1 Introduction

The 2460 class is geared towards getting you familiar with MATLAB. MATLAB is short for Matrix Laboratory, so you can imagine that we’ll be dealing with matrices. MATLAB can function as a basic calculator, a visualization tool, a programming language, and more! Throughout the semester, we’ll be teaching you things to get you accustomed to the program, as well as preparing you for the projects due in APPM 2360. You can download MATLAB from CU by visiting http://www.colorado.edu/oit/. However, both ECCR 143 and ECCR 252 computer labs have current versions of MATLAB installed. There are also many other labs on campus that have versions of MATLAB. See the CU website of the Office of Information and Technology for details.

2 Pseudo-Code

Programming a computer can seem daunting, and overwhelming when it’s not working. The key to understanding what the computer is doing is to use pseudo-code. This is a way to write, by hand, what you want the computer to do at each step. Then, you can follow the instructions yourself, and see what the computer should know at each step.

3 Variables

Matlab does not think of variables the same way that we people do in math classes. You can think of Matlab as storing variables inside a filing cabinet, with a note card for each variable. If I tell Matlab:

\[ x = 2 \]

Matlab will create a new file (or bucket), label it ’x,’ and write the value 2 on the back.

4 Assignment

When we want to save a value to a variable, that is called assigning the value. In the example above, we are assigning the value 2 to the variable \( x \). In pseudo-code, we use a backwards arrow (←) to denote assignment. We use = for something else, which we we will cover later in the semester. The arrow helps us to remember that Matlab starts on the right side of the assignment, computes a value, then saves it to the variable on the left. Consider this line of pseudo code:

\[ x ← 2 + 3 \]

Here, the program starts on the left, adds 2 and 3 together, gets 5, then writes 5 to (the note card associated with) \( x \).
In math classes, an equation such as \( x = 2 + y \) makes perfect sense. \( x \) depends on \( y \). We could plot it, take the derivative, etc. However, to Matlab, this is gibberish. If you type this into Matlab, you will get an error. This is because Matlab works in terms of concrete values. The first thing Matlab will do is go to the variable (note card) \( y \), and retrieve the value, which it will then add to 2. However, there is no such variable! We need to tell Matlab what \( y \) is first. These lines of pseudo-code do make sense:

\[
\begin{align*}
y &\leftarrow 5 \\
x &\leftarrow 2 + y
\end{align*}
\]

Now, in the second line, the program would go the variable \( y \), retrieve the value (5), add it to 2, get 7, then store it in the variable \( x \).

This way of dealing with variables can be more powerful than our typical algebraic perspective. For example, in a math class, \( x = x + 2 \) is an impossibility. However, Matlab can handle this expression:

\[
\begin{align*}
x &\leftarrow 3 \\
x &\leftarrow x + 2
\end{align*}
\]

Let’s go through this one step at a time. First, Matlab creates a variable \( x \), and stores the value 3 in it. Now, remember that we always start on the right side of an assignment. First, Matlab retrieves the value of \( x \) (3), adds it to 2, gets 5. Then, it stores it in the variable on the left side of the assignment, which happens to be \( x \). There is no issue with the fact that we used \( x \) on both sides of the assignment. The value stored in the variable \( x \) simply changed from 3 to 5.

## 5 Vectors

Since this class deals with vectors, we will certainly want to be able to work with them in Matlab. Most anything we do this semester will use vectors. Using the note card visualization, a vector variable will have a row of numbers on the back of the note card. We use brackets to tell Matlab that we are creating a vector, with commas between each of the numbers, known as elements. For example, I could create a vector of the numbers 1 through 4 using the following pseudo-code:

\[ x \leftarrow [1, 2, 3, 4] \]

What if I wanted to create a vector of the numbers between 1 and 1000, or 1000000? This method would require that I spend a LOT of time typing. Instead, we can create a whole vector all at once using the following template:

\[ x \leftarrow \text{firstnumber} : \text{increment} : \text{lastnumber} \]

So, for the above example, the following pseudo-code generates the same vector:

\[ x \leftarrow 1 : 1 : 4 \]

The first number in the vector is 1, we want to increment by 1, and the last number is 4. If I wanted a vector of all of the multiples of 10 under 10000, I could type:

\[ x \leftarrow 0 : 10 : 9999 \]
6 Indexing a vector

Now that we can create a vector, we want to be able to use, and manipulate them. Suppose I have a vector which contains the temperatures in Boulder on Jan 14 for a period of 5 years. It might look something like this, where the first element is last year’s temperature, and the last is from 2010:

\[ \text{temps} \leftarrow [30, 40, 24, 55, 18] \]

Now, what if I wanted to use the temperature from 2011? Then we would want to index the vector. We do this using parenthesis next to the variable name of the vector, and the number corresponding to the location of the element we want. 2011 is in the 4th element, so we can type:

\[ \text{2011temp} \leftarrow \text{temps}(4) \]

Now the variable 2011temp would contain the value 55.

What if we wanted to go in reverse? That is, instead of retrieving one of the elements, we wanted to change the temperature we have recorded for 2012. We would want to assign a value to temps(3), so it would go on the left side of the arrow, and the value we want to save on the right.

\[ \text{temps}(3) \leftarrow 51 \]

What if we wanted to change the temperatures for both 2012 and 2011 at once? We can use the colon once again. When indexing, the colon reads like through. That is, \( x(1 : 3) \) would index the first through the third entries of the vector \( x \). Thus, to change the temperatures for 2011 and 2012, we could type:

\[ \text{temps}(2 : 3) = [30, 33] \]

Then, the vector temps would look like:

\[ \text{temps} = [30, 30, 33, 55, 18] \]

7 Homework

Complete the handout. It is due in the beginning of next class.
Homework 1

Instructions

Complete the right column, by filling in the value of the variable that is altered in the corresponding line of pseudocode.

<table>
<thead>
<tr>
<th>Pseudo-code</th>
<th>Variable value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \leftarrow 8 )</td>
<td>( x = 8 )</td>
</tr>
<tr>
<td>( y \leftarrow x/2 )</td>
<td>( y = 4 )</td>
</tr>
<tr>
<td>( x \leftarrow x \times y )</td>
<td></td>
</tr>
<tr>
<td>( z \leftarrow x - y \times 5 )</td>
<td></td>
</tr>
<tr>
<td>( vect \leftarrow [1, 4, 3, 8] )</td>
<td></td>
</tr>
<tr>
<td>( A \leftarrow vect(3) )</td>
<td></td>
</tr>
<tr>
<td>( A \leftarrow A + vect(4)/vect(2) )</td>
<td></td>
</tr>
<tr>
<td>( newvect \leftarrow y : z : x )</td>
<td></td>
</tr>
<tr>
<td>( vect(1) \leftarrow newvect(2) )</td>
<td></td>
</tr>
<tr>
<td>( vect(2 : 4) \leftarrow [x, y, A] )</td>
<td></td>
</tr>
</tbody>
</table>