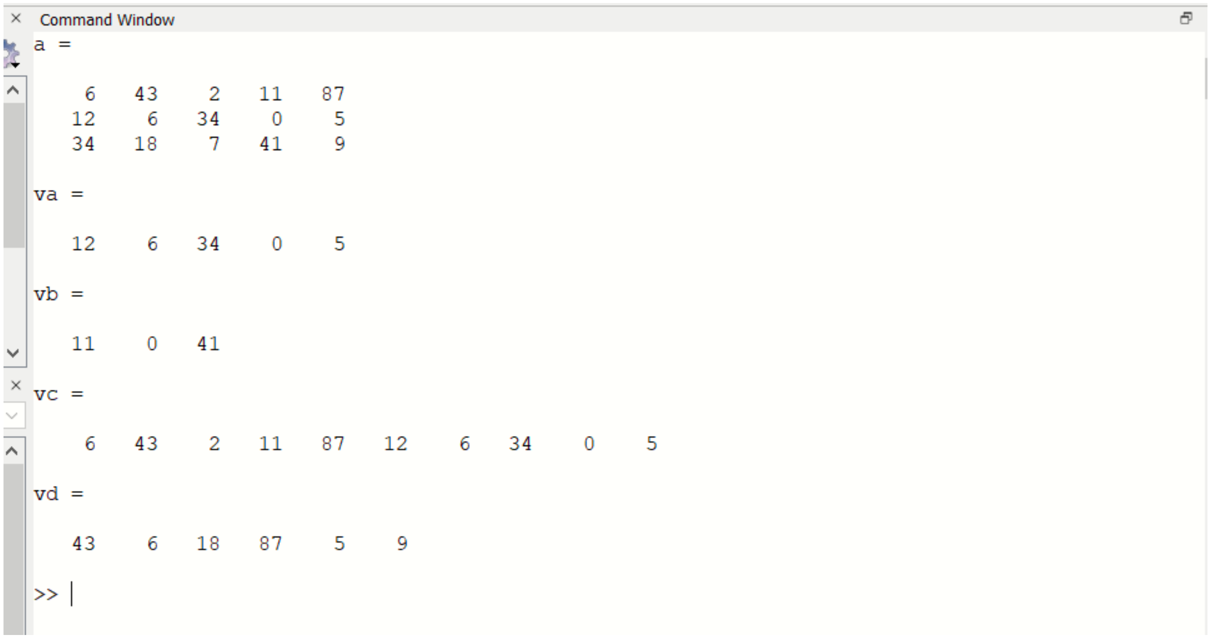
→ Solution :-

Code :



```
clear
clc

% matrix A
a = [ 6,43,2,11,87 ; 12,6,34,0,5 ; 34,18,7,41,9 ]

% a) row vector va contains elements of second row of A
va = a(2,:)

% b) row vector vb contains elements of fourth column of A
vb1 = a(:,4);
vb =[];
[n m] = size(vb1);
for i = 1:n
  vb(1,i) = vb1(i,1);
endfor
vb

% c) row vector vc contains elements of first and second row of A
vc = [ a(1,:) a(2,:)]

% d) row vector vd contains elements of second and fifth column of A
vd1 = [ a(:,2);a(:,5) ];
vd =[];
[n m] = size(vd1);
for i = 1:n
  vd(1,i) = vd1(i,1);
endfor
vd
```

Output :



```
a =

     6    43     2    11    87
    12     6    34     0     5
    34    18     7    41     9

va =

    12     6    34     0     5

vb =

    11     0    41

vc =

     6    43     2    11    87    12     6    34     0     5

vd =

    43     6    18    87     5     9

>> |
```

❖ **Create the following three matrices:**

$$\mathbf{A} = \begin{bmatrix} 5 & 2 & 4 \\ 1 & 7 & -3 \\ 6 & -10 & 0 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 11 & 5 & -3 \\ 0 & -12 & 4 \\ 2 & 6 & 1 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 7 & 14 & 1 \\ 10 & 3 & -2 \\ 8 & -5 & 9 \end{bmatrix}$$

a) Calculate A + B and B + .A. to show that addition of matrices is commutative.

b) Calculate A+ (B + C) and (A+ B) + C to show that addition of matrices is associative.

c) Calculate S(A + C) and :SA+ SC to show that, when matrices arc multiplied by a scalar, the multiplication is distributive.

d) Calculate A*(B + C) and A* B + A*C to show that matrix multiplication is distributive.

