 [Translated from Vietnamese to English - www.onlinedoctranslator.com](https://www.onlinedoctranslator.com/en/?utm_source=onlinedoctranslator&utm_medium=docx&utm_campaign=attribution)

**Topic**: Let's create a student management program:

Request:

* Create a function menu, for example, select 1 to view student information, select 2 to add students, select 3 to delete students, select 4 to export the list to a .txt file, ..., you can add if you want.**Add a new function that calculates class statistics and outputs a graph if possible**. The export function you should add is to choose the desired output format, for example.
* Finally press button 6 to escape the program.

=> I can think of the Switch function.

Each student has the following fields of information: Use Struct here.

* Full name
* Year of Birth
* Gender (**can give choice between 2 options for example, advanced feature**)
* Class
* Student status (**same, studying or dropping out as gender**)
* Home town (**can add foreigners feature if not falling in 63 provinces**)
* Phone number

For

If

While loop

I first process each part first:

* Creating menu options is the easiest part, so we'll do that first. Here, we prefer to use the old MATLAB GUI.
* **To make the program user-friendly**.
* We can code a part of the MATLAB program as follows:

**disp('This is a student information management program');**

**disp('Choose 1 if you want to see student info');**

**disp('2 if you want to add new student');**

**disp('3 to delete student');**

**disp('4 to export the data to .txt file');**

**choice = menu('Please choose which options you like',1, 2, 3, 4);**

**switch choice**

**case 1**

**disp('ll');**

**case 2**

**disp('22');**

**case 3**

**disp('33');**

**otherwise**

**disp('44');**

**end**

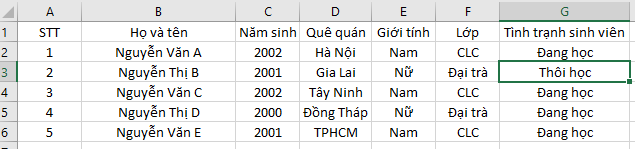
**Here we build a few directions in advance to go into more advanced code later. Let's just build it up so we can have a rib**.

Here the purpose we give the disp is to make it less verbose if we leave it as a message in the menu.

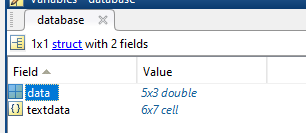
* However, I like to add the while loop.
* Let the code be longer for cool.
* Prevent the user from pressing the x button, to make the user enter the values ​​1, 2, 3, 4
* **However, we have to add a button just for the end of the program**.

Next we see the rest of the features related to the database, so first we need to create a database.

* **Initially, my thinking is to create an excel file and then import it and turn it into a struct file**(like picture below). Because struct files are extremely convenient to handle in MATLAB.



* In addition, most people now enter data into an excel file and then put it into more advanced software for processing. So we'll do exactly the same thing.
* Next we need 2 functions,**1 function to import**and**1 function to convert table to struct**.
* About the table to struct fortunately we found a function named**table2struct**. This function will convert table to structure array. So through that we know that to import data, we need a function to import into a table.
* **So the function importdata cannot be used because it is imported as a struct and that is not the type of struct we want**. It is as follows:



Wrong code:

**database = importdata('student database.xlsx');**

* It separates numbers from letters. That can't be.
* In return we will use function**readtable**, which has the effect of creating table from file. And the syntax is, we just need to enter the file name.
* To save lines of code and reduce variable declaration, we combine**table2struct**and**readtable**as follows:

**database = table2struct(readtable('student database.xlsx'));**

It's actually very easy, readtable creates a table, then uses the same result to put it in table2struct and then save it to a variable.

So basically we have finished importing data. Now we go into processing each branch.

* Dealing with branch 1 is always easiest because it's just a view.
* To view, a table view will be more intuitive and easier to implement than a struct view.
* So the above code to import data needs to be modified a bit as follows:

**table\_form = readtable('student database.xlsx');**

**struct\_form = table2struct(table\_form);**

The purpose for us to separate like that is so that we have both table form and struct form for easy handling.

