MATLAB Project 4

Determinants, Eigenvalues, Diagonalization & Functions

1 Introduction

The first portion of this project will parallel what we have been learning in class, namely, Determinants (Ch. 3), Eigenvalues (Sections 5.1-5.2) and Diagonalization (Section 5.3). This portion should be brief as MATLAB has built in functions to compute all of these concepts automatically!

The second portion will provide a basic introduction to defining your own functions within MATLAB. All of our work here will be done within an m-file, then we will run our m-file from the MATLAB command prompt to access our function. The combination of m-files and writing your own functions is key to doing any real (computational) mathematics in MATLAB – without them you're merely performing a few computations.

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_Project4.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 Determinants

We have seen how the method of cofactor expansion can be used to compute the determinant of any $n \times n$ matrix. We have also seen how annoying this can be when n starts becoming large (you did a 5×5 by hand, imagine trying to do a 10×10, and even then $n = 10$ isn't *that* large). Lucky for us, MATLAB has a built in function to compute the determinant of any $n \times n$ matrix.

Define the following three matrices in MATLAB (you'll need the first two for the exercise too!)

$$A = \begin{pmatrix} 8 & 11 & 2 & 8 \\ 0 & -7 & 2 & -1 \\ -3 & -7 & 2 & 1 \\ 1 & 1 & 2 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & -2 & 0 & 5 \\ 0 & 7 & 1 & 5 \\ 0 & 4 & 4 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & -1 & 2 & -1 & 3 \\ 2 & 1 & 0 & 4 & -1 \\ 1 & -2 & 1 & 2 & -1 \\ -3 & 0 & 2 & 4 & 2 \\ 2 & -2 & 3 & 4 & 2 \end{pmatrix}.$$

The function to compute the determinant of a matrix in MATLAB is simply `det()`. For instance, try computing

- `>> det(A)`
- `>> det(B)`
- `>> det(C)`

(Do C and $\det(C)$ look familiar to you? Wasn't this much easier than cofactor expansion?!)

	**Exercise 1. (a)** Use MATLAB to compute the determinants of the following matrices: $$A + B, A - B, AB, A^{-1}, B^T$$
	**(b)** Which of the matrices from part (a) are **not** invertible? Explain.
	**(c)** Before this exercise you computed $\det(A)$ and $\det(B)$. Suppose that you forgot what the entries of A and B were, but you still remembered the value of their determinants. Which of the determinants in part (a) would you still be able to compute by hand, even without knowing the entries of A and B. Explain.

5 Eigenvalues and Eigenvectors

Recall that given a matrix A an eigenvalue of A is a number λ such that $A\mathbf{v} = \lambda\mathbf{v}$ for some nonzero vector $\mathbf{v}$, called the eigenvector corresponding to λ. While it isn't difficult to compute the eigenvalues and eigenvectors of small matrices, it can become impossible to do by hand for large ones (remember, a $n \times n$ matrix has an nth degree characteristic equation that cannot generally be solved when $n \geq 5$). Lucky for us again, MATLAB can compute both eigenvalues and eigenvectors! The command `eig()` does the work for us.

Still using the matrix B from the previous section,

$$B = \begin{pmatrix} 1 & -2 & 0 & 5 \\ 0 & 7 & 1 & 5 \\ 0 & 4 & 4 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

we can find the eigenvalues of B:

```
>> b = eig(B)

   b =
       1
       8
       3
       2
```

Hence, the eigenvalues are $1, 8, 3, 2$. (As a side note, recall from class that the value of the determinant is also given by the product of the eigenvalues – if you multiply these eigenvalues together you should get the same value you did in the previous section!)

Determining the eigenvectors along with the eigenvalues is extremely easy – we use the exact same function! There is only one small change:

```
>> [P,D] = eig(B)

    P =
        1.0000   -0.1980    0.2357    0.9074
        0         0.6931   -0.2357   -0.1815
        0         0.6931    0.9428    0.3630
        0         0         0         0.1089

    D =
        1 0 0 0
        0 8 0 0
        0 0 3 0
        0 0 0 2
```

MATLAB returns the matrix P consisting of the eigenvectors of B as its columns and a diagonal matrix D with the corresponding eigenvalues along the diagonal.

Let's check that these quantities truly are eigenvalues/eigenvectors by selecting the second eigenvector and second eigenvalue from P and D, respectively:

```
>> v = P(:,2);  %This stores the 2nd column of P in the variable v
>> lambda = D(2,2); %This stores the 2nd entry on the diagonal of D in lambda
```

Now compute

```
>> B*v - lambda*v
```

you should obtain 0 since $B\mathbf{v} = \text{lambda} \cdot \mathbf{v}$.

Exercise 2. **(a)** Enter the following matrix V into MATLAB:

$$V = \begin{pmatrix} 9 & -4 & -2 & 0 \\ -56 & 32 & -28 & 44 \\ -14 & -14 & 6 & -14 \\ 42 & -33 & 21 & -45 \end{pmatrix}.$$

Use MATLAB to compute the eigenvalues and eigenvectors of V.

(b) Only by looking at the eigenvalues determine whether V is invertible. Explain your reasoning.

6 Diagonalization

Recall that a matrix A is called diagonalizable if we can find an invertible matrix P such that $A = PDP^{-1}$ where D is a diagonal matrix. (Note, this is equivalent to saying that $P^{-1}AP$ is a diagonal matrix.)

We also seen that not every matrix is diagonalizable. Recall that to diagonalize an $n \times n$ matrix A we must find a basis of $\mathbb{R}^n$ consisting of eigenvectors of A. Then, forming a matrix P whose columns are the elements of this basis, we obtain $A = PDP^{-1}$ where D is the diagonal matrix whose entries on the diagonal are the eigenvalues of A corresponding to the eigenvectors in the respective columns of P. If $\mathbb{R}^n$ does not possess such a basis of eigenvectors then A is not diagonalizable.

