Assignment 3
Large-scale Optimization FTN0452

Li Ju

November 22, 2023

1 Problem 7.1

Proof. Recall the definition of $P_\Omega(x)$, we have

$$P_\Omega(x) = z^\star = \arg\min_{z\in\Omega} \frac{1}{2}\|z - x\|^2$$

Since $f(z) = \frac{1}{2}\|z - x\|^2$ is strongly convex, we know that the projection exists and is unique, with the optimality condition $(x - z^\star)^\top (z - z^\star) \leq 0, \forall z \in \Omega$.

If $\|x\| \leq 1$, we have $z^\star = x$. Then we have

$$(x - z^\star)^\top (z - z^\star) = \underbrace{(x - x)^\top}_{0}(z - x) \leq 0$$

The optimality condition holds for $z^\star = x$.

If $\|x\| > 1$, we have $z^\star = \frac{x}{\|x\|}$. Then we have

$$\begin{aligned}
(x - z^\star)^\top (z - z^\star) &= \left(x - \frac{x}{\|x\|}\right)^\top \left(z - \frac{x}{\|x\|}\right) \\
&= \left(1 - \frac{1}{\|x\|}\right) x^\top \left(z - \frac{x}{\|x\|}\right) \\
&= \left(1 - \frac{1}{\|x\|}\right) \left(x^\top z - \frac{x^\top x}{\|x\|}\right) \\
&= \underbrace{\left(1 - \frac{1}{\|x\|}\right)}_{>0} \left(x^\top z - \|x\|\right) \\
&\leq \left(1 - \frac{1}{\|x\|}\right) \left(\|x\|\|z\| - \|x\|\right) \\
&= \underbrace{(\|x\| - 1)}_{>0} \underbrace{(\|z\| - 1)}_{\leq 0} \\
&\leq 0
\end{aligned}$$

Since we have $z \in \Omega$ (i.e. $\|z\| - 1 \leq 0$) and $\|x\| - 1 > 0$, the optimality condition holds. □

2 Problem 7.4

2.1 i

The optimality condition for the constrained optimization problem is $-\nabla_x f(x^\star) \in N_\Omega(x^\star)$, where $f(x) = c^\top x$ is the objective function and $N_\Omega(x)$ denotes the normal cone of x on Ω. By the definition of the normal cone

and the fact that $\nabla f(x) = c$, we have the general *minimum principle* that

$$-c^\top (z - x^\star) \leq 0, \forall z \in \Omega \tag{1}$$

Case a $\Omega = \{x \mid \|x\|_2 \leq 1\}$:

The solution $x^\star$ is characterised by $-c^\top x^\star = \|c\|\|x^\star\|$.

Case b $\Omega = \{x \mid x \geq 0, \sum_{i=1}^n x_i = 1\}$:

$$x^\star \in \{x \mid x > 0, \sum_{i \in N_{\min}} x_i = 1\}$$

where $N_{\min} = \underset{i \in \{1,2,\ldots,n\}}{\arg\min}\ c_i$.

Case c $\Omega = \{x \mid 0 \leq x_i \leq 1, i = 1, 2, \ldots, n\}$:

$$x^\star \in \{x \mid x_i = 0, x_j = 1, 0 \leq x_k \leq 1, \forall i \in N_{\text{pos}}, j \in N_{\text{neg}}, k \in N_{\text{zero}}\}$$

where $N_{\text{pos}} = \{i \mid c_i > 0, i \in \{1, 2, \ldots, n\}\}$, $N_{\text{neg}} = \{i \mid c_i < 0, i \in \{1, 2, \ldots, n\}\}$ and $N_{\text{zero}} = \{i \mid c_i = 0, i \in \{1, 2, \ldots, n\}\}$.

2.2 ii & iii

The implementation of projected gradient method is deferred to Appendix A.1. Figure 1 visualizes update paths of projected gradient descent for different c with a constant initial weight. It is observed that the experimental results are consistent with theoretical characterization of optima.

Additionally, since the objective function $f(x) = c^\top x$ is not strongly convex, the global optima is not unique. The solution given by projected gradient method relies on the start point x_0. With different x_0, different optima may be found.

3 Problem 8.5

Given $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ The subgradients $\partial f(\boldsymbol{x})$ and directional derivatives $f'(\boldsymbol{x}; \boldsymbol{v})$ w.r.t. $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$ for norms are given as follows:

a). The ℓ_1-norm $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$.

