Exercise: Comparing Gradient-Based Algorithms on Convex Quadratic Functions

Introduction to Optimization lecture
at Ecole Centrale Paris / ESSEC Business School

Dimo Brockhoff
firstname.lastname@inria.fr

November 13, 2015

Abstract

In the lecture, we have seen the general idea of numerical optimization algorithms, following decent directions and a few concrete examples of decent directions as well as line search variants. Here, we will have a closer look on their performance on specific convex quadratic functions of the form $f(x) = \mathbf{x}^T A\mathbf{x}$.

1 Line Search in Descent Algorithms

The purpose of this first part is to understand the interest of implementing a line search procedure in descent algorithms. Algorithm 1 (in Section 4) reminds the general scheme for a descent algorithm in which the line search is called in line 4.

We start by considering a *gradient descent algorithm* where at each iteration, the descent direction corresponds to the negative gradient of the objective function $f : \mathbb{R}^n \to \mathbb{R}$ to be minimized, that is

$$\mathbf{d}_k = -\nabla f(\mathbf{x}_k) \ .$$

The result of the line-search procedure is called the step-size and is denoted as σ_k, that is

$$\sigma_k = \text{LineSearch}(\mathbf{x}_k, \mathbf{d}_k)$$

such that the update of the current solution $\mathbf{x}_k$ reads

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \sigma_k \mathbf{d}_k \ .$$

We consider first a degenerated line-search procedure that consists in taking a *constant step-size* equal to $\sigma > 0$. In this first part of the exercise, we will test the *gradient descent algorithm* with *constant step-size* on the functions

$$f_\alpha(\mathbf{x}) = \alpha \sum_{i=1}^{n} \mathbf{x}_i^2 \ , \alpha > 0 \ .$$

1. What is the optimum of f_α?
2. Compute the gradient of f_α.
3. Implement a function `falpha` that takes as input a vector `x` of $\mathbb{R}^n$ and a scalar $\alpha \in \mathbb{R}$ and outputs $f_\alpha(\mathbf{x})$. Likewise, implement a function `gradientFalpha` that takes as input $\mathbf{x} \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ and outputs the gradient of f_α in $\mathbf{x}$.
4. Plot a few level sets of your own choice for $f_\alpha(\mathbf{x})$ and $\alpha = 1/2$. Useful python commands for plotting are contained in the `pyplot` module of `matplotlib` which you can load as

   ```
   from matplotlib import pyplot as plt
   ```

   and which provides a MATLAB-like interface. Probably the best way to plot level sets is via the `contour` command which you can evoke as

   ```
   plt.contour(X,Y,Z)
   ```

   and in which `X` and `Y` can be created for example by

   ```
   delta = 0.025
   x = np.arange(-2.0, 2.0, delta)
   y = np.arange(-2.0, 2.0, delta)
   X, Y = np.meshgrid(x, y)
   ```

and where `Z` is a two-dimensional numpy array with the f-values to display (i.e. `Z[i][j]` should contain the function value to be displayed in the point `(X[i][j],Y[i][j])`). Remember that adding a '?' after a python command in ipython displays the corresponding help.

5. Implement the gradient descent algorithm with fixed step-size. Save the sequence $\mathbf{x}_k$. Implement as stopping criteria: a maximum of iterations equal to 10^6 and $\|\nabla f(\mathbf{x}_k)\| \leq 10^{-12}$. We advise to write a function that takes as input the objective function, the gradient function, the initial search point, and the step-size σ and returns the sequence $\mathbf{x}_k$ (and implicitly the number of iterations to reach the stopping criterion).

6. Consider $\alpha = 1/2$ and $\sigma = 0.1$. For $n = 2$, plot the trajectory of the algorithm, that is, plot the evolution of the vectors $\mathbf{x}_k$ in the 2D-plane. We will consider two runs, the first one initialized at $(10, 10)$ and the second one at $(-5, 10)$. Comment what you observe and explain.

