Basic Large and Huge Scale Optimization Algorithms for Solving a System of Linear Equations

Problem Description

The goal of this exercise is implementation of various large and huge scale optimization methods in order to optimize a quadratic function given by the following formula:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$$

where $\mathbf{A}$ is a positive, semi-definite matrix.
The convergence criterion is set to 10^{-5}, i.e. we check that $||\nabla f(\mathbf{x})|| \leq 10^{-5}$. Besides that, total iteration count is limited to 5000. The computations were done on Aalto's Python Jupyter Hub using a randomly generated, positive-definite 100×100 matrix.

Algorithms

For all implementations we choose a fixed α (sometimes also referred to as *learning rate*) and set it to $\frac{1}{L}$ where L is a Lipschitz constant and in our case can be calculated as the largest eigenvalue of matrix $\mathbf{A}$. This can be derived from $\mu < \nabla^2 f < L$ and the fact that Hessian ($\nabla^2 f$) is in our case equal to matrix $\mathbf{A}$.

Gradient Descent Algorithm

```
CPU times: user 935 ms, sys: 52.3 ms, total: 987 ms
Wall time: 241 ms
```

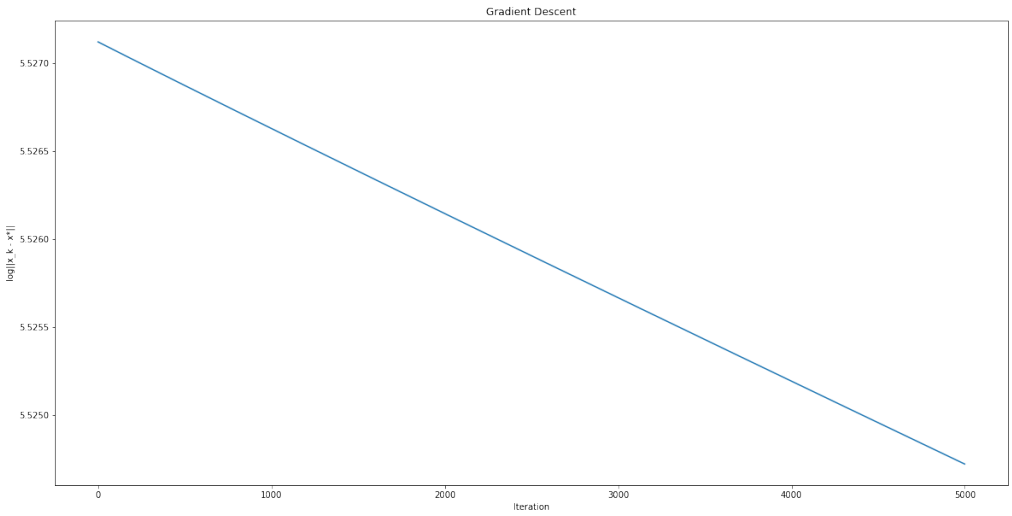

Figure 1: Log Euclidean distance from optimum through iterations using Gradient Descent Algorithm.

Conjugate Gradient Algorithm

```
CPU times: user 2.84 s, sys: 165 ms, total: 3 s
Wall time: 749 ms
```

