SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University under section 3 of the UGC Act, 1956)

Accredited with "A" Grade by NAAC | Approved by AICTE

SCSA2609- Neural Networks Using MATLAB

(FOR 3rd YEAR B.E COMPUTER SCIENCE – ARTIFICIAL INTELLIGENCE & ROBOTICS)

NAME : ______________________________

REGISTER NO : ______________________________

SEMESTER : ______________________________

INDEX

Name of the Laboratory : ______________________________________

Subject Code : ____________________________________

Name of the Staff In-Charge: ____________________________________

S.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	DATE OF SUBMISSION	MARK (10)	SIGNATURE
1						
2 (a)						
2 (b)						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						

EXP 1: Study of MATLAB

~Space for Certificate - MATLAB Onramp~

EXP 2(a): Program to perform basic operations in MATLAB

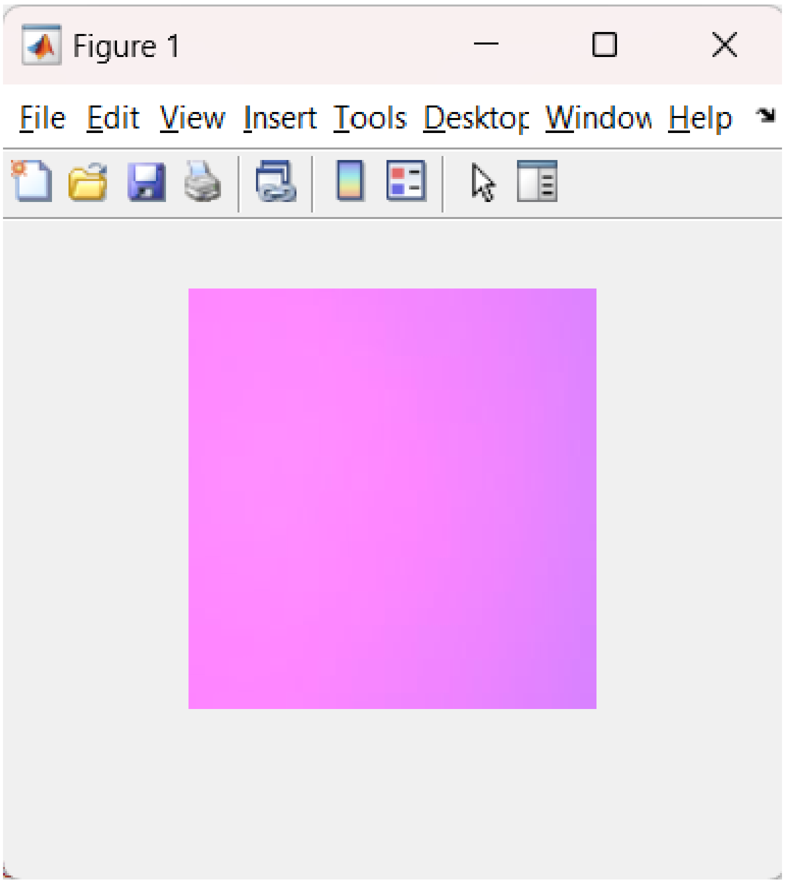
(i) Addition of two images:

- Pixel-wise Addition

Code:

```
Red = imread("Red.jpg");
Blue = imread("blue.jpg");
A = Red + Blue;
imshow(A)
```

Output:

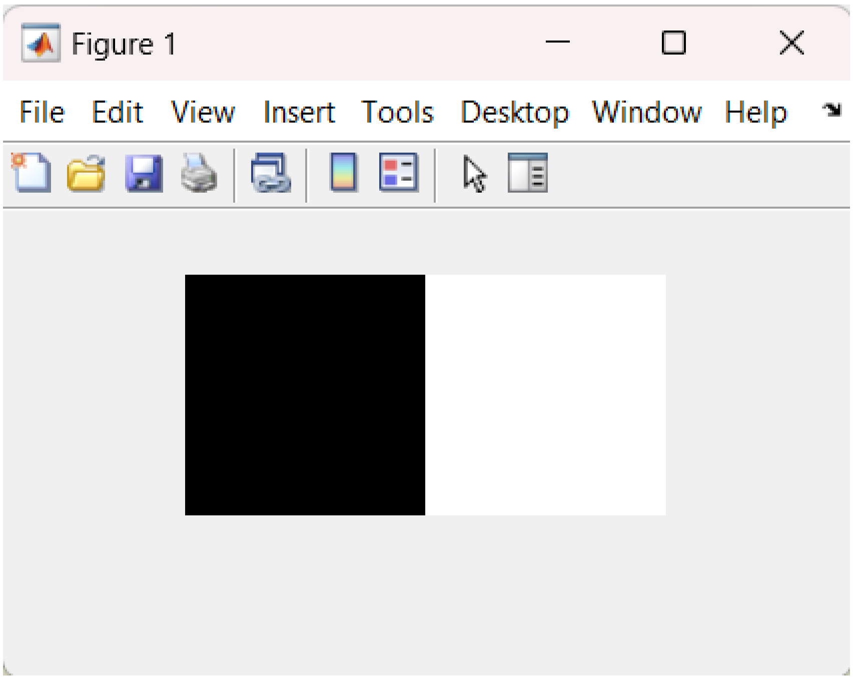


- Adding two images next to each other

Code:

```
black = zeros(120,120);
white = 255*ones(120,120);
imshowpair(black, white, "montage")
```

Output:

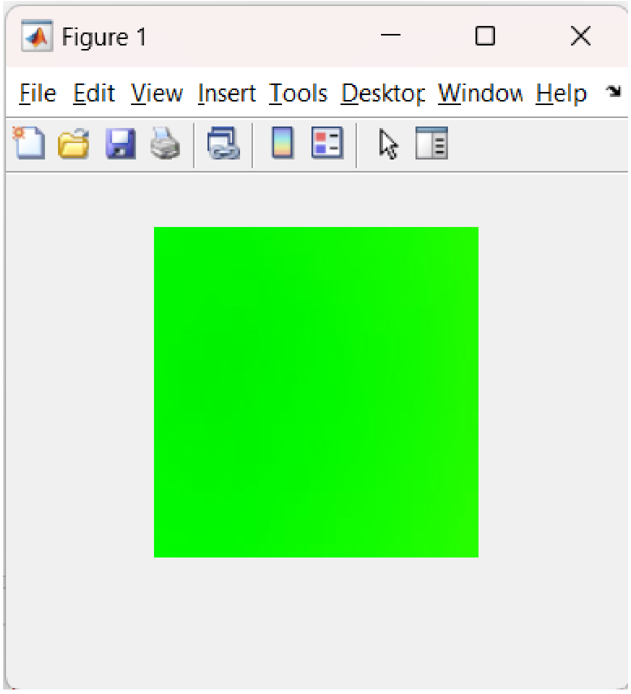


(ii) Subtraction of two images (Pixelwise):

Code:

```
Yellow = imread("yellow.jpg");
Red= imread("Red.jpg");
A = Yellow - Red;
imshow(A)
```

Output:

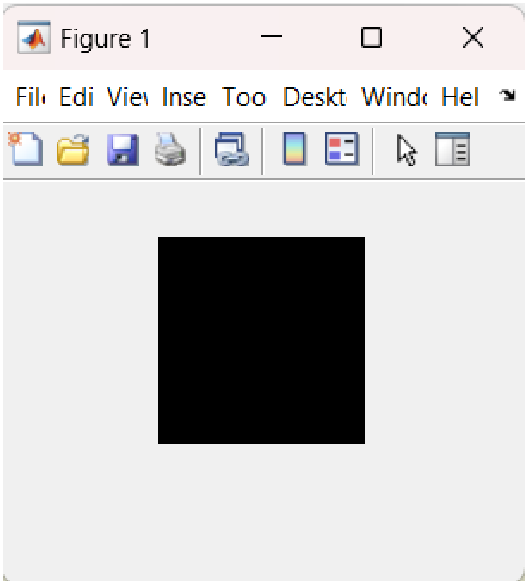


(iii) AND Operation

Code:

```
black = zeros(120,120);
white = 255*ones(120,120);
and = black & white;
imshow(and)
```

Output:

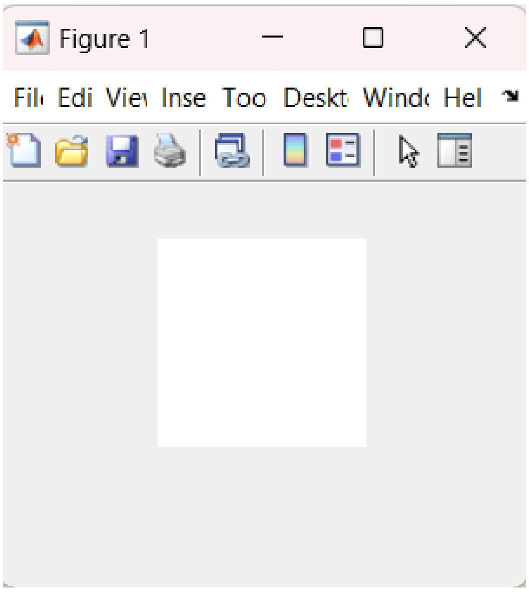


(iv) OR Operation

Code:

```
black = zeros(120,120);
white = 255*ones(120,120);
or = black | white;
imshow(or)
```

Output:

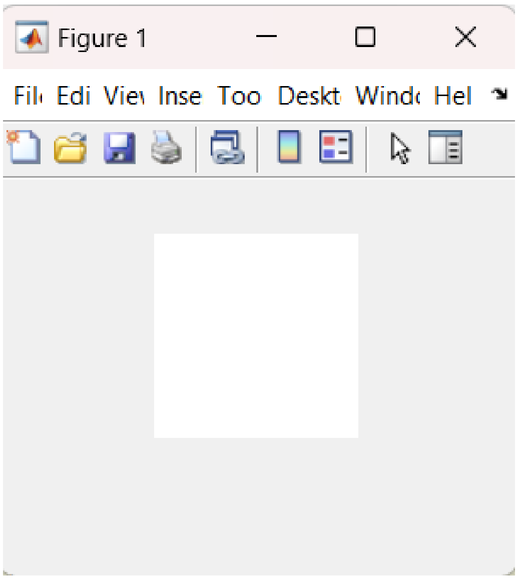


(v) XOR Operation

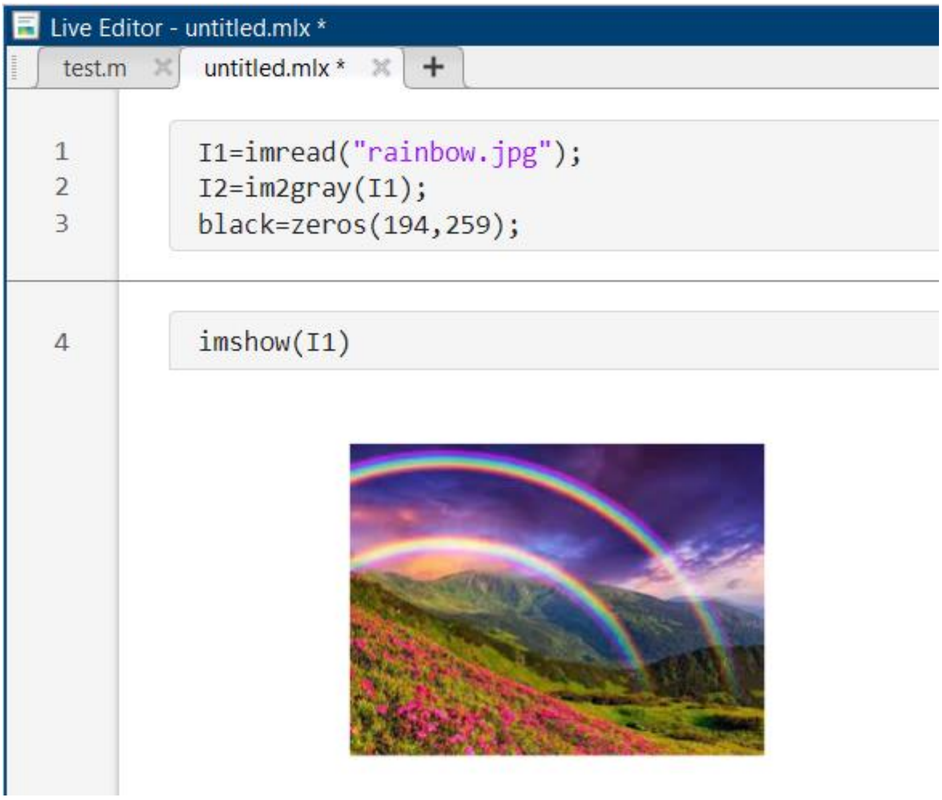
Code:

```
black = zeros(120,120);
white = 255*ones(120,120);
xor = xor(black, white);
imshow(xor)
```

