P3: Special Functions, User-Defined Functions, Plotting Areas, and Compatibility Test

**Due Wednesday, March 30th at 11:59 am**

## I. Overview

In this third MATLAB project, you will learn how to apply user-defined functions to MATLAB problems, explore how to use MATLAB to do calculus, find the spectral angle of a vector, and determine whom you’re most compatible with.

## II. Procedure

Question: What amount of “collaboration” with fellow students is allowed?

Answer: You are encouraged to discuss verbally the project with other students. You can brainstorm together solution approaches, and you can teach each other how to do things with MATLAB. However, allowed collaboration ends with this verbal discussion. At no time can you copy work others have done, 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, or simply looking over shoulder and writing down the MATLAB commands, or exchanging verbally 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 proctor or instructor for assistance.

Question: Can I use MATLAB functions or operations we have not covered in class?

Answer: In this particular project, you can use explicit loops (for or while) or the “If” statement. MATLAB’s implicit looping (array operations & intrinsic functions) is allowed. In addition, you cannot use anything from the MATLAB toolboxes (e.g., the symbolic toolbox). Only the standard/basic MATLAB package is allowed.

Create a folder entitled uuuuup3 where uuuuu are all the characters of your cougarnet username. As you work on this project, save all of 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.

Begin 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. 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.

Some of the tasks will require you to produce plots. Be sure you follow the instructions—display an appropriate "prompt" in the command window after your plot command (e.g., "See Figure 1 for plot showing Waldo, original file. Press any key to continue.”), followed by the pause command. 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 you command window should be done either using some sort of display or write command.

In the Project 3 assignment on the ENGI 1331 website, in addition to this assignment document you will find plain text files containing data and a function file to be loaded in some of the tasks of this assignment. Copy these files to your uuuuup3 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 different file 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 M-file should have comments and blank lines as shown in this example:

% ENGI 1331 Project 3 – your name – your cougarnet ID

clc; close all; clear;

disp('ENGI 1331 Project 3 – your name – your cougarnet ID');

% Problem 01 Area Between Curves

disp(' '), disp(' '), disp('Problem 01 Area Between the Curves’'), disp(' ')

Handling Images: For those problems that require an image, each image should be numbered using the figure(n) command, as specified in the problem, UNLESS OTHERWISE SPECIFIED. Prior to each image, use the disp command in your script file to indicate in the command window that a figure is being displayed. Be sure to include the pause command AFTER your plot. For example:

disp('See Figure 1')

figure(1); plot(x,y); pause;

# Problem 1- Plotting Functions, Intersection Points, Area

For this problem, you will be plotting the area defined by multiple functions intersecting to create enclosed spaces. You will import a data file containing the coefficients of a function which you need to define in MATLAB such that you can apply the **fminbnd**, **fzero**, and **integral** functions. You will also need to calculate the area using 3 different Riemann sums techniques to compare with the MATLAB computed result. Finally, you will produce a plot of filled areas.

The MATLAB function fminbnd() finds the minimum of a function of one variable within a fixed interval. x = fminbnd(fun,x1,x2) returns a value x that is a local minimizer of the function that is described in fun in the interval x1 < x < x2. The variable fun is a function handle.

The MATLAB function fzero() calculates the closest root of a nonlinear function. x = fzero(fun,x0) tries to find a point x where fun(x) = 0. This solution is where fun(x) changes sign. fzero() cannot find a root of a function such as x^2.

The MATLAB function integral() evaluates definite integrals of a function. q = integral(fun,xmin,xmax) numerically integrates the function fun from xmin to xmax using global adaptive quadrature and default error tolerances. The variable fun is a function handle, xmin and xmax are scalar values. The output q is the number that results from evaluating the definite integral.

## Task 1:

Download the files “coefficients.txt” and “MyFun.m” from the course website and store them in your current working directory. Put in your script file a command that will import “coefficients.txt” into MATLAB as a vector. We will run your script with different data, and the size of the vector may change. The vector contains i elements that correspond to the coefficients of a polynomial. This is your first function:

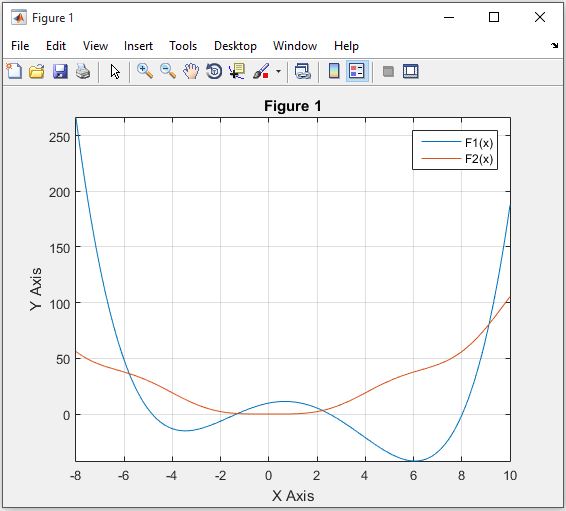
Where F1 is an nth order polynomial and the coefficients Ci, Ci-1, … C1 are the elements of the imported vector. Create an anonymous function using polyval() for F1 in your script.

