The supporting material provided below is for the second edition of the book published in 2021 and consists of two parts. The first part contains errata corrections found to date, while the second part contains solutions of *selected* end-of-chapter problems (see the note below that explains what the term "*selected*" is meant). We will try our best to keep the material up-to-date.

PART 1

ERRATA CORRECTIONS for the Second Edition of
***PRACTICAL OPTIMIZATION – ALGORITHMS and ENGINEERING APPLICATIONS,* Springer, 2021**
Andreas Antoniou and Wu-Sheng Lu
(Revision date: May 29, 2024)

Page xi, line 8, change "optimization/" to "~optimize/".

Page 112, Prob. 4.11, line 3, change "$\frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{X}$" to "$\frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$".

Page 126, line 15, change "g_x" to "$\mathbf{g}_x$".

Page 380, line 1, change "**of**" to "**in**".

Page 454, in Eq. (13.79c), delete "[5*pt*]".

Page 456, 2 lines below Eq. (13.87c), change "$\mathbf{M} = \mathbf{A}\mathbf{E}^{-1}\mathbf{F}\mathbf{A}^T$" to "$\mathbf{M} = -\mathbf{A}\mathbf{E}^{-1}\mathbf{F}\mathbf{A}^T$".

Page 464, line 7, change "**C** = – diag…" to "**C** = diag…".

Page 521, line –7, change "Eqs. (14.94a)-(14.94c)" to "Eqs.(14.84a)-(14.84c)".

Page 530, in the title of Algorithm 14.14, change "the problem" to "the problem in".

Page 532, Prob. 14.6, in both part (*a*) and part (*b*), change "$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$" to "$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$".

Page 532, Prob. 14.8, in part (*a*), line 4, change "***y***" to "***u***".

Page 536, Prob. 14.13, in part (*b*), line 3, change "$x_1 \leq 0$" to "$-x_1 \leq 0$".

Page 553, line 2, change "**Z**" to "$\mathbf{Z}_k$".

Page 554, in the last expression, change "if $\boldsymbol{\delta}_k^T$" to "if $\boldsymbol{\delta}_x^T\boldsymbol{\gamma}_k$".

Page 577, line –6, change "0.45" to "0.45π".

PART 2

SOLUTIONS of SELECTED END-of-CHAPTER PROBLEMS for
PRACTICAL OPTIMIZATION – ALGORITHMS and ENGINEERING APPLICATIONS
Second edition, Springer, 2021,
by Andreas Antoniou and Wu-Sheng Lu
(Date of the latest revision: June 24, 2024)

Important Notice

This document provides the reader solutions of ***selected*** end-of-chapter problems for the second edition of the book. Here "selected problems" are referred to those problems that are *not* present in the first edition of the book.

Another important thing to note is that, for the problems which are given in the first edition, their solutions from the solutions manual for the first edition may require modifications before they become valid for the second edition of the book. This is largely due to the notation as well as formulation changes made in the second edition. However, in most cases, the modifications (if necessary) are expected to be straightforward.

Thirdly, some of the solutions provided below involve a software package for specifying and solving convex programs known as CVX [R1]. To be familiar with the software, visit the the link below.

[R1] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.0 beta. http://cvxr.com/cvx, September 2013.

Chapter 1 The Optimization Problem

1.8

(**a**) With $n = 2$, $\mu_1 = 5$, $\mu_2 = 12$, $\sigma_1^2 = 9$, $\sigma_2^2 = 196$, $\rho_{12} = \rho_{21} = -0.5$, and $\mu_* = 7$, we computer parameters

$$q_{11} = 9, q_{12} = q_{21} = -21, \text{and } q_{22} = 196$$

which specify the problem in Eqs. (1.12a)-(1.12d) as

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{w}) = 9w_1^2 - 42w_1w_2 + 196w_2^2 \\ \text{subject to:} \quad & 5w_1 + 12w_2 \geq 7 \\ & w_1 \geq 0, w_2 \geq 0 \\ & w_1 + w_2 = 1 \end{aligned}$$

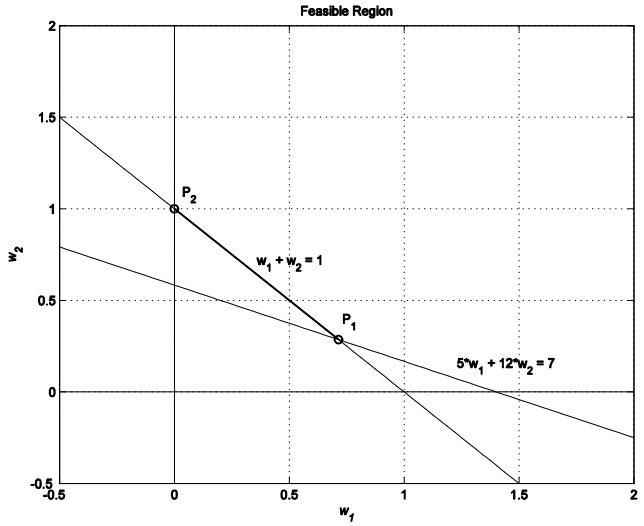
(**b**) As illustrated in the figure below, the feasible region in this case is a portion of the segment of the straight line $w_1 + w_2 = 1$ in the first quadrant of the w_1-w_2 plane with endpoints P_1 and P_2, where $P_2 = [0\ 1]^T$ and P_1 is determined by the equations

$$\begin{aligned} w_1 + w_2 &= 1 \\ 5w_1 + 12w_2 &= \mu_* = 7 \end{aligned}$$

whose solution is given by $w_1 = 5/7 = 0.7143$ and $w_2 = 2/7 = 0.2857$, i.e., $P_1 = [0.7143 \quad 0.2857]^T$.

(**c**) The solution in this case can readily be identified as point P_1 because as soon as the candidate solution point starts to leave P_1 towards P_2 (see the figure below), it suggests a buying policy that

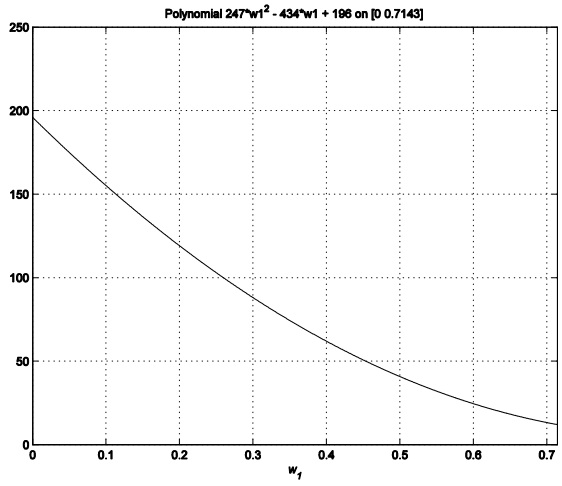
invests more money on the risky security 2, therefore increases the objective function.



