Numerical Optimization Lab 05:
Newton Method and Backtracking Strategy

Francesco Della Santa

Abstract

In this lesson, we will learn to implement the *Newton* optimization method with and without the *backtracking strategy*.

1 Exercises

Exercise 1 (Implementation of the Newton Method)**.** Write a Matlab function *newton.m* that implements the *Newton* optimization method (see Appendix A) and that takes the following inputs and outputs. **Sugestion:** for simplicity, use direct methods for solving linear systems (i.e., the "\" symbol in Matlab).

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;
- `Hessf`**:** a *function handle* variable that, for each column vector $\boldsymbol{x} \in \mathbb{R}^n$, returns the matrix $H_f(\boldsymbol{x}) \in \mathbb{R}^{n \times n}$, where H_f is the *Hessian matrix* of f;
- `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.

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 vector j-th vector $\boldsymbol{x}_j \in \mathbb{R}^n$ generated by the iterations of the method.

Once you have written the function, test it using the data inside the file *test_functions2.mat*. In particular, plot:

- the loss f_i (given in *test_functions2.mat*), $i = 1, 2, 3$, using the Matlab function `contour` and the sequence `xseq`;

- the loss f_i (given in *test_functions2.mat*), $i = 1, 2, 3$, using the Matlab function `surf` and the sequence `xseq`.

Exercise 2 (Backtracking Implementation for Newton)**.** Write a Matlab function *newton_bcktrck.m* that implements the *Newton* optimization method with the *backtracking strategy* (see Appendix C) and that takes the following inputs and outputs. **Sugestion:** for simplicity, use direct methods for solving linear systems (i.e., the "\" symbol in Matlab).

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;

`Hessf`**:** a *function handle* variable that, for each column vector $\boldsymbol{x} \in \mathbb{R}^n$, returns the matrix $H_f(\boldsymbol{x}) \in \mathbb{R}^{n\times n}$, where H_f is the *Hessian matrix* of f;

`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, lesser than 1, used for reducing α_0;

`btmax`**:** maximum number of steps for updating 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 vector 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_functions2.mat* with `c1` $= 10^{-4}$, `rho` $= 0.8$ and `btmax` $= 50$; then, plot together:

- the loss f_i (given in *test_functions2.mat*), $i = 1, 2, 3$, using the Matlab function `contour` and the sequence `xseq`;
- the loss f_i (given in *test_functions2.mat*), $i = 1, 2, 3$, using the Matlab function `surf` and the sequence `xseq`.
- the barplot of the values in `btseq` using the function `bar`, with respect to the functions f_i.

Exercise 3 (Direction Check for Newton)**.** Improve the function of the previous exercise adding an extra input variable called "`verbose`", that must be a boolean value (i.e., `true` or `false`). If `verbose` is `true` and the direction $\boldsymbol{p}_k$ is not a descent direction, let the function returns a *warning* that informs the user about the problem.

Exercise 4 (Modified Newton - Homework)**.** Create a new matlab function that implements a Modified Newton method. **Suggestion:** starts from the function of the previous exercise and, instead of returning a warning when $\boldsymbol{p}_k$ is an ascent direction, fix the problem computing $B_k = H_f(\boldsymbol{x}_k) + \textit{Correction}$.

A Newton

Let the function $f : \mathbb{R}^n \to \mathbb{R}$ be given. The Newton descent method is an iterative optimization method that, starting from a given vector $\boldsymbol{x}_0 \in \mathbb{R}^n$, computes a sequence of vectors $\{\boldsymbol{x}_k\}_{k\in\mathbb{N}}$ characterized by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha \boldsymbol{p}_k \,, \quad \forall\, k \geq 0 \,, \tag{1}$$

where the descent direction $\boldsymbol{p}_k$ is solution of the linear system

$$H_f(\boldsymbol{x}_k)\boldsymbol{p} = -\nabla f(\boldsymbol{x}_k) \,, \tag{2}$$

and ∇f, H_f are the gradient and the Positive Definite[1] (PD) Hessian matrix of f, respectively.