And here is the code to view the database:

**switch choice**

**case 1**

**disp(table\_form);**

**To view the data, we simply need to disp the table form**.

**For case 2, we will use what we learned in the lesson to create an account with a password**.

* Specifically as follows:

**z = numel(acc)+1;**

**acc\_z\_ten\_dang\_nhap = input('create new username: ');**

**acc\_z\_pass = input('create your password: ');**

**acc(z).ten\_dang\_nhap = acc\_z\_ten\_dang\_nhap;**

**acc(z).pass = acc\_z\_pass;**

**save('du\_lieu.mat','acc');**

Explanation of the above line of code:

* First acc is a struct, here we count the number of elements in that struct.
* **Then we have z as a counter variable, here z = number + 1 means we will create a new account and the z will be effective in indexing the order in the database.**.
* Then we temporarily store the account and password in 2 temporary variables.
* **Finally assign 2 values ​​of those 2 temporary variables to the struct in z order**. Pretty simple.

**Thereby learning how to check each line of code to see if it runs well without affecting the program, then what code we are sure is correct, let's say it is code, if not, leave the comment form all and run it slowly.**.

So, basically, we have the following code framework:

**case 2**

**n = number(struct\_form)+ 1; This +1 is for the purpose of avoiding overwriting**

**new\_student\_fullname = input('please enter new student full name: ');**

**new\_student\_year\_of\_birth = input('please enter new student year of birth: ');**

**new\_student\_place\_of\_birth = input('please enter new student place of birth: ');**

**new\_student\_gender = input('please enter new student gender: ');**

**new\_student\_class = input('please enter new student class: ');**

**new\_student\_status = input('please enter new student status: ');**

These will be like the password post that will be saved to the struct later.

However, here we want to upgrade a bit more, if the gender is for the user to enter, it will be very easy to have the following situation:

* **The same male, but there may be people who enter Male, some people who enter male, later it will cause a lot of difficulties in filtering.**.
* **Next is to prevent users from entering a value that is not male or female, to avoid creating error data**.

And we have the line of code below

**new\_student\_gender = menu('please select gender', 'male', 'female');**

**switch new\_gender**

**case 1**

**new\_student\_gender = 'male';**

**otherwise**

**new\_student\_gender = 'female';**

**end**

A little code explanation:

* **We create a variable named gender and there display menu to choose between male and female**.
* **If male or female, use switch case and save that value in new gender**.

The last thing is just to save it to the struct:

**struct\_form(n).No = n;**

**struct\_form(n).FullName = new\_student\_fullname;**

**struct\_form(n).YearOfBirth = new\_student\_year\_of\_birth;**

**struct\_form(n).PlacesOfBirth = new\_student\_place\_of\_birth;**

**struct\_form(n).Gender = new\_student\_gender;**

**struct\_form(n).Class = new\_student\_class;**

**struct\_form(n).StudentStatus = new\_student\_status;**

This is quite similar to creating a password account so it doesn't need much explanation.

* **But here we also want to change the class and status to a binary option like gender, because in the class BK with status there are only a few options to choose from.**.

=> The method will be the same gender, just different name.

Here we should think about how after entering a new student's name, it will still return to the loop unless another button is pressed.

**Next we deal with the export feature because it seems easier than deleting**.

* Here we want to increase the difficulty by allowing users to choose the file format they can export, that is, between excel and text file.

**data\_type = menu('please choose the data type you want', 'excel', 'txt file');**

* Regarding the export feature, the easiest export feature is to export as a text file.**So now we need to find a function that can be output as a text file**.
* I originally planned to use fprintf, but the fprintf was very annoying in arranging the data, so I decided to find another function,**maybe it has the option to export the file into different formats for example**.
* Then here we find a pretty cool function**writetable**. Here is a function that allows to write table to file,**That makes us pay special attention to the format that only allows tables**.