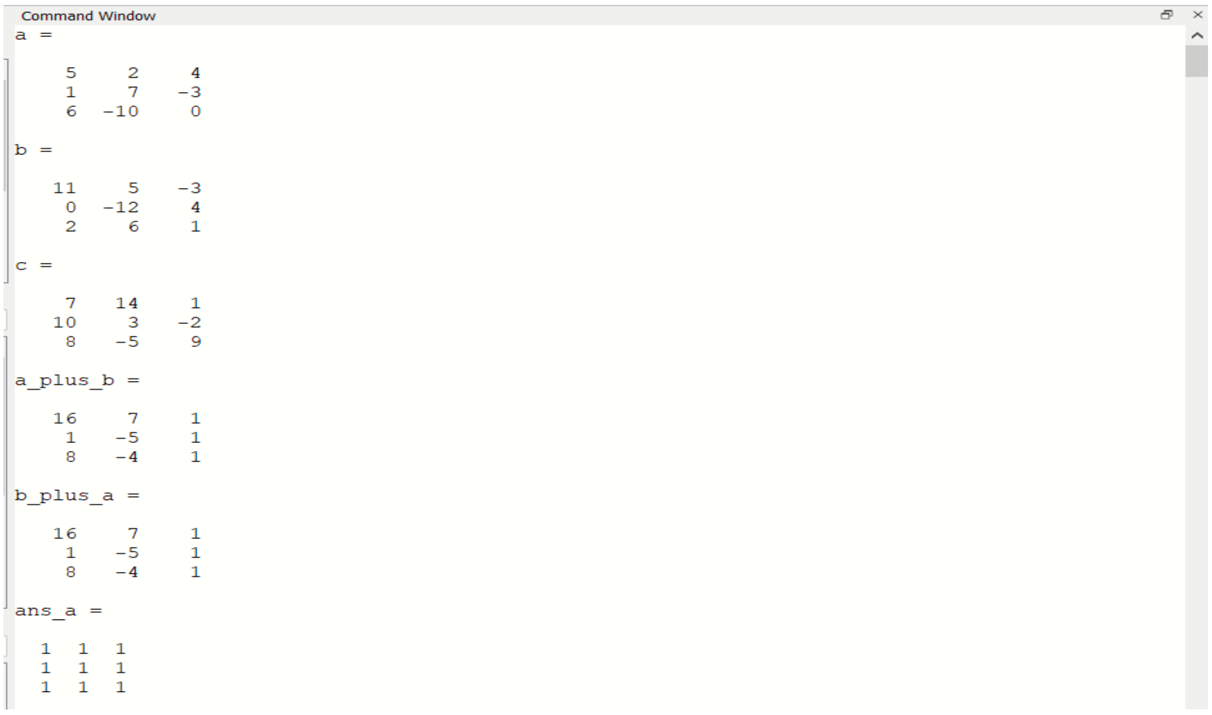
→ Solution :-

Code :

```
clear
clc

% matrix A
a = [ 5,2,4 ; 1,7,-3 ; 6,-10,0 ]

% matrix B
b = [ 11,5,-3 ; 0,-12,4 ; 2,6,1 ]

% matrix c
c = [ 7,14,1 ; 10,3,-2 ; 8,-5,9 ]

% A+B = B+A
a_plus_b = a + b
b_plus_a = b + a
ans_a = a_plus_b == b_plus_a

% A+(B+C) = (A+B)+C
a_mul_b_plus_c = a*(b+c)
a_mul_b_plus_a_mul_c = a*b + a*c
ans_b = a_mul_b_plus_c == a_mul_b_plus_a_mul_c

% 5(A+C) = 5A+5C
const_star_a_plus_c = 5 * (a + c)
const_star_a_plus_const_star_c = 5*a + 5*c
ans_c = const_star_a_plus_c == const_star_a_plus_const_star_c

% A*(B+C) = A*B+A*C
a_star_b_plus_c = a * (b + c)
a_star_b_plus_a_star_c = a*b + a*c
ans_d = a_star_b_plus_c == a_star_b_plus_a_star_c
```

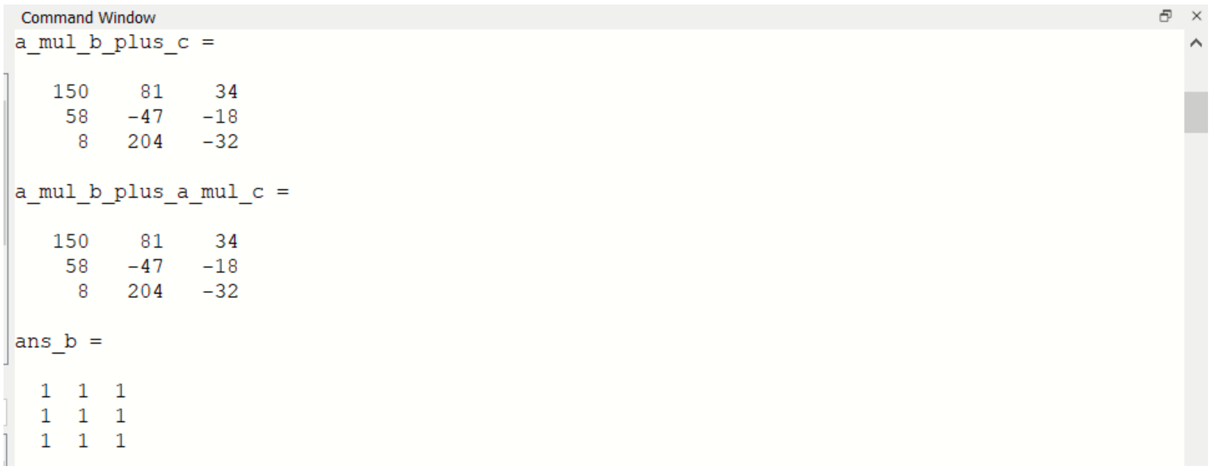
Output :

