ENSC 180: Introduction to Engineering Analysis

Assignment 4 Solutions

1. Given [A]= [4 3 1;3 7 -1; 1 -1 9], [B]= [10 8 7; 3 -3 0; 14 1 7] and [C]= [1 -1; 4 7; 9 5]. Using MATLAB built-in functions, write a script or use the command line to perform the following operations and print out the resulting matrices.

   (a) A+B (b) A*C (c) A^T (d) AA^T (e) CC^T (f) $A^{-1}B^{-1}$ (g) rank of A and C (h) determinant of A (i) solve the equation system [A]{x} =[C]. (20 marks)

<u>Code;</u>

```
%% A4 Q1 - Performing matrix operations
% Given three matrices A, B and C, this script performs various matrix
% operations using these variables and prints the results upon completion

% Define the matrix variables;
A= [4 3 1;3 7 -1; 1 -1 9];
B= [10 8 7; 3 -3 0; 14 1 7];
C= [1 -1; 4 7; 9 5];

% Perform the operations;
% >>1 mark for a-f, 0.5 mark for g1 and g2, 2 marks for i; 10 total<<
a = A+B;
b = A*C;
c = A.'; % .' is the transpose, ' is the complex conjugate transpose. A.', A' and
transpose(A) all acceptable. (no difference with real numbers)
d = A*A.';
e = C*C';
f = inv(A)*inv(B); %(A^ - 1)*(B^ - 1) also acceptable
g1 = rank(A);
g2 = rank(C);
h = det(A);
i = A\C;    %pinv(A)*C, inv(A)*C and (A^-1)*C are also acceptable, but C\A or other
variants are NOT acceptable (x=A\b solves the equation A*x=b, while x=b\A solves
b*x=A)
            %>>2 marks if correct, 1 mark if switched C and A, 0 else<<

% Print out the results; >>10 marks total; for each output that does not
% match answers, deduct 1 mark each for a max of 10<<
disp("a = A+B"); disp(a);
disp("b = A*C;"); disp(b);
disp("c = A.'"); disp(c);
disp("d = A*A.'"); disp(d);
disp("e = C*C'"); disp(e);
disp("f = inv(A)*inv(B)"); disp(f);
disp("g1 = rank(A)"); disp(g1);
disp("g2 = rank(C)"); disp(g2);
disp("h = det(A);"); disp(h);
disp("i = A\C;"); disp(i);
```

Outputs;

a = A+B

```
    14    11     8
     6     4    -1
    15     0    16
```

b = A*C;

```
    25    22
    22    41
    78    37
```

c = A.'

```
     4     3     1
     3     7    -1
     1    -1     9
```

d = A*A.'

```
    26    32    10
    32    59   -13
    10   -13    83
```

e = C*C'

```
     2    -3     4
    -3    65    71
     4    71   106
```

f = inv(A)*inv(B)

```
    0.1200    0.3375   -0.1293
   -0.0173   -0.1140    0.0238
   -0.0946   -0.2301    0.1122
```

g1 = rank(A)

```
     3
```

g2 = rank(C)

```
     2
```

h = det(A);

```
   154
```

i = A\C;

```
   -0.9091   -2.0000
    1.1364    2.0000
    1.2273    1.0000
```

2. Write a user-defined function to add and multiply two matrices without using the MATLAB matrix-wise + and * operations. The function should have the two matrices as input, check whether the addition and multiplication operations are valid for the given matrices and if valid, manually compute (using for-loops) and present the output matrix. If the operations are not valid, the function should print a statement to that effect. Print out the results of the function using inputs [A],[B], [A],[C] and [C],[A]. (20 marks)

