MATLAB PROJECT 1

Please include this page in your Group file, as a front page. Type in the group number and the names of all members WHO PARTICIPATED in this project.

GROUP # ____5_____

FIRST & LAST NAMES (UFID numbers are NOT required):

_1.___Ekaterina Krysova_____

_2.___Elton Li_____

_3.___Thomas Pena Reina_____

_4.____Dany Rashwan_____

_5._____Manuel Vera_____

_6._____Corey Wolfe_____

By including your names above, each of you had confirmed that you did the work and agree with the work submitted.

Project 1

Exercise 1

```
format
format compact
type jord
```

```
function J=jord(n,r)
    J = [];
 if (mod(n, 1) ~= 0 || n <= 1)
  disp('Jordan Block cannot be built')
  return
    end
 J = diag(diag(eye(n - 1)), 1) + r * eye(n);
end
```

```
r=rand(1)
```

```
r = 0.9649
```

```
%(a)
n=0;
jord(n,r)
```

```
Jordan Block cannot be built
ans =
     []
```

```
%(b)
n=-2;
jord(n,r)
```

```
Jordan Block cannot be built
ans =
     []
```

```
%(c)
n=3.5;
jord(n,r)
```

```
Jordan Block cannot be built
ans =
     []
```

```
%(d)
n=-2.5;
jord(n,r)
```

```
Jordan Block cannot be built
ans =
     []
```

```
%(e)
n=4;
jord(n,r)
```

```
ans = 4×4
    0.9649    1.0000         0         0
```

```
0    0.9649    1.0000         0
0         0    0.9649    1.0000
0         0         0    0.9649
```

Exercise 2

```
type added
```

```
%Creates the function 'added'
function C = added(A,B)

%First, the function verifies whether matrices A and B have the same size.
[a1,b1] = size(A);
[a2,b2] = size(B);
C = [];

%If not, 'the matrices are not of the same size and cannot be added',
%Then assigns an empty matrix to C. After that, the program terminates.
if(a1 ~= a2 || b1 ~= b2)
    disp('the matrices are not of the same size and cannot be added');
    C = [];
    return;

%If matrix can be added, calculates sum C of A + B using for loops.
%Outputs and displays C
else
    for i = 1 : a1
    for j = 1 : b1

%Sum C - option 2
C(i,j) = A(i,j) + B(i,j);
    end
    end
end

%Logical “if” statement to verify whether the calculated matrix C matches
%the output for a built-in MATLAB function A+B.

M = A + B;
    for i = 1 : a1
    for j = 1 : b1
%If the outputs C and A+B do not match, outputs 'check your code!'
if(M(i,j) ~= C(i,j))
disp('check your code!');
return;
end
    end
    end
end
```

Part A

```
A=magic(3), B=ones(4)
```

```
A = 3×3
     8     1     6
     3     5     7
     4     9     2
B = 4×4
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

```
added(A,B)
```

```
the matrices are not of the same size and cannot be added
```

```
ans =

     []
```

Part B

```
A=ones(3,4), B=ones(3,3)
```

```
A = 3×4
     1     1     1     1
     1     1     1     1
     1     1     1     1
B = 3×3
     1     1     1
     1     1     1
     1     1     1
```

```
added(A,B)
```

```
the matrices are not of the same size and cannot be added

ans =

     []
```

Part C

```
A=randi(100,3,4), B=randi(100,3,4)
```

```
A = 3×4
    71     5    70     4
     4    10    32    44
    28    83    96    39
B = 3×4
    77    49    71    68
    80    45    76    66
    19    65    28    17
```

```
added(A,B)
```

```
ans = 3×4
   148    54   141    72
    84    55   108   110
    47   148   124    56
```

Parts 1 & 2

```
%Runs function for parts 1 and 2
C = added(A,B)
```

```
C = 3×4
   148    54   141    72
    84    55   108   110
    47   148   124    56
```

```
%Inputs the scalar to test for parts 1 and 2
k=fix(10*rand(1))+5;
```

Part 1 - Commutative property

