Large Scale Data Analysis ELEC-E5431
Name: Nguyen Xuan Binh
Student ID: 887799

Problem Setup:

The optimization problem to be addressed is a simple quadratic function minimization, that is,

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$$

where the matrix $\mathbf{A}$ and vector $\mathbf{b}$ are appropriately generated. Use the same $\mathbf{A}$ and $\mathbf{b}$ while comparing different methods.

Hints:

- $\mathbf{A}$ should be such that $\mathbf{x}^T\mathbf{A}\mathbf{x}$ is non-negative, that is, $\mathbf{A}$ is positive semi-definite, and $\mathbf{b}$ should be in the range of $\mathbf{A}$.
- In fact, by solving the above unconstrained minimization problem, you solve a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$. Indeed, the gradient of the objective function is $\mathbf{A}\mathbf{x} - \mathbf{b}$, and it should be equal to 0 at optimality. Thus, you can find optimal $\mathbf{x}^*$ using back-slash (or matrix inversion followed by computing the product $\mathbf{A}^{-1}\mathbf{b}$) operators in MATLAB. The optimal objective value can be then obtained by simply substituting such $\mathbf{x}$ into the objective of the above optimization problem. It is suitable for small and mid size problems, but the matrix inversion is prohibitively too expensive to be able to solve a system of linear equations for large scale problems. Thus, the only option for large scale problems is the use of algorithms that you implement in this assignment!

To be able to produce convergence figures for the algorithms that you test, let the dimension of $\mathbf{x}$ be 100 variables or few 100's (but after producing the figures also play with higher dimensions to see when the matrix inversion fails, but the large scale optimization methods still work fine and some also quite fast).

Set the tolerance parameter for the stopping criterion for checking the convergence to 10^{-5}. For example, check if $\|\nabla f(\mathbf{x})\| \leq 10^{-5}$, and limit the total number of iterations by 5000 if the predefined tolerance is still not achieved.

Task 1: *Gradient Descent Algorithm.*

Implement Gradient Descent Algorithm for solving the above optimization problem. Use correctly selected fixed step size α. Draw the experimental convergence rate, i.e., draw the plot $\log \|\mathbf{x}_k - \mathbf{x}^*\|_2$ versus iteration number k, for the algorithm and compare it with the theoretically predicted one.

2.4 Steepest Descent (Gradient Descent)

The iterate is given as

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \quad \alpha_k > 0.$$

How to select the step size α_k:

1. Fixed: use rules based on L and μ (trivial),
2. Backtracking (computationally easy),
3. exact line search (computationally may be hard).

For the above ways of step size selection 2 and 3, we typically have global convergence at unspecified rate.

The "greedy" strategy of getting good decrease in the current search direction may lead to better practical convergence results.

For the above way of step size selection, fixed step size selection focuses on convergence rate.

Task 2: *Conjugate Gradient Algorithm.*

Implement Conjugate Gradient Algorithm for solving the above optimization problem. Use correctly selected fixed step size. Draw the experimental convergence rate, i.e., draw the plot $\log \|\mathbf{x}_k - \mathbf{x}^*\|_2$ versus iteration number k, for the algorithm and compare it with the theoretically predicted one.

2.6 Conjugate Gradient (CG)

The iterate is given as

$$x_{k+1} = x_k + \alpha_k \rho_k, \rho_k = -\nabla f(x_k) + \delta_k \rho_{k-1}$$

The same as heavy-ball with $\beta_k = \frac{\alpha_k \delta_k}{\alpha_{k-1}}$, but in CG α_k and β_k are selected in particular way and the method does it itself.

CG can be implemented in a way that does not require knowledge (estimate) of L and μ:

- Choose α_k to minimize f along ρ_k,
- Choose δ_k by a variety of formulae (Fletcher-Reeves, Polak-Ribiere, etc.) all of these formulae are equivalent if f is convex quadratic, e.g.,

$$\delta_k = \frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_{k-1})\|_2^2}.$$

Restarting periodically with $\rho_k = -\nabla f(x_k)$ is useful, e.g., every n iterations or when ρ_k is not a descent direction.
For quadratic f: convergence analysis is based on eigenvalues of A and Chebyshev polynomials (min-max argument), linear convergence with rate $1 - \frac{2}{\sqrt{\varkappa}}$ (like heavy-ball).

Task 3: *FISTA.*

Implement FISTA for solving the above optimization problem. Use correctly selected fixed step size. Draw the experimental convergence rate, i.e., draw the plot $\log \|\mathbf{x}_k - \mathbf{x}^*\|_2$ versus iteration number k, for the algorithm and compare it with the theoretically predicted one.

2.8 FISTA (Beck & Teboulle 2009)

Simpler generic convergence analysis compared to Nesterov, adopted to composite objective function - proximal method. Otherwise the accelaration idea is the same by Nesterov.

Algorithm:

Initialize: Choose x_0; set $y_1 = x_0, t_1 = 1$.

Iterate:

$$x_k = y_k - \frac{1}{L}\nabla f(y_k)$$
$$t_{k+1} = \frac{1}{2}\left(1 + \sqrt{1 + 4t_k^2}\right)$$
$$y_{k+1} = x_k + \frac{t_k - 1}{t_{k+1}}(x_k - x_{k-1}).$$

For both strongly and weakly convex f, converges with $\frac{1}{k^2}$.

When L is not known, increase an estimate of L until it's big enough.

Task 4: *Coordinate Descent.*

Implement the Coordinate Descent method for solving the above optimization problem. Use correctly selected fixed step size. Draw the experimental convergence rate, i.e., draw the plot $\log \|\mathbf{x}_k - \mathbf{x}^*\|_2$ versus iteration number k, for the algorithm.

4.4.1 Deterministic and Stochastic CD

The update rule is

$$x_{j+1,i_j} = x_{j,i_j} - \alpha_j[\nabla f(x_j)]_{i_j}.$$

- Deterministic: choose i_j in fixed order (cyclic).
- Stochastic: choose i_j at random.

Convergence: Deterministic (Luo & Tseng 1992) – Linear rate (Beck & Tetruashvili, 2013).

Stochastic – linear rate (Nesterov, 2012).

Task 5: *Comparisons.*

Compare the results (in terms of the iterations required and the overall computation time) for different methods (including, for example, the standard MATLAB back-slash operator) and draw your overall conclusions. Observe up to which dimension Python or MATLAB still can invert a matrix, that is, define the dimension after which the problem turns to be large scale in the context of your implementation.

Because my laptop can still invert a matrix of dimension of 10000 pretty fast (20-30 seconds), and the matrix is already very heavy. If I increase the dimension even higher, my laptop will not have enough memory to store the matrix, so I am not sure what is the boundary between the large and small scale in my case. Therefore, I choose dimension 10 as small, 100 as large and 1000 as huge scale, and see how the algorithms perform for each case.

You can see all of the tasks completed below in the attached PDF file generated from the ipynb file. Additionally, you can run the file optimization.ipynb in the zipped project file. This file contains every information, from generating matrix data, algorithm implementations to convergence rate analysis.