Output:

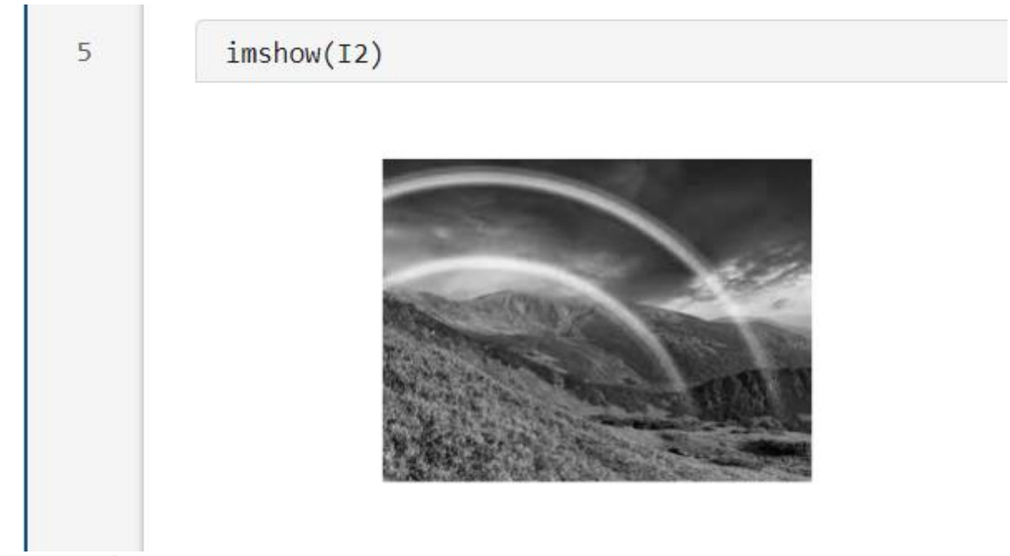


(vi) Grayscale and RGB Images of an Image

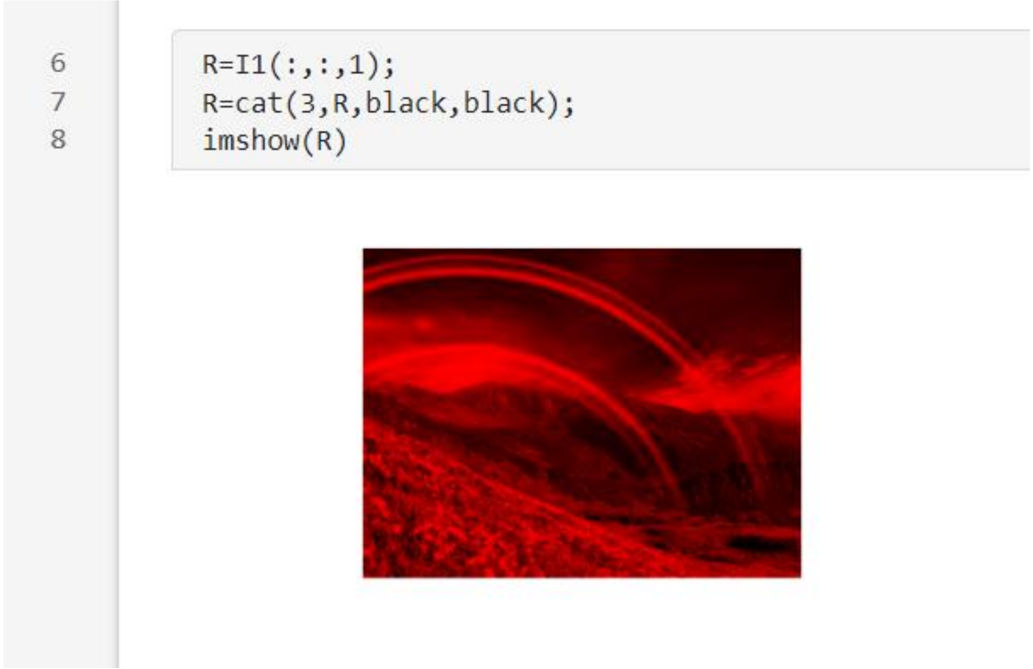


```
I1=imread("rainbow.jpg");
I2=im2gray(I1);
black=zeros(194,259);
```

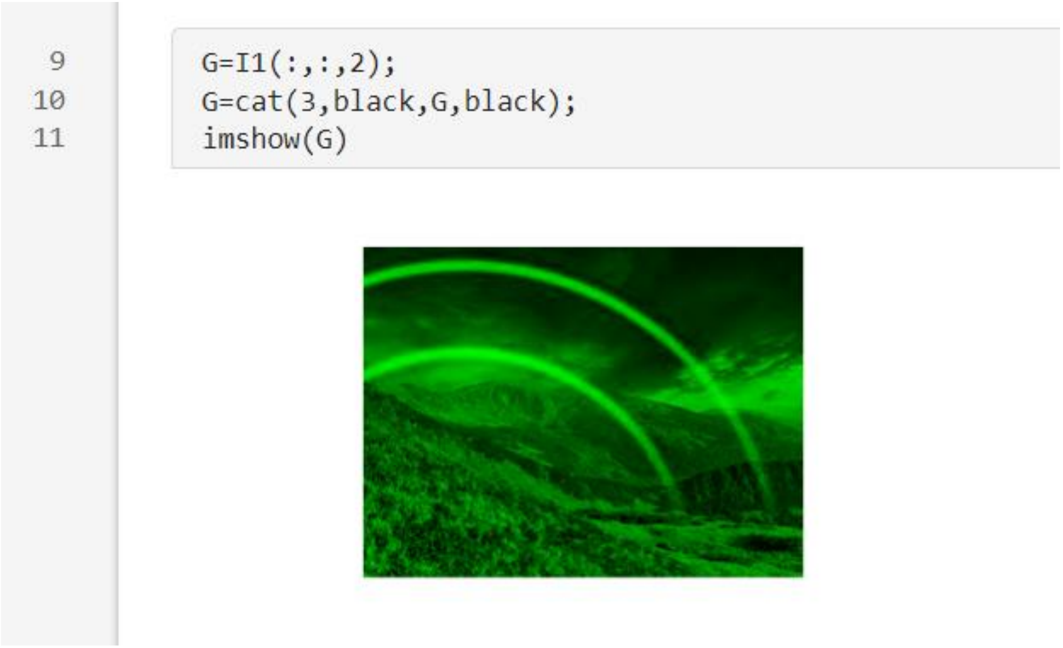
```
imshow(I1)
```



```
imshow(I2)
```

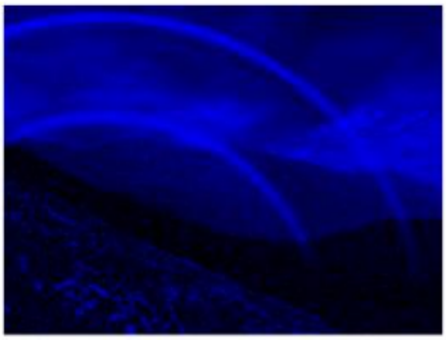


```
R=I1(:,:,1);
R=cat(3,R,black,black);
imshow(R)
```

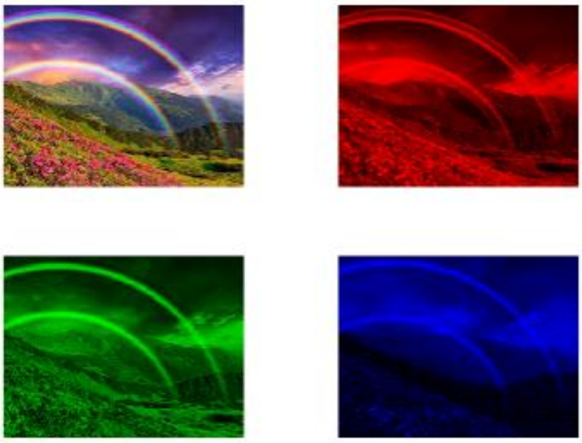


```
G=I1(:,:,2);
G=cat(3,black,G,black);
imshow(G)
```

```
B=I1(:,:,3);
B=cat(3,black,black,B);
imshow(B)
```



```
subplot(2,2,1), imshow(I1)
subplot(2,2,2), imshow(R)
subplot(2,2,3), imshow(G)
subplot(2,2,4), imshow(B)
```



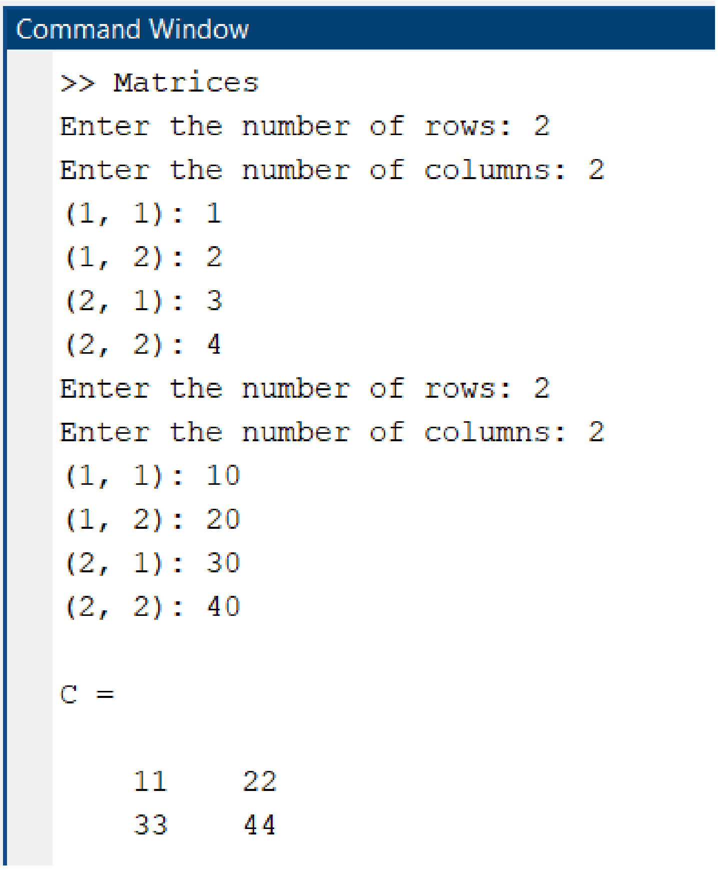
EXP 2(b): Program to perform matrix operations in MATLAB

