MATLAB PROJECT 1

GROUP # 27

_1.__ Jake Sanchez

_2. Nicolas Santiago

_3.__Brandon Miguel

_4. David Rowe

_5.__Charles Richardson

_6. Nic Morita

Exercise #1

```
format
format compact
type jord
```

```
function J=jord(n,r)
if n>1 && ceil(n)==n
    onesVector=r*ones(n,1);
    J=diag(onesVector);
    for i=[1:n-1]
        J(i,i+1)=1;
    end
else
    J=[];
    disp("Jordan Block cannot be built")
end
end
```

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

```
r = 0.0975
```

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

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

```
     []
```

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

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

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

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

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

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

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

```
ans = 4×4
    0.0975    1.0000         0         0
         0    0.0975    1.0000         0
         0         0    0.0975    1.0000
         0         0         0    0.0975
```

Exercise #2

```
type added
```

```
function C = added(A,B)
m=size(A); n=size(B);
if m==n

else
    disp('the matrices are not of the same size and cannot be added')
    C = [];
    return
end
C = zeros(size(A));
[m,n] = size(A);
for i=1:m
    for j=1:n
        C(i,j) = A(i,j) + B(i,j);
    end
end
if C==A+B

else
    disp('check the code!')
end
```

```
%(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 =
     []
```

```
%(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 =
     []
```

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

```
A = 3×4
    28    97    96    15
    55    16    49    43
    96    98    81    92
B = 3×4
    80     4    68    40
    96    85    76    66
    66    94    75    18
```

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

```
ans = 3×4
   108   101   164    55
   151   101   125   109
   162   192   156   110
```

```
%(1)
if added(A,B)==added(B,A)
    disp('commutative property holds for the given A and B')
end
```

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

```
%(2)
k=fix(10*rand(1))+5
```

```
k = 12
```

```
if k*added(A,B)==added(k*A,k*B)
    disp('distributive property holds for the given A and B')
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)
    %how the output is supposed to look like
    G = eye(n);
    %assigning the values where they need to go G(i,i) = cos(theta);
    G(i,j) = -sin(theta);
    G(j,i) = sin(theta);
    G(j,j) = cos(theta);
else%if one of the conditions does not hold true
    G = [];%it should output empty matrix
    disp('Givens rotation matrix cannot be constructed')%output statement
end
```

```
%(1)
n=1; i=1; j=2; theta=pi;
G = givensrot(n,i,j,theta)
```

```
Givens rotation matrix cannot be constructed
G =
     []
```

```
%(2)
n=4; i=3; j=2; theta=pi/2;
G = givensrot(n,i,j,theta)
```

```
Givens rotation matrix cannot be constructed
G =
     []
```

```
%(3)
n=5; i=2; j=4; theta=pi/4;
G = givensrot(n,i,j,theta)
```

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

```
%(4)
n=2; i=1; j=2; theta=-pi/2;
G = givensrot(n,i,j,theta)
```

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

```
%(5)
n=3; i=1; j=2; theta=pi;
G = givensrot(n,i,j,theta)
```

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

```
%predicted GI1
I=eye(3)
```

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

```
GI = [-1 0 0;0 -1 0; 0 0 1]
```

```
GI = 3×3
    -1     0     0
     0    -1     0
     0     0     1
```

```
%Now multiplying by each column
G.*I(:,1)
```

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

```
G.*I(:,2)
```

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

```
G.*I(:,3)
```

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

```
%actual calculation
G*I
```

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

```
%Typed and displayed GI in Live Script
%now comparing the two predicted and actual matrices
if(closetozeroroundoff(GI,7)==closetozeroroundoff(G.*I,7)) disp('My prediction was correct')
else
    disp('My prediction was incorrect')
end
```

```
My prediction was incorrect
```

```
%new vector that would add on to the transformation under matrix G
x = ones(3,1)
```

```
x = 3×1
     1
     1
     1
```

```
G.*x
```

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

Exercise #4

```
type toeplitze
```

```
function A=toeplitze(m,n,a)
if length(a)~=(m+n-1)
    A=[];
    disp('Dimensions mismatch!!')
else
    A=zeros(m,n);
    for i=1:m
        for j=1:n
        A(i,j)=a(n+i-j);
        end
    end
end
end
```

```
%a
toeplitze(4,2,1:5)
```

