Total Points: / 100

Submission Instruction: Please submit this homework on Canvas in a single pdf format. The filename should be "HWXX_FullName_RedID.pdf" (ex. HW03_JamesGault_12345678.pdf).
Please copy your Matlab code in the given box. Adjust the box size as needed.
Please also submit all your m files separately. **Don't zip them**.

Full Name: Evan Peres

Red ID: 129964468

Email address: eperes2481@sdsu.edu

2D Arrays (Write your code in the box.)

1. Write a function with header [M] = myDelMatrix(x) that deletes the last column and the last row of an input matrix and outputs the modified matrix. Assume that the input matrix has a dimension of at least 2x2. (/ 10)

Test Cases:

```
>> x = [1 6 2 3 6; 2 6 9 1 3; 1 5 7 2 4];
>> M = myDelMatrix(x)
M=
     1 6 2 3
      2 6 9 1
```

```
function [M] = myDelMatrix(x)

   M = x(1:end-1,1:end-1);

end
```

2. Write a function with header y = myFilterMatrix(x, b) that takes a 2D array x as an input and outputs all values strictly greater than b into a column array. Use logical indexing array. (/ 10)

Test Cases:

```
>> x = [1 6 2 3 6; 2 6 9 1 3; 1 5 7 2 4];
>>y = myFilterMatrix(x,5)
y =
          6
          6
          6
          9
```

Total Points: / 100

7

```
function y = myFilterMatrix(x, b)

   indices = x>b;
   y = x(indices);

end
```

3. Write a function with header [M] = myCheckerBoard(n) where M is an n × n matrix with the following form:

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Note that the upperleft element should always be 1. You may assume that n is a strictly positive integer. (/ 20)

Test Cases:

```
>>M= myCheckerBoard(1)
M=
        1
>>M= myCheckerBoard(2)
M=
        1       0
        0       1
>> M= myCheckerBoard(3)
M =
        1       0       1
        0       1       0
        1       0       1
```

```
function [M] = myCheckerBoard(n)
   M = zeros(n);
   for i = 1:n
      for j = 1:n
         if (mod(j, 2) == 0) && (mod(i,2) == 0)
             M(i,j) = 1;
         elseif (mod(j, 2) == 1) && (mod(i,2) == 1)
            M(i,j) = 1;
         end
      end
   end
end
```

Total Points: / 100

Vector&Matrix (Write your code in the box.)

4. Write a function with header [M1,M2] = mySplitMatrix(M) where M is a matrix, M1 is the left half of M, and M2 is the right half of M. In the case where there is an odd number of columns, the middle column should go to M1. Hint: The size and **ceil** functions will be useful for this. You may assume that M has at least two columns. (/ 10)

Test Cases:

```
>> M = [1 2 3; 4 5 6; 7 8 9]
M=
     1     2     3
     4     5     6
     7     8     9
>> [M1, M2] = mySplitMatrix(M)

M1 =
     1     2
     4     5
     7     8
M2 =
     3
     6
     9
```

```
function [M1,M2] = mySplitMatrix(M)

  [rows collumns] = size(M)
  M1 = M(:,1:ceil(collumns/2))
  M2 = M(:,ceil(collumns/2)+1:end)

end
```

5. Write a function with header [out] = myIsOrthogonal(v1,v2, tol) where v1 and v2 are column vectors of the same size and tol is a scalar value strictly larger than 0. The output argument, out, should be 1 if the angle between v1 and v2 is within tol of $\pi/2$, i.e., $|\pi/2 - \theta| < tol$, and 0 otherwise. You may assume that v1 and v2 are column vectors of the same size, and that tol is a positive scalar. (/ 10)

Test Cases:

```
>> out = myIsOrthogonal([1;0.001],[0.001;1],0.01)

out =
     1
>> out = myIsOrthogonal([1;0.001],[0.001;1],0.001)

out =
```

Total Points: / 100

```
     0
>> out = myIsOrthogonal([1;0.001],[1;1],0.01)

out =
     0
>> out = myIsOrthogonal([1;1],[-1;1],1e-10)

out =
     1
>> out = myIsOrthogonal([1; 0; 1; -1; 1; 1],[0; 1;0; 1;1;0],1e-10)

out =
     1
```

```
function [out] = myIsOrthogonal(v1,v2, tol)
   format long
   A = norm(v1)
   B = norm(v2)
   dotAB = dot(v1,v2)

   theta = acos(dotAB/(A*B))

   if abs(pi/2 - theta) < tol
      out = 1;
   else
      out = 0;
   end
end
```

Total Points: / 100

Linear Algebra (Write your code in the box.)

6. Write a function with header [N, x] = myNumSols(A,b) where A and b are a matrix and compatibly-sized column vector, respectively, N is the number of solutions of the system Ax = b, and x is a solution to the same system. If there are no solutions to the system of equations, then x should be an empty matrix. If there is one or an infinite number of solutions, then myNumSols should return one using the methods described in the our lecture. You may assume that b is a column vector and that the number of elements in b is the same as the number of rows in A. The output x should be a column vector. You may assume that if the system has an infinite number of solutions, then the A matrix will have more columns than rows, i.e., A is fat. In this case, you should solve the system using x = pinv(A)∗b. (/ 20)

Test Cases:

```
>> A= reshape(1:15, 3, 5);
>> b = [-5; -4; -3]
>> [N, x] = myNumSols(A,b)
N =
        Inf
x =
        1.0000
        0.6000
        0.2000
        -0.2000
        -0.6000
>> b = [-1.5; 2 ; 7 ]
>> [N, x] = myNumSols(A,b)
N=
        0
x =
        [ ]

>> A = 3*eye(5)
>> b = [1; 2; 3; 4; 5];
>> [N, x] = myNumSols(A,b)
N =
        1
x =
        0.3333
        0.6667
        1.0000
        1.3333
        1.6667
```