The subgradient is $\partial\|\boldsymbol{x}\|_1 = \{\boldsymbol{g} \in \mathbb{R}^{\boldsymbol{n}} : (g_1, g_2, \ldots, g_n)\}$ where

$$g_i \in \begin{cases} \{\text{sgn}(x_i)\} & x_i \neq 0 \\ [-1, 1] & x_i = 0 \end{cases} \quad \forall i \in \{1, 2, \ldots, n\}$$

The directional derivative is $f'(\boldsymbol{x}; \boldsymbol{v}) = \sum_{i=1}^n u_i$ where

$$u_i = \begin{cases} \text{sgn}(x_i) v_i & x_i \neq 0 \\ |v_i| & x_i = 0 \end{cases} \quad \forall i \in \{1, 2, \ldots, n\}$$

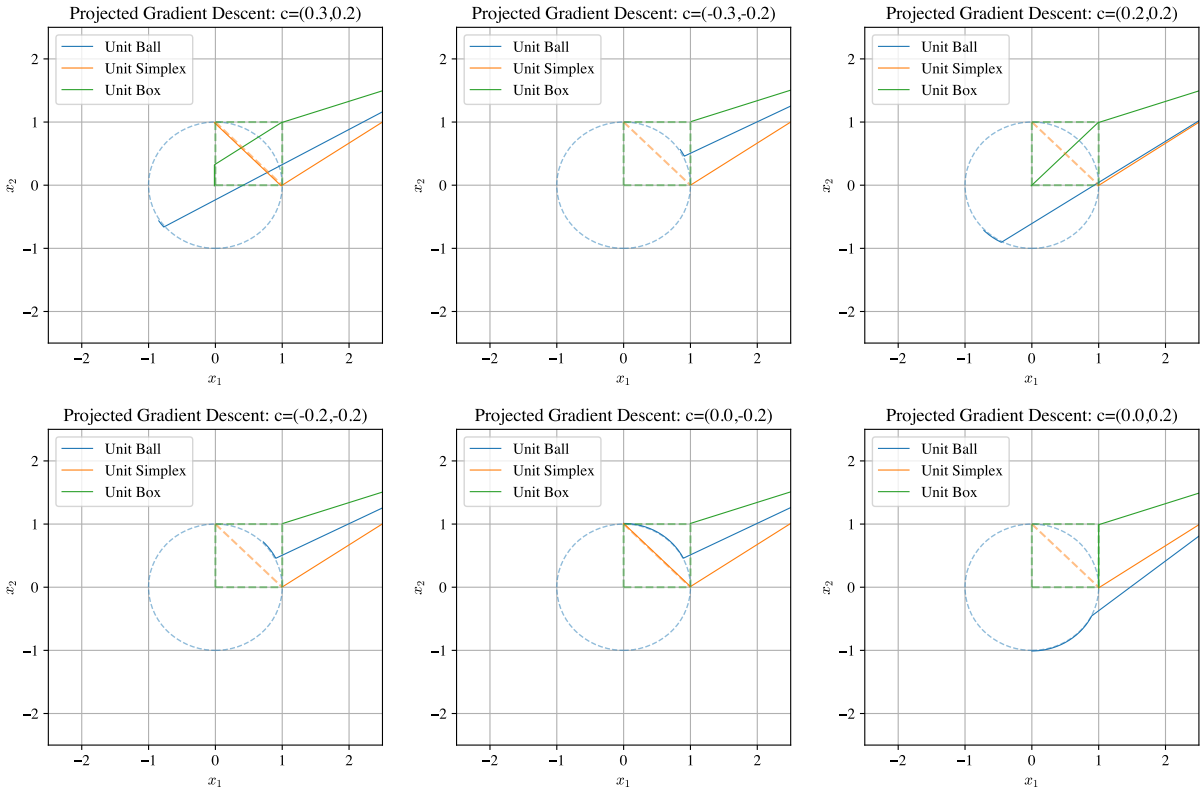

Figure 1: Update paths of projected gradient descent for different c with a constant initial weight.

b). The ℓ_2-norm $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2$.