An alternative and more quantitative way to explain why P_1 is the solution point is to evaluate polynomial $f(w_1, w_2) = 9w_1^2 - 42w_1w_2 + 196w_2^2$ along the line from P_2 to P_1. To do this, we substitute $w_2 = 1 - w_1$ into $f(w_1, w_2)$, which yields

$$\tilde{f}(w_1) = 9w_1^2 - 42w_1(1-w_1) + 196(1-w_1)^2 = 247w_1^2 - 434w_1$$

where w_1 varies from 0 to 0.7143. The plot of $\tilde{f}(w_1)$ over interval [0, 0.7143] is depicted in the figure below. It is observed that the minimum value of the function is achieved at the end point $w_1 = 0.7143$ which together with $w_2 = 1 - w_1 = 0.2857$ corresponds to point P_1.



(**d**) For comparison, we observe that

(i) when investing security 1 only, the expected return is 5 and risk is 9;

(ii) when investing security 2 only, the expected return is 12 and risk is 196; and

(iii) when investing in both securities 1 and 2 using the solution obtained in part (c), the expected

return is 7 and the risk is found to be $\sum_{i=1}^{2}\sum_{j=1}^{2} q_{ij} w_i w_j = 12.0204$, indicating a fairly moderate risk increase from the lowest possible risk while having the expected return increased from 5 (if buying security 1 only) to 7 (if using 71.43% of the money to buy security 1 and remaining 28.57% to purchase security 2).

(**e**) The feasible region in this case is also a portion of the line $w_1 + w_2 = 1$ in the first quadrant with the same endpoint $P_2 = [0\ 1]^T$, but the endpoint P_1 is now determined by the equations

$$w_1 + w_2 = 1$$
$$5w_1 + 12w_2 = \mu_* = 9$$

whose solution is given by $w_1 = 3/7 = 0.4286$ and $w_2 = 4/7 = 0.5714$, i.e.,

$$P_1 = [0.4286 \quad 0.5714]^T$$

With an argument similar to that given in the first case, we conclude that point P_1 represents the solution of the minimization problem. That is, the minimum expected return is increased almost twice from 5 to 9 if the investor uses 42.86% of the money to buy security 1 and the rest 57.14% to purchase security 2. In doing so, however, the investor takes a considerably higher risk

$$\sum_{i=1}^{2}\sum_{j=1}^{2} q_{ij} w_i w_j = 55.3673. \quad \blacksquare$$

Chapter 2 Basic Principles

2.9

(**a**) Let $\boldsymbol{H}$ be an $n \times n$ symmetric matrix whose ith diagonal element is zero, and $\boldsymbol{v}$ be an n-dimensional column vector whose ith element is one and other elements are all zero. Obviously, $\boldsymbol{v}^T \boldsymbol{H} \boldsymbol{v}$ picks the ith diagonal element which is zero, thus we have $\boldsymbol{v}^T \boldsymbol{H} \boldsymbol{v} = 0$. By definition, therefore, $\boldsymbol{H}$ cannot be positive or negative definite.

(**b**) Let $\boldsymbol{H}$ be an $n \times n$ symmetric matrix whose ith diagonal element is positive and jth element is negative. Let $\boldsymbol{v}$ be the vector defined in part (a) and $\boldsymbol{u}$ be an n-dimensional column vector whose jth element is one and other elements are all zero. Obviously we have $\boldsymbol{v}^T \boldsymbol{H} \boldsymbol{v} > 0$ and $\boldsymbol{u}^T \boldsymbol{H} \boldsymbol{u} < 0$. By definition, therefore, $\boldsymbol{H}$ is indefinite. $\blacksquare$

2.20

(**a**) Denote matrix $\boldsymbol{A}$ column-wise as

$$\boldsymbol{A} = \{a_{ij}, i = 1, 2, ..., n; j = 1, 2, ..., m\} = \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_m \end{bmatrix}$$

and compute the jth component of gradient $\nabla F(\boldsymbol{u})$:

$$\frac{\partial F}{\partial u_j} = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial x_i}{\partial u_j} = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} a_{ij} = \boldsymbol{a}_j^T \nabla f \qquad \text{(S2.1)}$$

Hence

$$\nabla F(\boldsymbol{u}) = \begin{bmatrix} \dfrac{\partial F}{\partial u_1} \\ \dfrac{\partial F}{\partial u_2} \\ \vdots \\ \dfrac{\partial F}{\partial u_m} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1^T \nabla f \\ \boldsymbol{a}_2^T \nabla f \\ \vdots \\ \boldsymbol{a}_m^T \nabla f \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \vdots \\ \boldsymbol{a}_m^T \end{bmatrix} \nabla f = \boldsymbol{A}^T \nabla f$$

(**b**) From Eq. (S2.1), it follows that

$$\frac{\partial^2 F}{\partial u_j \partial u_k} = \frac{\partial}{\partial u_k} \sum_{i=1}^{n} a_{ij} \frac{\partial f}{\partial x_i} = \sum_{i=1}^{n} a_{ij} \sum_{l=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_l} \frac{\partial x_l}{\partial u_k} = \sum_{i=1}^{n} a_{ij} \sum_{l=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_l} a_{lk} = \boldsymbol{a}_i^T \nabla^2 f \, \boldsymbol{a}_k .$$

Hence

$$\nabla^2 F = \boldsymbol{A}^T \nabla^2 f \, \boldsymbol{A} \tag{S2.2}$$

(**c**) Since $f(\boldsymbol{x})$ is smooth and convex, $\nabla^2 f$ is positive semidefinite and hence Eq. (S2.2) implies that $\nabla^2 F$ is positive semidefinite, thus $F(\boldsymbol{u})$ is convex with respect to $\boldsymbol{u}$.

(**d**) If $m \le n$, $\boldsymbol{A}$ is of full column rank, and $f(\boldsymbol{x})$ is strictly convex, then $\nabla^2 f$ is positive definite which in conjunction with (S2.2) implies that $\nabla^2 F$ is positive definite. Therefore, $F(\boldsymbol{u})$ is strictly convex with respect to $\boldsymbol{u}$. ■

Chapter 4 One-Dimensional Optimization

4.11

(**a**) With $\boldsymbol{x} = \boldsymbol{x}_k + \alpha \boldsymbol{d}_k$, the quadratic function can be expressed as

$$\begin{aligned} f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) &= \frac{1}{2} (\boldsymbol{x}_k + \alpha \boldsymbol{d}_k)^T \boldsymbol{Q} (\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) + \boldsymbol{b}^T (\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) + c \\ &= \frac{1}{2} \left(\boldsymbol{d}_k^T \boldsymbol{Q} \boldsymbol{d}_k \right) \alpha^2 + \boldsymbol{d}_k^T \left(\boldsymbol{Q} \boldsymbol{x}_k + \boldsymbol{b} \right) \alpha + \text{constant} \end{aligned}$$

