CMPUT 466/566 — Programming Exercise #5(a) - Grads

Instructor: R Greiner, B cao
Due Date: 5pm, Monday 11/Mar/2019

Graduate Students are required to answer this question, and also the Coursera Programming Exercises identified as *C#4 (Neural nets)*.
(Undergraduates are required, instead, to answer two Coursera Programming Exercises – both *C#3 (MultiClass + NN)* and *C#4 (Neural nets)*, but not this question. They can get up to 10% extra credit for solving this question.)

This exercise is intended to further your understanding of Conjugate Gradient.

Relevant reading: [HTF: parts of Ch 11]
+ "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain File"

Question 1 *[30 points]* *Conjugate Gradient* [Implementation]
The purpose of this question is to understand Conjugate Gradient (CG). For now, let's forget about the *estimation* part of parameter optimization by assuming we already know everything there is to know about the error function, and just want to find the values of the parameters that minimize this function. To make life easy, we will consider an error function that is a simple quadratic over the pair of parameters, $[x, y]$

$$f([x, y]) \quad = \quad a \times (x-1)^2 \; + \; b \times y^2 \tag{1}$$

for some specified values of a and b. Your task is to write a program that can efficiently "descend" from a given initial vector $[x_1, y_1]$ to produce the values of $[x, y]$ that minimize this function. (Yes, we all know this minimal value is at $[x^*, y^*] = [1, 0]$. But here the path is important...)

You will implement three routines — simple gradient descent (GD), GD with line search, and CG — and compare their behavior over the following initial set-ups.

a	b	initial_weight vector, (x_1, y_1)
1	1	$[10, 0]$
1	1	$[10, 10]$
1	2	$[10, 10]$
1	10	$[0.1, 0.1]$
1	10	$[5, 10]$
1	10	$[20, 10]$
1	1000	$[40, 40]$

In particular, you should indicate *how many iterations each algorithm required to converge* (see convergence criteria below). You should also submit a ZIP file of the relevant programs, such that the top-level Matlab call

```
» Weights = GradientDescent( 1, 10, [20,10], χ, 1E-6, 100)
```

deals with the $a = 1$, $b = 10$ and initial_weight= $[20, 10]$ setting; here `Weights(1,:)` should be the given initial_weight, and `Weights(j+1,:)` should be the weight vector w_{j+1} *right after finishing* the $j^{th}(j \geq 1)$ iteration of the algorithm. The routine will stop if the euclidean norm of $(w_{j+1} - w_j)$ (*i.e.*, the difference of weight vectors in current and last iterations) is less than 10^{-6}, or, if the number of iterations exceeds 100. This routine should cover all three different processes shown below, depending on the value of χ provided (see below). You may also wish to plot these results, to help visualize what is going on.

For notation, at iteration j, you have the current weights w_j and can directly compute the gradient g_j, as well as the (constant) Hessian H, which you will use to compute the new weights w_{j+1}.

a [10]: Setting the fourth argument "$\chi = 0$" should call your implementation of simple gradient descent, which uses the simple heuristic of traveling $\eta = \frac{\sqrt{a^2+b^2}}{j}$ along the negative gradient $-g_j$.

b [3]: Given the weight vector w_j and gradient g_j, you want to descend along $-g_j$ to a new vector $w_{j+1} \leftarrow w_j - \eta\, g_j$ that produces the smallest $f(.)$ value — *i.e.*,

$$\eta \quad = \quad \operatorname*{argmin}_t f(w_j - t g_j) \tag{2}$$

In class, we provided an iterative way to do this, which is useful if $f(\cdot)$ is unknown. Here, we know $f(\cdot)$. Provide a simple direct formula for computing Equation 2 based on our particular $f(\cdot)$ from Equation 1.

[Hint: Take the derivative of values along this line.]

c [7]: Setting "$\chi = 1$" should call your implementation of gradient descent using "line-search": that is, your code should determine the optimal distance to travel along the gradient g_j, to minimize $f(\,[x, y]\,)$ — using your answer to (b) above.

d [10]: Setting "$\chi = 2$" should call your implementation of conjugate gradient: Here, at iteration j, you should compute the new weights w_{j+1} using the following formulae:

$$\begin{aligned} w_{j+1} &= w_j + \alpha_j\, d_j \\ d_1 &= -g_1 \\ d_{j+1} &= -g_{j+1} + \beta_j d_j \qquad \text{for } j \geq 1 \end{aligned}$$

$$\beta_j \quad = \quad \frac{g_{j+1}^T (g_{j+1} - g_j)}{g_j^T g_j} \qquad \alpha_j \quad = \quad -\frac{d_j^T g_j}{d_j^T H d_j}$$

Once again, you can just directly compute the Hessian matrix H, and every other quantity needed to go from w_j to w_{j+1}, without any iterations.

You should hand-in

- your well-commented Matlab/Octave implementation of
  `GradientDescent(a,b,w_init,`χ`,diff,num_iter)`
  – your top-level function that implements the specific gradient descent procedures, as

described above.

- You must also hand-in 3 different m-files (Matlab/Octave functions), and have `GradientDescent` call one of these functions depending on the value of χ:
  - `PE5aSimpleGD.m` (simple gradient)
  - `PE5aGDLineSearch.m` (gradient descent with line search)
  - `PE5aConjugateGradient.m` (conjugate gradient descent)
- A pdf report on how these (sub)routines works, named `report.pdf`.
  You should specify the number of iterations each algorithm required to converge. You should also include your response to Question [b] in this report.

To submit your solution: create a directory `PE_5a` and put all your deliverables (`.m` and `.pdf` files) inside it. Submit only ONE zip file that contains this folder via **eClass**.