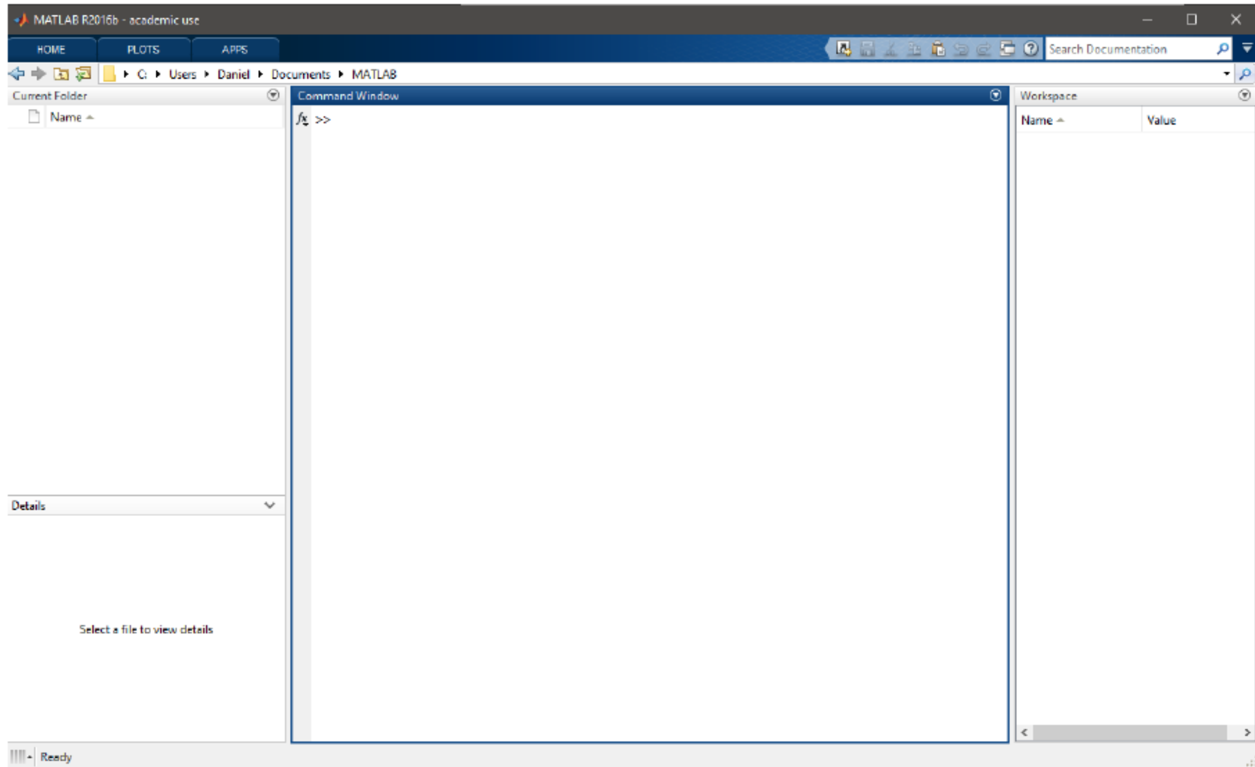
MATLAB Basics

MATLAB's User Interface

First, let's cover what's in the default layout:



There's a few different panels that contain helpful information:

- **Current Folder** - Displays current files in the specified folder
- **Details** - View information or a preview of certain files
- **Command Window** - Enter commands, indicated by the prompt (>>)
- **Workspace** - Explore details of data and stored variables

Most of your focus will be looking at the **Command Window** or through m-files, which will be discussed later. This is a useful place to do basic computations or test functions quickly. It is also where warnings and error messages will be displayed, as well as a location to access documentation using 'help', which will be explained later.

MATLAB is a scientific calculator

We can do elementary arithmetic by MATLAB. The commends of elementary arithmetic we use in MATLAB are addition (+), substraction (−), multiplication(∗), and division (/). Typing $2 + 1 + 3 + 1$ in the Command Window, we get

```
>> 2+1+3+1
ans =
          7
```

The calculations in MATLAB follow the order of operations, such as multiplication or division is granted a higher precedence than addition or substraction:

```
>> 2+1*3+1
ans =
          6
```

or we use brackets "( )" to change the precedence:

```
>> (2+1)*3+1
ans =
          10
```

Doing division, we get

```
>> 1/4
ans =
          0.2500
```

Rounding and Scientific Notation

In MATLAB, it rounds numbers up to 4 decimal places:

```
>> 0.13538
ans =
          0.1354
```

