EGR 141: Project 2

Summary: The goal of this project is to practice all of our skills up to this point. Now that we have general functions, loops, and if statements, we can create just about about anything. Do not use any concepts we have not covered yet, such as advanced string operations, character array operations, or cell arrays.

- Each of the following problems should have a script and, possibly, a function associated with them.
- For each problem, the script file should be called something appropriate, such as *Proj2_1_yourName.m*
- Include any functions that you needed to create in order to complete the problem. Name them whatever is indicated in the problem.
- Inside your script, solve each of the given problems. In between each problem, type *pause;* Clearly indicate where the code for each problem begins by using a comment block. Start each new problem with a *clear .*
- If my example output "lines up nicely" then your output should as well.
- All output statements should output variables, not pre-computed constants. For example, if I ask you to output $r/2$ when $r = 3$, then you should set r to be three then output as *fprintf(' r/2 = %f ',r/2);* and not *fprintf('r/2 =1.5')* or *fprintf(r/2 = %f',3/2)*.
- Note that example output for each problem is not necessarily correct output (I intentionally change numbers so my answers will not always match your answers).

1. (4 pts) Create a script which asks the user to enter in two 10 digit integers, then displays the product of the integers to the screen. Sounds easy? Try this:

   *fprintf('%20.0f',1111111111*1111111111)*

   The issue is that the computer cannot store a long integer exactly. You need to find a way around this. Test your code by multiplying each pair of numbers:

   - 1111111111, 2222222222
   - 1234567890, 2345678901
   - 9999999999, 9999999999

   Your output should be a mathematical representation of the operation, along with the 19 or 20 digit representation of the answer (NOT a floating point approximation, exponential notation, or any format that does not appear to be an integer).

   Note: Your numbers must be inputted as floats/ints not as any other datatype (vectors, strings, chars, etc). Your code only needs to work for 10 digit integers, but it should work for *any* 10 digit integers.

```
Lab 6 - Multiplication
 10 digit integer multiplier
         11111111111
*        11111111111
---------------------
  123456790120987654321
```

2. (3 pts) Create a function, called *meanKs* that implements a **K-means clustering algorithm** for a set of 2-dimensional data (*(x,y)* values). The basic algorithm for separating given data into k clusters is as follows:

(a) Assume N-data points are given in vectors x and y (corresponding to (x, y) coordinates.

(b) Randomly choose k coordinates in your set to be your cluster centers. Store them in vectors *centerX* and *centerY* (the x and y coordinates of each center point).

(c) Find the distance between each point and each cluster center (so if there are c clusters, then for each N points we compute c different distances)

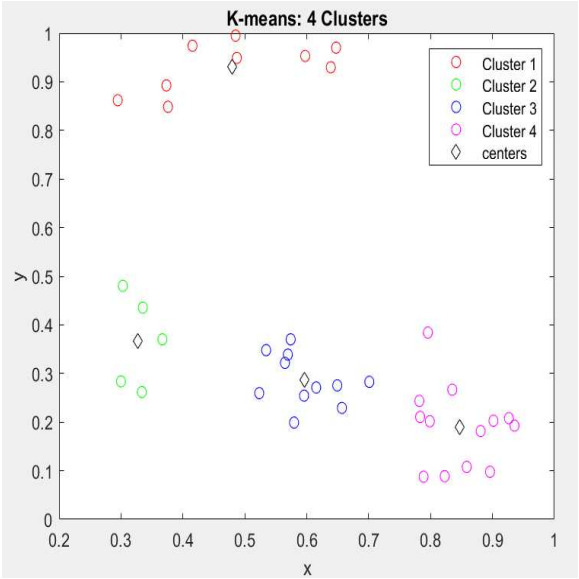
(d) For each data point, assign it to the cluster whose center is the minimum distance from the point.

(e) Recalculate the center of each cluster by averaging all coordinates in each individual cluster (average all points in cluster 1 to obtain the new center for cluster 1. Repeat for other clusters).

(f) Repeat, starting at step (c) until no cluster is reassigned (no data point changes which cluster it is in)

Your function should have the following as input: x, y, and a the number of desired clusters, k. Note that x and y have length N. Inside the function, run the above algorithm. The function should return the "cluster number" of each (x, y) point as well as the location of each cluster center.

In your MATLAB Script, test your function using the data given in the *kMeansData.txt* file, which has the x-values in the first column and the corresponding y-values in the second column. Run your function, then graph all data points. Indicate which cluster each point is in using color. Also plot the center of your clusters (note that your centers are not necessarily a data point).Your figure should have appropriate labels and a legend. Make sure your code works for anywhere from 2 to 6 clusters (it is okay if it works for more).



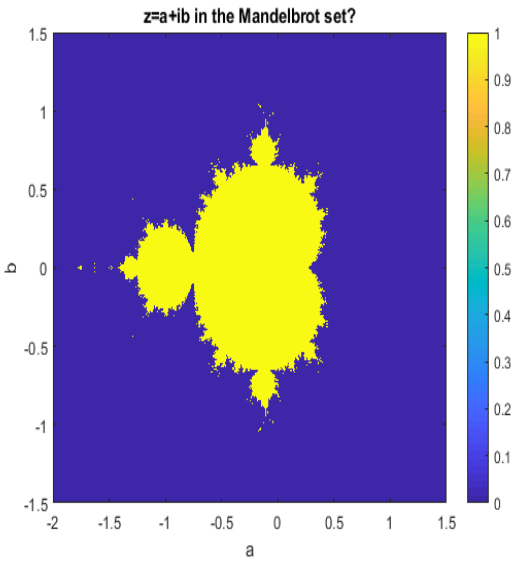
3. (3 pts) We say a complex number $c = a + bi$ is an element of the ***Mandelbrot Set*** if the sequence

$$z_{n+1} = z_n^2 + c, \; z_1 = 0$$

converges. In other words, it has the property that

$$\lim_{n \to \infty} |z_n| < M$$

for some number M. Put another way, the sequence given by z_n does not grow to infinity. If one lets $-2 \leq a \leq 1.5$ and $-1.5 \leq b \leq 1.5$, then plots a 1 if the element $c = a + bi$ is in the Mandelbrot Set and a 0 otherwise, then we find:



Note that we receive a surface, as a and b define a $2D$ domain. To simplify the process of determining if $c = a + bi$ is in the Mandelbrot Set, we

- Run the recurrence $z_{n+1} = z_n^2 + c$ for only 20 iterations. If $|z_n| > MAX$ for any n, then we declare $c = a + bi$ to *not* be part of the Mandelbrot set, so we plot a 0 at the $c = a + bi$ grid point. Otherwise, we display at 1 at the grid point. For the standard Mandelbrot set, we use $MAX = 2$.
- Once we determine if every point under consideration is in the Mandelbrot set or not, we plot the set using *imagesc*.

One can use a similar definition for other sets (Mandelbrot-like sets), although the theory of convergence is not as quite well defined. For your problem, perform the same process as above, except:

(a) Using subplot, make a 2×1 set of graphs. Use $MAX = 10$, 1024 uniformly distributed points in each direction (for each of a and b), and 20 iterations for the recurrence.

(b) On the first plot, use the iteration $z_{n+1} = \sin\left(z_n^2\right) + c$

(c) On the second plot, use $z_{n+1} = e^{\left(z_n^5\right)} + c$

A few notes/hints:

- Don't run for $N = 1024$ when testing you code. It will take far to long to test.
- $|z|$ for a complex number $z = a + bi$ is defined to be

$$|z| = z\bar{z} = (a + bi)(a - bi) = \sqrt{a^2 + b^2}$$

- You *will* be graded on how efficient your code is.