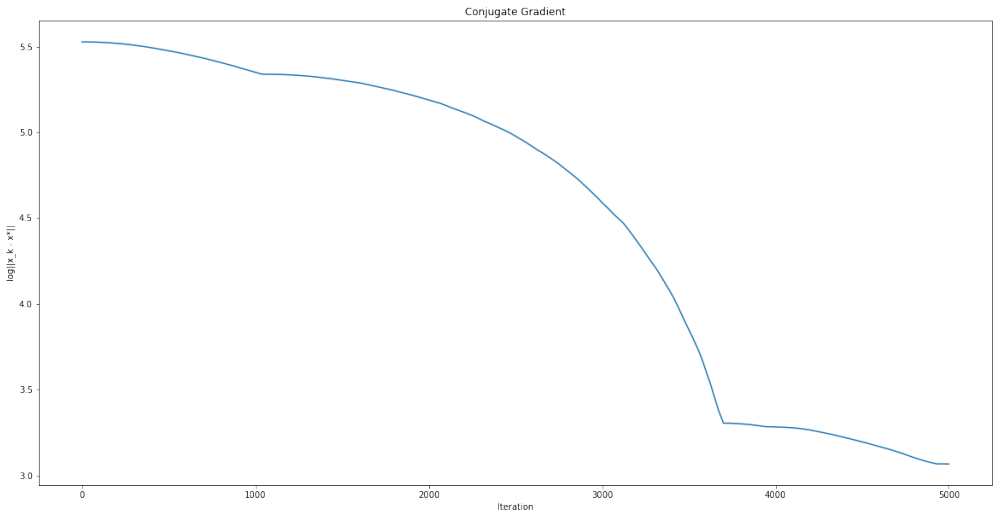

Figure 2: Log Euclidean distance from optimum through iterations using Conjugate Gradient Algorithm.

Nesterov's I Algorithm

```
CPU times: user 1.12 s, sys: 45.8 ms, total: 1.16 s
Wall time: 287 ms
```

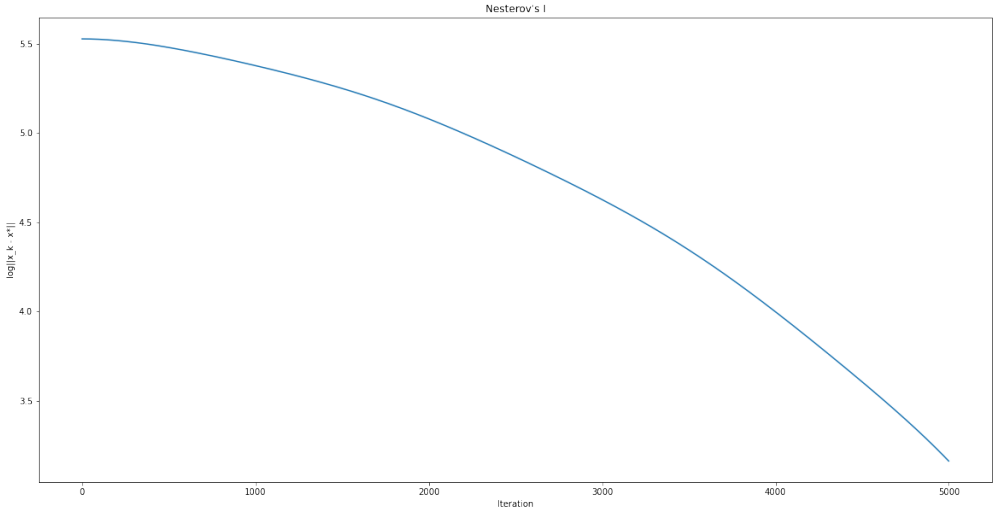

Figure 3: Log Euclidean distance from optimum through iterations using Nesterov's I Algorithm.

Coordinate Descent Algorithm

```
CPU times: user 59.9 ms, sys: 7.87 ms, total: 67.8 ms
Wall time: 66.5 ms
```

