Name:

AMATH 515 Homework 4
Due: Monday Feb 17, by midnight.

This assignment will be coding based, and more involved, so make sure that you start early. You will implement and compare different optimization algorithms, and write a short report on your findings. We have provided implementations of objective functions in objectives.py and some basic scaffolding for optimization algorithms in solvers.py. You are required to hand in the pdf of a report, written in the style of an academic publication, that summarizes your approach and your numerical results. A template is provided for this report as described below. You must also submit a Jupyter notebook that implements your algorithms and produces the figures that you include in your report. You will be graded on both the report and the code.

Structure of your report

Your report should be a maximum of 3 pages in the style of the Latex template provided on Canvas (template.tex). You are not allowed to change the font and margins of the template. **a part of your grade will be the cleanliness and general quality of the written report.** Your report must consist of the following sections:

(1) Title, name of author, email address
(2) Introduction - very briefly describe the content of the report and summary of main findings
(3) Methods - Briefly describe important details of the algorithms you implemented
(4) Results - This should take majority of your report. Perform the tasks outlined below and provided the required plots or tables as well as any other experiments you performed. Be sure to describe your findings for each task concisely and clearly
(5) Conclusion - summary of main findings and concluding remarks, if any.

3 pages is a very limited space so make sure to be efficient with content, figures, and tables. In general:

- Make sure figures are not too big and if you are shrinking figures be sure to change the font size for title, legends, etc to make them readable.
- Make sure all figures and tables have axis labels, legends, and captions as needed. You will lose points otherwise.
- Make sure all figures and tables are discussed and referenced in the main text.

Tasks

(1) Implement a function that performs a bisection algorithm for line search using the (weak) Wolfe conditions. For parameters $0 < c_1 < c_2 < 1$, the Wolfe conditions (as

outlined in Lecture 8) are

$$f(x^k + \alpha^k d^k) \leq f(x^k) + c_1 \alpha^k (d^k)^T \nabla f(x^k), \tag{W1}$$

$$c_2 (d^k)^T \nabla f(x^k) \leq (d^k)^T \nabla f(x^k + \alpha^k d^k). \tag{W2}$$

where d^k denotes the descent direction and α^k is the step size at the current step. The first condition eq. (W1) asserts that some proportion of the decrease predicted by a linearization of the objective function is achieved, and prevents us from taking too large of a step. The second condition eq. (W2) asserts that the directional derivative in your search direction is sufficiently decreased, meaning that there is not too much decrease left to be obtained in the direction d_k, preventing us from taking too small of a step. The condition eq. (W2) is also crucial in implementation of DFP and BFGS algorithms, as it ensures that the curvature estimate $(y_k)^T s_k$ is positive.

To implement your line search algorithm, define

$$W_1(t) := \frac{f(x^k + t d^k) - f(x^k)}{t (d^k)^T \nabla f(x^k)} \tag{1}$$

$$W_2(t) := \frac{(d^k)^T \nabla f(x^k + t d^k)}{(d^k)^T \nabla f(x^k)} \tag{2}$$

so that the eqs. (W1) and (W2) are satisfied for $\alpha^k = t$ when $W_1(\alpha^k) \geq c_1$ and $W_2(\alpha^k) \leq c_2$ respectively; note, the inequalities are flipped since the d^k are descent directions and so $(d^k)^T \nabla f(x^k)$ is negative. Then the bisection algorithm can be summarized in Algorithm 1 below. **In your methods section explain in a couple of sentences how this algorithm works. What do a, b, t represent?** What happens when $f(x^k + t d^k k)$ is nan?

Algorithm 1 Bisection for weak Wolfe conditions (Version 1)

Input: Function f, Gradient ∇f, previous iterate x_k, search direction d_k, constants $0 < c_1 < c_2 < 1$,initial stepsize t_0.
Initialize: Set $a = 0$, $b = \infty$, $t = t_0$
while $W_1(t) < c_1$ **or** $W_2(t) > c_2$ **do**
  if $W_1(t) < c_1$ **then**
    $b = t$
    $t = (b + a)/2$
  else if $W_2(t) > c_2$ **then**
    $a = t$
    $t = \min(2 \cdot a, (a + b)/2)$
  end if
end while
Return $\alpha = t$

You can also implement the algorithm directly using the Wolfe conditions and without using the $W_1(t)$ and $W_2(t)$ functions defined above. The resulting method is summarized in Algorithm 2 below.

