MAS374 Optimization Theory
Homework #9

20150597 Jeonghwan Lee*

Department of Mathematical Sciences, KAIST

December 19, 2021

I worked on this programming assignment by using Python 3 (version 3.7.7). I utilized PyCharm 2021.1 Community Edition as an integrated development environment (IDE).

Problem 1.

Consider the optimization problem

$$\begin{aligned} &\min_{\mathbf{y}\in\mathbb{R}^n} \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 \\ &\text{subject to } \mathbf{A}\mathbf{y} \preceq \mathbf{b}, \end{aligned} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is any given fixed point in $\mathbb{R}^n$, and $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$.

(a) The Lagrangian function of the primal convex QP (1) is $\mathcal{L}(\cdot,\cdot) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, where

$$\begin{aligned} \mathcal{L}(\mathbf{y},\boldsymbol{\lambda}) &:= \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 + \boldsymbol{\lambda}^\top(\mathbf{A}\mathbf{y}-\mathbf{b}) \\ &= \frac{1}{2}\mathbf{y}^\top\mathbf{y} - \left(\mathbf{x}-\mathbf{A}^\top\boldsymbol{\lambda}\right)^\top\mathbf{y} + \frac{1}{2}\mathbf{x}^\top\mathbf{x} - \boldsymbol{\lambda}^\top\mathbf{b}. \end{aligned}$$

Thus we have $\nabla_{\mathbf{y}}\mathcal{L}(\mathbf{y},\boldsymbol{\lambda}) = \mathbf{y} - \left(\mathbf{x}-\mathbf{A}^\top\boldsymbol{\lambda}\right)$, which implies that

$$\operatorname{argmin}\{\mathcal{L}(\mathbf{y},\boldsymbol{\lambda}) : \mathbf{y}\in\mathbb{R}^n\} = \left\{\mathbf{x}-\mathbf{A}^\top\boldsymbol{\lambda}\right\}. \tag{2}$$

Therefore, the Lagrange dual function for (1) is given by $g(\cdot) : \mathbb{R}^m \to [-\infty,+\infty)$, where

$$\begin{aligned} g(\boldsymbol{\lambda}) &:= \inf\{\mathcal{L}(\mathbf{y},\boldsymbol{\lambda}) : \mathbf{y}\in\mathbb{R}^n\} \\ &\overset{\text{(a)}}{=} \mathcal{L}\left(\mathbf{x}-\mathbf{A}^\top\boldsymbol{\lambda},\boldsymbol{\lambda}\right) \\ &= -\frac{1}{2}\left(\mathbf{x}-\mathbf{A}^\top\boldsymbol{\lambda}\right)^\top\left(\mathbf{x}-\mathbf{A}^\top\boldsymbol{\lambda}\right) + \frac{1}{2}\mathbf{x}^\top\mathbf{x} - \mathbf{b}^\top\boldsymbol{\lambda}. \end{aligned}$$

(b) For any non-empty convex & closed set $\Omega \subseteq \mathbb{R}^n$, let $\mathcal{P}_\Omega(\cdot) : \mathbb{R}^n \to \Omega$ denote the Euclidean projection map of $\mathbb{R}^n$ onto Ω, *i.e.*, $\operatorname{argmin}\{\|\mathbf{x}-\mathbf{y}\|_2 : \mathbf{y}\in\Omega\} = \{\mathcal{P}_\Omega(\mathbf{x})\}$ for every $\mathbf{x}\in\mathbb{R}^n$. Here, we highlight that the optimal solution to the optimization problem

$$\min_{\mathbf{y}\in\Omega} \|\mathbf{x}-\mathbf{y}\|_2$$

*E-mail: sa8seung@kaist.ac.kr

uniquely exists due to the projection theorem.

Now, let $\mathbf{y}^* = \mathcal{P}_{\mathcal{X}}(\mathbf{x})$ be the optimal solution to the primal convex QP (1) and $\boldsymbol{\lambda}^* \in \mathbb{R}^m$ be the optimal solution to the dual problem associated to (1):

$$\begin{aligned} \max_{\boldsymbol{\lambda} \in \mathbb{R}^m} \quad & -\frac{1}{2}\left(\mathbf{x} - \mathbf{A}^\top \boldsymbol{\lambda}\right)^\top \left(\mathbf{x} - \mathbf{A}^\top \boldsymbol{\lambda}\right) + \frac{1}{2}\mathbf{x}^\top \mathbf{x} - \mathbf{b}^\top \boldsymbol{\lambda} \\ \text{subject to } & \boldsymbol{\lambda} \succeq \mathbf{0}. \end{aligned} \tag{3}$$