So we have to go back to the branch when choosing switch case = 2 that is when adding new students.

* When adding new students, we need to save that database, usually it will be saved in struct form. So here, we have to think of a way to convert this to**table**. After about 20 minutes, our code solution is as follows:

**table\_form = struct2table(struct\_form);**

**writetable(table\_form, 'student database.xlsx');**

The purpose of these two lines of code is as follows:

* When we create new student information, that data is temporarily stored as a struct.
* Then we convert it to a table form and save it to the table\_form variable.
* Here because the data is to be in the form of a table, it is easy to use the function**writetable**to save that new data into the original excel file I have available.

Then in the output file, it's quite easy. We just need to simply write the following:

**case 4**

**data\_type = menu('please choose the data type you want', 'excel', 'txt file');**

**switch data\_type**

**case 1**

**writetable(table\_form, 'student database final.xlsx', 'Sheet', 1);**

**case 2**

**writetable(table\_form,'student database final.txt','Delimiter',' ');**

**end**

Next we move on to the statistics section for the outgoing file because it is easier than deleting.

* **Here I will use the knowledge I learned in R programming and subplot to draw graphs**.

First I want to draw a pie chart showing % male and % female in this database.

* **First contacting R, I know that to work effectively with the table, we must first get the subset of that table.**. Since we only care about gender here, we will take the subset of gender out.
* **Then we will convert that table to array. Since matlab only works with arrays.**
* **Then from that array we just need to count the string pattern in it.**

c = {'car','tree','car','bag','horse','car','tree'};

nnz(strcmp(c,'car'))

[**https://www.mathworks.com/help/fixedpoint/ref/embedded.fi.nnz.html#:~:text=Description,-example&text=N%20%3D%20nnz(%20X%20)%20returns, fewer%20than%20232%20elements**](https://www.mathworks.com/help/fixedpoint/ref/embedded.fi.nnz.html#:~:text=Description,-example&text=N%20%3D%20nnz(%20X%20)%20returns,fewer%20than%20232%20elements)**.**

And after a while of searching and trying, we have the following line of code:

**gender\_subset = table\_form( : , 5);**

**array\_gender = table2array(gender\_subset);**

**number\_of\_male = nnz(strcmp(array\_gender, 'Male'));**

Explain the following line of code:

* First we get the subset of the table by simply specifying the range we want to get.**( : ,5)**which means we take all the rows of column 5.
* **Then when we have the subset, the next thing we do is just convert table2array using the built-in function**.
* After the conversion is done, we count the number of 'Male' in the array.**Here we use a combination of 2 functions**.
* The first strcmp function is for compare and will return 1 if it matches and 0 if it doesn't.
* Then we use function**nnz**means Number of Non Zero element, to count the number of non-zero values, here is to count the matching 1 value.
* After that, the rest is quite simple, do one more for female and draw a piechart and you're done.

**no\_female = nnz(strcmp(array\_gender, 'Female'));**

**pie\_chart\_data = [no\_male no\_female];**

**pie(pie\_chart\_data);**

Next we will add data to draw a regional histogram, for example, there are many code sentences for it.

* As we learned in R, there are two types of variables:**categorical variables**and**numerical variables**.
* Normally variable**string**would be a categorical, which is most evident in the histogram graph.

=>**So that requires us to convert our data from string to categorical variable**.