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

```
%b
toeplitze(4,3,1:5)
```

```
Dimensions mismatch!!
ans =
     []
```

```
%c
toeplitze(4,3,1:7)
```

```
Dimensions mismatch!!
ans =
     []
```

```
%d
```

```
toeplitze(3,4,randi(10,1,6))
```

```
ans = 3×4
     1     1     3     1
     9     1     1     3
     7     9     1     1
```

```
%e
toeplitze(4,4,[zeros(1,3),1:4])
```

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

```
%1
b=randi([0, 100],[1,9])
```

```
b = 1×9
    32    95     3    44    38    77    80    18    49
```

```
a=triu(b)
```

```
a = 1×9
    32    95     3    44    38    77    80    18    49
```

```
A = toeplitze(5,5,a)
```

```
A = 5×5
    38    44     3    95    32
    77    38    44     3    95
    80    77    38    44     3
    18    80    77    38    44
    49    18    80    77    38
```

```
%2
a=eye(5,5)
```

```
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
```

```
A=toeplitze(5,5,a)
```

```
Dimensions mismatch!!
A =
     []
```

(Part 2)

```
%a
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 issymmetric(T)
disp('T is symmetric matrix')
end
```

```
T is symmetric matrix
```

```
%b
T=toeplitz([1 2 3 4 5 6],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

```
type stochastic
```

```
%Accepts a square matrix with nonnegative entries as input
%Output L and R are left stochastic and right stochastic matrices
%genetated, when possible
function [S1,S2,L,R]=stochastic(A)
L=[];
R=[];

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

%Check whether A has a zero col.
colZ = false;
for j = 1:size(A:1)
    if (sum(A(:,j)) == 0)
        colZ = true;
        break;
    end
end

%Check whether A has a zero row.
rowZ = false;
for i = 1:size(A,2)
    if (sum(A(i,:)) == 0)
        rowZ = true;
        break;
    end
end


%Return function if zero rows AND cols present
if (colZ && rowZ)
```

```
    disp('A is neither left nor right stochastic and cannot be scaled to either of them')
    return
end

%Check if matrix is a left stochastic
stoC = true;
for i = 1:size(A,1)
    if (sum(A(:,i)) ~= 1)
        stoC = false;
        break;
    end
end

%Check if matrix is a right stochastic
stoR = true;
for i = 1:size(A,2)
    if (sum(A(i,:)) ~= 1)
        stoR = false;
        break;
    end
end

%Displays cases
if (stoC && stoR)
    disp('Matrix is a doubly stochastic')
    L = A
    R = A
    return
elseif (stoC)
    disp('Matrix is a left stochastic')
    L = A
    return
elseif (stoR)
    disp('Matrix is a right stochastic')
    R = A
    return
else
    disp('A is neither left nor right stochastic but can be scaled to a stochastic matrix')
end

%Verifying there are no 0 entires in the column summ
goodC = true;
for i = 1:size(S1)
    if(ismember(S1(i), 0))
        goodC = false;
        break;
    end
end

%Verifying there are no 0 entires in the row sum
goodR = true;
for j = 1:size(S2)
    if(ismember(S2(j), 0))
        goodR = false;
        break;
    end
end

%Scaling matrix with non 0 entries.
if (goodC && goodR)
    L = A .* (1 ./ S1);
    R = A .* (1 ./ S2);
    %Check if L and R are equal
    if (isequal(closetozeroroundoff(L, 7),closetozeroroundoff(R, 7)))
        disp('A has been scaled to a doubly stochastic matrix')
```

```
        disp(L)
        return
    end
elseif (goodC)
    L = A .* (1 ./ S1);
    disp('A has been scaled to a left stochastic matrix')
    disp(L)
    return
elseif (goodR)
    R = A .* (1 ./ S2);
    disp('A has been scaled to a right stochastic matrix')
    disp(R)
    return
end
end
```

```
%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
Matrix is a right stochastic
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
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
Matrix is a left stochastic
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
ans = 1×4
     1     1     1     1
```

```
%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 nor right stochastic but can be scaled to a stochastic matrix
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 nor right stochastic but can be scaled to a stochastic matrix
A has been scaled to a left stochastic matrix
    0.5000         0         0
         0         0         0
    0.5000    1.0000    1.0000
ans = 1×3
    1.0000    1.0000    0.5000
```