Here, we recall our assumption that the primal feasible set $\mathcal{X} := \{\mathbf{y} \in \mathbb{R}^n : \mathbf{A}\mathbf{y} \preceq \mathbf{b}\} \subseteq \mathbb{R}^n$ has a non-empty relative interior. Let $\tilde{\mathbf{y}} \in \mathsf{relint}(\mathcal{X})$. Since $\tilde{\mathbf{y}}$ is clearly a strictly feasible point of the primal convex QP (1), the Slater's condition for convex optimization problems (*Proposition 8.7* in [2]) implies the strong duality between the primal problem (1) and the corresponding dual problem (3). Hence, the Karush-Kuhn-Tucker (KKT) conditions for the pair $(\mathbf{y}^*, \boldsymbol{\lambda}^*)$ hold:

(i) Lagrangian stationarity: $\nabla_{\mathbf{y}} \mathcal{L}\left(\mathbf{y}^*, \boldsymbol{\lambda}^*\right) = \mathbf{0}$;

(ii) Complementary slackness: $\left(\boldsymbol{\lambda}^*\right)^\top \left(\mathbf{A}\mathbf{y}^* - \mathbf{b}\right) = 0$;

(iii) Primal feasibility: $\mathbf{A}\mathbf{y}^* \preceq \mathbf{b}$;

(iv) Dual feasibility: $\boldsymbol{\lambda}^* \succeq \mathbf{0}$.

The condition (i) yields

$$\mathbf{y}^* - \left(\mathbf{x} - \mathbf{A}^\top \boldsymbol{\lambda}^*\right) = \mathbf{0} \in \mathbb{R}^n,$$

thereby we have

$$\mathbf{y}^* = \mathcal{P}_{\mathcal{X}}(\mathbf{x}) = \mathbf{x} - \mathbf{A}^\top \boldsymbol{\lambda}^*. \tag{4}$$

At this point, one can ask the following natural question: Does $\mathbf{x} - \mathbf{A}^\top \boldsymbol{\lambda}^*$ really belong to the primal feasible set $\mathcal{X}$? This question can be settled by performing Lagrangian duality analysis of the dual problem (3). The Lagrangian function $\mathcal{L}_d(\cdot, \cdot) : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is given by

$$\mathcal{L}_d(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \frac{1}{2}\left(\mathbf{x} - \mathbf{A}^\top \boldsymbol{\lambda}\right)^\top \left(\mathbf{x} - \mathbf{A}^\top \boldsymbol{\lambda}\right) - \frac{1}{2}\mathbf{x}^\top \mathbf{x} + \mathbf{b}^\top \boldsymbol{\lambda} - \boldsymbol{\nu}^\top \boldsymbol{\lambda}.$$

Let $\boldsymbol{\nu}^* \in \mathbb{R}^m$ be an optimal solution to the dual formulation of the dual problem (3). Since the strong duality for the dual problem (3) clearly holds due to the the Slater's condition for convex programs (*Proposition 8.7* in [2]), the Karush-Kuhn-Tucker (KKT) conditions for the pair $(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ hold:

(v) Lagrangian stationarity: $\nabla_{\boldsymbol{\lambda}} \mathcal{L}_d\left(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*\right) = \mathbf{0}$;

(vi) Complementary slackness: $\left(\boldsymbol{\nu}^*\right)^\top \boldsymbol{\lambda}^* = 0$;

(vii) Primal feasibility: $\boldsymbol{\lambda}^* \succeq \mathbf{0}$;

(viii) Dual feasibility: $\boldsymbol{\nu}^* \succeq \mathbf{0}$.

From the condition (v), we obtain

$$\mathbf{0} = \nabla_{\boldsymbol{\lambda}} \mathcal{L}_d\left(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*\right) = \left(\mathbf{A}\mathbf{A}^\top\right) \boldsymbol{\lambda}^* - \left(\mathbf{A}\mathbf{x} - \mathbf{b} + \boldsymbol{\nu}^*\right). \tag{5}$$

It follows that

$$\mathbf{A}\left(\mathbf{x}-\mathbf{A}^\top\boldsymbol{\lambda}^*\right) \overset{\text{(b)}}{=} \mathbf{A}\mathbf{x}-(\mathbf{A}\mathbf{x}-\mathbf{b}+\boldsymbol{\nu}^*) = \mathbf{b}-\boldsymbol{\nu}^* \overset{\text{(c)}}{\preceq} \mathbf{b},$$

where the step (b) follows from the equation (5), and the step (c) holds due to the condition (viii), thereby $\mathbf{x}-\mathbf{A}^\top\boldsymbol{\lambda}^* \in \mathcal{X}$ as desired.

