Numerical Optimization - Hand-In 3

Isabela Blucher

February 23, 2018

Exercise 3.3

We first define a strongly convex quadratic function of the form $f(x) = \frac{1}{2}x^TQx + b^Tx, Q > 0$ with gradient $\nabla f(x) = Qx + b$. Now, we suppose p is a descent direction and we know $\phi(\alpha) = f(x + \alpha p), \alpha \geq 0$. Any minimizer α^* of $\phi(\alpha)$ satisfies that $\phi'(\alpha^*) = \nabla f(x + \alpha^* p)^T p = 0$.
The one-dimensional minimizer is unique and from the definitions above we know that

$$[Q(x + \alpha^* p) + b]^T p = (Qx + b)^T p + \alpha^* p^T Q p = 0$$

Manipulating the equation above and using the gradient definition we find the expression given by (3.55)

$$\alpha^* = -\frac{(Qx + b)^T p}{p^T Q p} = -\frac{\nabla f(x)^T p}{p^T Q p}$$

Exercise 3.4

Let the strongly convex quadratic function be of the form $f(x) = \frac{1}{2}x^TQx + b^Tx + d$ with Q positive-definite. Also x_k is the current iterate, p_k is a descent direction and $0 < c < \frac{1}{2}$. From Exercise 3.3 we know that the one-dimensional minimizer along $x_k + \alpha_k p_k$ is $\alpha^* = -\frac{\nabla f(x)^T p}{p^T Q p}$.
Substituting the α_k in $f(x_k) + (1 - c)\alpha_k \nabla f_k^T p_k$ we obtain

$$f(x_k) - \frac{(\nabla f_k^T p_k)^2}{p_k^T Q p_k} + c\frac{(\nabla f_k^T p_k)^2}{p_k^T Q p_k}$$

Since we know $\nabla f_k = Qx_k + b$, manipulating the previous expression we get

$$f(x_k + \alpha_k p_k) = f(x_k) - \frac{(\nabla f_k^T p_k)^2}{p_k^T Q p_k} + \frac{1}{2}\frac{(\nabla f_k^T p_k)^2}{p_k^T Q p_k}$$

From this equation we can see that the first inequality in the Goldstein condition is satisfied.
For the second inequality we reduce the previous expression to

$$f(x_k + \alpha_k p_k) = f(x_k) - \frac{1}{2}\frac{(\nabla f_k^T p_k)^2}{p_k^T Q p_k}$$

which is definitely smaller than

$$f(x_k) + c\alpha_k \nabla f_k^T p_k = f(x_k) - c\frac{(\nabla f_k^T p_k)^2}{p_k^T Q p_k}$$

And thus, we can confirm that the one-dimensional minimizer α^* of a strongly convex quadratic function f, always satisfies the Goldstein conditions.

Programming Assignment

Exercise 3.1 of the book states "Program the steepest descent and Newton algorithms using the backtracking line search, Algorithm 3.1. Use them to minimize the Rosenbrock function (2.22). Set the initial step length $\alpha_0 = 1$ and print the step length used by each method at each iteration. First try the initial point $x_0 = (1.2, 1.2)^T$ and then the more difficult starting point $x_0 = (-1.2, 1)^T$."

Table 1: Performance of the line-search algorithms on the Rosenbrock function

Algorithm	Initial step-length (α)	Starting point	Number of iterations until convergence
Steepest descent	1	(1.2, 1.2)	1231
		(-1.2, 1)	1940
Newton's method	1	(1.2, 1.2)	9
		(-1.2, 1)	22

The results on Table 1 illustrate that the convergence rate for the Newton's method is much faster than for the steepest descent algorithm. Both algorithms were implemented with the backtrack line-search to define the step-length for each iteration.