```
%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
Matrix is a doubly stochastic
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
ans = 1×3
     1     1     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 nor right stochastic but can be scaled to a stochastic matrix
A has been scaled to a doubly stochastic matrix
    0.5333    0.0667    0.4000
    0.2000    0.3333    0.4667
    0.2667    0.6000    0.1333
ans = 1×3
    15    15    15
```

```
%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 nor right stochastic but can be scaled to a stochastic matrix
ans = 1×3
    33    75   117
```

```
%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 nor right stochastic but can be scaled to a stochastic matrix
ans = 1×5
     4     5     5     5     5
```

```
%k
A=randi(10,5,5);A(:,1)=0;A(1,:)=0
```

```
A = 5×5
     0     0     0     0     0
     0     1     9    10     9
     0     1     8     7     1
     0     6     2     9     2
     0     1     7     5     2
```

```
stochastic(A)
```

```
the vector of sums down each column is
S1 = 1×5
     0     9    26    31    14
the vector of sums across each row is
S2 = 5×1
     0
    29
    17
    19
    15
A is neither left nor right stochastic and cannot be scaled to either of them
ans = 1×5
     0     9    26    31    14
```

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
```

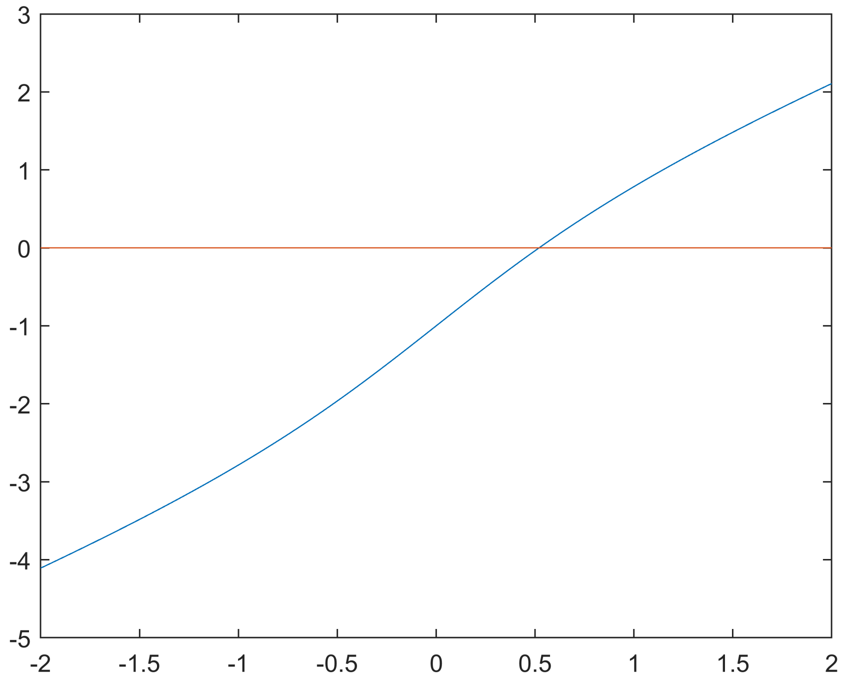
```
G1=eval(['@(x)' char(diff(G(x)))])
```

```
G1 = function_handle with value:

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

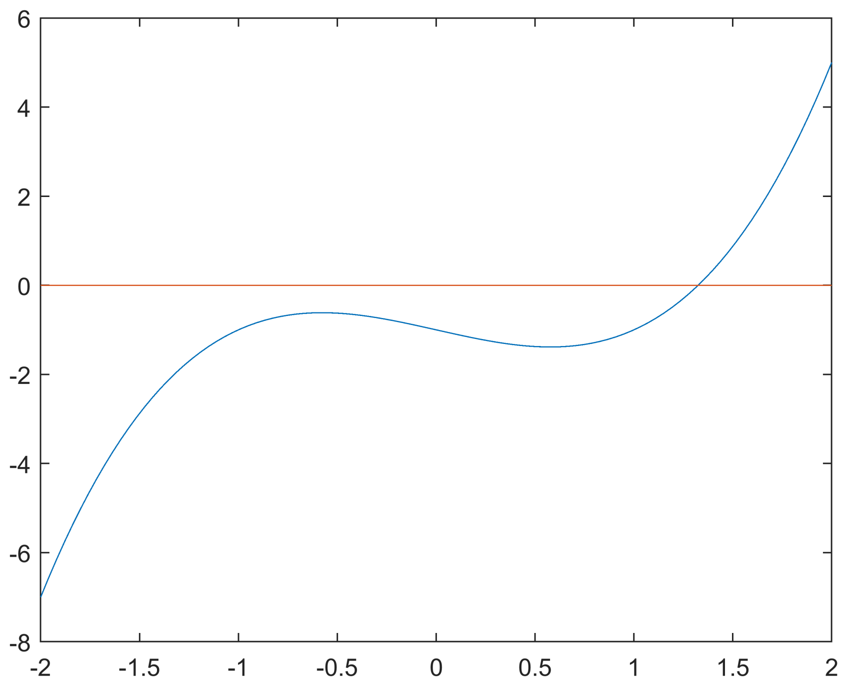
```
yzero=@(x) 0.*x.^(0)
```