Problem 2.

(a) The next code defines a function `dual_proj(l)` which takes a vector $\mathtt{l} \in \mathbb{R}^m$ and returns its projection $\mathcal{P}_{\boldsymbol{\Lambda}}(\mathtt{l}) = [\mathtt{l}]_{\boldsymbol{\Lambda}}$ onto the dual feasible set $\boldsymbol{\Lambda} := \mathbb{R}^m_+$:

```python
import numpy as np

#### ---- Problem 2(a) ---- ####

def dual_proj(l):
    dim = l.shape[0]
    proj_l = np.zeros(dim)
    for i in range(dim):
        if l[i] >= 0:
            proj_l[i] = l[i]
        else:
            proj_l[i] = 0
    return proj_l
```

Here, we note that for every $\boldsymbol{\lambda} \in \mathbb{R}^m$,

$$[\mathcal{P}_{\boldsymbol{\Lambda}}(\boldsymbol{\lambda})]_i = \lambda_i^+ = \max\{\lambda_i, 0\}, \ \forall i \in [m].$$

(b) The next code defines a function `dual_grad(l, x, A, b)` which takes a vector $\mathtt{l} \in \mathbb{R}^m$ and returns $-\nabla_{\boldsymbol{\lambda}} g(\mathtt{l}) \in \mathbb{R}^m$:

```python
import numpy as np

#### ---- Problem 2(b) ---- ####

def dual_grad(l, x, A, b):
    grad = np.dot(np.dot(A, np.transpose(A)), l) - (np.dot(A, x) - b)
    return grad
```

Here, we note that

$$-\nabla_{\boldsymbol{\lambda}} g(\boldsymbol{\lambda}) = \left(\mathbf{A}\mathbf{A}^\top\right)\boldsymbol{\lambda} - (\mathbf{A}\mathbf{x}-\mathbf{b}), \ \forall \boldsymbol{\lambda} \in \mathbb{R}^m. \tag{6}$$

(c) One can observe from (6) that

$$-\nabla_{\boldsymbol{\lambda}} g(\mathbf{v}) + \nabla_{\boldsymbol{\lambda}} g(\mathbf{u}) = \left(\mathbf{A}\mathbf{A}^\top\right)(\mathbf{v}-\mathbf{u}), \ \forall \mathbf{u},\mathbf{v} \in \mathbb{R}^m,$$

thereby we obtain

$$\|-\nabla_{\boldsymbol{\lambda}} g(\mathbf{v}) + \nabla_{\boldsymbol{\lambda}} g(\mathbf{u})\|_2 = \left\|\left(\mathbf{A}\mathbf{A}^\top\right)(\mathbf{v}-\mathbf{u})\right\|_2 \le \left\|\mathbf{A}\mathbf{A}^\top\right\|_{2\to 2} \cdot \|\mathbf{v}-\mathbf{u}\|_2, \ \forall \mathbf{u},\mathbf{v} \in \mathbb{R}^m. \tag{7}$$

From the inequality (7), one can conclude that the function $-g(\cdot) : \mathbb{R}^m \to \mathbb{R}$ is L-smooth, *i.e.*, $-g(\cdot) : \mathbb{R}^m \to \mathbb{R}$ has a L-Lipschitz continuous gradient with the Lipschitz constant

$$L := \left\| \mathbf{A}\mathbf{A}^\top \right\|_{2\to 2} = \lambda_{\max}\left(\mathbf{A}\mathbf{A}^\top\right) = \sigma_{\max}(\mathbf{A})^2.$$

Lastly, we delineate the projected gradient method for solving the dual problem (3). With initialization step $\boldsymbol{\lambda}_0 := \mathbf{0} \in \boldsymbol{\Lambda}$, we perform the iterative procedure with the following update rule:

$$\boldsymbol{\lambda}_{k+1} = \mathcal{P}_{\boldsymbol{\Lambda}}\left(\boldsymbol{\lambda}_k - s_k\left\{-\nabla_{\boldsymbol{\lambda}} g\left(\boldsymbol{\lambda}_k\right)\right\}\right), \ \forall k \in \mathbb{Z}_+, \tag{8}$$