A.1 Recalling the Motivations of Newton

We can approximate $f(\boldsymbol{x}+\boldsymbol{p})$, with a Taylor expansion, as a quadratic model, i.e.:

$$f(\boldsymbol{x}+\boldsymbol{p}) \simeq f(\boldsymbol{x}) + \boldsymbol{p}^\top \nabla f(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{p}^\top H_f(\boldsymbol{x})\,\boldsymbol{p} =: m_{f,\boldsymbol{x}}(\boldsymbol{p}) \,. \tag{3}$$

Assuming that $H_f(\boldsymbol{x})$ is Positive Semidefinite, for each $\boldsymbol{p} \in \mathbb{R}^n$, we have that $\boldsymbol{p}^\top H_f(\boldsymbol{x})\,\boldsymbol{p} \geq 0$ and, therefore, $m_{f,\boldsymbol{x}}(\boldsymbol{p})$ is *convex*[2]. Then, we can compute the $\boldsymbol{p}^*$ that minimizes $m_{f,\boldsymbol{x}}(\boldsymbol{p})$, i.e. the $\boldsymbol{p}^*$ such that

$$\underbrace{\nabla f(\boldsymbol{x}) + H_f(\boldsymbol{x})\boldsymbol{p}^*}_{=\nabla_{\boldsymbol{p}}\, m_{f,\boldsymbol{x}}(\boldsymbol{p}^*)} = \boldsymbol{0} \,. \tag{4}$$

In other words, $\boldsymbol{p}^*$ is solution of the linear system (2) and is a *descent direction* for $f(\boldsymbol{x})$.

In the end, we recall the PROs and CONs of the Newton method:

- (+) Under proper assumptions ($H_f(\boldsymbol{x}_k)$ PD, $\alpha = 1$, $\boldsymbol{x}_0$ "good" starting point) the Newton method has fast (quadratic) rate of convergence;
- (−) Computationally more expensive than Steepest Descent (computation/storage of $H_f(\boldsymbol{x}_k)$ and computation of $\boldsymbol{p}_k$ solving (2));
- (−) $\boldsymbol{p}_k$ is a descent direction only if $H_f(\boldsymbol{x}_k)$ is PD $\Rightarrow$ Sometimes is useful to work with a "corrected" Hessian matrix, if $H_f(\boldsymbol{x}_k)$ is non-PD; i.e., we can use a *PD* matrix $B_k := H_f(\boldsymbol{x}_k) + \textit{Correction}$ (see the *Modified Newton* method in the course slides of Prof. Pieraccini).

B The Rosenbrock and the Himmelblau Test Functions

When you have to test an optimization method, it is useful to use some "standard" test functions[3]. In this laboratory we use the the 2-dimensional *Rosenbrock* function and the *Himmelblau* function that are the functions f_2 and f_3, respectively, in the file *test_functions2.mat*.

- **Rosenbrock** ($n = 2$):

$$\begin{aligned} f_2(x_1,x_2) &= 100(x_2 - x_1^2)^2 + (1-x_1)^2 \\ \nabla f_2(x_1,x_2) &= \begin{bmatrix} 400x_1^3 - 400x_1x_2 + 2x_1 - 2 \\ 200(x_2 - x_1^2) \end{bmatrix} \\ H_{f_2}(x_1,x_2) &= \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix} \end{aligned} \tag{5}$$

[1] $A \in \mathbb{R}^{n\times n}$ is Positive/Negative Definite iff, for each $\boldsymbol{x} \in \mathbb{R}^n$, it holds $\boldsymbol{x}^\top A\,\boldsymbol{x} \gtrless 0$ (Semidefinite iff $\gtreqless 0$).

[2] A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex/concave iff $f(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) \lesseqgtr t\,f(\boldsymbol{x}_1) + (1-t)f(\boldsymbol{x}_2)$, for each $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n$, for each $t \in [0,1]$.

[3] E.g., see https://en.wikipedia.org/wiki/Test_functions_for_optimization

- **Himmelblau**:

$$
\begin{aligned}
f_3(x_1, x_2) &= (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \\
\nabla f_3(x_1, x_2) &= \begin{bmatrix} 4x_1^3 + 4x_1x_2 - 40x_1 + 2x_2^2 - 14 \\ 4x_2^3 + 4x_1x_2 - 26x_2 + 2x_1^2 - 22 \end{bmatrix} \\
H_{f_3}(x_1, x_2) &= \begin{bmatrix} 12x_1^2 + 4x_2 - 40 & 4x_1 + 4x_2 \\ 4x_1 + 4x_2 & 12x_2^2 + 4x_1 - 26 \end{bmatrix}
\end{aligned}
\tag{6}
$$

C 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.

$$
f(\underbrace{\boldsymbol{x}_{k+1}}_{\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k}) \leq f(\boldsymbol{x}_k) + c_1 \alpha_k \nabla f(\boldsymbol{x}_k)^\top \boldsymbol{p}_k \,, \tag{7}
$$

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 $\boldsymbol{x}_k$.

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.

Remark C.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[4]), 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 problem-dependent and/or method-dependent. In particular, 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 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.

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