Line Search

Luca Formaggia

February 2022

1 Line search algorithm for unconstrained minimization problems

A set of techniques for the unconstrained minimization problem: *Find* $\mathbf{x} \in \mathbb{R}^n$ *such that*

$$\mathbf{x} = \operatorname{argmin}_{\mathbf{y}\in\mathbb{R}^n} f(\mathbf{y}),$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a cost function, are based on the following general algorithm.

```
Data: $\mathbf{x}_0 \in \mathbb{R}^n$, $\epsilon > 0$
Compute a descent direction $\mathbf{d}_k \in \mathbb{R}^n$;
Compute a good step $\alpha_k$;
Advance $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$;
Stop if $|f\mathbf{x}_{k+1}| \leq \epsilon$;
Result: $\mathbf{x} = \mathbf{x}_{k+1}$
```

Algorithm 1: Line Search

In this example we limit to schemes that uses only the knowledge of the gradient ∇f, or of an approximation of it. The structure can be modified with some (but not great) effort to account also for Newton method, which requires the knowledge of the Hessian. Indeed, I have implemented already some quasi-Newton methods, which build an approximation of the Hessian (or of its inverse).
It is not the scope of this note to give a description of line search algorithms, which can be found in good references like J. Nocedal and S. Wright, Numerical Optimization, or also the nook Numerical Mathematics by A.Quarteroni and F. Saleri. Here, we want to provide some insight on the algorithmic choices I have made.

1. I rely on the *Eigen* library for numerical algebra operations. The use of traits allows changing the basic types and eases a possible refactoring to support a different library;

2. The descent direction may be done with different algorithms, for instance *gradient* (also called steepest descent), *BFGS* etc. I decided to use polymorphism. So they all derive from a base class, DescentDirectionBase.

3. I have constructed a factory for the different descent direction classes. However, here, differently than in the example on numerical quadrature, I have not made the factory a global variable, nor implemented the automatic loading of rules in the factory. The reason is that I wanted to show that in simple situations it is not needed to get into that complexity. Of course, the factory and the leading may be in the future adapted for a more plugin-type architecture.

4. The choice of of the step α_k is here made in the method backtrack with a simple backtracking (Armijo rule) that imposes the first Wolfe condition (sufficient decrease condition). However, in the aggregate storing the options for the algorithm I have already set a possible coefficient for the second Wolfe condition (curvatore condition), for possible later extensions.

5. The polymorphic classes for the descent direction are *clonable*. In this specific example it is not so relevant, but in future development it is always better to enable the composition with a polymorphic object support copy operations.

2 Bound constrained problem

Unconstrained minimization techniques may be easily extended to the box constrained case: *Find* $\mathbf{x} \in \mathbb{R}^n$ *such that*

$$\left\{ \mathbf{x} = \operatorname{argmin}_{\mathbf{y} \in \mathbb{R}^n} f(\mathbf{y}), \mathbf{a} \leq \mathbf{x} \leq \mathbf{b}. \right.$$

where $\mathbf{a} \in [\{-\infty\} \cup \mathbb{R}]^n$ and $\mathbf{b} \in [\{+\infty\} \cup \mathbb{R}]^n$ are bounds. If we indicate with Π the projection on the interval $\mathcal{I}$m defined by the bounds, the general algorithms changes into

```
**Data:** $\mathbf{x}_0 \in \mathcal{I}$, $\epsilon_1, \epsilon_2 > 0$
Compute a descent direction $\mathbf{d}_k \in \mathbb{R}^n$;
Compute a good step $\alpha_k$ so that $(\mathbf{x}_k + \alpha_k \mathbf{d}_k) \in \mathcal{I}$;
Advance $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$;
Stop if $|f\mathbf{x}_{k+1}| \leq \epsilon_1$ or $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \leq \epsilon_2$ ;
**Result:** $\mathbf{x} = \mathbf{x}_{k+1}$
```

Algorithm 2: Line Search for Box-Constraints

These methods are normally called "projected methods". You find the description in any good book. The version for bounds given by intervals is simple, more complex situations arise when the constraints are general convex sets.

3 Other possible extensions

- You can add many other formulas for the computation of descent direction: *L-BFGS*, *gradient with momentum (heavy ball) .....*
- You can implement a more sophisticated backtracking that enforces also the second Wolfe condition (curvature condition). However, in this case, I will let the user having the possibility of choosing whether activating it or not. The procedure for satisfying a curvature condition is indeed quite heavy. The advantage is that you guarantee convergence of certain methods that (according to the theory) may fail if you enforce only the first condition.
- More complex, but possible, is to use the present framework in the context of "big data" and implement a *Stochastic gradient* (which is a line search technique). However, maybe in that case is better start from scratch.
- Implement Newton method. Here you need to expand the aggregate providing the input information to account for a function that represent the Hessian.