Note: We will run your script with different data so the order of the polynomial may change.

Your second function F2 will come from the user defined function called “MyFun.m”. Do not hard code this function in your script. We will test it with a different (but similar) function. Create an anonymous function for F2.

You are trying to analyze the functions to calculate the enclosed areas between the curves. You will compare these values with a Riemann sum, and you will also display which intersection point is furthest in terms of vertical distance to the minimum of F1.

For this task, plot F1 and F2 on the same graph over the domain -8 to 10 as Figure 1. Remember to use the figure() command. Include a legend, title, and axis labels. Figure 1 should look similar to the following:



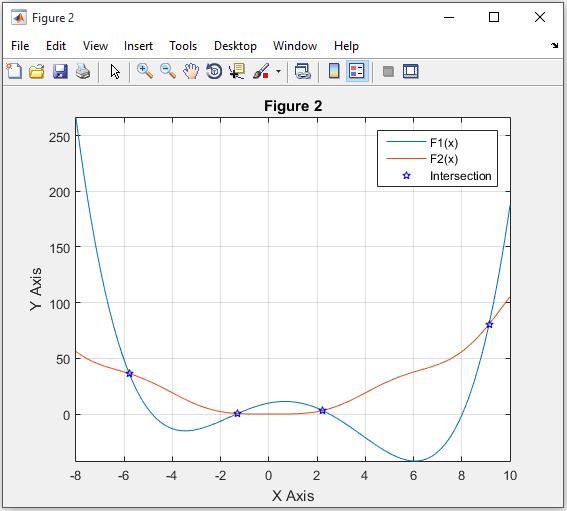
**Task 2:**

For this task, you will create a user-defined function to calculate all the intersection points of your two functions. You will need to use the fzero command. The original functions given have 4 intersection points, but your script should work for any reasonable number of intersection points within the given domain (-8 to 10).

To do this, use fzero inside a **for** loop over the domain to ensure that you find all the intersection points. You will find some more than once, which means you need to remove the duplicates. Due to the inconsistency of fzero, you will need to round your values from fzero before removing duplicates. Using the MATLAB’s built in function unique() may help you with this task. Test your user-defined function with different functions to ensure that it works for different intersection points.

In your main script, store the x values of the intersection points in a vector called “x\_int”, and the y values in a vector called “y\_int”. Clearly display these values in the command window.

Reproduce Figure 1, but this time include markers for all the intersection points. Call this new plot Figure 2. Remember to use the figure() command. Include a legend, title, and axis labels. It should look similar to the following:



**Task 3:**

Next, you should calculate each area defined between the intersection points using the integral command (make sure to not hardcode your program and expect the intersection points to change). The number of enclosed areas may change with the data we use to test your script.

For the Riemann sums, you should calculate the Left Hand, Right Hand, and Midpoint sums using 4 rectangles. To define your rectangle limits, divide the width of the intersection points by the number of rectangles. The Riemann sums should be calculated in a user-defined function, one that you can call upon within your code.

For each area, clearly display in the command window the value calculated using integral and the value for each Riemann sum.

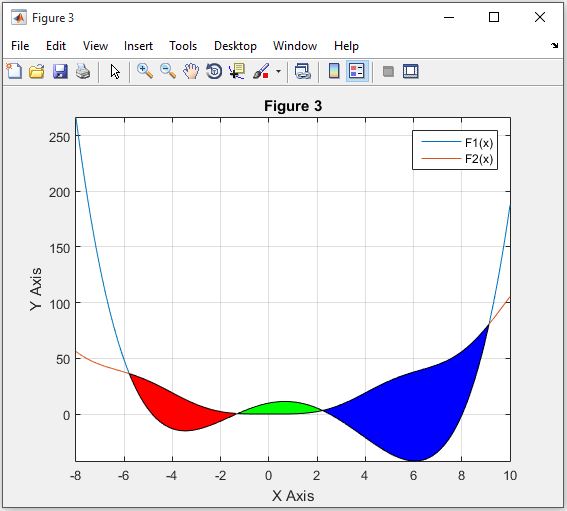
**Task 4:**

For this task, you should use fminbnd() to calculate the minimum of F1 and compare that value to the intersection points. Determine which intersection point on each side (left and right) is furthest from the minimum in terms of vertical distance (y-value).

You will need to display which intersection point is furthest from the minimum of F1 for each side (right side being towards infinity, left side towards negative infinity), and display in the command window the (x,y) coordinates associated with both.

