Chapter 1

Getting Started

Where to use Matlab/Octave

Matlab physically resides on each of the computers in the Olin Hall labs. See your instructor if you need an account on these machines.

Personal Copy of Matlab

If you are going to go to graduate school at some point, or if you think Matlab might be something you will want a personal copy, Matlab is available to download from the Mathworks website. Its a good deal- $99 for Matlab and 10 of the most used toolboxes (some of which we don't have).

Free version is called Octave

There is a version of Matlab, called Octave, that is available on the internet via any web browser (search for Octave online). This is a very convenient option if you just have a few computations to make.

Lastly, Octave is also available for free to download to your home computer. As is often the case with open source software, the documentation can be hard to read and help can be hard to locate. In the past, Octave for the Mac has been tricky, so use it only if you're pretty comfortable tweaking a Mac. The Linux and Windows distributions are fairly reliable.

Of course, if you have any trouble putting Octave on your own machine, there's always the Olin Hall computer labs!

1.1 Setting up Matlab in Olin

When you first log in: Using a computer in the Math/CS lab, go to the upper left corner, and choose the top icon on the launch pad. You should get a search dialog, so you can type:

```
matlab
```

You will then get the Matlab icon, so select that and Matlab will start. To keep the Matlab icon on the launcher, go to the icon, right-click. In older Ubuntu versions, select "Lock to Launcher", in more recent versions, select something like "Copy to Favorites".

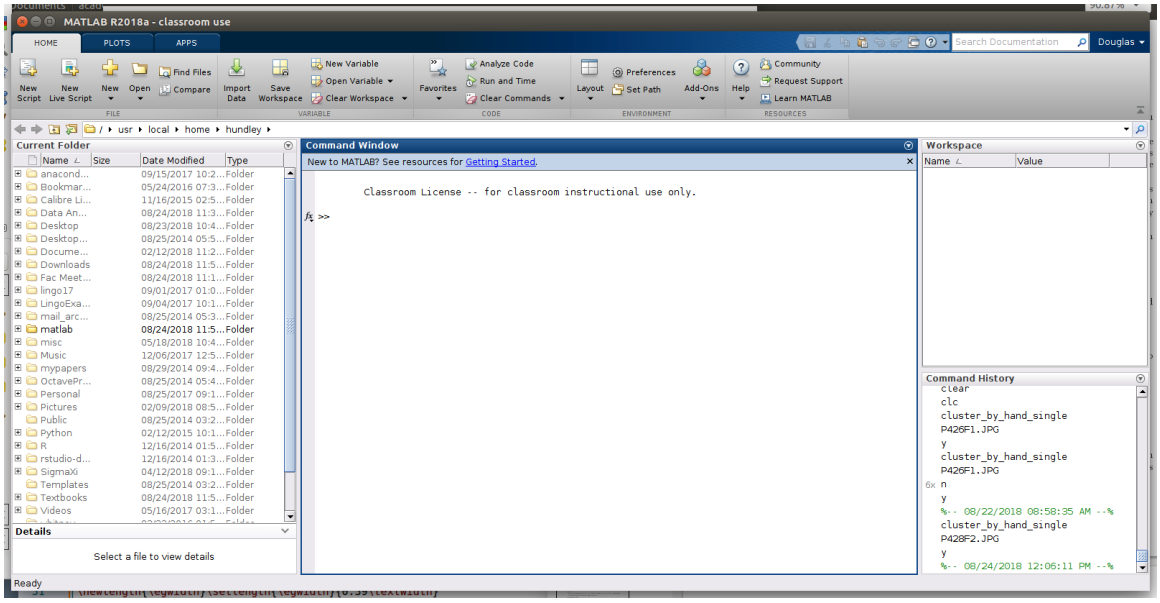

Figure 1.1: The opening interface for Matlab (2018). The middle window is the *command window*, and is where we type Matlab commands. The other windows are there to give us information about the Workspace, the current directory, and a history of commands. Yours should be basically blank at this point.

