DDA5002_HW5

by Xiaocao_225040374

Problem 1 - Solution

(a) Proof of f is not Lipschitz smooth.

Lipschitz smooth(1D case):

$$\exists L > 0, s.t. \|\nabla f(x) - \nabla f(y)\| = \|f'(x) - f'(y)\| \le L\|x - y\|$$

Derivative of f:

$$f'(x) = \begin{cases} \frac{3}{2}\sqrt{x}, x > 0 \\ -\frac{3}{2}\sqrt{-x}, x < 0 \\ 0, x = 0 \end{cases}$$

Suppose that f is Lipschitz smooth and pick $y = 0$ and $x > 0$, then:

$$|f'(x) - f'(0)| \le L|x - 0|$$

$$|\frac{3}{2}\sqrt{x} - 0| \le L|x|$$

$$\frac{3}{2\sqrt{x}} \le L$$

As $x \to 0^+$, we have $L \to +\infty$, so f is **not Lipschitz smooth**.

(b) Proof of $\forall \bar{\alpha} > 0, \exists x_0, \text{s.t.} f(x_1) \ge f(x_0)$

Update rule:

$$x_1 = x_0 - \bar{\alpha}\nabla f(x_0) = x_0 - \bar{\alpha} f'(x_0)$$

Then the condition becomes:

$$f(x_1) \ge f(x_0)$$

$$|x_1|^{3/2} \ge |x_0|^{3/2}$$

$$|x_1| = |x_0 - \bar{\alpha} f'(x_0)| = |x_0 - \frac{3}{2}\bar{\alpha}\sqrt{|x_0|}| \ge |x_0|$$

Let $\sqrt{|x_0|} = z$, then:

$$|z^2 - \frac{3\bar{\alpha}}{2}z| \ge z^2 \Rightarrow |z - \frac{3\bar{\alpha}}{2}| \ge z$$

Pick $z = \frac{3\bar{\alpha}}{4}$ and $x_0 = \pm(\frac{3}{4}\bar{\alpha})^2$ will suffice.

Problem 2 - Solution

(a)

Since matrix A is symmetric, the gradient of f is $\nabla f(x) = 2Ax$.

$$\begin{aligned}\nabla f(x_0) &= 2Ax_0\\ &= 2\begin{bmatrix}2 & 0\\ 0 & 5\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}4\\10\end{bmatrix}.\end{aligned}$$

(i) constant step size

$$\begin{aligned}x_1 &= x_0 - \alpha\nabla f(x_0)\\ &= \begin{bmatrix}1\\1\end{bmatrix} - 0.1\begin{bmatrix}4\\10\end{bmatrix} = \begin{bmatrix}0.6\\0\end{bmatrix}.\end{aligned}$$

The function value at the new iterate is

$$f(x_1) = 2(0.6)^2 + 5(0)^2 = 0.72.$$

(ii) exact line search

Denote $g = \nabla f(x_0)$. Consider $\phi(\alpha) = f(x_0 - \alpha g)$.

$$\begin{aligned}\phi(\alpha) &= (x_0 - \alpha g)^\top A(x_0 - \alpha g)\\ &= x_0^\top Ax_0 - \alpha g^\top g + \alpha^2 g^\top Ag.\end{aligned}$$

$$\phi'(\alpha) = -g^\top g + 2\alpha g^\top Ag = 0,$$

Solve and get the optimal step size

$$\alpha^\star = \frac{g^\top g}{2g^\top Ag} = \frac{29}{266}.$$

Then calculate the update for x_1:

$$\begin{aligned}x_1 &= x_0 - \alpha^\star g\\ &= \begin{bmatrix}1\\1\end{bmatrix} - \frac{29}{266}\begin{bmatrix}4\\10\end{bmatrix} = \begin{bmatrix}\frac{75}{133}\\ -\frac{12}{133}\end{bmatrix}.\end{aligned}$$

$$\begin{aligned}f(x_1) &= 2\left(\frac{75}{133}\right)^2 + 5\left(\frac{12}{133}\right)^2\\ &\approx 0.6769.\end{aligned}$$

(iii) backtracking line search

Substituting the parameter values at x_0,

$$f(x_0 - \alpha g) \le 7 - 58\alpha.$$

- Starting from $\alpha = 1$: the Armijo condition is not satisfied. Then continue to reduce α until $\alpha = 0.04$, where the Armijo condition is satisfied.

$$\begin{aligned}x_1 &= x_0 - 0.04\nabla f(x_0)\\ &= \begin{bmatrix}0.84\\0.6\end{bmatrix},\end{aligned}$$

And the result is $f(x_1) = 3.2112$.

