Introduction to Optimization - Homework 2

Due October 13th 2023, 9:00

Exercise 1 - Armijo's Rule

Let $f\colon \mathbb{R}^N \to \mathbb{R}$ be μ-strongly convex and L-smooth, and let $x_0 \in \mathbb{R}^N$.

1. Fix $\beta, \sigma \in (0,1)$ and $a \geq 2/L$, and define the step size for the n-th iteration of the gradient method using Armijo's Rule: $\alpha_n = \beta^{m_n} a$, where m_n is the smallest positive integer m for which
$$f(x_n - (\beta^m a)\nabla f(x_n)) - f(x_n) \leq -\sigma(\beta^m a)\|\nabla f(x_n)\|^2$$
Show that $1 - \frac{L\beta^{m_n - 1}a}{2} < \sigma$.

   Hint: Suppose the inequality is false, and use the Descent Lemma to arrive at a contradiction with the definition of m_n.

2. Deduce that
$$f(x_{n+1}) - \min(f) \leq \left(1 - \frac{4\mu\beta\sigma(1-\sigma)}{L}\right)(f(x_n) - \min(f))$$
for all $n \geq 1$.

   Hint: Recall that μ-strongly convex functions satisfy a PL-inequality.

3. Conclude that $f(x_n)$ converges linearly to $\min(f)$.

4. Explain why 3 implies that x_n converges linearly to the unique minimizer of f.

Exercise 2 - Computational Exercise

Introduction

In this computational exercise you will solve the least squares problem using different methods. The aim of the exercise is to compare the computed solutions, the complexity of the methods, their convergence properties.

The Linear Least Squares problem is such: for $A \in \mathbb{R}^{M\times N}$ and $b \in \mathbb{R}^M$ we need to determine the value of
$$\min_{x\in\mathbb{R}^N} \frac{1}{2}\|Ax - b\|^2. \tag{1}$$

We want to test and compare different methods discussed in class. Each method should be coded as a function that takes the following input:

1. the matrix A,
2. the vector b,
3. the initial guess x_0,
4. a tolerance for convergence ϵ,
5. a maximal number of iterations m.

Each function should return at least the following:

1. the last iterate,
2. a flag confirming convergence,
3. the number of iterations,
4. an array of past iterates.

The matrices should be generated at random. Please set the seed to be 42 to make sure the pseudo-randomly generated matrices are identical each time you run the code. Codes 1 and 2 demonstrate how to do so in Python and in Matlab.

```python
import numpy as np
np.random.seed(42)
A = np.random.random((M,N))
b = np.random.random((1,M))
```

Code 1: Code to generate pseudo-random matrices in Python

```matlab
rng(42);
A = rand(M, N);
b = rand(1, M);
```

Code 2: Code to generate pseudo-random matrices in Matlab

Methods

The methods expected to be coded include the following:

1. Gradient Descent with constant step size,
2. Gradient Descent with Armijo rule,
3. Gradient Descent with exact line search,
4. Conjugate Gradient Method,
5. Quasi-Newton Method (BFGS),
6. Polyak's Heavy Ball accelerated Gradient Descent,
7. Nesterov's momentum applied to Gradient Descent, and
8. Stochastic Gradient Descent.

Some notes and hints:

1. Choose an appropriate stopping rule. This could, for instance, be to stop when $\|\nabla f(x_k)\| < \epsilon$.
2. For the Gradient Descent with exact line search, evaluate the optimal value of α on paper first, rather than solving a subproblem at each step.
3. The Quasi-Newton BFGS method relies on the constants $0 < c_1 < c_2 < 1$ defined by the Wolfe conditions needed for the BFGS method. In the 2006 edition of source [1] page 33 (page 38 of the 1999 edition), the values $c_1 = 10^{-4}$ and $c_2 = 0.9$ are suggested. Instead of implementing line search to determine c_1 and c_2 for the Wolfe conditions, you may use the function `scypi.optimize.line_search`, since in the documentation of this method it is stated the developers used the specifications given in source [1].
4. For the acceleration methods, first try a step-size is given by $1/L$ and the acceleration factor by $\beta_k = \frac{k-1}{k+\alpha-1}$, where $\alpha > 3$ is a constant. Try different values of α. Include the best found as "Non-Optimal Heavy Ball acceleration" and "Non-Optimal Nesterov acceleration".
5. For Polyak's Havy Ball method, the optimal step-size is given by $\frac{4}{(\sqrt{L}+\sqrt{\mu})^2}$ and the optimal acceleration factor is given by $\left(\frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}\right)^2$, where L is the Lipschitz constant of the gradient and μ is the PL-constant. Recall from Homework 1 that $L = \max \sigma(A^T A)$ and $\mu = \min \sigma(A^T A)$. For Nesterov's momentum method, the optimal step-size is given by $1/L$ and the optimal acceleration factor is given by $\frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$. Include these as "Optimal Heavy Ball acceleration" and "Optimal Nesterov acceleration".

Deliverables

Compare the methods by testing them on different matrices. Test the methods with random square matrices A of size 2^k for integers $1 \leq k \leq 7$. Take the average of the number of iterations over 3 matrices for each k. (You can test for larger k too and take the average of more attempts, as long as the run time is reasonable). In a single figure, for each method, plot the number of iterations required for convergence against the number k, on a loglog scale. Make sure to add sufficient meta-data to your plot for it to be self-explanatory, namely a title, axis labels, a legend, etc. You may select $m = 10^7$ and $\epsilon = 10^{-4}$. Note that the generation of this plot could take roughly 10 minutes, depending on your implementations and on your machine.

You are expected to hand in a report written in LaTeX. This report should include the above plot, as well as a comment on it. This comment should answer the following questions:

- How do the methods compare in terms of number of iterations?
- Does the answer depend on the dimension of the space?
- Is the behavior different from what the theory predicts?
- What other findings do we have?

Add your code as an appendix to the report. This can be done, for instance, using the package "listings".