CSC338. Homework 9

Due Date: Wednesday March 25, 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 `hw9.py`.
- PDF file named `hw9_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. 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 posted 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 2. Newton's Method in One Dimension

Part (a) – 6 pt

Implement Newton's Method to find a minima of a real function in one dimension.

```
def newton_1d(f, df, ddf, x, n=10):
    """ Return the list of iterates computed when using
    Newton's Method to find a local minimum of a function `f`,
    starting from the point `x`. The parameter `df` is the
    derivative of `f`. The parameter `ddf` is the second
    derivative of `f`. The length of the returned list
    should be `n+1`.

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

    >>> newton_1d(lambda x: x**3,
    ...           lambda x: 3 * (x**2),
    ...           lambda x: 6 * x,
    ...           x=1.0, n=5)
    [1, 0.5, 0.25, 0.125, 0.0625, 0.03125]
    """
```

Part (b) – 4 pt

Consider `f1` and `f2` from Question 1 Part (b).

Complete the functions `df1`, `df2` which compute the derivatives of `f1` and `f2`.

Complete the functions `ddf1`, `ddf2` which compute the second derivatives of `f1` and `f2`.

```
def df1(x):
    return None
def ddf1(x):
    return None

def df2(x):
    return None
def ddf2(x):
    return None
```

Part (c) – 2 pt

Run Newton's Method on the two functions f_1 and f_2 for $n = 30$ iterations, starting at $x = 1$.

Save the returned lists in the variables `newton_f1` and `newton_f2`.

```
newton_f1 = None
newton_f2 = None
```

Question 3.

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) – 6 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 = []
```