(**b**) Using the expression from part (a), we compute the first-order derivative $\dfrac{d f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k)}{d\alpha}$ and set it to zero to obtain

$$\frac{d f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k)}{d\alpha} = \left(\boldsymbol{d}_k^T \boldsymbol{Q} \boldsymbol{d}_k \right) \alpha + \boldsymbol{d}_k^T \left(\boldsymbol{Q} \boldsymbol{x}_k + \boldsymbol{b} \right) = 0$$

which yields an α minimizing $f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k)$ as

$$\alpha_k = -\frac{\boldsymbol{d}_k^T \left(\boldsymbol{Q} \boldsymbol{x}_k + \boldsymbol{b} \right)}{\boldsymbol{d}_k^T \boldsymbol{Q} \boldsymbol{d}_k}$$

Note that $\boldsymbol{g}_k \triangleq \nabla f(\boldsymbol{x})\big|_{\boldsymbol{x}=\boldsymbol{x}_k} = \boldsymbol{Q}\boldsymbol{x}_k + \boldsymbol{b}$, the above expression becomes

$$\alpha_k = -\frac{\boldsymbol{d}_k^T \boldsymbol{g}_k}{\boldsymbol{d}_k^T \boldsymbol{Q} \boldsymbol{d}_k} \quad \blacksquare$$

Chapter 5 Basic Multidimensional Gradient Methods

5.5

(**a**) The objective is quadratic and convex, and can be expressed as $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{p}$ where

$$\boldsymbol{Q} = \begin{bmatrix} 10 & -9 \\ -9 & 8.15 \end{bmatrix} \text{ and } \boldsymbol{p} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Hence the exact solution can be readily obtained as

$$\boldsymbol{x}^* = -\boldsymbol{Q}^{-1}\boldsymbol{p} = \begin{bmatrix} -16.300000000000349 \\ -18.000000000000387 \end{bmatrix}.$$

The MATLAB code below (lines 21 to 35) implements Algorithm 5.1 where the line search step is performed using the formula in Eq. (5.5).

```
1    % Example: x0 = [1; 1];
2    % [xs,xs1,xs2,er1,er2,cpu1,cpu2] = prob5_5(x0,1480,11,1000,1000);
3    % Input:
4    % x0: initial point
5    % K1: number of iterations for the method in part (a)
6    % K2: number of iterations for the method in part (b)
7    % N1: number of runs for the method in part (a)
8    % N2: number of runs for the method in part (b)
9    % Output:
10   % xs: exact solution of the problem
11   % xs1: solution obtained using the method described in part (a)
12   % xs2: solution obtained using the method described in part (b)
13   % er1: 2-norm error of the solution from part (a)
14   % er2: 2-nprm error of the solution from part (b)
15   % cpu1: average CPUtime of the method from part (a)
16   % cpu2: average CPUtime of the method from part (b)
17   function [xs,xs1,xs2,er1,er2,cpu1,cpu2] = prob5_5(x0,K1,K2,N1,N2)
18   Q = [10 -9; -9 8.15];
19   p = [1; 0];
20   xs = -(Q\p);
21   xk = x0;
22   t0 = cputime;
23   for i = 1:N1
24       k1 = 0;
25     while k1 < K1
```

```
26        gk = Q*xk + p;
27        ak = (gk'*gk)/(gk'*Q*gk);
28        xk_new = xk - ak*gk;
29        xk = xk_new;
30        k1 = k1 + 1;
31      end
32    end
33    cpu1 = (cputime - t0)/N1;
34    xs1 = xk;
35    er1 = norm(xs1-xs);
36    t0 = cputime;
37    for i = 1:N2
38      k2 = 1;
39      x_old = x0;
40      g_old = Q*x_old + p;
41      a = (g_old'*g_old)/(g_old'*Q*g_old);
42      xk = x_old - a*g_old;
43      while k2 < K2
44        gk = Q*xk + p;
45        dlt = xk - x_old;
46        gam = gk - g_old;
47        ak = (dlt'*dlt)/(dlt'*gam);
48        x_old = xk;
49        g_old = gk;
50        xk = x_old - ak*g_old;
51        k2 = k2 + 1;
52      end
53    end
54    cpu2 = (cputime - t0)/N2;
55    xs2 = xk;
56    er2 = norm(xs2-xs);
```

With $\boldsymbol{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, it took the algorithm 1480 iterations to converge to the solution

$$\boldsymbol{x}_a^* = \begin{bmatrix} -16.299999999999585 \\ -17.999999999999538 \end{bmatrix}.$$

(**b**) The MATLAB code (lines 36 to 56) also implements Algorithm 5.1 where the line search step is performed using the Barzilai-Borwein formula in Eq. (5.15).

With $\boldsymbol{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, it took the algorithm 11 iterations to reach a solution

$$\boldsymbol{x}_b^* = \begin{bmatrix} -16.300000000000139 \\ -18.000000000000153 \end{bmatrix}.$$

(**c**) Note that the solutions from part (a) and (b) are of comparable accuracy. We run the above code 1000 times to compute average CPU time for the two methods. We normalized the CPU time required by the method in part (a) to 1 unit, the CPU time required by the method in part (b) was found to be 0.0172 units. ■

5.24 Let $\hat{f}(\boldsymbol{\delta})$ be the second-order approximation of $f(\boldsymbol{x})$ at $\boldsymbol{x}$, namely,

$$\hat{f}(\boldsymbol{\delta}) = f(\boldsymbol{x}) + \boldsymbol{\delta}^T \boldsymbol{g}(\boldsymbol{x}) + \tfrac{1}{2}\boldsymbol{\delta}^T \boldsymbol{H}(\boldsymbol{x})\boldsymbol{\delta}$$

If $\boldsymbol{H}(\boldsymbol{x})$ is positive definite, the minimizer of $\hat{f}(\boldsymbol{\delta})$, $\boldsymbol{\delta}^*$, can be found by solving

$$\nabla \hat{f}(\boldsymbol{\delta}) = \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{H}(\boldsymbol{x})\boldsymbol{\delta} = \mathbf{0}$$

which yields $\boldsymbol{\delta}^* = -\boldsymbol{H}(\boldsymbol{x})^{-1}\boldsymbol{g}(\boldsymbol{x})$. Therefore,

$$\inf_{\boldsymbol{\delta}} \hat{f}(\boldsymbol{\delta}) = \hat{f}(\boldsymbol{\delta}^*) = f(\boldsymbol{x}) - \tfrac{1}{2}\boldsymbol{g}(\boldsymbol{x})^T \boldsymbol{H}^{-1}(\boldsymbol{x})\boldsymbol{g}(\boldsymbol{x})$$