with the constant step-size $s_k = \frac{1}{L} = \sigma_{\max}\left(\mathbf{A}\right)^{-2}$. Note that the stopping criterion is

$$\left\|\left(\mathbf{x} - \mathbf{A}^\top\boldsymbol{\lambda}_k\right) - \left(\mathbf{x} - \mathbf{A}^\top\boldsymbol{\lambda}_{k+1}\right)\right\|_2 = \left\|\mathbf{A}^\top\left(\boldsymbol{\lambda}_{k+1} - \boldsymbol{\lambda}_k\right)\right\|_2 \leq \texttt{tol},$$

where $\texttt{tol} := 2^{-40}$. The next code defines a function `solve_dual(x, A, b)` which makes use of the projected gradient method (8) to solve the dual problem (3) within an accuracy $\texttt{tol} := 2^{-40}$ and returns the computed optimal solution $\boldsymbol{\lambda}^* \in \boldsymbol{\Lambda}$:

```python
import numpy as np
import math

#### ---- Problem 2(c) ---- ####

def solve_dual(x, A, b):
    tol = 2 ** -40
    dim = A.shape[0]
    u, s, vh = np.linalg.svd(A, full_matrices=True)
    L1 = s[0] ** 2
    learning_rate = 1/L1
    next_itr = np.zeros(dim)
    distance = math.inf
    while distance > tol:
        current_itr = next_itr
        next_itr = dual_proj(current_itr - (learning_rate * dual_grad(current_itr, x, A, b
    )))
        distance = np.linalg.norm(np.dot(np.transpose(A), next_itr) - np.dot(np.transpose(
    A), current_itr), 2)
    return next_itr
```

Problem 3.

(a) From the equation (4), we know that for every $\mathbf{x} \in \mathbb{R}^n$,

$$\mathcal{P}_{\mathcal{X}}(\mathbf{x}) = [\mathbf{x}]_{\mathcal{X}} = \mathbf{x} - \mathbf{A}^\top\boldsymbol{\lambda}^*,$$

where $\boldsymbol{\lambda}^* \in \mathbb{R}^m$ is the optimal solution to the dual problem (3). So the source code defining the function `primal_proj(x, A, b)` can be written as follows:

```python
import numpy as np

#### ---- Problem 3(a) ---- ####
```

```python

def prim_proj(x, A, b):
    dual_opt = solve_dual(x, A, b)
    return x - np.dot(np.transpose(A),dual_opt)
```

(b) Note that $\nabla_{\mathbf{x}} f_0(\mathbf{x}) = \mathbf{H}\mathbf{x} + \mathbf{c}$ for every $\mathbf{x} \in \mathbb{R}^n$. Therefore, the source code defining two functions `grad_f0(x, H, c)` and `f0(x, H, c)` which evaluate the value of $\nabla_{\mathbf{x}} f_0(\mathbf{x}) \in \mathbb{R}^n$ and $f_0(\mathbf{x}) \in \mathbb{R}$, respectively:

```python
import numpy as np

#### ---- Problem 3(b) ---- ####

def grad_f0(x, H, c):
    return np.dot(H, x) + c

def f0(x, H, c):
    return 1/2 * np.dot(np.transpose(x), np.dot(H, x)) + np.dot(np.transpose(c), x)
```

(c) It's clear that

$$\left\|\nabla_{\mathbf{x}} f_0(\mathbf{v}) - \nabla_{\mathbf{x}} f_0(\mathbf{u})\right\|_2 = \left\|\mathbf{H}\left(\mathbf{v} - \mathbf{u}\right)\right\|_2 \leq \left\|\mathbf{H}\right\|_{2\to 2} \cdot \left\|\mathbf{v} - \mathbf{u}\right\|_2, \ \forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^n. \tag{9}$$

From the inequality (9), one can conclude that the function $f_0(\cdot) : \mathbb{R}^n \to \mathbb{R}$ is L-smooth, *i.e.*, $f_0(\cdot) : \mathbb{R}^n \to \mathbb{R}$ has a L-Lipschitz continuous gradient with the Lipschitz constant

$$L := \left\|\mathbf{H}\right\|_{2\to 2} = \lambda_{\max}(\mathbf{H}).$$

Finally, we take a closer inspection on the projected gradient method for solving our original QP:

$$\begin{aligned} p^* = \min_{\mathbf{x} \in \mathbb{R}^n} \ & f_0(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{H}\mathbf{x} + \mathbf{c}^\top \mathbf{x} \\ \text{subject to } & \mathbf{A}\mathbf{x} \preceq \mathbf{b}. \end{aligned} \tag{10}$$

