The Chinese University of Hong Kong, Shenzhen
SDS · School of Data Science

Xiao Li · Junfeng Wu · Ming Yan · Fall Semester 2023

MAT 3007 – Optimization

Assignment 8

Due: 11:59pm, Dec. 8 (Friday), 2023

Instructions:

- Homework problems must be carefully and clearly answered to receive full credit. Complete sentences that establish a clear logical progression are highly recommended.
- Please submit your assignment on Blackboard.
- The homework must be written in English.
- Late submission will not be graded.
- Each student must not copy homework solutions from another student or from any other source.

Problem 1 (A One-Dimensional Problem): (approx. *20* points)
We consider the optimization problem

$$\min_x \ f(x) := \frac{1}{4}(x^2-1)^2 + \frac{1}{2}(x-2)^2 \qquad \text{s.t.} \qquad x \in [0,2].$$

Implement the bisection **or** the golden section method to solve this problem and output a solution with accuracy at least 10^{-5}.

Problem 2 (The Gradient Method): (approx. *30* points)
In this exercise, we want to solve the optimization problem

$$\min_{x\in\mathbb{R}^2} \ f(x) := 2x_1^4 + \frac{2}{3}x_1^3 + x_1^2 - 2x_1^2x_2 + \frac{4}{3}x_2^2. \tag{1}$$

via the gradient descent method.

Implement the gradient method that was presented in the lecture as a function `gradient_method` in `MATLAB` or `Python`.

The following input functions and parameters should be considered:

- `obj`, `grad` – function handles that calculate and return the objective function $f(x)$ and the gradient $\nabla f(x)$ at an input vector $x \in \mathbb{R}^n$. You can treat these handles as functions or fields of a class or structure `f` or you can use f and ∇f directly in your code. (For example, your function can have the form `gradient_method(obj,grad,...)`).
- x^0 – the initial point.
- `tol` – a tolerance parameter. The method should stop whenever the current iterate x^k satisfies the criterion $\|\nabla f(x^k)\| \le$ `tol`.

We want to investigate the performance of the gradient method for different step size strategies. In particular, we want to test and compare backtracking and exact line search. The following parameters will be relevant for these strategies:

- $\sigma, \gamma \in (0,1)$ – parameters for backtracking and the Armijo condition. (At iteration k, we choose α_k as the largest element in $\{1, \sigma, \sigma^2, \dots\}$ satisfying the condition $f(x^k - \alpha_k \nabla f(x^k)) - f(x^k) \leq -\gamma \alpha_k \cdot \|\nabla f(x^k)\|^2$).
- You can use the golden section method to determine the exact step size α_k. The parameters for the golden section method are: `maxit` (maximum number of iterations), $\texttt{tol}_\alpha$ (stopping tolerance for α, i.e. $\alpha_r - \alpha_l < \texttt{tol}_\alpha$), $[0, \texttt{a}]$ (the interval of the step size with starting point 0 and `a`).

You can organize the latter parameters in an appropriate `options` class or structure. It is also possible to implement separate algorithms for backtracking and exact line search. The method(s) should return the final iterate x^k that satisfies the stopping criterion.

a) Apply the gradient method with backtracking and parameters $(\sigma, \gamma) = (0.5, 0.1)$ and exact line search ($\texttt{maxit} = 100$, $\texttt{tol}_\alpha = 10^{-6}$, $\texttt{a} = 2$) to solve the problem $\min_x f(x)$.

   The algorithms should use the stopping tolerance $\texttt{tol} = 10^{-5}$. Test the methods using the initial point $x^0 = (1,1)^\top$ and report the behavior and performance of the methods. In particular, compare the number of iterations and the point to which the different gradient descent methods converged.

b) Let us define the set of initial points

$$\mathcal{X}^0 := \left\{ \begin{pmatrix} -3 \\ -3 \end{pmatrix}, \begin{pmatrix} 3 \\ -3 \end{pmatrix}, \begin{pmatrix} -3 \\ 3 \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \end{pmatrix} \right\}$$

   Run the methods:

   – Gradient descent method with backtracking and $(\sigma, \gamma) = (0.5, 0.1)$,

   – Gradient method with exact line search and $\texttt{maxit} = 100$, $\texttt{tol}_\alpha = 10^{-6}$, $\texttt{a} = 2$,

   again for every initial point in the set $\mathcal{X}^0$ using the tolerance $\texttt{tol} = 10^{-5}$. For each algorithm/step size strategy create a single figure that contains all of the solution paths generated for the different initial points. The initial points and limit points should be clearly visible. Add a contour plot of the function f in the background of each figure.