which implies that

$$f(\boldsymbol{x}) - \inf_{\boldsymbol{\delta}} \hat{f}(\boldsymbol{\delta}) = \tfrac{1}{2}\boldsymbol{g}(\boldsymbol{x})^T \boldsymbol{H}^{-1}(\boldsymbol{x})\boldsymbol{g}(\boldsymbol{x}) = \tfrac{1}{2}\lambda(\boldsymbol{x})^2 \qquad \text{(S5.1)}$$

where $\lambda(\boldsymbol{x}) = \left[\boldsymbol{g}(\boldsymbol{x})^T \boldsymbol{H}^{-1}(\boldsymbol{x})\boldsymbol{g}(\boldsymbol{x})\right]^{1/2}$ is the Newton decrement. If point x is near a solution of the problem of minimizing $f(\boldsymbol{x})$, then $\hat{f}(\boldsymbol{\delta})$ becomes a fairly accurate approximation of $f(\boldsymbol{x})$ and hence $f^* \triangleq \min f(\boldsymbol{x}) \approx \inf_{\boldsymbol{\delta}} \hat{f}(\boldsymbol{\delta})$ which in conjunction with (S5.1) gives

$$f(\boldsymbol{x}) - f^* \approx \tfrac{1}{2}\lambda(\boldsymbol{x})^2 .$$

This justifies the use of the Newton decrement in stopping criterion for the Newton method. ■

Chapter 6 Conjugate-Direction Methods

6.11

(**a**) Let rank($\boldsymbol{A}$) = r with $r \le \min\{m,n\}$. Below we deal with the general case of $r < \min\{m,n\}$ so that the SVD of $\boldsymbol{A}$ has the form

$$\boldsymbol{A} = \boldsymbol{U}\begin{bmatrix} \boldsymbol{S}_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\boldsymbol{V}^T \text{ with } \boldsymbol{S}_r = \text{diag}\{\sigma_1, \sigma_2, ..., \sigma_r\}$$

where $\boldsymbol{U} \in R^{m\times m}$ and $\boldsymbol{V} \in R^{n\times n}$ are orthogonal matrices and $\sigma_1 \ge \sigma_2 \ge ... \ge \sigma_r > 0$. The normal equation $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$ can now be written as

$$\begin{bmatrix} \boldsymbol{S}_r^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{S}_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\tilde{\boldsymbol{b}}$$

where $\tilde{\boldsymbol{x}} = \boldsymbol{V}^T \boldsymbol{x}$ and $\tilde{\boldsymbol{b}} = \boldsymbol{U}^T \boldsymbol{b}$. So if we denote

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \tilde{\boldsymbol{x}}_r \\ \boldsymbol{\phi} \end{bmatrix} \text{ and } \tilde{\boldsymbol{b}} = \begin{bmatrix} \tilde{\boldsymbol{b}}_r \\ \tilde{\boldsymbol{b}}_c \end{bmatrix}$$

with $\tilde{\boldsymbol{x}}_r \in R^{r\times 1}$, $\boldsymbol{\phi} \in R^{(n-r)\times 1}$, $\tilde{\boldsymbol{b}}_r \in R^{r\times 1}$ and $\tilde{\boldsymbol{b}}_c \in R^{(m-r)\times 1}$, then the above equation is reduced to $\boldsymbol{S}_r \tilde{\boldsymbol{x}}_r = \tilde{\boldsymbol{b}}_r$, and hence $\tilde{\boldsymbol{x}}_r = \boldsymbol{S}_r^{-1} \tilde{\boldsymbol{b}}_r$. Note that $\tilde{\boldsymbol{b}}_r$ can be written as $\tilde{\boldsymbol{b}}_r = \boldsymbol{U}_r^T \boldsymbol{b}$, where $\boldsymbol{U}_r$ consists of the first *r* columns of ***U***. Therefore, the solutions of the equation $\boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T \boldsymbol{b}$ always exist and are given by

$$\boldsymbol{x} = \boldsymbol{V} \begin{bmatrix} \boldsymbol{S}_r^{-1} \boldsymbol{U}_r^T \boldsymbol{b} \\ \boldsymbol{\phi} \end{bmatrix}$$

where $\boldsymbol{\phi} \in R^{(n-r)\times 1}$ is arbitrary. ■

(**b**) The objective function can be expressed as

$$f(\boldsymbol{x}) = \| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \|_2^2 = \boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{b} + \boldsymbol{b}^T \boldsymbol{b}$$

whose Hessian $2\boldsymbol{A}^T \boldsymbol{A}$ is constant and positive semidefinite and hence $f(\boldsymbol{x})$ is convex and quadratic. Therefore, point $\boldsymbol{x}^*$ is a minimizer if and only if $\boldsymbol{x}^*$ is a stationary point of $f(\boldsymbol{x})$, i.e., it satisfies $\nabla f(\boldsymbol{x}) = 2\boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{A}^T \boldsymbol{b} = \boldsymbol{0}$ which is precisely the normal equation

$$\boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T \boldsymbol{b}$$

(**c**) One approach to solving linear equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ with $\boldsymbol{A}^T \boldsymbol{A}$ nonsingular is to treat the problem as solving the optimization problem $\min \| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \|_2^2$ which, according to the result of part (b), is reduced to solve the normal equation $\boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T \boldsymbol{b}$. Now, in the case of $\boldsymbol{A}^T \boldsymbol{A}$ being nonsingular, Algorithm 6.6 is applicable because $\boldsymbol{A}^T \boldsymbol{A}$ in this case is symmetric and positive definite. Therefore, the generalization can be done by writing the normal equation as $\hat{\boldsymbol{A}}\boldsymbol{x} = \hat{\boldsymbol{b}}$ with $\hat{\boldsymbol{A}} = \boldsymbol{A}^T \boldsymbol{A}$ and $\hat{\boldsymbol{b}} = \boldsymbol{A}^T \boldsymbol{b}$, and applying Algorithm 6.1 to $\hat{\boldsymbol{A}}\boldsymbol{x} = \hat{\boldsymbol{b}}$.

Generalized Algorithm 6.6 Conjugate-gradient algorithm for the solution of the system of linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ with $\boldsymbol{A}^T \boldsymbol{A}$ nonsingular.

- Compute $\boldsymbol{A}^T \boldsymbol{A}$ and rename it as matrix ***A***. Compute $\boldsymbol{A}^T \boldsymbol{b}$ and rename it as ***b***.

