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 # _25________

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

_1. Allie Saari____

_2. Marc Miller____

_3. Nitin Ramados____

4.Maureen Riepe ____

_5. Daniel Louis____

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

Exercise 1

```
type jord
```

```
function J=jord(n,r)
if n<1 || n~=fix(n)
 disp('Jordan Block cannot be built')
 J= [];
else
    z=zeros(1,n);

   for i=1:n
       z(1,i)=r;

   end
  J= diag(z)+ diag(ones(1,n-1),1);
end
```

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

```
r = 0.6407
```

```
jord(0,r)
```

```
Jordan Block cannot be built

ans =

     []
```

```
jord(-2,r)
```

```
Jordan Block cannot be built

ans =

     []
```

```
jord(3.5,r)
```

```
Jordan Block cannot be built

ans =

     []
```

```
jord(-2.5,r)
```

```
Jordan Block cannot be built

ans =

     []
```

```
jord(4,r)
```

```
ans = 4×4
    0.6407    1.0000         0         0
         0    0.6407    1.0000         0
         0         0    0.6407    1.0000
         0         0         0    0.6407
```

Exercise 2

```
type added.m
```

```
function [C] = added(A,B)
C=[];
k=fix(10*rand(1))+5;
if isequal(size(A),size(B))
    %compared dimensions of A and B
    n = size (A,2);
    i=1:n; %accounted for all posible dimensions
    C(:,i)= A(:,i)+ B(:,i);
    disp (C);
    if C ~= A + B %verified that code executes appropriate command
    disp ('check the code')
    end
    if  (A(:,i)+ B(:,i))== B(:,i)+ A(:,i)
        disp ('commutative property holds for the given A and B')
    end

    if k*(A(:,i)+ B(:,i))== k*B(:,i)+ k*A(:,i)
        disp ('distributive property holds for the given A and B')
    end
else disp('the matrices are not of the same size and cannot be added')
    C=[];


end
```

```
added(magic(3),ones(4))
```

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

ans =

     []
```

```
added(ones(3,4),ones(3,3))
```

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

ans =

     []
```

```
added(randi(100,3,4),randi(100,3,4))
```

```
   101   164    55   151
   101   125   109   100
   192   156   110    94

commutative property holds for the given A and B
distributive property holds for the given A and B
ans = 3×4
   101   164    55   151
   101   125   109   100
   192   156   110    94
```

Exercise 3

```
type givensrot
```

```
function G=givensrot(n,i,j,theta)
if 1<=i && i<j && j<=n && n>=2
  G=eye(n);
  G(i,i)=cos(theta);
  G(i,j)=-sin(theta);
  G(j,i)=sin(theta);
  G(j,j)=cos(theta);
  return
else
    G=[];
    disp('Givens rotation matrix cannot be constructed')
    return

end
```

(1)

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

```
Givens rotation matrix cannot be constructed

G =

     []
```

(2)

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

```
Givens rotation matrix cannot be constructed

G =

     []
```

(3)

```
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
```

(4)

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

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

(5)

```
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);
Prediction=[-1,0,0;0,-1,0;0,0,1];
G*I
```

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

```
if closetozeroroundoff(Prediction-G*I,7)==0
    disp('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);
G*x
```

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

Exercise 4

Part 1

```
type toeplitze
```

```
function A=toeplitze(m,n,a)
A=zeros(m,n);
lengthA = length(a);
if lengthA==(m+n-1)
    for i=1:m
        for j=1:n
            A(i,j)=a(n+i-j);
        end
    end
else
    fprintf('There is a dimension mismatch.')
end
end
```

(a)

```
m=4; n=2; a=1:5
```

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

(b)

```
m=4; n=3; a=1:5
```

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

c)

```
m=4; n=3; a=1:7
```

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

(d)

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

```
a = 1×6
     9     2     4    10    10     8
```

(e)

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

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

(1)

```
a=triu(randi(100,6,6))
```

```
a = 6×6
    62    84    98    42    74    24
     0    40    55    50    96    72
     0     0    34    70     4    63
     0     0     0    98    36    60
     0     0     0     0    67    67
     0     0     0     0     0     5
```

(2)

```
a=eye(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
```

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
```

(b)

```
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
```

Excercise 5

```
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
```

```
[S1,S2,L,R]=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
A is right stochastic
    0.5000         0    0.5000         0
         0         0    1.0000         0
    0.5000         0    0.5000         0
         0         0         0    1.0000
```

```
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
```

```
[S1,S2,L,R]=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
A is left stochastic
    0.5000         0    0.5000         0
         0         0         0         0
    0.5000    1.0000    0.5000         0
         0         0         0    1.0000
```

```
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
```

```
[S1,S2,L,R]=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
A is scaled to right stochastic
    0.5000         0    0.5000
         0         0    1.0000
         0         0    1.0000
```

```
A=transpose(A)
```

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

```
[S1,S2,L,R]=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 is scaled to left stochastic
    0.5000         0         0
         0         0         0
    0.5000    1.0000    1.0000
```

```
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
```

```
[S1,S2,L,R]=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
A is doubly stochastic
    0.5000         0    0.5000
         0    0.5000    0.5000
    0.5000    0.5000         0
```

```
A=magic(3)
```

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

```
[S1,S2,L,R]=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 is scaled to doubly stochastic
    0.5333    0.0667    0.4000
    0.2000    0.3333    0.4667
    0.2667    0.6000    0.1333
```

