Challenge 1. A gradient method for the minimization of a multivariate function

Luca Formaggia, Alberto Artoni, Beatrice Crippa

Academic Year 2023-2024

Consider the following minimization problem.
Let $f : \mathbb{R}^n \to \mathbb{R}$ be a given function that has a minimum. Find $\mathbf{x} \in \mathbb{R}^n$ such that

$$\mathbf{x} = \underset{\mathbf{y} \in \mathbb{R}^n}{\operatorname{argmin}} \ f(\mathbf{y}). \tag{1}$$

A very common (yet not the most efficient) technique for solving this problem is the *gradient method*:
Given an initial guess $\mathbf{x}_0$, do

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k), \quad k = 0, 1, \ldots, k_{max}, \tag{2}$$

until one of the following conditions is satisfied:

- $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| < \epsilon_s$ (control on the step length);
- $|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \epsilon_r$ (control on the residual);
- $k > k_{max}$ (limit of the max number of iterations, in which case we have no convergence).

The choice of the step α_k (also called learning rate in the machine learning context) is critical. By taking μ small, e.g. $\mu = 0.2$, and given an initial value α^0, we can choose α_k with the following strategies.

- Exponential decay: $\alpha_k = \alpha^0 e^{-\mu k}$.
- Inverse decay: $\alpha_k = \dfrac{\alpha^0}{1 + \mu k}$.
- Approximate line search: find α_k so that $f(x_k) - f(x_{k+1})$ is sufficiently large.

A possible technique to solve the line search problem is the Armijo rule, an effective but costly method. The Armijo rule computes α_k as follows:
given α^0 and a parameter $\sigma \in (0, 0.5)$ check if the sufficient decrease condition

$$f(\mathbf{x}_k) - f(\mathbf{x}_k - \alpha^0 \nabla f(\mathbf{x}_k)) \geq \sigma \alpha^0 \|\nabla f(\mathbf{x}_k))\|^2, \tag{3}$$

is satisfied. If not, set $\alpha^0 = \alpha^0/2$ and repeat. The value that finally satisfies the condition is the desired α_k.

1 The challenge

Write a function that takes in input two *function wrappers* that defines f, ∇f, the initial condition, the tolerances, the initial step α^0, the maximal number of iterations and all other needed parameters and returns the computed minumum. Aggregate all parameters in a `struct`.
Implement the Armijo rule (but if you want you can let the user choose among different strategies). For the treatment of vectorial quantities, since we have not seen the Eigen library yet, you may use `std::vector` and you need to implement the function for the norm of a standard vector. But if you want to try, use the Eigen vectors.
You are free to choose how to organize your code and to use any of the techniques seen during lectures and labs. The originality of the code is valued. The important requirements are:

- The code should be contained in a `git` repository mirrored on GitHub;
- The code must have a `README.md` with a minimal explanation;
- The code must have *a working makefile*: we should be able to compile your code just by typing `make`;
- You should write comments in your code to help us understand what you are doing and why;
- The main should enable us to run a test case where the function to be minimized is

$$f(\mathbf{x}) = f(x_1, x_2) = x_1x_2 + 4x_1^4 + x_2^2 + 3x_1,$$

  with starting point $(0, 0)$ and tolerances $\epsilon_r = 10^{-6}$, $\epsilon_s = 10^{-6}$.

1.1 Suggestions

- If you want to give the user the choice of different strategies for the computation of α_k, remember that the use of an if statement inside a loop is computationally inefficient. Since we are not dealing with classes and polymorphism yet, a possibility (but then the choice cannot be made runtime) is to create a function template with an enumerator as template parameter, and then select the choice with `if constexpr`. In this case, you do not loose efficiency at the price of less flexibility.
- You may try to write also a function that computes the gradient by finite differences (we have seen an example at a lecture) and give the user the possibility to choose that instead of the exact gradient. You can exploit the flexibility of function wrappers to make the selection of the two options simple.
- You may want to try to define the function and the derivative using the `muParser` facility and read the functions from a file. You loose efficiency but gain in flexibility. It is more complex for the derivative (for vector functions you need `muParserX`).

2 General rules for the Challenges

- When finished, write the link to the GitHub repository on the `WeBeep` site (Challenges section) so that I can clone it.
- Challenges are meant to be a tool to help you exercise programming, put in practice what seen at the lecture, and get ready for the project. Not really to grade you. They are graded in order to give some satisfaction to the ones of you who are making the effort and do things nicely, but this is not their main purpose.

  So, if you need help, ask for it. Use the Forum on `WeBeep`, so that the answer may be useful also to others.
- As I said, you are free to choose how to organize data, how to provide the input to your code, or to name variables etc. For instance, you may decide to write a functor instead of a simple function. Try to put in practice what you have seen at the lectures and write a clean code.

3 Extras

We warmly invite you to try to implement other schemes. Your extra commitment will be appreciated and taken into account.
Consider for instance the following *momentum* or *heavy-ball method.*
The minimization problem 1 is solved by taking the following update rule:
given $\mathbf{d}_0 = -\alpha_0 \nabla f(\mathbf{x}_0)$

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{d}_k, \\ \mathbf{d}_{k+1} &= \eta \mathbf{d}_k - \alpha_{k+1} \nabla f(\mathbf{x}_{k+1}). \end{aligned} \tag{4}$$

Here, η is the memory parameter. Sometimes it has a fixed value $\eta \simeq 0.9$, or it may be taken as $(1 - \alpha_k)$ (we must ensure that $\alpha_k < 1$).
In this method we cannot apply Armijo rule since the direction $\mathbf{d}_k$ cannot be guaranteed to be a descent direction. You either use a fixed number α or you use other techniques.
A different (equivalent) formulation of the heavy ball (with which you understand the name better) is: *set* $\mathbf{x}_1 = \mathbf{x}_0 - \alpha_0 \nabla f(\mathbf{x}_0)$ *and then, for* $k = 1, 2, \ldots$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) + \eta(\mathbf{x}_k - \mathbf{x}_{k-1}) \tag{5}$$

A further variant is Nesterov iteration where the previous iterate is replaced by

$$\begin{aligned} \mathbf{y} &= \mathbf{x}_k + \eta(\mathbf{x}_k - \mathbf{x}_{k-1}) \\ \mathbf{x}_{k+1} &= \mathbf{y} - \alpha_k \nabla f(\mathbf{y}) \end{aligned} \tag{6}$$

Havy-ball and Nesterov have better convergence properties, at the price of have to fix two hyperparameters instead of just one.
The literature on these class of methods is huge since they are adopted in the context of deep learning.
Another interesting method of this class is ADAM, whose description is easily found on the web. It is more useful for large scale optimization problems in the presence of noise, but one may also implement it for simple optimization problems like the ones of this challenge, just for fun!