See Figure 1 to see the opening interface that you'll use. You might take a few minutes to just look around the interface- Try some different tabs and see what happens.

1.1.1 How does Matlab work?

You can make Matlab do computations three different ways:

- Type commands directly into the keyboard in the command window.
- Have your Matlab commands typed into a separate text file (called a **script file**), and then have Matlab read these commands in. This is very nice- it gives you documentation and allows you to run similar computations several times without having to re-type the commands. We will typically use script files to do homework problems.
- Define your own functions by typing a separate text file (called an **m-file**).

  Note that the difference between a function and a script is that the script simply executes the commands that are listed. A function takes values in and produces some output. Both scripts and functions end in `*.m`, but the very first line in a function is the word "function". We'll construct both scripts and functions below.

Saving your work

Matlab keeps track of your past history while you are typing on the keyboard, but for answering homework, it is best to use a *script file*, which is just a text file with Matlab commands on it. More on this later.

Matlab has a very nice text editor that you can use to type out and save Matlab functions and scripts- To access the editor, type `edit` in the Matlab command window.

For Octave online, we don't have the editor option, but you can use any text editor. If the problem is a short one, I may just ask you to submit a screenshot of your work.

1.2 Introductory Commands

On the left, we show the commands you can type in the command window, and to the right we give some commentary. Matlab uses the standard mathematical operations:

1.2.1 Numbers

```
2+3;
123.3*sin(3.2)
```

If you leave the semicolon off the end of the line, Matlab prints the result (this is OK when typing live, but usually not good for scripts- We'll see that later).

```
2^5;
exp(4);
cos(1.3);
log(3)
```

The function `exp(x)` is used for e^x. Sine and cosine assume the input is in radians. The logarithm assumes natural log.

```
i^2
(1+i)*(2-i)
pi
exp(1)
```

The number $i = \sqrt{-1}$ is defined in Matlab- But only if it hasn't been defined by you to be anything else. The constant π uses lowercase p (unlike Maple). To get the constant e, use `exp(1)`.

```
5/0
0/0
eps
```

In the first case, you'll see `Inf`, which represents "infinity". This is actually incorrect- It should be $\pm\infty$, since we don't know how the denominator is reaching zero. In the second case, Matlab gives `NaN`, which means "Not a Number". Do you know why this would be? (think limits). The last value is a very small number below which we typically would think of a quantity as "zero".

1.2.2 Helpful Administrative Commands

The following commands are useful as you begin to use Matlab more and more:

```
clear;
clc;
whos;
help
% Help for a specific command
help sin
doc
demo
```

The up arrow key is useful (try it!). Here we first clear the workspace memory of all variables, and secondly we clear the command window. Give the others a try to see what they do. Notice that we can use a percent sign to put in a *comment* (Comments are not processed by Matlab).

1.3 Assigning values to variables

To assign data to a variable name, we would use the equal sign. For example, $x = 3$ assigns the value of 3 to the variable x. The array (or matrix) is the basic data type in Matlab. Data entry is straightforward.

```
A=[1 2 3 4; 5 6 7 8];
A=[1 2 3 4
5 6 7 8];
A(2,3)
```

A is an array with two rows and 4 columns and can be entered in either manner. Inside an array, the semicolon indicates the end of a row. The $(2, 3)$ element (2d row, 3d column) of A is accessed as this.

```
x=[1;0;1]; y=[-1;2;3];
dot(x,y)
cross(x,y)
```

The dot product and cross product are available.

The dot

If we want to take an array of numbers and perform some operation to every value, we can use the dot. Here are some examples of how this works:

```
x=[1,4,7,10];
x.^2
sin(x)
y=[2 3 4 5];
x./y
```

The array x is first formed, then we take the square of each element. Similarly, the third line takes the sine of each element. If we define y as another array of the same size, we can "divide" x by y, where the first element will be $1/2 = 0.5$ and the last element is $10/5 = 2$.

Special Commands: The colon operator

- `a:b`

  Produces a vector of integers from a to b in a row.