```
B=[1 2;3 4;5 6]; A=B*B'
```

```
A = 3×3
     5    11    17
    11    25    39
    17    39    61
```

```
[S1,S2,L,R]=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
A is scaled to left stochastic
    0.1515    0.1467    0.1453
    0.3333    0.3333    0.3333
    0.5152    0.5200    0.5214
```

```
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
```

```
[S1,S2,L,R]=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
A is scaled to left stochastic
    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
```

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

```
A = 5×5
     0     0     0     0     0
     0     3     6     3     2
     0     6     2     9     3
     0     7     2     3     7
     0     9     3    10     5
```

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

```
the vector of sums down each column is
S1 = 1×5
     0    25    13    25    17
the vector of sums across each row is
S2 = 5×1
     0
    14
    20
    19
    27
S1 and S2 have zero entries
```

Exercise 6

```
type newtons
```

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

    var = fzero(fun,x0);
    fprintf('The MATLAB approximation of the real zero of the function is:')
    var

    N = 0;

    num = 1000;

    while num > 1e-12
           newX = x0 - fun(x0)/dfun(x0);
           num = abs(x0-newX);
           x0 = newX;
           N = N+1;
    end

    fprintf('The number of iterations of the loop is:')
    N

    root = x0;
format long
```

Part (a):

```
fun = F;
dfun = F1;
x0 = 0.520;
root = newtons(fun,dfun,x0)
```

```
The MATLAB approximation of the real zero of the function is:

var = 0.5203

The number of iterations of the loop is:

N = 3

root =    0.520268992719590
```

```
x0 = -1.9;
root = newtons(fun,dfun,x0)
```

```
The MATLAB approximation of the real zero of the function is:

var =    0.520268992719590

The number of iterations of the loop is:

N =      6

root =    0.520268992719590
```

```
x0 = 0.999;
root = newtons(fun,dfun,x0)
```

```
   The MATLAB approximation of the real zero of the function is:

var =    0.520268992719590

   The number of iterations of the loop is:

N =      5

root =    0.520268992719590
```

Part (b):

```
fun = G;
dfun = G1;
%(1): 1.3
x0 = 1.3;
root = newtons(fun,dfun,x0)
```

```
   The MATLAB approximation of the real zero of the function is:

var =    1.324717957244746

   The number of iterations of the loop is:

N =      4

root =    1.324717957244746
```

```
%(2): 1
x0 = 1;
root = newtons(fun,dfun,x0)
```

```
   The MATLAB approximation of the real zero of the function is:

var =    1.324717957244746

   The number of iterations of the loop is:

N =      6

root =    1.324717957244746
```

```
%(3): 0.6
x0 = 0.6;
root = newtons(fun,dfun,x0)
```

```
   The MATLAB approximation of the real zero of the function is:

var =    1.324717957244746
```

```
   The number of iterations of the loop is:

N =     13

root =    1.324717957244746
```

```
%(4): 0.577351
x0 = 0.577351;
root = newtons(fun,dfun,x0)
```

```
   The MATLAB approximation of the real zero of the function is:

var =    1.324717957244746

   The number of iterations of the loop is:

N =     39

root =    1.324717957244746
```

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

```
x0 =    0.577350269189626
```

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

```
   The MATLAB approximation of the real zero of the function is:

var =    1.324717957244746

   The number of iterations of the loop is:

N =     96

root =    1.324717957244746
```

```
%(6): 0.577
x0 = 0.577;
root = newtons(fun,dfun,x0)
```

```
   The MATLAB approximation of the real zero of the function is:

var =    1.324717957244746

   The number of iterations of the loop is:

N =    101

root =    1.324717957244746
```

```
%(7): 0.4
x0 = 0.4;
root = newtons(fun,dfun,x0)
```

```
   The MATLAB approximation of the real zero of the function is:

var =    1.324717957244746

   The number of iterations of the loop is:

N =     14

root =    1.324717957244746
```

```
%(8): 0.1
x0 = 0.1;
root = newtons(fun,dfun,x0)
```

```
   The MATLAB approximation of the real zero of the function is:

var =    1.324717957244746

   The number of iterations of the loop is:

N =     35

root =    1.324717957244746
```

Bonus:

For choices 1-8 (excluding 4-6) it is evident that when the initial approximation is close to the root it takes fewer loops to identify the zero. For 1, our initial approximation was about 0.1 away and it only took 4 passes through the loop. For 3, we were about 0.7 away and the loop took 13 iterations. For 8, we were 1.2 away and therefore it took 35 loops to get the root. Each of these test support that the inital guess and how close it is to the actual root greatly impacts the time the loop takes, in terms of iterations.

For choices 4-6 a strange pattern takes place. Our initial approximation in 4-6 was closer to the actual root than choice 8, yet, numbers 4-6 took drastically longer to execute. This is due to the fact that the initial guess is close to the positive zero of the derivative of G(x). What this means is that we are dividing by a number very close to zero and therefore equalling a very large number. In order to get back down to a smaller number many loops are needed. In number 5, we can see that we start off with a very large number when outputting each loops iteration and we slowly decrease the size until finally getting down to the root. The first pass of the loop outputs 6.237e+15 which is a very large number and very far away from the actual root value of about 1.32. The formula decreases the size of the output through each loop. However it takes 96 iterations to finally get the root. It can be seen that if the initial guess is close to the zero of the derivative the formula will need substantial time to calculate a root. Overall, Newton's method is very dependent, in terms of execution time, on the initial guess.