- Then run Algorithm 6.6. ■

6.12 The MATLAB code implementing Algorithm 6.6 is listed below.

```
% Input:
%   {A,b} -- data for linear equation A and b
%   x0: initial point
%   epsi: termination tolerance
% Output:
%   xs: solution point
%   rs: norm of the equation error A*xs - b.
%   k: number of iterations at convergence
function [xs,rs,k] = conj_gradient_eq(A,b,x0,epsi)
k = 0;
xk = x0;
rk = b - A*xk;
dk = rk;
rk2_old = rk'*rk;
err = norm(rk);
while  err >= epsi
    adk = A*dk;
    ak = rk2_old/(dk'*adk);
    xk = xk + ak*dk;
    rk = rk - ak*adk;
    rk2 = rk'*rk;
    bk = rk2/rk2_old;
    rk2_old = rk2;
    dk = rk + bk*dk;
    err = norm(rk);
    k = k + 1;
end
format long
disp('Solution point:')
xs = xk;
disp('norm of residue at solution point:')
rs = err;
disp('Number of iterations performed:')
k
```

With

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 0 \\ -1 \\ 3 \\ 35 \end{bmatrix},$$

we compute

$$\boldsymbol{A}^T\boldsymbol{A} = \begin{bmatrix} 4 & 15 & 40 & 85 \\ 15 & 85 & 259 & 585 \\ 40 & 259 & 820 & 1885 \\ 85 & 585 & 1885 & 4369 \end{bmatrix}, \quad \boldsymbol{A}^T\boldsymbol{b} = \begin{bmatrix} 37 \\ 290 \\ 969 \\ 2284 \end{bmatrix}$$

and rename them as $\boldsymbol{A}$ and $\boldsymbol{b}$, respectively, and run the MATLAB code given above with initial point $\boldsymbol{x}_0$ = [–10 10 –10 10]^T and tolerance $\varepsilon = 10^{-6}$. It took the algorithm five iterations to converge to a solution

$$\boldsymbol{x}^* = \begin{bmatrix} 2.999999999911471 \\ -3.999999999862959 \\ -0.000000000131162 \\ 1.000000000039993 \end{bmatrix}.$$

For comparison, the exact solution of the equation is [3 –4 0 1]^T. ■

Chapter 7 Quasi-Newton Methods

7.16 With

$$\boldsymbol{S}_{k+1} = \boldsymbol{I} + \left(1 + \frac{\boldsymbol{\gamma}_k^T\boldsymbol{\gamma}_k}{\boldsymbol{\gamma}_k^T\boldsymbol{\delta}_k}\right)\frac{\boldsymbol{\delta}_k\boldsymbol{\delta}_k^T}{\boldsymbol{\gamma}_k^T\boldsymbol{\delta}_k} - \frac{\boldsymbol{\delta}_k\boldsymbol{\gamma}_k^T + \boldsymbol{\gamma}_k\boldsymbol{\delta}_k^T}{\boldsymbol{\gamma}_k^T\boldsymbol{\delta}_k},$$

the search direction $\boldsymbol{d}_{k+1} = -\boldsymbol{S}_{k+1}\boldsymbol{g}_{k+1}$ can be computed as follows:

$$\boldsymbol{d}_{k+1} = -\boldsymbol{S}_{k+1}\boldsymbol{g}_{k+1}$$

$$= -\boldsymbol{g}_{k+1} - \left(1 + \frac{\boldsymbol{\gamma}_k^T\boldsymbol{\gamma}_k}{\boldsymbol{\gamma}_k^T\boldsymbol{\delta}_k}\right)\frac{\boldsymbol{\delta}_k\boldsymbol{\delta}_k^T\boldsymbol{g}_{k+1}}{\boldsymbol{\gamma}_k^T\boldsymbol{\delta}_k} + \frac{\boldsymbol{\delta}_k\boldsymbol{\gamma}_k^T\boldsymbol{g}_{k+1} + \boldsymbol{\gamma}_k\boldsymbol{\delta}_k^T\boldsymbol{g}_{k+1}}{\boldsymbol{\gamma}_k^T\boldsymbol{\delta}_k}$$

$$= -\boldsymbol{g}_{k+1} - \left(1 + \frac{\boldsymbol{\gamma}_k^T\boldsymbol{\gamma}_k}{\eta_4}\right)\frac{\boldsymbol{\delta}_k\boldsymbol{\delta}_k^T\boldsymbol{g}_{k+1}}{\eta_4} + \frac{\boldsymbol{\delta}_k\boldsymbol{\gamma}_k^T\boldsymbol{g}_{k+1} + \boldsymbol{\gamma}_k\boldsymbol{\delta}_k^T\boldsymbol{g}_{k+1}}{\eta_4}$$

$$= -\boldsymbol{g}_{k+1} - \left(1 + \frac{\boldsymbol{\gamma}_k^T\boldsymbol{\gamma}_k}{\eta_4}\right)\eta_1\boldsymbol{\delta}_k + \frac{\boldsymbol{\delta}_k\boldsymbol{\gamma}_k^T\boldsymbol{g}_{k+1} + \boldsymbol{\gamma}_k\boldsymbol{\delta}_k^T\boldsymbol{g}_{k+1}}{\eta_4}$$

$$= -\boldsymbol{g}_{k+1} - \eta_3\boldsymbol{\delta}_k + \eta_2\boldsymbol{\delta}_k + \eta_1\boldsymbol{\gamma}_k$$

$$= -\boldsymbol{g}_{k+1} + \eta_1\boldsymbol{\gamma}_k + (\eta_2 - \eta_3)\boldsymbol{\delta}_k. \quad ■$$

Chapter 9 Applications of Unconstrained Optimization

9.1

(**a**) The gradient $\nabla\boldsymbol{D}$ defined by Eqs. (9.1a) and (9.1b) is invariant to a constant shift of its pixel values because each of Eqs. (9.1a) and (9.1b) computes difference between two pixel values that remains the same when all pixel values are shifted by a single constant *s*.

(**b**) Normalized gradient $\nabla \boldsymbol{D} / \| \nabla \boldsymbol{D} \|_2$ is invariant to multiplication of its pixel values by a single constant of c as long as c is positive because

$$\nabla c\boldsymbol{D} / \| \nabla c\boldsymbol{D} \|_2 = c\nabla \boldsymbol{D} / \left(| c | \cdot \| \nabla \boldsymbol{D} \|_2 \right) = \nabla \boldsymbol{D} / \| \nabla \boldsymbol{D} \|_2 . \quad \blacksquare$$

9.2 The HOG of an image is determined the pairs $\{\Theta, \| \nabla \boldsymbol{D} \|_2\}$, whew the Θ is calculated using Eq. (9.2b) and hence invariant to the illumination change $\hat{\boldsymbol{D}} = c\boldsymbol{D}$. In addition, the illumination change implies that $\| \nabla \hat{\boldsymbol{D}} \|_2 = c \| \nabla \boldsymbol{D} \|_2$. Since each component of the HOG , is a partial sum of the components of $\| \nabla \hat{\boldsymbol{D}} \|_2$ and hence is linearly proportional to the scaling factor c, we conclude that $\hat{\boldsymbol{x}} = c\boldsymbol{x}$. $\blacksquare$

