Project 1

Group 22

Members:

Kevin Lemos

Matthew Leonard

Johnny Li

Vinicius Martinson Santos

Hannah Mceachern

% EXERCISE 1

>> diary on

% 1) A (0,1)-matrix also called a binary matrix, Boolean matrix, or logical matrix. The matrix has to be of the size 5x4 with all elements either 0 or 1. The numbers 0 and 1 must appear randomly in this rectangular array. After you get the first output, arrow-up 2 more times to get another two random outputs.

>> randi([0,1], 5, 4)

ans =

1 0 0 0

1 0 1 0

0 1 1 1

1 1 0 1

1 1 1 1

% An alternant matrix - a matrix in which successive columns have a particular function applied to their entries. Begin with a random 4x1 vector x and create a matrix of the size 4x6 whose successive 6 columns are formed from the elementwise powers from 0 to 5 of x

>> rand(4,1)

ans =

0.7513

0.2551

0.5060

0.6991

>> % 4x6

>> x = rand(4,1)

x =

0.8909

0.9593

0.5472

0.1386

>> A = ones(4,6)

A =

1 1 1 1 1 1

1 1 1 1 1 1

1 1 1 1 1 1

1 1 1 1 1 1

>> y = 0:5

y =

0 1 2 3 4 5

>> A(1:end, :) = x.^y

A =

1.0000 0.8909 0.7937 0.7071 0.6300 0.5612

1.0000 0.9593 0.9202 0.8828 0.8468 0.8124

1.0000 0.5472 0.2994 0.1639 0.0897 0.0491

1.0000 0.1386 0.0192 0.0027 0.0004 0.0001

>> % An anti-diagonal matrix - an n-by-n anti-diagonal matrix is a matrix where all the entries are zero except those on the diagonal going from the lower left corner to the upper right corner (↗), known as the anti-diagonal. Take n = 5 and the elements on the anti-diagonal must be random integers in the range from 0 to 9. (You can use a matlab built-in function flipud)

>> flipud(diag(diag(randi([0 9], 5))))

ans =

0 0 0 0 5

0 0 0 7 0

0 0 3 0 0

0 9 0 0 0

1 0 0 0 0

>> % An arrowhead matrix – a square matrix containing zeros in all entries except for the first row, first column, and main diagonal. The size of the matrix has to be 6x6 and the entries in the first row, first column, and the main diagonal are random integers in the range between 10 and 100.

>> D = diag(diag(randi([10,100],6)))

D =

80 0 0 0 0 0

0 24 0 0 0 0

0 0 69 0 0 0

0 0 0 23 0 0

0 0 0 0 97 0

0 0 0 0 0 33

>> q = zeros(6,6)

q =

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

>> q = q + D

q =

80 0 0 0 0 0

0 24 0 0 0 0

0 0 69 0 0 0

0 0 0 23 0 0

0 0 0 0 97 0

0 0 0 0 0 33

>> q(1,:) = randi([10,100],6,1)

q =

22 89 62 60 23 87

0 24 0 0 0 0

0 0 69 0 0 0

0 0 0 23 0 0

0 0 0 0 97 0

0 0 0 0 0 33

>> q(:,1) = randi([10,100],1,6)

q =

66 89 62 60 23 87

41 24 0 0 0 0

56 0 69 0 0 0

46 0 0 23 0 0

16 0 0 0 97 0

31 0 0 0 0 33

% EXERCISE2

>> type('multi.m')

function [C, CRows, CColumns] = multi( A, B )

%MULTI Summary of this function goes here

% Detailed explanation goes here

size\_of\_A = size(A);

size\_of\_B = size(B);

if size\_of\_A(2) ~= size\_of\_B(1),

disp('The dimensions of A and B disagree');

C = [];

CRows = [];

CColumns = [];

return

end

C = A \* B;

% CRows = ...

% CColumns = ...

CRows = C;

CColumns = C;

end

>>

>>

>> A=randi(10,2,3)

A =

8 5 3

7 6 8

>> B=magic(2)

B =

1 3

4 2

>> [C, CRows, CColumns] = multi(A, B)

The dimensions of A and B disagree

C =

[]

CRows =

[]

CColumns =

[]

>> A \* B

Error using \*

Inner matrix dimensions must agree.

>>

>>

>> A= magic(5)

A =

17 24 1 8 15

23 5 7 14 16

4 6 13 20 22

10 12 19 21 3

11 18 25 2 9

>>

>>

>> B= ones(4,6)

B =

1 1 1 1 1 1

1 1 1 1 1 1

1 1 1 1 1 1

1 1 1 1 1 1

>>

>>

>> [C, CRows, CColumns] = multi(A, B)

The dimensions of A and B disagree

C =

[]

CRows =

[]

CColumns =

[]

>> A \* B

Error using \*

Inner matrix dimensions must agree.