```
%Verifies if the sum of A and B is the same as the sum of B and A.
%If it is the case, the program outputs 'commutative property holds for the given A and B'
C1 = added(B,A);
if isequal(C,C1)
    disp('commutative property holds for the given A and B');
else
    return;
end
```

```
commutative property holds for the given A and B
```

Part 2 - Distributive property

```
%Verifies that the product of a scalar k by the matrix C, kC, is the same as the sum of
%kA and kB. If this is the case, the program outputs
%'distributive property holds for the given A and B'
C1 = added(k*B,k*A);
if isequal(k*C, C1)
    disp('distributive property holds for the given A and B');
else
    return;
end
```

```
distributive property holds for the given A and B
```

Exercise 3

```
type givensrot
```

```
function G=givensrot(n,i,j,theta)

if (1 > i || i >= j || j > n || n < 2)
    G = [];
    return
end

G = eye(n);
G(i,i) = cos(theta);
G(j,j) = cos(theta);
G(i,j) = -sin(theta);
G(j,i) = sin(theta);




end
```

```
G=givensrot(1,1,2,pi)
```

```
G =

     []
```

```
G=givensrot(4,3,2,pi/2)
```

```
G =

     []
```

```
G=givensrot(5,2,4,pi/4)
```

```
G = 5×5
    1.0000         0         0         0         0
         0    0.7071         0   -0.7071         0
         0         0    1.0000         0         0
         0    0.7071         0    0.7071         0
         0         0         0         0    1.0000
```

```
G=givensrot(2,1,2,-pi/2)
```

```
G = 2×2
    0.0000    1.0000
   -1.0000    0.0000
```

```
G=givensrot(3,1,2,pi)
```

```
G = 3×3
   -1.0000   -0.0000         0
    0.0000   -1.0000         0
         0         0    1.0000
```

```
I = eye(3)
```

```
I = 3×3
     1     0     0
     0     1     0
     0     0     1
```

```
e1 = [1; 0; 0]
```

```
e1 = 3×1
     1
     0
     0
```

```
e2 = [0; 1; 0]
```

```
e2 = 3×1
     0
     1
     0
```

```
e3 = [0; 0; 1]
```

```
e3 = 3×1
     0
     0
     1
```

```
GIp = [-1 0 0; 0 -1 0; 0 0 1] % predicted
```

```
GIp = 3×3
    -1     0     0
     0    -1     0
     0     0     1
```

```
G * e1  %test column one
```

```
ans = 3×1
   -1.0000
    0.0000
         0
```

```
G * e2 %test column two
```

```
ans = 3×1
   -0.0000
   -1.0000
         0
```

```
G * e3 %test column three
```

```
ans = 3×1
     0
     0
     1
```

```
GIa = G * I %actual
```

```
GIa = 3×3
   -1.0000   -0.0000         0
    0.0000   -1.0000         0
         0         0    1.0000
```

```
if (closetozeroroundoff(GIa, 7) == closetozeroroundoff(GIp, 7))
    fprintf 'your prediction is correct!'
end
```

```
your prediction is correct!
```

```
type closetozeroroundoff
```

```
function B=closetozeroroundoff(A,p)
A(abs(A)<10^-p)=0;
B=A;
end
```

```
x = ones(3,1); %definition vector

Gx = G * x %vector Gx which is x under transformationd defined by G
```

```
Gx = 3×1
   -1.0000
   -1.0000
    1.0000
```

Exercise 4

Part 1

```
type toeplitze
```

```
function A=toeplitze(m,n,a)

% check wheter the number of entries in the vector (a) is (m+n-1)
i = 1:m; j=1:n;
if (m+n-1 == size(a,2)) %if true it finds toepltiz matrix
    for i = 1:m
        for j = 1:n
            A(i,j) = a(n+i-j);
        end
    end
else
    fprintf("Dimensions mismatch")
    A = [];  % returns an empty matrix
end
```

```
%(a)
m=4; n=2; a=1:5
```

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

```
A=toeplitze(m,n,a)
```

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

```
%(b)
m=4; n=3; a=1:5
```

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

