# CS4 Homework 7: Introduction to MATLAB & Linear Algebra

**Due at Noon, Tuesday, April 18, 2017**

The total points in this Homework sum to 300, but it will only count as one regular homework.

The primary goal of Part I and Part II of this homework is to familiarize you with MATLAB. Most of the questions are straightforward, and multiple hints are included, the challenge for you will be to convert to using MATLAB from Python. Please feel free to post to Piazza for additional support.

Part III of this homework tests your understanding of new technical material related to Linear Algebra and the Matrix Project.

**Setup**

For each homework assignment, there may be support files that you will need to complete the assignment. These can be copied to your home directory by using the following command in a Brown CS Terminal window:

```
cs4_install hw07
```

There should now be a hw01 folder within your homeworks folder. Using Terminal, you can move into the hw01 folder with the cd command.

Create a script inside this folder called cs4hw01.m with answers and comprehensive test cases. Use the following `Publish` friendly format to organize your work in cs4hw01.m

```
%% Question 7.1.1
% Put comments for any written answers to non-coding questions here,
% as well as any calculations and script calls.

Use the following template for functions that you are asked to write.
%%
% *A) Test cases*
assert(isequal(yourFunction(a,b,c), [1 2 2])); % Put your assert statements here.
...
%%
% *B) Function Listing(s)*
dbtype yourFunction.m % Where yourFunction.m is the name of a function you have written.
...
%% Question 7.1.2
...
```

---

## Contents


- Part I: Arrays & Branching
- Question 7.1.1 (ITM 2.7): Make a matrix (5)
- Question 7.1.2: Array Clipping (5)
- Question 7.1.3 (ITM 11.3): 2-D Plotting (5)
- Question 7.1.4 (ITM 3.2): Regular Patterns (10)
- Question 7.1.5 (ITM 3.4): Splitting (10)
- Question 7.1.6 (ITM 3.12): Area of an Annulus (5)
- Question 7.1.7: Linear Indices (5)
- Question 7.1.8 (ITM 4.1): Tactical tips (5)

- Question 7.1.9 (ITM 4.2): Multiple Dispatch (5)
- Question 7.1.10 (ITM 4.3): Triangle Test (10)
- Question 7.1.11 (ITM 4.4): Vector Padding (5)
- Question 7.1.12 (ITM 4.5): Vector Padding w/o if-statements (5)
- Question 7.1.13 (ITM 4.7): Alarms (10)
- Question 7.1.14 (ITM 4.8): Roots (20)
- Part II: Iteration
- Question 7.2.1: Even and odd entries (15)
- Question 7.2.2 (ITM 5.6): Compound Interest (15)
- Question 7.2.3 (ITM 5.11): Prime Fibonacci numbers (20)
- Question 7.2.4: Binary Search (30)
- Question 7.2.5 (ITM 11.9): Power play (10)
- Question 7.2.6 (ITM 11.11): Stochastic Fractal (10)
- Part III: Linear Algebra
- Question 7.3.1 (ITM 12.1) - Matrix multiplication is distributive (15)
- Question 7.3.2 (ITM 12.2) - Orthogonal vectors (10)
- Question 7.3.3 (ITM 12.4) - Linearly independent columns (15)
- Question 7.3.4 - Determinants (20)
- Question 7.3.5 (ITM 12.8) - Linear systems (20)
- Question 7.3.6 (ITM 12.9) - Network flow (20)

## Part I: Arrays & Branching

Avoid using `for` and `while` loops to solve the problems in part I, instead use array indexing and MATLAB's, elementwise operations and the other array support commands that are built into MATLAB.

## Question 7.1.1 (ITM 2.7): Make a matrix (5)

Create the following matrix in a single assignment. Try to use as few numbers as possible. Hint: Use :, ones, [, ], ',' and ';'

```
M =

     0     0     0     0     0     1
     0     0     0     0     0     1
     1     2     3     4     5     1
     0     2     4     6     8     1
     8     7     2     5     9     1
```

## Question 7.1.2: Array Clipping (5)

Create an anonymous function myLimit that changes all values in an array that are greater than 1 to 1. Include tests of your function. Hint: Use logical arrays, elementwise array multiplication and addition.

