February 6, 2019

MATH 5650/6650

Homework 2

Due Thursday, February 14

For this assignment, you will conduct a numerical investigation into the line search method. You are not required to write any *big* programs from scratch (maybe small ones like in Question 2), but rather to use and explore the code provided. To complete the assignment, you need the following files:

For Matlab

`steepest_descent.m` implements the steepest descent method,

`backtrack.m` computes acceptable step length by backtracking,

`test_sd_quadratic.m` tests (explores) the steepest descent function for quadratic functions,

`test_sd_rosenbrock.m` tests the steepest descent function for quadratic functions,

`quadratic.m` quadratic benchmark function,

`rosenbrock.m` Rosenbrock benchmark function.

For Python

`line_search.py` file contains steepest descent method, and backtracking algorithm.

`test_steepest_descent.py` tests the steepest descent function,

`benchmark_functions.py` contains the quadratic and Rosenbrock benchmark functions

Fixed parameter values

Throughout the assignment, the following parameter values should remain fixed:

$\texttt{tol} = 10^{-6}$	convergence tolerance
$\texttt{x}_0 = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$	Initial guess
$\texttt{c}_1 = 10^{-4}$	sufficient decrease parameter
$\rho_\texttt{s} = 0.9$	shrinking factor for backtracking

Questions

1. Run the line search algorithm for the quadratic benchmark function
$$f(x) = c + b^T x + \frac{1}{2} x^T A x,$$
with $c = 0, b = [1,1]^T$, and $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. It should converge in one step to the optimum x^*. Assuming that the step length parameter $\alpha = 1$ is chosen, show *by hand* that for any initial guess x_0, the next iterate x_1 will be the minimizer.

2. Now change A to $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ and run the steepest descent algorithm again. It should now take longer to converge. Verify numerically that the convergence is linear, i.e.
$$\|x_{k+1} - x^*\| \leq \rho \|x_k - x^*\|,$$
as $k \to \infty$ (explain your reasoning) and determine the value of $\rho \in (0,1)$. On paper, write down the calculated value ρ and show how you calculate it. Write a function, `compute_rho` (submit it) to compute ρ from the iterates x_k and the minimizer x^*.

3. One reason for the slower convergence might be that the backtracking algorithm is giving us bad step lengths α. For the quadratic function f with positive definite matrix A, there is actually a way to compute the optimal step lengths exactly as $\min_{\alpha>0} \phi(\alpha) = \min_{\alpha>0} f(x_k + \alpha p_k)$. Verify that $\phi(\alpha)$ is a quadratic function in α and hence show that it attains its minimum at $\alpha^* = \frac{p_k^T p_k}{p_k^T A p_k}$.

4. Replace the backtracking algorithm in your steepest descent code with the exact formula for α^* (for this you need A, which is just the Hessian of f) and run your steepest descent method with the same matrix A as in Question 2. It should converge faster. Submit the convergence plots, i.e. semilog plots of (i) the error in the function value $|f(x_k) - f(x^*)|$, (ii) the norm of the gradient $\|\nabla f(x_k)\|$, and (iii) the error $e_k = \|x_k - x^*\|$.

5. We now investigate the role of the conditioning of the Hessian on the convergence rate. Consider the sequence of matrices

$$A_n = \begin{bmatrix} \frac{1}{n} & 0 \\ 0 & 1 \end{bmatrix}, \quad \text{where } n = 1, 10, 100, 1000.$$

The condition number $\kappa(A) = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}$ of the matrix A is the ratio of the largest eigenvalue and the smallest eigenvalue of A, in our case the largest diagonal entry over the smallest. In our context, a high condition number means that the function's contour lines are ellipses that are wide in one direction and narrow in another. For each n, run the steepest descent method until convergence (you may have to increase the `k_max` parameter). In each case, compute $\kappa(A_n)$, and the convergence rate ρ_n. Plot $\kappa(A_n)$ on the x-axis against ρ_n on the y-axis. Submit the plot, as well as a printout of the values or $\kappa(A_n)$ and ρ_n.

6. Finally, let's try a different search direction. Instead of the steepest descent direction $p_k = -\nabla f(x_k)$, you will use the Newton step

$$p_k = -\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k),$$

with step length $\alpha = 1$. The Newton method is based on approximating the objective function f locally by a quadratic function. Applying it to the quadratic function results in convergence in one step (try it!). To be fair, we will compare the performance of the steepest descent direction and the Newton direction for the Rosenbrock function (with scaling parameter $A = 10$). Run `steepest_descent` using (i) $p_k = -\nabla f(x_k)$ with backtracking, and (ii) $p_k = -\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$ and $\alpha = 1$. In each case, plot the convergence graphs (submit them). Show computationally that the Newton algorithm converges quadratically, i.e. $\|x_{k+1} - x^*\| \leq M\|x_k - x^*\|^2$ as $k \to \infty$. Explain your work and submit a printout of your results.