ENGI 1331H Project 4 Spring 2019

MATLAB Final Project

Due @11:59am on Friday, April 19, 2019

For any questions regarding this project please visit or email the TAs listed on Blackboard.

I. Overview

In this final MATLAB project, you will be working extensively with GUIs and an Arduino. Section 1 will focus on GUIs and how to make them in MATLAB, Section 2 will focus on setting up the game circuit, Section 3 will focus on communicating with the Arduino using MATLAB, and Section 4 will focus on creating a GUI that sends information to the Arduino.

II. Guidelines

Question: What amount of "collaboration" with fellow students is allowed?
Answer: You are encouraged to verbally discuss the project with other students. You can brainstorm together solution approaches, and you can teach each other how to do things with MATLAB. Allowed collaboration ends with this verbal discussion. At no time can you copy work others have done or have someone else do any of the work for you, or do any of the work for someone else. "Copying" includes obtaining an electronic copy; simply looking over someone's shoulder and writing down the MATLAB commands; or verbally exchanging the MATLAB commands. Everything in the MATLAB script file itself must be your work and your work alone. If you need more help, ask your TAs or instructor for assistance.

ANY VIOLATION OF THIS OR OF THE ACADEMIC HONESTY POLICY WILL BE PUNISHED BY A TWO LETTER COURSE GRADE DEDUCTION. YOU WILL NOT BE ALLOWED TO WITHDRAW FROM THE CLASS. ANY ATTEMPT TO WITHDRAW WILL RESULT WITH YOU BEING REINSTATED.

III. Procedure

Create a folder entitled uuuuup4 where uuuuu are all the characters of your cougarnet username. As you work on this project, save all the specified files in this folder. This should be set as your current working directory in MATLAB. When finished you will compress the folder and submit it to Blackboard Learn as instructed in the document "Turning in Your Work" under the Assignments section. Do not include any input data files in the folder you turn in as these values will be changed during grading.

Comment and display statements are an organizational feature for both you and others who may view the files. As projects and script files increase in length and complexity, these statements

have increasing importance. Use both comment and display statements as necessary/directed in this project.

Begin Section 2 by creating a script file named main.m. This file should begin with a title comment that includes your name, email address, and cougarnet ID. This information should also be displayed in the command window.

Additional comments should briefly describe the project and the important parameters. Be sure to put *clear*, *clc*, and *close all* as the first executable instructions in your script before the title display.

The computations will be divided into tasks described below. Work the problems and tasks in the order given below. Preceding each task, provide a comment with the task number, e.g. "Task 1", followed by suitable explanatory comments of what the task involves. The task number and explanatory comments should also be displayed in the command window when your script file is run. **This documentation is required! In other words, both your script and your output must be easy to read with each section well delineated.**

Put semicolons at the end of most command lines so that only those lines you specifically need to display will be displayed. Any text displayed in your command window should be done either using some sort of display or write command.

In the Project 4 assignment on the ENGI 1331H website, along with this assignment document you will find files containing data to be loaded. Copy these files to your uuuuup4 directory, as that is the first place that MATLAB will look for them. However, your solution must be kept sufficiently general so that if these values are changed, or a file with different values is loaded, the script will still execute correctly—without anybody making additional changes. We will execute your script with a different set of values!

Your main.m should have comments, blank lines, and section breaks (%%) in a format comparable to what is shown below:

```
%% ENGI 1331H Project 4 – your name – your cougarnet ID – email address
clc; close; clear;
disp('ENGI 1331H Project 4 – your name – your cougarnet ID – email address');
disp('')
%% Section 3 – Programming the Arduino
disp('Section 3 – Programming the Arduino');
disp('')
```

```
% Task 3.1 – Set Up Arduino Connection
disp('Task 3.1 – Set Up Arduino Connection');
code;
disp(‘’)

% Task 3.2 – Create UpdateBoard() Function
disp('Task 3.2 – Create UpdateBoard() Function');
code;
```

IV. Helpful Tips

In past projects, many points have been lost when students make simple mistakes they could have avoided from reading the instructions. We’re putting some helpful hints here so you can get the most out of your project submission.