9.3 Since the exponential function is monotonically increasing, Eq. (9.9) implies that

$$e^{\boldsymbol{\theta}_{i^*}^T \hat{\boldsymbol{x}}} = \max_{0 \le i \le K-1} e^{\boldsymbol{\theta}_i^T \hat{\boldsymbol{x}}}$$

which in conjunction with Eq. (9.8) leads to

$$P(l = i^* \mid \boldsymbol{x}, \Theta) = \max_{0 \le i \le K-1} P(l = i \mid \boldsymbol{x}, \Theta) \quad \blacksquare$$

9.12

(**a**) Given the objective function $f(\boldsymbol{x}) = \frac{1}{2} \sum_{i=1}^{m} \left(\| \boldsymbol{x} - \boldsymbol{s}_i \|_2 - r_i \right)^2$, we compute

$$\begin{aligned}
\frac{\partial f(\boldsymbol{x})}{\partial x_j} &= \sum_{i=1}^{m} \left(\| \boldsymbol{x} - \boldsymbol{s}_i \|_2 - r_i \right) \frac{\partial \left(\| \boldsymbol{x} - \boldsymbol{s}_i \|_2 - r_i \right)}{\partial x_j} \\
&= \sum_{i=1}^{m} \left(\| \boldsymbol{x} - \boldsymbol{s}_i \|_2 - r_i \right) \frac{(x_j - s_j^{(i)})}{\| \boldsymbol{x} - \boldsymbol{s}_i \|_2} \\
&= \sum_{i=1}^{m} \left(1 - \frac{r_i}{\| \boldsymbol{x} - \boldsymbol{s}_i \|_2} \right) (x_j - s_j^{(i)})
\end{aligned}$$

which gives

$$\nabla f(\boldsymbol{x}) = \sum_{i=1}^{m} \left(1 - \frac{r_i}{\| \boldsymbol{x} - \boldsymbol{s}_i \|_2} \right) (\boldsymbol{x} - \boldsymbol{s}_i)$$

and verifies Eq. (9.49).

(**b**) To compute the Hessian, we evaluate

$$\frac{\partial^2 f(\boldsymbol{x})}{\partial x_j \partial x_k} = \frac{\partial}{\partial x_k} \sum_{i=1}^{m} \left(1 - \frac{r_i}{\| \boldsymbol{x} - \boldsymbol{s}_i \|_2} \right) (x_j - s_j^{(i)}) .$$

If $k = j$, then the above expression leads to

$$\frac{\partial^2 f(\boldsymbol{x})}{\partial^2 x_j} = \frac{\partial}{\partial x_j}\sum_{i=1}^{m}\left(1-\frac{r_i}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2}\right)(x_j - s_j^{(i)})$$

$$= \sum_{i=1}^{m}\left(\frac{r_i}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2^3}\right)(x_j - s_j^{(i)})^2 + \sum_{i=1}^{m}\left(1-\frac{r_i}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2}\right)$$

$$= \sum_{i=1}^{m}\left(\frac{r_i}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2^3}\right)(x_j - s_j^{(i)})^2 + m - \sum_{i=1}^{m}\frac{r_i}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2}$$

$$= \sum_{i=1}^{m}\left(\frac{r_i}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2^3}\right)(x_j - s_j^{(i)})^2 + \tau$$

If $k \neq j$, then we have

$$\frac{\partial^2 f(\boldsymbol{x})}{\partial x_j \partial x_k} = \frac{\partial}{\partial x_k}\sum_{i=1}^{m}\left(1-\frac{r_i}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2}\right)(x_j - s_j^{(i)})$$

$$= \sum_{i=1}^{m}\left(\frac{r_i}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2^3}\right)(x_k - s_k^{(i)})(x_j - s_j^{(i)})$$

Based on these, we obtain

$$\nabla^2 f(\boldsymbol{x}) = \sum_{i=1}^{m}\left(\frac{r_i}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2^3}\right)(\boldsymbol{x}-\boldsymbol{s}_i)(\boldsymbol{x}-\boldsymbol{s}_i)^T + \tau\boldsymbol{I}$$

where $\boldsymbol{I}$ is the identity matrix. This verifies Eqs. (9.50a) and (9.50b).

(**c**) Given the objective function $f(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{m}\left(\|\boldsymbol{x}-\boldsymbol{s}_i\|_2 - \|\boldsymbol{x}\|_2 - r_i\right)^2$, we compute

$$\frac{\partial f(\boldsymbol{x})}{\partial x_j} = \sum_{i=1}^{m}\left(\|\boldsymbol{x}-\boldsymbol{s}_i\|_2 - \|\boldsymbol{x}\|_2 - r_i\right)\left(\frac{x_j - s_j^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_j}{\|\boldsymbol{x}\|_2}\right)$$

$$= \sum_{i=1}^{m} c_i\left(\frac{x_j - s_j^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_j}{\|\boldsymbol{x}\|_2}\right)$$

Therefore,

$$\nabla f(\boldsymbol{x}) = \sum_{i=1}^{m} c_i\left(\frac{\boldsymbol{x}-\boldsymbol{s}_i}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}\right) = \sum_{i=1}^{m} c_i\left(\boldsymbol{q}_i - \tilde{\boldsymbol{x}}\right)$$

and this verifies Eq. (9.56a).

(**d**) To compute the Hessian, we evaluate

$$\frac{\partial^2 f(\boldsymbol{x})}{\partial x_j \partial x_k} = \frac{\partial}{\partial x_k}\sum_{i=1}^{m}\left(\|\boldsymbol{x}-\boldsymbol{s}_i\|_2 - \|\boldsymbol{x}\|_2 - r_i\right)\left(\frac{x_j - s_j^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_j}{\|\boldsymbol{x}\|_2}\right)$$

If $k = j$, then the above expression leads to

$$\frac{\partial^2 f(\boldsymbol{x})}{\partial^2 x_j} = \frac{\partial}{\partial x_j}\sum_{i=1}^{m}\left(\|\boldsymbol{x}-\boldsymbol{s}_i\|_2 - \|\boldsymbol{x}\|_2 - r_i\right)\left(\frac{x_j - s_j^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_j}{\|\boldsymbol{x}\|_2}\right)$$

$$= \sum_{i=1}^{m}\left(\frac{x_j - s_j^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_j}{\|\boldsymbol{x}\|_2}\right)^2 + \sum_{i=1}^{m}\left(\|\boldsymbol{x}-\boldsymbol{s}_i\|_2 - \|\boldsymbol{x}\|_2 - r_i\right)\left(\frac{1}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{\left(x_j - s_j^{(i)}\right)^2}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2^3} - \frac{1}{\|\boldsymbol{x}\|_2} + \frac{x_j^2}{\|\boldsymbol{x}\|_2^3}\right)$$

$$= \sum_{i=1}^{m}\left(\frac{x_j - s_j^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_j}{\|\boldsymbol{x}\|_2}\right)^2 + \sum_{i=1}^{m} c_i\left(\frac{1}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{\left(x_j - s_j^{(i)}\right)^2}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2^3} - \frac{1}{\|\boldsymbol{x}\|_2} + \frac{x_j^2}{\|\boldsymbol{x}\|_2^3}\right)$$

$$= \sum_{i=1}^{m}\left(\frac{x_j - s_j^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_j}{\|\boldsymbol{x}\|_2}\right)^2 + \frac{1}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2}\sum_{i=1}^{m} c_i\left(1 - \frac{\left(x_j - s_j^{(i)}\right)^2}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2^2}\right) - \frac{1}{\|\boldsymbol{x}\|_2}\sum_{i=1}^{m} c_i\left(1 - \frac{x_j^2}{\|\boldsymbol{x}\|_2^2}\right)$$

