Instruction guide for creating QMB adaptive problems

# Basic Workflow

The basic workflow is simple, consisting of 3 steps

1. Create problem in Excel
2. Run Matlab script to generate XML files from the Excel file.
3. Load and test XML files on studio.EdX.org

The following sections will break down these sections in detail. Step 1 is the trickiest and receives the most attention.

# Creating problems in Excel

There are separate Excel files for each unit of the course, e.g. “Arrays problems.xlsx” or “Loops problems.xlsx”. These are in the Github project under “QMB-Problem-Maker\matlab\_attempt\Excel problems”.

Inside these Excel files is a sheet for each Content Group of the unit, so there are sheets named things like “CG1.3.1” or “CG4.3.4”. Each sheet will have multiple problems.

## Simple example

The basic layout for a problem is demonstrated in the following example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| sum5 | questionText | <p> What is $AA + BB/$?  </p> |  |  |
|  | variable | AA | randi(10,1) |  |
|  | variable | BB | AA + 1 |  |
|  | variable | CC | AA + BB |  |
|  | answer | CC | KC1 | TRUE |
|  | problemType | Numerical |  |  |
|  | dynamic | TRUE |  |  |
|  | solutionText | <p>  $AA + BB/$ is the sum of numbers $AA/$ and $BB/$. Therefore, the answer is $CC/$ </p> |  |  |
|  | tolerance | 0.02 |  |  |
|  | contentGrouping | CG3.1.2 |  |  |

The top cell in the 1st column is the **problem ID** (also called the **Item ID** by EdX), which in this case is sum5. Notice that all the rows below it are empty. This marks the boundaries between problems, i.e. a new problem begins when a non-empty cell is encountered in the 1st column.

The 2nd column contains the name of a parameter of the problem, e.g. **questionText** is the text displayed to the user that asks the question, and **solutionText** is the explanation that is displayed after the student gets the question correct.

Columns 3 and above contain information relating to the value of that parameter. In most cases, only column 3 is needed, e.g. for the **dynamic** property, the only necessary info is TRUE or FALSE, which specifies whether this problem is dynamic (can have multiple instances) or static (can only have one instance).

The two special properties that require multiple columns are **variable** and **answer**.

## Variables

### Variables are define with Matlab statements that are executed with eval()and saved as strings

For **variables**, the text in the 2nd column is a variable name. In the example above, there are three variables names AA, BB, and CC. The 3rd column is a matlab command that will be evaluated and “stored” to the variable name. I use “stored” in quotes because there this doesn’t create a Matlab variable per se. Instead, the statement in the 3rd column is evaluated with the Matlab function eval, and the output is converted to a string.

Then, whenever the variable name appears, it will be replaced with the string that is the output of the eval(). For example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | AA | randi(10,1) |  |

This will have Matlab execute the line randi(10,1)which will generate a random integer. This integer will be converted to a string, e.g. '2'. Now, whenever the sequence of letters AA is encountered (even in following **variable** lines, the letter AA will be replace with the number 2. This is the reason the number is converted to a string, so variable letters can be replaced with strrep(), e.g. strrep('AA','2').

### Variable statements can call other variables

The next variable line is

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | BB | AA + 1 |  |

This will assign the string '3' to BB. What actually gets executed by Matlab is eval('2 + 1'), since the statement in this case is simply AA + 1, and AA was replaced with the value 2 using strrep().

The final line is

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | CC | AA + BB |  |

This will assign the string '5' to CC. AA and BB have values of 2 and 3, respectively, and the statement tells Matlab to add them together.

### Variable lines must be in order of execution

It’s important to note that for successive **variable** lines, you can only reference variables that have already been evaluated. For example, the following two lines would produce an error

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | AA | BB |  |
|  | variable | BB | randi(5,1) |  |

The AA line tries to reference BB, which has not yet been declared. Matlab would execute eval(BB) since it was unable to replace BB with a value using strrep().

### Order does not matter for any other parameter

In the above example, the first row with **questionText** parameter contains the variables AA and BB, despite coming before the **variable** lines. This is allowed. The **variable** lines are always executed first. Then, the script will go back and replace variable names with their values in *every* cell. The letters AA and BB will be replaced with the numbers 2 and 3 in the **questionText** and **solutionText**. The letters CC will be replaced with the number 5 in the **solutionText** and **answer**.

In other words, you could have the **solutionText** parameter even before the **questionText** parameter and the variable values will still be replaced. The order of the other parameters does not matter. For example, you could have the row with **contentGrouping** parameter before the row with the **dynamic** parameter. The only lines that need to be in order are the **variable** lines.

### Use the variable name CODE to run Matlab statements without assigning them to a variable

If you name a variable CODE, then the Matlab statement in the 3rd column will be evaluated without assigning it to a variable. This is useful for multi-line statements like if statments. You’ll still need to fit the multi-line statement into a single line by using commas and semicolons. For example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | CODE | if AA < 5, myArray = 1:10; end |  |
|  | variable | DD | myArray(7) |  |

Matlab will execute this statement, which creates a Matlab variable named myArray, which can then be called from other lines. If you need multiple if statements, use multilple lines with the variable name CODE.

