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 # 14

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

1. Kyle Helstrom
2. John Henges
3. Chris Hall
4. Blake Goby
5. Conner Giordano
6. Harold Goslee

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

Exercise 1

Worked on by: Kyle Helstrom

```
format
format compact
type jord
```

```
function J=jord(n,r)
if (n > 1 && mod(n,1) == 0)

        vector = r * ones(n,1);
        J = diag(vector);

        for i = 1:(n-1)

            J(i,i+1) = 1;

        end
else
    J = [];
    fprintf('Jordan Block cannot be built')
end
end
```

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

```
r = 0.0975
```

```

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

```
Jordan Block cannot be builtJ =
     []
```

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

```
Jordan Block cannot be builtJ =
     []
```

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

```
Jordan Block cannot be builtJ =
     []
```

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

```
Jordan Block cannot be builtJ =
     []
```

```
%(e)
```

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

```
J = 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

Worked on by: John Henges

```
clear()

type added
```

```
function C = added(A, B)

[m, n] = size(A);
[k, p] = size(B);

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

if C ~= (A + B)
    fprintf('check the code!')
end
```

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

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

```
B = ones(4)
```

```
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 =

     []
```

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

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

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

```
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 =

     []
```

```


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

```
A = 3×4
    13    42    95    34
    19     5    50    91
    24    91    49    37
```

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

```
B = 3×4
    12    25    14    58
    79    41    95     6
    39    10    96    24
```

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

```
ans = 3×4
    25    67   109    92
    98    46   145    97
    63   101   145    61
```

```


C = added(A, B)
```

```
C = 3×4
    25    67   109    92
    98    46   145    97
    63   101   145    61
```

```
D = added(B, A)
```

```
D = 3×4
    25    67   109    92
    98    46   145    97
    63   101   145    61
```

```
if C == D
     'commutative property holds for the given A and B'
end
```

```
ans =
'commutative property holds for the given A and B'
```

```


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

```
k = 8
```

```
C = k * C
```

```
C = 3×4
         200         536         872         736
         784         368        1160         776
         504         808        1160         488
```

```
sum = (k * A) + (k * B)
```

```
sum = 3×4
         200         536         872         736
         784         368        1160         776
         504         808        1160         488
```

```
if C == sum
    fprintf('the distributive property holds for the given A and B')
end
```

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

Project 1

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);
else
   G=[];
   fprintf('Givens rotation matrix cannot be constructed')
   return
end

end
```

```
type closetozeroroundoff
```

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

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

```
Givens rotation matrix cannot be constructed
G =

     []
```

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

```
Givens rotation matrix cannot be constructed
G =

     []
```

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

```
G = 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
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
     1     0
     0     1
```

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

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

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

```
GI = zeros(3);
GI(1,1) = -1;
GI(2, 2) = -1;
GI(3,3) = 1
```

```
GI = 3×3
    -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(G*I, 7) == closetozeroroundoff(GI, 7))
    fprintf('The two matrices are equivalent!')
else
    fprintf('The two matrices are not equivalent')
end
```

```
The two matrices are equivalent!
```

```
x = ones(3,1);
G * x
```

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

Project 1: Exercise 4 (Toeplitz Matrix)

By: Blake Goby (group 14)

Part 1:

```
type toeplitze.m
```

```
function A=toeplitze(m,n,a)
% This function will go about reading the length of a vector (a) and
% transform it into a toeplitze matrix if satisfied correctly.
% Defining length of Vector a:
length_of_a=length(a);

%   When length of vector a equals (m+n-1), create toeplitze matrix:
if length_of_a==(m+n-1)
    for i=1:m
        for j=1:n
            A(i,j)=a(n+i-j);
        end
    end
else % if the vector a does not match in size... show:
    disp('Dimensions Mismatch')
end

end
```

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

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

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

```
Dimensions Mismatch
```

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

```
Dimensions Mismatch
```

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

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

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

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

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