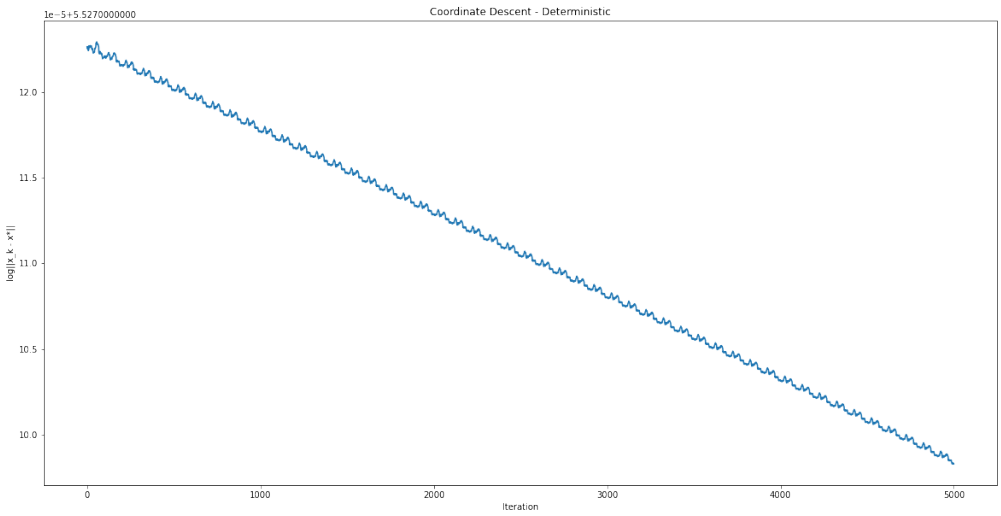

Figure 4: Log Euclidean distance from optimum through iterations using Deterministic Coordinate Descent Algorithm.

```
CPU times: user 85.4 ms, sys: 5 µs, total: 85.4 ms
Wall time: 85.1 ms
```

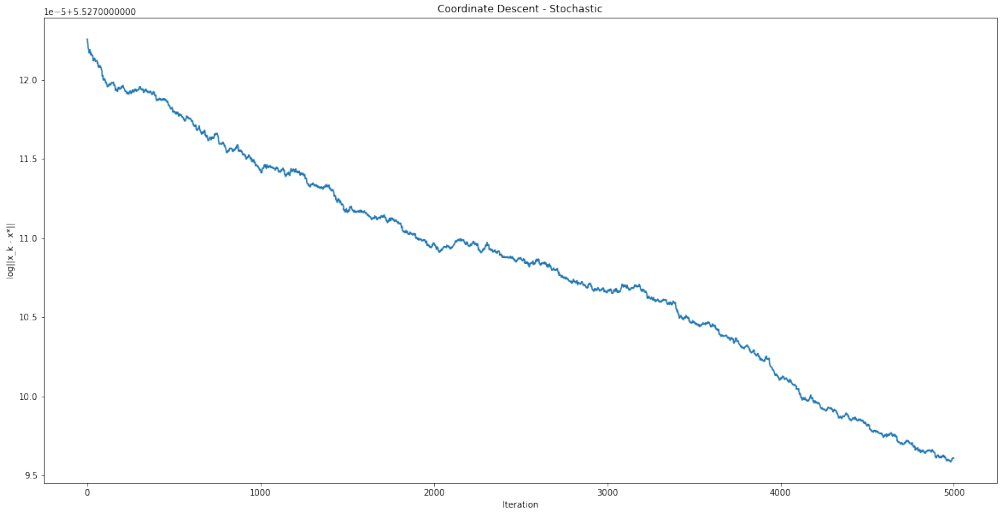

Figure 5: Log Euclidean distance from optimum through iterations using Stochastic Coordinate Descent Algorithm.

Results Comparison and Performance Analysis

The following analysis was performed on a matrix initialized by the given parameters in the code. It is important to note that by trying different initializations of matrix $\mathbf{A}$ the results and differences between approaches varied. Coordinate Descent and Gradient (Steepest) Descent had the fastest computation times, however, they weren't able to reach same levels of convergence as the other two methods in the same amount of iterations. It is also worth noting that stochastic implementation of Coordinate Descent performed slightly better than deterministic approach (at some runs it performed significantly better due to its stochasticity). Nesterov's I approach and Conjugate Gradient were both better at reaching a solution closer to optimum in set 5000 iterations with a fixed step size. Besides that, with two mentioned algorithms and same value of α I was able to reach sub threshold convergence at around 25000 iterations.
All methods have linear convergence with different convergence rates.

Besides algorithm testing, I also performed tests on scalability of the problem and noticed that when matrix dimensions surpass order of magnitude of 10^4, the computation times would significantly increase when using matrix inverse to compute optimal $\mathbf{x}$.