Convex and Nonsmooth Optimization
HW5: Gradient and Newton Methods

Michael Overton

Spring 2022

Note: although I refer to MATLAB repeatedly below, one big advantage to using Python instead is the availability of automatic differentiation (AD) tools. You may these if you wish, translating the MATLAB templates below accordingly. Although I won't be able to answer any questions about Python or its AD tools, perhaps other students will be able to answer any questions you post on Campuswire about this.

1. Write a MATLAB function to implement the Gradient Method with a backtracking line search. The first line should be

   ```
   function [f_all,gnorm_all] = gradmeth(fun,x0,tol,maxit)
   ```

   as in the template gradmeth.m. The first input parameter `fun` is an anonymous function, which means that calls to `fun` inside `gradmeth` will actually be to some function whose name is unknown until `gradmeth` is called. This is the function that `gradmeth` is supposed to minimize; its implementation must return both the function value `f` at a given point `x` and its gradient `g` there. The second input parameter to `gradmeth` is the starting point, the third is a termination tolerance on the norm of the gradient and the fourth is the max number of iterations to be allowed. Use a `while` loop to implement the main iteration, and code the backtracking line search (see p.3 of my notes on the gradient method or BV p.464) in a separate function, since we will also need it for Newton's method below. Use $\alpha = 1/4$, $\beta = 1/2$. The first output argument of `gradmeth` should be a vector of function values $f(x^{(k)})$ for plotting purposes (not including the intermediate function values computed in the backtracking line searches, if any), and the second should be the corresponding vector of gradient norms $\|\nabla f(x^{(k)})\|$. (Normally we would also output the $x^{(k)}$ as well, particularly the final value, but we won't need those in this homework.)

Test my template on the quadratic function $f(x) = \frac{1}{2}x^T Ax + x^T b$ coded in quad.m by calling the script example.m, which defines A and b, and make sure you understand how all this works before replacing the template gradmeth.m by your own code.

(a) Assuming A is positive definite, as in example.m, we can compute the minimizer of $\frac{1}{2}x^T Ax + x^T b$, namely $-A^{-1}b$, by the statement `x=-A\b`, which defines p^*. (Type `help \` at the MATLAB prompt if you are not familiar with the crucial "matrix left divide" operator `\`.) By what factor does the gradient method reduce $f(x) - p^*$ in 100 iterations, using the starting point in example.m? How does this compare with the theory that we discussed in class (see last page of my gradient method notes or BV Section 9.3)? You can obtain the parameters M and m for f by computing the eigenvalues of $\nabla^2 f(x) = A$ using `eig`.

(b) Now run your gradient method on the more interesting problem defined in BV Ex. 9.30 (p. 519). For the data defining the problem, set $n = 100$, $m = 50$, and the vectors a_i as the columns of the matrix saved in Adata.mat. (Note the two different meanings of m in this homework!) In writing the code for this objective function, set `f` to `inf` and `g` to be `nan*ones(n,1)` if `x` is not in $\textbf{dom} f$. Then the line search will reject these points as long as you start with a point in the domain. You will need to carefully code the gradient; it's a good idea to check the formula against "difference quotients" to make sure you didn't make a mistake: see p. 72-74 of my book if you don't know how to do this.

Run your gradient method on this problem, starting from $x^{(0)} = 0$, terminating when the norm of the gradient is 10^{-6} (written `1e-6`) giving you a reasonable estimate of the minimal value p^*: print this value. Using this estimate, along with the function values returned by `gradmeth`, plot the values $f(x^{(k)}) - p^*$ using a log plot, for example with `semilogy(f_all - p_star,'x')`. The plot should be made after exiting from `gradmeth`. Make another log plot of the gradient norms. Use `legend`, `title`, `xlabel`, `ylabel` to label the plots nicely. Use the results to *estimate* the condition number M/m, and explain how you did this.

2. Write another function `newtmeth` that implements Newton's method in a similar fashion. Now the function to be minimized must return a third output argument, the Hessian $H = \nabla^2 f(x)$. Since it is easy to make a mistake coding the Hessian matrix, check your computed

result against difference quotients for the gradients. Computing the Newton search direction requires solving a system of linear equations, which is most easily done using the matrix left divide operator `\`. You should never use `inv` to solve a system of equations, because it may introduce unnecessary error and, especially if the matrix is large and sparse (not this case), be quite inefficient. MATLAB checks to see if the matrix is symmetric, and if it is, solves the linear system using Cholesky factorization; should the matrix turn out not to be positive definite, so that Cholesky breaks down, it would then switch to a different method called LDL^T factorization, where D is block diagonal with blocks of order 1 or 2, but that won't happen in the strictly convex case if you code the Hessian matrix correctly. You could alternatively explicitly compute the Cholesky factorization using `chol` and then solve the system of equations using forward and backward substitution, but this isn't necessary.

(a) Explain why it's trivial to minimize the quadratic function in question 1(a) above by Newton's method.

(b) Apply Newton's method to the minimization problem of question 1(b) above. Set the termination tolerance on the gradient norm to a small value such as 10^{-8} to ensure that you observe the quadratic convergence phase. Superimpose a plot of $f(x^{(k)}) - p^*$ on top of the function value plot produced in question 1(b), and do the same with the gradient norm plot, modifying the legends accordingly. Be sure to use log plots (using `semilogy`). Do you observe quadratic convergence? If not, there is probably a bug in your code.

How many iterations of Newton's method are required? How does this compare with the theory discussed on the last page of these notes. The values for γ and η are given on the 3rd page of the notes. Now that you are computing the Hessian, you can estimate M and m by computing the largest and smallest eigenvalues of the Hessian at several different points, including the starting point 0 and the computed minimizer. Likewise you can estimate L, the Lipschitz constant for the Hessian, by computing the norm of the differences of the Hessian at various points in the domain.

Turn in MATLAB code listings as well as answers to *all* the questions above. Make sure that your codes have *plenty of comments* explaining what they do. If anything is not clear, don't hesitate to ask questions on the Campuswire page.