And as a (postive) number is small than 0.0011, we will get the expression of scientific notation:

```
>> 0.0002131
ans =
          2.1310e-04
```

which is 2.131×10^{-4} and this

```
>> 0.000213167
ans =
          2.1317e-04
```

which is 2.1317×10^{-4}

The Powers of numbers

In MATLAB, we use caret (^) to represent exponent:

```
>> 2^3
ans =
        8
```

and

```
>> 2^-3
ans =
        0.1250
```

Inf and NaN

We will get a scientific notation expression if the exponent is small enough:

```
>> 2^(-1074)
ans =
        4.9407e-324
```

and the exponent is too small such that the result of the calculation is a number of smaller value than the computer can actually store in memory, then we get an underflow condition:

```
>> 2^(-1075)
ans =
        0
```

On the other hand, we will get an overflow condition if the number is too big such that the calculation is greater than that which a given storage location can store:

```
>> 2^(1024)
ans =
    Inf
```

Here **Inf** means infinity. Also there is a notation **NaN** which means "Not a Number". We will get this notation when we calculate $\frac{0}{0}$:

```
>> 0/0
ans =
    NaN
```

Trigonometric and Inverse Trigonometric Function

In MATLAB, we have trigonometric functions:

`sin()` for sine, `cos()` for cosine, `tan()` for tangent, `cot()` for cotangent, `sec()` for secent, and `csc()` for cosecant function.

Instead of degree, radian is the unit to measure the angle of the trigonometric function in MATLAB:

```
>> sin(3.1415926)
ans =
    5.3590e-08
```

which is very closed to 0. We also can try $\cos(\pi)$:

```
>> cos(3.1415926)
ans =
    -1.0000
```

There is a function used to calculate π. Try typing `pi` in commend window:

```
>> pi
ans =
    3.1416
```

Then we can try `sin(pi/2)` and `cos(pi)`:

```
>> sin(pi/2)
ans =
    1

>> cos(pi)
ans =
    -1
```

Also, we have inverse trigonometric functions:

`asin()` for arcsine, `acos()` for arccosine, `atan()` for arctangent, `acot()` for arccotangent, `asec()` for arcsecent, and `acsc()` for arccosecant function.

You can try to put some numbers in those functions. See if the answers are the same as what you learned in Calculus.

Variables and Arrays

The following commands show how to enter numbers, vectors and matrices, and assign them to variables. In MATLAB, we use equal sign (=) to assign numbers, vectors, or matrices to variables. We can type a variable 'a' in MATLAB:

```
>> a
```

Undefined function or variable 'a'.

This means we haven't assigned any things to variable a yet.
Let's start with setting a variable. Typing $a = 1$, for example, will display

```
>> a = 1
a =
        1
```

You can also do computations, such as computing sums or trig functons. Try typing $b = a + 3,\ c = \sin(b)$. This will compute the new value for b using the value of a, and then compute c. You should see this in the command window:

```
>> b = a + 3, c = sin(b)
b =
        4
c =
    -0.7568
```

Note that we've separated inputs by using a comma (,) between the two operations. These will do both computations and display them both in the command window. You can also suppress outputs to the command window using a semicolon (;). You will need to do this most of the time in your homework assignments, so take note of it. For example, suppose we type $d = b * a$;, then the output looks like

```
>> d = b*a;
```

But, you can tell that it did the computation. If you look at your Workspace panel, you'll see the variable d and it's value (4). You could also check that it did this by typing d in the command window. That will tell you the value of the variable.

So far, we have only dealt with scalars. We are more interested in working with vectors and matrices, so we will create *arrays*. A one-dimensional array is a vector and a two-dimensional array is a matrix.

To create a row vector, use brackets ([ ]) to start and end an array, and separate the elements with a comma (,). Type $v = [1, 2, 3]$ in the command window, you should see

```
>> v = [1,2,3]
v =
        1       2       3
```

This is a **row** vector. So it's size is 1×3, since there is only one row, but three columns. Similarly, we can make a column vector by separating elements with a semicolon:

```
>> u = [1;2;3]
u =
        1
        2
        3
```

which has size 3×1. It's important that you keep track of the size and order of these vector lengths. Column vectors and row vectors won't add together unless you make them both a row vector or both a column vector. You can make a column vector into a row vector or vice-versa by using transpose, which is done by using the apostraphe ('):

```
>> vcol = v'
vcol =
        1
        2
        3
```

To create a matrix, you will use the same ideas. Elements in a row are separated by a comma and elements in a column are separated by semicolons. So, to make a 3×3 matrix, you can type

```
>> A = [1,2,3;4,5,6;7,8,9]
A =
        1       2       3
        4       5       6
        7       8       9
```

Be careful when entering these. MATLAB will respond with an error if you make a different number of elements in a row, for example. So, if your rows has seven entries, you need to enter seven entries in each row.