(b) Implement the strategies in Python

In [4]:
```python
# Constant step size
import numpy as np

A = np.array([[2.0, 0.0],
              [0.0, 5.0]])
x0 = np.array([1.0, 1.0])
```

```python
def f(x: np.ndarray) -> float:
    return float(x.T @ A @ x)

def grad(x: np.ndarray) -> np.ndarray:
    return 2.0 * (A @ x)

def gd_constant(x0: np.ndarray, alpha: float, iters: int = 10):
    x = x0.copy().astype(float)
    hist = []
    for k in range(iters):
        g = grad(x)
        x_new = x - alpha * g
        hist.append((k, alpha, x.copy(), f(x)))
        x = x_new
    hist.append((iters, alpha, x.copy(), f(x)))
    return hist

h1 = gd_constant(x0, alpha=0.1, iters=10)
def print_hist(title: str, hist):
    print("\n" + title)
    print("- | alpha       | x^T                | f(x)")
    print("--+-------------+-------------------+----------------")
    for k, alpha, x, fx in hist:
        if np.isnan(alpha):
            a_str = "    -"
        else:
            a_str = f"{alpha: .6f}"
        print(f"{k:2d}|{a_str:>11} | [{x[0]: .6f}, {x[1]: .6f}] | {fx: .10f}")

print_hist("Constant step size", h1)
```

```
Constant step size
- | alpha       | x^T                | f(x)
--+-------------+-------------------+----------------
 0|   0.100000 | [ 1.000000,  1.000000] |  7.0000000000
 1|   0.100000 | [ 0.600000,  0.000000] |  0.7200000000
 2|   0.100000 | [ 0.360000,  0.000000] |  0.2592000000
 3|   0.100000 | [ 0.216000,  0.000000] |  0.0933120000
 4|   0.100000 | [ 0.129600,  0.000000] |  0.0335923200
 5|   0.100000 | [ 0.077760,  0.000000] |  0.0120932352
 6|   0.100000 | [ 0.046656,  0.000000] |  0.0043535647
 7|   0.100000 | [ 0.027994,  0.000000] |  0.0015672833
 8|   0.100000 | [ 0.016796,  0.000000] |  0.0005642220
 9|   0.100000 | [ 0.010078,  0.000000] |  0.0002031199
10|   0.100000 | [ 0.006047,  0.000000] |  0.0000731232
```

In [5]:
```python
# Exact line search
def gd_exact_line_search(x0: np.ndarray, iters: int = 10):
    x = x0.copy().astype(float)
    hist = []
    for k in range(iters):
        g = grad(x)
        gg = float(g.T @ g)
        gAg = float(g.T @ (A @ g))
        alpha = gg / (2.0 * gAg)

        hist.append((k, alpha, x.copy(), f(x)))
        x = x - alpha * g
    hist.append((iters, np.nan, x.copy(), f(x)))
    return hist
h2 = gd_exact_line_search(x0, iters=10)
print_hist("Exact line search", h2)
```

```
Exact line search
- | alpha        | x^T                | f(x)
--+--------------+-------------------+----------------
 0|   0.109023 | [ 1.000000,  1.000000] |  7.0000000000
 1|   0.207143 | [ 0.563910, -0.090226] |  0.6766917293
 2|   0.109023 | [ 0.096670,  0.096670] |  0.0654159566
 3|   0.207143 | [ 0.054513, -0.008722] |  0.0063237767
 4|   0.109023 | [ 0.009345,  0.009345] |  0.0006113211
 5|   0.207143 | [ 0.005270, -0.000843] |  0.0000590966
 6|   0.109023 | [ 0.000903,  0.000903] |  0.0000057129
 7|   0.207143 | [ 0.000509, -0.000082] |  0.0000005523
 8|   0.109023 | [ 0.000087,  0.000087] |  0.0000000534
 9|   0.207143 | [ 0.000049, -0.000008] |  0.0000000052
10|          - | [ 0.000008,  0.000008] |  0.0000000005
```

