CSC338. Assignment 3

Due Date: April 9, 10pm

What to Hand In

Please hand in 2 files:

- Python File containing all your code, named `a3.py`.
- PDF file named `a3_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 15% penalty if you need a remark due to small issues that renders your code untestable. (Please note the penalty is higher than in A1!)

Make sure to remove or comment out all matplotlib or other expensive code before submitting your code! Expensive code can render your code untestable, and you will incur the 15% penalty for remark.

Submit the assignment on **MarkUs** by 10pm 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.

Part (a) – 3 pt

Consider the problem of finding the roots of the functions f_1, f_2, and f_3. What is the (absolute) condition number of each problem?

1. $f_1(x) = sin(\frac{x}{100})$, at $x = 0$
2. $f_2(x) = x^3 - 5x^2 + 8x - 4$, at $x = 2$
3. $f_3(x) = x^3$, at $x = 0$

Include your solution in your PDF writeup. Show your work.

Part (b) – 3 pt

What is the convergence rate of each of the following sequences? Your answer should be either "linear", "superlinear but not quadratic", "quadratic", "cubic", or "something else". Include your solution in your PDF writeup. Justify your answer.

1. $x_n = 10^{-2n}$
2. $x_n = 2^{-n^2}$
3. $x_n = 2^{-n \log n}$

Question 2. Interval Bisection

Part (a) – 3 pt

Write a function `bisect` that returns a list of intervals where the root of the function `f(x)` lies. Each interval should be half the size of the previous, and should be obtained using the interval bisection method.

```
def bisect(f, a, b, n):
    """Returns a list of length n+1 of intervals
    where f(x) = 0 lies, where each interval is half
    the size of the previous, and is obtained using
    the interval bisection method.

    Precondition: f continuous,
```

```
                a < b
                f(a) and f(b) have opposite signs

    Example:
    >>> bisect(lambda x: x - 1, -0.5, 2, n=5)
    [(-0.5, 2),
     (0.75, 2),
     (0.75, 1.375),
     (0.75, 1.0625),
     (0.90625, 1.0625),
     (0.984375, 1.0625)]
    """
```

Part (b) – 2pt

Suppose you would like to use the interval bisection method to find the root of a function $f(x)$, starting with an interval (a, b).

What is the minimum number of interval bisection iterations necessary to guarantee that your estimate of a root is accurate to 10 decimal places?

Question 3. Fixed-Point Iteration

Part (a) – 3 pt

Write a function `fixed_point` to find the fixed-point of a function `f` by repeated application of `f`. The function should return a list of values `[x, f(x), f(f(x)), ...]`.

```
def fixed_point(f, x, n=20):
    """ Return a list lst = [x, f(x), f(f(x)), ...] with
    `lst[i+1] = f(lst[i])` and `len(lst) == n + 1`

    >>> fixed_point(lambda x: math.sqrt(x + 1), 3, n=5)
    [3,
     2.0,
     1.7320508075688772,
     1.6528916502810695,
     1.6287699807772333,
     1.621348198499395]
    """
```

Part (b) – 3 pt

To find a root of the equation

$$f(x) = x^2 - 5x + 4 = 0$$

we can consider finding the fixed-point of each of these functions:

1. $g_1(x) = \frac{x^2+4}{5}$
2. $g_2(x) = \sqrt{5x - 4}$
3. $g_3(x) = 5 - \frac{4}{x}$

Suppose we use the `fixed_point` function to find the fixed point of each of these functions. If we choose a value x_0 close to 1, would you expect the fixed point iteration to converge to 1?

In your PDF write up, explain why you would expect the iteration to converge (or not).

Part (c) – 3 pt

Run the function that you wrote in part (a) to find the fixed points of g_1, g_2 and g_3. Start at $x_0 = 3$.

Does fixed-point iteration converge for each of the functions? Include the output of your call to `fixed_point` in your PDF writeup.

If the iteration converges, what do you think is the approximate the convergence rate of convergence? (linear, superliner, quadratic, etc).

Include your solution in your PDF writeup.

Question 4. Newton's Method for Root Finding

Part (a) – 3 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

$$f(x) = x^2 - 5x + 4 = 0$$

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 - 5 * x + 4

newton_root = None
```

Part (c) – 6 pts

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

1. $h_1(x) = (x-2)(x-5)(x-1)$

2. $h_2(x) = x\cos(\pi x) - x$
3. $h_3(x) = e^{-2x+4} + e^{x-2} - x$

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 problem, do you expect Newton's method to converge if we start close to the root $x = 2$?

Include your solution in your PDF writeup.

Question 5. 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.

Question 6. Golden Section Search

Part (a) – 6 pt

Write a function `golden_section_search` that uses the Golden Section search to find a minima of a function `f` on the interval `[a, b]`. You may assume that the function `f` is unimodal on `[a, b]`.

The function should evaluate `f` only as many times as necessary. There will be a penalty for calling `f` more times than necessary.

Refer to algo 6.1 in the textbook, or slide 19 of the slides accompanying the textbook.

```
def golden_section_search(f, a, b, n=10):
    """ Return the Golden Section search intervals generated in
    an attempt to find a minima of a function `f` on the interval
    `[a, b]`. The returned list should have length `n+1`.

    Do not evaluate the function `f` more times than necessary.

    Example: (as always, these are for illustrative purposes only)

    >>> golden_section_search(lambda x: x**2 + 2*x + 3, -2, 2, n=5)
    [(-2, 2),
     (-2, 0.4721359549995796),
     (-2, -0.4721359549995796),
     (-1.4164078649987384, -0.4721359549995796),
     (-1.4164078649987384, -0.8328157299974766),
     (-1.1934955049953735, -0.8328157299974766)]
    """
```

Part (b) – 2 pt

Consider the following functions, both of which are unimodal on `[0, 2]`

$f_1(x) = 2x^2 - 2x + 3$

$f_2(x) = -xe^{-x^2}$

Run the golden section search on the two functions for $n = 10$ iterations.

Save the returned lists in the variables `golden_f1` and `golden_f2`

```
golden_f1 = None
golden_f2 = None
```

Question 7.

Consider the function

$$f(x_0, x_1) = 2x_0^4 + 3x_1^4 - x_0^2x_1^2 - x_0^2 - 4x_1^2 + 7$$

Part (a) – 2 pt

Derive the gradient. Include your solution in your PDF writeup.

Part (b) – 2 pt

Derive the Hessian. Include your solution in your PDF writeup.

Part (c) – 5 pt

Write a function `steepest_descent_f` that uses a variation of the steepest descent method to solve for a local minimum of $f(x_0, x_1)$ from part (a). Instead of performing a line search as in Algorithm 6.3, the parameter α will be provided to you as a parameter. Likewise, the initial guess (x_0, x_1) will be provided.

Use `(1, 1)` as the initial value and perform 10 iterations of the steepest descent variant on the function f. Save the result in the variable `steepest`. (The result `steepest` should be a list of length 11)

```
def steepest_descent_f(init_x0, init_x1, alpha, n=5):
    """ Return the $n$ steps of steepest descent on the function
    f(x_0, x_1) given in part(a), starting at (init_x0, init_x1).
    The returned value is a list of tuples (x_0, x_1) visited
    by the steepest descent algorithm, and should be of length
    n+1. The parameter alpha is used in place of performing
    a line search.

    Example:

    >>> steepest_descent_f(0, 0, 0.5, n=0)
    [(0, 0)]
    """
    return []

steepest = []
```