The subgradient is

$$\partial f(\boldsymbol{x}) = \begin{cases} \{\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}\} & \boldsymbol{x} \neq \mathbf{0} \\ \{\mathbf{0}\} & \boldsymbol{x} = \mathbf{0} \end{cases}$$

The directional derivative is

$$f'(\boldsymbol{x}; \boldsymbol{v}) = \begin{cases} \frac{\boldsymbol{x}^\top \boldsymbol{v}}{\|\boldsymbol{x}\|_2} & \boldsymbol{x} \neq \mathbf{0} \\ 0 & \boldsymbol{x} = \mathbf{0} \end{cases}$$

c). The ℓ_∞-norm $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_\infty$.

Let $x_{\max} \coloneqq \max\{|x_1|, |x_1|, \dots |x_n|\}$ and $f_i(\boldsymbol{x}) = |x_i|$. We have $f(\boldsymbol{x}) = x_{\max} = \max\{f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \dots f_n(\boldsymbol{x})\}$

The subgradient is $\partial f(\boldsymbol{x}) = \mathbf{Conv}(\bigcup\{\partial f_i(\boldsymbol{x}) : |x_i| = x_{\max}\}) \quad \forall i \in 1, 2, \dots n$, where

$$\partial f_i(\boldsymbol{x}) = \begin{cases} \{\text{sgn}(x_i)\boldsymbol{e}_i\} & x_i \neq 0 \\ \mathbf{Conv}(\{-\boldsymbol{e}_i, \boldsymbol{e}_i\}) & x_i = 0 \end{cases} \quad \forall i \in \{1, 2, \dots, n\}$$

Here $\mathbf{Conv}(*)$ denotes the convex hull, $\boldsymbol{e}_i$ denotes the i-th standard basis of $\mathbb{R}^n$.

If $m_{\max} = 0$, indicating that $\boldsymbol{x} = \mathbf{0}$, we have

$$\partial f(\mathbf{0}) = \mathbf{Conv}(\bigcup\{\mathbf{Conv}(\boldsymbol{e}_i), \forall i \in \{1, 2, \dots n\}\}) = \{\boldsymbol{g} \in \mathbb{R}^n : \sum_{i=1}^{n} |g_i| = 1\}$$

If $m_{\max} \neq 0$, let $P = \{i : x_i = x_{\max}, \forall i \in \{1, 2, \dots n\}\}$, $N = \{i : x_i = -x_{\max}, \forall i \in \{1, 2, \dots n\}\}$, and $C = \{i : |x_i| \neq x_{\max}, \forall i \in \{1, 2, \dots n\}\}$. We have

$$\begin{aligned} \partial f(\boldsymbol{x}) &= \mathbf{Conv}(\bigcup\{\{\boldsymbol{e}_i : i \in P\}, \{-\boldsymbol{e}_j : j \in N\}\}) \\ &= \{\boldsymbol{g} \in \mathbb{R}^n : \sum |g_i| = 1, \text{sgn}(g_i) = \text{sgn}(x_i), g_j = 0 \quad \forall j \in C, \forall i \in \bigcup\{N, P\}\} \end{aligned}$$

Then the directional derivative is

$$f'(\boldsymbol{x}; \boldsymbol{v}) = \begin{cases} \max(\{\text{sgn}(x_i)v_i : i \in \bigcup\{N, P\}\}) & \boldsymbol{x} \neq \mathbf{0} \\ \max(\{|v_i| : i \in \{1, 2, \dots, n\}\}) & \boldsymbol{x} = \mathbf{0} \end{cases}$$

4 Problem: ℓ_1-regularized linear regression

4.1 i

Each individual parts of the objective function is characteriazed as follows:

Regression term Let $T_1(x) = \frac{1}{2}\sum_{i=1}^{N}(a_i^\top x - b_i)^2$, $A = (a_1, a_2, \dots a_N)^\top \in \mathbb{R}^{N \times n}$ and $b = (b_1, b_2, \dots, b_N)^\top \in \mathbb{R}^n$. We have

$$\begin{aligned} T_1 &= \frac{1}{2}(Ax - b)^\top (Ax - b) \\ &= \frac{1}{2}x^\top A^\top Ax + \mathcal{O}(x) \end{aligned}$$

The Hessian of $T_1(x)$ is given by $A^\top A$. The positive-definiteness of $A^\top A$ implies that $T_1(x)$ is convex, and is Lipschitz smooth with $L = \lambda_{\max}(A^\top A)$, where $\lambda_{\max}(*)$ denotes the largest eigenvalue.