## Question 7.1.3 (ITM 11.3): 2-D Plotting (5)

Plot the functions

$$f(x) = 3 + e^{-x} sin(6x)$$

and

$$g(x) = 4 + e^{-x} cos(6x)$$

for 0 <=x <= 5 on a single axis using the hold command. Give the plot axis labels, a title, and a legend.

Put your solution directly in your hand-in script.

Hint: Use `plot, xlabel, ylabel, title, legend`

### Question 7.1.4 (ITM 3.2): Regular Patterns (10)

Write a function with header [M]=myCheckerBoard(n), where M is an n x n matrix consisting of zeroes and ones laid out in a checkerboard pattern. Assume n is odd. For n=5, M would have the following form:

```
M =

  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
```

Hint: use linear indexing and `reshape`.

Additional (No extra credit): Do you see a one line solution for n even? How about both cases?

### Question 7.1.5 (ITM 3.4): Splitting (10)

Write a function [M1, M2] = mySplitMatrix(M) where M is a matrix, M1 is the left half of M, 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. Assume has M has at least two columns.

Hint: use `size` and `ceil`.

### Question 7.1.6 (ITM 3.12): Area of an Annulus (5)

Let $r_1$ and $r_2$ be the radius of circles with the same center and let $r_2 > r_1$. Write a function with header `[A] = myDonutArea(r1,r2)`, where A is the area outside of the circle with radius r1 and inside the circle with radius r2. Make sure that myDonutArea is vectorized. Assume that r1 and r2 are row vectors of the same size.

### Question 7.1.7: Linear Indices (5)

Create a function with header

ind = myLinearIndex(i,j,m,n)

which computes the linear index of the element (i,j) in an m by n matrix. Include test cases.

Hint: try using a matrix generated by `reshape(1:m*n, m,n)` as one of your test cases.

### Question 7.1.8 (ITM 4.1): Tactical tips (5)

Write a function with signature

```
[tip] = myTipCalc(bill, party)
```

Where `bill` is the total cost of a meal and `party` is the number of people in the group. The `tip` should be calculated as 15% for a party strictly less than six people, 18% for a party strictly less than eight, 20% for a party less than 11, and 25% for a party of 11 or more. **The tip amount must be rounded to the nearest penny.**

### Question 7.1.9 (ITM 4.2): Multiple Dispatch (5)

Write a function that computes multiple operations on a pair of arrays. Your function should have the header

```
[f] = myMultOperation(a,b,operation)
```

The input argument `operation` is a string that is either `'plus'`, `'minus'`, `'mult'`, `'div'`, or `'pow'`, and `f` should be computed as a+b, a-b, a.*b, a./b, and a.^b for the respective values for operation. If an operation that is not recognized is used, report this fact using the `error` function. Make sure your function is vectorized.

Hint: The `strcmp` function is an easy way to compare strings.

Below are some examples:

```
x=[1 2 3 4];
y=[2 3 4 5];
myMultOperation(x,y,'plus')
```

```
ans =

     3     5     7     9
```

```
myMultOperation(x,y,'minus')
```

```
ans =

    -1    -1    -1    -1
```

```
myMultOperation(x,y,'exp')
Error using myMultOperation
exp isn't an allowed operation.
```

## Question 7.1.10 (ITM 4.3): Triangle Test (10)

Consider a triangle with vertices at (0,0), (1,0), and (0,1). Write a function with signature

```
[S] = myInsideTriangle(x,y)
```

Where `S` is the string `'outside'` if the point (`x`, `y`) is outside of the triangle, `'border'` if the point is exactly on the border of the triangle, and `'inside'` if the point is on the inside of the triangle.

hint: Test if the point is outside the unit square, then check if the point is on or below the diagonal, etc.