a) A+B = B+A



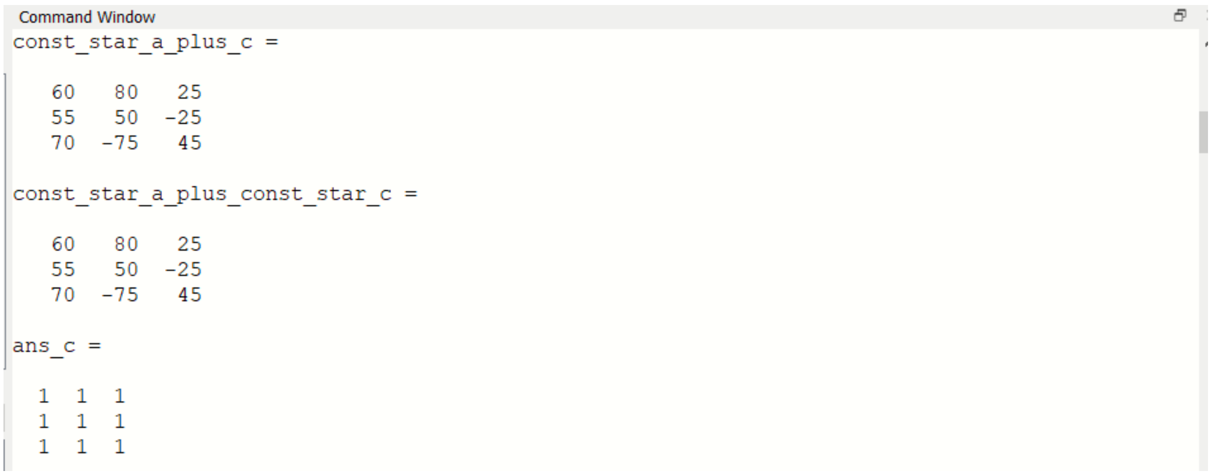
```
a =

     5     2     4
     1     7    -3
     6   -10     0

b =

    11     5    -3
     0   -12     4
     2     6     1

c =

     7    14     1
    10     3    -2
     8    -5     9

a_plus_b =

    16     7     1
     1    -5     1
     8    -4     1

b_plus_a =

    16     7     1
     1    -5     1
     8    -4     1

ans_a =

  1  1  1
  1  1  1
  1  1  1
```

b) A+(B+C) = (A+B)+C



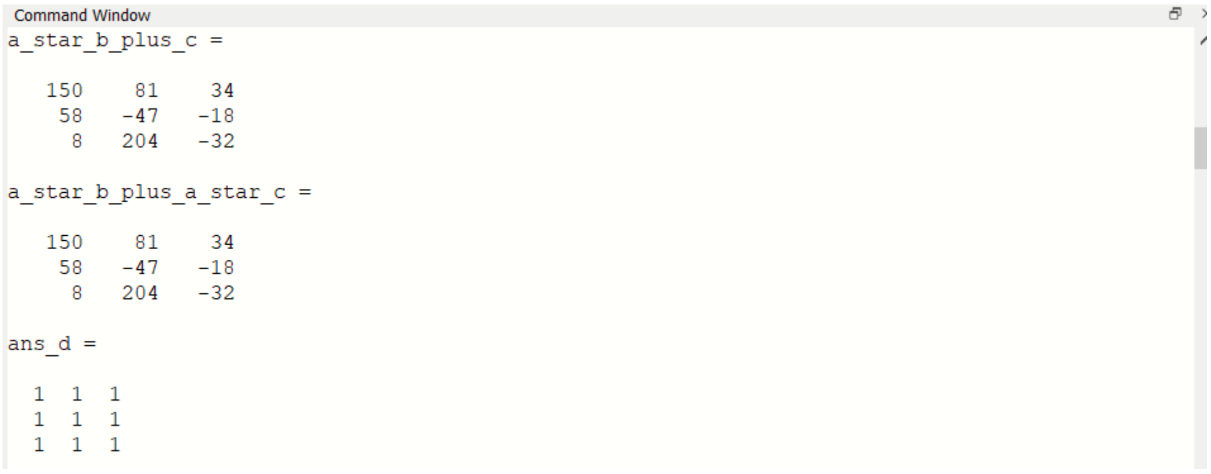
```
a_mul_b_plus_c =

   150    81    34
    58   -47   -18
     8   204   -32

a_mul_b_plus_a_mul_c =

   150    81    34
    58   -47   -18
     8   204   -32

ans_b =

  1  1  1
  1  1  1
  1  1  1
```

c) 5(A+C) = 5A+5C



```
const_star_a_plus_c =

    60    80    25
    55    50   -25
    70   -75    45

const_star_a_plus_const_star_c =

    60    80    25
    55    50   -25
    70   -75    45

ans_c =

  1  1  1
  1  1  1
  1  1  1
```

d) A*(B+C) = A*B+A*C

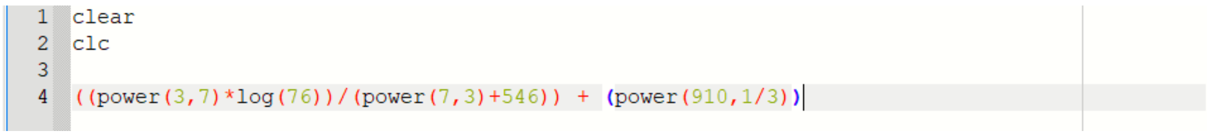


```
a_star_b_plus_c =

   150     81     34
    58    -47    -18
     8    204    -32

a_star_b_plus_a_star_c =

   150     81     34
    58    -47    -18
     8    204    -32

ans_d =

  1  1  1
  1  1  1
  1  1  1
```

