CSE 6010
Assignment 1
Solving Nonlinear Equations

Initial Submission Due Date: 11:59pm on Thursday, August 31
Final Submission Due Date: 11:59pm on Thursday, September 7
Submit to Gradescope through Canvas
48-hour grace period applies to all deadlines

Root-finding is a common scientific task where we seek solutions of nonlinear equations. Sometimes we seek solutions of multiple equations with multiple variables. In this assignment, we will look at an example of such a case by finding points of intersection of two ellipses given by the following functions:

$$f(x,y) = x^2 + y^2 - 1 = 0, \qquad g(x,y) = 5x^2 + 21y^2 - 9 = 0.$$

The first equation describes a circle centered at the origin with radius 1 and the second specifies an intersecting ellipse centered at the origin. We would expect that there will be four solutions, one in each quadrant. We can use substitution to find the four intersection points as

$$\left(\frac{\sqrt{3}}{2}, \frac{1}{2}\right), \left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right), \left(-\frac{\sqrt{3}}{2}, \frac{1}{2}\right), \left(-\frac{\sqrt{3}}{2}, -\frac{1}{2}\right).$$

There are many different ways to find a solution. For this assignment, you will use two different techniques in the same code (one after the other). Because the focus is on getting familiar with C, the methods will be largely specified here, and your task will be to implement these methods.

Method 1: Newton's method:

When it is possible to take derivatives of the desired function, Newton's method often converges quickly. We will not derive Newton's method for systems here, but essentially we form an iterative scheme that can be written in matrix form as follows:

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = -J^{-1}(x_n, y_n) \begin{pmatrix} f(x_n, y_n) \\ g(x_n, y_n) \end{pmatrix}.$$

Here the matrix J is formed by taking partial derivatives of the ellipse functions as follows:

$$J(x_n, y_n) = \begin{pmatrix} 2x_n & 2y_n \\ 10x_n & 42y_n \end{pmatrix}.$$

Although the method is easiest to describe in matrix form, in practice, you should not explicitly form matrices or arrays in your code but rather just keep track of the four entries of the 2x2 matrix and the two entries of the matrix-vector product (e.g., compute $x_{n+1} = J_{11}(x_n, y_n) * f(x_n, y_n) + J_{12}(x_n, y_n) * g(x_n, y_n)$ and similarly for y_{n+1}). You will need to find the inverse of the matrix J as part of your iterative scheme, which can be done analytically for a 2x2 matrix M as

$$M^{-1} = \frac{1}{m_{11}m_{22} - m_{12}m_{21}} \begin{pmatrix} m_{22} & -m_{12} \\ -m_{21} & m_{11} \end{pmatrix}.$$

As you will see, when Newton's method converges, it converges much more quickly than the fixed-point iteration approach, when it converges. Note that the method should fail if the matrix J is singular (whenever $J_{11}J_{22} - J_{12}J_{21} = 0$).

Method 2: Fixed-point iteration

Fixed-point iteration is another method that can be used to solve multiple equations. The basic idea is to develop a related function that can be iterated (perform operations using some numbers, then perform the same operation on the output of the previous iteration, and repeat). This type of calculation, if properly designed, can converge to the same output. For this assignment, use the following iterative functions:

$$x_{n+1} = \sqrt{1 - y_n^2}, \qquad y_{n+1} = \frac{1}{\sqrt{21}}\sqrt{9 - 5x_n^2}.$$

It can be shown that this iterative scheme, if it converges, will converge to a solution of the desired equations. Note that because square roots are involved, you should expect to find only the intersection point with positive values using this method. In addition, the method should fail if at any point it needs to take the square root of a negative number.

Your assignment

Write a C program to find an intersection point of the ellipses. Your program should use Newton's method and then repeat the process using the fixed-point iteration scheme given above.

For this assignment, you are provided a "scaffold" consisting of four files: main.c, solution.c, solution.h, and Makefile. **You should only modify, and only submit, solution.c,** updated to implement the methods described above.

- The file main.c defines variables, sets up initial values, calls the functions to implement the methods, and prints output to the screen.
- The file solution.h includes prototypes for the functions being called in main.c and that you will specify in solution.c.
- The file solution.c includes the beginnings of the functions including initialization of some variables. It also defines status codes that your functions should return as appropriate. Your code should return the following four values at the end of each function (you will need to specify how to calculate them appropriately):
    - the final values of x and y,
    - the number of iterations performed (where the number of iterations is incremented by one after having completed the iteration), and
    - the status code.
- Makefile can be used on the command line to compile the code; alternatively, you can compile the code yourself without using Makefile. To use Makefile, simply enter the command `make` on the command line. (You may need to install make first; linux will let you know!)