In [6]:
```python
# Backtracking line search
def armijo_backtracking_alpha(
    x: np.ndarray, gamma: float = 0.5,
    sigma: float = 0.2, alpha0: float = 1.0
):
    g = grad(x)
    fx = f(x)
    gg = float(g.T @ g)
    alpha = alpha0
    while f(x - alpha * g) > fx - gamma * alpha * gg:
        alpha *= sigma
    return alpha

def gd_backtracking(
    x0: np.ndarray, iters: int = 10,
    gamma: float = 0.5, sigma: float = 0.2, alpha0: float = 1.0
):
    x = x0.copy().astype(float)
    hist = []
    for k in range(iters):
        alpha = armijo_backtracking_alpha(x, gamma=gamma, sigma=sigma, alpha0=alpha0)
        hist.append((k, alpha, x.copy(), f(x)))
        x = x - alpha * grad(x)
    hist.append((iters, np.nan, x.copy(), f(x)))
    return hist

h3 = gd_backtracking(x0, iters=10, gamma=0.5, sigma=0.2, alpha0=1.0)
print_hist("Backtracking line search", h3)
```

```
Backtracking line search
- | alpha        | x^T                | f(x)
--+--------------+-------------------+----------------
 0|   0.040000 | [ 1.000000,  1.000000] |  7.0000000000
 1|   0.040000 | [ 0.840000,  0.600000] |  3.2112000000
 2|   0.040000 | [ 0.705600,  0.360000] |  1.6437427200
 3|   0.040000 | [ 0.592704,  0.216000] |  0.9358760632
 4|   0.040000 | [ 0.497871,  0.129600] |  0.5797325822
 5|   0.040000 | [ 0.418212,  0.077760] |  0.3800355455
 6|   0.200000 | [ 0.351298,  0.046656] |  0.2577045257
 7|   0.040000 | [ 0.070260, -0.046656] |  0.0207567362
 8|   0.040000 | [ 0.059018, -0.027994] |  0.0108844732
 9|   0.040000 | [ 0.049575, -0.016796] |  0.0063259515
10|          - | [ 0.041643, -0.010078] |  0.0039761036
```

Problem 3 - Solution

(a) The global minimizer is (0, 0), the derivations are as follows:

The global minimizer must satisfy:

$$\nabla f(x,y) = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} x^{\frac{1}{3}} + (x+y) \\ x+y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

so:

$$\begin{cases} x^{\frac{1}{3}} + (x+y) = 0 \\ x+y=0 \end{cases}$$

and the **unique** solution is $x = 0, y = 0$. Besides, $\forall x > 0, y > 0, f(x,y) > f(0,0) = 0$, which shows that $(0,0)$ is **the unique global minimizer**.

(b) Newton's Method:

(i) Finding $\mathbf{x}_{k+1}$

The expression for (x_{k+1}, y_{k+1}) is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)$$

where $\mathbf{x}_k = (x_k, y_k)^T$.

Now we will compute the components one by one:

1. The inverse of Hessian matrix:

$$[\nabla^2 f(\mathbf{x}_k)]^{-1} = \begin{pmatrix} \frac{1}{3}{x_k}^{-\frac{2}{3}} + 1 & 1 \\ 1 & 1 \end{pmatrix}^{-1} = (3{x_k}^{\frac{2}{3}}) \begin{pmatrix} 1 & -1 \\ -1 & \frac{1}{3}{x_k}^{-\frac{2}{3}} + 1 \end{pmatrix}$$

2. The first order derivative:

$$\nabla f(\mathbf{x}_k) = \begin{pmatrix} {x_k}^{\frac{1}{3}} + x_k + y_k \\ x_k + y_k \end{pmatrix}$$

Put together, we have:

$$[\nabla^2 f]^{-1} \nabla f = \left(3x_k^{\frac{2}{3}}\right) \begin{pmatrix} 1 & -1 \\ -1 & \frac{1}{3}{x_k}^{-\frac{2}{3}} + 1 \end{pmatrix} \begin{pmatrix} x_k^{\frac{1}{3}} + x_k + y_k \\ x_k + y_k \end{pmatrix} = \begin{pmatrix} 3x_k \\ -2x_k + y_k \end{pmatrix}$$