❖ **Calculate :** $\frac{3^7\,(\log 76)}{7^3+546} + \sqrt[3]{910}$

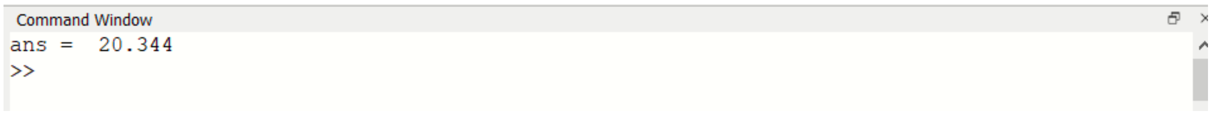
→ **Solution :-**

Code :



```
clear
clc

((power(3,7)*log(76))/(power(7,3)+546)) + (power(910,1/3))
```

Output :

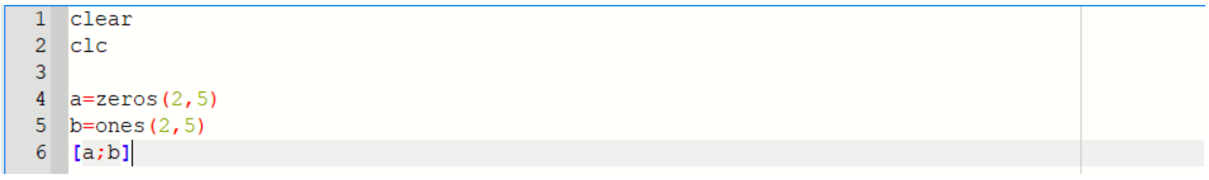


```
ans =  20.344
>>
```

❖ **Using the ones and zeros commands, create a 4 x 5 matrix in which the first two rows are O's and the next two rows are 1 's.**

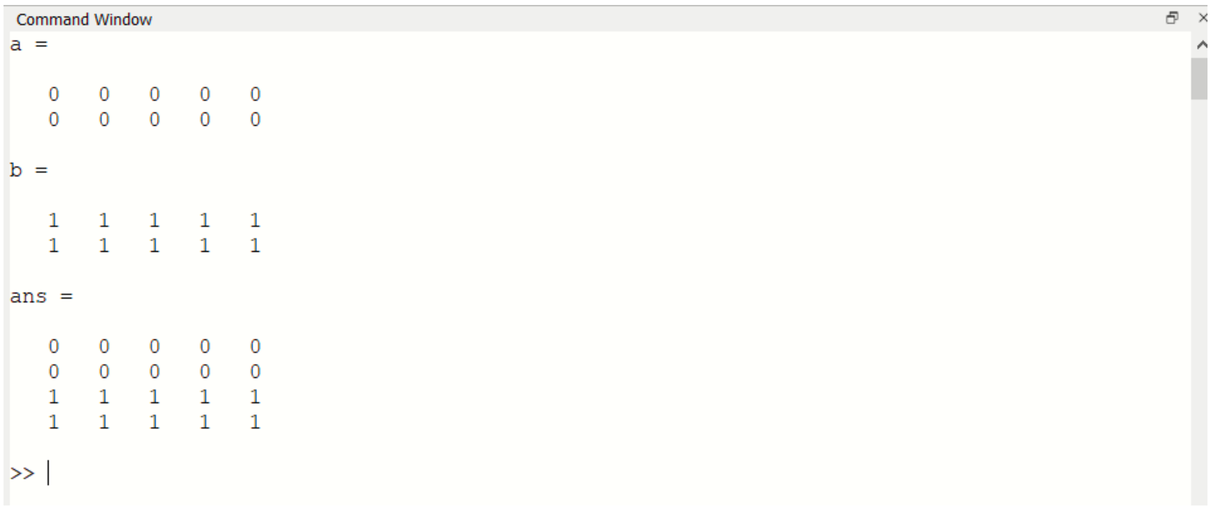
→ **Solution :-**

Code :



```
clear
clc

a=zeros(2,5)
b=ones(2,5)
[a;b]
```

Output :



```
a =

     0     0     0     0     0
     0     0     0     0     0

b =

     1     1     1     1     1
     1     1     1     1     1

ans =

     0     0     0     0     0
     0     0     0     0     0
     1     1     1     1     1
     1     1     1     1     1

>>
```

❖ **Take your own photo(RGB image) and create following images and save them for future use.**

a) Gray scale image.
b) Black and white image.
c) Over Exposed image.
d) Under Exposed image.
e) keep your face only-crop rest of the image.
f) Resize the image to 256*256.