The transformation steps are as follows:

* We have to get a subset of the region in the table form of our data.
* **Next is to turn it from table form to array**. Here we try to convert it from table form to categorical but it doesn't work. In general, the array form is the basic form MATLAB works on.
* **Next from array to categorical using function categorical**.
* Finally use that variable to draw histogram and you're done. We have the following code:

**subplot(2, 2, 2);**

**x = linspace(0, 10, 100);**

**PlacesOfBirth\_subset = table\_form( : , 4);**

**array\_PlacesOfBirth = table2array(PlacesOfBirth\_subset);**

**categorical\_form\_of\_POB = categorical(array\_PlacesOfBirth);**

**histogram\_graph = histogram(categorical\_form\_of\_POB);**

Part of the fact that we know that histogram graphs have to work with categorical values ​​is that we go to the MATLAB documentation and we see

<https://www.mathworks.com/help/matlab/ref/matlab.graphics.chart.primitive.histogram.html>

**There is a sample graph, when it displays the number of yes and no votes, its data must be in the form of a categorical variable to be displayed.**.

=> After looking at the types of MATLAB Plots page, we see that there is an interesting plot called**geobubble**, it will roughly visualize data values ​​at specific geographic locations.

=>**So we decided that instead of using a histogram to show the distribution of geographic locations of Places of Birth, we decided to use a histogram to use for the year of birth and Places of Birth to use a geobubble graph.**.

To use a graph, we simply need to go to the documentation of that graph and that's it:

* The syntax of that code is as follows:
* **geobubble (tbl, latvar, lonvar);**
* **Where tbl is the table containing data to be plotted.**
* **latvar is a variable for latitude.**
* **lonvar is the variable for longitude.**

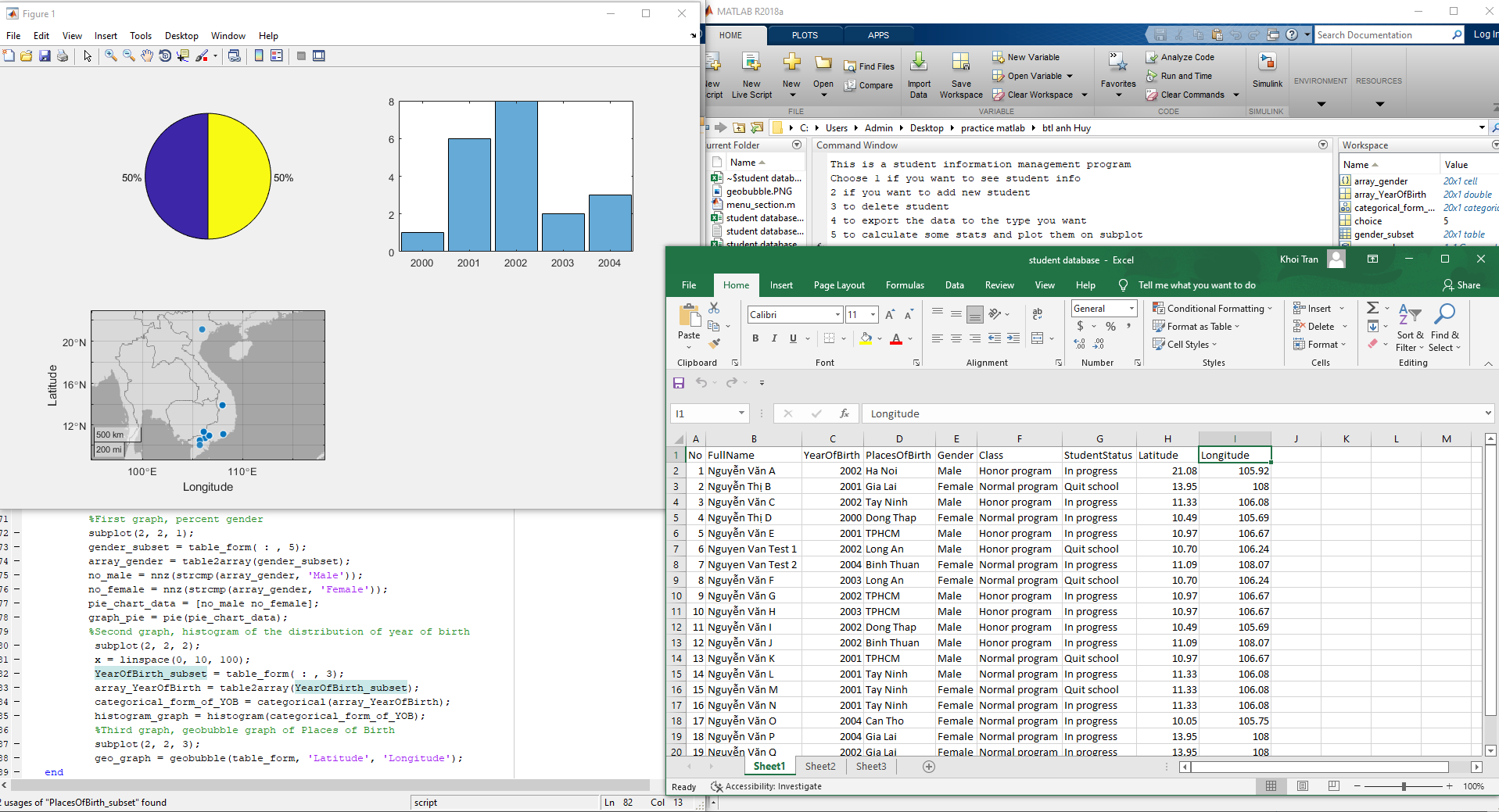
**=> Thus, we think that this graph can use data in the form of a table.**