Getting Help: This project is due April 19, 2019. You will likely need assistance with at least the last section of this project. Begin work well before the due date so you can seek help when you require it.

Zipping: Make sure you know what you should and should not include in your .zip submission folder. Placing in images and input data will result in lost points.

Organization: Clear organization is expected for project submissions in higher level courses and in industry. Learning how to create a polished project is a valuable skill. You should be proud of how your project looks!

- Put **clear**, **close**, and **clc** commands at the beginning of your script
- Use display statements and pauses between tasks
- Put semicolons at the end of most command lines

Section 1 - Introduction to GUIs

The purpose of this section is to introduce you to the basics of Graphic User Interfaces (GUIs), as you will be creating a GUI in the final section of this project. For this section, you will be building a GUI that graphs a sine or cosine function based on user-entered parameters. An example of the finished GUI is provided at the end of this section. It is strongly recommended that you go through the PowerPoint on GUIs before starting this section. You will need to create a .mlapp file for this section. Your completion of this section will be graded as a checkpoint on April 9th for the TTh section, and April 10th for the Wednesday section, but you will still need to include the .mlapp file for this section in your final project upload.

Task 1.1:

Open MATLAB, and navigate to create a new App. This will open up MATLAB's App Designer, where all your work for this section will be done. Save this file as section1.mlapp.

Task 1.2:

Add a button, axes, and a drop down from the components tab to your main app layout. Each component on your layout has certain properties which can be found and edited in the component properties tab. Edit the properties of the drop-down component so that it has two options: sin and cos. The button will be used to plot the graph once your parameters have been entered.

Task 1.3:

The components that you have added to your layout do not have a section of code designated to them. First you need to add a callback function that triggers when your button is pressed. Go to code view and create a callback function for your button component, and a new section of editable code should appear. Write code that will plot a graph on your axes component based on which function (sin or cos) was selected in the drop down component. The graph should show the function on the interval $[0, 4\pi]$. Note: The x axis will have values beyond 4π, do not worry about this. The sin function follows the formula $y = A\sin(Bx + C)$, where A is amplitude, B is frequency, and C is phase shift.

Hint: In order to reference the properties of components, you must use the following format: **app.ComponentName.PropertyName**. Use the plot(axes, X, Y) function to make your graph. Remember that when referring to other variables and components, you must add the prefix "app." before the name.

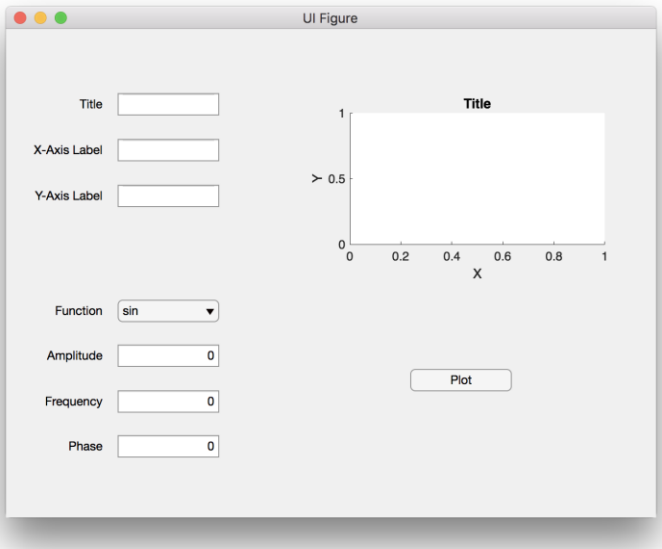
Task 1.4:

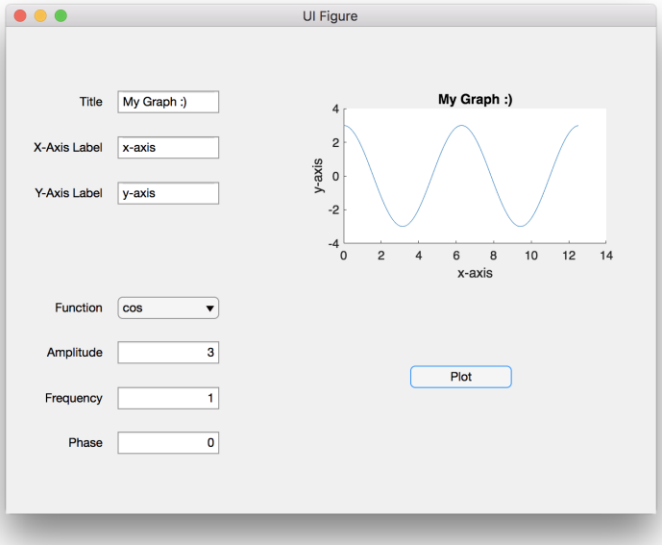
Now you will add the ability to adjust the amplitude, frequency, and phase angle of the graph. Add three edit field (numeric) components to your app. Edit the callback function from the previous task so that the graph will reflect the initial input parameters when your button component is pressed.

Task 1.5:

Now you will add the ability to set the title, x-axis label, and y-axis label of the graph. Add three edit field (text) components. Again, edit your callback function so that the title and labels of the graph will update when your button is pressed.

The first picture shows the app before any parameters have been entered; the second shows the app after parameters have been entered and the Plot button has been pushed. Note that this is just an example; your GUI does NOT need to look exactly like this one.





Section 2 - Arduino Hardware Setup

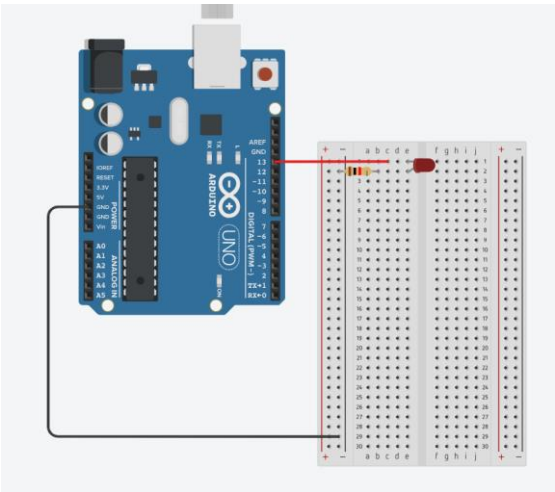
You will not be writing anything in your main.m file for this section. Any MATLAB coding done in this section should be done using your command window. You will only need to code in your main.m file for Section 2.

Task 2.1:

Make sure you have all of the required materials for the project. If you are missing something, you need to buy it ASAP or ask the TAs for a replacement. If you bought the Arduino Starter Pack, you should have everything you need. You will also need to download the MATLAB Arduino toolkit.

Task 2.2:

Build a small circuit including one LED light and a 1 kΩ resistor on one side of your breadboard. Below is a sample of what one LED circuit should look like.



Task 2.3:

In the command window of MATLAB, create your Arduino object using the arduino() function. Use the writeDigitalPin() command to turn your LED light on and off.