`Example`

```
myInsideTriangle(.1,.1)
```

```
ans =
```

```
inside
```

### Question 7.1.11 (ITM 4.4): Vector Padding (5)

Write a function with signature `[out] = myMakeSize10(x)`, where `x` is a vector, and `out` is the first 10 elements of x if x has more than 10 elements, or out is the array x padded with enough zeros to make it length 10 if x has fewer than 10 elements. Do not change the shape of x (i.e. if x is a column vector, out should also be a column vector; similarly, if x is a row vector, out should also be row vector). If x is empty or 1 by 1, out should be a row vector.

Use if-statements (e.g., `if`, `elseif` and `else`) to implement your solution.

Examples

```
myMakeSize10(1:2)
```

```
ans =

     1     2     0     0     0     0     0     0     0     0
```

```
myMakeSize10([3 ; 4])
```

```
ans =

     3
     4
     0
     0
     0
     0
     0
     0
     0
     0
```

```
myMakeSize10(1:20)
```

```
ans =

     1     2     3     4     5     6     7     8     9    10
```

### Question 7.1.12 (ITM 4.5): Vector Padding w/o if-statements (5)

Rewrite myMakeSize10 without using if-statements (i.e., using only logical and array operations). Name your new function `myMakeSize10Challenge`.

Tip: You should be able to reuse the test cases you created when you wrote `myMakeSize10`.

Hint: Use the default behaviors of (,),[,], and assignment = to your advantage.

## Question 7.1.13 (ITM 4.7): Alarms (10)

Most engineering systems have redundancy. That is, an engineering system has more than is required to accomplish its purpose. Consider a nuclear reactor whose temperature is monitored by three sensors. An alarm should go off if any two of the sensor readings disagree. Write a function with signature `[response] = myNukeAlarm(S1,S2,S3)` where `S1`, `S2`, and `S3` are the temperature readings for sensor 1, sensor 2, and sensor 3, respectively. The output response should be the string `'alarm!'` if any two of the temperature readings disagree by strictly more than 10 degrees and `'normal'` otherwise.

## Question 7.1.14 (ITM 4.8): Roots (20)

Let $Q(x)$ be the quadratic equation $Q(x) = ax^2 + bx + c$ for some scalar values $a$, $b$, and $c$. A root of a function, $Q(x)$, is a value of its arguments, $r$, such that the function's value is zero, $Q(r) = 0$.

The two roots of a quadratic equation can be described by the quadratic formula, which is

$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

A quadratic equation has either two real roots (i.e., $b^2 > 4ac$), two imaginary roots (i.e., $b^2 < 4ac$), or one root, $r = -b/2a$. Write a function with header `[nRoots, r] = myNRoots(a,b,c)` where `a`, `b`, and `c` are the coefficients of the quadratic $Q(x)$, nRoots is 2 if $Q$ has two real roots, 1 if Q has one root, -2 if $Q$ has two imaginary roots, and r is an array containing the roots of $Q$.

## Part II: Iteration

Use for and while loops where needed to solve the problems in part II.

## Question 7.2.1: Even and odd entries (15)

Write a function with header `[Q] = myOddEven(M)` where `Q(i,j) = M(i,j)` if `i*j` is odd and `Q(i,j) = -M(i,j)` if `i*j` is even. You may assume that `M` is a non-empty, matrix of doubles. Do not assume `M` will always be square.

Example Output:

```
myOddEven([3 4; 6 7])
```

```
ans =

     3    -4
    -6    -7
```

## Question 7.2.2 (ITM 5.6): Compound Interest (15)

An interest rate, $i$, on a principle amount $P_0$, defines a payment amount for the use of the principle over time. Compound interest is accumulated according to the formula $P_n = (1+i)P_{n-1}$, where $n$ is the number of compounding periods (usually months or years). Write a function with header `[periods] = mySavingPlan(P0, i, goal)` where periods is the whole number of periods it will take your total savings account balance to attain goal at i% interest per period. Assume no additional deposits or withdrawals. Assume $i$ is positive.

Use iteration to solve this problem, not a closed-form solution.