**=> Thus, we do not need to perform complicated data transformation steps.**

In Matlab's website showing how to use geobubble graph, we see an example of how to draw a graph. Next to there is try this example, so we click. Specifically, the steps are as follows:

* Import our data as a table form, this we can use the readtable function.
* **After that, just use the geobubble function, specify the table form of our data, and 2 variables containing the Latitude and Longitude values.**.
* The code is as follows:

**geo\_graph = geobubble(table\_form, 'Latitude', 'Longitude');**



Although we have drawn a very nice graph as follows, but we want to go further.**I want the size of these bubbles to be proportional to the frequency of each province in this table data**.

* **So the next step we take is to look online to see if MATLAB supports creating frequency tables**.
* And we found a pretty cool function, called**tabulate**. The tabulate function has the following syntax:

**tabulate( X );**

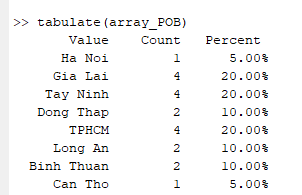
**Where x is a vector. So we have to add a step to convert data from table form to vector**.

**POB\_subset = table\_form( : , 4);**

**array\_POB = table2array(POB\_subset);**

**tabulate(array\_POB)**

And it shows the following result:



* We plan to use the percent as a way to specify the size.
* However, we remember one thing, that using this tabulate table will not work because it lacks latitude and longitude. While if using table\_form table will be missing the percent. So we have to think of how to add columns to the table together:

Code: Below is a piece of code for us to add to try. We intend to add the frequency of the provinces between the column student status and latitude.

**h = tabulate(array\_POB)**

**o = h( : , 3)**

**new\_table = addvars(table\_form, o, 'StudentStatus', 'Latitude')**

**It disp error is not same number of rows**. That's because when we count the frequency, the number of rows is essentially reduced. Because cells with the same province are collected for counting. So the error is not the same row is natural.

So we have to think of another way. That way is:

* **First calculate the frequency of each province, convert that into a table form.**
* **Next from the original table\_form variable, separate the latitude and longitude columns, continue to put in the frequency table. The goal is to group it together for the same number of rows.**
* **Finally, take those 2 columns and add them to the table frequency of the province.**
* **Finally use that table to draw geobubble and you're done.**

