MATLAB Project 3

Matrix Algebra & m-Files

1 Introduction

Once again, this project will be consist of two parts: the first will give you some experience working with matrix operations in MATLAB – you will see a lot of the Chapter 2 material here; the second will be your first introduction to actual programming within the MATLAB environment; m-files and some basic programming concepts will be covered.

2 Obligatory Tasks

As with your last projects, you will need to open your favorite word processor (like Microsoft Word) so that you can complete the Exercises listed in this project. Be sure to include your full name at the top of your document!

If you choose to submit your project electronically, you must using the following naming convention (points WILL BE DEDUCTED if you fail to do so!)

`Lastname_Firstname_MATLAB_Project3.docx`

(or whatever file extension you chose to use).

3 Exercise Key

The several icons will appear next to each Exercise you are asked to complete. Use the below table to determine what information should be included in your submitted document to fully complete the exercise.

Icon:	Item:	Comments:
	Commands entered in MATLAB & resulting output	You should copy relevant input and output from MATLAB and paste it into your document. You need only include commands that worked.
	Plots & Graphs	Include all graphs generated in an exercise unless the problem specifically tells you which/how many to include.
	Full sentence response	Each exercise contains a question that you should use at least one or two complete sentences to answer. Even if you're stuck, write down any reasoning or ideas you've had.
	Requires work by hand	Do scratch work by hand. Leave space in your document and write your scratch work directly on the assignment to turn in.

4 Standard Matrix Operations

We briefly worked with some operations on vectors in Project 2 and, luckily for us, all of those operations work exactly the same for matrices (because, after all, vectors are just special examples of matrices).

If A and B are matrices of appropriate size then we have the typical commands:

- `>> A+B`

  Sum of two matrices.

- `>> A-B`

  Difference of two matrices.

- `>> c*A`

  Scalar multiple of the matrix A (where c is any scalar).

- `>> A^n`

  Raises A to the nth power (note A must be square!).

- `>> A'`

  Transpose of A (i.e., use a ' for transpose rather than a ^{T}).

Exercise 1. (a) Define the following matrices in MATLAB:

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 1 & 2 \\ 0 & 2 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 & 3 \\ 1 & 5 & 1 \\ 1 & 1 & 3 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & -1 & 4 \\ 0 & 2 & -1 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

(b) Compute the expression

$$D = (2A^2B + 3A^T)^2$$

(c) Compute the expressions

$$E = AC \quad \text{and} \quad F = CA$$

We see that we cannot compute the product AC, but could we compute AC^T? What about C^TA?

(d) Compute $D\mathbf{x}$.

5 Matrix Inversion

We have seen in class that inverting matrices isn't as simple as inverting numbers where we can simply use a $^{-1}$. Thus, rather than trying to use $^{-1}$ to compute the inverse of a matrix, MATLAB uses the `inv` command.

If you want to find the inverse of a square matrix A simply execute the command

```
>> inv(A)
```

and MATLAB will find the inverse of A – if one exists!

Exercise 2. **(a)** Try using the `inv` command to find the inverse of the matrix

$$\begin{pmatrix} 1 & 1 \\ 100 & 100 \end{pmatrix}$$

Why does MATLAB give this strange answer?

(b) Enter the following matrix into MATLAB

$$A = \begin{pmatrix} 4 & 9 \\ 5 & 11 \end{pmatrix}.$$

Let's find the inverse of this matrix

```
>> B=inv(A)
```

and check that it satisfies the requirements for being an inverse. Compute AB and BA, what do these products equal?

(c) Enter the following vector into MATLAB

$$\mathbf{x} = \begin{pmatrix} 5 \\ 10 \end{pmatrix}.$$

Compute the product Ax, and store the result into the variable $\mathbf{y}$ in MATLAB.

(d) Without performing any computations, what will the product $B\mathbf{y}$ equal?

6 Application of Matrix Operations

Consider the following problem: Suppose we have six cities with airports, say Dan Diego, San Francisco, Chicago, New York, Moscow, and Tokyo. We are interested in counting the number of ways we can travel from one city to another with at most n stopovers. Suppose for example that there are direct flights from

- San Diego to San Francisco
- San Francisco to any of the remaining five cities
- Chicago to San Francisco, New York, and Moscow
- New York to San Diego, San Francisco, Chicago, and Moscow
- Moscow to Chicago, New York, and Tokyo
- Tokyo to San Francisco, New York, and Moscow

and we want to count the number of ways we can get from San Diego to Moscow with at most 3 stopovers. For example, the trip San Diego $\mapsto$ San Francisco $\mapsto$ Tokyo $\mapsto$ New York $\mapsto$ Moscow is a trip with 3 stopovers (we are stopping over in San Francisco, Tokyo, and New York).

At first glance this problem has little to do with matrices and linear algebra, but it turns out there is a very easy and elegant way of solving problems of this type using *incidence matrices*.

The first step is to number our cities in the order they are listed: San Diego will be 1, San Francisco is 2, and so on. Then, let us create our incidence matrix A by the following rule: if we can get from the ith city to the jth city, then the entry A_{ij} of the matrix will be set to 1, otherwise we set that entry to 0. By convention we set all of the diagonal entries, A_{ii} to 0, since you cannot take a flight from a city to itself. Thus, we obtain

the following matrix A for our situation:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

We can now find a useful interpretation for the entries of the matrix A^2. Take, for example:

$$(A^2)_{36} = A_{31}A_{16} + A_{32}A_{26} + A_{33}A_{36} + A_{34}A_{46} + A_{35}A_{56} + A_{36}A_{66},$$

(note that the above equality is simply the result of compute $A * A$ and looking at what the entry in the 3rd row and 6th column is). Observe that any of the terms $A_{3k}A_{k6} = 1$ if and only if both A_{3k} and A_{k6} equal 1; that is, if and only if we can fly from Chicago (the 3rd city) to the kth city, and then from the kth city to Tokyo (the 6th city). Thus, $(A^2)_{36}$ gives us the number of ways of flying from Chicago to Tokyo with exactly one stopover. Similarly, one can show that the number of ways of flying from city i to city j with *exactly* n stopovers is $(A^{n+1})_{ij}$.