Because both methods are iterative, they will need to begin from some user-provided values for x_0 and y_0. The file main.c is set up to allow your program to accept these values as keyboard input. (We will talk more soon about why the `&` before the variable name in `scanf` is necessary.)

You will need to fill in solution.c so that your program proceeds to find an intersection point, first using Newton's method, then using fixed-point iteration, in both cases beginning from the initial values.

In each case, the iteration should proceed until either of the two stopping criteria below is reached:

- 50 iterations have occurred,
- Both the x and y iterates have not changed during a given iteration by more than 10^{-10}. In other words, for some iteration n, it is the case that both $|x_n - x_{n+1}| < 10^{-10}$ and $|y_n - y_{n+1}| < 10^{-10}$.

Note that you will need to calculate several mathematical functions, including absolute value and square root. You should use the functions `fabs` and `sqrt`, respectively, where `fabs` is appropriate for taking the absolute value of `double` variables. You will need to include the `math.h` file along with `stdio.h` and when compiling, including the compiler flag `-lm` at the end of your compilation command. The scaffolded code and Makefile are set up this way already.

The tolerance and maximum number of iterations should be specified using `#define` statements, which for this assignment you may specify in your solution.c file.

Be sure to test for cases where Newton's method has a singular Jacobian (as specified above) and for cases where the fixed-point iteration may seek to take the square root of a negative number. In such a case, you should report the result via the status code and break out of the loop.

Your code must be well structured and documented to receive full credit. Be sure to include comments that explain your code statements and structure.

Your final submission will be tested against several test cases. One will be available for upon submission, so you can check whether your code gets that case correct (keep in mind you know where the intersection points are in general), while the others will remain hidden.

Final submission: You should submit to Gradescope through Canvas the following three files.

(1) your code, named solution.c. **Do not submit main.c, solution.h, or Makefile.**

(2) a text file (not formatted in a word processor, for example) named README.txt that includes the compiler and operating system you used for compiling and running your code along with instructions on how to compile and run your program. (For this assignment, it is expected that this file will be quite short and you may just indicate you used the provided Makefile, if you used it.)

(3) a series of three slides composed in PowerPoint or similar software, saved either in PowerPoint or as a PDF and named slides.pptx or slides.pdf, structured as follows:

- Slide 1: your name and a brief explanation of how you developed/structured your program. This should not be a recitation of material included in this assignment document but should focus on the main structural and functional elements of your program (e.g., the purpose of any loops you used, the purpose of any if statements you used to change the flow of the program, the purpose of any functions you created, etc.). In other words, assuming that the

mathematics behind what you are doing is already known, what were the main things you did to translate those requirements into code? You are limited to one slide.
- Slide 2: a brief statement explaining why you believe your program works correctly. For example, you could select a case or cases where you know the answer (explain how) and verify that your code achieves the correct answer. You are again limited to one slide.
- Slide 3: a brief exposition of what you felt were the most challenging one or two aspects of this assignment and how your final submission improves upon your initial submission, as appropriate—also limited to one slide.

Initial submission: Your initial submission does not need to include the slides. It will not be graded for correctness or even tested but rather will be graded based on the appearance of a good-faith effort to attempt the majority of the assignment. Even implementing one method would be acceptable provided it appears to be nearly complete. No feedback will be given as you will have other opportunities for feedback during class or office hours as well as an additional week to finalize the assignment.

Some hints:

- Start early, and start small! Make sure you can compile and run something like a helloworld program first and make small, progressive modifications.
- Implement one strategy first to make sure it works appropriately, then work on implementing the other method.
- Test using initial values less than one in magnitude, which should be acceptable to the fixed-point iteration scheme, and values much larger than one in magnitude, which may not be acceptable for that method.
- If reading the values in is uncomfortable for you, start by defining the values within the program, either directly using assign statements or using `#define`, and modify to read in values later.
- If working with the code provided is confusing at first, feel free to write your own separate code at first and work on integrating it into the scaffold after you have the methods working correctly.
- Consider printing the values of x and y for each iteration to the screen when you are testing/debugging. (But be sure you can turn this functionality off later.)
- The code scaffold includes some use of pointers, which will be covered during class on 8/31. If you would like to get a head start, you have several options.
    - Write the code without worrying about integrating it into the scaffolding until after we cover pointers (your initial submission is not required to use the scaffolding, but it should be C code that may or may not compile or run).
    - Use a resource like https://www.geeksforgeeks.org/c-pointers/# to start learning about pointers.
    - Use the lecture resources Week 2 (which should be uploaded by the weekend between Weeks 1 and 2) to learn about what pointers are, why they are used here, and what the syntax means.