Matrix Manipulations, For Loops, and Complex Numbers in Matlab

Note: This lab contains a combination of useful Matlab examples as well as a few exercises for you to work through on your own. Questions that I want you to answer in your lab write up are written in bold type.

Introduction

Often, there are multiple ways to accomplish the same task in Matlab (or any programming language). In this lab, we illustrate some basic matrix manipulations using three different methods. In practice, it is almost always best to use the built in Matlab commands, but understanding how to use for loops or write the code out fully will help to understand what Matlab is doing when you use the built in commands.

Example 1: Element-wise Multiplication – vectors

It is common to encounter a situation where you have two arrays (or vectors) and want to perform the same operation (say addition or multiplication) on each value. For example, Say we have $A = [1 \ 2 \ 3]$ and $B = [2 \ 3 \ 4]$. We would like to create an array $C$ where each element is the product of the corresponding elements in $A$ and $B$. In this case we want $C = [2 \ 6 \ 12]$. There are multiple ways to do this in Matlab, as shown below.

Method A: Write everything out.

Perhaps the first method that comes to mind is to create the array $C$ one element at a time. For each of the three numbers, we can write a code to calculate the product and place it in array $C$, as shown here.

```
%Example Adding 2 arrays
A = [1 2 3];
B = [2 3 4];

%element wise multiplication - written out
C1(1) = A(1) * B(1);
C1(2) = A(2) * B(2);
C1(3) = A(3) * B(3);
```

Method B: Using a for loop

In this case, it is possible to write everything out since there are only 3 numbers in the array. But what if the two arrays we wanted to multiply each had 100 or even 1000 elements? You probably don’t want to do that much typing! In this case an alternative would be to use a for loop as follows,
The for loop performs the exact same series of commands as method A, but using slightly less code. The for loop is also useful if you don’t necessarily know the length of the vectors A and B. In this case we could replace ‘3’ with length(A) or length(B).

**Method C: Built in Matlab Operator**

The for loop works well, and produces exactly the same result as method A. However, there is a third way to do this using a built in Matlab function. In Matlab, placing a ‘.’ before an operator indicates that you want to do perform the operation ‘element-wise’, i.e., to perform the same operation on each element of an array. In this case, we can simply write the following to multiply each element in A by the corresponding element in array B.

```matlab
%element wise multiplication - Matlab notation
C3 = A.*B;
```

**Question 1.** In Matlab, write a program that defines an array, ‘angles’, containing the angles from 0 degrees to 360 degrees in increments of 10 degrees. Using a for loop, create an array z, whose elements are complex numbers corresponding to $e^{j(\text{angles})}$. To do this your program will first need to convert the angles from degrees into radians. You can use the function exp(value) to calculate $e^{\text{value}}$. This problem has a few steps

a. Using a for loop, create the array z. What are its entries (copy them into your report)?

b. It is common to plot complex numbers in the complex plane, using the real part of the number as the x coordinate, and the imaginary part of the number as the y coordinate. For example, the value $1 + 4j$ would be plotted at the point (1,4). Using the Matlab commands “Real” and “Imag”, create two new vectors x and y which contain the real and imaginary parts of z respectively.

c. Plot the values contained in z in the complex plane using plot(x,y).

d. In a new figure, plot the values contained in x vs angle. Make a separate plot of the values of y vs angle. What do you notice about these plots?

**Example 2: Element wise multiplication – matrices**

Above, we showed how a for loop could be used to do element-wise multiplication of 1-d arrays. We can perform the same operation for 2-d arrays (also referred to as matrices), except this time we need an extra for loop in method B. In this example we will do an element wise multiplication of the matrix $A = [1 \ 2 \ 3; \ 2 \ 3 \ 4; \ 3 \ 4 \ 5]$ with $B = [1 \ 2 \ 3; \ 4 \ 5 \ 6; \ 7 \ 8 \ 9];$
Method A:
Again, if we were really determined, we could do this by writing out nine lines of code, as follows:

\[
\begin{align*}
A &= \begin{bmatrix} 1 & 2 & 3;  & 2 & 3 & 4;  & 3 & 4 & 5 \end{bmatrix}; \\
B &= \begin{bmatrix} 1 & 2 & 3;  & 4 & 5 & 6;  & 7 & 8 & 9 \end{bmatrix}; \\
C1(1,1) &= A(1,1) \cdot B(1,1); \\
C1(1,2) &= A(1,2) \cdot B(1,2); \\
C1(1,3) &= A(1,3) \cdot B(1,3); \\
C1(2,1) &= A(2,1) \cdot B(2,1); \\
C1(2,2) &= A(2,2) \cdot B(2,2); \\
C1(2,3) &= A(2,3) \cdot B(2,3); \\
C1(3,1) &= A(3,1) \cdot B(3,1); \\
C1(3,2) &= A(3,2) \cdot B(3,2); \\
C1(3,3) &= A(3,3) \cdot B(3,3);
\end{align*}
\]

Method B:
To do this with a for loop, we now need two levels of loops. This is referred to as “Nested for loops”. The outer loop loops over rows and the inner loop loops over columns. The resulting code looks like this:

```matlab
for r = 1:3
    for c = 1:3
        C2(r,c) = A(r,c) \cdot B(r,c);
    end
end
```

Method C:
Again, in practice, we would generally use the built in Matlab operator ‘.*’ to perform this operation.

\[
C3 = A \cdot B;
\]

**Question 2: Create a multiplication table.**

In the example above, we performed element wise multiplication for two matrices that were the same size. However, nested for loops can also be used to create a matrix based on the entries of one or more 1 dimensional arrays. In this problem, the goal is to create a “multiplication table” using a single row vector to generate a table of all the possible products.

a. Create a starting vector \( xx \) that contains all the numbers from 1 to 10. How would you do this without typing all 10 numbers out?
b. Use a set of nested for loops to generate a 10 by 10 array where each element is the product of the associated row and column entries of xx.

c. Can you think of a way to do this using operations that are already built into Matlab, such as ‘*’ and ‘transpose’?

Question 3: Write a program to plot complex numbers

a. In this final part of the lab you should write a Matlab program that lets you input complex numbers and plot them.

b. Use the ‘input’ command to read an input from the ‘Command’ window, like this

   A = input('Enter a complex number in the form a + j*b  ')

This command will print ‘Enter a complex number in the form a + j*b’ in the command window and then wait for you to type something, let’s say you type

   1 + j*2

After you press ‘return’ the program then assigns this value to the variable A.

c. Using a ‘for loop’ make a program that will let you read in any number of complex numbers and then plot them all on the same plot.

Report

For this lab, please submit a write-up that includes your answers to each of the questions above. Please upload both the write up and your Matlab code to blackboard under the assignment ‘Lab 3’.