Graded Homework 2 Numerical Optimization Fall 2023

The solutions are due by 11:59PM on Sunday November 12, 2023

Programming Exercise

Codes

You are to add an inexact Newton method (that includes an option for Steepest Descent) and a Quasi-Newton method to your codes and empirically explore the performance of the three methods compared to each other and to overall quality of efficiency and effectiveness.

Comments on Codes

- For the inexact Newton method you should used the Truncated CG approach by modifying your CG code produced for earlier homework assignments.
- For the Quasi-Newton method you may choose out of the main versions presented in the notes and references
    1. Standard BFGS with a constant guard on the positivity of the curvature condition, i.e., Powell's form.
    2. Cautious BFGS of Li and Fukishima
    3. Damped BFGS
- For the Quasi-Newton you may use either the H_k form or the B_k form for your codes but the H_k form is recommended for this assignment.
- For the line search termination you must implement
    1. The Wolfe conditions (strong or standard);
    2. The Armijo-Goldstein conditions (backtracking or interpolation). However, make sure you take the appropriate action to make sure the codes work given your choice.
- Be careful of the numerical problems in the Armijo-Goldstein test and for either termination you should switch to a fixed α_k. This should be 1 for these methods but you can explore the effect of taking $\alpha_k = \alpha < 1$ for $k > k_0$.
- For the initial step size selection you must implement the either of the two methods on slides 30 and 31 of Set 9 of the class notes, and both of the BB initial stepsizes on slide 32.

- You may find it most convenient to put all of this in one code with appropriate condition execution of the choice of approach in each section. Your codes for the strictly convex quadratic optimization provides the appropriate high level control.

Tasks

Your overall task is to evaluate and compare the performance of the three codes with respect to the different choices listed above. More specifcally,

1. When collecting performance information make sure it includes for each step the value of the cost function $f(x_k)$, a norm of the gradient $\nabla f(x_k)$ (which can be compared to the norm of the initial gradient or unit roundoff), initial step size $\alpha_k^{(0)}$, final step size initial step size α_k, number of function and gradient evaluations, number of matrix vector products, number of inner products (for the Truncated CG iteration for example), number of vector additions (for the Truncated CG iteration for example), and any other major primitive in the iteration. For each of these you should have a floating point operation count parameterized in terms of problem size and any other relevant parameter.

2. When analyzing and comparing the performance the primitive counts and their complexities can be used to report the total number of floating point operations for each step, the total used up to any step, and the final total when the iteration is terminated. This can be used to provide a machine-independent performance comparison of methods when run on the same problem from the same initial $x_{(k)}$. In fact, plotting information such as the norm of the gradient vs number of floating point operations used rather than time is an excellent way to augment other plots that use iterations as the independent variable.

3. You should assess the performance of each method with respect to the different choices of

   - initial step size;
   - line search termination criteria;
   - termination of the overall iteration.

4. You should use the performance information of each method to compare the methods for efficiency in terms of totall floating point operations vs the quality of the solution, i.e., norm of the gradient (absolute and relative to size of the initial gradient and the cost function at the final iterate, e.g., $\|\nabla f(x_k)\|$, $\|\nabla f(x_k)\|/\|\nabla f(x_0)\|$, $\|\nabla f(x_k)\|/|f(x_k)|$.

5. You are free to make use of the benchmark problems and any other problems from the notes, reference papers posted, or other optimization texts, in particular Nocedal and Wright's textbook and Beck's Introduction to Nonlinear Optimization. It is a good

idea to run some strictly convex quadratic problems for which you have already used your earlier codes. Strictly convex but not necessarily quadratic are another good class of problem since they have a unique global minimizer.

6. Note that for Steepest Descent it is often useful to scale the problem with a matrix that is simpler than the Hessian inverse, i.e., Newton's method, or an inexact form. For example, using a diagonal matrix D_k can improve things. So the line search would be run $f(x_k - \alpha D_k \nabla f_k)$ and the step would then be $x_{k+1} = x_k - \alpha_k D_k \nabla f_k$ where D_k is a positive diagonal matrix. If the Hessian $\nabla^2 f(x_k)$ has positive diagonal elements then D_k can be set to the matrix with elements $(e_j^T \nabla^2 f(x_k) e_j)^{-1}$. For example, consider $f(x) = 1000\xi_1^2 + 40\xi_1\xi_2 + \xi_2^2$ over $\mathbb{R}^2$ with an without $D_k = diag(1/1000, 1)$. This is a convex quadratic problem.

7. Also note that Newton's method can often be a nice scaling, if computationally affordable, to accelerate Steepest Descent. For example, consider using Steepest Descent with Armijo-Goldstein backtracking on the cost function $f(x) = 100\xi_1^4 + 0.01\xi_2^4$ and Newton's method. Clearly this is a poorly scaled problem. Note that the Hessian for this problem is not always positive definite and does not satisfy a Lipschitz condition. Nevertheless, Newton works.

8. Another problem discussed by Beck is $f(x) = \sqrt{\xi_1^2 + 1} + \sqrt{\xi_2^2 + 1}$. The Hessian is always positive definite but does not have a lower bound on its minimum eigenvalue. Consider the behavior of Newton's method starting at, say $x_0 = (\xi_1^{(0)}, \xi_2^{(0)})^T = (1, 1)^T$, and compare it to Steepest Descent with Armijo-Goldstein with backtracking.

9. The Rosenbrock function is a classic benchmark that has extreme level contours and can cause difficulties:

$$f(\xi_1, \xi_2) = 100(\xi_2 - \xi_1^2)^2 + (1 - \xi_1)^2$$

with $x^* = (1, 1)$ (you should verify this). Steepest Descent, Newton and BFGS methods will have significantly different behavior.