>>

>>

>> A = magic(4)

A =

16 2 3 13

5 11 10 8

9 7 6 12

4 14 15 1

>>

>> B = ones(4,3)

B =

1 1 1

1 1 1

1 1 1

1 1 1

>>

>> [C, CRows, CColumns] = multi(A, B)

C =

34 34 34

34 34 34

34 34 34

34 34 34

CRows =

34 34 34

34 34 34

34 34 34

34 34 34

CColumns =

34 34 34

34 34 34

34 34 34

34 34 34

>> A \* B

ans =

34 34 34

34 34 34

34 34 34

34 34 34

>>

>>

>> A = ones(4)

A =

1 1 1 1

1 1 1 1

1 1 1 1

1 1 1 1

>>

>> B = diag([2,3,4,5])

B =

2 0 0 0

0 3 0 0

0 0 4 0

0 0 0 5

>>

>> [C, CRows, CColumns] = multi(A, B)

C =

2 3 4 5

2 3 4 5

2 3 4 5

2 3 4 5

CRows =

2 3 4 5

2 3 4 5

2 3 4 5

2 3 4 5

CColumns =

2 3 4 5

2 3 4 5

2 3 4 5

2 3 4 5

>>

>> A \* B

ans =

2 3 4 5

2 3 4 5

2 3 4 5

2 3 4 5

>>

diary on

format compact

%Exercise 3

type givensrot

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

%Produces an nxn Givens rotation matrix

G=eye(n);

if 1<=i && i<j && j<=n

    G(i,i)=cos(theta);

    G(j,j)=cos(theta);

    G(i,j)=-sin(theta);

    G(j,i)=sin(theta);

else

    G=[];

end

end

%(1)

n=4,i=3,j=2,theta=(pi/2)

n =

     4

i =

     3

j =

     2

theta =

    1.5708

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

G =

     []

%(2)

n=5,i=2,j=4,theta=(pi/4)

n =

     5

i =

     2

j =

     4

theta =

    0.7854

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

G =

    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

%(3)

n=3,i=1,j=2,theta=(pi)

n =

     3

i =

     1

j =

     2

theta =

    3.1416

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

G =

   -1.0000   -0.0000         0

    0.0000   -1.0000         0

         0         0    1.0000

e1=[1;0;0]

e1 =

     1

     0

     0

e2=[0;1;0]

e2 =

     0

     1

     0

e3=[0;0;1]

e3 =

     0

     0

     1

x=[1;1;1]

x =

     1

     1

     1

G\*e1

ans =

   -1.0000

    0.0000

         0

G\*e2

ans =

   -0.0000

   -1.0000

         0

G\*e3

ans =

     0

     0

     1

G\*x

ans =

   -1.0000

   -1.0000

    1.0000

%the output of G\* e1/e2/e3 produced the individual collumns of G matrix.

%The outpur of G\* x produced the diagonal of the G matrix which could be thought of as the rotation discussed in the geometrical meaning.

diary off

diary Project1

diary on

%Exercise#4-Matthew Leonard

type toeplitz

function A=toeplitz(m,n,a)

for i=1:m

for j=1:n

A(i,j)=a(n+i-j);

end

end

end

%1)

%a)

m=4

m =

4

n=3

n =

3

a=transpose([1:6])

a =

1

2

3

4

5

6

A=toeplitz(m,n,a)

A =

3 2 1

4 3 2

5 4 3

6 5 4

%a=[1;2;3;4;5;6]

%b)

m=3

m =

3

n=4

n =

4

a=randi(10,6,1)

a =

6

7

9

10

6

2

A=toeplitz(m,n,a)

A =

10 9 7 6

6 10 9 7

2 6 10 9

%a=[6;7;9;10;6;2]

%c)

m=4

m =

4

n=4

n =

4

a=[zeros(3,1);[1:4]']

a =

0

0

0

1

2

3

4

A=toeplitz(m,n,a)

A =

1 0 0 0

2 1 0 0

3 2 1 0

4 3 2 1

%a=[0;0;0;1;2;3;4]

%2)

m=5

m =

5

n=5

n =

5

c=randi([10,100],m,n)

%produces random integer between 10 and 100 in each element of 5 by 5 matrix

c =

23 32 66 60 44

33 94 53 93 61

86 41 42 36 16

33 27 85 78 14

84 32 63 78 58

a=triu(c,1)