So finally we get the expression of $\mathbf{x}_{k+1}$:

$$\mathbf{x}_{k+1} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} - \begin{pmatrix} 3x_k \\ -2x_k + y_k \end{pmatrix} = \begin{pmatrix} -2x_k \\ 2x_k \end{pmatrix}$$

(ii) Proof of Non-Convergence

Start from (x_0, y_0) with $x_0 \neq 0$, the sequence of iteration is:

$$\begin{cases} x_k = (-2)^k x_0 \neq 0 \\ y_k = 2x_{k-1} = 2(-2)^{k-1} x_0 \neq 0 \end{cases}$$

For all k, we have $\mathbf{x}_k \neq \mathbf{x}^* = \mathbf{0}$, so Newton Iterations starting from a non-zero initial point never converges to the global minimizer.

(c) Expression of projected gradient descent:

One iteration of the projected gradient descent method:

1. Compute an intermediate point: $\mathbf{z}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$.
2. Project onto constraint: $\mathbf{x}_{k+1} = \mathrm{Proj}_\Omega(\mathbf{z}_{k+1})$.

For $\mathbf{x}_k = (x_k, y_k)$ on $x_k + y_k = 1$:

$$\nabla f(x_k, y_k) = \begin{pmatrix} x_k^{\frac{1}{3}} + 1 \\ 1 \end{pmatrix}$$

$$z_{k+1,1} = x_k - \alpha_k (x_k^{\frac{1}{3}} + 1)$$

$$z_{k+1,2} = y_k - \alpha_k$$

The projection of a point (z_1, z_2) onto the line $x + y = c$ is $((z_1 - z_2 + c)/2, (z_2 - z_1 + c)/2)$. Here $c = 1$.

$$x_{k+1} = \frac{(x_k - \alpha_k(x_k^{\frac{1}{3}} + 1)) - (y_k - \alpha_k) + 1}{2}$$

Since $y_k = 1 - x_k$:

$$x_{k+1} = \frac{x_k - (1 - x_k) - \alpha_k x_k^{\frac{1}{3}} + 1}{2} = x_k - \frac{\alpha_k}{2} x_k^{\frac{1}{3}}$$

And $y_{k+1} = 1 - x_{k+1}$. The iteration is:

$$x_{k+1} = x_k - \frac{\alpha_k}{2} x_k^{\frac{1}{3}}$$

$$y_{k+1} = 1 - x_{k+1}$$

Problem 4 - Solution

(a) Show that the projection onto L can be expressed as $P_L(x) = p + \frac{v^T(x-p)}{\|v\|^2} v$.

The projection $P_L(x)$ minimizes $\|y - x\|^2$ for $y \in L$.

Let $y(t) = p + tv$. We minimize $h(t) = \|(p - x) + tv\|^2 = \|p - x\|^2 + 2t(p - x)^T v + t^2 \|v\|^2$.

Setting $\frac{dh}{dt} = 2(p - x)^T v + 2t\|v\|^2 = 0$, we find $t^* = \frac{(x-p)^T v}{\|v\|^2} = \frac{v^T(x-p)}{\|v\|^2}$.

Thus, $P_L(x) = p + t^* v = p + \frac{v^T(x-p)}{\|v\|^2} v$.

(b) Explain that if the projection of x onto L lies in B (i.e., $P_L(x) \in B$), then $P_C(x) = P_L(x)$.

Let $y_L = P_L(x)$. By definition, y_L is the unique point on line L closest to x.

If $P_L(x) \in B$, then $y_L \in B$. Since $y_L \in L$, it follows that $y_L \in L \cap B$, so $y_L \in C$.

For any $y \in C$, we know $y \in L$. By the definition of $P_L(x)$, $\|y_L - x\|^2 \le \|y - x\|^2$ for all $y \in L$.

Since $C \subseteq L$, this inequality also holds for all $y \in C$.

As $y_L \in C$ and it is the closest point to x among all points in C, y_L must be $P_C(x)$.

Thus, if $P_L(x) \in B$, then $P_C(x) = P_L(x)$.