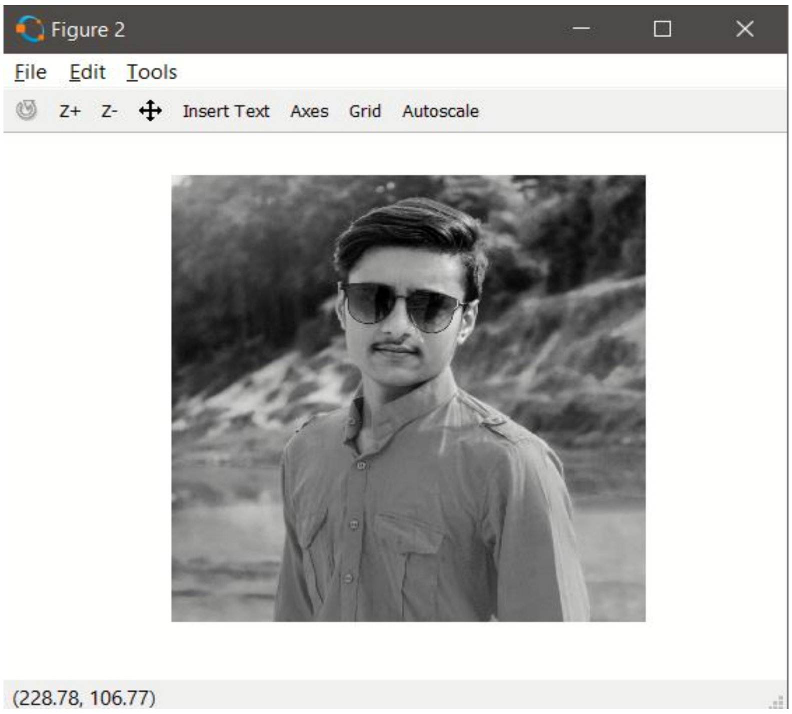
→ Solution :-

Code :

```
clear
clc
% Read image into a
a=imread('G:\My Pic\n.jpg');

% Gray scale image
b=rgb2gray(a);
figure
imshow(b)

% Black and white image
%c=double(b)/255;
c=im2bw(b);
figure
imshow(c)

% Over Exposed image
d=b+60;
figure
imshow(d)

% Under Exposed image
d=b-50;
figure
imshow(d)

% face only-crop rest of the image
f=imcrop(a,[370 35 565 620]);
figure
imshow(f)

% Resize the image to 256*256.
g=imresize(a,[256,256]);
figure
imshow(g)
```

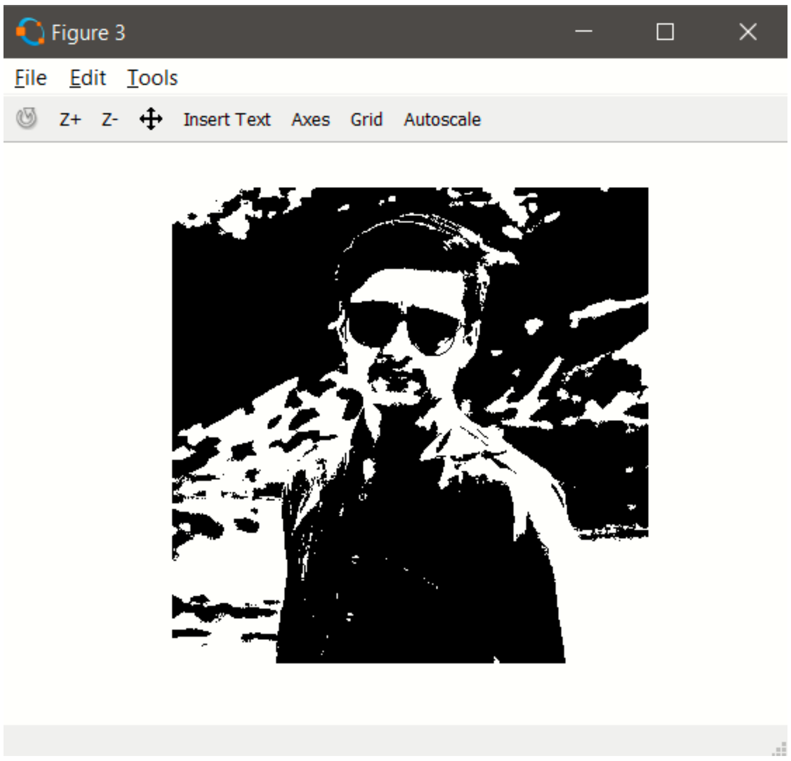
Output :

a) Gray scale image

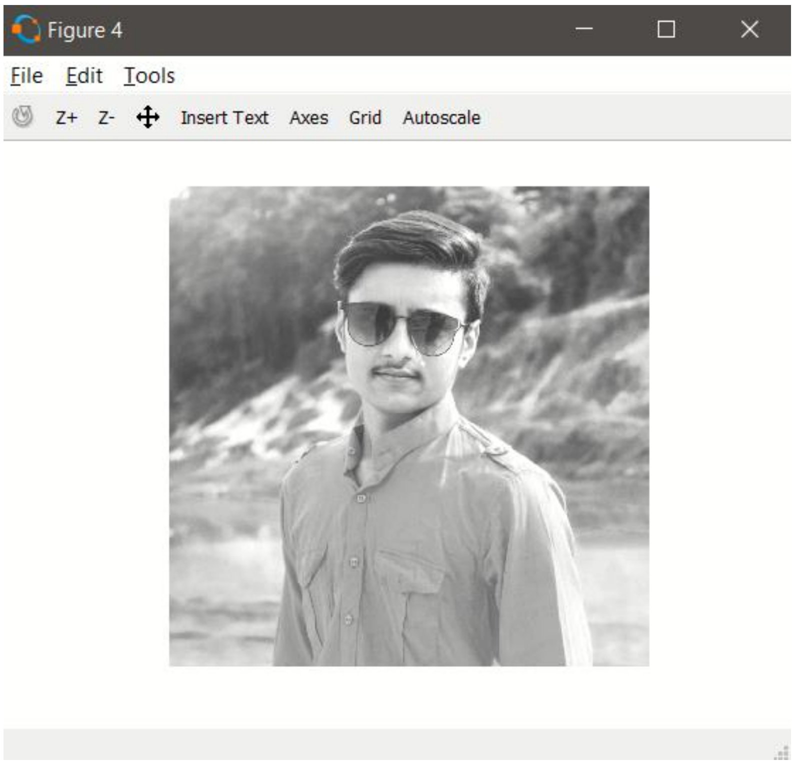


(228.78, 106.77)

