Homework 9

Due 4:00 PM, Wednesday, April 17, 2019

Introduction

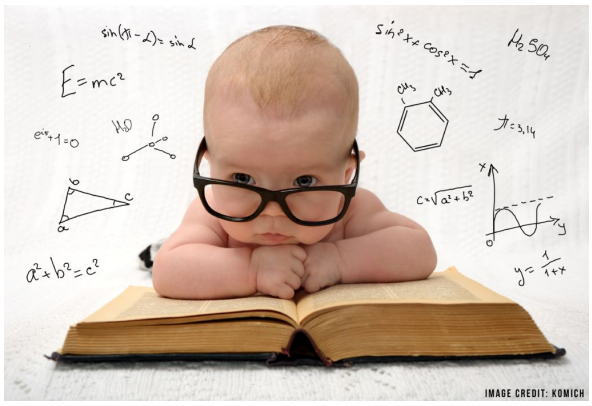
Installation and Handin

Part I: Iteration

- Problem 9.1a) Even and odd entries (15)
- Problem 9.1b) Compound Interest (15)
- Problem 9.1c) Prime Fibonacci numbers (20)
- Problem 9.1d) Binary Search (30)
- Problem 9.1e) Power play (10)
- Problem 9.1f) Stochastic Fractal (10)

Part II: Linear Algebra

- Problem 9.2a) Matrix multiplication is distributive (15)
- Problem 9.2b) Orthogonal vectors (10)
- Problem 9.2c) Linearly independent columns (15)
- Problem 9.2d) Determinants (20)
- Problem 9.2e) Linear systems (20)
- Problem 9.2f) Network flow (20)



VS.

Introduction

Professor Doggo, a famous mathematician, has been recruited by the puppies to challenge the babies with a series of difficult math, graphing, and linear algebra questions that will leave them dazed and confused. In an attempt to thwart the plan, the babies have hired you to help them

solve the questions using the powers of MATLAB! This homework is split into two parts. Part I covers concepts of iteration in MATLAB, while Part II tests your understanding of technical material related to the linear algebra concepts learned in lecture.

Please feel free to post to Piazza for additional support.

Installation and Handin

Homework Setup. For this homework assignment, there is no stencil code you need to install. That said, we recommend you create a `hw09` folder inside of your `~/course/cs0040/homeworks` directory to keep your scripts organized.

To start, create a script inside your `hw09` directory called **`cs4hw09.m`** to contain your answers and comprehensive test cases. Use the following publish-friendly format to organize your work in **`cs4hw09.m`**:

```
%% Question 9.1a
% 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 9.1b
...
```

Homework hand-in. 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 hw09
```

from a Brown CS Terminal window from your `~/course/cs004/homeworks/hw09` directory. The entire contents of `~/course/cs004/homeworks/hw09` will be handed in. Check for a confirmation email to ensure that your assignment was correctly submitted using the `cs4_handin` command. You can resubmit this assignment using the `cs4_handin` command at any time, but be careful, as only your most recent submission will be graded.

Part I: Iteration

Use `for` and `while` loops where needed to solve the problems in Part I.

Problem 9.1a) Even and odd entries (15)

Write a function with header `[Q] = myOddEven(M)` where, for each element, `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 nonempty, matrix of doubles. Do not assume `M` will always be square.

Example Output:

```
myOddEven([3 4; 6 7])
ans =
      3 -4
      -6 -7
```

Problem 9.1b) Compound Interest (15)

An interest rate, *i*, on a principal amount P_0, defines a payment amount for using 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.

Examples:

```
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
```

Problem 9.1c) 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. Remember, in the

Fibonacci sequence, the next number is found by adding the previous two numbers (0,1,1,2,3,5…) Assume N is a non-negative integer. Do not use the recursive implementation of Fibonacci numbers. Use an iterative approach.

Hints: 0 nor 1 is prime. You are welcome to use the MATLAB `isprime` function, or write your own function.

Examples:

```
myNFibPrimes(3)
ans =
     2 3 5
myNFibPrimes(8)
ans =
     2 3 5 13 89 233 1597 28657
```

Problem 9.1d) Binary Search (30)

Bisection is called binary search when it is applied to discrete values. Create a function that 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 function