Example Output:

```
mySavingPlan(1000, 0.05, 2000) % Rate of 5 percent
```

```
ans =

    15
```

```
mySavingPlan(1000, 0.07, 2000) % Rate of 7 percent
```

```
ans =

    11
```

```
mySavingPlan(500, 0.07, 2000) % 7 percent with a bigger difference between P_0 and goal
```

```
ans =

    21
```

## Question 7.2.3 (ITM 5.11): Prime Fibonacci numbers (20)

Write a function with header [fibPrimes] = myNFibPrimes(N), where fibPrimes is a list of the first N numbers that are both a Fibonacci number and prime. Assume N is a non-negative integer.

Do not use the recursive implementation of Fibonacci numbers. Use an iterative approach.

Hints: 1 is not prime. You are welcome to to use the MATLAB `isprime` function, or write your own function.

Example Output:

```
myNFibPrimes(3)
```

```
ans =

     2     3     5
```

```
myNFibPrimes(8)
```

```
ans =

  Columns 1 through 6

           2            3            5           13           89          233

  Columns 7 through 8
```

```
    1597      28657
```

### Question 7.2.4: Binary Search (30)

Bisection is called binary search when it is applied to discrete values. Create a function that a performs binary search for a value v on an array A assuming that the values of A(:) are unique and in ascending order. Your function should return the linear index of the matching element, if one is found, and [] if no matching element is found.

Use the following signature for your functions

```
function i = myBinarySearch(A, v)
```

Be sure to include test cases for missing elements, and extreme cases, like the desired element at either end of A(:). Note: Your function should work for arrays of any size so long as all the values in A are unique, and A(:) is in ascending order.

Example Output:

```
myBinarySearch([1 2 4],4)
```

```
ans =

     3
```

### Question 7.2.5 (ITM 11.9): Power play (10)

Write a function with header [] = myPolyPlotter (n,x) that plots the polynomials p_k(x) = x^k for k = 1, ..., n. Make sure your plot has axis labels, a title, legend, and grid lines. Use `doc legend` to learn how to control the `Location` of a legend.

hint: Use a 'cell arrary' to build-up the list of legend labels, e.g., `leg(end+1) = {['x^' num2str(i)]}`

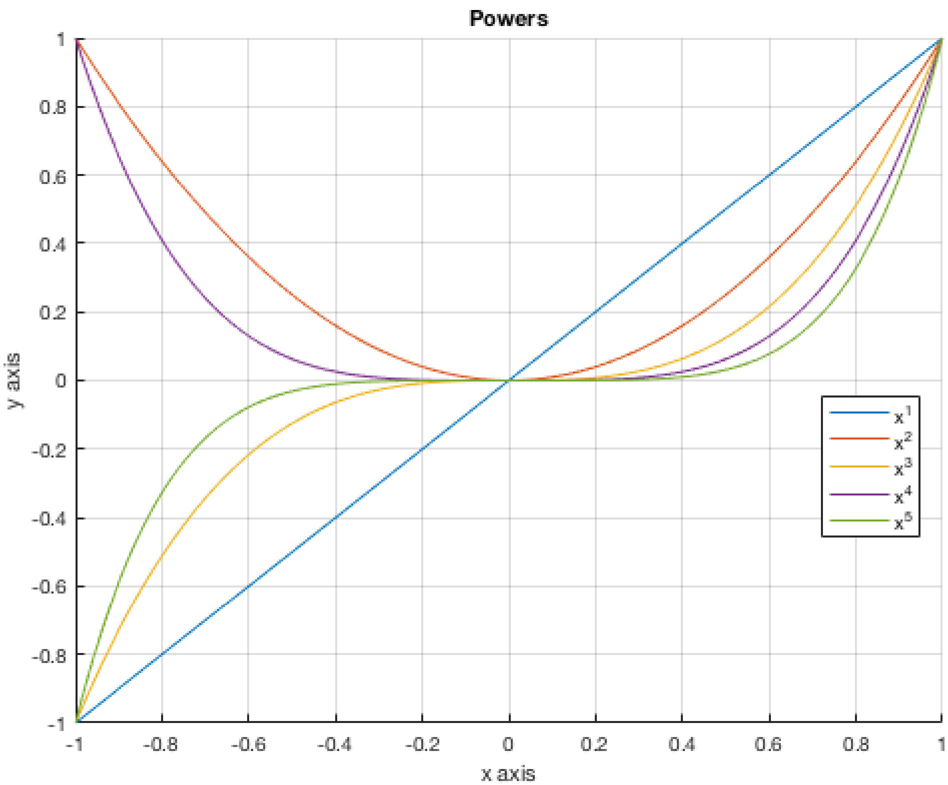
You do not need to write any test cases for `myPolyPlotter`.