(i) Addition of two matrices

Code:

```
r1= input('Enter the number of rows: ');
c1 = input('Enter the number of columns: ');
A = zeros(r1, c1);
for i = 1:r1
   for j = 1:c1
      A(i, j) = input(sprintf('(%d, %d): ', i, j));
   end
end
r2= input('Enter the number of rows: ');
c2 = input('Enter the number of columns: ');
if r2==r1 && c2==c1
   B = zeros(r2, c2);
   for i = 1:r2
      for j = 1:c2
         B(i, j) = input(sprintf('(%d, %d): ', i, j));
      end
   end
   C = A + B
else
   disp('Error: Matrices have different size. Execution Terminated.');
end
```

Output:



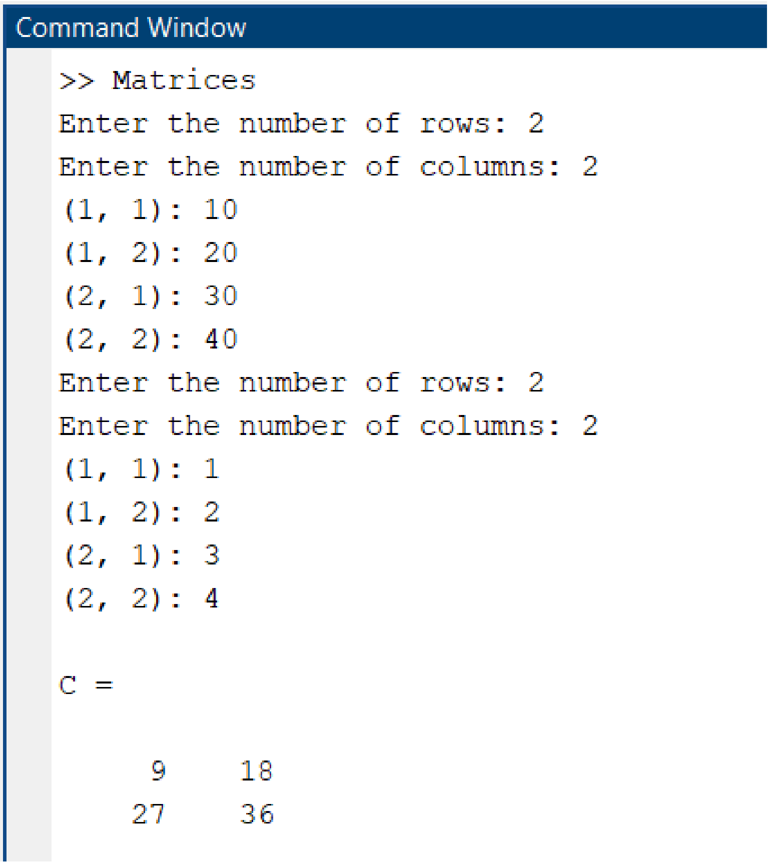
```
>> Matrices
Enter the number of rows: 2
Enter the number of columns: 2
(1, 1): 1
(1, 2): 2
(2, 1): 3
(2, 2): 4
Enter the number of rows: 2
Enter the number of columns: 2
(1, 1): 10
(1, 2): 20
(2, 1): 30
(2, 2): 40

C =

    11    22
    33    44
```

(ii) Subtraction of two matrices

Code:

```
r1= input('Enter the number of rows: ');
c1 = input('Enter the number of columns: ');
A = zeros(r1, c1);
for i = 1:r1
    for j = 1:c1
        A(i, j) = input(sprintf('(%d, %d): ', i, j));
    end
end
r2= input('Enter the number of rows: ');
c2 = input('Enter the number of columns: ');
if r2==r1 && c2==c1
    B = zeros(r2, c2);
    for i = 1:r2
        for j = 1:c2
            B(i, j) = input(sprintf('(%d, %d): ', i, j));
        end
    end
    C = A - B
else
    disp('Error: Matrices have different size. Execution Terminated.');
end
```

Output:



```
>> Matrices
Enter the number of rows: 2
Enter the number of columns: 2
(1, 1): 10
(1, 2): 20
(2, 1): 30
(2, 2): 40
Enter the number of rows: 2
Enter the number of columns: 2
(1, 1): 1
(1, 2): 2
(2, 1): 3
(2, 2): 4

C =

     9    18
    27    36
```

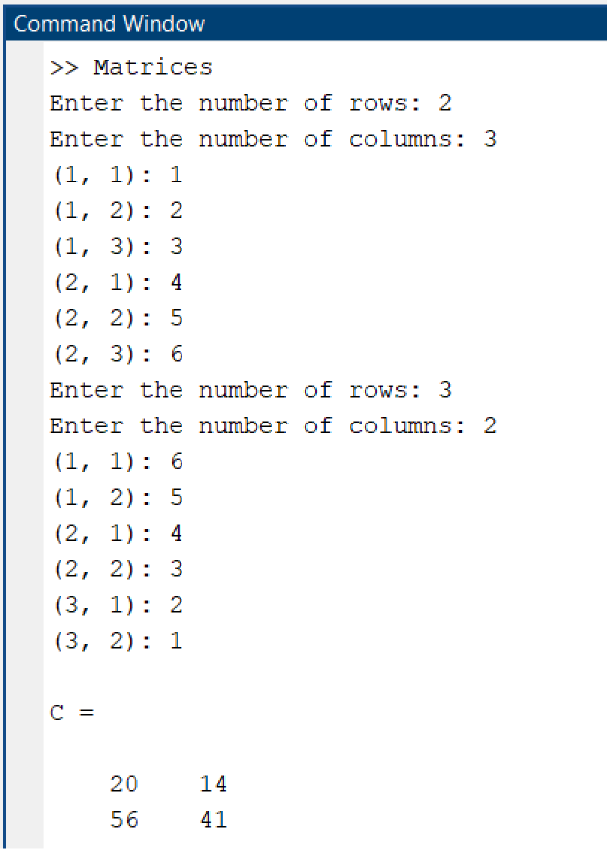
(iii) Multiplication of two matrices

- Normal Matrix Multiplication

Code:

```
r1= input('Enter the number of rows: ');
c1 = input('Enter the number of columns: ');
A = zeros(r1, c1);
for i = 1:r1
   for j = 1:c1
      A(i, j) = input(sprintf('(%d, %d): ', i, j));
   end
end
r2= input('Enter the number of rows: ');
c2 = input('Enter the number of columns: ');
if c1==r2
   B = zeros(r2, c2);
   for i = 1:r2
      for j = 1:c2
         B(i, j) = input(sprintf('(%d, %d): ', i, j));
      end
   end
   C = A * B
else
   disp('Error: Matrices have different size. Execution Terminated.');
end
```

Output:



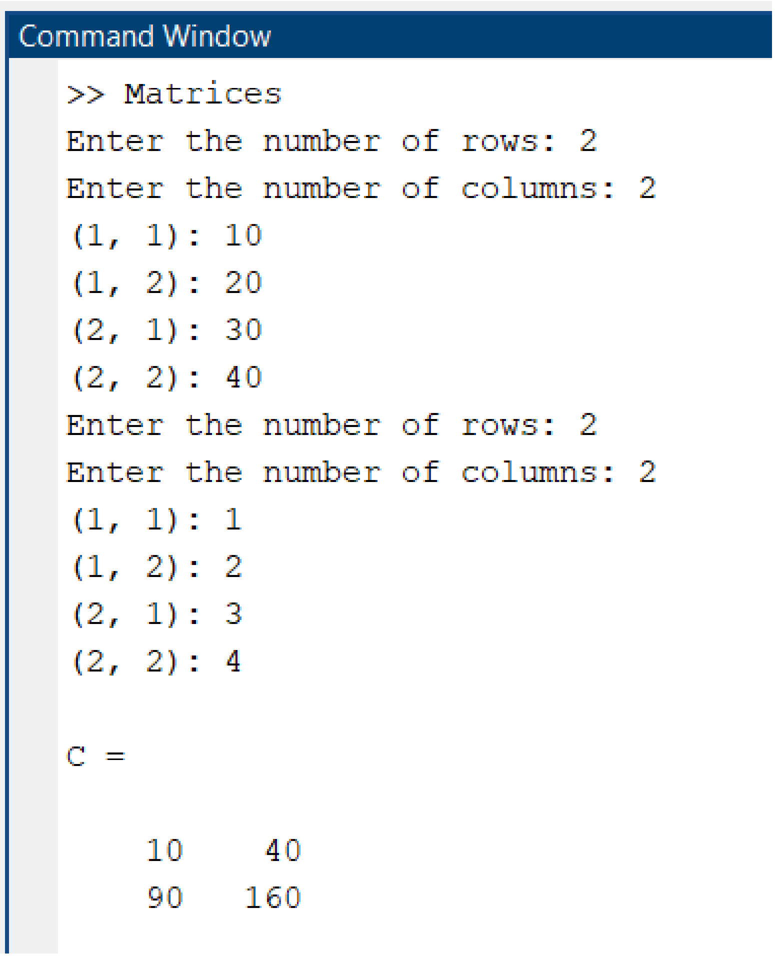
```
>> Matrices
Enter the number of rows: 2
Enter the number of columns: 3
(1, 1): 1
(1, 2): 2
(1, 3): 3
(2, 1): 4
(2, 2): 5
(2, 3): 6
Enter the number of rows: 3
Enter the number of columns: 2
(1, 1): 6
(1, 2): 5
(2, 1): 4
(2, 2): 3
(3, 1): 2
(3, 2): 1

C =

    20    14
    56    41
```

- Element-by-Element Matrix Multiplication

Code:

```
r1= input('Enter the number of rows: ');
c1 = input('Enter the number of columns: ');
A = zeros(r1, c1);
for i = 1:r1
   for j = 1:c1
      A(i, j) = input(sprintf('(%d, %d): ', i, j));
   end
end
r2= input('Enter the number of rows: ');
c2 = input('Enter the number of columns: ');
if r2==r1 && c2==c1
   B = zeros(r2, c2);
   for i = 1:r2
      for j = 1:c2
         B(i, j) = input(sprintf('(%d, %d): ', i, j));
      end
   end
   C = A .* B
else
   disp('Error: Matrices have different size. Execution Terminated.');
end
```

Output:



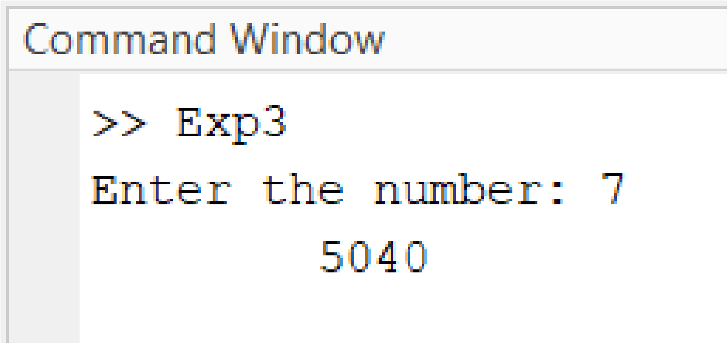
```
>> Matrices
Enter the number of rows: 2
Enter the number of columns: 2
(1, 1): 10
(1, 2): 20
(2, 1): 30
(2, 2): 40
Enter the number of rows: 2
Enter the number of columns: 2
(1, 1): 1
(1, 2): 2
(2, 1): 3
(2, 2): 4

C =

    10    40
    90   160
```