```
% Task (1):
m=6;
n=6;
a=randi([0 100],1,11)
```

```
a = 1×11
    26    80     2    93    73    49    58    23    46    97    55
```

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

```
T = 6×6
    49    73    93     2    80    26
    58    49    73    93     2    80
    23    58    49    73    93     2
    46    23    58    49    73    93
    97    46    23    58    49    73
    55    97    46    23    58    49
```

```
triu(T)
```

```
ans = 6×6
    49    73    93     2    80    26
     0    49    73    93     2    80
     0     0    49    73    93     2
     0     0     0    49    73    93
     0     0     0     0    49    73
     0     0     0     0     0    49
```

```
% Task (2):
m=5;
n=5;
a=[0,0,0,0,1,0,0,0,0]
```

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

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

```
ans = 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 issymmetric(T)
    disp('T is a Symmetric Matrix')
end
```

```
T is a Symmetric Matrix
```

```
r=1:5;
c=1:6;
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
```

Project 1

Exercise 5

Worked on by: Conner Giordano

```
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)
if (n > 1 && mod(n,1) == 0)

        vector = r * ones(n,1);
        J = diag(vector);

        for i = 1:(n-1)

            J(i,i+1) = 1;

        end
else
    J = [];
    fprintf('Jordan Block cannot be built')
end
end
```

```
type stochastic.m
```

```
function [S1,S2,L,R]=stochastic(A)
    L=[];
    R=[];
    zeroRow = 0;
    zeroColumn = 0;
    if size(A,1) == size(A,2)
        n = size(A,1);
        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)
        for i=1:n
            if A(i,:)== closetozeroroundoff(0,7)
                zeroRow = 1;
            end
            if A(:,i) == closetozeroroundoff(0,7)
                zeroColumn = 1;
            end
        end
        if zeroColumn == 1 && zeroRow == 1
            fprintf('A cannot be scaled to be right nor left stochastic in any way')

        else
            if all(S1 == closetozeroroundoff(1,7),2) && all(S2 == closetozeroroundoff(1,7),1)
                fprintf('A is doubly stochastic')
                L = A
                R = A
                return
```

```
            else if all(S1 == closetozeroroundoff(1,7),2)
                fprintf('A is left stochastic')
                L = A
                return

            else if all(S2 == closetozeroroundoff(1,7),1)
                fprintf('A is right stochastic')
                R = A
                return
                end
                end
            end
            if all(S1 ~= closetozeroroundoff(0,7),2) && all(S2 ~= closetozeroroundoff(0,7),1)
                fprintf('A is neither left nor right stochastic but can be scaled to be either of them\n')
                L = zeros(n);
                R = zeros(n);
                for i=1:n
                    L(:,i) = 1/S1(1,i) .* A(:,i);
                end
                for i=1:n
                    R(i,:) = 1/S2(i,1) .* A(i,:);
                end

                if isequal(closetozeroroundoff(L,7),closetozeroroundoff(R,7))
                    fprintf('A is doubly stochastic after scaling')
                    R
                else
                    fprintf('Right stochastic of A after scaling\n')
                    R
                    fprintf('Left stochastic of A after scaling\n')
                    L
                end
            else if all(S1 ~= closetozeroroundoff(0,7),2)
                fprintf('A is neither left nor right stochastic but can be scaled to be left stochastic\n')
                L = zeros(n);
                for i=1:n
                    L(:,i) = 1/S1(1,i) .* A(:,i);
                end
                fprintf('A cannot be scaled to be right stochastic\n')

                fprintf('Left stochastic of A after scaling\n')
                L
            else if all(S2 ~= closetozeroroundoff(0,7),1)
                fprintf('A is neither left nor right stochastic but can be scaled to be right stochastic\n')
                R = zeros(n);
                for i=1:n
                    R(i,:) = 1/S2(i,1) .* A(i,:);
                end
                fprintf('Right stochastic of A after scaling\n')
                R
                fprintf('A cannot be scaled to be left stochastic')

                end
                end
            end
        end
    else
        fprintf('matrix A is not square')
        return
    end
end
```

