Numerical Optimization Lab 04: Steepest Descent and Backtracking Strategy

Francesco Della Santa

Abstract

In this lesson, we will implement the *steepest descent* optimization method with the *backtracking strategy*.

1 Exercises

Exercise 1 (Backtracking Implementation for Steepest Descent)**.** Write a Matlab function *steepest_desc_bcktrck.m* that implements the *steepest descent* optimization method with the *backtracking strategy* (see Appendix A) and that takes the following inputs and outputs.

INPUTS:

- `x0`: a *column vector* of n elements representing the starting point for the optimization method;
- `f`: a *function handle* variable that, for each column vector $\boldsymbol{x} \in \mathbb{R}^n$, returns the value $f(\boldsymbol{x})$, where $f : \mathbb{R}^n \to \mathbb{R}$ is the *loss function* that have to be minimized;
- `gradf`: a *function handle* variable that, for each column vector $\boldsymbol{x} \in \mathbb{R}^n$, returns the value $\nabla f(\boldsymbol{x})$ as a *column* vector, where $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ is the *gradient* of f;
- `alpha0`: a *real scalar value* characterizing the step length of the optimization method;
- `kmax`: an *integer scalar value* characterizing the maximum number of iterations of the method;
- `tolgrad`: a *real scalar value* characterizing the tolerance with respect to the norm of the gradient in order to stop the method.
- `c1`: the factor c_1 for the Armijo condition that must be a scalar in $(0, 1)$;
- `rho`: fixed factor, less than 1, used to reduce α;
- `btmax`: maximum number of steps allowed to update α during the backtracking strategy.

OUTPUTS:

- `xk`: the last vector $\boldsymbol{x}_k \in \mathbb{R}^n$ computed by the optimization method before it stops;
- `fk`: the value $f(\boldsymbol{x}_k)$;
- `gradfk_norm`: the euclidean norm of $\nabla f(\boldsymbol{x}_k)$;
- `k`: index value of the last step executed by the optimization method before stopping;
- `xseq`: a matrix/vector in $\mathbb{R}^{n \times k}$ such that each column j is the j-th vector $\boldsymbol{x}_j \in \mathbb{R}^n$ generated by the iterations of the method.
- `btseq`: row vector in $\mathbb{R}^k$ such that the j-th element is the number of backtracking iterations done at the j-th step of the steepest descent.

Once you have written the function, test it using the data inside the file *test_functions.mat* with `alpha0` $= 5$, `c1` $= 10^{-4}$, `rho` $= 0.8$ and `btmax` $= 50$; then, plot:

- the loss f (given in *test_functions.mat*) with the sequence `xseq` using the Matlab function `contour`;
- the barplot of the values in `btseq` using the function `bar`;
- the loss f with the sequence `xseq` using the Matlab function `surf`.

A Backtracking

Let the function $f : \mathbb{R}^n \to \mathbb{R}$ be given. The backtracking strategy for an iterative optimization method consists of looking for a value α_k satisfying the Armijo condition at each step k of the method, i.e.

$$f(\underbrace{\boldsymbol{x}_{k+1}}_{\boldsymbol{x}_k+\alpha_k\boldsymbol{p}_k}) \leq f(\boldsymbol{x}_k) + c_1\alpha_k\nabla f(\boldsymbol{x}_k)^\top \boldsymbol{p}_k\,, \tag{1}$$

where $c_1 \in (0,1)$ (typically, the standard choice is $c_1 = 10^{-4}$).

We recall that the Armijo condition suggests that a "good" α_k is such that you have a sufficient decrease in f and, moreover, the function value at the new point $f(\boldsymbol{x}_{k+1}) = f(\boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k)$ is under the "reduced tangent hyperplane" of f at $\boldsymbol{x}_k$[1]. To better explain the Armijo condition, we look at the function $\phi(\alpha) := f(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k)$, such that $\phi'(\alpha) = \nabla f(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k)^\top \boldsymbol{p}_k$ and $\phi'(0) = \nabla f(\boldsymbol{x}_k)^\top \boldsymbol{p}_k$ (see Figure 1).