The update of x_k for each iteration $k = 1, 2, ...$ is dependent on the current step-length and descent direction, which means that the stopping criteria is related to that. The selected stopping criteria for the algorithms is that the norm of the gradient of the current x_k is less or equal a small ε, that is $\|\nabla f(x_k)\| \leq \varepsilon$. As we want the algorithm to find a minimizer, and the value of the gradient at the minimizer is 0, we pick very small values (for example $\varepsilon = 10^{-6}$ for our stopping threshold ε. Another stopping criteria used as a "backup" for the first one is the number of iterations of the algorithm. We can set a large enough number (for example 10^6) of iterations to be the maximum and if it doesn't converge to a minimizer before that, it will stop anyway.

The performance of the algorithms are very sensitive to changes in the parameters for the back-tracking line search. Table 2 shows how much it affects the algorithms' rate of convergence in terms of number of iterations.

Table 2: Performance of the algorithms with relation to the back-tracking line search parameters for starting point (1.2, 1.2)

	c	ρ	Number of iterations
Steepest descent	10^{-4}	10^{-4}	2329
	10^{-4}	0.5	18692
	0.5	10^{-4}	2328
	0.5	0.5	1231
Newton's method	10^{-4}	10^{-4}	6712
	10^{-4}	0.5	9
	0.5	10^{-4}	68496
	0.5	0.5	9

It seems that for the steepest descent algorithm the best value for both the parameters is 0.5, which is when the step-length α is the most optimal for each iteration and thus, the convergence to the minimizer is quicker.
For Newton's method, the best choice is a big value of ρ, where, for either a big or small value of c, it took 9 iterations for the algorithm to converge to a minimizer. These tests conclude that the algorithms can be very sensitive to significant changes in the parameters c and ρ of the back-tracking line search.

For the plot, the convergence was measured in terms of the euclidean norm of the current x_k to the minimizer, so $\|x_k - x^*\|$. The graph was generated by setting fixed ranges for the axes for both algorithms, so both could be plotted in the same figure. Also, both algorithms had starting point (-1.2, 1) and parameters c = $\rho = 0.5$.

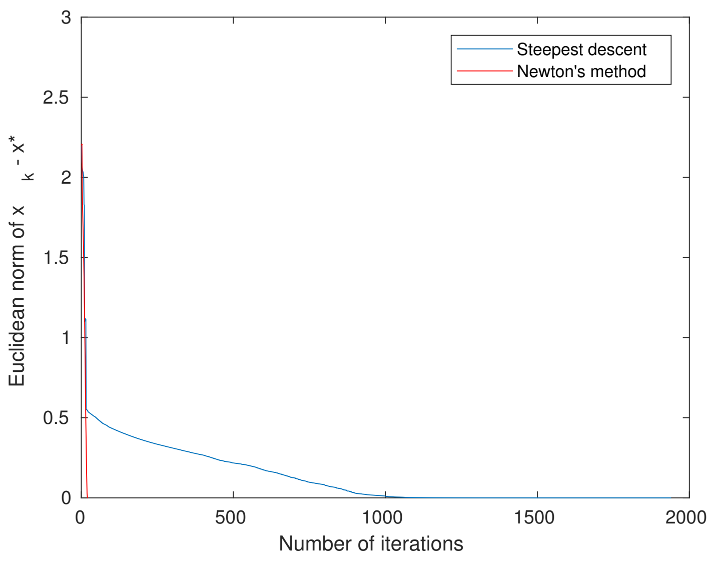

Figure 1: Plot of the convergence rates of both algorithms

Figure 1 shows, by visual inspection that the convergence rate for both algorithms were as expected in relation to the theorems in the textbook.

For starting iterates, one should be concerned with testing for all cases. Starting with points which are close to our minimizer, and therefore, according to the textbook theorems, should converge to a solution and also testing for points which may "break" the algorithm, which means that the iterations would stop not on account of convergence but maximum number of iterations. It is important to examine in the latter case if the algorithm was able to find an approximate point to the expected x^*. Measuring an algorithm in terms of robustness means that we have to be able to find an approximation even for numerically challenging inputs and in terms of efficiency means that it should converge fast enough to our expected solution.