Task 2.4:

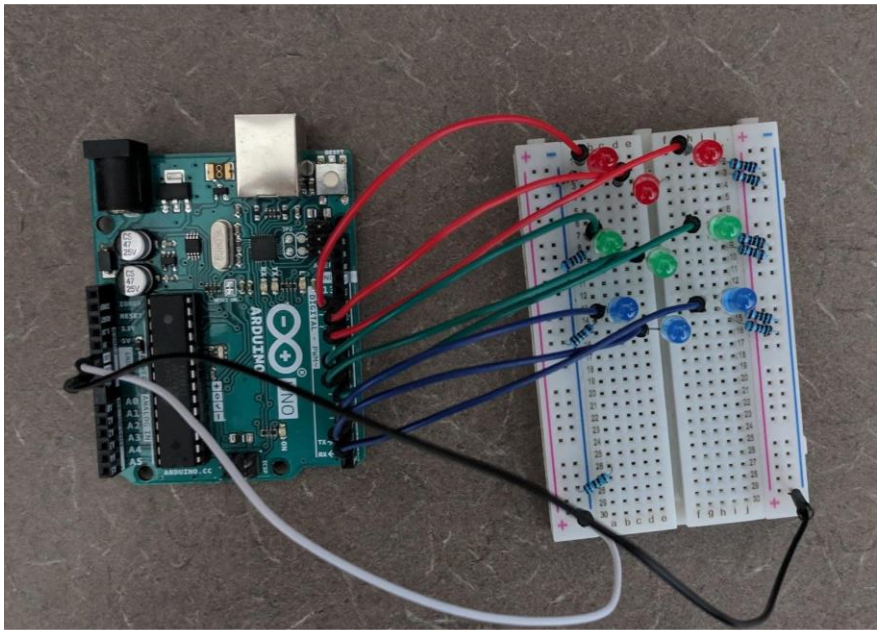
Use your breadboard to create a 3x3 ‘grid’ of LED lights. They do not need to be perfectly in place, but it should be clear that the lights form a grid. Each light must be on its own circuit so that you can control separately from the other lights. In total, you will have 9 small circuits each with an LED light. Below is a sample of what the completed 3x3 LED grid could look like. It is

important that you follow the standard pin connection setup for grading purposes. This pin setup should look like this on your breadboard:

[‘D12’ ‘D11’ ‘D10’

‘D9’ ‘D8’ ‘D7’

‘D6’ ‘D5’ ‘D4’]



Task 2.5:

In the command window, create your Arduino object and test every LED using the writeDigitalPin() function. You must visit one of the TAs by April 16th if you are in the TTh section or April 17th if you are in the Wednesday section so that they can check your hardware setup. It is recommended that you DO NOT WAIT until the deadline to check this part. We also encourage you to start working on the next sections as soon as possible, even if your hardware has not been checked yet.

Section 3 - Programming the Arduino

In this section you will be creating a main.m file that in which you manipulate the LEDs in various ways. Keep in mind that the LEDs should be off unless the task requires you to light them up. After every task is complete, the LEDs should all be off and your program should pause before continuing. Make sure to save frequently and save often so that you do not lose your work.

Task 3.1:

In your main.m file, create an Arduino object and assign it to your Arduino. Create a 3x3 matrix called *pins* containing the pin labels for each LED. The position of these labels in the array should correspond to where the LEDs are on your breadboard (the values in this array should be strings such as 'D3'). This array should look like this:

['D12' 'D11' 'D10'

'D9' 'D8' 'D7'

'D6' 'D5' 'D4']

Create a second 3x3 matrix that represents the status of the LEDs on the grid using the values 0 and 1. Store this matrix in a variable called *board*. It does not matter what values are stored in the matrix *board* in this task.

Task 3.2:

Create a user defined function UpdateBoard() that "updates" the LEDs on your breadboard in order to reflect the values in the *board* matrix. This function should only require a single for loop and should have no output. Make sure that this function is working properly, as you will use it in almost every other task. For this task, you need to make sure that an LED will turn **off** if its corresponding element in *board* is set to **0**, and **on** if that element is set to **1**.
Hint: You will have to have your Arduino object as one of your inputs.

Task 3.3:

Now that you have all required code, you should be able to manipulate the LEDs of your Arduino easily. Use a loop to make all of the LEDs blink simultaneously three times (turn on for 1 second, then turn off for 1 second). It is ok if there is a minor cascading effect when the LEDs turn on and off.