Recall the following theorem we learned in class:
Theorem: If an $n \times n$ matrix has n distinct eigenvalues then the matrix is diagonalizable.

Note, however, if the matrix does not have n distinct eigenvalues this theorem does not give us any information! Other methods must be used to determine if the matrix is diagonalizable.

Exercise 3. (a) Let

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

Use MATLAB to find an invertible matrix P and a diagonal matrix D such that $F = PDP^{-1}$.

(b) Use MATLAB to compare F^{10} and $PD^{10}P^{-1}$.


(c) Let

$$\mathbf{f} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Compute $F\mathbf{f}, F^2\mathbf{f}, F^3\mathbf{f}, F^4\mathbf{f}, F^5\mathbf{f}$. Describe the pattern you see.


(d) Given a sequence of numbers $\{1, 1, 2, 3, 5, 8, 13, \ldots\}$ where each term is the sum of the previous two, find the 30th term of this sequence. (See below for a hint if you are stuck.)

The sequence in the above exercise is called a Fibonacci sequence. It has many interesting properties and appears often in nature. The above exercise points in the direction of how to find an explicit formula for the nth term in this sequence (it is not obvious a priori that such a formula must even exist). To obtain this formula simply note that if we let

$$f_0 = f_1 = 1 \quad \text{and } f_{n+2} = f_{n+1} + f_n$$

be our Fibonacci sequence and let

$$\mathbf{f} = \begin{pmatrix} f_0 \\ f_1 \end{pmatrix},$$

then

$$F\mathbf{f} = \begin{pmatrix} f_1 \\ f_0 + f_1 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix},$$

also

$$F^2\mathbf{f} = F(F\mathbf{f}) = F\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} f_2 \\ f_1 + f_2 \end{pmatrix} = \begin{pmatrix} f_2 \\ f_3 \end{pmatrix},$$

and in general,

$$F^n\mathbf{f} = \begin{pmatrix} f_n \\ f_{n+1} \end{pmatrix}.$$

Thus, to get the general formula we need to compute F^n, (which is easily done by computing PD^nP^{-1}), multiply it by the vector $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and read off the first row of the resulting vector. If you like, you may perform these calculations by hand at your leisure and derive an interesting formula for the nth Fibonacci number involving the golden ratio.

7 Functions

A MATLAB function is actually a miniature program that you write within MATLAB. This program performs a user-defined sequence of operations specified in an m-file. A function accepts one or more MATLAB variables as its inputs, operates on them in some way, and then returns one or more MATLAB variables as outputs.

We have already seen how to create an m-file and use it to perform operations MATLAB, so we need some way to distinguish one of those m-files we've already seen to one of these new m-files containing a function. The way to do this (and the way that **every** m-file containing a function must begin) is to write the following in the **first line** of the m-file:

```
function [output_parameter_list] = function_name(input_parameter_list)
```

The first word must always be `function`, following that are the (optional) output parameters enclosed in square brackets (this may be omitted if there are no output parameters for your function). The `function_name` will be the name of your function, and the command that you type to call it. The `function_name` **must also be the same as the m-file name** (without the ".m" extension) in which the function is stored, e.g., if you want to define a function `foo` in MATLAB the m-file must be saved as `foo.m`. Following the file name is the (optional) `input_parameter_list`, these are the MATLAB variables which you going to be operating on. Both the `input_parameter_list` and `output_parameter_list` can contain more than one variable – they are comma separated lists of MATLAB variables.

We now provide two simple examples of MATLAB functions:

The function `traparea` below computes the area of a trapezoid given the dimensions a, b and h, where a and b are the lengths of the parallel sizes and h is the distance between these sides. We write the function in two different ways. The first has no output variable and the area is displayed upon its computation. The second has an output variable named `area` and the display of the area from its computation is suppressed (since we used a semicolon), now the area is displayed when the output variable is returned to the you.

m-file name: `traparea.m`

Contents of m-file:

```
function traparea(a,b,h)
    0.5*(a+b)*h
```

m-file name: `traparea.m`

Contents of m-file:

```
function area = traparea(a,b,h)
    0.5*(a+b)*h;
```

The second example converts given Cartesian coordinates to polar coordinates. Note we are using two output variable here:

m-file name: `cart2polar.m`

Contents of m-file:

```
function [r,theta] = cart2polar(x,y)
    r = sqrt(x^2 + y^2);
    theta = atan2(y,x);
```

To evaluate either of these functions you simply need to call the desired function. For instance,

```
>> traparea(2,1,1);
```

and

```
>> [a,b] = cart2polar(2,-2);
```

Note here that we used `[a,b]` when evaluating `cart2polar`, without it, the function only would have returned the value of the first output variable. So, anytime you have more than one output variable in your function, you must set your function equal to that number of variables when evaluating it!

Exercise 4. Using the concepts from this project as well as Project 3 (for-loops and if-then statements) create a function in MATLAB that takes as an input an arbitrary square matrix A and a scalar r and performs the following tasks:

1. Computes rA.
2. Determines whether A is invertible, and if so computes A^{-1}.
3. Computes the sum of r and all of the eigenvalues of A. (The command `length(B)` will tell you the number of elements in a vector, or the larger value between the number of rows and the number of columns in a matrix. This might come in handy here!)

Also, be sure that all of the matrices/values are displayed in MATLAB (i.e., don't suppress the output of rA, A^{-1} or the sum, print them to the screen).

If you cannot create *one* function/m-file that completes all of these tasks, it is OK to create a separate function/m-file for each task. (But try to avoid having to do this!)