- `a:b:c`

  Produces the numbers from a to c by adding b each time" $a, a + b, a + 2b, \ldots$, up until the last number is biggest that is less than or equal to c.

- `linspace(a,b,c)`

  Produces c numbers evenly spaced from the number a to the number b (inclusive).

```
x=2:9;
y=1:2:8
z=10:-1.3:3
w=linspace(2,5,40);
w=linspace(2,5)
```

First, x is an array with the integers from 2 to 9. Secondly, y is array with integers $1, 3, 5, 7$. Can you predict what z will be? The last line produces 40 numbers evenly spaced beginning with 2 and ending with 5. If we leave the third number off, we get 100 values as the default.

Matlab commands associated with Arrays

- Random arrays (handy if you just need some quick data!)

  `A=rand(m,n)` Produces an $m \times n$ array of random numbers (uniformly distributed) between 0 and 1. If you just want a single random number between 0 and 1, just type `rand`

  `A=randn(m,n)` produces an $m \times n$ array of random numbers (with a normal distribution) with zero mean and unit variance. If you want a single random number (with a normal distribution), just type `randn`

- `A=zeros(m,n)` Produces an $m \times n$ array of zeros.

- `A=ones(m,n)` Produces an $m \times n$ array of ones.

- `A=eye(n)` Produces an $n \times n$ identity matrix.

- `A=repmat(B,m,n)` Matrix A is constructed from matrix (or vector) B by replicating B m times down and n times across.

  Example: Let $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$. Then `A=repmat(B,2,3)` creates the array:

```
A =
     1     2     1     2     1     2
     3     4     3     4     3     4
     1     2     1     2     1     2
     3     4     3     4     3     4
```

Matrix Arithmetic

- Transposition is denoted by the single quote character `'`. That is, `A'` $= A^T$. (CAUTION: If A contains complex numbers, then `A'` is the *conjugate transpose* of A, sometimes denoted as $A^* = \bar{A^T}$)

- Matrix addition and subtraction is performed automatically and is only defined for matrices of the same size.

- Scalar addition. If we want to add a constant c to every item in an array A, type: `A+c`

- Scalar Multiplication: We can multiply every number in the array by a constant: If A is the array and c is the constant, we would write: `B=c*A`

- Matrix Multiplication: Use the regular multiplication sign for standard matrix multiplication. If A is $m \times n$ and B is $n \times p$, then `A*B` is an $m \times p$ matrix, as we did in linear algebra.

- Elementwise Multiplication. We can multiply and divide the elements of an array A and an array B *elementwise* by `A.*B` and `A./B`

  Exponentiation is done in a similar way. To square every element of an array A, we would write: `A.^2` This is the same as saying `A.*A`

- Functions applied to arrays: Matlab will automatically apply a given function to each element of the array. For example, `sin(A)` will apply the sine function to each element of the array A, and `exp(A)` will apply e^x to each element of the array. If you write your own functions, you should always decide ahead of time how you want the function to operate on a matrix.

1.4 More on Arrays

- Random arrays (handy if you just need some quick data!)

  `A=rand(m,n)` Produces an $m \times n$ array of random numbers (uniformly distributed) between 0 and 1. If you just want a single random number between 0 and 1, just type `rand`

  `A=randn(m,n)` produces an $m \times n$ array of random numbers (with a normal distribution) with zero mean and unit variance. If you want a single random number (with a normal distribution), just type `randn`

- `A=zeros(m,n)` Produces an $m \times n$ array of zeros.

- `A=ones(m,n)` Produces an $m \times n$ array of ones.

- `A=eye(n)` Produces an $n \times n$ identity matrix.

- `A=repmat(B,m,n)` Matrix A is constructed from matrix (or vector) B by replicating B m times down and n times across.