```
A=toeplitze(m,n,a)
```

```
Dimensions mismatch
A =

     []
```

```
%(c)
m=4; n=3; a=1:7
```

```
a = 1×7
     1     2     3     4     5     6     7
```

```
A=toeplitze(m,n,a)
```

```
Dimensions mismatch
A =
```

```
[]
```

```
%(d)
m=3; n=4; a=randi(10,1,6)
```

```
a = 1×6
     2    10     9     9     3     6
```

```
A=toeplitze(m,n,a)
```

```
A = 3×4
     9     9    10     2
     3     9     9    10
     6     3     9     9
```

```
%(e)
m=4; n=4; a=[zeros(1,3), 1:4]
```

```
a = 1×7
     0     0     0     1     2     3     4
```

```
A=toeplitze(m,n,a)
```

```
A = 4×4
     1     0     0     0
     2     1     0     0
     3     2     1     0
     4     3     2     1
```

```
%(1)
m=6; n=6;
a=zeros(1,11);
for i=1:6
    randomNum = randi(100);
    a(i) = randomNum;
end
a
```

```
a = 1×11
     3    43    32    17    18    43     0     0     0     0     0
```

```
A=toeplitze(m,n,a)
```

```
A = 6×6
    43    18    17    32    43     3
     0    43    18    17    32    43
     0     0    43    18    17    32
     0     0     0    43    18    17
     0     0     0     0    43    18
     0     0     0     0     0    43
```

```
%(2)
m=5;n=5;a=zeros(1,9);a(5)=1;a
```

```
a = 1×9
     0     0     0     0     1     0     0     0     0
```

```
A=toeplitze(m,n,a)
```

```
A = 5×5
     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
```

Part 2

```
r=1:5;
T=toeplitz(r)
```

```
T = 5×5
     1     2     3     4     5
     2     1     2     3     4
     3     2     1     2     3
     4     3     2     1     2
     5     4     3     2     1
```

```
if isequal(T, transpose(T))
    fprintf("T is a symmetric matrix")
end
```

```
T is a symmetric matrix
```

```
c=[1,2,3,4,5,6];
T=toeplitz(c,r)
```

```
T = 6×5
     1     2     3     4     5
     2     1     2     3     4
     3     2     1     2     3
     4     3     2     1     2
     5     4     3     2     1
     6     5     4     3     2
```

Exercise 5

Theory: A vector with nonnegative entries is called a probability vector if the sum of its entries is 1. A square matrix is called right stochastic if its rows are probability vectors; a square matrix is called left stochastic if its columns are probability vectors; and a square matrix is called doubly stochastic if both, the rows and the columns, are probability vectors.

The function "stochastic()" takes a square, non-negative matrix and determines if it is a left, right, or doubly stochastic matrix. If it isn't, it determines if it can be scaled to a left and a right stochastic matrices, calculates and displays them.

```
type stochastic.m
```

```
function [S1,S2,L,R] = stochastic(A)
L = [];
R = [];

[numR, numC] = size(A);
if numR ~= numC %checks if matrix A is square
    fprintf('The matrix is not square. Please try again\n')
    return;
end
areNeg = any(A(:) < 0);
if areNeg %checks if there are any negative entries in A
    fprintf('The matrix has negative entries. Please try again\n')
    return;
end

fprintf('the vector of sums down each column is\n')
S1 = sum(A,1) %sums the entries of each column and creates a vector
fprintf('the vector of sums across each row is\n')
S2 = sum(A,2)  %sums the entries of each row and creates a vector

column0 = any(S1 == 0); %checks if there are any zero columns
row0 = any(S2 == 0); %checks if there are any zero rows

checkLS = all(S1 == 1); %checks if the sum of each column is equal to 1
checkRS = all(S2 == 1); %checks if the sum of each column is equal to 1

recS1 = 1./S1; %creates the reciprocal of S1
recS2 = 1./S2; %creates the reciprocal of S2