**Task 5:**

For this task, create a filled plot of the areas using the fill() command, where each area is filled with a different color. The fill() command must be called on once for each area. It is best to do this within a **for** loop. Give each area a different color by indexing the text string 'rgbymck' to select a different color specifier for each area. Use the hold on and hold off commands to accomplish this. Remember to use the figure() command. Include a legend, title, and axis labels. Reproduce the plot shown below as Figure 3:



Remember, when we grade your project, “coefficients.txt” and “MyFun.m” will be different, resulting in different functions. The number of intersection points may change and the number of areas may change. Your code should be robust enough to handle this. To keep this within a reasonable limit, you may assume that there will be at least one and no more than seven enclosed areas. We suggest that you rigorously test your script with different functions for F1 and F2 to ensure that it is sufficiently general.

Note:

Remember to submit your user-defined functions used to calculate Riemann sums and Intersection points with your main script in your project folder to receive credit.

# Problem 2 - Compatibility Test

In this problem, you will use the results from the survey you all took to find whom you are most and least compatible with your section and in all the sections combined. We have numerically quantified the results of the survey by assigning each answer a number, and now you will analyze the data. This will be done by calculating the spectral angle and the Euclidean distance between your answers and the answers of your classmates.

Spectral Angle Mapper (SAM): Spectral angle is a measurement of similarity between two vectors and .

Where is the spectral angle between vector and , refers to the Euclidean norm and refers to the inner-product between vector and .

The Euclidean distance measures the difference between vectors and .

Where is the Euclidean distance between vectors and , and refers to the Euclidean norm.

**Task 1:**

For this first part, you will download the file “survey\_answer.csv” from the course website, and load it into MATLAB with the variable name “survey”. In this matrix you will find the numerical representations of each student’s answers to the survey. Each student has also been given an identifier number. You can find your identification number in the document “ID\_numbers.xlsx”. The structure of the data is as follows: the data will be arranged in a matrix with each column vector assigned to a specific student. There are N columns in each matrix. We call the columns vectors V1…VN. The first element in V is the student’s identifier number and the rest of the elements are that student’s numerical answers to the survey.

|  |  |
| --- | --- |
|  | IN |
|  | a1,N |
| VN = |  |
|  | a9,N |
|  | a10,N |

Where a1,N … a10,N are the Nth student’s numerical representations of their answers to the 10 survey questions, and IN is that student’s identifier number.

Ask the user to input an identifier number. Use their input to extract that students numerical answers from either data set. Then, display their identifier number and answers to the survey. Nothing should be hardcoded throughout this problem. It should be completely based off of initial ID which the user inputs. Meaning that your code should work for any user ID that the grader inputs, not just yours. We will grade it by inputting a random ID number.

**Task 2:**

Write a user defined function “SAM.m” that takes two vectors A and B as inputs, and outputs the spectral angle between vectors A and B. You will use this function in the next task to calculate the spectral angle between the student’s answers and everyone else’s answers. This will tell you how compatible the student is with each person. The smaller the spectral angle, the more compatible. A larger spectral angle means less compatible.

Write another user defined function. This one will be called Euclidean.m, and it will take two vectors A and B as inputs and output the Euclidean distance between the two vectors. You can use this as a measure of similarity between the two vectors, where a smaller distance means more similar.

**Task 3:**

Use both the spectral angle and the Euclidean distance to determine who the student is most and least compatible with in their section and overall, and display the identification numbers of those students. You should display 8 ID’s here. Four for each method: most compatible overall, most compatible in his/her section, least compatible overall, and least compatible in his/her section. It is important to clearly display these in the command window so that the grader can easily determine which is which. We want to give all the points you deserve, so make your answers easy for us to find.

The identifier number will allow you to determine who is in the same section as the student and who is not. There are four sections of ENGI 1331H, and each have been given a range of values. View the table below:

|  |  |
| --- | --- |
| **Instructor** | **Range** |
| McCave | 100-199 |
| Hazlett | 200-299 |
| Claydon | 300-399 |
| Sappington | 400-499 |

For instance, if the ID number is 101, then that student is the first student in McCave’s section. If the ID is 312, then the student is the 12th in Claydon’s section. This information will be useful to you when calculating the most/least compatible person in the same section as the student.

Note:

When you calculate which section the student is in; you may want to round the ID number down. Look up how to use MATLAB’s floor function.

Be careful not to include the identifier numbers in the similarity calculations, and not to calculate the angle/distance between the students answers with their own answers (because you are obviously most compatible with yourself). Test your script by inputting your identification number and other identification numbers. If you do not know your identifier number, please check the spreadsheet on blackboard. Your code should be general enough to work for whichever identification number that we choose to input. We will input a random ID number when we grade it.

Finally, your code should display answers to the following questions. Do the two methods give different results? Which one would you recommend as a measurement of similarity between two vectors?

Congratulations! You are finished with project 3!

Turn in your Project:

Follow the instructions on Blackboard. **Be sure to include your main script along with all user defined functions.** Donot include data files or the function “MyFun.m”