Referencing myArray will call the actual Matlab variable, unlike the variable AA, which is a string that is inserted in the correct place with strrep(). This trick is helpful for arrays, since the following lines would produce an error

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | AA | [1, 5, -9, 0] |  |
|  | variable | BB | AA(2) |  |

Here, the first variable AA is note save as an array, but the string '[1 5 -9 0]'. Matlab will first run

eval('[1, 5, -9, 0]') which returns the array, which is then converted to a string with mat2str(). Therefore, when AA is referenced in the next line, Matlab will replace the letter AA with the string and try to execute

eval('[1 5 -9 0](2)'). This will produce an error.

To index from an array, you can use a CODE statement to create a Matlab variable, e.g.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | CODE | myArray = [1, 5, -9, 0]; |  |
|  | variable | BB | myArray(2) |  |

Or, you can use the extract() function I wrote for just this purpose.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | AA | [1, 5, -9, 0] |  |
|  | variable | BB | extract(AA,2) |  |

\*\*For a long time, I used num2str() instead of mat2str(), so AA would actually be the string without square brackets, e.g. '1 5 -9 0'. That’s while you’ll see things like extract([AA],2) which would enclose the variable in square brackets to recreate the array.\*\*

### If a variable returns a cell array, Matlab will extract the first element

I didn’t add support for cell arrays. If you try to define a cell array with the line

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | XX | {'hey',1,[0 4]} |  |

Then the variable XX will have the first element of this cell array, i.e. the string 'hey'. I added this functionality mostly for the randsample() function. For example, if you wish to select a random string from a list of strings:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | YY | randsample({'one','two','three'},1) |  |

Normally, the randsample() function would return a 1x1 cell containing a string. Since I added the “take the first element” functionality, the above line will return the string inside the cell.

### Variable names must not be substrings of each other

Since variable values are filled in with the strrep() function, you must make sure that the name of one variable is not a substring of another. For example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | RR | randi(10,1) |  |
|  | variable | ARRAY | randi(20,5) |  |

Here, the array name RR can be found within the variable name ARRAY. Now, whenever you wish to call the ARRAY variable, the strrep function will replace the two R characters with the value stored in RR, so you’ll get something like A8AY.

### Excel removes single quotation marks when it’s the first character in a cell, so enclose strings in square brackets

If you wish to define a string with the line:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | STR | 'Hello World!' |  |

Excel will remove the first quotation mark ', so you’ll see

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | STR | Hello World!' |  |

This behavior can probably be turned off somewhere in the settings, but I just got into the habit of enclosing strings in square brackets, e.g.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | variable | STR | ['Hello World!'] |  |

This only occurs when the quotation mark is the first character in the cell. Other quotation marks are fine.

### Summary of Variable tips

* Variables are defined with Matlab statements that are executed with eval()and saved as strings
* Variable statements can call other variables
* Variable lines must be in order of execution
* Order does not matter for any other parameter
* Use the variable name CODE to run Matlab statements without assigning them to a variable
* If a variable returns a cell array, Matlab will extract the first element
* Variable names must not be substrings of each other
* Excel removes single quotation marks when it’s the first character in a cell, so enclose strings in square brackets

## Answers

Answer rows are simple. The answer row in our earlier example was:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | answer | CC | KC1 | TRUE |

The 3rd column contains the actual answer. For Multiple Choice and Checkbox problems, this will be the text that is displayed to the user. For Numerical or Text problems, this will be the number or string, respectively, against which EdX will compare the user’s input.

In this line, the 3rd column has the variable name CC. This will be replaced with the value assigned to CC, which in our case was the number 5.

The 4th column contains a comma-separated list of Knowledge Componenet IDs. Our example has one Knowledge Component nmed KC1.

The 5th column contains either TRUE or FALSE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| loop5 | questionText | <p> How many times does the following loop run? </p> <p> $$for i = AA:BB:CC  i end/$$ </p> |  |  |
|  | variable | AA | randi([1,10],1) |  |
|  | variable | BB | randi(3,1) |  |
|  | variable | CC | randi([11 20],1) |  |
|  | variable | ARRAY\_STRING | mat2string(AA:BB:CC) |  |
|  | variable | ANS | length(AA:BB:CC) |  |
|  | answer | ANS | L2 | TRUE |
|  | problemType | Numerical |  |  |
|  | dynamic | TRUE |  |  |
|  | solutionText | <p> A For loop will run multiple times with a different value for $i/$ each time. Specifically, it will run once for each element in the vector $AA:BB:CC/$. </p> <br/> <p> Since $AA:BB:CC/$ creates the array $ARRAY\_STRING/$ which has $ANS/$ elements, the answer is $ANS/$. </p> |  |  |
|  | tolerance | 0.02 |  |  |
|  | contentGrouping | CG3.1.2 |  |  |