Make your graph match the following:

```
myPolyPlotter(5,-1:.01:1)
```

```
leg =

     {}


leg =

    'x^1'


leg =

    'x^1'    'x^2'
```

```
leg =

    'x^1'    'x^2'    'x^3'


leg =

    'x^1'    'x^2'    'x^3'    'x^4'


leg =

    'x^1'    'x^2'    'x^3'    'x^4'    'x^5'
```




## Question 7.2.6 (ITM 11.11): Stochastic Fractal (10)

In this problem you will generate a set of points random points and plot them. The points are defined as follows: $x_1 = 0$ and $y_1 = 0$, and the points $(x_i, y_i)$ for i = 2, ..., n are generated by the following probabilistic relationships:

With 1% probability:

$x_i = 0$

$y_i = .16y_{i-1}$

With 7% probability:

$x_i = .2x_{i-1} - .26y_{i-1}$

$y_i = .23x_{k-1} + .22y_{i-1} + 1.6$

With 7% probability:

$x_i = -.15x_{i-1} + .28y_{i-1}$

$y_i = .26x_{i-1} + .24y_{i-1} + 0.44$

With 85% probability:

$x_i = .85x_{i-1} + .04y_{i-1}$

$y_i = -.04x_{i-1} + .85y_{i-1} + 1.6$

Write a function with signature `[] = myFern(n)` that generates the points (x_i,y_i) for i=1,..., n and plots them using black dots.

Hint: Your code will run faster if you generate all the points, and then plot them using a single plot command.

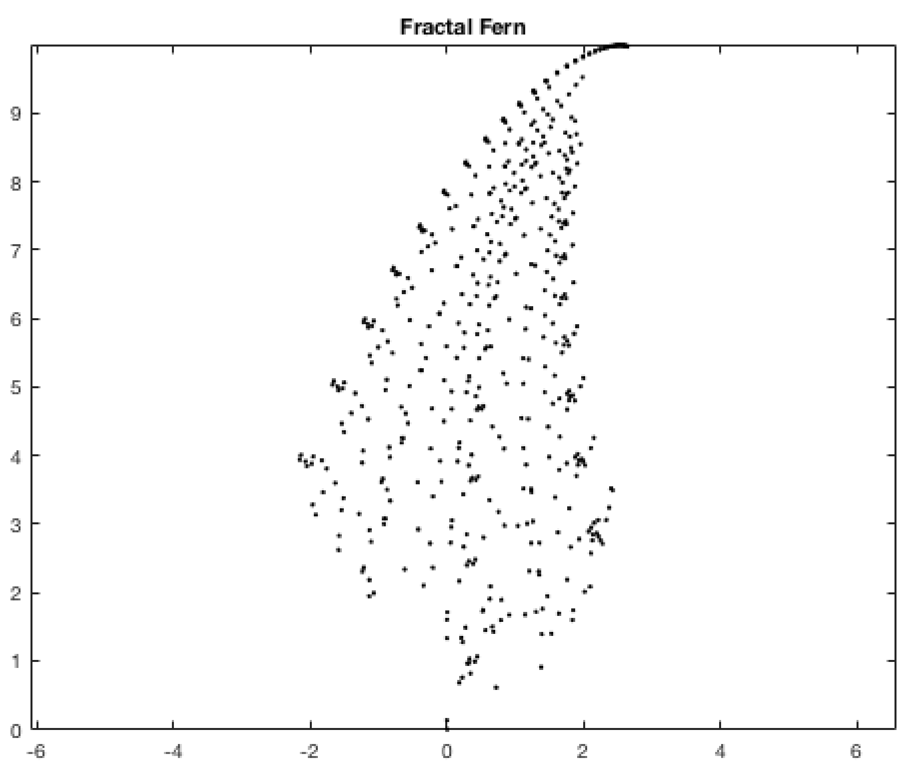
Try your function for n = 1000 and n = 10000. The image generated is called a stochastic fractal. Despite being very easy to generate, fractals often exhibit a lot of complexity. They have application in both computer graphics and image compression.