%if there is a zero row and a zero column, indicates it isn't stochastic
%and can't be scaled
if column0 && row0
    fprintf('A is neither left nor right stochastic and cannot be scaled to either of them\n');
elseif checkLS && checkRS %checks if A is both left and right stochastic (doubly stochastic)
    L = A
    R = A
    fprintf('A is doubly stochastic\n');
elseif checkLS %checks if A is left stochastic
    L = A
    fprintf('A is only left stochastic\n');
elseif checkRS %checks if A is right stochastic
    R = A
    fprintf('A is only right stochastic\n');
else
    fprintf('A is neither left stochastic nor right stochastic but can be scaled to a stochastic matrix\n\n');
    %if there are no zero rows nor zero columns, creates both a left and right stochastic matrix
    if ~column0 && ~row0
        L = bsxfun(@times,A,recS1) %creates the left stochastic matrix
        R = A.*recS2 %creates the right stochastic matrix
```

```
        diffLR = abs(L-R);
        %indicates it is a doubly stochastic matrix if L and R are equal
        if closetozeroroundoff(diffLR, 7) == zeros(size(A))
            fprintf('After scaling the matrix A, both the left and right stochastic matrices were equal, so the resu
        else %if they are not equal, displays both the left and right stochastic matrices
            fprintf('After scaling the matrix A, the resultant left stochastic matrix was L\n\n');
            display(L);
            fprintf('And the resultant right stochastic matrix was R\n\n')
            display(R);
        end
    elseif ~column0 && row0 %if there were any zero rows but no zero columns, creates a left stochastic matrix
        fprintf('After scaling the matrix A, the resultant left stochastic matrix was:\n');
        L = bsxfun(@times,A,recS1)
    elseif column0 && ~row0 %if there were any zero columns but no zero rows, creates a right stochastic matrix
         fprintf('And the resultant right stochastic matrix was:\n')
        R = A.*recS2
    end
end
end
```

```
type closetozeroroundoff.m
```

```
function B = closetozeroroundoff(A,p)
A(abs(A)<10^-p)=0;
B=A;
end
```

```
type jord.m
```

```
function J=jord(n,r)
    J = [];
 if (mod(n, 1) ~= 0 || n <= 1)
  disp('Jordan Block cannot be built')
  return
    end
 J = diag(diag(eye(n - 1)), 1) + r * eye(n);
end
```

These functions will be the ones used in this exercise.

Now, the stochastic() function will be applied to the following matrices

```
%(a)
A=[0.5,0,0.5,0; 0,0,1,0;0.5,0,0.5,0;0,0,0,1]
```

```
A = 4×4
    0.5000         0    0.5000         0
         0         0    1.0000         0
    0.5000         0    0.5000         0
         0         0         0    1.0000
```

```
stochastic(A)
```

```
the vector of sums down each column is
S1 = 1×4
     1     0     2     1
the vector of sums across each row is
S2 = 4×1
     1
     1
     1
     1
```

```
R = 4×4
    0.5000         0    0.5000         0
         0         0    1.0000         0
    0.5000         0    0.5000         0
         0         0         0    1.0000
A is only right stochastic
ans = 1×4
     1     0     2     1
```

```
%(b)
A=transpose(A)
```

```
A = 4×4
    0.5000         0    0.5000         0
         0         0         0         0
    0.5000    1.0000    0.5000         0
         0         0         0    1.0000
```

```
stochastic(A)
```

```
the vector of sums down each column is
S1 = 1×4
     1     1     1     1
the vector of sums across each row is
S2 = 4×1
     1
     0
     2
     1
L = 4×4
    0.5000         0    0.5000         0
         0         0         0         0
    0.5000    1.0000    0.5000         0
         0         0         0    1.0000
A is only left stochastic
ans = 1×4
     1     1     1     1
```

We could see in part (a) that the matrix was only right stochastic because each row was a probability vector. In part (b), the matrix was transposed, and now that matrix was only left stochastic.