  Example: Let $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$. Then `A=repmat(B,2,3)` creates the array:

```
A =
     1     2     1     2     1     2
     3     4     3     4     3     4
     1     2     1     2     1     2
     3     4     3     4     3     4
```

Accessing Submatrices

Let A be an $m \times n$ array of numbers. Then:

The notation:	Yields:
`A(i,j)`	The (i,j)th element
`A(i,:)`	The entire ith row
`A(:,j)`	The entire jth column
`A(:,2:5)`	The 2d to fifth columns, all rows
`A(1:4,2:3)`	A 4×2 submatrix

Example: What kind of an array would the following command produce?

```
A([1,3,6],[2,5])
```

A 3×2 matrix consisting of the elements:

$$\begin{matrix} A(1,2) & A(1,5) \\ A(3,2) & A(3,5) \\ A(6,2) & A(6,5) \end{matrix}$$

Example: Create a 5×5 zero array, and change it to:

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Answer:

```
A=zeros(5,5);  %Create the matrix of zeros
b=[1 2 3;4 5 6; 7 8 9];
A(2:4,2:4)=b;
```

Note also the use of the `%` sign. It is used to denote *comments*; that is, Matlab would ignore everything on the same line after the `%` sign.

Adding/Deleting Columns and Rows:

Its straightforward to insert and/or delete rows and columns into a matrix. Before doing it, we define `[ ]` as "the empty array": the array with nothing in it.

In the following, let A be a 4×5 array, let b be a 1×5 row, and c be a 4×1 column.

Examples of use (each of these are independent from the previous):

The command:	Produces:
`A(1,:)=[];`	Delete the first row.
`A([1,3],:)=[];`	Delete rows 1 and 3.
`A(:,3)=[];`	Delete column 3.
`A(:,1:2:5)=[];`	Delete the odd columns.
`A(1,:)=b;`	Put **b** as row 1.
`A(:,6)=c;`	Add c as the last column.
`d=[c , A(:,1:3)];`	d is c and columns $1-3$ of A.
`A=[A(:,1), c, A(:,2:5)];`	Insert c as column 2 of A, others shift 1 over.
`A=[A(1,:); b; A(2:4,:)];`	Insert b as row 2 of A, others shifted 1 down.

Example: Matlab comes with some built-in data sets. One such set is the image of a clown. For fun, we'll load the array in, display it, then we'll remove all of the even rows and columns, then re-display it:

```
X=load clown.mat
whos
X(4:6, 6:10) %Look at some of the values in the array X.
```

```
image(X);
colormap(map);
X(2:2:200,:)=[];
X(:,2:2:320)=[];
image(X);
```

Once you're done, you may want to clear the memory and the screen:

```
clear
clc
```

If you want to re-do the clown again, you do not need to retype it! Use the up-arrow key to bring back the commands you typed. You can also type the first few characters, then use the up-arrow key:

```
X=(up arrow)
```

Solving $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$

To solve $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$, Matlab has two basic commands: `x=A\b` or `x=pinv(A)*b`. The command `pinv(A)` computes the *pseudoinverse* of A, which we will discuss later in the section dealing with the Singular Value Decomposition.

In linear algebra, there were three possible outcomes for solving $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$. They were:

1. A unique solution.
2. An infinite number of solutions.
3. No solution.

Matlab will always give exactly one solution. We need to interpret that solution in the second two cases. In the case of an infinite number of solutions (we have free variables in this case, also called *an underdetermined system*), the two methods may give different answers:

- `x=A\b` provides the most zeros in $\mathbf{x}$.
- `x=pinv(A)*b` gives $\mathbf{x}$ with the smallest norm.

Example: Let $A = \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix}$, with $b = \begin{bmatrix} 9 \\ 3 \end{bmatrix}$. Then the full solution is:

$$\mathbf{x} = \begin{bmatrix} 9+2t \\ 3-t \\ t \end{bmatrix} \tag{1.1}$$

In Matlab, the result of typing `x=A\b` is $x = [0, -15/2, -9/2]^T$ and the result of typing `x=pinv(A)*b` is $x = [4, 11/2, -5/2]$.