```
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
```

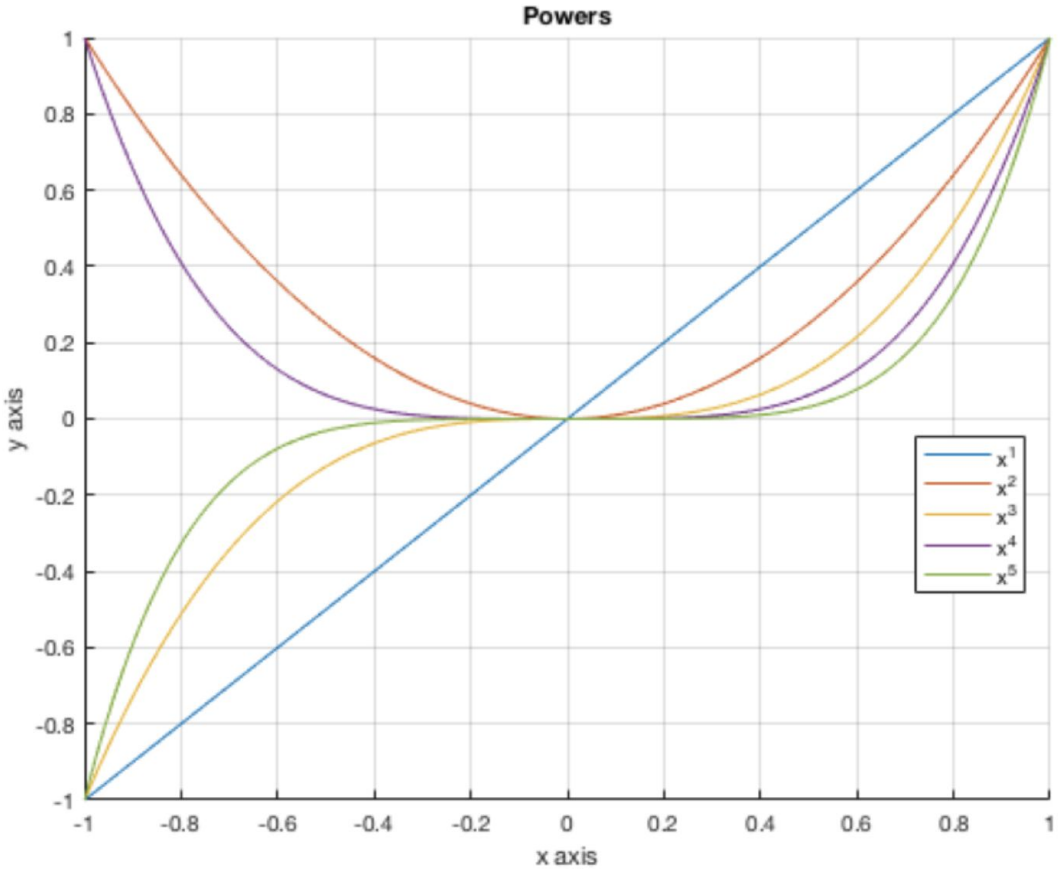
Problem 9.1e) Power play (10)

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, a legend, and grid lines**. Use `doc legend` to learn how to control the location of a legend. **Hint:** Use a 'cell array' 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`.

Ensure your graph matches the following (it's printing out the legend labels on each iteration):

```
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'
```



Part II: Linear Algebra

Problem 9.2a) Matrix multiplication is distributive (15)

Show that matrix multiplication distributes over matrix addition. Essentially, provide a proof showing that $A(B+C) = AB + AC$ for any given set of matrices `A`, `B`, and `C` that are of compatible dimensions.

You may include your answer in comments within your **cs4hw09.m** file, or take a photo or screenshot of your solution, and save it in your **hw09** submission folder as **distributive.jpg**.

Problem 9.2b) Orthogonal vectors (10)

Write a function with the header

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

where `v1` and `v2` are nonempty, nonzero vectors of the same size and `tol` is a positive scalar value. The output argument, `out`, should return `true`, if the angle `th` between `v1` and `v2` is within `tol` of the value $\pi/2$ (meaning **`|π/2 - th| < tol`**). Otherwise, the output argument, `out`, should return `false`.

Hint: Solve for `th` using the dot product equation, $x \bullet y = ||x||\,||y|| * cos(\theta)$, where `th` is θ.

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
```

Problem 9.2c) 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 matrix `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 by adding one column of `A` to `B` (i.e. `[B,A(:,i)]` where `i` is the column index) and using `rank([B,A(:,i)])` to test whether the column `A(:,i)` should be added to your eventual output `B`. Think about what `rank([B,A(:,i)]` should be equal to in order to warrant being linearly independent.