Task 3.4:

Read in **light_sequence.csv** to your main.m file. This file contains a 3x3 matrix of the integers 1-9 positioned randomly. Make your LEDs light up in an order that follows the numbers in the matrix. For example, if the value in (3,2) of the data matrix is 6, then the LED in the third row and second column should be the sixth LED to light up. A new LED should light up every 2 seconds.

Task 3.5:

Ask for the user to input a row number and column number corresponding to a position on the grid. Light the selected LED up. Then make the 8 remaining LEDs light up in a random order, with a new LED lighting up every 2 second until all the LEDs are lit. This random order should be different each time the program is run.

Task 3.6:

Create a user defined function RandLight() that will light up a random number of LEDs (1-9) in random locations on your breadboard. Your function should display the randomly selected number of LEDs each time it runs to the command window. Call the function 5 times, with a 5 second delay between each iteration.

Task 3.7:

Ask for the user to input a row number and column number corresponding to a position on the grid. Light the selected LED up. Make the program wait 5 seconds and then simultaneously light up all LEDs directly adjacent (diagonals don't count) to a currently lit LED. Repeat this process until all the LEDs are lit, with a 5 second delay between each iteration.

Section 4 - Creating a Graphic User Interface

In this section, you will create a game of battleship using MATLAB App Designer. Battleship is a game where a player guesses the location of several hidden ships until they "hit" all of the ships or until the number of guesses they have runs out. In this game, the player will be guessing the location of randomly arranged ships on the 3x3 LED board you made in the previous section.

A GUI consists of components that a user can interact with to perform tasks. Most components in your GUI will execute a specific "callback" function when the user interacts with it, which you will code in App Designer. App Designer uses "properties" instead of variables. If you have a property called *position* to store the location of an element within an array, you can get this value by typing *app.position* instead of *position*. Each component in the GUI has its own unique properties as well - for instance, the value displayed by a spinner component is stored in its *value* property, and you can get this value by typing *app.spinner.value*. (Be careful - not all spinners have the name *spinner*. You can name your components anything you want when you create them - make sure you use the correct name of the component you are getting a value from). To be able to store values in your own properties, you must initialize them in the properties section. To initialize a property, type its name in the properties section. If you want a property to start out with a specific value, set the property equal to that value and do not put a semicolon at the end of the statement.

Rather than provide step by step instructions for tasks in this section, you will simply have requirements that your game must meet. How you specifically code your program is up to you.

REQUIRED FEATURES

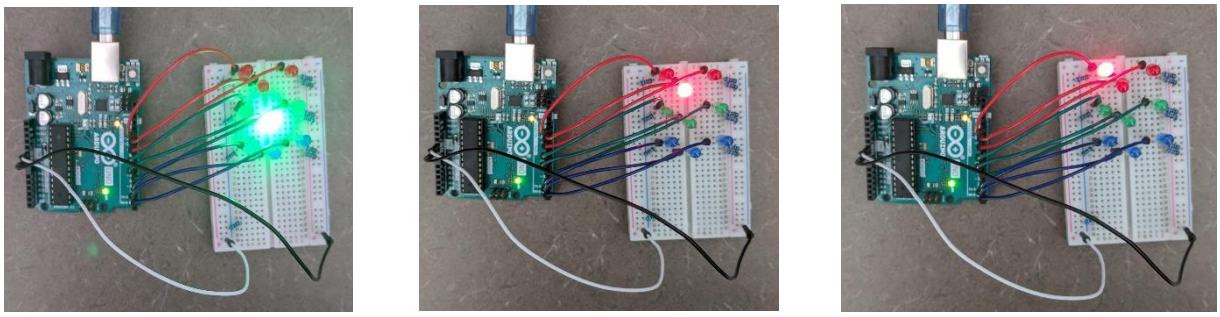
- Parameter Setting Spinner Fields
    1. A 'Number of Ships' spinner: This spinner should be used to select the number of ships before the game starts. It should only range from 1-4 ships.
    2. A 'Number of Guesses' spinner: This spinner should be used to select the number of guesses the player will have before the game starts. It should only range from 1-9 and the game should not begin if the number of ships is greater than the number of guesses.

