CSC338. Homework 8

Due Date: Wednesday March 18, 9pm

Please see the guidelines at https://www.cs.toronto.edu/~lczhang/338/homework.html

What to Hand In

Please hand in 2 files:

- Python File containing all your code, named `hw8.py`.
- PDF file named `hw8_written.pdf` containing your solutions to the written parts of the assignment. Your solution can be hand-written, but must be legible. Graders may deduct marks for illegible or poorly presented solutions.

If you are using Jupyter Notebook to complete the work, your notebook can be exported as a .py file (File -> Download As -> Python). Your code will be auto-graded using Python 3.6, so please make sure that your code runs. There will be a 20% penalty if you need a remark due to small issues that renders your code untestable.

Make sure to remove or comment out all matplotlib or other expensive code before submitting your homework!

Submit the assignment on **MarkUs** by 9pm on the due date. See the syllabus for the course policy regarding late assignments. All assignments must be done individually.

```
import math
import numpy as np
```

Question 1. Fixed-Point Iteration – 3pts

In homework 7, we considered using fixed-point iteration to find a root of the equation $f(x) = x^2 - 3x + 2 = 0$ using these functions:

1. $g_1(x) = \frac{x^2+2}{3}$
2. $g_2(x) = \sqrt{3x - 2}$
3. $g_3(x) = 3 - \frac{2}{x}$

Analyze the convergence properties of each of the corresponding fixed-point iteration schemes for the root x = 2 by analyzing $g_i'(x)$. Do you expect the fix-point iteration to diverge or converge? Do these expectations match your empirical results from homework 7?

Question 2. Newton's Method

Part (a) – 4 pts

Write a function `newton` to find a root of `f(x)` using Newton's Method. This Python function should take as argument both the mathematical function `f` and its derivative `df`, and return a list of successively better estimates of a root of `f` obtained from applying Newton's method.

```
def newton(f, df, x, n=5):
    """ Return a list of successively better estimate of a root of `f`
    obtained from applying Newton's method. The argument `df` is the
    derivative of the function `f`. The argument `x` is the initial estimate
    of the root. The length of the returned list is `n + 1`.

    Precondition: f is continuous and differentiable
                  df is the derivative of f

    >>> def f(x):
```

```
...     return x * x - 4 * np.sin(x)
>>> def df(x):
...     return 2 * x - 4 * np.cos(x)
>>> newton(f, df, 3, n=5)
[3,
 2.1530576920133857,
 1.9540386420058038,
 1.9339715327520701,
 1.933753788557627,
 1.9337537628270216]
"""
```

Part (b) – 2 pts

Use your function from part (a) to solve for a root of

$$f(x) = x^2 - 3x + 2 = 0$$

Start with $x_0 = 3$, and stop when the root is accurate to at least 8 significant decimal digits.

Show your work in your python file, and store the root you find in the variable `newton_root`.

```
def f(x):
    return x ** 2 - 3 * x + 2

newton_root = None
```

Part (c) – 6 pts

Consider the following non-linear equations $h_i(x) = 0$.

1. $h_1(x) = x^3 - 5x^2 + 8x - 4$
2. $h_2(x) = xcos(20x) - x$
3. $h_3(x) = e^{-2x} + e^x - x - 4$

Write out the statement for updating the iterate x_k using Newton's method for solving each of the equations $h_i(x) = 0$.

For each h_i, do you expect Newton's method to converge?

Include your solution in your PDF writeup.

Part (d) – 3 pt

Use the `newton` function to try and solve $h_i(x) = 0$, for `n = 100` iterations, starting with `x = 1.5`.

Save the return values of calls to the function `newton` to the variables `newton_h1`, `newton_h2`, `newton_h3`,.

```
newton_h1 = None
newton_h2 = None
newton_h3 = None
```

Question 3. Secant Method

Part (a) [5 pt]

Write a function `secant` to find a root of `f(x)` using the secant method. The function should return a list of successively better estimates of a root of `f` obtained from applying the secant method.

```
def secant(f, x0, x1, n=5):
    """ Return a list of successively better estimate of a root of `f`
    obtained from applying secant method. The arguments `x0` and `x1` are
    the two starting guesses. The length of the returned list is `n + 2`.

    >>> secant(lambda x: x ** 2 + x - 4, 3, 2, n=6)
    [3,
     2,
     1.6666666666666667,
     1.5714285714285714,
     1.5617977528089888,
     1.5615533980582523,
     1.561552812843596,
     1.5615528128088303]
    """
```

Part (b) [1 pt]

Use the `secant` function to find a root of $f(x) = x^3 + x^2 + x - 4$, accurate up to 8 significant decimal digits.

Show your work in your Python file. You can choose starting positions x_0 and x_1.

Save the result in the variable `secant_root`.

```
secant_root = None
```

Part (c) [4 pt]

Show that the iterative method

$$x_{k+1} = \frac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$$

is mathematically equivalent to the secant method for solving a scalar nonlinear equation $f(x) = 0$.

Include your solution in your PDF writeup.

Part (d) [2 pt]

When implemented in finite-precision floating-point arithmetic, what advantages or disadvantages does the formula given in part (c) have compared with the formula for the secant method given in lecture?

This is the formula given in lecture:

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$$

Include your solution in your PDF writeup.