**POB\_subset = table\_form( : , 4);**

**array\_POB = table2array(POB\_subset);**

**fre\_of\_POB = tabulate(array\_POB);**

**table\_new = cell2table(fre\_of\_POB);**

**lat\_subset = table\_form( : ,8);**

**long\_subset = table\_form( : ,9);**

**array\_lat = table2array(lat\_subset);**

**array\_long = table2array(long\_subset);**

**fre\_lat = tabulate(array\_lat);**

**fre\_long = tabulate(array\_long);**

**lat\_value\_fre = fre\_lat( : ,1);**

**long\_value\_fre = fre\_long( : ,1);**

**new\_table = addvars(table\_new, lat\_value\_fre, long\_value\_fre);**

**new\_geo = geobubble(new\_table, 'lat\_value\_fre', 'long\_value\_fre', 'SizeVariable', 'fre\_of\_POB2');**

**Basically, the disp size is fine, but it's just the latitude and longitude difference.**That is, latitude and longitude do not match province. So here we have to rearrange a bit.

* **Sort by fall to big, by frequency**.
* The purpose is because the frequency of each province will correspond to the frequency of the coordinate pair. That is, for example, Ho Chi Minh City has the highest frequency, then the frequency coordinates of Ho Chi Minh City are also the highest. So we just need to arrange the frequency of the provinces from small to large, arrange the frequency of the coordinates from small to large.
* **Finally in the frequency of the coordinates, separate the column of coordinates and throw it into the frequency table and we will have a complete table**.

**POB\_subset = table\_form( : , 4);**

**array\_POB = table2array(POB\_subset);**

**fre\_of\_POB = tabulate(array\_POB);**

**fre\_of\_POB = sortrows(fre\_of\_POB, 2);**

**table\_new = cell2table(fre\_of\_POB);**

**lat\_subset = table\_form( : ,8);**

**long\_subset = table\_form( : ,9);**

**array\_lat = table2array(lat\_subset);**

**array\_long = table2array(long\_subset);**

**fre\_lat = tabulate(array\_lat);**

**fre\_lat = sortrows(fre\_lat, 2);**

**fre\_long = tabulate(array\_long);**

**fre\_long = sortrows(fre\_long, 2);**

**lat\_value\_fre = fre\_lat( : ,1);**

**long\_value\_fre = fre\_long( : ,1);**

**new\_table = addvars(table\_new, lat\_value\_fre, long\_value\_fre);**

**new\_geo = geobubble(new\_table, 'lat\_value\_fre', 'long\_value\_fre', 'SizeVariable', 'fre\_of\_POB2');**

* **Sort, there are also many types of sort**. The normal sort function we look for is sort array. That is, for example, we have the following array:

[ 2 5 8 1 4 3 ]

* When using sort array it will sort itself.
* However, since what we are working with is a cell array. So it can't use the normal sort. That's why we use the sortrows function. This function allows us to sort rows of matrix or table.