The Functions colon (:) and linspace()

In MATLAB, we can make an arithmetic sequence or array by using colon(:). We can type this:

```
>> 1:10
ans =
        1  2  3  4  5  6  7  8  9  10
```

So `J:K` is the same as a vector $[J, J+1, ..., J+m]$, where $m = K - J$. In the case where both J and K are integers, this is simply $[J, J+1, ..., K]$. This syntax returns an empty matrix if $J > K$.

If we want to have a vector included all even positive numbers which are less then or equal to 10, we can type `2:2:10`

```
>> 2:2:10
ans =
        2  4  6  8  10
```

So `2:2:10` is the same as `[2, 2+1*2, 2+2*2,2+3*2, 2+2*4]`.

There is another function we can use to get an even numbers array. We use `linspace()`:

```
>> x=linspace(2,10,5)
x =
        2  4  6  8  10
```

which means `linspace(2,10,5)` generates 5 points between 2 and 10.

For a matrix, we can use colon to get all elements of that matrix, regraded as a column vector:

```
>> A(:)
ans =
        1
        4
        7
        2
        5
        8
        3
        6
        9
```

Some Basic Functions

It isn't very practical to enter in matrix or vector elements by hand. There's a few useful functions to generate matrices quickly. We will cover some here and maybe more in class if time permits. The simplest one is zeros(a,b), which generates a matrix of zeros that have a rows and b columns. Try $Z = zeros(2, 3)$, you should see

```
>> Z = zeros(2,3)
Z =
        0        0        0
        0        0        0
```

You can do similar things such as $ones(a, b)$, which is a matrix of size $a \times b$ that only has 1's as the elements. Another useful one is $eye(a)$, which will generate a size $a \times a$ diagonal matrix with 1's on the diagonal. You should test these to see how they work.

Another useful tool is to generate random matrices. There's a few ways to do this all of which

have different properties. The most basic one is the function magic. Type $M = magic(3)$, this will generate a 3×3 matrix that has entries $1, 2, 3..., 9$. It should display something like

```
>> M = magic(3)
M =
        8     1     6
        3     5     7
        4     9     2
```

Note that this will be different for most everyone. The entry locations are random. Try using $rand(a, b)$ or $randn(a, b)$. These will generate random matrices of size $a \times b$ from different distributions. For example, $N = rand(2, 4)$ looks like

```
 >> N = rand(2,4)
N =
        0.2785    0.9575    0.1576    0.9572
        0.5469    0.9649    0.9706    0.4854
```

Again, these numbers will be different for everyone and they will change each time you try this. But, these should be numbers between 0 and 1.

Array Indexing

Not only is it important to generate matrices, but to be able to access specific elements in a matrix. We will cover that now. Consider our matrix A again from above:

```
A =
        1     2     3
        4     5     6
        7     8     9
```

Suppose we want to extract the number 6. We can do this by selecting the row and column of the matrix. In this case, we'd want to type $A(2, 3)$, since 6 is in the second row and third column. Typing this will display

```
>> A(2,3)
 ans =
        6
```

You can also do this by using a single value, by counting down the columns starting on the left. So, for example, typing $A(8)$ will also give us 6, because it is the second to last element if counting down starting from the left. However, this is usually annoying to code in practice and rarely used.

This can also be used to edit particular elements of a matrix. Suppose we want to change that 6 to a 7. You can do the same thing, but set the new value after the equals, so:

```
>> A(2,3) = 7
```

```
A =
     1     2     3
     4     5     7
     7     8     9
```

You can also pull out multiple elements, such as an entire row or column vector. First set a 4×4 matrix B

```
>> B = [1,2,3,4;4,5,6,7;7,8,9,0;  9,0,1,2]
B =
     1     2     3     4
     4     5     6     7
     7     8     9     0
     9     0     1     2
```

If I want the second to fourth elements in row two of B, for example, I can type $B(2, 2:4)$, which gives

```
>> B(2,2:4)
ans =
     5     6     7
```

If you want the entire row (or entire column), you can just type : instead. So, if we do $B(:, 2)$, this will give the second column of B:

```
>> B(:,2)
ans =
     2
     5
     8
     0
```

You should test this out a few times to get a feel for how it works. But, the main idea is the elements of a matrix have a particular location, which is *indexed*. Using those indices, you can select specific elements of the matrix. A couple questions in your homework involve this and we will cover it more in the online demonstration.

Dot Operators

Reviewing the addition operator, we can do addition of two scalars, addition of two same size arrays. What happens if we do addition of a scalar and a array?

```
>> 2+[1,3,1]
ans =
     3     5     3
```

We get an element-by-element addition.