	**Exercise 3. (a)** Enter the matrix A given above into MATLAB. State the number of ways to get from San Diego to Moscow with exactly 3 stopovers.
	**(b)** Find the number of ways to get from San Francisco to Chicago with *at most* 4 stopovers. (Note, this is not the same as finding the number of ways with *exactly* 4 stopovers!)

Warning! As you have probably realized, the method described above includes silly trips such at San Francisco $\mapsto$ New York $\mapsto$ San Francisco $\mapsto$ New York as a trip with 2 stopovers, so, while the values given in your matrix for Exercise 3 are correct, they do contain nonsensical traveling options.

7 m-files

An m-file, or script file, is a simple text file where you can place your MATLAB commands. When the file is run (much like when an program is run), MATLAB reads the commands and then executes them exactly as if you had typed them in one-by-one to the command prompt. There are several ways of creating an m-file – the easiest being from within MATLAB, the process for some versions may vary, but for R2015a simply click "New" in the upper left corner (if you click the drop down option on the "New" button, just select "Script"). You can then input all of your commands into the new script file that has opened and save the file by clicking the "Save" button. You can also create an m-file using any text editor, just be sure to save your file as "filename.m", rather than ".txt"!.

Create a new m-file and enter the following lines of code:

```
A = [1,2,3;-2,1,0;4,5,-1]
rref(A)
```

Save your m-file as "example.m" and then go to your Command Window in MATLAB. Rather than typing commands, simply enter the name of your m-file

```
>> example     %do not enter the ".m"
```

You should see the matrix A that you created in the m-file as well as its reduced row echelon form.

When using m-files, all of the variables that you set within your m-file can be used just as if you typed them in the command window once you execute the file. (One exception to this is when your m-file defines a function to be used by MATLAB – more on these in the next project!)

One of the main advantages to using m-files is that you can easily make changes to your inputs – editing one line out of 100 is much easier in a saved m-file than it would be to reenter those 100 lines into the command window one at a time! Along those same lines, m-files give you the ability to write some complex programs (and functions) that can be executed by MATLAB.

8 If-Then Statements

There are two basic tools that will most likely play an integral part in almost any MATLAB program you write: if-then statements, and for-loops. If-then statements tell MATLAB when (or when not) to perform some action. The most basic syntax for an if-then in MATLAB looks like:

```
if *expression*
      *statements*
end
```

If the `*expression*` is true then MATLAB will execute whatever your indicated `*statements*` are. For instance, create a new m-file titled "if-then-example.m" and enter the following

```
A=rand()*10;  %rand() returns a random number between 0 and 1.
if A>5
    disp('A is greater than 5!')
end
```

If you execute "if-then-example" in the command window several times you will see that sometimes you receive the message `A is greater than 5!`, while others you don't. It all depends on if the random number (that is assigned each time the m-file is executed) is large enough.

We can add a bit of code to our if-then statement to tell MATLAB what to do if the `*expression*` turns out to be false with an `else` statement:

```
if *expression*
     *statements*
else
     *statements*
end
```

Adding the else statement to our m-file we have

```
A=rand()*10;  %rand() returns a random number between 0 and 1.
if A>5
    disp('A is greater than 5!')
else
    disp('A is less than or equal to 5!)
end
```

Run the m-file several times – you should now always receive a message from MATLAB based on the size of `A`.

Generally, when writing your `*expression*` for the if-then statement, you will use one of MATLAB's built in relational operators, $<, >, <=, >=, ==, \sim=$, where the last two correspond to "equal to" and "not equal to", respectively. There is also an `elseif` statement that can be included in any if-then statement, but we will omit its usage here (it shouldn't be too difficult for you to figure out anyway!).

9 For-Loops

For statements loop (continue to execute) a specific number of times, and keep track of each iteration with an incrementing index variable. The colon notation we saw in Project 2 plays an essential role here. Recall the colon notation used the convention: `Starting Value:Increment Value:Ending Value`, and the increment value may be omitted if its value is 1. For example, `1:3:10` would mean we start at 1, increment by 3 and end once we reach 10 – 1, 4, 7, 10.

Now that we remember how to work with the colon notation, the easiest way to learn about for-loops is simply to see one. Create a new m-file titled "for-loop-example.m" and enter the following:

```
for n = 1:3:10
    disp(n)
end
```

Execute this m-file and you should obtain the sequence of numbers given in the colon notation example.

For-loops are great at computing series. For example, we can compute

$$\sum_{n=1}^{100} 2n - 1$$

by entering the following into our m-file:

```
P=0;   %P will be the value of our partial sum, we will add to it each iteration
for n = 1:100
     P=P+(2*n-1); %We add to the old value of P the new quantity 2n-1 and store that back in P
end
disp(P)
```

The use of semicolons becomes very important when dealing with for-loops. Try deleting the semicolon from the line `P=P+(2*n-1);` and see what happens – suppressing output is a nice feature isn't it?!

	**Exercise 4. (a)** Create an m-file in which you define a 2×2 matrix and use an if-then statement to display a message stating whether or not the matrix is invertible. Include the contents of your m-file as well as the output given by MATLAB in your writeup.
	**(b)** Use a for-loop to compute the value of the following series: $$\sum_{n=3}^{32} \sqrt{n}\left(1+\frac{1}{n}\right)$$ Include the contents of your m-file as well as the output given by MATLAB in your writeup.