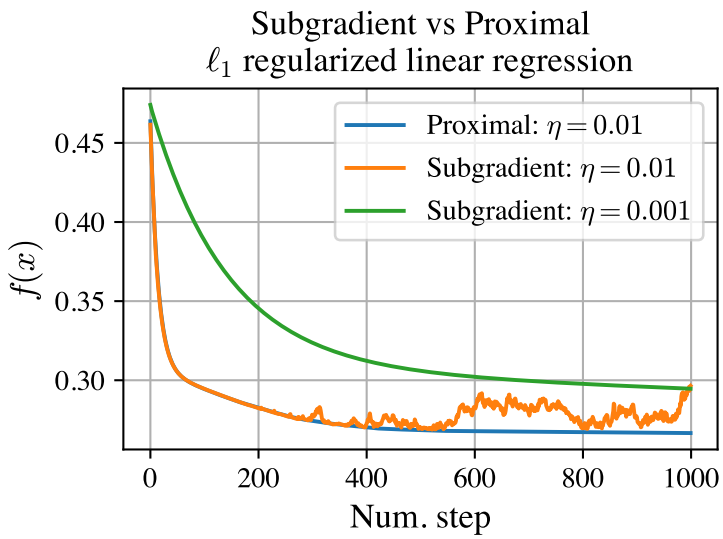

Figure 2: Optimization dynamics of subgradient descent and proximal gradient descent for ℓ_1 regularized linear regression.

Regularization term Let $T_2(x) = \lambda \|x\|_1$. For any $a, b \in \mathbb{R}^n$ and $0 \leq \alpha \leq 1$ we have

$$(1-\alpha)|a_i| + \alpha|b_i| = |(1-\alpha)a_i| + \alpha|b_i| \geq |(1-\alpha)a_i + \alpha b_i|, \forall i \in \{1, 2, \dots n\}$$

by the triangular inequality, we also have

$$(1-\alpha)\sum_{i=1}^{n}|a_i| + \alpha\sum_{i=1}^{n}|b_i| \geq \sum_{i=1}^{n}|(1-\alpha)a_i + \alpha b_i|$$

Thus, $T_2(x)$ is convex. Also as we showed in Section 3, ℓ_1 norm is not smooth at $\boldsymbol{x} \in \{\boldsymbol{x} \in \mathbb{R}^n : x_i = 0, \forall i \in \{1, 2, \dots n\}\}$.

4.2 ii

To improve the optimization stability, the data are normalized such that each feature has 0 mean and unit standard deviation. Also the 9-th column is removed from the data, since it remains constant in all samples. The problem is optimized with subgradient descent with $\lambda = 0.1$ and step size $\eta = 0.01$ and $\eta = 0.001$ for 1000 iterations. For the subgradient method, a subgradient is randomly chosen from the set of subgradients. The code of the implementation is deferred to Appendix A.2 and the experimental results are visualized in Figure 2.

Here we optimize the constrained objective function with subgradient descent and proximal gradient descent. Since $T_2(x)$ is well-defined, we have analytical solution for each step of both optimization methods. It is clearly observed that subgradient method is not guaranteed that each step will decrease the objective values. With the same step size $\eta = 0.01$, subgradient method causes fluctuation in the optimization dynamics while the proximal method is able to converge stably. With smaller step size $\eta = 0.001$ the subgradient method can converge in a more stable way. The experimental results are consistent with the theoretical analysis, where we have convergence rate of $\frac{1}{T}$ for both subgradient and proximal method in the strongly convex case.

5 Problem: ADMM

5.1 i

The original problem can firstly be reformulated in the form of

$$\begin{aligned} \min_{x} \quad & c^\top x \\ s.t. \quad & Ax + y = b \\ & y \leq 0 \\ & l \geq x \leq u \end{aligned}$$

This can be further reformulated with indicator functions as follows

$$\begin{aligned} \min_{x,x,y,y} \quad & c^\top x + I_{\Omega_1}(x,y) + I_{\Omega_2}(\tilde{x}) + I_{\Omega_3}(\tilde{y}) \\ s.t. \quad & x = \tilde{x} \\ & y = \tilde{y} \end{aligned}$$

where $\Omega_1 = \{(x \in \mathbb{R}^n, y \in \mathbb{R}^m) : Ax + y = b\}$, $\Omega_2 = \{\tilde{x} : l \leq \tilde{x} \leq u\}$, and $\Omega_3 = \{\tilde{y} : \tilde{y} \geq 0\}$. It is easy to confirm that Ω_1 Ω_2 and Ω_3 are convex.