With initialization step $\mathbf{x}_0 := \mathbf{0} \in \mathbb{R}^n$, we perform the iterative procedure with the following update rule:

$$\mathbf{x}_{k+1} = \mathcal{P}_{\mathcal{X}}\left(\mathbf{x}_k - s_k \nabla_{\mathbf{x}} f_0\left(\mathbf{x}_k\right)\right), \ \forall k \in \mathbb{Z}_+, \tag{11}$$

with the constant step-size $s_k = \frac{1}{L} = \lambda_{\max}(\mathbf{H})^{-1}$. The next source code defines a function `solve_dual(H, c, A, b)` which utilizes the projected gradient method (11) to solve the original QP (10) within an accuracy `eps` $:= 2^{-40}$ and returns the computed optimal solution $\mathbf{x}^* \in \mathcal{X}$:

```python
import numpy as np
import math

#### ---- Problem 3(c) ---- ####

def solve_prim(H, c, A, b):
    eps = 2 ** -40
    dim = H.shape[0]
    u, s, vh = np.linalg.svd(H, full_matrices=True)
    L2 = s[0]
    learning_rate = 1/L2
```

```
    next_itr = np.zeros(dim)
    distance = math.inf
    while distance > eps:
        current_itr = next_itr
        next_itr = prim_proj(current_itr - (learning_rate * grad_f0(current_itr, H, c)), A
    , b)
        distance = np.linalg.norm(next_itr - current_itr, 2)
    return next_itr

x_opt = solve_prim(H, c, A, b)

# printing the results
print_results(x_opt, H, c)
```

At this point, we clarify the reason why the projected gradient method (11) works well with initialization $\mathbf{x}_0 = \mathbf{0} \in \mathbb{R}^n$, even though $\mathbf{0} \in \mathbb{R}^n$ might not be in the primal feasible set $\mathcal{X} \subseteq \mathbb{R}^n$. Note that the projected gradient method (11) with initialization $\mathbf{x}_0 = \mathbf{0}$ is essentially the same with the projected gradient method (11) with initialization $\mathbf{x}_1 = \mathcal{P}_{\mathcal{X}}\left(-\frac{1}{\lambda_{\max}(\mathbf{H})}\mathbf{c}\right) \in \mathcal{X}$. Since $\mathbf{H} \in \mathcal{S}^n_{++}$, where $\mathcal{S}^n_{++}$ denotes the set of all $n \times n$ real symmetric, positive definite matrices, it holds that

$$\lambda_{\min}(\mathbf{H}) \cdot \mathbf{I}_n \preceq \mathbf{H} = \nabla^2_{\mathbf{x}} f_0(\mathbf{x}) \preceq \lambda_{\max}(\mathbf{H}) \cdot \mathbf{I}_n, \ \forall \mathbf{x} \in \mathbb{R}^n. \tag{12}$$

From (12), one can see that the objective function $f_0(\cdot) : \mathbb{R}^n \to \mathbb{R}$ is a $\lambda_{\min}(\mathbf{H})$-strongly convex and $\lambda_{\max}(\mathbf{H})$-smooth function. Therefore, $f_0(\cdot) : \mathbb{R}^n \to \mathbb{R}$ is $\lambda_{\min}(\mathbf{H})$-strongly convex and $\lambda_{\max}(\mathbf{H})$-smooth on the *primal feasible set* $\mathcal{X} \subseteq \mathbb{R}^n$ as well. According to *Theorem 3.10* in [1], the iterative procedure (11) of the projected gradient method with initialization $\mathbf{x}_1 = \mathcal{P}_{\mathcal{X}}\left(-\frac{1}{\lambda_{\max}(\mathbf{H})}\mathbf{c}\right) \in \mathcal{X}$ for solving the original QP (10) satisfies

$$\|\mathbf{x}_k - \mathbf{x}^*\|_2^2 \leq \exp\left\{-\frac{k-1}{\kappa(\mathbf{H})}\right\} \|\mathbf{x}_1 - \mathbf{x}^*\|_2^2, \ \forall k \in \mathbb{N}, \tag{13}$$