```
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('x is the MATLAB approximation of the real zero\n')
n = 0;
xn = x0;
while(abs(xn-x)>=10^(-12))
    xn = xn - fun(xn)/dfun(xn);
    n = n+1;
end
fprintf('\nnumber of iterations = %i \n', n)
root = xn;
end
```

Part (a)

```
fun=F;
dfun=F1;
root=newtons(fun, dfun, 0.5)
```

```
x =
   0.520268992719590
x is the MATLAB approximation of the real zero

number of iterations = 3
root =
   0.520268992719590
```

```
root=newtons(fun, dfun, 0.55)
```

```
x =
   0.520268992719590
x is the MATLAB approximation of the real zero

number of iterations = 3
root =
   0.520268992719590
```

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

```
x =
   0.520268992719590
x is the MATLAB approximation of the real zero

number of iterations = 3
root =
   0.520268992719579
```

Part (b)

```
fun=G;
dfun=G1;
%(1)
root=newtons(fun, dfun, 1.3)
```

```
x =
```

```
   1.324717957244746
x is the MATLAB approximation of the real zero

number of iterations = 3
root =
   1.324717957244843
```

```
%(2)
root=newtons(fun, dfun, 1)
```

```
x =
   1.324717957244746
x is the MATLAB approximation of the real zero

number of iterations = 5
root =
   1.324717957244790
```

```
%(3)
root=newtons(fun, dfun, 0.6)
```

```
x =
   1.324717957244746
x is the MATLAB approximation of the real zero

number of iterations = 12
root =
   1.324717957244747
```

```
%(4)
root=newtons(fun, dfun, 0.577351)
```

```
x =
   1.324717957244746
x is the MATLAB approximation of the real zero

number of iterations = 38
root =
   1.324717957244746
```

```
%(5)
x0=1/sqrt(3)
```

```
x0 =
   0.577350269189626
```

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

```
x =
   1.324717957244746
x is the MATLAB approximation of the real zero

number of iterations = 95
root =
   1.324717957244746
```

```
%(6)
root=newtons(fun, dfun, 0.577)
```

```
x =
   1.324717957244746
x is the MATLAB approximation of the real zero
```

```
number of iterations = 100
root =
   1.324717957244807
```

```
%(7)
root=newtons(fun, dfun, 0.4)
```

```
x =
   1.324717957244746
x is the MATLAB approximation of the real zero

number of iterations = 13
root =
   1.324717957244746
```

```
%(8)
root=newtons(fun, dfun, 0.1)
```

```
x =
   1.324717957244746
x is the MATLAB approximation of the real zero

number of iterations = 34
root =
   1.324717957244746
```

BONUS POINTS RESPONSE

The general pattern is that the number of iterations increases as the intitial approximation gets farther from thereal zero. However, choices (4)-(6) don't follow this pattern. This is because the intitial approximations for (4)-(6)are close to the zero of G1 (the derivative of G). Choice (5) is the actual 0 of G1, and therefore shouldn't evenwork since it would lead to dividing by 0, but MATLAB rounds the value of 1/sqrt(3).