Project 3  due Thursday 10/10 in lecture

Goals and Instruction
In this project, you will develop programs with
- loops and/or vectorized code
- user-defined functions
- two-dimensional arrays (matrices)
- file input

There are three questions. Question 2 is intentionally under-specified so that you have the freedom (and the job) of crafting an entire solution (design, implement, test, and finally present the result). First skim and then carefully read each question entirely before starting to write algorithms/programs!

Unless otherwise specified, submit all M-files, output, and required plots. Follow the instructions on CS100M→Homework→Projects in assembling your project for submission.

1  Numerical differentiation

The derivative of a continuous function \( f(x) \) is defined by the equation

\[
f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}.
\]

In many applications, derivatives are calculated numerically instead of analytically determined. Three popular numerical approximation methods are

- forward differencing: \( f'(x) = \frac{f(x+h) - f(x)}{h} \)
- backward differencing: \( f'(x) = \frac{f(x) - f(x-h)}{h} \)
- central differencing: \( f'(x) = \frac{f(x+h) - f(x-h)}{2h} \)

Write a program to calculate the derivative of Newton’s serpentine

\[
f(x) = \frac{4x}{x^2 + 1}
\]

for \( x \in [-2,2] \). Use the three numerical methods listed above and compare the results graphically with the analytical derivative \( f'(x) = \frac{4(1-x^2)}{(x^2+1)^2} \) (evaluated at specified points). For calculation and for plotting, use the increment \( h = 0.5 \).

You may use vectorized code and/or loops in your program and you may define your own function(s). Do not use MATLAB pre-defined functions related to differentiation. Your plot should have a legend (see help legend in MATLAB). Submit your program file(s) and the plot. For extra amusement, experiment with different values of \( h \) but do not submit the results. See all those definitions in Calculus come alive!

Some hints and thoughts:
- Consider forward differencing. If the function is sampled (calculated) at regular intervals, then the differencing formula may be written as \( f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} \), where \( h = x_{i+1} - x_i \).
- At least one extra function evaluation is necessary at the boundaries. E.g., in forward differencing, you need to evaluate \( f(2+h) \) in order to approximate \( f'(2) \).
2 Error in central differencing

Let's consider central differencing in more detail and analyze the approximate solution at \( x = 0.5 \). We will use the analytical derivative \( f'(x) = \frac{4 \cdot x^3 + 1}{(2x + 1)^2} \) to calculate the true value of \( f'(0.5) \). The difference between the true and approximated values is the approximation error. The tolerance is the amount of error that we are willing to accept. In computing, we are usually willing to accept this error due to practical limitations on computing time or memory. The smaller the tolerance we choose, the more accurate the approximation becomes.

How small does \( h \) (in the central differencing formula) have to be if our error tolerance is \( 10^{-3} \)?

You must write a program, not use Calculus, to find your answer. Write a user-defined function to evaluate Newton’s serpentine at an \( x \) value (the \( x \) value is supplied as an input argument). Note: if you have already defined such a function for Question 1, then simply use it and do not print a second copy for submission.

You need to decide what kind of output is necessary to show (justify) your answer. Print and submit your program file(s) and output. If you have any additional comments or explanations regarding your solution, write them as comments in the program file.

What will graders look for? A program that demonstrates a coherent and clear strategy to approximate the answer. We are not looking for one particular “correct” number; your solution simply needs to be of the right order of magnitude.

3 Local minimum elevation

Elevation data of a rectangular area is stored in a matrix. Write a program to find the local minimum elevation and its location given a user specified starting point.

We define the local area of a cell to be all adjacent cells, including the diagonal directions. Therefore the local area of a cell not on an edge of the matrix is the 3×3 submatrix with that cell in the center. Two examples are shown below:

<table>
<thead>
<tr>
<th>elevation matrix</th>
<th>elevations in local area of location (3,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 6 6 8 8</td>
<td>3 3 9</td>
</tr>
<tr>
<td>4 3 3 9 8</td>
<td>5 8 8</td>
</tr>
<tr>
<td>4 5 8 8 8</td>
<td>6 7 7</td>
</tr>
<tr>
<td>6 6 7 7 2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>elevations in local area of location (4,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5</td>
</tr>
<tr>
<td>6 6</td>
</tr>
</tbody>
</table>

From a user specified starting point, one wishes to travel to a local minimum elevation. If multiple cells in the local area have the same minimum elevation, pick any one of them as the next location to visit. If the current location has the minimum elevation in the local area, then the final local minimum has been found.

Given the elevation matrix above and a starting location \((4,3)\), the path to the final local minimum is:

Location \((4,3)\) elevation 7 ← starting point
Location \((3,2)\) elevation 5
Location \((2,2)\) elevation 3 ← final local minimum

Note that the final step could have been a move to location \((2,3)\).

The program begins by reading in the elevation data stored in an ASCII file elev.dat. It then displays the size of the data set (numbers of rows and columns) and prompts the user for a starting location. The program checks to see if the user entered location is within range and if necessary, re-prompt the user until a valid location is entered. You can assume that the matrix in elev.dat is not smaller than 2×2.

Your program must use the algorithm specified above. Your program must call a user defined function localMin. Function localMin takes as input argument a 3×3 matrix and have three output arguments: the minimum value in
the 3×3 matrix; the necessary adjustment (-1, 0, or 1) to the current row number to move to the minimum value; and the necessary adjustment to the current column number to move to the minimum value. If the minimum value occurs in multiple locations, function localMin can pick any one of these locations with the minimum elevation. The output of your program should indicate the path, as shown in the example above (but you can leave out the words “starting point” and “final local minimum” or put them on different lines).

Test cases:

- (a) Start at Location (5,6), i.e., row 5 and column 6
- (b) Enter any out-of-bounds location as the first user input, then enter a valid location that is within range

Hints:

- To “read in” an ASCII data file, use the command load elev.dat where elev.dat is the file name. After this command is executed, the workspace (memory) will contain a matrix variable called elev. Matrix elev contains the data in the file elev.dat.

- We require the input argument for function localMin to be a 3×3 matrix, so how do you do make things work even for the case where the current location is on an edge? How about a “border”? What values do you put in the “border”?

Bonus: Look up how to create a contour plot in MATLAB. Add to the end of your program code to create a contour plot of the elevation data in elev.dat. The plot and the contours should be labeled. Include the plot for submission.