If $k \neq j$, then we have

$$\frac{\partial^2 f(\boldsymbol{x})}{\partial x_j \partial x_k} = \frac{\partial}{\partial x_k}\sum_{i=1}^{m}\left(\|\boldsymbol{x}-\boldsymbol{s}_i\|_2 - \|\boldsymbol{x}\|_2 - r_i\right)\left(\frac{x_j - s_j^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_j}{\|\boldsymbol{x}\|_2}\right)$$

$$= \sum_{i=1}^{m}\left(\frac{x_k - s_k^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_k}{\|\boldsymbol{x}\|_2}\right)\left(\frac{x_j - s_j^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_j}{\|\boldsymbol{x}\|_2}\right) + \sum_{i=1}^{m}\left(\|\boldsymbol{x}-\boldsymbol{s}_i\|_2 - \|\boldsymbol{x}\|_2 - r_i\right)\left(-\frac{\left(x_k - s_k^{(i)}\right)\left(x_j - s_j^{(i)}\right)}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2^3} + \frac{x_k x_j}{\|\boldsymbol{x}\|_2^3}\right)$$

$$= \sum_{i=1}^{m}\left(\frac{x_k - s_k^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_k}{\|\boldsymbol{x}\|_2}\right)\left(\frac{x_j - s_j^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_j}{\|\boldsymbol{x}\|_2}\right) + \sum_{i=1}^{m} c_i\left(-\frac{\left(x_k - s_k^{(i)}\right)\left(x_j - s_j^{(i)}\right)}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2^3} + \frac{x_k x_j}{\|\boldsymbol{x}\|_2^3}\right)$$

$$= \sum_{i=1}^{m}\left(\frac{x_k - s_k^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_k}{\|\boldsymbol{x}\|_2}\right)\left(\frac{x_j - s_j^{(i)}}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2} - \frac{x_j}{\|\boldsymbol{x}\|_2}\right) - \frac{1}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2}\sum_{i=1}^{m} c_i\frac{\left(x_k - s_k^{(i)}\right)\left(x_j - s_j^{(i)}\right)}{\|\boldsymbol{x}-\boldsymbol{s}_i\|_2^2} + \frac{1}{\|\boldsymbol{x}\|_2}\sum_{i=1}^{m} c_i\frac{x_k x_j}{\|\boldsymbol{x}\|_2^2}$$

Summarizing the above expressions, we obtain

$$\nabla^2 f(\boldsymbol{x}) = \sum_{i=1}^{m}\left[(\boldsymbol{q}_i - \tilde{\boldsymbol{x}})(\boldsymbol{q}_i - \tilde{\boldsymbol{x}})^T + c_i\left(\boldsymbol{Q}_{1i} - \boldsymbol{Q}_2\right)\right]$$

and this verifies Eq. (9.56b). ■

Chapter 10 Fundamentals of Constrained Optimization

10.24

(**a**) The Lagrangian of the problem is given by

$$L(\boldsymbol{x},\boldsymbol{\lambda}) = \boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}),$$

hence the Lagrange dual function is defined as

$$q(\boldsymbol{\lambda}) = \inf_{\boldsymbol{x}} L(\boldsymbol{x},\boldsymbol{\lambda}) = \inf_{\boldsymbol{x}}\left\{\boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})\right\}$$

The minimizer for the convex problem $\inf_{\boldsymbol{x}}\left\{\boldsymbol{x}^T\boldsymbol{x}+\boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})\right\}$ is given by $\boldsymbol{x}^*=-\frac{1}{2}\boldsymbol{A}^T\boldsymbol{\lambda}$ and hence the Lagrange dual function is obtained as

$$q(\boldsymbol{\lambda})=-\tfrac{1}{4}\boldsymbol{\lambda}^T\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\lambda}-\boldsymbol{\lambda}^T\boldsymbol{b}$$

(**b**) The Lagrangian of the problem is given by

$$L(\boldsymbol{x},\boldsymbol{\mu})=\tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x}+\boldsymbol{p}^T\boldsymbol{x}+c+\sum_{i=1}^{q}\mu_i\left(\tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{H}_i\boldsymbol{x}+\boldsymbol{p}_i^T\boldsymbol{x}+c_i\right)$$

hence the Lagrange dual function is defined as

$$q(\boldsymbol{\mu})=\inf_{\boldsymbol{x}}\left\{\tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x}+\boldsymbol{p}^T\boldsymbol{x}+c+\sum_{i=1}^{q}\mu_i\left(\tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{H}_i\boldsymbol{x}+\boldsymbol{p}_i^T\boldsymbol{x}+c_i\right)\right\}$$

$$=\inf_{\boldsymbol{x}}\left\{\tfrac{1}{2}\boldsymbol{x}^T\left(\boldsymbol{H}+\textstyle\sum_{i=1}^{q}\mu_i\boldsymbol{H}_i\right)\boldsymbol{x}+\left(\boldsymbol{p}+\textstyle\sum_{i=1}^{q}\mu_i\boldsymbol{p}_i\right)^T\boldsymbol{x}+c+\sum_{i=1}^{q}\mu_i c_i\right\}$$

The above infimum is achieved at

$$\boldsymbol{x}^*=-\left(\boldsymbol{H}+\textstyle\sum_{i=1}^{q}\mu_i\boldsymbol{H}_i\right)^{-1}\left(\boldsymbol{p}+\textstyle\sum_{i=1}^{q}\mu_i\boldsymbol{p}_i\right)$$

and hence the Lagrange dual function is obtained as

$$q(\boldsymbol{\mu})=-\left(\boldsymbol{p}+\textstyle\sum_{i=1}^{q}\mu_i\boldsymbol{p}_i\right)^T\left(\boldsymbol{H}+\textstyle\sum_{i=1}^{q}\mu_i\boldsymbol{H}_i\right)^{-1}\left(\boldsymbol{p}+\textstyle\sum_{i=1}^{q}\mu_i\boldsymbol{p}_i\right)+\sum_{i=1}^{q}\mu_i c_i+c \quad \blacksquare$$