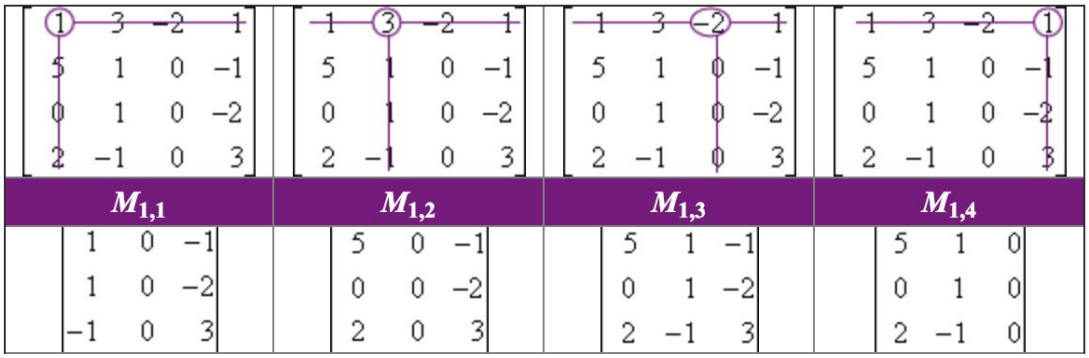
Example Output:

```
Example 1)
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

Example 2)
A = [4 -3 1 7 8; 1 0 -2 9 4; -2 1 1 0 3]
myMakeLinInd(A)
A =
     4  -3  1  7  8
     1  0  -2  9  4
    -2  1  1  0  3
ans =
     4  -3  7
     1  0  9
    -2  1  0
```

Problem 9.2d) 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 i^{th} and all the columns of `A` except the j^{th}. $A^{\{i,j\}}$ is used to denote the minors of `A`.

Examples of minors are as follows:



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 determinant equation can be rewritten as:

$$det(A) = A_{i,j} * C_{i,j}$$

Write a function with the signature below that returns the determinant of A. Assume A is a square matrix.

```
function [d] = myRecDet(A)
```

Hint: Since the formula is recursive, you should use recursion!

NOTE: `myRecDet` should use the above formula to compute the determinant; you should **NOT** use MATLAB's function `det`.

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
```

Problem 9.2e) Linear systems (20)

Write a function with header:

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

where `A` is a matrix and `b` is a compatible-sized column vector. `N` is the number of solutions of the system, and `x` is a solution to the same system.

If there are no solutions to the system of equations (i.e. `Ax = b`), then `x` should be an empty matrix and `N` should be `0`. If there is one or an infinite number of solutions, then `myNumSols` should return one solution for `x`, using the methods described in lecture, and either `1` or `Inf` for `N`.

Assume that `b` is a column vector and that the number of elements in `b` is equal to the number of rows in the input matrix `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, if `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, i.e.

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

Remember to use norm when comparing vectors and matrices.

Example Outputs:

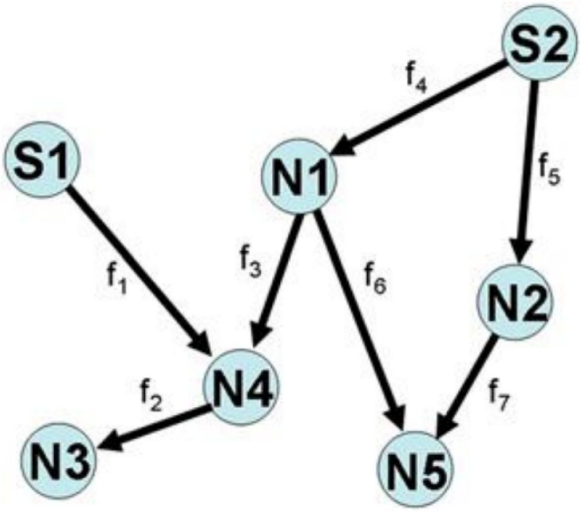
```
A = reshape(1:15, 3 ,5);
b = [-5; -4; -3];
[N, x] = myNumSols(A,b)
N = Inf

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
```

Problem 9.2f) 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. 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, $f_{in} = f_{out} + d_i$ (i.e. $f_1 + f_3 = d_4 + f_2$).

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` row vector denoting the flows in the network of supply stations and demands shown in the diagram above. The flows contained in f should satisfy all constraints of power generation and demands of the given system. 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. (You might want to use your solution from the previous problem!)

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/hw08` 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 hw09
```

using a CIT computer terminal window. The entire contents of your `~/course/cs004/homeworks/hw09` 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.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this document or have any concerns about this and any other CS4 document by posting on Piazza or filling out our anonymous feedback form.