(MATLAB HINT: You can get Matlab to return numbers as fractions by typing `format rat`)

In the case of an overdetermined system (a system with no solution), Matlab will automatically return the *least squares* solution- that is, the answer $\mathbf{x}^*$ will be the minimum of $\|A\mathbf{x} - \mathbf{b}\|$:

$$\|A\mathbf{x}^* - \mathbf{b}\| \leq \|A\mathbf{x} - \mathbf{b}\|, \text{ for all } \mathbf{x}$$

In general, you should always use the forward slash (for help, type `help slash`): `x=A\b` which automatically determines a best numerical method. That is, sometimes its best (numerically) not to explicitly compute the pseudoinverse first.

1.5 Exercise Set I

1. What is the Matlab command to create the array x which holds the integers: $2, 5, 8, 11, \ldots 89$

2. (Referring to the array above) What would the Matlab command be that zeros out the even-numbered indices (That is, $x(2), x(4), x(6), \ldots$)?

3. What is the difference in Matlab between typing: `x=[1 2 3]` and `x=[1,2,3]` and `x=[1;2;3]`? What happens if you type a semicolon at the end of the commands (i.e., `x=[1 2 3];`)?

4. (Referring to the last question) For each of those, what happens if you type `x.^2+3`? What happens if you forget the period (e.g., `x^2+3`)

5. What do the following commands do: `x=2;3;6;`, `x=2:3:6;`, `a=2.3:0.5:3.5;`

6. Describe the output for each of the following Matlab commands. Recall that typing a semicolon at the end of the line suppresses Matlab output- to see the results, leave off the semicolon.

```
A=rand(3,4);
A([1,2],3)=zeros(2,1);
B=sin(A);
C=B+6;
D=2*B';
E=A./2;
F=sum(A.*A);
```

7. What will Matlab do if you type in:

```
A=rand(3,4);
A(:)
A(7)
```

   NOTE: This is very bad programming style! Don't do it unless you know what you're doing!!

8. What is the Matlab command to perform the following:

   (a) Given an array x, add 3 to each of its values.

   (b) Given an array A, remove its first column and assign the result to a new array B.

9. What will the following code fragment do?

```
a=1:10;
for k=1:10
  h=ceil(length(a)*rand);
  b(k)=a(h);
  a(h)=[];
end
```

Compare this with `a=ceil(10*rand(10,1))` and `a=randperm(10)`

10. Use the Quick Summary sheet to help you write a code fragment that takes a random matrix X and re-sorts the columns so that the first column has the smallest size and the last column has the greatest size.

1.6 Exercise Set II

1. Let x be a row. What happens if you type `plot(x)`?

2. Write a Matlab script file to plot $y = \sin(x)$ in red, $y = \sin(2x)$ in black, and $y = \sin(3x)$ in green, all on the same plot. You can assume that $x \in [-4, 8]$.

3. When we compute with numbers, some errors can occur. Try typing each of the following into Matlab, and see what happens:

   - `eps` (This is machine epsilon)
   - `1/0` (Think about what this means before trying it!)
   - `-1/0`
   - `0/0`
   - `1/Inf`
   - `Inf+Inf, Inf-Inf, Inf/Inf, Inf/0`

4. Try to reason out what you think Matlab will do with each of the following, then type it in and record what you get:

```
x=[1 3 2 1 3];
max(x)
[vals, idx]=sort(x)
find(x==max(x))
mean(x)
sum(x)
```

5. Write a Matlab function that will take in two matrices A, B of the same dimensions, and will output a matrix C so that

$$C(i, j) = \max\{A(i, j), B(i, j)\}$$

   The function will use the built-in `max` function- See Matlab's help file by typing (in the command window) `doc max`

6. What does this function do (think about what each line does individually; use the help features in Matlab)? The input B is an $m \times n$ matrix.

```
function A=mystery(B)

[m,n]=size(B);
Temp=sqrt(sum(B.*B));
A=B./repmat(Temp,m,1);
```

(Hint: It is often very useful to have a matrix where each column has unit length).