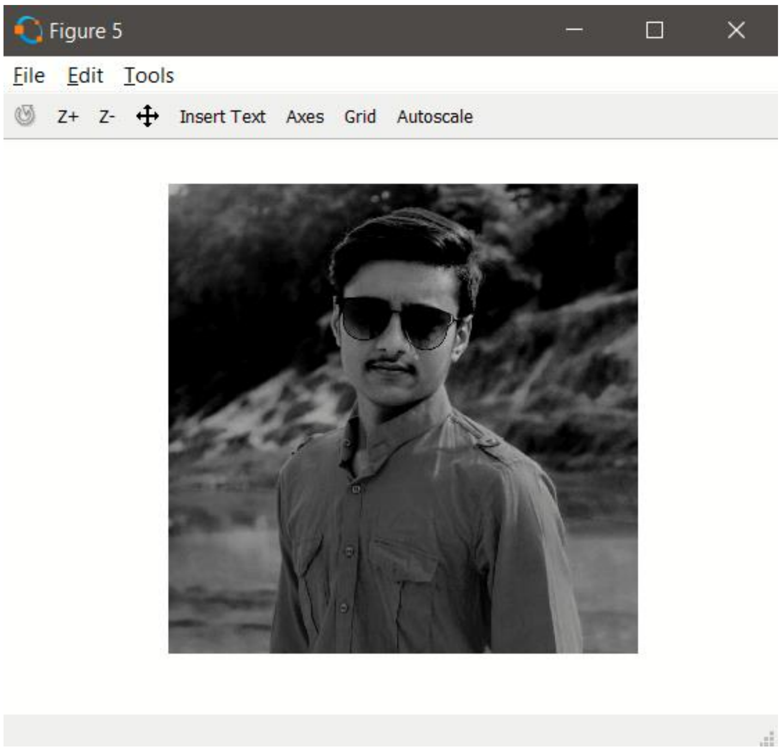
b) Black and white image



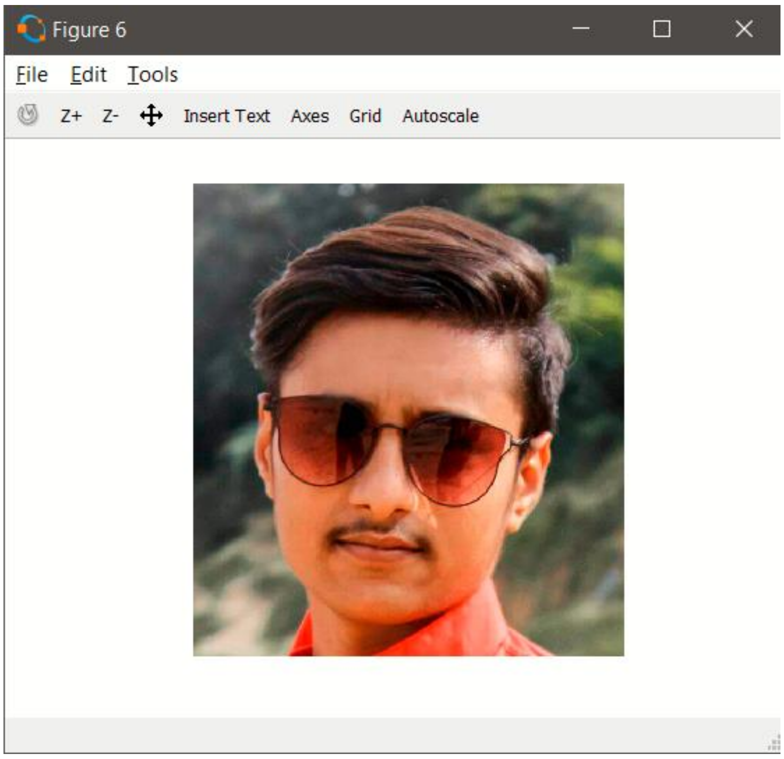
c) Over Exposed image



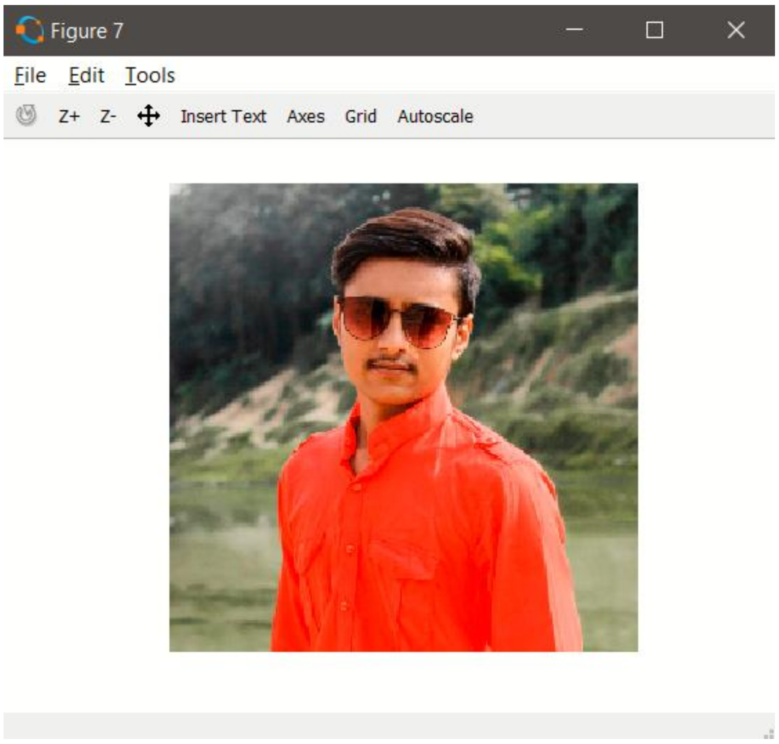
d) Under Exposed image



e) keep your face only-crop rest of the image



f) Resize the image to 256*256



❖ **Take your own photo and process them for following results using loop controlling structures.**