Problem 3 (A Limited-Memory Version of Adagrad): (approx. *25* points)
In this exercise, we investigate a limited-memory version of the gradient descent method with adaptive diagonal scaling. The update of the method is given by

$$x^{k+1} = x^k - \alpha_k D_k^{-1} \nabla f(x^k), \tag{2}$$

where $\alpha_k \geq 0$ is a suitable step size and $D_k \in \mathbb{R}^{n \times n}$ is a diagonal matrix that is chosen as follows: $D_k = \mathrm{diag}(v_1^k, v_2^k, ..., v_n^k)$ and

$$v_i^k = \sqrt{\epsilon + \sum_{j=t_m(k)}^{k} (\nabla f(x^j))_i^2}, \quad t_m(k) = \max\{0, k-m\}, \quad \forall\ i = 1, ..., n.$$

Here, $\epsilon > 0$, the memory constant $m \in \mathbb{N}$, and the initial point $x^0 \in \mathbb{R}^n$ are given parameters.

a) Show that the direction $d^k = -D_k^{-1}\nabla f(x^k)$ is a descent direction for all $k \in \mathbb{N}$ (assuming $\nabla f(x^k) \neq 0$).

b) Write a `MATLAB` or `Python` code and implement the gradient method with adaptive diagonal scaling (Adagrad) and backtracking (i.e., implement the abstract descent method with d^k as descent direction). Run Adagrad on problem (1) introduced in **Problem 2** and compare its performance with the tested approaches in **Problem 2** using the same initial points and stopping tolerance as in **Problem 2** b). You can use the following parameters:

- $(\sigma, \gamma) = (0.5, 0.1)$ – parameter for backtracking.
- $\epsilon = 10^{-6}$ – scaling parameter in the update (2).
- $m = 25$ – memory parameter.

Generate a plot of the different solution paths of Adagrad using the different initial points from $\mathcal{X}^0$ and add a contour plot of f in the background.

c) How does Adagrad behave when different memories m are chosen? (Suitable other choices might include $m \in \{5, 10, 15, 25\}$)?

Problem 4 (Globalized Newton's Method): (approx. *25* points)

Implement the globalized Newton method with backtracking that was presented in the lecture as a function `newton_glob` in `MATLAB` or `Python`.

The pseudo-code for the full Newton method is given as follows:

Algorithm 1: The Globalized Newton Method

```
1 Initialization:  Select an initial point $x^0 \in \mathbb{R}^n$ and parameter $\gamma, \gamma_1, \gamma_2, \sigma \in (0,1)$ and tol.
  for $k = 0, 1, ...$ do
2     If $\|\nabla f(x^k)\| \leq$ tol, then STOP and $x^k$ is the output.
3     Compute the Newton direction $s^k$ as solution of the linear system of equations:
          $\nabla^2 f(x^k) s^k = -\nabla f(x^k)$.
4     If $-\nabla f(x^k)^\top s^k \geq \gamma_1 \min\{1, \|s^k\|^{\gamma_2}\}\|s^k\|^2$, then accept the Newton direction and set
        $d^k = s^k$. Otherwise set $d^k = -\nabla f(x^k)$.
5     Choose a step size $\alpha_k$ by backtracking and calculate $x^{k+1} = x^k + \alpha_k d^k$.
```

The following input functions and parameters should be considered:

- `obj`, `grad`, `hess` – function handles that calculate and return the objective function $f(x)$, the gradient $\nabla f(x)$, and the Hessian $\nabla^2 f(x)$ at an input vector $x \in \mathbb{R}^n$. You can treat these handles as functions or fields of a class or structure `f` or you can use f, ∇f, and $\nabla^2 f$ from part a) and b) directly in the algorithm. (For example, your function can have the form `newton_glob(obj,grad,hess,...)`).
- x^0 – the initial point.
- `tol` – a tolerance parameter. The method should stop whenever the current iterate x^k satisfies the criterion $\|\nabla f(x^k)\| \leq$ `tol`.
- $\gamma_1, \gamma_2 > 0$ – parameters for the Newton condition.
- $\sigma, \gamma \in (0, 1)$ – parameters for backtracking and the Armijo condition.

You can again organize the latter parameters in an appropriate `options` class or structure. You can use the backslash operator `A\b` in `MATLAB` or `numpy.linalg.solve(A,b)` to solve the linear system of equations $Ax = b$. If the computed Newton step $s^k = -\nabla^2 f(x^k)^{-1}\nabla f(x^k)$ is a descent direction and satisfies

$$-\nabla f(x^k)^\top s^k \geq \gamma_1 \min\{1, \|s^k\|^{\gamma_2}\}\|s^k\|^2,$$

we accept it as next direction d^k. Otherwise, the gradient direction $d^k = -\nabla f(x^k)$ is chosen. The method should return the final iterate x^k that satisfies the stopping criterion.

a) Test your approach on the Rosenbrock function $f : \mathbb{R}^2 \to \mathbb{R}$

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

with initial point $x^0 = (-1, -0.5)^\top$ and parameter $(\sigma, \gamma) = (0.5, 10^{-4})$ and $(\gamma_1, \gamma_2) = (10^{-6}, 0.1)$. (Notice that γ is smaller here). Besides the globalized Newton method also run the gradient method with backtracking ($(\sigma, \gamma) = (0.5, 10^{-4})$) on this problem and compare the performance of the two approaches using the tolerance `tol` $= 10^{-7}$.

Does the Newton method always utilize the Newton direction? Does the method always use full step sizes $\alpha_k = 1$?

b) Repeat the performance test described in **Problem 2** b) and **Problem 3** b) for problem (1) using the globalized Newton method. You can use the parameter $(\sigma, \gamma) = (0.5, 0.1)$, $(\gamma_1, \gamma_2) = (10^{-6}, 0.1)$, and `tol` $= 10^{-5}$.

Plot all of the solution paths obtained by Newton's method for the different initial points in $\mathcal{X}^0$ in one figure (with a contour plot of f in the background).