5.2 ii

To apply ADMM to solve the problem, we first decompose the problem.
Let

$$\begin{aligned} & f(x,y) = c^\top x \\ & g(\tilde{x},\tilde{y}) = 1 \\ & k(x,y,\tilde{x},\tilde{y}) = \begin{pmatrix} x - \tilde{x} \\ y - \tilde{y} \end{pmatrix} \end{aligned}$$

With constant α, the augmented Lagrangian w.r.t. the constraint $x = \tilde{x}$ and $y = \tilde{y}$ is then

$$\begin{aligned} \mathcal{L}_\alpha(x,y,\tilde{x},\tilde{y},\lambda) = f(x,y) + g(\tilde{x},\tilde{y}) - \lambda^\top k(x,y,\tilde{x},\tilde{y}) + \frac{\alpha}{2}\|k(x,y,\tilde{x},\tilde{y})\|_2^2 \\ \text{s.t.} Ax + y = b \\ l \leq \tilde{x} \leq u,\ y \geq 0 \end{aligned}$$

At t-th iteration, the ADMM steps are given as follows:

x, y update:

$$\begin{aligned} (x,y)^{t+1} = \arg\min_{x,y}\ & c^\top x - \lambda^{t\top} k(x,y,\tilde{x}^t,\tilde{y}^t) + \frac{\alpha}{2}\|k(x,y,\tilde{x}^t,\tilde{y}^t)\|_2^2 \\ \text{s.t. } & Ax + y = b \end{aligned}$$

$\tilde{x}, \tilde{y}$ update:

$$\begin{aligned} (\tilde{x},\tilde{y})^{t+1} = \arg\min_{\tilde{x},\tilde{y}}\ & -\lambda^{t\top} k(x^{t+1},y^{t+1},\tilde{x},\tilde{y}) + \frac{\alpha}{2}\|k(x^{t+1},y^{t+1},\tilde{x},\tilde{y})\|_2^2 \\ \text{s.t. } & l \leq \tilde{x} \leq u,\ \tilde{y} \geq 0 \end{aligned}$$

λ update:

$$\lambda^{t+1} = \lambda^t - \alpha \begin{pmatrix} x^{t+1} - \tilde{x}^t \\ y^{t+1} - \tilde{y}^{t+1} \end{pmatrix}$$

5.3 iii

It is not obvious how to solve subproblems to update x, y and $\tilde{x}, \tilde{y}$ in each step. Here we further simplify the ADMM iterations.

x, y **update** The optimality condition for the subproblem is

$$\begin{aligned}(x^\star, y^\star) &\in \Omega = \{(x, y) : Ax + y = b\},\\ -\nabla_x \mathcal{L}(x^\star, y^\star, \tilde{x}^t, \tilde{y}^t, \lambda^t) &+ (A, I)^\top \mu^\star = 0\end{aligned}$$

where $\mu^\star$ denotes the Lagrange multiplier associated with $Ax + y = b$. $\mu^\star$ can be analytically solved as

$$\mu^\star = \lambda_y^t - \alpha(y - \tilde{y}^t)$$

Since we have

$$\begin{aligned}\nabla_x \mathcal{L}(x, y, \tilde{x}^t, \tilde{y}^t, \lambda^t) &= c - \lambda_x^t + \alpha(x - \tilde{x}^t)\\ \nabla_y \mathcal{L}(x, y, \tilde{x}^t, \tilde{y}^t, \lambda^t) &= -\lambda_y^t + \alpha(y - \tilde{y}^t)\end{aligned}$$

The optimality condition can be thus written in the matrix form

$$\begin{pmatrix} I & A^\top \\ A & -I \end{pmatrix} \begin{pmatrix} x \\ y - \tilde{y}^t - \frac{\lambda_y^k}{\alpha} \end{pmatrix} = \begin{pmatrix} \tilde{x}^t + \frac{\lambda_x^t}{\alpha} - \frac{c}{\alpha} \\ \tilde{y}^t + \frac{\lambda_y^t}{\alpha} \end{pmatrix} \tag{2}$$