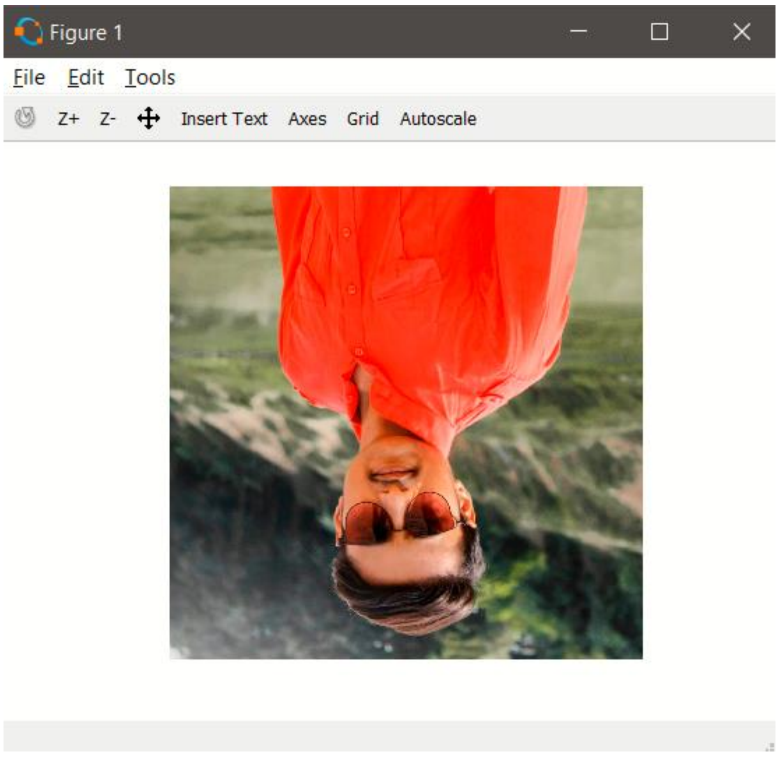
a) **flip your image vertically.**
b) **create the mirror image.**
c) **rotate the image by 90 degree.**
d) **rotate the image by 270 degree.**

→ **Solution :-**

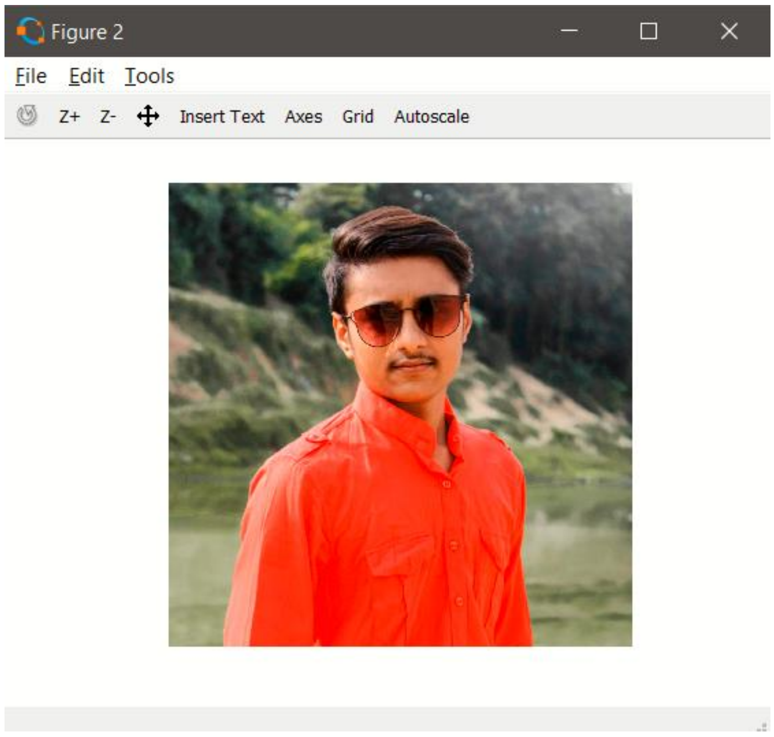
Code :

```
clear
clc

% Read image into a
a=imread('G:\My Pic\n.jpg');

% flip your image vertically
b=flipud(a);
figure
imshow(b);

% create the mirror image
c=fliplr(a);  %c=flip(a,2);
figure
imshow(c);

% rotate the image by 90 degree
d=imrotate(a,90);
figure
imshow(d);

% rotate the image by 270 degree
e=imrotate(a,270);
figure
imshow(e);

figure
subplot(2,2,1);
imshow(b)
subplot(2,2,2);
imshow(c)
subplot(2,2,3);
imshow(d)
subplot(2,2,4);
imshow(e)
```