7. For $n = 10$, $\sigma = 0.1$, consider the functions f_α for $\alpha = 1/2$, $\alpha = 1/20$ and $\alpha = 1/200$. Initialize the algorithm at $\mathbf{x}_0 = (10, \ldots, 10)$. Report the number of iterations needed to reach the stopping criterion of $\|\nabla f(\mathbf{x}_k)\| \leq 10^{-12}$. Perform the same experiments for $\sigma = 0.01$. Comment the results.

8. Explain the result theoretically. You can start by investigating what is the optimal step-size for the function f_α.

We will now compare the result of the gradient descent algorithm with fixed step-size and with the Armijo rule.

9. Implement the Armijo line search procedure. The Armijo rule is reminded in Algorithm 2 (in Section 4). Take $\beta = \theta = 1/2$. For the implementation, we suggest to return the found step-size σ and the number of calls to the function f.

10. Implement the gradient descent algorithm with Armijo rule as line search procedure.

11. Using the same settings as in Question 6, report the number of gradient calls and function calls needed to reach a gradient with norm smaller than 10^{-12}. Compare to the gradient descent with fixed step-size, conclude.

2 Gradient versus Newton direction in descent algorithms

We now consider the function

$$f_\alpha^{\text{elli}} = \frac{1}{2}\sum_{i=1}^{n} 10^{\alpha(\frac{i-1}{n-1})}\mathbf{x}_i^2 \ .$$

12. Compute the gradient and Hessian matrix of f_α^{elli}.

13. Compute the condition number of the Hessian matrix of f_α^{elli} (We remind that the condition number of a matrix corresponds to the ratio between the largest and smallest eigenvalue).

14. Implement the descent algorithm with the Newton direction as descent direction, that is
$$\mathbf{d}_k = -\text{Hess(f)}^{-1}\nabla f \ .$$
We will use the Armijo rule as line-search procedure.

15. Report for f_α^{elli}, $\alpha \in \{1, 2, 3\}$, dimension $n = 10$, initial search point $x_0 = (10, \ldots, 10)$and initial stepsize of $\sigma = 10$ the number of gradient calls and the number of function calls to reach $\|\nabla f\| \leq 10^{-12}$ for the descent algorithm with gradient and Newton as descent directions. Explain the results.

3 Optional

If you want to continue on the topic, you can for example

- Run any optimizer, already provided by pythons `scipy.optimize` library (to be imported with `from scipy import optimize`). One potential candidate is the BFGS method for which the python command is then `scipy.optimize.fmin_bfgs`.

- Investigate in addition the performance of all algorithms on the Rosenbrock function
$$f_{\text{Rosen}}(\mathbf{x}) = \sum_{i=1}^{n-1} 100(x_i - x_{i+1})^2 + (x_i - 1)^2 \ .$$

4 Algorithms

Algorithm 1 General framework for a descent algorithm to optimize $f : \mathbb{R}^n \to \mathbb{R}$. The descent direction and the LineSearch procedure depend on f.

```
Initialize $\mathbf{x}_0 \in \mathbb{R}^n$, $k = 0$
while stopping criteria not met do
    compute descent direction $\mathbf{d}_k$
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \text{LineSearch}(\mathbf{x}_k, \mathbf{d}_k)\, \mathbf{d}_k$
    $k = k + 1$
end while
```

Algorithm 2 Armijo rule

```
Input: descent direction $\mathbf{d}$, point $\mathbf{x}$, objective function $f(\mathbf{x})$ and its gradient $\nabla f(\mathbf{x})$, parameters $\sigma_0 = 10$, $\theta \in [0, 1]$ and $\beta \in (0, 1)$
Output: step-size $\sigma$

Initialize $\sigma$: $\sigma \leftarrow \sigma_0$
while $f(\mathbf{x} + \sigma\mathbf{d}) > f(\mathbf{x}) + \theta\sigma\nabla f(\mathbf{x})^T\mathbf{d}$ do
    $\sigma \leftarrow \beta\sigma$
end while
```