By furthur simplify the matrix form, we have following update rules for the step of updating x, y:

$$\begin{pmatrix} x^{t+1} \\ y^{t+1} \end{pmatrix} = \begin{pmatrix} I & A^\top \\ A & -I \end{pmatrix}^{-1} \begin{pmatrix} \tilde{x}^t + \frac{\lambda_x^t}{\alpha} - \frac{c}{\alpha} \\ \tilde{y}^t + \frac{\lambda_y^t}{\alpha} \end{pmatrix} + \begin{pmatrix} 0 \\ \tilde{y}^t + \frac{\lambda_y^t}{\alpha} \end{pmatrix}$$

$\tilde{x}, \tilde{y}$ **update:** In this step, the objective of the subproblem is

$$\begin{aligned}(\tilde{x}^\star, \tilde{y}^\star) &= \underset{\tilde{x}\in\Omega_2, \tilde{y}\in\Omega_3}{\arg\min} \; -\lambda^\top \begin{pmatrix} x^{t+1} - \tilde{x} \\ y^{t+1} - \tilde{y} \end{pmatrix} + \frac{\alpha}{2} \left\| \begin{pmatrix} x^{t+1} - \tilde{x} \\ y^{t+1} - \tilde{y} \end{pmatrix} \right\|_2^2 \\ &= (P_{\Omega_2}(\tilde{x}^\circ), P_{\Omega_3}(\tilde{y}^\circ))\end{aligned}$$

$$\text{where} \quad (\tilde{x}^\circ, \tilde{y}^\circ) = \underset{\tilde{x}, \tilde{y}}{\arg\min} -\lambda^\top \begin{pmatrix} x^{t+1} - \tilde{x} \\ y^{t+1} - \tilde{y} \end{pmatrix} + \frac{\alpha}{2} \left\| \begin{pmatrix} x^{t+1} - \tilde{x} \\ y^{t+1} - \tilde{y} \end{pmatrix} \right\|_2^2$$

The optimality condition for $\tilde{x}$ and $\tilde{y}$ with no constraints is given by

$$\begin{aligned}\nabla_{\tilde{x}} \mathcal{L}(x^{t+1}, y^{t+1}, \tilde{x}^\circ, \tilde{y}^\circ, \lambda^t) &= \lambda_x^{t+1} - \alpha(x^{t+1} - \tilde{x}^\circ) = 0\\ \nabla_{\tilde{y}} \mathcal{L}(x^{t+1}, y^{t+1}, \tilde{x}^\circ, \tilde{y}^\circ, \lambda^t) &= \lambda_y^{t+1} - \alpha(y^{t+1} - \tilde{y}^\circ) = 0\end{aligned}$$

Thus, we have $\tilde{x}^\circ = x^{t+1} - \frac{\lambda_x^t}{\alpha}$ and $\tilde{y}^\circ = y^{t+1} - \frac{\lambda_y^t}{\alpha}$.

Then the update rules for $\tilde{x}$ and $\tilde{y}$ are given by

$$\begin{aligned}\tilde{x}^{t+1} &= P_{\Omega_2}(x^{t+1} - \frac{\lambda_x^t}{\alpha})\\ \tilde{y}^{t+1} &= P_{\Omega_3}(y^{t+1} - \frac{\lambda_y^t}{\alpha})\end{aligned}$$

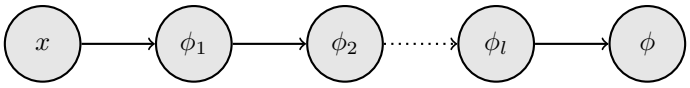

Figure 3: Computation graph for nested function 11.1

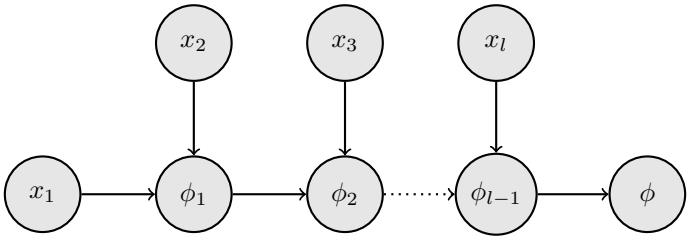

Figure 4: Computation graph for nested function 11.4

5.4 iv

The bottleneck for the computation is the inversion of matrix $\begin{pmatrix} I & A^\top \\ A & -I \end{pmatrix}$, which requires $\mathcal{O}((m+n)^3)$ flops. In this problem, since we have a constant α, it remains constant between iterations as well. Since the matrix is symmetric and invertible, we can apply factorization method to invert it. If the matrix is positive definite, we can use Cholesky decomposition with $\frac{1}{3}(m+n)^3$ flops. Otherwise we can use LU decomposition with $\frac{2}{3}(m+n)^3$ flops.