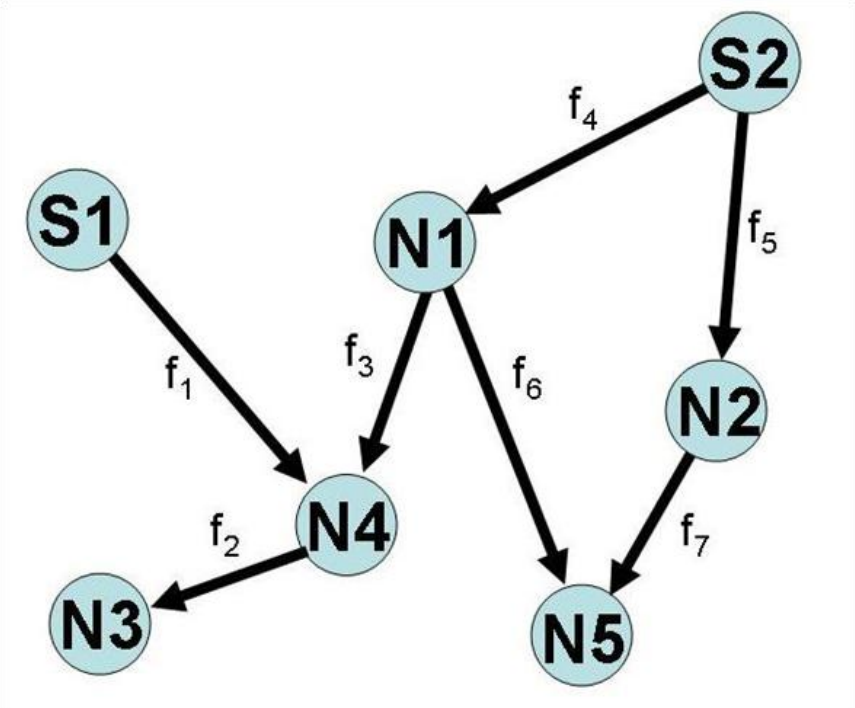
Total Points: / 100

```
function [N, x] = myNumSols(A,b)
   augM = [A b];
   [rowesc,jb] = rref(augM)
   [m, n] = size(A);

   if any(jb == n+1) % if pivot in augmented collumn
      N = 0;
   elseif length(jb) ~= n % if no free variables
      N = Inf;
   else % else
      N = 1;
   end

   if N == 0 % no solutions
      x = [];
   elseif N == 1 % one solution
      x = A\b;
   elseif N == Inf % inf solutions
      x = pinv(A)*b;
   end
end
```

7. Consider the following network consisting of two power supply stations denoted by S1 and S2 and five power recipient nodes denoted by N1 to N5. The nodes are connected by power lines which are denoted by arrows, and power can flow between nodes along these lines in both directions.



Total Points: / 100

Let d_i be a positive scalar denoting the power demands for node i, and assume that this demand must be met exactly. The capacity of the power supply stations is denoted by S. Power supply stations must run at their capacity. For each arrow, let f_j be the power flow along that arrow. Negative flow implies that power is running in the opposite direction of the arrow.

Write a function with header [f] = myFlowCalculator(S, d) where S is 1 × 2 vector representing the capacity of each power supply station, and d is a 1 × 5 row vector representing the demands at each node, i.e., d(1) is the demand at node 1. The output argument, f, should be a 1 × 7 row vector denoting the flows in the network, i.e., f(1) = f_1 in the diagram. The flows contained in f should satisfy all constraints of the system, i.e., power generation and demands. Note that there may be more than one solution to the system of equations.

Note that the total flow into a node must equal the total flow out of the node plus the demand, i.e., for each node i, $f_{inflow} = f_{outflow} + d_i$. You may assume that $\Sigma S_j = \Sigma d_i$.

Hint: you may need to set up 7 linear equations and call the function **myNumSols** in P6 to solve it.(/ 20)

Test Case:

```
>> f = myFlowCalculator([10 10], [4 4 4 4 4])
>> f=
10.0000    4.0000   -2.0000    4.5000    5.5000    2.5000    1.5000
>> f = myFlowCalculator([10 10], [3 4 5 4 4])
>> f=
10.0000    5.0000   -1.0000    4.5000    5.5000    2.5000    1.5000
```

```
function [f] = myFlowCalculator(S, d)
   % Define how the flow of energy in each node
   b = [S,d];
   b = rot90(b,3);

   % Define the flow of energy in the system
   A = zeros(7);
   A(1,1)=1;
   A(2,4)=1; A(2,5)=1;
   A(3,3)=-1; A(3,4)=1; A(3,6)=-1;
   A(4,5)=1; A(4,7)=-1;
   A(5,2)=1;
   A(6,1)=1; A(6,2)=-1; A(6,3)=1;
   A(7,6)=1; A(7,7)=1;

   % myNumSols
   augM = [A b];
   [rowesc,jb] = rref(augM)
```

```
    [m, n] = size(A);

    if any(jb == n+1) % if pivot in augmented collumn
        N = 0;
    elseif length(jb) ~= n % if no free variables
        N = Inf;
    else % else
        N = 1;
    end

    if N == 0 % no solutions
        f = [];
    elseif N == 1 % one solution
        f = A\b;
    elseif N == Inf % inf solutions
        f = pinv(A)*b;
    end
    f = rot90(f);
end
```