- Game Buttons
    1. A 'Start Game' button: This button will begin the game with the parameters you currently have selected. No lights should be on and clicking any other buttons (except those that set the parameters) should do nothing before this button is pressed. After pressing this button, the cursor LED should light up in the middle of the 3x3 grid.

2. A ‘Quit Game’ button: This button will turn off all of the LEDs. It is not required that the value of the spinners be reset back to one. The start button should start the game again after the Quit Game button is pressed. Pressing Quit Game essentially resets the GUI as if you just opened it. Pressing any button except for start after pressing Quit Game should do nothing.

- Game Status Info Fields
    1. A ‘Shots Remaining’ info field: This field will display the number of shots remaining at any given point in the game. When the start button is pressed, this field should be set equal to the given initial parameter. The number in this field should decrease by 1 every time a guess is made. When this number reaches 0, the game should end immediately.
    2. A ‘Ships Remaining’ info field: This field will be filled in with the number of ships remaining at any given point in the game. When the start button is pressed, this field should be set equal to the given initial parameter. The number in this field should decrease by 1 every time a ship is hit. When this number reaches 0, the game should end immediately. If the number of ships and the number of guesses reaches 0 at the same time, it counts as a win.

- Movement/Selection Buttons
    - Note: Once the game starts, you will have 1 LED selected at a time. This LED represents the position of the cursor, which should “move” accordingly in response to your movement commands. The player should be able to see the currently selected LED. The easiest way to do this is to make the LED light up if it is selected.
    1. Move left/right/up/down buttons: These buttons should move the position of the LED cursor one spot in the corresponding direction. <u>If the movement would move the selection off the 3x3 grid, the LED cursor should stay in place.</u>



The cursor starts at the center, then we move up and to the left. The currently lit LED changes in response to our commands.

2. A ‘Make Shot’ button: This button will check if there is a ship at the selected location and update the info fields appropriately. Make sure the game remembers the guesses - a user should not be able to guess the same location twice.

- Game End Responses
    1. Victory response: When the last ship is destroyed, the board should blink in a special pattern celebrating the player's victory. The GUI should also print a display statement in the MATLAB command window saying the game has ended in a win. (You can choose this pattern, but make sure you describe it in comments)
    2. Defeat response: When the last shot is used (and there are still ships remaining), the board should blink in a special pattern commemorating the player's failure. The GUI should also print a display statement in the MATLAB command window saying that the game has ended in a loss. (You can choose this pattern, but make sure you describe it in comments)

- Hit/Miss response
    1. Make 2 lamp components that display whether a guess was a hit or a miss. A lamp should only glow after a guess has been made on that specific location. The lamps should also remember past guesses (If you make a guess at a specific spot, and then leave that spot and come back to it, the lamp should still display the information from the initial guess). The game should NOT allow you to guess the same location twice - we will check for this!

THINGS TO KEEP IN MIND

1. uiwait and uiresume are functions that temporarily pause and resume the program. You will need these to keep the LED selection light on.
2. If the number of ships and the number of guesses reaches 0 at the same time, it counts as a win.

Extra Credit

1. Make 2 extra lamps that display if a guess is close to or far away from a ship. A guess is nearby a ship if any adjacent LEDs are ships, not including diagonals. Otherwise, a guess is considered far away.
2. Add a text field that displays any error on the user's part if they try to do something that is not allowed (moving the cursor off the grid, starting with more ships than guesses, etc.). We recommend using a multiline text field for this.

Final GUI Example