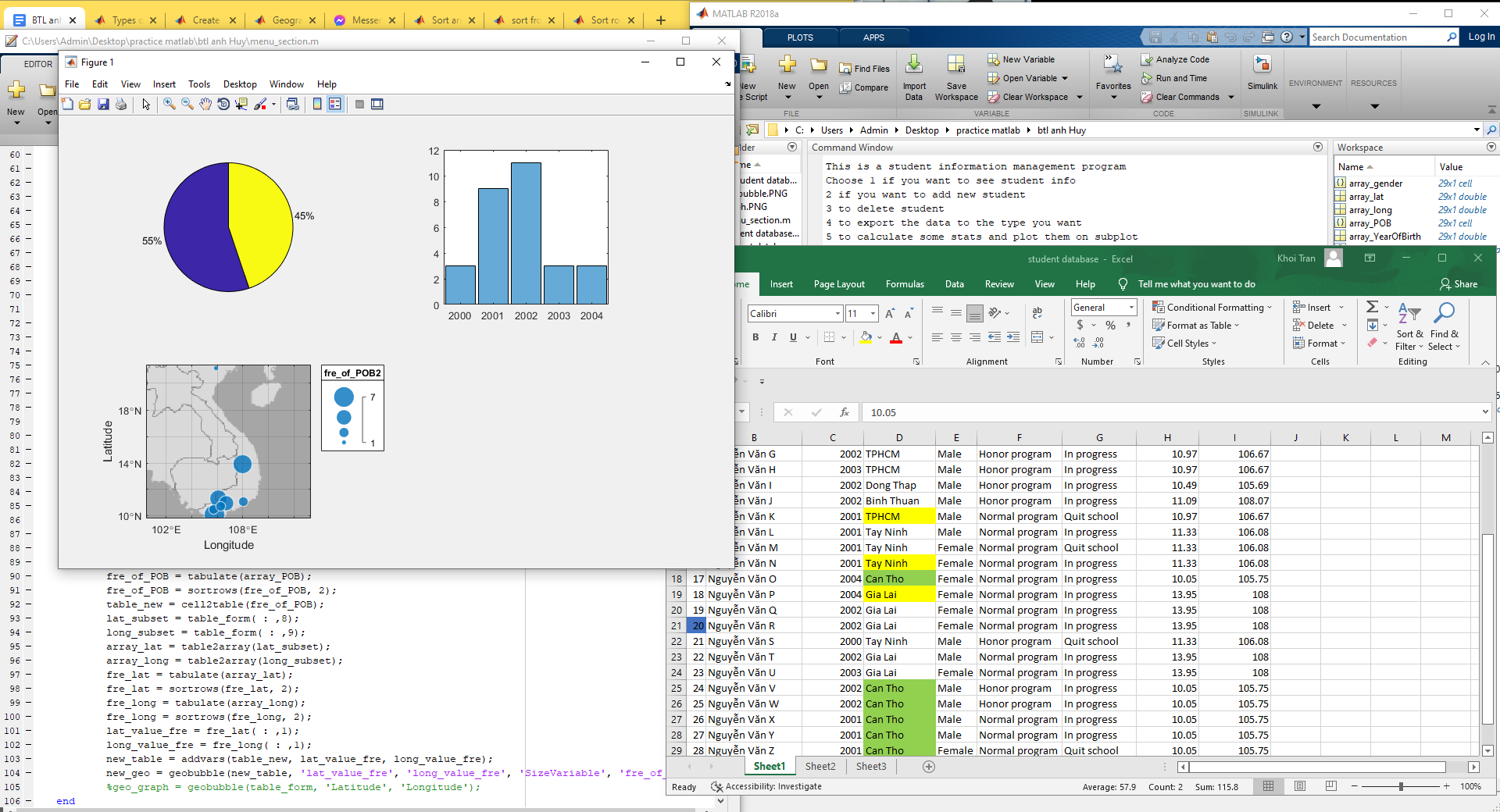
<https://www.mathworks.com/help/matlab/ref/double.sortrows.html>

* Syntactically it would look like this:

**sortrows( A, column )**

* With A being your matrix or table.
* **Column is the column I want to sort. It will always sort by value from smallest to largest. This is very similar to Excel**.

So we have finished the code bubble and it produces the following result:



**Now we will process to delete student data**.

We know the best way to remove a row or column from a matrix**is to set that row or column equal to a pair of empty square brackets [ ]**.

* So we think we can table it using the same way, and we search the internet and find the following way:

<https://www.mathworks.com/help/matlab/matlab_prog/add-and-delete-table-rows.html>

* **It's basically the same**. So this is how we think so that users can delete the rows they want.

**i = input('please choose row number you want to delete: ');**

**table\_form(i, : ) = [];**

Code explanation:

* **Basically we set i to be the row number that the user wants to delete.**
* **Then we just need to let i be a variable indicating the row position, then select all the columns and assign that row value to [ ] the empty set, and that's it.**.

However, we want to raise the level to a higher level:

**delete\_row\_options = menu('Please choose number of rows you want to delete', 1, 2, 3);**

* **That is, here we allow the user to choose the number of lines they want to delete**.