Algorithm 2 Bisection for weak Wolfe conditions (Version 2)

Input: Function f, Gradient ∇f, previous iterate x_k, search direction d_k, constants $0 < c_1 < c_2 < 1$,initial stepsize t_0.
Initialize: Set $a = 0$, $b = \infty$, $t = t_0$
while $f(x^k + td^k) - f(x^k) > c_1 t (d^k)^T \nabla f(x^k)$ **or** $(d^k)^T \nabla f(x^k + td^k) < c_2 (d^k)^T \nabla f(x^k)$ **do**
  if $f(x^k + td^k) - f(x^k) > c_1 t (d^k)^T \nabla f(x^k)$ **then**
    $b = t$
    $t = (b + a)/2$
  else if $(d^k)^T \nabla f(x^k + td^k) < c_2 (d^k)^T \nabla f(x^k)$ **then**
    $a = t$
    $t = \min(2 \cdot a, (a + b)/2)$
  end if
end while
Return $\alpha = t$

(2) Implement steepest descent, Newton's method, DFP, and BFGS using the line search algorithm from Task (1). Recall that all of these algorithms take the form

$$x^{k+1} = x^k + \alpha^k d^k \quad \text{where} \quad d^k = -B(x^k)\nabla f(x^k)$$

and the matrix $B(x^k)$ is chosen as

- (Steepest descent) $B(x^k) = I$
- (Newton[1]) $B(x^k) = \nabla^2 f(x^k)^{-1}$
- (DFP) $B(x^{k+1}) = B(x^k) - \frac{\left(B(x^k)y^k\right)\left(B(x^k)y^k\right)^T}{(y^k)^T B(x^k) y^k} + \frac{s^k (s^k)^T}{(y^k)^T s^k}$
- (BFGS) $B(x^{k+1}) = \left(I - \frac{s^k (y^k)^T}{(y^k)^T s^k}\right) B(x^k) \left(I - \frac{y^k (s^k)^T}{(y^k)^T s^k}\right) + \frac{s^k (s^k)^T}{(y^k)^T s^k}$

where, following the lecture notes, we defined $s^k := x^{k+1} - x^k$ and $y^k = \nabla f(x^{k+1}) - \nabla f(x^k)$.

In the notation for the line search, this means that $d_k = -B(x^k)\nabla f(x^k)$, and for DFP and BFGS, $B(x^k)$ meant to be an approximation to the inverse of the Hessian at x^k. When implementing each algorithm be sure to track the gradient norms, function values, and cumulative time at each iteration as these will be the main quantities to report in your results section.

(3) Apply these algorithms to the following optimization problems. We have already implemented these in objectives.py, and you can load them into your code. For

[1]If the Hessian is not positive definite, it is possible for the Newton step to produce a search direction which is not a descent direction. In this case, it is often beneficial to replace the search direction with the negative gradient, taking a gradient descent step, and then continuing with Newton's method.

each problem clarify your choice of c_1, c_2 in the linesearch step as well as any other choice of parameters as needed; you can efficiently summarize these in a table.

You will likely want to specify stopping criteria, such as a tolerance for the gradient norm, and terminate when the gradient norm is below that value. For the logistic regression problem, you may want to specify a relatively loose tolerance (around 10^{-5}), as you can lose accuracy due to numerical roundoff below that value for the quasi-Newton methods.

(a) The Rosenbrock function,

$$f(x) = \sum_{k=1}^{n} (x_i - 1)^2 + 100 \sum_{k=1}^{n-1} (x_i^2 - x_{i+1})^2 \tag{3}$$

Test this for $n = 2, 5, 10, 50$, initialized at $x_0 = (-1, -1, ..., -1)$. **For each value of n present a plot of the value of f as a function of the number of iterations of the four algorithms you implemented above as well as the values of the gradient norms. You can present a single figure for each choice of n, overlaying the curves for the four algorithms.**

(b) Logistic regression on MNIST:

$$f(x) = \sum_{i=1}^{m} \{\log(1 + \exp(\langle a_i, x\rangle)) - b_i \langle a_i, x\rangle\} + \frac{\lambda}{2}\|x\|^2. \tag{4}$$

Where a_i is the pixel representation of the i'th image in the MNIST handwritten digits dataset, and $b_i = 0$ is the digit written (either 0 or 1). Test this for $\lambda = 0.001, 0.01, 0.1$ initialized at $x_0 = 0$. **For each value of λ present a plot of the value of f as a function of the number of iterations of the four algorithms you implemented above as well as the values of the gradient norms. You can present a single figure for each choice of λ, overlaying the curves for the four algorithms.**

(c) Discuss your findings in these two problems. How do the methods perform? how do they compare in terms of running time? how do they compare otherwise? From these two examples which method would you say is the better choice and why?

(d) Find another test problem/objective function to run your code on. Explain how you came up with this objective. What properties does it have that might make it interesting for experimentation? You can find some nice example at the following webpage: `https://www.sfu.ca/~ssurjano/optimization.html`

(4) Write up your results in a pdf which you will submit to gradescope, with figures comparing the convergence of different algorithms and the computational cost. You may want to use a jupyter notebook to experiment with your algorithms. Test different values of c_1, c_2, and explain what happens as you vary them.