%creates an upper triangular matrix

a =

0 32 66 60 44

0 0 53 93 61

0 0 0 36 16

0 0 0 0 14

0 0 0 0 0

A=toeplitz(m,n,a)

A =

0 0 0 0 0

32 0 0 0 0

0 32 0 0 0

0 0 32 0 0

0 0 0 32 0

%a=[0;0;0;0;0;32;0;0;0]

%3)

m=4

m =

4

n=4

n =

4

a=eye(m,n)

%creates a 4 by 4 identity matrix

a =

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

A=toeplitz(m,n,a)

A =

0 0 0 1

0 0 0 0

1 0 0 0

0 1 0 0

%a=[1;0;0;0;0;1;0]

diary off

diary on

format compact

% Exercise#5

%By Johnny Li

type stochastic

function P=stochastic(A)

%Pre-Setup

[rows, columns] = size(A);

S1=sum(A,1);

S2=sum(A,2);

P=[rows columns];

Test=0;

%Stochastric Matrix: accepts a square matrix A with nonnegative entries as

%an input. The output will be a stochastic matrix P as defined

if rows ~= columns

disp('Error: Not a square matrix!');

return;

end

entries = A >= 0;

if sum(entries(:)) ~= numel(A)

disp ('Error: Matrix A has a negative entries');

return;

end

%1) check whether a matrix A contains both a zero column and a zero row.

%If yes, the output has to be a message. The outputs in this case are S1,

%S2, and P = [].

for i=1:rows

for j=1:columns

R2=sum(A(i,:));

C2=sum(A(:,j));

if (R2 == zeros(1,rows))

if(C2 == zeros(columns,1))

disp('A is not stochastic and cannot be scaled to stochastic.')

disp('S1:')

disp(S1)

disp('S2:')

disp(S2)

P=[];

break;

end

end

end

%Break second for loop.

if(R2 == 1)

if (C2 ==1)

Test=1;

break;

end

end

end

%2)function checks whether a matrix A is: doubly stochastic,left

%stochastic, only right stochastic, or neither left nor right

%stochastic but can be scaled to stochastic.

C=all(abs(S1 - 1) < 1E-5);

R=all(abs(S2 - 1) < 1E-5);

if C && R

disp('Matrix A is a double stochastic matrix')

P=A;

elseif C

disp('Matrix A is a only a left stochastic matrix')

P=A;

elseif R

disp('Matrix A is a only a right stochastic matrix')

P=A;

%4)Function to output the message “neither left nor right

%stochastic but can be scaled to stochastic”, S1 and S2.

elseif (Test == ~1)

disp('Matrix A is neither left nor right stochastic but can be scaled to stochastic.')

disp('S1:')

disp(S1)

disp('S2:')

disp(S2)

%5)If S1 does not have any zero entry, use this vector to modify A

%into a left-stochastic matrix P by scaling each of its columns by

%the reciprocal of the corresponding entry of S1. If S1 has a zero

%entry, use the vector S2 to modify A into a right-stochastic

%matrix P by scaling each of its rows by the reciprocal of the

%corresponding entry of S2.

if (find(R2) == ~0)

for i1=1:rows

for j1=1:columns

A(j1,i1) = A(j1,i1)/S1(1,i1);

P=A;

end

end

elseif (find(C2) == ~0)

for i2=1:rows

for j2=1:columns

A(i2,j2) = A(i2,j2)/S2(i2,1);

P=A;

end

end

end

end

end

A=[0.5, 0, 0.5; 0, 0, 1; 0.5, 0, 0.5]

A =

0.5000 0 0.5000

0 0 1.0000

0.5000 0 0.5000

P=stochastic(A)

Matrix A is a only a right stochastic matrix

P =

0.5000 0 0.5000

0 0 1.0000

0.5000 0 0.5000

A=A'

A =

0.5000 0 0.5000

0 0 0

0.5000 1.0000 0.5000

P=stochastic(A)

Matrix A is a only a left stochastic matrix

P =

0.5000 0 0.5000

0 0 0

0.5000 1.0000 0.5000

A=[0.5, 0, 0.5; 0, 0, 1; 0, 0, 0.5]

A =

0.5000 0 0.5000

0 0 1.0000

0 0 0.5000

P=stochastic(A)

Matrix A is neither left nor right stochastic but can be scaled to stochastic.

S1:

0.5000 0 2.0000

S2:

1.0000

1.0000

0.5000

P =

0.5000 0 0.5000

0 0 1.0000

0 0 1.0000

A=A'

A =

0.5000 0 0

0 0 0

0.5000 1.0000 0.5000

P=stochastic(A)

Matrix A is neither left nor right stochastic but can be scaled to stochastic.