If they choose 1 line it's too easy.

**disp(table\_form);**

**r = input('please choose row number you want to delete: ');**

**table\_form(r, : ) = [];**

**disp(table\_form);**

This code is the same as the one above, we made a sample.

As for deleting 2 and 3 lines, we do the following code:

**disp(table\_form);**

**r1 = input('please enter the first row you want to delete: ');**

**r2 = input('please enter the second row you want to delete: ');**

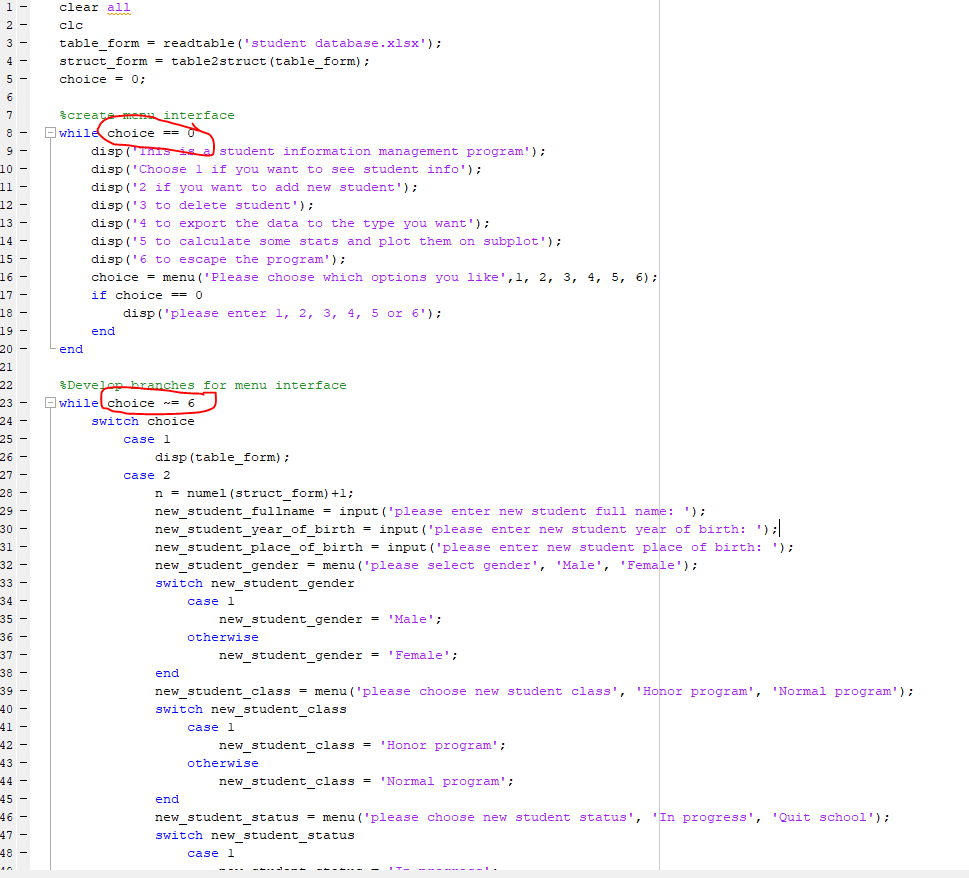
**table\_form([r1 r2], : ) = [];**

**disp(table\_form);**

The more lines you want to delete, set as many variables as r1, r2, r3,... and then put them in the matrix below.

* **Here, r1 and r2 will represent the lines that we want to delete, then it will act in the line of code below as a number indicating the number of lines we want to delete (assign null values) to it.**.

**Finally, we set the program to press button 6 to escape**.



* The reason we have 2 while loops is because 1 while loop == 0 is to prevent the user from pressing the X.
* The while loop ~= 6 is to force the user to have only one way to end the program, which is to press the number 6.