Video #12: Writing External Text Files

You've had experience earlier in the semester with saving .mat files, which are a special kind of file that can only be read by MATLAB. Suppose we want to store information from our program to an external file that we can read ourselves, either in the MATLAB editor or in another application. Text files enable us to do that. We'll start here with the example of our chemical elements. Suppose we have variables in our MATLAB workspace that store the atomic numbers, names, symbols, and masses of some elements and we want to write this information out to an external text file. How do we do this? First we're going to consider how we could print the information in the Command Window with the nice structured format that we see here, where the content is shown in these columns that are all aligned.

We'll construct a loop with an index from 1 up to the length of one of the vectors, atomNums, and use a function called fprintf to print the information line-by-line. The first input to fprintf is a format specifier of the sort we're familiar with from reading text files with textscan. It indicates that each line of text will consist of an integer specified with %u, two strings %s and a floating point number %f, and each line will end with a line break or carriage return, and we use that \n that we saw earlier to specify the line break. Then we need to say where the integer, strings, and float will come from, so after the format string, we list the actual content that will replace the format symbols for each line, in the order that these format strings appear. For %u we'll insert the i-th element of the vector atomNums, then for the two strings, we'll insert names{i} and symbols{i} (they're stored in cell arrays so we use curly braces here) and for the float, we insert masses(i). <run section> We see all the right information printed, but it's not yet nicely formatted in these aligned columns that we're aiming for. But there's a way to do that in the format string that we supply to fprintf - we tuck an integer between the percent sign % and the letter that describes the type of information. This integer specifies a fixed number of spaces to allocate for this information. So we allocate 3 spaces for the atomic number, exactly 10 spaces for the name, 4 spaces for the symbol, and the floating point number is different. We have 5.2 here - the 5 is the total number of spaces we allocate for the number, and the 2, after a decimal point, indicates how many digits we want to print after the decimal point when we display the number. Here we say, print two digits as shown in our earlier picture. <run section and view printout>. It's close to what we want, but notice that the content within each column is right-justified within the allocated space. But in our picture, the first three columns are left-justified, they line up on their left side. We can modify our format string once more to add a minus sign in the first three elements, which specifies that they should be left justified instead. <run section> Now it looks like what we want.

So we can use fprintf instead of disp, to print information from our program in a way that gives us more control over the formatting of the information. With fprintf, we also don't need to use num2str to convert numbers to strings in order to concatenate numbers and strings together. How do we now write this information to a text file? We'll again do this in three steps, similar to reading the content of a text file. We'll open the file for writing with fopen, write text to the file with fprintf, and close the file with fclose. So let's create a new file named elementsNew.txt

- we again provide this file name as an input to fopen, and this time, we also provide a second input ‘w’ that says, open this file for writing. (If we don’t provide a second input, MATLAB assumes by default, that we’re opening the file for reading, and generate an error if this file doesn’t exist, in the case of reading.) Here, it’s assumed that we want to create a new file named elementsNew.txt - if this file name does exist already, the old file will be replaced by the new one. There is an option to add new text to an existing file. In this case, we would be appending the new text, and we can specify that by providing the character ‘a’ as the second input. But here, it’s ok to create a new file, so we open it with ‘w’ for writing. Then we call fprintf as we did before, but when we’re writing to a text file, we provide a file ID as the first input, to specify the particular file to use. When we’re done writing the content to the file, we close the file as we did before. Let’s run this section and view the elementsNew.txt file.

A couple additional points about fprintf. First, we don’t need a loop to print a vector of values. In this example, we first create three vectors named xdata, ydata, and results. We open a file results.txt for writing, and then write a single line in the file with the literal string ‘experimental results’. We can print literal strings with fprintf like this. Then we’ll print part of the next line, and we’ll insert a line break at the beginning that causes the next printout to be started on a new line, and just write the literal string xdata. The next call to fprintf shows how you can write a whole vector of values all at once - the content is printed on a horizontal line, and this format string ‘%6.2f’ is applied to each element of the vector. So we see the content of the xdata values shown in a horizontal line, with each number displayed in a region of 6 spaces - it’s right justified within its space because there’s no minus sign, and there’s two digits after the decimal point. We then have similar statements to write the content of the other two vectors - the labels ydata and results are written with a line break at the beginning, causing them to appear on separate lines, and the label is followed by the full content of the vectors on a single horizontal line. Finally, we close the file at the end with fclose.

The final example here just reinforces the idea that we can include literal text inside the format string that we provide as input to fprintf. Here we have vectors of data on the blood pressure measurements for some patients with these initials. We open a file BP.txt for writing and then use fprintf in a loop, to write out the data for each patient line-by-line. The format string includes literal text - patient, systolic and diastolic, in addition to the three values that are a string and two integers, and there are three things to insert in each line, the patient’s name from a cell array is inserted in this first place where a string is expected, and the two pressure measurements that are stored in vectors are inserted in the 2nd and 3rd places. And we close the file at the end. <run section and look at the BP.txt file, with content as we indicated>

So that’s the basics of how we can create an external text file that stores information that may be data that was collected by our program, or simulation results, or other information, that may be a combination of textual and numerical information. One of the benefits of writing text files in this way is that we can then read the content ourselves in the MATLAB editor or in other applications that can print the content of .txt files.