EGR 111
Matrices and Relational Operators

This lab is an introduction to writing your own MATLAB functions. The lab also introduces relational operators and logical operators which allows MATLAB to compare values and count the number of values that satisfy a given condition.

New MATLAB Commands: matrix indexing, `<, >, ==, <=, >=, ~=`

1. Matrices

As we have seen before, MATLAB variables can hold a single value (for example `x = 4`) in which case the variable is called a scalar. MATLAB variables can also store a list of values (for example y = [1, 2, 3, 4]) in which case the variable is called a 1D array or a vector. In addition, MATLAB variables can hold a 2D table of values in which case the variable is called a 2D array or a matrix. Matrices are useful for representing tables of data and images.

For example, try typing the following command:

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

MATLAB prints out the following matrix:

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

In the command above, the square brackets tell MATLAB that we are defining a matrix in a similar way that we used them to define vectors. The elements are separated by either commas (as shown above) or spaces in a similar way as we have done before with row vectors (for example `y = [1, 2, 3]`). The rows of the matrix are separated by semicolons in the same way that we do for a column vector (for example `y = [1; 4; 7]`).

To access an element of a matrix, we need to specify the row and column number (in that order). For example, assuming that you defined the matrix `A` as shown above, you can access the element in the second row and third column by typing the following:

```
A(2,3)
```

MATLAB will respond with "`ans = 6`" which is the value in the second row and third column.

We can also access a given element to change its value using the assignment operator as follows:

```
A(2,3) = 100
```

The command above overwrites the element in the second row and third column of the matrix `A` with the new value of 100 leaving the other values unchanged. The old value (which was 6) is discarded. MATLAB will print out the following matrix:

```
A =
     1     2     3
     4     5   100
     7     8     9
```

We can also use the colon operator to access multiple elements of a matrix in the same way that we did for a vector. For example, type the following command:

```
A(1:3,3) = 10
```

The expression "`1:3`" expands to [1 2 3], and so MATLAB will access the elements that are in rows 1, 2, and 3 and in the third column and set those to the value 10:

```
A =
     1     2    10
     4     5    10
     7     8    10
```

We can use the keyword "end" to mean the last row or the last column just like we did with vectors. So, for example, type the following command:

```
A(2,1:end) = 44
```

When we do not specify the increment when using the colon operator, MATLAB assumes that the increment is 1, so the above command will access elements in the second row and columns [1 2 3] and change them to the value 44:

```
A =
     1     2    10
    44    44    44
     7     8    10
```

If we use the colon operator in indexing without specifying the start, increment, or end values, then MATLAB assumes that the start is 1, the increment is 1, and the end value is "`end`". So we can access a whole row or column as follows:

```
A(3,:) = 55
```

The above command will change all of the elements in the third row to 55:

```
A =
     1     2    10
    44    44    44
    55    55    55
```

We can delete a whole row or column using the colon operator as follows:

```
A(1,:) = []
```

The command above will delete all of the elements in the first row:

```
A =
    44     44     44
    55     55     55
```

Thus we can use indexing to access any subset of a matrix by specifying the rows and columns that we want to access.

Exercise 1: Write a script file named MatricesEx1.m that does the following:

1. Clear the existing variables then create the matrix `x` = [1 2 3 4; 5 6 7 8; 9 10 11 12]. Then multiply this matrix by 5 and store it in the matrix `y`. Then change elements in the second row of `y` to the value 100.
2. Use the ones command to create a 4 by 5 matrix (that is a matrix with 4 rows and 5 columns) called z filled with all ones (for details on the ones command, type "`help ones`" in the Command Window). Next change all of the elements in the first row to 10 and then change all of the elements in the last column to 100.

Checkpoint 1: Show your instructor your script file from Exercise 1.

2. Relational Operators

The relational operators in Table 1 are used to compare two numerical values.

Table 1. Relational Operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
~=	Not equal to

Let's look at some simple examples. First, determine if 5 is less than 8:

```
>> 5 < 8
ans =
     1
```

Notice that the answer is 1. In MATLAB, 1 means *true* and 0 means *false* (Actually, any non-zero value is also treated as true.) That is, it is *true* that 5 is less than 8.

Next let's see if 5 equals 8:

```
>> 5 == 8
ans =
     0
```

Note that the double equal sign == is used when we are comparing two values (5 == 8), whereas the single equal sign = is used when assigning a value to a variable (x = 4).

Relational operators also work on variables, vectors, and matrices. For example, if we want to identify which values in vector x are negative, we could compare x to 0 as follows.

```
>> x = [-20 -10 10 20 30]
x =
   -20   -10    10    20    30
>> x < 0
ans =
     1     1     0     0     0
```

The command `x < 0` above compares each element of x to 0, and returns a vector of ones and zeros where the ones indicate for which elements of x the comparison `x < 0` is true.

When using these operators, one has to be careful of operator precedence. **The arithmetic operators have precedence over the relational ones**, so in the following command, the addition and division are computed first resulting in 7 < 8, and then the comparison is performed.

```
>> 3 + 4 < 16 / 2     % + and / are executed first
ans =
     1
```

Compare with the following:

```
>> 3 + (4<16) /2    %  < is executed first
ans =
     3.500
```

Exercise 2: Write a function called cmp that has two scalar input arguments and one output argument. If the two input arguments have the same value, the function should return 1, otherwise the function should return zero.

Exercise 3: Write a function called posneg that has one input argument and two output arguments. The first output argument should be the absolute value of the input and the second output argument should be the absolute value times -1. For example, if the input is -4, the first output will be 4 and the second output will be -4. if the input is 10, the first output will be 10 and the second output will be -10.

Checkpoint 2: Show the instructor your functions from Exercise 3 and 4.