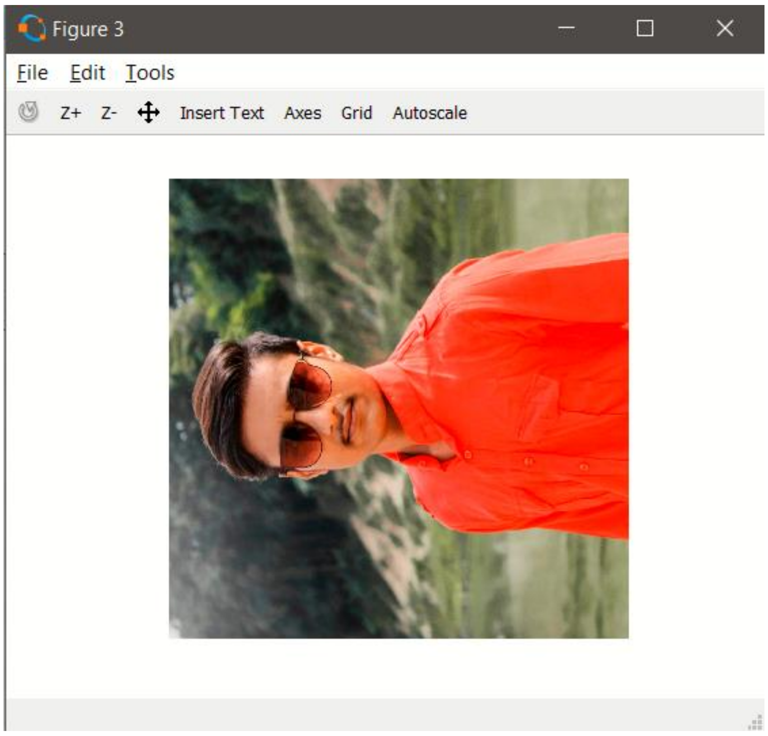
a) flip your image vertically



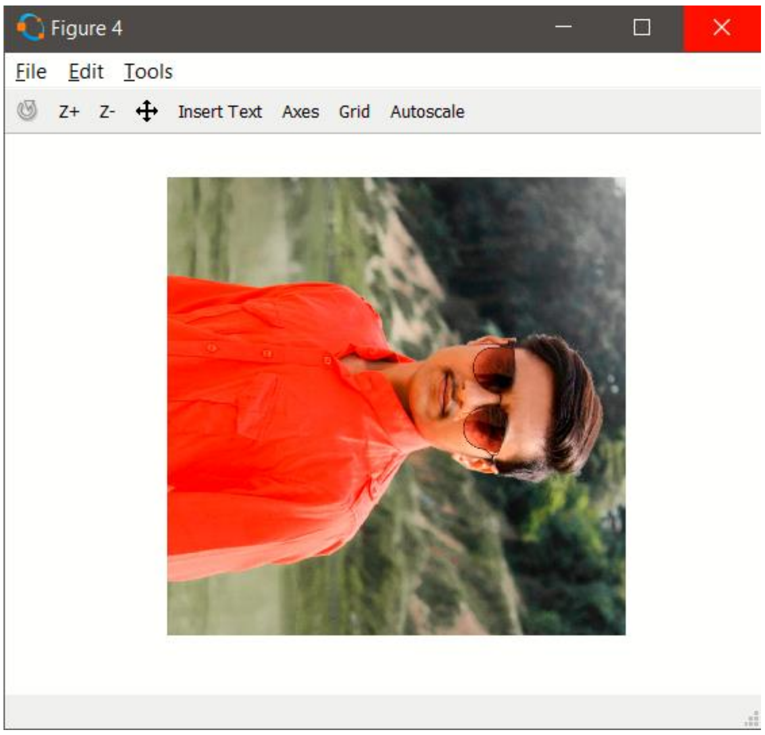
b) create the mirror image



c) rotate the image by 90 degree



d) rotate the image by 270 degree



e) subplot

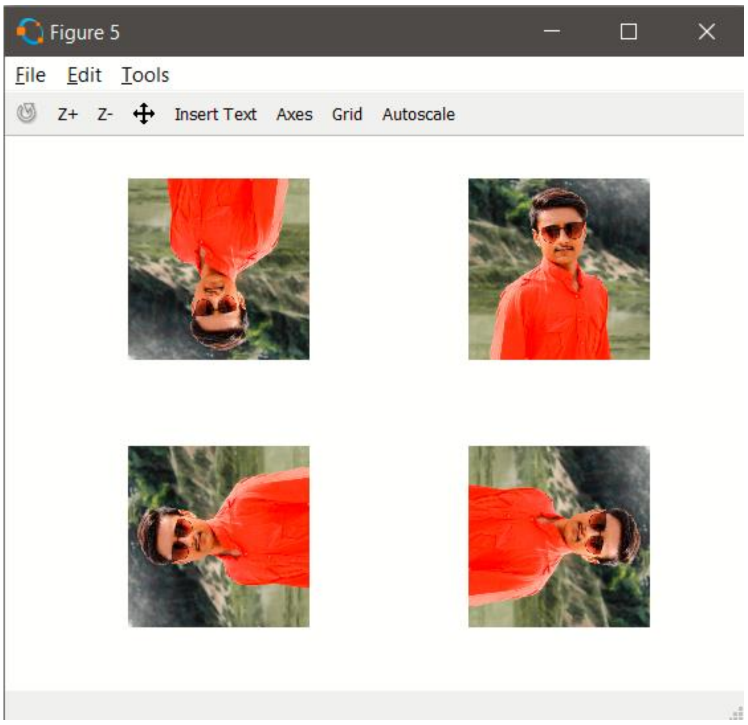


Figure 5
File Edit Tools
Z+ Z- Insert Text Axes Grid Autoscale