You do not need to create test cases for myFern.

hint: You can make the dots bigger using the LineWidth attribute, e.g., `plot(x,y,'.', 'LineWidth',3)`

Below is a figure generated by `myFern(500)`.

```
close all
figure(1)
myFern(500)
```



## Part III: Linear Algebra

### Question 7.3.1 (ITM 12.1) - Matrix multiplication is distributive (15)

Show that matrix multiplication distributes over matrix addition. That is, show that A(B+C) = AB + AC for any matrices A, B, and C of compatible size.

You may include your answer in comments within your `cs4hw07.m` file, or take a photo or screenshot of your solution, saved in your hw07 folder as distributive.jpg.

### Question 7.3.2 (ITM 12.2) - Orthogonal vectors (10)

Write a function with the header

```
[out] = myIsOrthogonal(v1,v2, tol)
```

where `v1` and `v2` are non-empty, nonzero vectors of the same size and `tol` is a positive scalar value. The output argument, `out`, should be true (1) if the angle between `v1` and `v2` is within `tol` of pi/2 (that is, $|\pi/2 - \theta| < tol$), and false (0) otherwise.

Example Outputs:

```
myIsOrthogonal([1 0], [0 1], 1e-10)
```

```
ans =

     1
```

```
myIsOrthogonal([1 1]', [0 1]', 1e-10)
```

```
ans =

     0
```

```
myIsOrthogonal([.1 0 0], [0 .2 0], 1e-10)
```

```
ans =

     1
```

### Question 7.3.3 (ITM 12.4) - Linearly independent columns (15)

Write a function that finds the linearly independent columns of a matrix. Use the following signature:

```
[B] = myMakeLinInd(A)
```

If the rank of A is n, this function should return the first n linearly independent columns of A. The result, B, will be a matrix of dimensions size(A,1) by rank(A), consisting of linearly independent columns. Assume A is a matrix.

Hint: build B incrementally and use `rank([B,A(:,i)])` to test whether or not you should add column `A(:,i)` to `B`.

Example Output:

```
A = [1 0 2 3 1; 0 0 1 0 1]
myMakeLinInd(A)
```

```
A =

     1     0     2     3     1
     0     0     1     0     1


ans =

     1     2
     0     1
```

## Question 7.3.4 - Determinants (20)

Laplace expansion (also known as cofactor expansion) is a method for computing the determinant of a matrix. If A is an n by n matrix, the minor to element (i,j) is the (n-1) by (n-1) matrix formed by all the rows of A except the ith and all the columns of A except the jth. Denote the minors of A via A^{i,j}. The determinant of A is given by the following formula:

For i, any index between 1 and n,

$\det(A) = \sum_{j=1}^{n}(-1)^{i+j}A_{i,j}\det(A^{i,j})$.

Remarks: The cofactor $C_{i,j}$ is defined as $(-1)^{i+j}\det(A^{i,j})$. So, the above expression can be rewritten as

$\det(A) = \sum_{j=1}^{n} A_{i,j}C_{i,j}$.

Computation can also proceed along a column; i.e.,

$\det(A) = \sum_{j=1}^{n} A_{j,i}C_{j,i}$.

yields the same result.

Write a function with signature

`function [d]=myRecDet(A)`

that returns the determinant of A. Assume A is a square matrix.

Hint: `myRecDet` should use the above formula to compute the determinant; you should not use MATLAB's function `det`. Since the formula is recursive, you should use recursion!