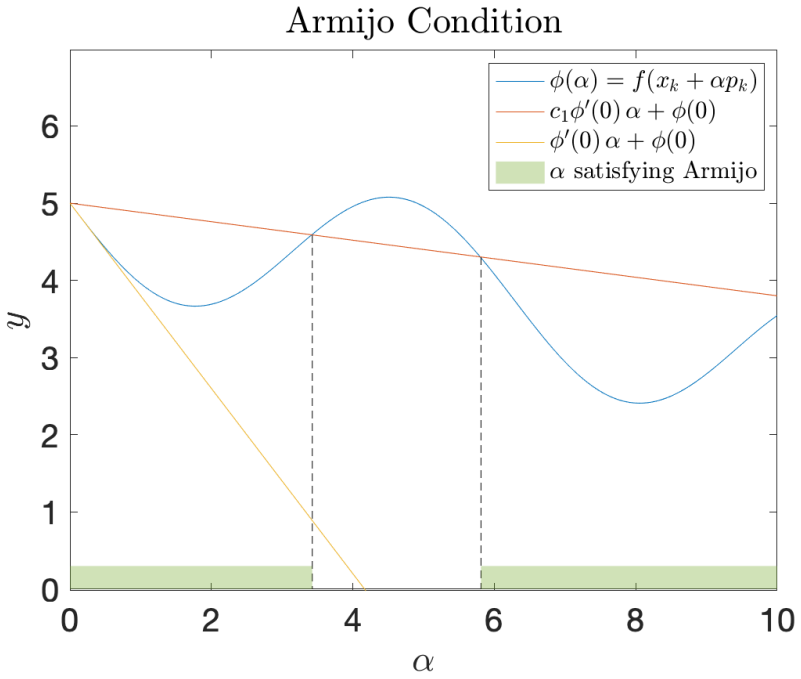

Figure 1: Example of the Armijo condition.

The backtracking strategy is an *iterative process* that looks for this value α_k. Given an arbitrary factor $\rho \in (0,1)$ and an arbitrary starting value $\alpha_k^{(0)}$ for α_k, we decrease iteratively $\alpha_k^{(0)}$, multiplying it by ρ, until the Armijo condition is satisfied. Then $\alpha_k = \rho^{t_k}\alpha_k^{(0)}$, for a $t_k \in \mathbb{N}$, if it satisfies Armijo but $\rho^{t_k-1}\alpha_k^{(0)}$ does not.

[1] i.e., the point $(\boldsymbol{x}_{k+1}, f(\boldsymbol{x}_{k+1})) \in \mathbb{R}^{n+1}$ is under the hyperplane $\{\boldsymbol{\xi} \in \mathbb{R}^{n+1} \mid [c_1 f_{x_1}(\boldsymbol{x}_k), \ldots, c_1 f_{x_n}(\boldsymbol{x}_k), -1] \cdot \boldsymbol{\xi} + f(\boldsymbol{x}_k) = 0\}$.

Remark A.1 (Few things to keep in mind)**.**

- The Armijo condition is often satisfied for very small values of α. Then, it is not enough to ensure that the algorithm makes reasonable progress; indeed, if α is too small, unacceptably short steps are taken.
- For simplicity, we consider ρ as a fixed parameter, but it can be chosen using already available information, changing with the iterations;
- Other conditions could be imposed to guarantee that not too short steps are taken (e.g., Wolfe conditions[2]), but they are not practical to be implemented.
- Practical implementations, instead of imposing a second condition, frequently use the backtraking strategy.
- The choice of $\alpha_k^{(0)}$ is method-dependent, but it is *crucial* to choose $\alpha_k^{(0)} = 1$ in Newton/Quasi-Newton methods for (possibly) get the second-order rate of convergence.
- In general, globalizing strategies (as the backtracking strategy) are methods specifically designed to help Newton's methods when we are far from the solution. Then, they are usually designed in such a way that they work only when needed: if close enough to $\boldsymbol{x}^*$, they are "switched off" so that Newton's works and eventually fast convergence is maintained.

[2] i.e., Armijo condition and curvature condition $\nabla f(\boldsymbol{x}_{k+1})^\top \boldsymbol{p}_k \geq c_2 \nabla f(\boldsymbol{x}_k)^\top \boldsymbol{p}_k$, $c_2 \in (c_1, 1)$.