```
%(c)
A=[0.5,  0,  0.5; 0,  0,  1; 0,  0,  0.5]
```

```
A = 3×3
    0.5000         0    0.5000
         0         0    1.0000
         0         0    0.5000
```

```
stochastic(A)
```

```
the vector of sums down each column is
S1 = 1×3
    0.5000         0    2.0000
the vector of sums across each row is
S2 = 3×1
    1.0000
    1.0000
    0.5000
A is neither left stochastic nor right stochastic but can be scaled to a stochastic matrix
```

```
And the resultant right stochastic matrix was:
R = 3×3
    0.5000         0    0.5000
         0         0    1.0000
         0         0    1.0000
ans = 1×3
    0.5000         0    2.0000
```

```
%(d)
A=transpose(A)
```

```
A = 3×3
    0.5000         0         0
         0         0         0
    0.5000    1.0000    0.5000
```

```
stochastic(A)
```

```
the vector of sums down each column is
S1 = 1×3
    1.0000    1.0000    0.5000
the vector of sums across each row is
S2 = 3×1
    0.5000
         0
    2.0000
A is neither left stochastic nor right stochastic but can be scaled to a stochastic matrix

After scaling the matrix A, the resultant left stochastic matrix was:
L = 3×3
    0.5000         0         0
         0         0         0
    0.5000    1.0000    1.0000
ans = 1×3
    1.0000    1.0000    0.5000
```

In part (c), the matrix given wasn't initially stochastic. However, since it only had a zero column, it could be scaled down to a right stochastic matrix. In part (d), similar to part (b), the previous matrix was transposed, and in this case, it wasn't stochastic until it was scaled, and this new matrix was left stochastic.

Taking into consideration parts (b) and (d), it can be concluded that if an stochastic matrix is transposed, it would change its type from left stochastic to right stochastic, and viceversa. This is because the rows become the columns and viceversa, changing the orientation of the probability vectors.

```
%(e)
A=[0.5,  0,  0.5; 0,  0.5,  0.5; 0.5,  0.5,  0]
```

```
A = 3×3
    0.5000         0    0.5000
         0    0.5000    0.5000
    0.5000    0.5000         0
```

```
stochastic(A)
```

```
the vector of sums down each column is
S1 = 1×3
     1     1     1
the vector of sums across each row is
S2 = 3×1
```

```
     1
     1
     1
L = 3×3
    0.5000         0    0.5000
         0    0.5000    0.5000
    0.5000    0.5000         0
R = 3×3
    0.5000         0    0.5000
         0    0.5000    0.5000
    0.5000    0.5000         0
A is doubly stochastic
ans = 1×3
     1     1     1
```

In part (e), the given matrix was doubly stochastic. In other words, each of its rows and each of its columns are probability vectors (add up to 1).

```
%(f)
A=magic(3)
```

```
A = 3×3
     8     1     6
     3     5     7
     4     9     2
```

```
stochastic(A)
```

```
the vector of sums down each column is
S1 = 1×3
    15    15    15
the vector of sums across each row is
S2 = 3×1
    15
    15
    15
A is neither left stochastic nor right stochastic but can be scaled to a stochastic matrix
L = 3×3
    0.5333    0.0667    0.4000
    0.2000    0.3333    0.4667
    0.2667    0.6000    0.1333
R = 3×3
    0.5333    0.0667    0.4000
    0.2000    0.3333    0.4667
    0.2667    0.6000    0.1333
After scaling the matrix A, both the left and right stochastic matrices were equal, so the resultant was the doubly
ans = 1×3
    15    15    15
```

In part (f), the magic matrix crated by Matlab has equal row and column sums. While it isn't a stochastic matrix, since the sum is equal in all rows and columns, it can be scaled to a doubly stochastic matrix, as it happened above.