<u>Code (**NOTE; marking scheme in code is wrong, double all their listed values!):</u>

```
function [add, mult] = userMatAddMult( A, B )
%[add, mult] = userMatAddMult( A, B ): Takes two real-valued matrices A and
%B as input.  Checks their size to determine if addition and multiplication
%of A and B are possible (A+B and A*B).  If possible, returns the matrices
%add and mult as a result and prints out their values.  If not possible,
%returns value NaN for the given operations and prints out error messages.

    %ADDITION REQUIREMENTS; the two matrices must be the same size

    %MULTIPLICATION REQUIREMENTS; the number of rows in A must equal the
    %number of columns in B (A=mxn, B=axb, then n=a)

    % Booleans, determine if operations are performed, default False/0
    doadd = 0;
    domult = 0;

    % Default the outputs to NaN
    add = NaN;
    mult = NaN;

    % ********** CHECK CASES *************
    % check for cases, extract dimensions of rows and columns
    [Ar,Ac] = size(A);
    [Br,Bc] = size(B);
    if size(A)==size(B)
        % equal sizes, perform both operations
        doadd = 1;
        domult = 1;
    else
        % A and B are matrices of different sizes, cannot add >>1 mark<<
        disp("ERROR: Cannot add, matrix dimensions do not agree");
        %Multiplication case
        if Ac == Br
            % columns of A match rows of B, can perform multiplication >>1 mark <<
            domult = 1;
        else
            % matrix dimensions are incompatible >> 1 mark <<
            disp("ERROR: Cannot multiply, inner matrix dimensions do not agree");
        end
    end
    % >> total 3 marks to check all the non-equal cases <<

    %perform the operations
    if doadd
        % ************* ADDITION **********
```

```
        % add each element
        add= zeros(size(A));
        for i=1:Ar
            for j=1:Ac
                add(i,j) = A(i,j)+B(i,j); % >> 1 mark <<
            end
        end
    end

    if domult
        % ************* MULTIPLICATION **********
        mult = zeros(Ar,Bc); % matrix size is outer dimensions
        for i=1:Ar
            for j=1:Bc
                % the element of mult(i,j) is the sum of products along the
                % row Ar and the column Bc.
                % Loop along said row/coln (each are same size)
                for k = 1:Ac
                    % Loop down vectors A(i,:) and B(:,j) to
                    % compute each product separately, then add to the
                    % current sum of the final matrix's element
                    product = A(i,k) * B(k,j); % >> 2 marks for doing the corect
element-wise products<<
                    mult(i,j) = mult(i,j) + product; % >> 2 marks for doing the
correct summation of products <<
                end
            end
        end
    end
    %>>total 5 marks for the operations<<

    %Display the results
    disp("Addition results;");disp(add);
    disp("Multiplication results;");disp(mult);
    %>> total 2 marks, -1 mark for each incorrect output up to max of 2 <<
End
```

Outputs;

Addition results;

14	11	8
6	4	-1
15	0	16

Multiplication results;

63	24	35
37	2	14
133	20	70

ERROR: Cannot add, matrix dimensions do not agree

Addition results;

NaN

Multiplication results;

25	22
22	41
78	37

ERROR: Cannot add, matrix dimensions do not agree

ERROR: Cannot multiply, inner matrix dimensions do not agree

Addition results;

NaN

Multiplication results;

NaN

3. Find the determinant of ([A]-α[I]) algebraically without using MATLAB, where [A] is given in Q1 above, I is an identity matrix and α is a scalar. Solve the equation det([A]-α[I]) =0 to find the values of α by plotting the equation using MATLAB over the range 0 ≤α ≤10. Display the plot and values of α.(20 marks)

Algebraically;

$$\det([A]-\alpha[I]) = \begin{vmatrix} 4-\alpha & 3 & 1 \\ 3 & 7-\alpha & -1 \\ 1 & -1 & 9-\alpha \end{vmatrix}$$

$$= (4-\alpha)\begin{vmatrix} 7-\alpha & -1 \\ -1 & 9-\alpha \end{vmatrix} - 3\begin{vmatrix} 3 & -1 \\ 1 & 9-\alpha \end{vmatrix} + \begin{vmatrix} 3 & 7-\alpha \\ 1 & -1 \end{vmatrix}$$

$$= (4-\alpha)[(7-\alpha)(9-\alpha)-1] - 3[(3)(9-\alpha)-(-1)] + [(3)(-1)-(7-\alpha)]$$

$$= (4-\alpha)[63-7\alpha-9\alpha+\alpha^2-1] - 3[27-3\alpha+1] + [-3-7+\alpha]$$

$$= (4-\alpha)[62-16\alpha+\alpha^2] - 3[28-3\alpha] + [-10+\alpha]$$

$$= 248-64\alpha+4\alpha^2-62\alpha+16\alpha^2-\alpha^3-84+9\alpha-10+\alpha$$

$$= \mathbf{154-116\boldsymbol{\alpha}+20\boldsymbol{\alpha}^2-\boldsymbol{\alpha}^3}$$

(15 marks total; 10 marks if correct procedure but incorrect answer due to algebra error)