```
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
A is 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
```

```
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
A is 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
```

```
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 be right stochastic
Right stochastic of A after scaling
R = 3×3
    0.5000         0    0.5000
         0         0    1.0000
         0         0    1.0000
A cannot be scaled to be left stochastic
ans = 1×3
    0.5000         0    2.0000
```

```
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 be left stochastic
A cannot be scaled to be right stochastic
Left stochastic of A after scaling
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
```

```
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
A is 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
```

```
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 be either of them
A is doubly stochastic after scaling
R = 3×3
    0.5333    0.0667    0.4000
    0.2000    0.3333    0.4667
    0.2667    0.6000    0.1333
ans = 1×3
    15    15    15
```

```
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 be either of them
Right stochastic of A after scaling
```

```
R = 3×3
    0.1515    0.3333    0.5152
    0.1467    0.3333    0.5200
    0.1453    0.3333    0.5214
Left stochastic of A after scaling
L = 3×3
    0.1515    0.1467    0.1453
    0.3333    0.3333    0.3333
    0.5152    0.5200    0.5214
ans = 1×3
    33    75   117
```

```
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 be either of them
Right stochastic of A after scaling
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
Left stochastic of A after scaling
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
ans = 1×5
     4     5     5     5     5
```

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

```
A = 5×5
     0     0     0     0     0
     0     4    10     4     7
     0     3    10     9     8
     0     5     6     1     7
     0     1     1     1     5
```

```
stochastic(A)
```

```
the vector of sums down each column is
S1 = 1×5
     0    13    27    15    27
the vector of sums across each row is
S2 = 5×1
     0
    25
    30
    19
     8
A cannot be scaled to be right nor left stochastic in any way
ans = 1×5
     0    13    27    15    27
```

Exercise #6

```
format
format compact
type newtons
```

```
function root = newtons (fun,dfun,x0)
format long
xn = x0;
x=fzero(fun,x0);

while abs(xn-x) > 10^-12
    xn1 = xn - fun(xn) / dfun(xn);
    xn = xn1;

end


root = xn;
end
```

```

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
```

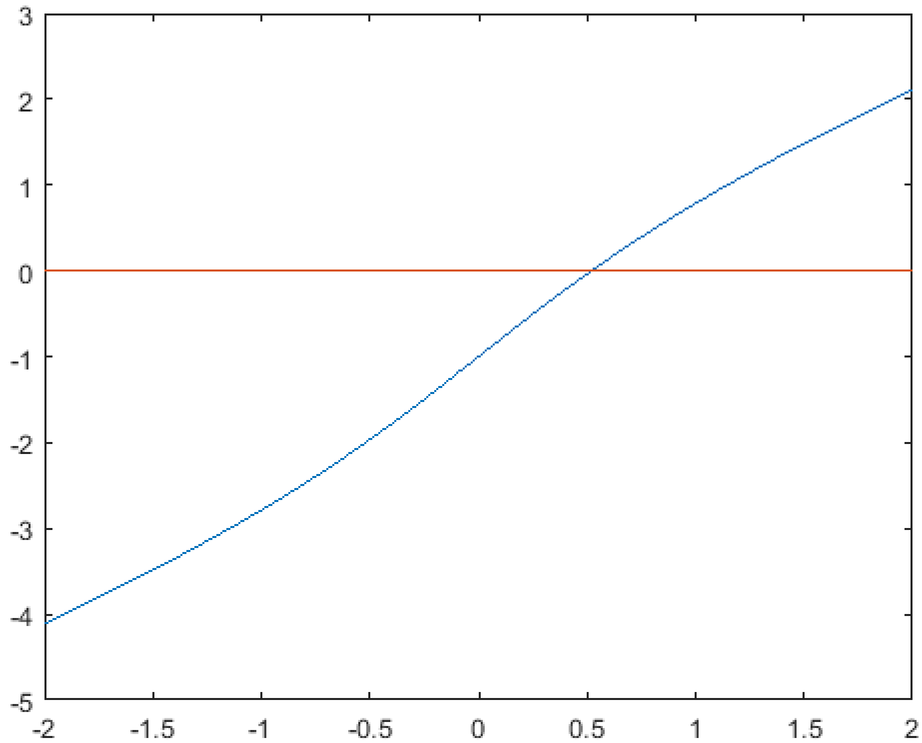
```