EXP 3: Program to calculate the factorial of a number by creating a script file by using while loop

Code:

```
n=input("Enter the number: ");
f=1;
while 1
   if n>0
      f=f*n;
      n=n-1;
   else
      break
   end
end
disp(f)
```

Output:



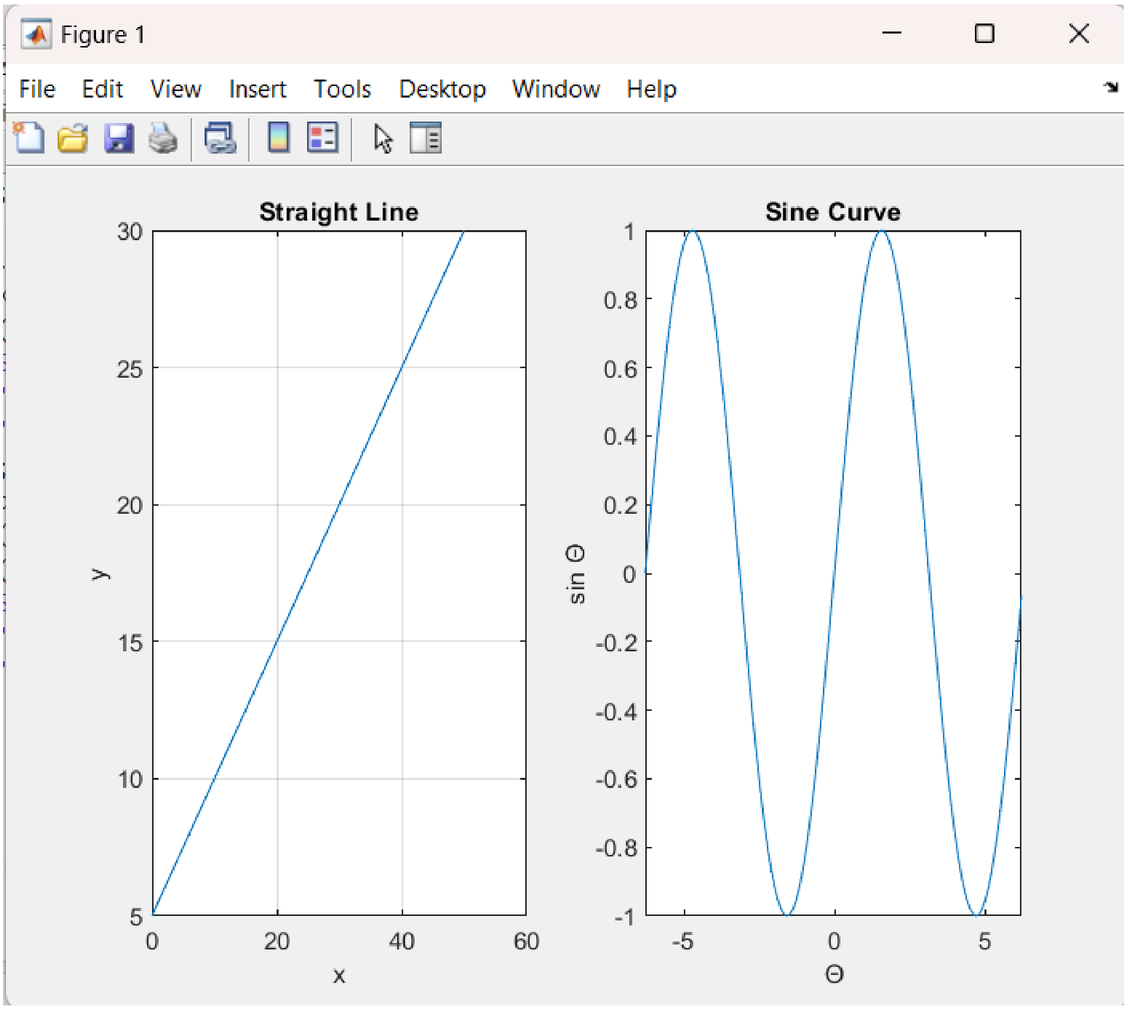
```
>> Exp3
Enter the number: 7
        5040
```

EXP 4: Program to plot the straight line and sine curve

Code:

```
m = 0.5;
c = 5;
x = 0:0.1:50;
y = (m*x)+c;
subplot(1,2,1), plot(x,y)
title('Straight Line');
xlabel('x');
ylabel('y');
grid on;
t = -2*pi:0.1:2*pi;
a = sin(t);
subplot(1,2,2), plot(t,a)
title('Sine Curve');
xlabel('θ');
ylabel('sin θ');
```

Output:

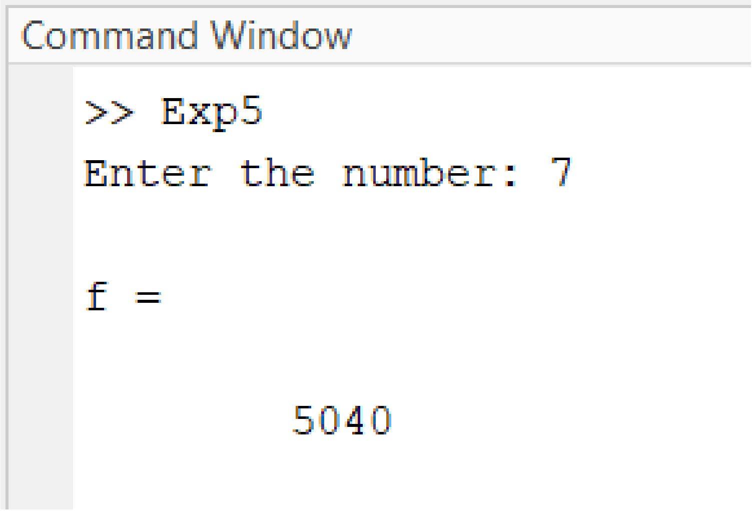


EXP 5: Program in MATLAB to find the factorial by creating a function file by using for loop

Code:

```
n=input("Enter the number: ");
f = fact(n)
function f = fact(n)
    f=1;
    for i=1:n
        f=f*i;
    end
end
```

Output:



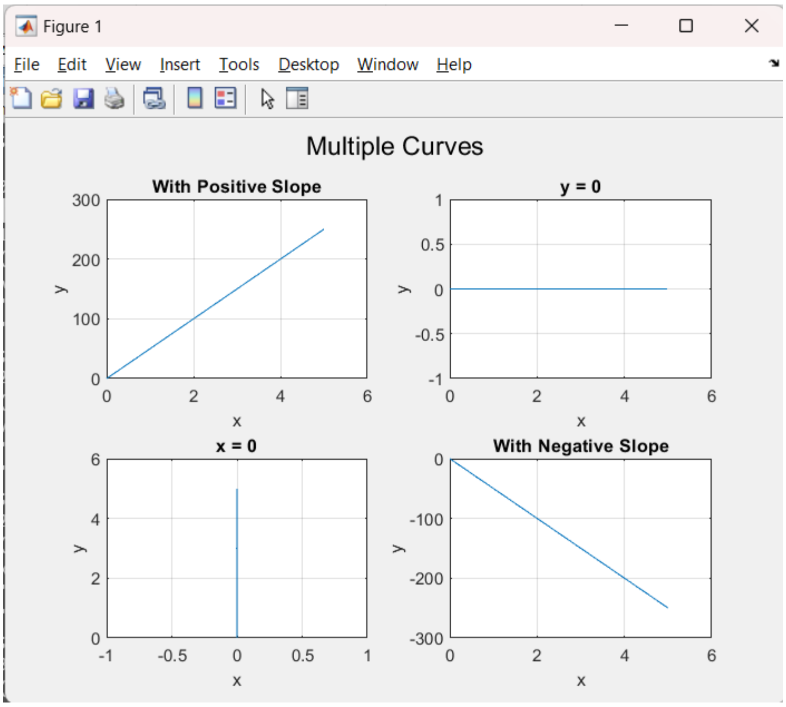
```
>> Exp5
Enter the number: 7

f =

        5040
```

EXP 6: Program to draw a graph with multiple curves

Code:

```
x = 0:0.1:5;
y = 0:0.1:5;
m1 = 50;
m2 = 0;
m3 = -50;
c = 0;
y1 = (m1*x)+c;
subplot(2,2,1), plot(x,y1)
title('With Positive Slope');
xlabel('x');
ylabel('y');
grid on;
y2 = (m2*x)+c;
subplot(2,2,2), plot(x,y2)
title('y = 0');
xlabel('x');
ylabel('y');
grid on;
x1 = (m2*y)+c;
subplot(2,2,3), plot(x1,y)
title('x = 0');
xlabel('x');
ylabel('y');
grid on;
y3 = (m3*x)+c;
subplot(2,2,4), plot(x,y3)
title('With Negative Slope');
xlabel('x');
ylabel('y');
grid on;
sgtitle('Multiple Curves');
```

Output:



Machine Learning Onramp

~Space for Certificate - Machine Learning Onramp~

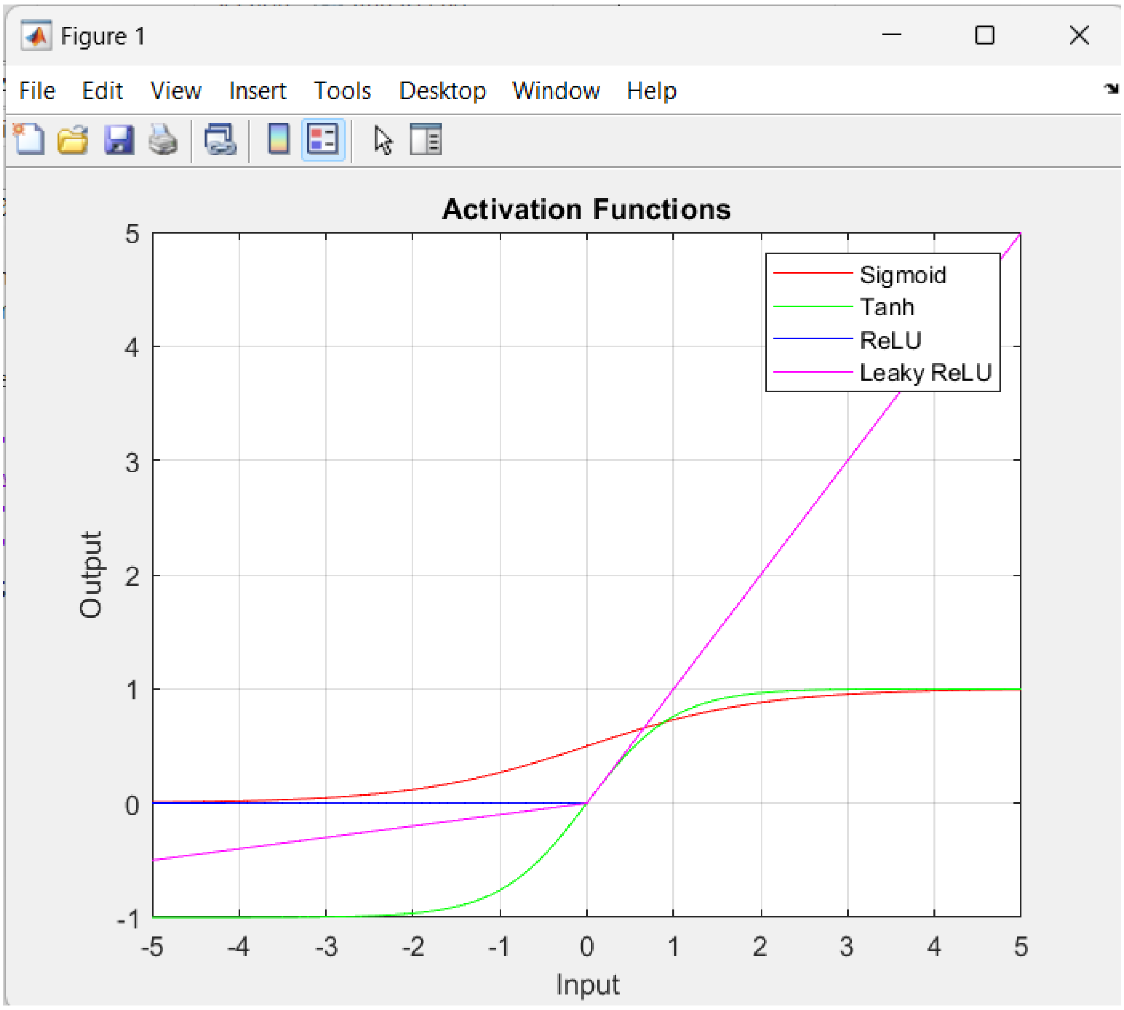
EXP 7: Program to plot Activation Function used in Neural Network