Solving for alpha;

```
%% A4 Q3 (part 2) - Solving det(A-alphaI)=0 for alpha graphically
% Assuming the student had algebraically determined the equation in terms
% of alpha, the equation is input and then plotted over a range of
% 0<=alpha<=10, then uses a for loop to determine and print out the
% location of the root/alpha value

% getting function values from 0 to 10
alpha = 0:0.1:10;
determinant = 154 - 116.*alpha + 20.*(alpha.^2) - (alpha.^3);
% >> 1 mark; -1 mark if no matrix-wise operations and/or -1 if incorrect
% formula <<

% plotting the function
figure;
plot(alpha,determinant);
title("det(A-alphaI)");
xlabel("alpha");
ylabel("function");
% >> 1 mark for correct graph <<

for i=1:length(determinant)-1
    if determinant(i)*determinant(i+1)<=0
        fprintf("The value of alpha is %.2f\n", alpha(i));
    end
end
%>> 3 marks for each correct root; -1 if incorrect formula <<
```

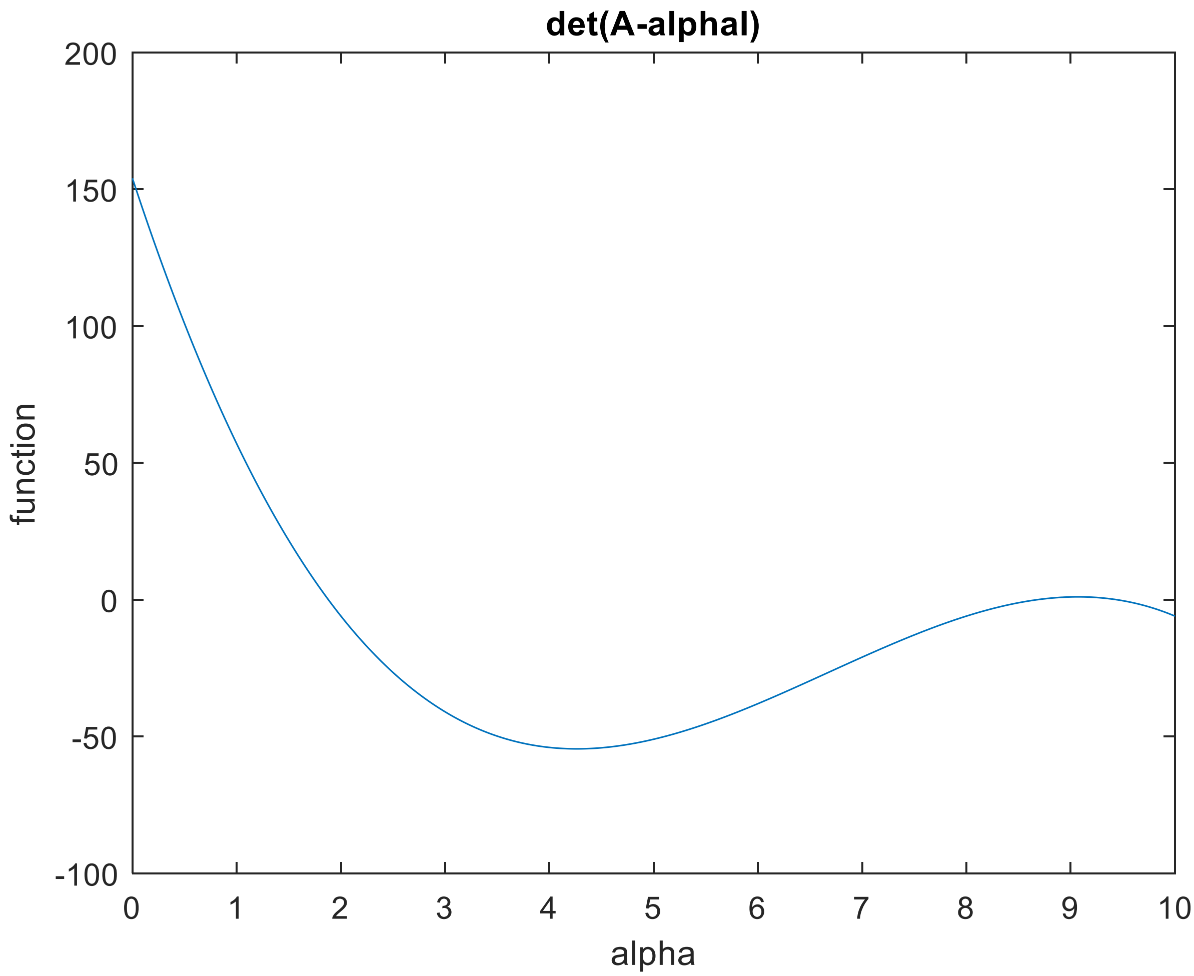
(5 marks total)

Output;