How about multiplication? We can do multiplication of two scalars, multiplication of a $m \times n$ matrix and a $n \times l$ matrix. What happens if we do multiplication of a scalar and a array?

```
>> 2*[1,3,1]
ans =
        2        6        2
```

But if we use `*` only, there is no such function for an element-by-element multiplication of two same size arrays or matrix .

So we have dot multiplication operator (`.*`)

```
>> [2,3,4] .* [1,3,1]
ans =
        2        9        4
```

which is `[ 2*1, 3*3, 4*1]`.

We also have dot division operator (`./`)

```
>> [2,3,4] ./ [1,3,1]
ans =
        2        1        4
```

which is `[ 2/1, 3/3, 4/1]`.

How about doing dot multiplication or division operator between two $m \times n$ matrices?

Solving Ax =b

One important beginning tool is solving the equation $Ax = b$, where A is a $m \times n$ matrix, x is a $n \times 1$ vector that we want to find, and b is a $m \times 1$ vector. MATLAB can do this by using the rightslash (\) operation, $A\backslash b$. For example, take A as we first defined it and $b = [1; 2; 3]$. Solving for x, we should see

```
>> x = A\b
x =
   -0.2333
    0.4667
    0.1000
```

Remember, you need to be careful about the operations used. If you tried to do an index division (.\), an error would pop up.

Using String Print

It will be useful to also know how to print strings and save them as variables. It is a way to display and explain the output of some computation, for example. Suppose we want

to write the full line 'The first entry in matrix A is 1'. This can be done using the sprintf command. To do so, we will need to specify the strings, with '' around them, and then call the variables we'd like to use in the string. For the example above, we would type

```
>> sprintf('The first entry in matrix A is %d.',A(1,1))
ans =
The first entry in matrix A is 1.
```

The parantheses indicate to call the variable after the comma. You can actually use multiple variables separated by comma's and call each one in order with the parantheses. The letter after the parantheses is the specifier, which tells MATLAB how you would like to convert the variable to a string. The d will tell MATLAB to store it as an integer, while g will tell it to store as a floating point number. There's others also and you can check the MATLAB documentation. For our purposes, we will usually only need d or g specifiers.

Notice that if you put a semicolon after, it will suppress the output like before. But, it will keep the string stored in a variable (in this case, ans is the variable).

Using help for documentation

If you're struggling, remember that you can email me with questions or post on the discussion board on Blackboard. However, some useful tools you can consider (besides just Googling) is using the help feature in MATLAB. In the Command Window, you can type 'help function'. MATLAB will display documentation on that function in the same window. Try 'help magic', you should see

```
>> help magic
 magic  Magic square.
    magic(N) is an N-by-N matrix constructed from the integers
    1 through N^2 with equal row, column, and diagonal sums.
    Produces valid magic squares for all N > 0 except N = 2.

    Reference page for magic
```

This is a useful way to diagnose what might be wrong with some of your code. Errors will sometime display information, but other times it will only tell you the line that produced the error. So, from that line, you may need to read up on the functions that you used to fix the error.

Above all, spend some time playing around in MATLAB. The best way to familiarize yourself with the environment is to sit down with it and figure out what works and what doesn't. Again, if you have questions, you're welcome to ask me or your peers. Don't expect everything you type to work first try. It takes practice, and even then, you'll still make mistakes here or there.

Exercise

1. (*Rounding and Scientific Notation*)Are the answers of the following two calculatings the same if we use MATLAB? Why?

$$1 - (7 * 1/7), \quad 1 - (1/7 + 1/7 + 1/7 + 1/7 + 1/7 + 1/7 + 1/7).$$

2. (*Rounding and Scientific Notation*)What do we get if we type "999999999" in MATLAB? How about "9999999999"? Try them.

3. (*Variables and Arrays*) Store the value 3.14 as the variable p.

4. (*Variables and Arrays*) Store a vector v that starts at 9 and goes to 1, decreasing by 1 for each entry in the vector.

5. (*The Functions colon (:) and linspace()*) How to get an array included all odd positive numbers which are less then or equal to 10 by using commend colon?

6. (*The Functions colon (:) and linspace()*) How to get an array included all odd positive numbers which are less then or equal to 10 by using the commend `linspace`?

7. (*Some Basic Functions*) Generate a matrix called A of size 3×3 using `rand`.

8. (*Using String Print*) Print the string 'The second element of v is: 8' using `sprintf` and the specifier d. Call the variable, don't just write 8.

9. (*Using String Print*) Print the string 'The first entry of matrix A is: **' where ** is the value using `sprintf`. Use the specifier g.