where $\kappa(\mathbf{H}) := \frac{\lambda_{\max}(\mathbf{H})}{\lambda_{\min}(\mathbf{H})} \in [1, +\infty)$ is the *condition number* of $\mathbf{H}$ (or the *condition number* of $f_0(\cdot) : \mathbb{R}^n \to \mathbb{R}$). From the bound (13), the projected gradient method (11) with initialization $\mathbf{x}_1 = \mathcal{P}_{\mathcal{X}}\left(-\frac{1}{\lambda_{\max}(\mathbf{H})}\mathbf{c}\right)$ achieves the target accuracy $\|\mathbf{x}_k - \mathbf{x}^*\|_2 \leq \epsilon$ for some $\epsilon \in (0, +\infty)$ in at most $2\kappa(\mathbf{H}) \log\left(\frac{\|\mathbf{x}_1 - \mathbf{x}^*\|_2}{\epsilon}\right) = \mathcal{O}\left(\kappa(\mathbf{H}) \log\left(\frac{1}{\epsilon}\right)\right)$ iterations. In a nutshell, the projected gradient method (11) with initialization $\mathbf{x}_0 = \mathbf{0}$ achieves a *linear rate of convergence* and this is the reason why the projected gradient method (11) with initialization $\mathbf{x}_0 = \mathbf{0} \in \mathbb{R}^n$ works well with initialization $\mathbf{x}_0 = \mathbf{0}$, even though $\mathbf{0} \in \mathbb{R}^n$ might not be in the primal feasible set $\mathcal{X} \subseteq \mathbb{R}^n$.

To sum up, the overall source code for this programming assignment is given as follows:

```
import numpy as np
import math

#### ---- Problem 2(a) ---- ####

def dual_proj(l):
    dim = l.shape[0]
    proj_l = np.zeros(dim)
    for i in range(dim):
        if l[i] >= 0:
```

```python
            proj_l[i] = l[i]
        else:
            proj_l[i] = 0
    return proj_l

#### ---- Problem 2(b) ---- ####

def dual_grad(l, x, A, b):
    grad = np.dot(np.dot(A, np.transpose(A)), l) - (np.dot(A, x) - b)
    return grad


#### ---- Problem 2(c) ---- ####

def solve_dual(x, A, b):
    tol = 2 ** -40
    dim = A.shape[0]
    u, s, vh = np.linalg.svd(A, full_matrices=True)
    L1 = s[0] ** 2
    learning_rate = 1/L1
    next_itr = np.zeros(dim)
    distance = math.inf
    while distance > tol:
        current_itr = next_itr
        next_itr = dual_proj(current_itr - (learning_rate * dual_grad(current_itr, x, A, b
    )))
        distance = np.linalg.norm(np.dot(np.transpose(A), next_itr) - np.dot(np.transpose(
    A), current_itr), 2)
    return next_itr


#### ---- Problem 3(a) ---- ####

def prim_proj(x, A, b):
    dual_opt = solve_dual(x, A, b)
    return x - np.dot(np.transpose(A),dual_opt)


#### ---- Problem 3(b) ---- ####

def grad_f0(x, H, c):
    return np.dot(H, x) + c

def f0(x, H, c):
    return 1/2 * np.dot(np.transpose(x), np.dot(H, x)) + np.dot(np.transpose(c), x)


#### --  A helper function which prints the results in a given format -- ####

def print_results(x_opt, H, c):
```

```python
    np.set_printoptions(floatmode="unique")  # print with full precision
    print("optimal value p* =")
    print("", f0(x_opt, H, c), sep="\t")
    print("\noptimal solution x* =")
    for coord in x_opt:
        print("", coord, sep='\t')
    return

# first example in page 3 of the document,
# written for you so you can test your code.

H = np.array([[6, 4],
              [4, 14]])
c = np.array([-1, -19])

A = np.array([[-3, 2],
              [-2, -1],
              [1, 0]])
b = np.array([-2, 0, 4])

#### ---- Problem 3(c) ---- ####

def solve_prim(H, c, A, b):
    eps = 2 ** -40
    dim = H.shape[0]
    u, s, vh = np.linalg.svd(H, full_matrices=True)
    L2 = s[0]
    learning_rate = 1/L2
    next_itr = np.zeros(dim)
    distance = math.inf
    while distance > eps:
        current_itr = next_itr
        next_itr = prim_proj(current_itr - (learning_rate * grad_f0(current_itr, H, c)), A
    , b)
        distance = np.linalg.norm(next_itr - current_itr, 2)
    return next_itr

x_opt = solve_prim(H, c, A, b)

# printing the results
print_results(x_opt, H, c)
```

References

[1] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Found. Trends Mach. Learn.*, 8(3–4):231–357, nov 2015.

[2] Giuseppe C Calafiore and Laurent El Ghaoui. *Optimization models*. Cambridge university press, 2014.