6 Problem 11.2

The computation graph for function 11.1 and 11.4 in the problem is shown in Figure 3 and Figure 4, respectively.

A Code

A.1 Projected Gradient Descent

```python
import numpy as np
from matplotlib import pyplot as plt


def project_onto_unit_ball(x):
    """Project point x onto unit ball."""
    l2norm = np.linalg.norm(x)
    x = x / l2norm if l2norm > 1 else x
    return x

def project_onto_unit_simplex(x):
    """Project point x onto simplex.
    Reference:
    https://gist.github.com/mblondel/6f3b7aaad90606b98f71
    """
    # get the dimension of x
    dim = len(x)

    # sort x in descending order
    mu = np.sort(x)[::-1]

    # calculate the cumulative sum of mu
    cumm_sum_mu = np.cumsum(mu)

    # find rho
    indices = np.arange(dim) + 1
    conditions = (mu * indices - (cumm_sum_mu - 1) > 0)
    rho = indices[conditions][-1]

    # calculate theta
    theta = (cumm_sum_mu-1)[rho-1] / float(rho)

    projected_x = np.maximum(x - theta, 0)

    return projected_x


def project_onto_unit_box(x):
    """Project point x onto unit box"""
    x = np.maximum(x, 0)
    x = np.minimum(x, 1)
    return x


def projection_gradient_descent_step(theta, grad, learning_rate, project_fn):
    theta -= learning_rate * grad
    theta = project_fn(theta)
    return theta
```

```python
def main():
    # solve $\min c^\top x$ with different constraints:
    # a). unit ball
    # b). unit simplex
    # c). unit box
    configs = {
        "Unit Ball": project_onto_unit_ball,
        "Unit Simplex": project_onto_unit_simplex,
        "Unit Box": project_onto_unit_box,
    }

    cs = [[0.3, 0.2], [-0.3, -0.2],
          [0, 0.2], [0, -0.2],
          [0.2, 0.2], [-0.2, -0.2]]
    # cs = cs[:1]
    cs = list(map(np.array, cs))

    max_steps = 1000
    for c in cs:
        paths = {}
        for constraint, proj_fn in configs.items():
            theta = np.array([4., 2.])
            learning_rate = 0.05
            grad = c
            paths[constraint] = []
            for _ in range(max_steps):
                paths[constraint].append(theta)
                theta = projection_gradient_descent_step(
                    theta, grad, learning_rate, proj_fn)
            print(f"Constraint: {constraint}", theta)

        # plot in 2D with ratio 4:4
        plot_lim = 2.5
        fig = plt.figure(figsize=(4, 4))
        ax = fig.add_subplot(111)
        ax.set_xlim(-plot_lim, plot_lim)
        ax.set_ylim(-plot_lim, plot_lim)
        ax.set_xlabel(r"$x_1$")
        ax.set_ylabel(r"$x_2$")
        grad = f"({grad[0]},{grad[1]})"
        ax.set_title(f"Projected Gradient Descent: c={grad}")

        colors = ['tab:blue', 'tab:orange', 'tab:green']
        for idx, (constraint, path) in enumerate(paths.items()):
            path = np.array(path)
            ax.plot(path[:, 0], path[:, 1], linewidth=0.8,
                    label=constraint, color=colors[idx],)

        # plot unit ball constraint
        ax.add_artist(
            plt.Circle((0, 0), 1, linestyle="--",
                       fill=False, color=colors[0], alpha=0.5))
        # plot unit simplex constraint
        ax.plot([0, 1], [1, 0], "--", color=colors[1], alpha=0.5)
```

```
        # plot unit box constraint
        ax.plot([0, 1], [0, 0], "--", color=colors[2], alpha=0.5)
        ax.plot([0, 0], [0, 1], "--", color=colors[2], alpha=0.5)
        ax.plot([1, 1], [0, 1], "--", color=colors[2], alpha=0.5)
        ax.plot([0, 1], [1, 1], "--", color=colors[2], alpha=0.5)

        ax.legend()
        plt.savefig(f"as3/projected_gd_{grad}.pdf", dpi=300)


if __name__ == "__main__":
    main()
```