```
%(g)
B=[1 2;3 4;5 6];
A=B*B'
```

```
A = 3×3
```

```
     5    11    17
    11    25    39
    17    39    61
```

```
stochastic(A)
```

```
the vector of sums down each column is
S1 = 1×3
    33    75   117
the vector of sums across each row is
S2 = 3×1
    33
    75
   117
A is neither left stochastic nor right stochastic but can be scaled to a stochastic matrix
L = 3×3
    0.1515    0.1467    0.1453
    0.3333    0.3333    0.3333
    0.5152    0.5200    0.5214
R = 3×3
    0.1515    0.3333    0.5152
    0.1467    0.3333    0.5200
    0.1453    0.3333    0.5214
After scaling the matrix A, the resultant left stochastic matrix was L
L = 3×3
    0.1515    0.1467    0.1453
    0.3333    0.3333    0.3333
    0.5152    0.5200    0.5214
And the resultant right stochastic matrix was R
R = 3×3
    0.1515    0.3333    0.5152
    0.1467    0.3333    0.5200
    0.1453    0.3333    0.5214
ans = 1×3
    33    75   117
```

In part (g), the matrix A was created by multiplying matrix B by its reciprocal. This matrix has the peculiarity that its rows are equal to the corresponding column. It isn't an stochastic matrix, but it can be scaled to a left or a right stochastic matrix. The rows in the left stochastic matrix are equal to the columns in the right stochastic.

```
%(h)
A=jord(5,4)
```

```
A = 5×5
     4     1     0     0     0
     0     4     1     0     0
     0     0     4     1     0
     0     0     0     4     1
     0     0     0     0     4
```

```
stochastic(A)
```

```
the vector of sums down each column is
S1 = 1×5
     4     5     5     5     5
the vector of sums across each row is
S2 = 5×1
     5
     5
     5
     5
```

```
     4
A is neither left stochastic nor right stochastic but can be scaled to a stochastic matrix
L = 5×5
    1.0000    0.2000         0         0         0
         0    0.8000    0.2000         0         0
         0         0    0.8000    0.2000         0
         0         0         0    0.8000    0.2000
         0         0         0         0    0.8000
R = 5×5
    0.8000    0.2000         0         0         0
         0    0.8000    0.2000         0         0
         0         0    0.8000    0.2000         0
         0         0         0    0.8000    0.2000
         0         0         0         0    1.0000
After scaling the matrix A, the resultant left stochastic matrix was L
L = 5×5
    1.0000    0.2000         0         0         0
         0    0.8000    0.2000         0         0
         0         0    0.8000    0.2000         0
         0         0         0    0.8000    0.2000
         0         0         0         0    0.8000
And the resultant right stochastic matrix was R
R = 5×5
    0.8000    0.2000         0         0         0
         0    0.8000    0.2000         0         0
         0         0    0.8000    0.2000         0
         0         0         0    0.8000    0.2000
         0         0         0         0    1.0000
ans = 1×5
     4     5     5     5     5
```

In part (h), the Jordan matrix generated is not initially stochastic. However, it can be scaled to both a left and a right stochastic matrix.

```
%(k)
A=randi(10,5,5)
```

```
A = 5×5
     1     2     2     7     8
     3    10     5     1     8
     6    10    10     9     4
    10     5     8    10     7
    10     9    10     7     2
```

```
A(:,1)=0
```

```
A = 5×5
     0     2     2     7     8
     0    10     5     1     8
     0    10    10     9     4
     0     5     8    10     7
     0     9    10     7     2
```

```
A(1,:)=0
```

```
A = 5×5
     0     0     0     0     0
     0    10     5     1     8
     0    10    10     9     4
     0     5     8    10     7
     0     9    10     7     2
```

```
stochastic(A)
```

```
the vector of sums down each column is
S1 = 1×5
     0    34    33    27    21
the vector of sums across each row is
S2 = 5×1
     0
    24
    33
    30
    28
A is neither left nor right stochastic and cannot be scaled to either of them
ans = 1×5
     0    34    33    27    21
```

Finally, for part (k), a random matrix was generated and its first row and first column were replaced with zeros. Since the matrix has a zero row and a zero column, it isn't a stochastic matrix and it cannot be scaled down to one.

Exercise 6

```
format
format compact
syms x
F = @(x) atan(x) + x - 1
```

```
F = function_handle with value:

    @(x)atan(x)+x-1
```

```
F1 = eval(['@(x)' char(diff(F(x)))])
```

```
F1 = function_handle with value:

    @(x)1/(x^2+1)+1
```

```
G=@(x) x.^3-x-1
```

```
G = function_handle with value:

    @(x)x.^3-x-1
```

```
G1=eval(['@(x)' char(diff(G(x)))])
```

```
G1 = function_handle with value:

    @(x)3*x^2-1
```

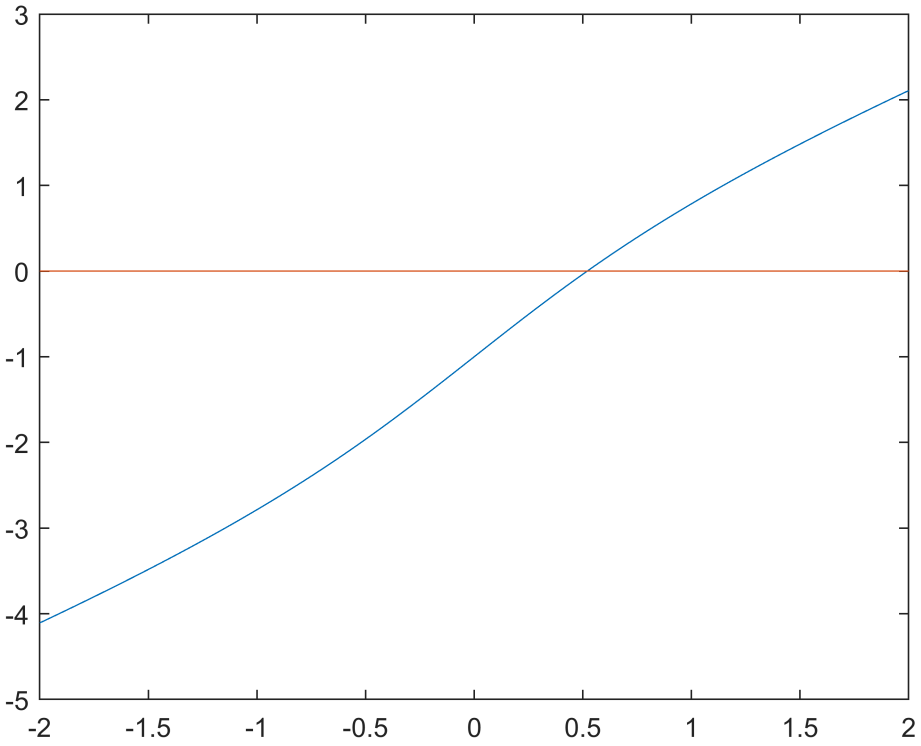
```

yzero=@(x) 0.*x.^(0)
```

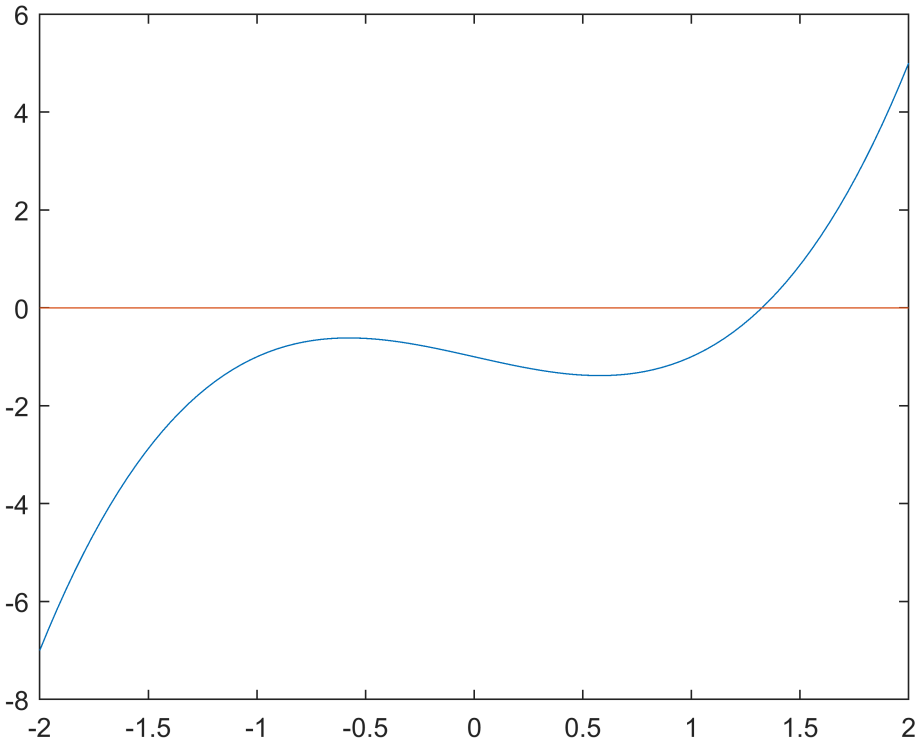
```
yzero = function_handle with value:

    @(x)0.*x.^(0)
```

```
x=linspace(-2,2);
plot(x,F(x),x,yzero(x));
```




```
plot(x,G(x),x,yzero(x));
```



```
syms x
p=x^3-x-1;
roots(sym2poly(p))
```

```
ans = 3×1 complex
   1.3247 + 0.0000i
  -0.6624 + 0.5623i
  -0.6624 - 0.5623i
```

```
type newtons
```

```
function root = newtons(fun,dfun,x0)
format long
x = fzero(fun,x0);
fprintf('MATLAB approximation of the real zero of the function: %.12f\n', x)

N = 0;
xN = x0;

while (abs(xN - x)) >= (10^(-12))
    xN = xN - (fun(xN) / dfun(xN));
    N = N + 1;
end
root = xN;
fprintf('Number of iterations: %d\n', N)
end
```

Part (a)

```
fun = F;
dfun = F1;

root = newtons(fun,dfun,-1.5)
```

```
MATLAB approximation of the real zero of the function: 0.520268992720
Number of iterations: 5
root =
   0.520268992719585
```

```
root = newtons(fun,dfun,0)
```

```
MATLAB approximation of the real zero of the function: 0.520268992720
Number of iterations: 4
root =
   0.520268992719590
```

```
root = newtons(fun,dfun,1.5)
```

```
MATLAB approximation of the real zero of the function: 0.520268992720
Number of iterations: 5
root =
   0.520268992719590
```

Part (b)

```
fun = G;
dfun = G1;
```

(1)

```
root = newtons(fun,dfun,1.3)
```

```
MATLAB approximation of the real zero of the function: 1.324717957245
Number of iterations: 3
root =
   1.324717957244843
```

(2)

```
root = newtons(fun,dfun,1)
```

```
MATLAB approximation of the real zero of the function: 1.324717957245
Number of iterations: 5
root =
   1.324717957244790
```

(3)

```
root = newtons(fun,dfun,0.6)
```

```
MATLAB approximation of the real zero of the function: 1.324717957245
Number of iterations: 12
root =
   1.324717957244747
```

(4)

```
root = newtons(fun,dfun,0.577351)
```

```
MATLAB approximation of the real zero of the function: 1.324717957245
Number of iterations: 38
root =
   1.324717957244746
```

(5)

```
x0 = 1/sqrt(3)
```

```
x0 =
   0.577350269189626
```

```
root = newtons(fun,dfun,x0)
```

```
MATLAB approximation of the real zero of the function: 1.324717957245
Number of iterations: 95
root =
   1.324717957244746
```

(6)

```
root = newtons(fun,dfun,0.577)
```

```
MATLAB approximation of the real zero of the function: 1.324717957245
Number of iterations: 100
root =
   1.324717957244807
```

(7)

```
root = newtons(fun,dfun,0.4)
```

```
MATLAB approximation of the real zero of the function: 1.324717957245
Number of iterations: 13
root =
   1.324717957244746
```

(8)

```
root = newtons(fun,dfun,0.1)
```

```
MATLAB approximation of the real zero of the function: 1.324717957245
Number of iterations: 34
root =
   1.324717957244746
```

BONUS 1)

"The further away the put the x0 from the solution the more iterations the function needs to perform before it reaches a close neighborhood of the solution (excluding choices (4)-(6).)"

BONUS 2)

"If we choose x0 for which the value of the derivative is close to 0 the function will need to iterate significantly more times to reach a close neighborhood of the solution"