Method-1:

Code:

```
x = -5:0.1:5;
sigmoid = 1./(1 + exp(-x));
tanh_func = tanh(x);
relu = max(0, x);
alpha = 0.1;
leaky_relu = max(alpha * x, x);
plot(x, sigmoid, 'r', x, tanh_func, 'g', x, relu, 'b', x, leaky_relu, 'm');
legend('Sigmoid', 'Tanh', 'ReLU', 'Leaky ReLU');
title('Activation Functions');
xlabel('Input');
ylabel('Output');
grid on;
```

Output:



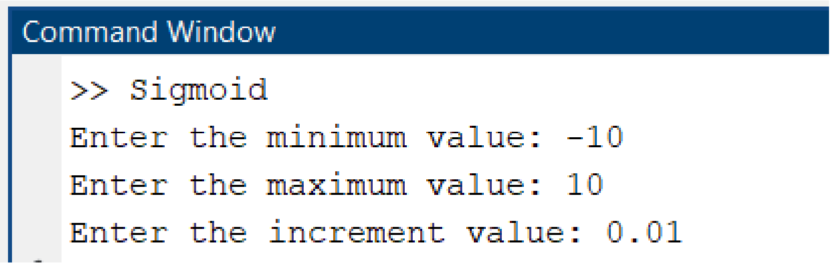
Method-2:

a. Sigmoid Function:

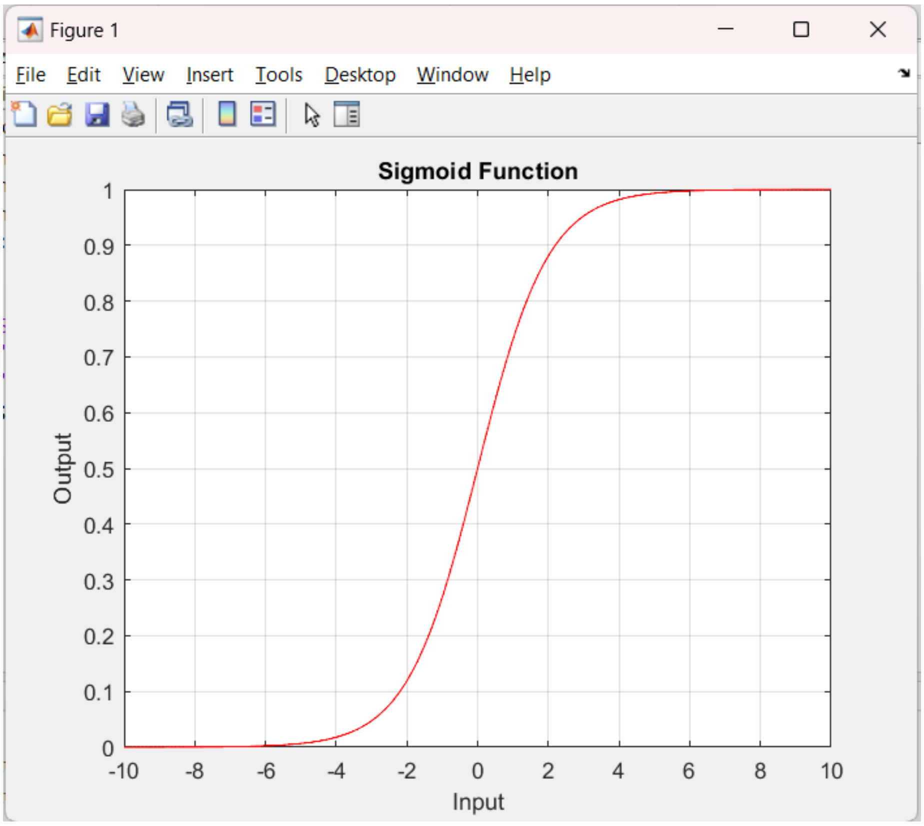
Code:

```
min = input('Enter the minimum value: ');
max = input('Enter the maximum value: ');
inc = input('Enter the increment value: ');
x = min:inc:max;
sigmoid = 1./(1 + exp(-x));
plot(x, sigmoid, 'r');
title('Sigmoid Function');
xlabel('Input');
ylabel('Output');
grid on;
```

Output:



```
>> Sigmoid
Enter the minimum value: -10
Enter the maximum value: 10
Enter the increment value: 0.01
```

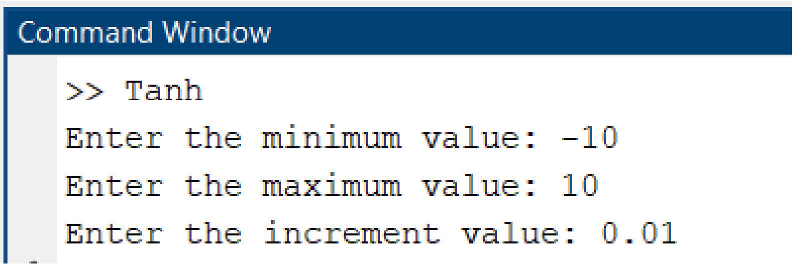


b. Tanh Function:

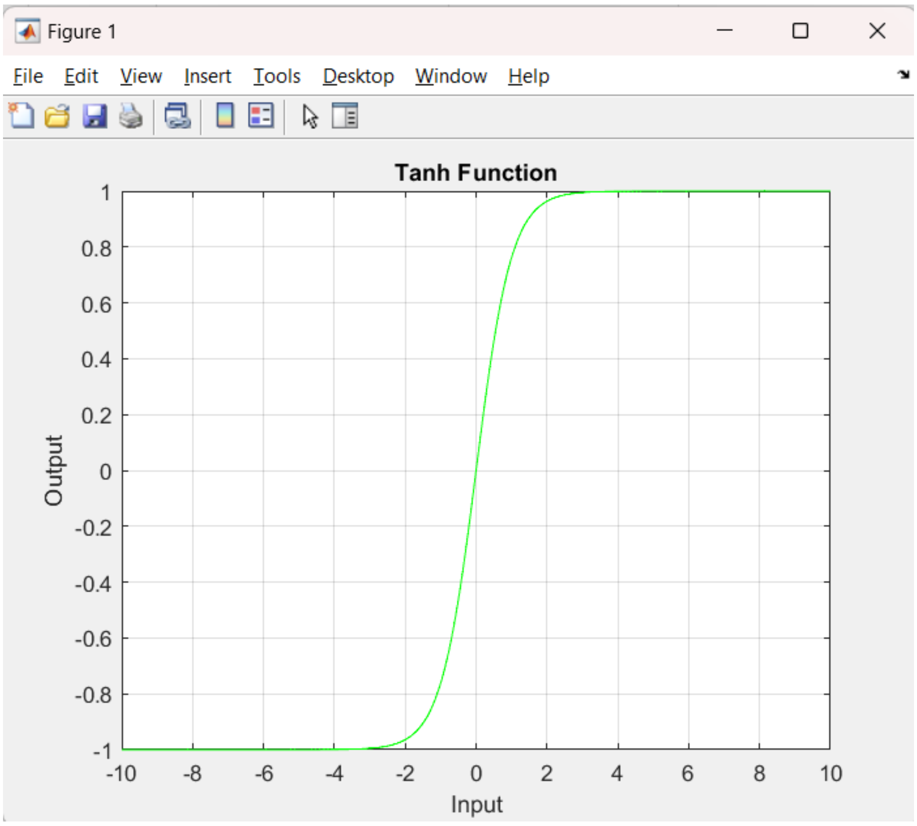
Code:

```
min = input('Enter the minimum value: ');
max = input('Enter the maximum value: ');
inc = input('Enter the increment value: ');
x = min:inc:max;
tanh_func = tanh(x);
plot(x, tanh_func, 'g');
title('Tanh Function');
xlabel('Input');
ylabel('Output');
grid on;
```

Output:



```
>> Tanh
Enter the minimum value: -10
Enter the maximum value: 10
Enter the increment value: 0.01
```

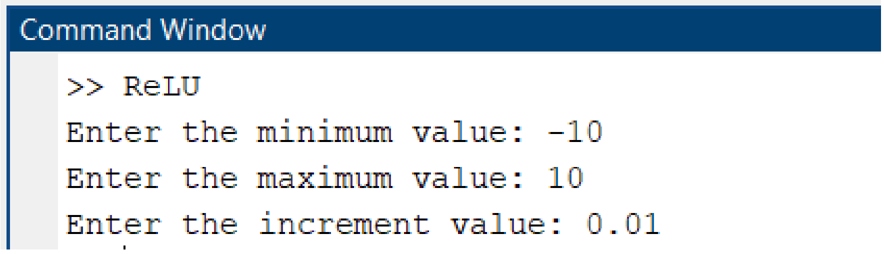


c. ReLU Function:

Code:

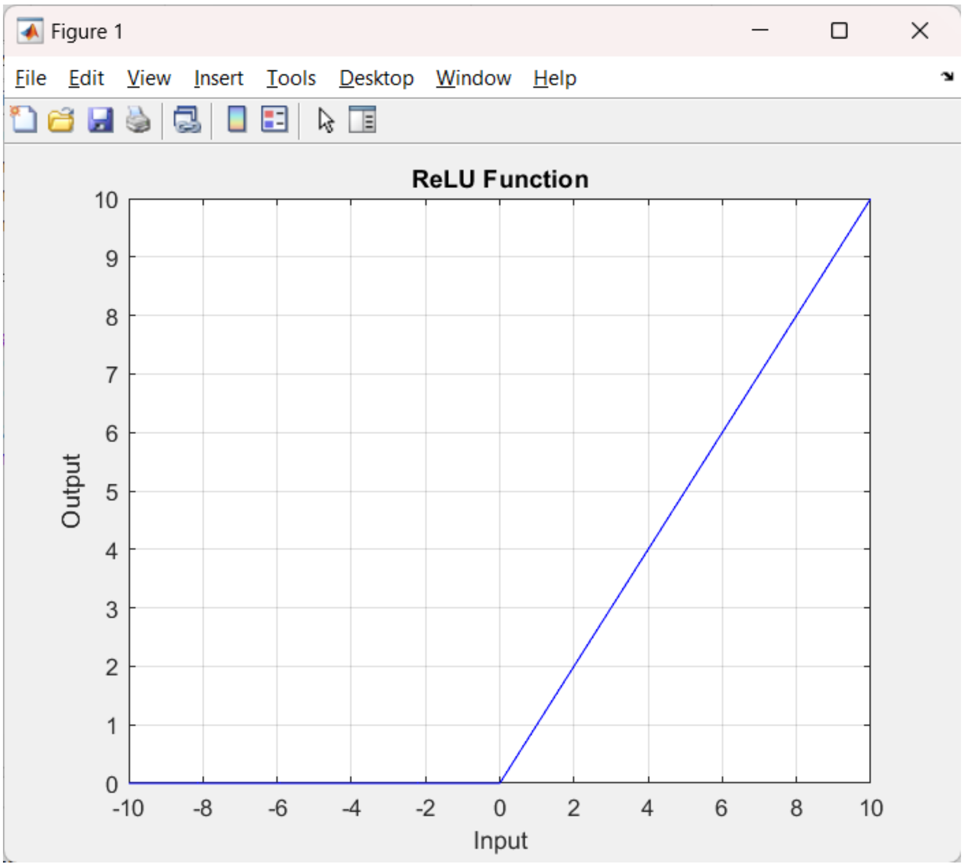
```
min = input('Enter the minimum value: ');
max = input('Enter the maximum value: ');
inc = input('Enter the increment value: ');
x = min:inc:max;
relu = func(x);
plot(x, relu, 'b');
title('ReLU Function');
xlabel('Input');
ylabel('Output');
grid on;
function y = func(x)
    y = max(0, x);
end
```

Output:



Command Window

```
>> ReLU
Enter the minimum value: -10
Enter the maximum value: 10
Enter the increment value: 0.01
```

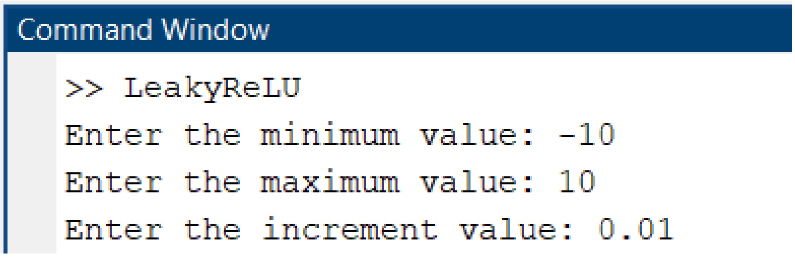


d. Leaky ReLU Function:

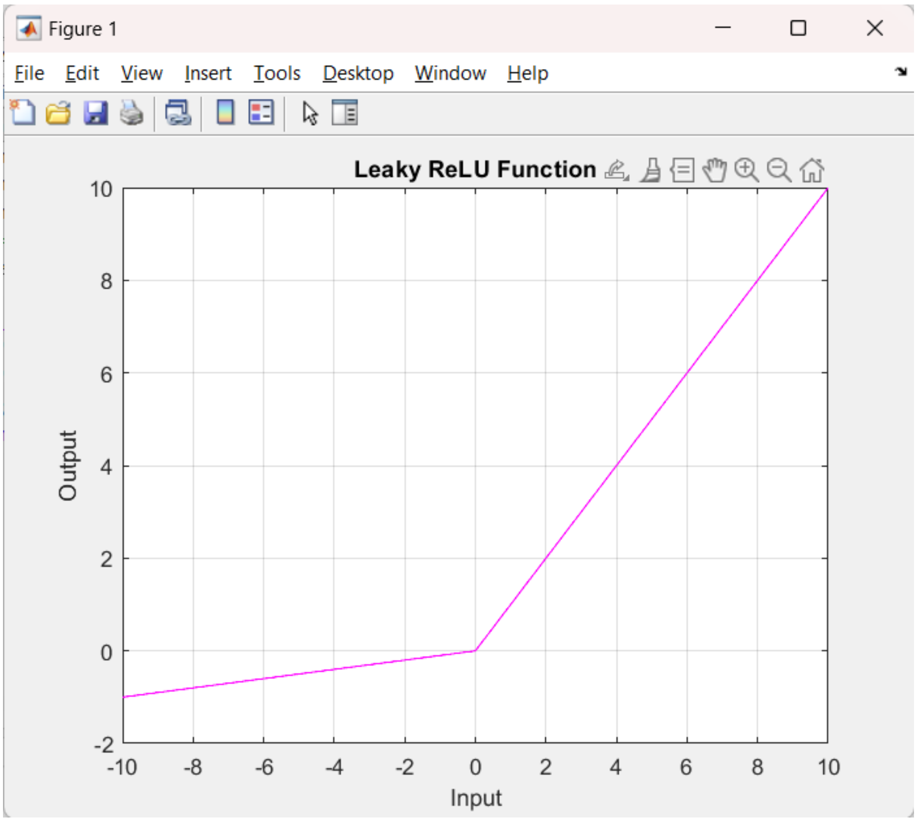
Code:

```
min = input('Enter the minimum value: ');
max = input('Enter the maximum value: ');
inc = input('Enter the increment value: ');
leaky_relu = func(x);
plot(x, leaky_relu, 'm');
title('Leaky ReLU Function');
xlabel('Input');
ylabel('Output');
grid on;
function y = func(x)
    y = max(0.1*x, x);
end
```

Output:



```
>> LeakyReLU
Enter the minimum value: -10
Enter the maximum value: 10
Enter the increment value: 0.01
```

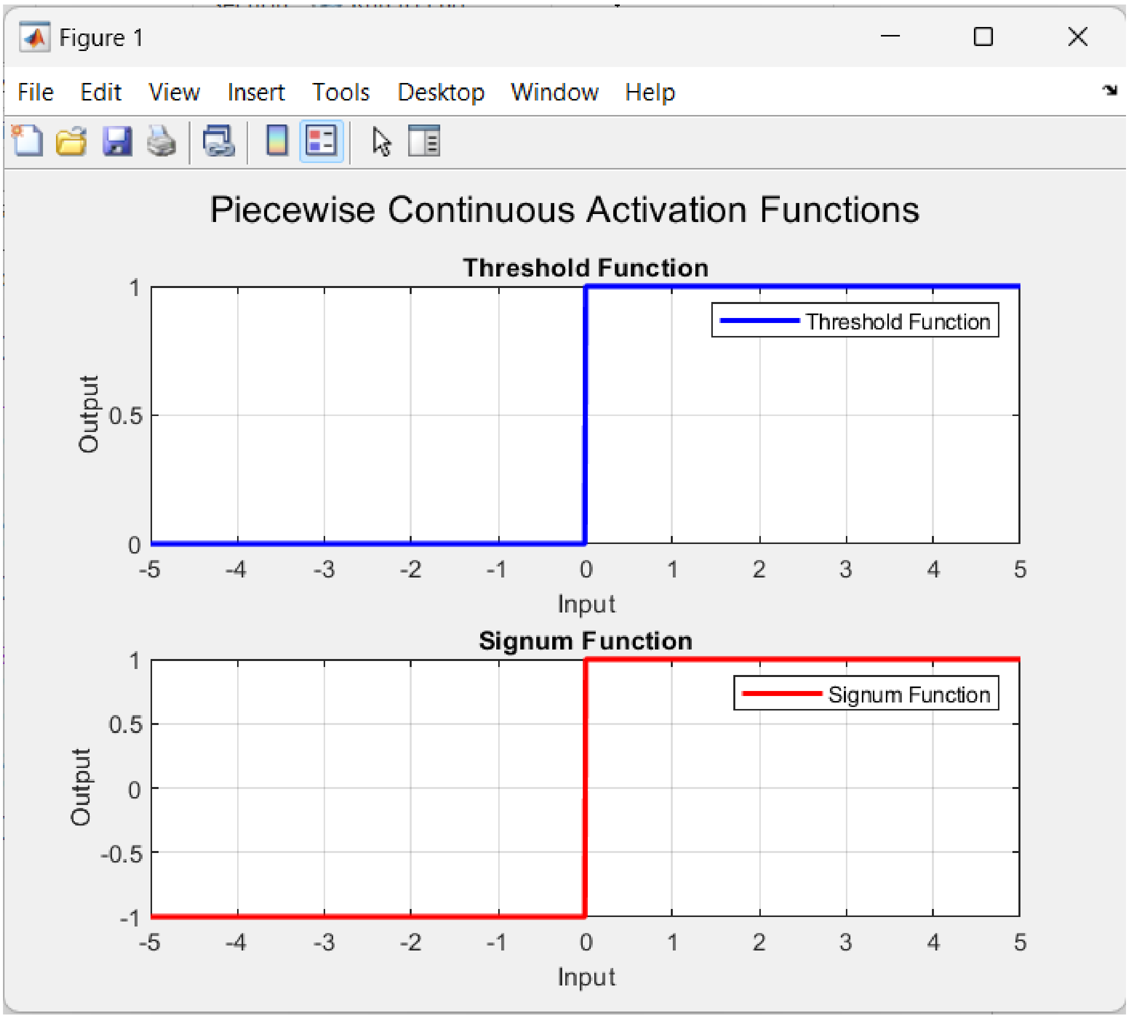


EXP 8: Program to plot piecewise continuous activation function (threshold and signum function in neural network)

Code:

```
x = linspace(-5, 5, 1000);
threshold_output = (x >= 0);
signum_output = sign(x);
figure;
subplot(2, 1, 1);
plot(x, threshold_output, 'b', 'LineWidth', 2);
title('Threshold Function');
xlabel('Input');
ylabel('Output');
grid on;
legend('Threshold Function');
subplot(2, 1, 2);
plot(x, signum_output, 'r', 'LineWidth', 2);
title('Signum Function');
xlabel('Input');
ylabel('Output');
grid on;
legend('Signum Function');
sgtitle('Piecewise Continuous Activation Functions');
```

Output:

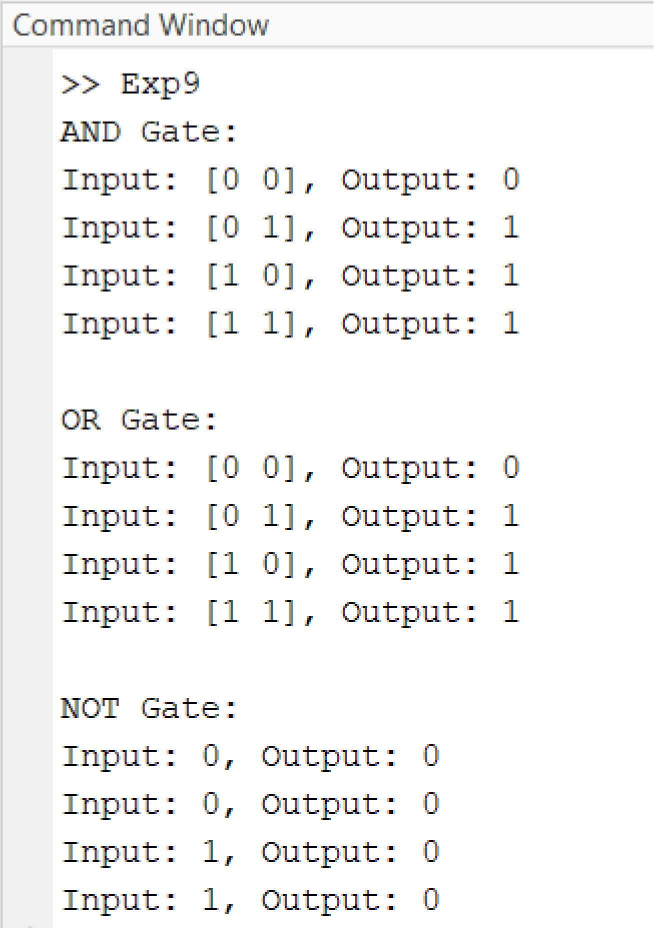


EXP 9: Program to realize gates using McCulloh Pitt model in MATLAB

Code:

```
threshold = 0;
n = [0 0; 0 1; 1 0; 1 1];
fprintf('AND Gate:\n');
w_and = [1 1];
for i = 1:size(n, 1)
    act_and = sum(n(i,:) .* w_and);
    out_and = act_and > threshold;
    fprintf('Input: [%d %d], Output: %d\n', n(i,1), n(i,2), out_and);
end
fprintf('\nOR Gate:\n');
w_or = [1 1];
for i = 1:size(n, 1)
    act_or = sum(n(i,:) .* w_or);
    out_or = act_or > threshold;
    fprintf('Input: [%d %d], Output: %d\n', n(i,1), n(i,2), out_or);
end
fprintf('\nNOT Gate:\n');
w_not = -1;
for i = 1:size(n, 1)
    act_not = n(i) * w_not;
    out_not = act_not > threshold;
    fprintf('Input: %d, Output: %d\n', n(i), out_not);
end
```

Output:



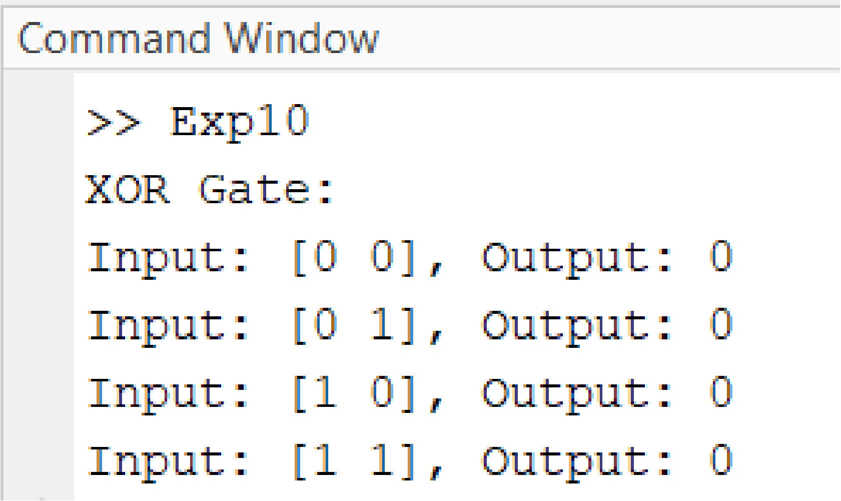
```
>> Exp9
AND Gate:
Input: [0 0], Output: 0
Input: [0 1], Output: 1
Input: [1 0], Output: 1
Input: [1 1], Output: 1

OR Gate:
Input: [0 0], Output: 0
Input: [0 1], Output: 1
Input: [1 0], Output: 1
Input: [1 1], Output: 1

NOT Gate:
Input: 0, Output: 0
Input: 0, Output: 0
Input: 1, Output: 0
Input: 1, Output: 0
```

EXP 10: Program to implement XOR gate using McCulloh-Pitts neuron

Code:

```
t = 0;
n = [0 0; 0 1; 1 0; 1 1];
w1 = [-1 -1; -1 -1; 1 1; 1 1];
b1 = [-0.5; -1.5; 0.5; -0.5];
w2 = [1; 1; -1; 1];
b2 = -0.5;
fprintf('XOR Gate:\n');
for i = 1:size(n, 1)
    out1 = sum(n(i,:) .* w1, 2) + b1;
    out1 = out1 > t;
    out_xor = sum(out1 .* w2) + b2;
    out_xor = out_xor > t;
    fprintf('Input: [%d %d], Output: %d\n', n(i,1), n(i,2), out_xor);
end
```

Output:



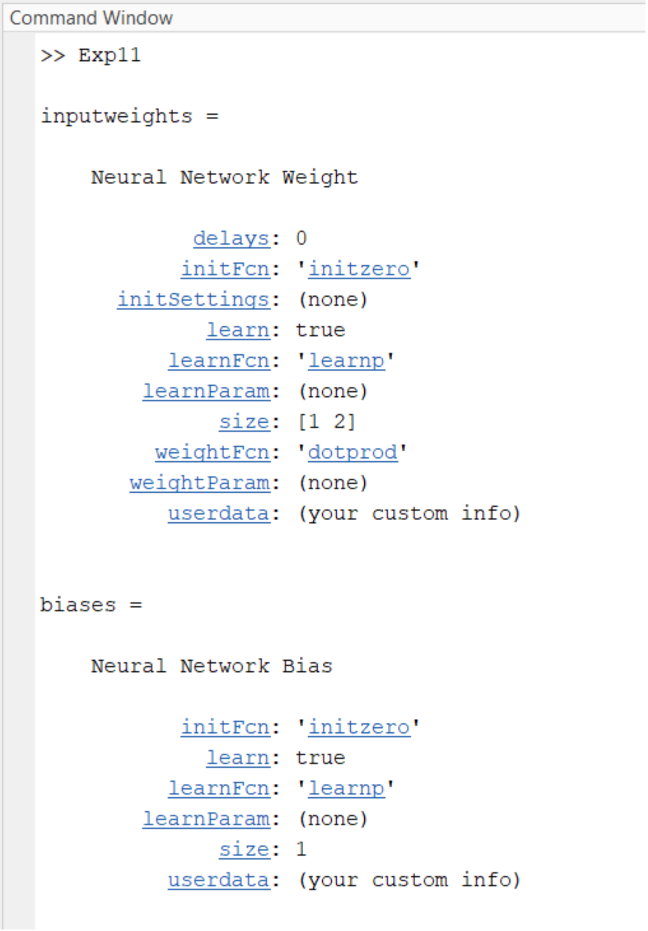
```
>> Exp10
XOR Gate:
Input: [0 0], Output: 0
Input: [0 1], Output: 0
Input: [1 0], Output: 0
Input: [1 1], Output: 0
```

EXP 11: Program to create Perceptron using commands

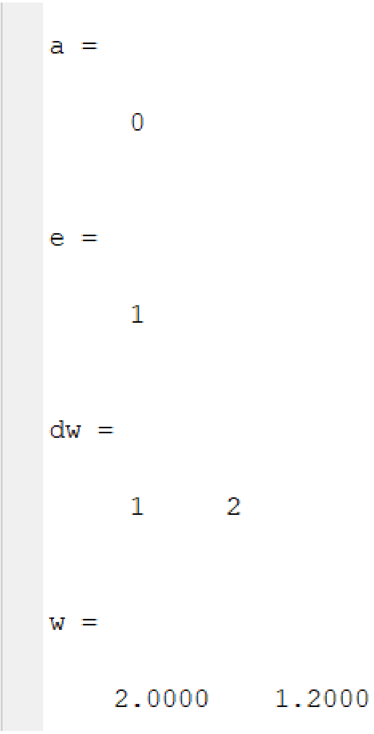
Code:

```
net = perceptron;
net = configure(net,[0;0],0);
inputweights = net.inputweights{1,1}
biases = net.biases{1}
net.b{1} =  [0];
w = [1 -0.8];
net.IW{1,1} = w;
p = [1; 2];
t = [1];
a = net(p)
e = t-a
dw = learnp(w,p,[],[],[],[],e,[],[],[],[],[])
w = w + dw
```

Output:



```
>> Exp11

inputweights =

    Neural Network Weight

            delays: 0
           initFcn: 'initzero'
      initSettings: (none)
             learn: true
          learnFcn: 'learnp'
        learnParam: (none)
              size: [1 2]
         weightFcn: 'dotprod'
       weightParam: (none)
          userdata: (your custom info)


biases =

    Neural Network Bias

           initFcn: 'initzero'
             learn: true
          learnFcn: 'learnp'
        learnParam: (none)
              size: 1
          userdata: (your custom info)
```



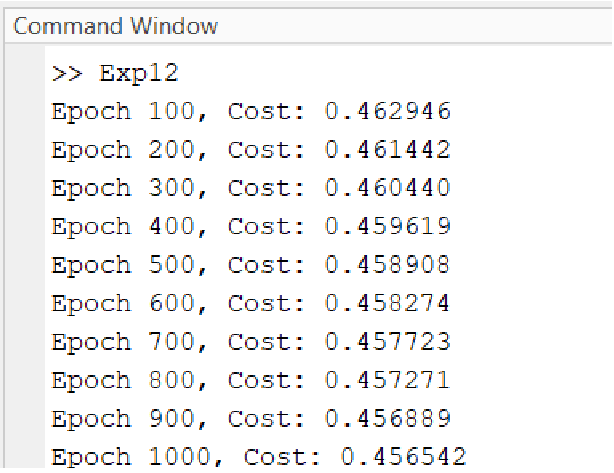
```
a =

     0


e =

     1


dw =

     1     2


w =

    2.0000    1.2000
```

EXP 12: Program for creating a Back Propagation Feed-forward neural network

Code:

```
num_samples = 1000;
num_features = 5;
num_outputs = 1;
X = randn(num_samples, num_features);
Y = randn(num_samples, num_outputs);
inputSize = size(X, 2);
hiddenSize = 10;
outputSize = size(Y, 2);
W1 = randn(inputSize, hiddenSize);
b1 = zeros(1, hiddenSize);
W2 = randn(hiddenSize, outputSize);
b2 = zeros(1, outputSize);
learningRate = 0.01;
numEpochs = 1000;
for epoch = 1:numEpochs
   z1 = X * W1 + b1;
   a1 = sigmoid(z1);
   z2 = a1 * W2 + b2;
   Y_pred = sigmoid(z2);
   delta2 = (Y_pred - Y) .* sigmoidGradient(z2);
   delta1 = (delta2 * W2') .* sigmoidGradient(z1);
   W2 = W2 - learningRate * (a1' * delta2);
   b2 = b2 - learningRate * sum(delta2);
   W1 = W1 - learningRate * (X' * delta1);
   b1 = b1 - learningRate * sum(delta1);
   cost = 0.5 * sum(sum((Y_pred - Y).^2)) / size(X, 1);
   if mod(epoch, 100) == 0
      fprintf('Epoch %d, Cost: %f\n', epoch, cost);
   end
end
function sigm = sigmoid(x)
   sigm = 1 ./ (1 + exp(-x));
end
function sigmGradient = sigmoidGradient(x)
   sigmGradient = sigmoid(x) .* (1 - sigmoid(x));
end
```

Output:



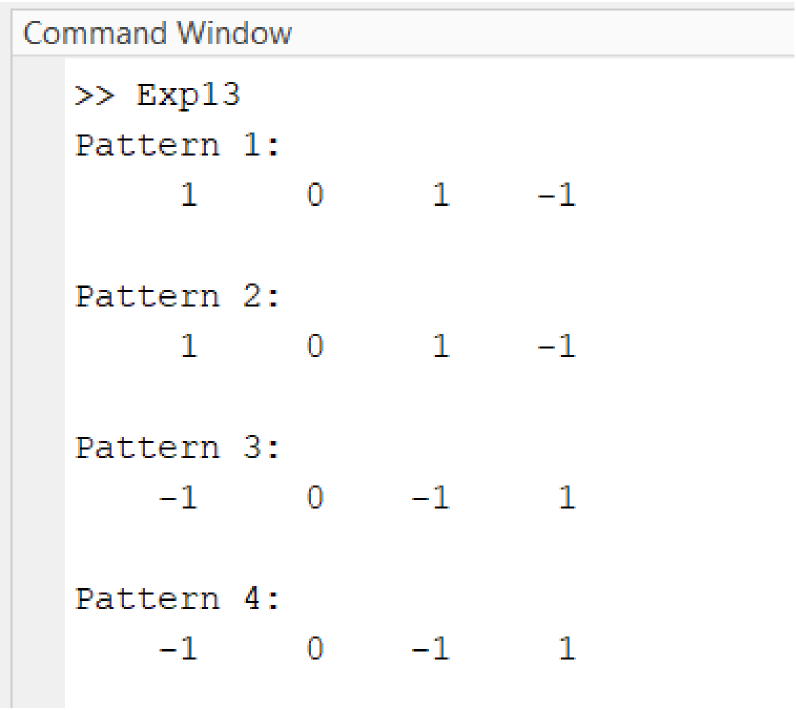
```
>> Exp12
Epoch 100, Cost: 0.462946
Epoch 200, Cost: 0.461442
Epoch 300, Cost: 0.460440
Epoch 400, Cost: 0.459619
Epoch 500, Cost: 0.458908
Epoch 600, Cost: 0.458274
Epoch 700, Cost: 0.457723
Epoch 800, Cost: 0.457271
Epoch 900, Cost: 0.456889
Epoch 1000, Cost: 0.456542
```

EXP 13: Program to design a Hopfield Network which stores 4 vectors

Code:

```
patterns = [1 1 1 -1;
         1 -1 1 -1;
         -1 1 -1 1;
         -1 -1 -1 1];
num_neurons = size(patterns, 2);
W = zeros(num_neurons);
for i = 1:size(patterns, 1)
   W = W + patterns(i, :)' * patterns(i, :);
end
W(logical(eye(size(W)))) = 0;
theta = zeros(1, num_neurons);
for i = 1:size(patterns, 1)
   recalled_pattern = recall(patterns(i, :), W, theta, 10);
   disp(['Pattern ', num2str(i), ':']);
   disp(recalled_pattern);
end
function output = recall(input, W, theta, max_iter)
   output = sign(input * W - theta);
   for i = 1:max_iter
      output = sign(output * W - theta);
   end
end
```

Output:



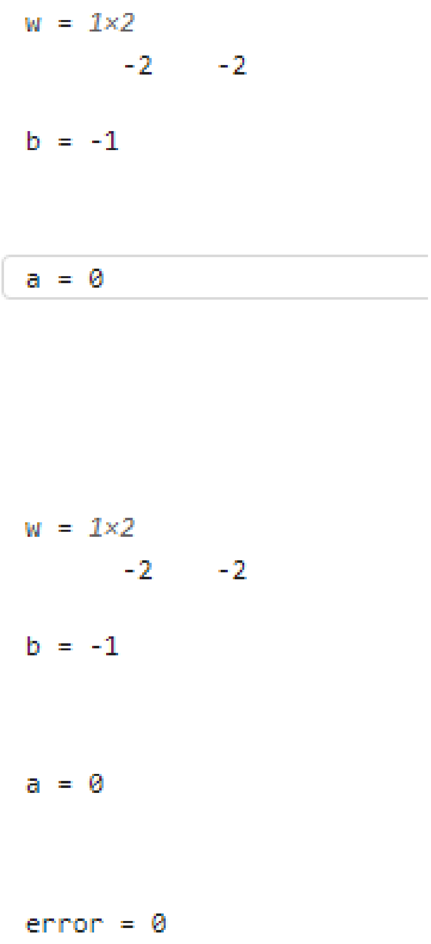
```
>> Exp13
Pattern 1:
     1     0     1    -1

Pattern 2:
     1     0     1    -1

Pattern 3:
    -1     0    -1     1

Pattern 4:
    -1     0    -1     1
```

EXP 14: Program to illustrate how the perception learning rule works for non-linearly separable problems

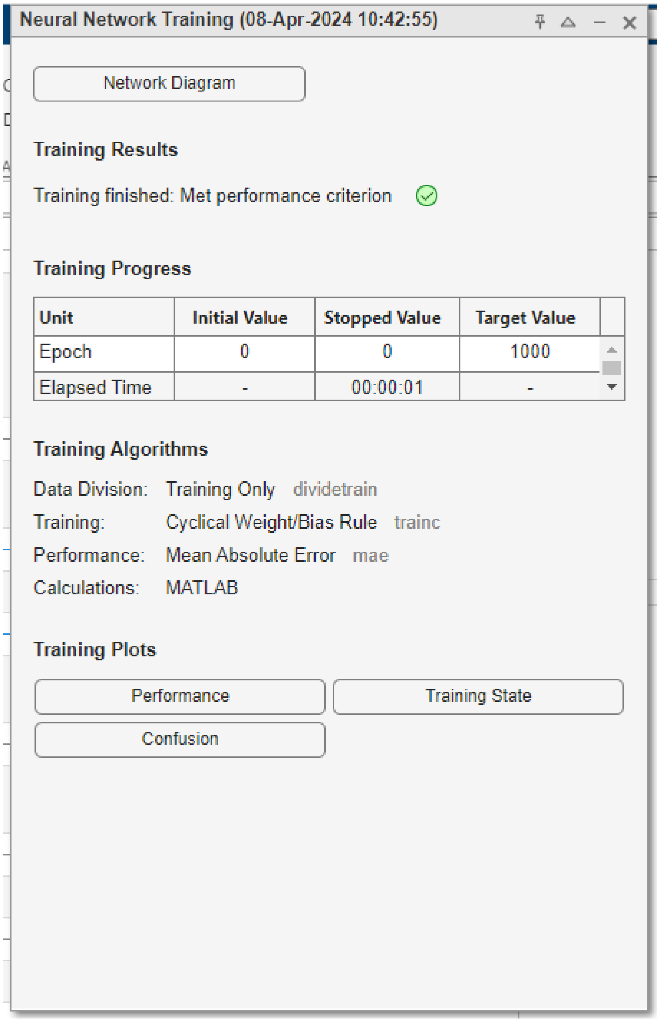
Code:

```
net = perceptron;
p = [2; 2];
t = [0];
net.trainParam.epochs = 1;
net = train(net,p,t);
w = net.iw{1,1}
b = net.b{1}
a = net(p)
net.trainParam.epochs = 1000;
net = train(net,p,t);
w = net.iw{1,1}
b = net.b{1}
```

Output:



```
w = 1×2
      -2     -2

b = -1


a = 0




w = 1×2
      -2     -2

b = -1


a = 0


error = 0
```



Neural Network Training (08-Apr-2024 10:42:55)

Training Results

Training finished: Met performance criterion ✓

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	0	1000
Elapsed Time	-	00:00:01	-

Training Algorithms

Data Division: Training Only dividetrain

Training: Cyclical Weight/Bias Rule trainc

Performance: Mean Absolute Error mae

Calculations: MATLAB

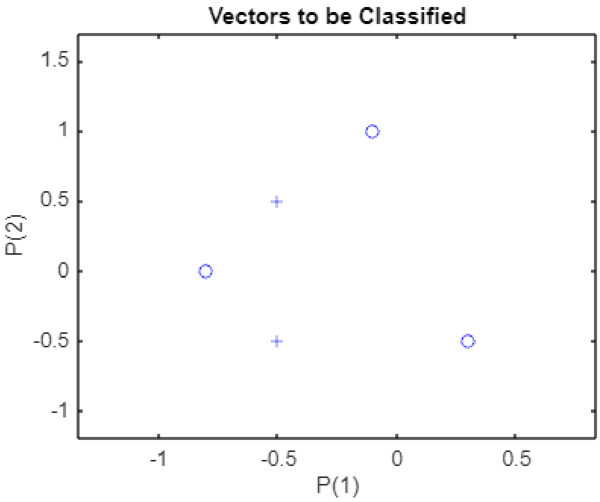
Training Plots

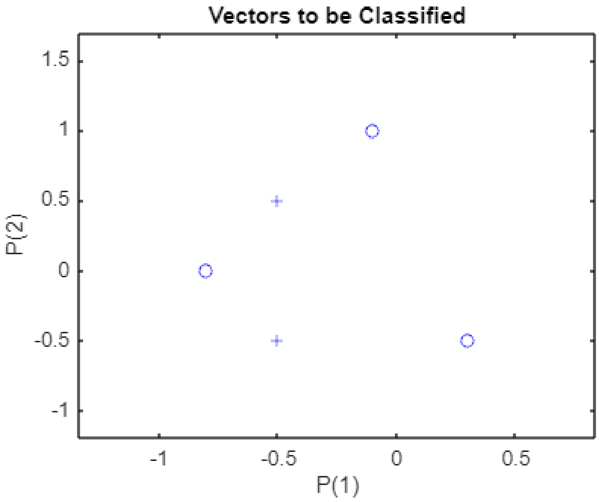
EXP 15: Program to illustrate linearly non-separable vectors

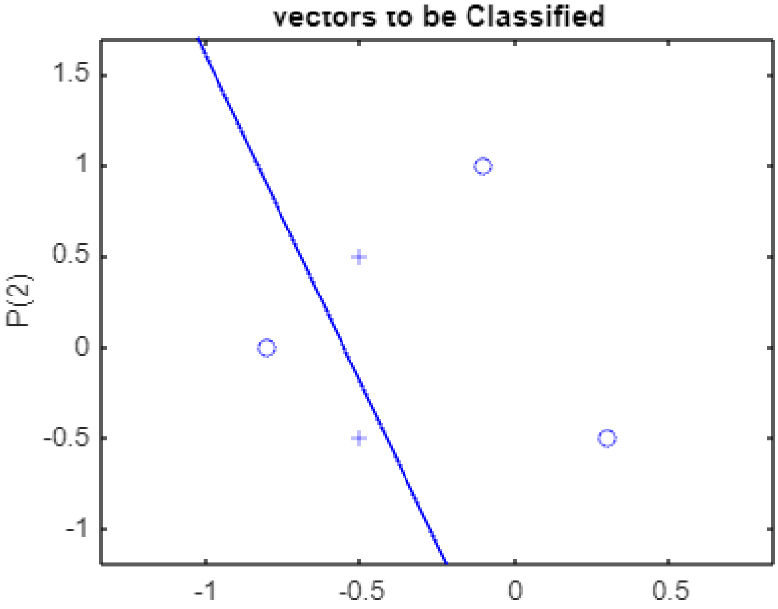
Code:

```
X = [ -0.5 -0.5 +0.3 -0.1 -0.8; ...
     -0.5 +0.5 -0.5 +1.0 +0.0 ];
T = [1 1 0 0 0];
plotpv(X,T);
net = perceptron;
net = configure(net,X,T);
hold on
plotpv(X,T);
linehandle = plotpc(net.IW{1},net.b{1});
for a = 1:25
  [net,Y,E] = adapt(net,X,T);
  linehandle = plotpc(net.IW{1},net.b{1},linehandle);  drawnow;
end
```

Output:


Vectors to be Classified
P(2)
P(1)


Vectors to be Classified
P(2)
P(1)


vectors to be Classified
P(2)