A.2 Subgradient Descent

```
# code for l1 regularized linear regression
import numpy as np
from matplotlib import pyplot as plt
from dataclasses import dataclass


# read text file
def read_data(filename):
    data = np.loadtxt(filename)
    return data


def standarize(A):
    # standarize each column of A
    min_val = A.min(axis=0)
    max_val = A.max(axis=0)

    standarized = []
    for i in range(A.shape[1]):
        # skip if all values are the same
        if min_val[i] == max_val[i]:
            continue
        else:
            standarized.append((A[:, i] - min_val[i])/(max_val[i] - min_val[i]))
    return np.array(standarized).T


@dataclass
class L1LRSolver:
    A: np.ndarray
    b: np.array
    lam: float
    step_size: float
    max_step: int
    x: np.array = None

    def get_value(self):
        N = self.A.shape[0]
        residual = self.A@self.x - self.b
```

```
        t1 = 0.5 * (residual.T @ residual)/N
        t2 = self.lam * np.abs(self.x).sum()

        return t1 + t2

    def get_gradient(self):
        N = self.A.shape[0]
        grad = (self.A.T @ (self.A @ self.x - self.b))/N
        return grad


class SubgradientDescent(L1LRSolver):
    def get_subgradient(self):
        # compute gradient of T1
        t1_grad = self.get_gradient()

        # compute subgradient of T2
        t2_grad = self.lam * np.sign(self.x)
        # a random subgradient is sampled if x = 0 from [-1, 1]
        num_zeros = np.sum(self.x < 1e-7)
        t2_grad[self.x < 1e-7] = np.random.uniform(-1, 1, num_zeros)

        subgrad = t1_grad + t2_grad
        return subgrad

    def step(self):
        # compute subgradient
        subgrad = self.get_subgradient()
        # update x
        self.x -= self.step_size * subgrad

        # compute value
        value = self.get_value()

        return value

    def solve(self):
        values = []
        for i in range(self.max_step):
            value = self.step()
            values.append(value)
        return np.array(values)


class ProximalGradientDescent(L1LRSolver):
    def prox_map(self):
        proxed = np.sign(self.x) * np.maximum(np.abs(self.x) - self.lam * self.step_size, 0.)
        return proxed

    def step(self):
        # compute gradient
        grad = self.get_gradient()
        # update x
        self.x -= self.step_size * grad
```

```
            # proximal map
            self.x = self.prox_map()

            # compute value
            value = self.get_value()

            return value

        def solve(self):
            values = []
            for i in range(self.max_step):
                value = self.step()
                values.append(value)
            return np.array(values)


    def main():
        data = read_data('./data/data.txt')

        A = data[:, :-2]
        A = standarize(A)
        b = data[:, -2]

        lam = 0.1
        max_step = 3000
        step_size = 1e-2

        # start from zero
        results = {}

        x = np.zeros(A.shape[1])
        step_size = 1e-2
        prox_solver = ProximalGradientDescent(A, b, lam, step_size, max_step, x)
        results[f'Proximal: $\eta={step_size}$'] = prox_solver.solve()

        x = np.zeros(A.shape[1])
        step_size = 1e-2
        sub_solver = SubgradientDescent(A, b, lam, step_size, max_step, x)
        results[f'Subgradient: $\eta={step_size}$'] = sub_solver.solve()

        x = np.zeros(A.shape[1])
        step_size = 2e-3
        sub_solver = SubgradientDescent(A, b, lam, step_size, max_step, x)
        results[f'Subgradient: $\eta={step_size}$'] = sub_solver.solve()

        # plot
        for key in results:
            plt.plot(results[key], label=key)
        plt.xlabel('Num. step')
        plt.ylabel(r'$f(x)$')
        plt.title("Subgradient vs Proximal\n$\ell_1$ regularized linear regression")
        plt.legend()

        plt.savefig('./l1lr.pdf', dpi=300)
```

```python
if __name__ == '__main__':
    main()
```