The value of alpha is 1.87

The value of alpha is 8.68

The value of alpha is 9.43



4. Write a MATLAB code to perform Gauss elimination for a general system of form [A]{x} = {b}. Assume [A] is a real square matrix and {b} is also real. Your code should determine whether the rank of [A] is less than the size of [A] based on the results of Gauss elimination and decide the next step. Use your code to solve Q1(i). (40 marks)

<u>Code:</u>

```
%% A4 Q4 - Performing Gauss Elimination for Q1(i)
% Uses the matrix A and columns of C, performs Gauss elimination to produce
% the solutions of Q1(i)

%Test the function
x1 = userGaussElim(A,C(:,1));
x2 = userGaussElim(A,C(:,2));

function x = userGaussElim(A,b)
% x = userGaussElim(A,b): Given an equation of the form Ax=b, with A being
% a real, square matrix and b being a real vector, performs Gauss
% elimination to determine the rank of A and the results of x

%NOTE: This function takes each column b individually, but can also be
%implemented with a matrix containing all the columns to do the elimination
%simultaneously on all of them; the implementation here was done for
%clarity to follow/describe the process, and focuses on the Gauss
%elimination and rank-checking aspects.  Do not penalize students for
%implementing to take a full matrix for b if their code performs correctly.

    % get number of rows and columns
    [r, c] = size(A);

    % create augmented matrix
    M = [A b];

    % PERFORM GAUSS ELIMINATION >> Total 20 marks for their algorithm for Gauss
elimination being logical and performing as expected for any general case (watch out
for accidental zero-division cases!)<<
    for i=1:c-1
        % Loop from first to second-last column of A

        for j = r:-1:i+1
            %Constrain the zero elements to lower-left corner of matrix

            % Loop from last element of column (bottom row) to the element prior to
the
            % column number.

            % eg. 3x3, last element of each column is in row 3, so 1st column will
            % have zeros in 3rd and 2nd elements, 2nd column will have zero
            % only in 3rd element, etc.

            %desired value to set to zero
            z = M(j,i);
            %value in the column directly above that element
            nonz = M(j-1,i);
```

```
            %Check the values (if either are already zero or not)
            if z == 0
                % Element already zero, do nothing and continue
                continue;
            elseif nonz == 0
                % Row above current element is zero, swap rows and continue
                swapRow = M(j-1,:);
                M(j-1,:) = M(j,:);  %take current row and move it up
                M(j,:) = swapRow;   %take previous row above and move it to current
spot
            else
                % Neither z nor nonz are zero, subtract some multiple of
                % the above row from the current row to render z==0.

                %find the scalar needed to set the non-zero equal to the
                %desired zero element (scalar.*nonz = z)
                scalar = z./nonz;

                %Apply that scalar to the row above z
                M(j-1,:) = M(j-1,:).*scalar;
                %subtract the rows
                M(j,:) = M(j,:)-M(j-1,:);
            end

            % end of iteration will have M with any updated row values and
            % a zero element at M(j,i)
        end
    end


    %Separate the augmented matrix back into the components
    A2 = M(1:r,1:c);
    b2 = M(1:r,c+1);

    % CHECK RANK AND PERFORM APPROPRIATE SUBSTITUTION >> Total 20 marks for all of
the rank conditions, performing back-substitution and getting the correct solutions
<<
    if sum(A2(r,:))==0 && b2(r) ~= 0
        % The system is inconsistent (cannot have zero = a non-zero value),
        % there is no solution
        disp("System inconsistent, no solution: ");
        x = NaN;
    elseif sum(A2(r,:))==0 && b2(r) == 0
        % The system is under-defined, there are infinite solutions
        disp("System under-defined, infinite solutions: ");
        x = Inf;
    else
        % rank is equal to number of equations, one solution
        disp("System consistent, one solution: ");
        % Use back substitution to determine the value of x
        x = zeros(r,1);
        for i = r:-1:1
            % Loop from bottom row upwards

            % the equation for each x(i) is x(i) =
            % (b(i)-x(i+1)A(i+1,i)-...x(r)A(r,i))/A(i,i)
            xA=0;
```

```
            for j=i+1:r
                % Loop down columns

                % compute the sum of x*As
                xA = xA + (x(j).*A2(i,j));
            end
            % compute the value for x(i)
            x(i) = (b2(i)-xA)./A2(i,i);
        end
    end
    disp(x);

end
```

<u>Output:</u>

System consistent, one solution:

-0.9091

1.1364

1.2273

System consistent, one solution:

-2.0000

2.0000

1.0000