Example Outputs:

```
myRecDet([1 2 3; 3 4 5; 5 6 1])
```

```
ans =

    12
```

```
myRecDet([1 2 0; 3 4 0; 5 6 0])
```

```
ans =

     0
```

### Question 7.3.5 (ITM 12.8) - Linear systems (20)

Write a function with header

```
[N, x] = myNumSols(A,b)
```

where A is a matrix and b is a compatibly-sized column vector. 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 solution, using the methods described in lecture. 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; that is, A is fat. In this case, you should solve the system using x = pinv(A)*b.

hint: For your test cases, when comparing floating numbers remember to use tolerances, e.g.

```
tol = 1e-10;
assert(abs(a-b)<tol)
```

Also remember to use norm when comparing vectors and matrices.

```
% Example Outputs:
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)
```

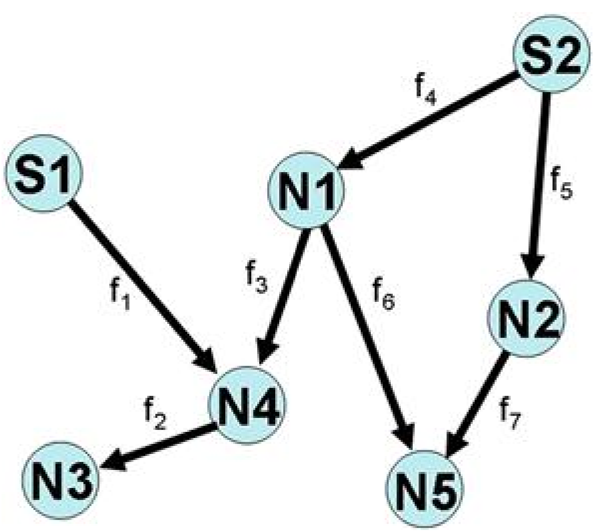
```
N =

     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
```

### Question 7.3.6 (ITM 12.9) - Network flow (20)

Consider the following network consisting of two power supply stations, denoted by S1 and S2, and five power recipients, denoted by N1 through N5. The stations and recipients are connected by power lines, which are denoted by arrows.



Let $d_i$ be the power demand of recipient $i$, and assume that this demand must be met exactly. Denote the output of the power supply stations by $s_j$. Assume supply matches demand (i.e., $\sum s_j = \sum d_i$).

The total outflow from each power station must equal its output.

The total flow into each recipient node must equal the total flow out of the node plus the demand; that is, for each node $i$, $f_{in} = f_{out} + d_i$ (e.g., $f_1 + f_3 = d_4 + f_2$).

Write a function with signature

```
[f] = myFlowCalculator(s,d)
```

where `s` is a 1x2 vector representing the capacity of each power supply station, and `d` is a 1x5 vector representing the demands at each node (e.g., `d(1)` is the demand at node 1). The output argument, `f`, should be a 1x7 vector denoting the flows in the network (e.g., $f(1) = f_1$ in the diagram). The flows contained in `f` should satisfy all constraints of the system, such as power generation and demands. As long as all constraints are met, flows can be positive, negative, or zero. There may be more than one solution to the system of equations.

Hint: Solve $Af' = [s';d']$, where $A$ incorporates the constraints above.

Example Outputs:

```
S = [10 10]; d = [4 4 4 4 4];
myFlowCalculator(S,d)
```

```
ans =

   10.0000    4.0000   -2.0000    4.5000    5.5000    2.5000    1.5000
```

```
S = [10 10]; d = [3 4 5 4 4];
myFlowCalculator(S,d)
```

```
ans =

   10.0000    5.0000   -1.0000    4.5000    5.5000    2.5000    1.5000
```

---

Make sure all the m-files you want to submit are in your Brown CS `~/course/cs004/homeworks/hw07` directory. Be sure to turn in ALL the files requested and that they are named exactly as specified, including spelling and case. When you're ready to submit the files, run

```
cs4_handin hw07
```

using a Brown CS account terminal window. The entire contents of your `~/course/cs004/homeworks/hw07` directory will be handed in.

Please check that you receive an email confirming that your submission was successful. If you do not receive an email and have continued issues with the handin script, please contact the HTAs.