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

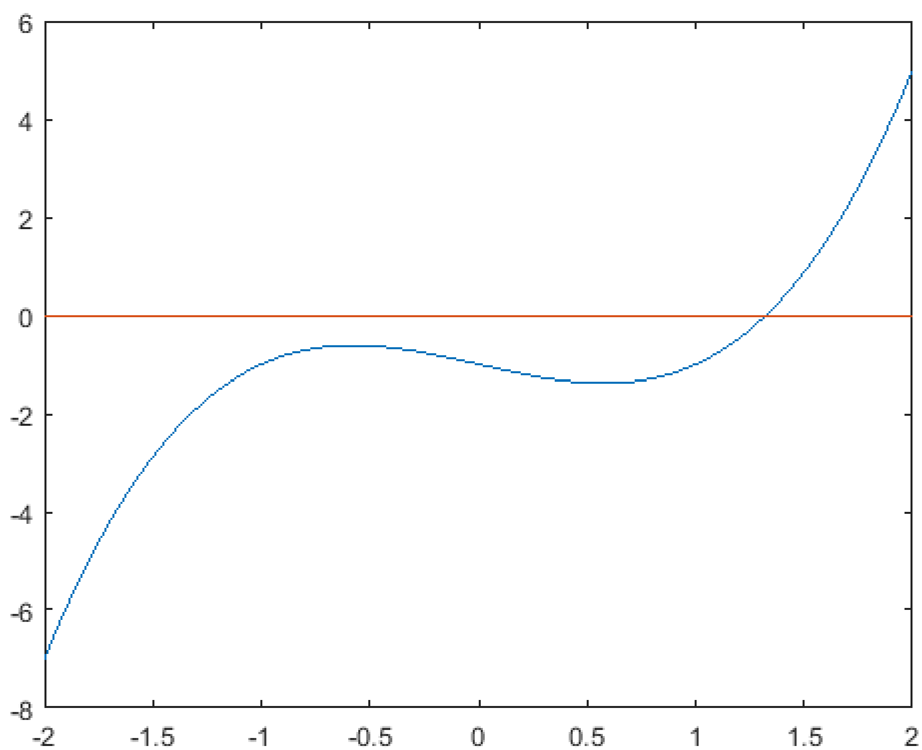
```
yzero = function_handle with value:

    @(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
```

p = $x^3 - x - 1$

```
roots(sym2poly(p))
```

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

Part A

```
fun=F;
dfun=F1;
x0=2;
root = newtons (fun,dfun,x0)
```

```
root =
   0.520268992719590
```

```
x0=.5;
root = newtons (fun,dfun,x0)
```

```
root =
   0.520268992719590
```

```
x0=.875;
root = newtons (fun,dfun,x0)
```

```
root =
   0.520268992719590
```

Part B

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

```
root =
   1.324717957244843
```

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

```
root =
   1.324717957244790
```

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

```
root =
   1.324717957244747
```

```
x0=0.577351;
```

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

```
root =
   1.324717957244746
```

```
x0=1/sqrt(3);
root = newtons (fun,dfun,x0)
```

```
root =
   1.324717957244746
```

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

```
root =
   1.324717957244807
```

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

```
root =
   1.324717957244746
```

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

```
root =
   1.324717957244746
```

```

% The closer the initial guess is to the real root, the less the iterations of Newton's Theorem

% If dfun(x0) is close to 0, then many more iterations of newton's theorem
% are required to reach an approximate root.
```

```

```