S1:

1.0000 1.0000 0.5000

S2:

0.5000

0

2.0000

P =

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 =

0.5000 0 0.5000

0 0.5000 0.5000

0.5000 0.5000 0

P=stochastic(A)

Matrix A is a double stochastic matrix

P =

0.5000 0 0.5000

0 0.5000 0.5000

0.5000 0.5000 0

A=magic(3)

A =

8 1 6

3 5 7

4 9 2

P=stochastic(A)

Matrix A is neither left nor right stochastic but can be scaled to stochastic.

S1:

15 15 15

S2:

15

15

15

P =

0.5333 0.0667 0.4000

0.2000 0.3333 0.4667

0.2667 0.6000 0.1333

A= diag([1,2,3])

A =

1 0 0

0 2 0

0 0 3

P=stochastic(A)

Matrix A is neither left nor right stochastic but can be scaled to stochastic.

S1:

1 2 3

S2:

1

2

3

P =

1 0 0

0 1 0

0 0 1

A=[0, 0, 0; 0, 0.5, 0.5; 0, 0.5, 0.5]

A =

0 0 0

0 0.5000 0.5000

0 0.5000 0.5000

P=stochastic(A)

A is not stochastic and cannot be scaled to stochastic.

S1:

0 1 1

S2:

0

1

1

P =

[]

diary off

% exercise 6

%(a)

x=linspace(0,4,8);

y=atan(x)+x-1;

plot(x, y); figure(gcf)

%initial approximation: 0.51

diary on

format compact

syms x

f=atan(x)+x-1

f =

x + atan(x) - 1

g = diff(f)

g =

1/(x^2 + 1) + 1

diary off

diary on

format compact

type newtons

function [ root ] = newtons( N, x )

%NEWTONS Exstimate the Nth apromimation of some function given an initial

%estimte x

format long;

a = x;

b = 0;

if N > 0 && floor(N) == N

for i=1:N

b = a - (atan(a+a-1))/(1/(a^2+1)+1);

a = b;

end

root = a;

else

disp("Error in newtons: N must be a positive int")

root = 0;

end

N = 5

N =

5

x = 0.51;

netwon(N, x)

{Undefined function or variable 'netwon'.}

root = newtons(N, x)

root =

0.499999825468722

%our function says the root of the function is 0.49999983, which is close to our guess of 0.51

diary on

format compact

x=linspace(0,4,8);

y=x .^3-x-1;

plot(x, y)

syms x

f=x^3-x-1

f =

x^3 - x - 1

g = diff(f)

g =

3\*x^2 - 1

N = 5, x = 1.5

N =

5

x =

1.500000000000000

newtons\_1(N , x)

a =

1.333333333333333

a =

1.324074074074074

a =

1.324771397980121

a =

1.324713555363625

a =

1.324718320054237

ans =

1.324718320054237

x=1;

newtons\_1(N , x)

a =

1.500000000000000

a =

1.333333333333333

a =

1.324074074074074

a =

1.324771397980121

a =

1.324713555363625

ans =

1.324713555363625

x = 0.6;

newtons\_1(N , x)

a =

3.483333333333333

a =

2.335552405143048

a =

1.665189395977849

a =

1.371783457175882

a =

1.322732208757471

ans =

1.322732208757471

x = 0.57;

newtons\_1(N , x)

a =

3.991811218186311

a =

2.653910191672096

a =

1.839969355766820

a =

1.432438961261267

a =

1.325148286888767

ans =

1.325148286888767

%a good approximation is 1.32514829

%(c)

newtons\_1(10 , x)

a =

3.991811218186311

a =

2.653910191672096

a =

1.839969355766820

a =

1.432438961261267

a =

1.325148286888767

a =

1.324682657176459

a =

1.324720867710020

a =

1.324717717379093

a =

1.324717977013926

a =

1.324717955615420

ans =

1.324717955615420

newtons\_1(10 , 0.6)

a =

3.483333333333333

a =

2.335552405143048

a =

1.665189395977849

a =

1.371783457175882

a =

1.322732208757471

a =

1.324885175428368

a =

1.324704200707010

a =

1.324719091194591

a =

1.324717863788590

a =

1.324717964947177

ans =

1.324717964947177

newtons\_1(100 , 0.6)

ans =

1.324717957244746

newtons\_1(100 , 0.57)

ans =

1.324717957244746

%it would apear that the two values do converge for the values at (3) and (4). This is becouse they return the same value at larger inputs

%I would say it converges quickly, considering that the larger N gets, the faster the function converges

diary off