Chapter 13 Quadratic, Semidefinite, and Second-Order Cone Programming

13.28

(**a**) The problem at hand is equivalent to

$$\begin{aligned}&\text{minimize} && \eta\\ &\text{subject to:} && \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2\le\eta\\ &&& \|\boldsymbol{B}\boldsymbol{x}\|_2\le\delta\end{aligned}$$

where η is an auxiliary scalar to be minimized. The above problem can in turn be expressed as

$$\begin{aligned}&\text{minimize} && \boldsymbol{b}_1^T\hat{\boldsymbol{x}}\\ &\text{subject to:} && \|\boldsymbol{A}_1^T\hat{\boldsymbol{x}}+\boldsymbol{c}_1\|_2\le\boldsymbol{b}_1^T\hat{\boldsymbol{x}}+d_1\\ &&& \|\boldsymbol{A}_2^T\hat{\boldsymbol{x}}+\boldsymbol{c}_2\|_2\le\boldsymbol{b}_2^T\hat{\boldsymbol{x}}+d_2\end{aligned}$$

where

$$\hat{\boldsymbol{x}}=\begin{bmatrix}\eta\\ \boldsymbol{x}\end{bmatrix},\ \boldsymbol{A}_1=\begin{bmatrix}\boldsymbol{0} & \boldsymbol{A}\end{bmatrix}^T,\ \boldsymbol{b}_1=\begin{bmatrix}1\\ \boldsymbol{0}\end{bmatrix},\ \boldsymbol{c}_1=-\boldsymbol{b},\ d_1=0;\ \boldsymbol{A}_2=\begin{bmatrix}\boldsymbol{0} & \boldsymbol{B}\end{bmatrix}^T,\ \boldsymbol{b}_2=\boldsymbol{0},\ \boldsymbol{c}_2=\boldsymbol{0},\ d_2=\delta,$$

which is obviously an SOCP problem.

(**b**) With $\boldsymbol{A}=\begin{bmatrix}2&0\\0&1\\0&0\end{bmatrix}$, $\boldsymbol{b}=\begin{bmatrix}4\\2\\3\end{bmatrix}$, $\boldsymbol{B}=\begin{bmatrix}1&0\\0&1\end{bmatrix}$, and $\delta=0.1$, we construct

$$\hat{\boldsymbol{A}}_1=\begin{bmatrix}1&0&0&0\\0&2&0&0\\0&0&1&0\end{bmatrix},\hat{\boldsymbol{c}}_1=\begin{bmatrix}0\\-4\\-2\\-3\end{bmatrix},\ \hat{\boldsymbol{A}}_2=\begin{bmatrix}0&0&0\\0&1&0\\0&0&1\end{bmatrix},\ \text{and}\ \hat{\boldsymbol{c}}_2=\begin{bmatrix}0.1\\0\\0\end{bmatrix}$$

Hence the data required in Eqs. (13.112) and (13.113) are found to be

$$\boldsymbol{A}=\begin{bmatrix}\hat{\boldsymbol{A}}_1 & \hat{\boldsymbol{A}}_2\end{bmatrix}=\begin{bmatrix}1&0&0&0&0&0&0\\0&2&0&0&0&1&0\\0&0&1&0&0&0&1\end{bmatrix},\ \boldsymbol{b}=\begin{bmatrix}1\\0\\0\end{bmatrix},\quad\text{and}\quad \boldsymbol{c}=\begin{bmatrix}0\\-4\\-2\\-3\\0.1\\0\\0\end{bmatrix}.$$

It can be readily verified that the initial set

$$\boldsymbol{x}_0=\begin{bmatrix}1\\0\\0\\0\\1\\0\\0\end{bmatrix},\ \boldsymbol{s}_0=\begin{bmatrix}6\\-4\\-2\\-3\\0.1\\0\\0\end{bmatrix},\ \text{and}\ \ \boldsymbol{y}_0=\begin{bmatrix}6\\0\\0\end{bmatrix}$$

are strictly feasible. With these, $\sigma=10^{-5}$, and $\varepsilon=10^{-6}$, it took the algorithm 14 iterations to converge to the solution (of the dual problem)

$$\boldsymbol{y}^*=\begin{bmatrix}5.233443506912794\\0.096795992130502\\0.025110470566740\end{bmatrix}$$

which gives a solution of the original LSQI problem as

$$\boldsymbol{x}^*=\begin{bmatrix}0.096795992130502\\0.025110470566740\end{bmatrix}.$$

The minimized objective function was found to be

$$\|\boldsymbol{A}\boldsymbol{x}^*-\boldsymbol{b}\|_2=5.233443477840112\,.$$

Note that the solution is feasible as it satisfies the constraint

$$\|\boldsymbol{B}\boldsymbol{x}^*\|_2=0.099999999123056<0.1\,.$$

Listed below is the MATLAB code which implements Algorithm 17.7 applying to Prob. 13.8(b). Note that the code requires two additional MATLAB functions, **find_ax.m** and **find_ay.m**, which are also listed below.

```
% Example: load data13_28a (or, load data13_28b)
% [xs,k] = prob13_28(A,b,c,x0,s0,y0,q,1e-5,1e-6);
% Input
% A,b,c: data required by formulation (13.112) and (13.113).
% x0, s0,y0: strictly feasible initial points.
% q: number of cones.
% sig: parameter \sigma required inn Eq. (13.13.116c);
% epsi: convergence tolerance.
% Output
% xs: solution of the dual problem y*.
% k: number of iterations used.
% Note: the code requires two additional functions find_ax.m and find_ay.m.
function [xs,k] = prob13_28(A,b,c,x0,s0,y0,q,sig,epsi)
% Data preparation
format long
x = x0;
s = s0;
y = y0;
gap = x'*s;
mu = gap/q;
e = ones(7,1);
k = 0;
% SOCP iterations
 while mu >=  epsi
   % Solve Eq. (13.116) for {dx,ds,dy}
   z1 = x(1:4);
   z2 = x(5:7);
   Z1 = [z1'; z1(2:4) z1(1)*eye(3)];
   Z2 = [z2'; z2(2:3) z2(1)*eye(2)];
   X = zeros(7,7);
   X(1:4,1:4) = Z1;
   X(5:7,5:7) = Z2;
   z1 = s(1:4);
   z2 = s(5:7);
   Z1 = [z1'; z1(2:4) z1(1)*eye(3)];
   Z2 = [z2'; z2(2:3) z2(1)*eye(2)];
   S = zeros(7,7);
   S(1:4,1:4) = Z1;
   S(5:7,5:7) = Z2;
   M = zeros(17,17);
   M(1:3,1:7) = A;
   M(4:10,15:17) = -A';
   M(4:10,8:14) = eye(7);
```

```
    M(11:17,1:14) = [S X];
    bw = [b-A*x; c-s+A'*y; sig*mu*e-X*s];
    delt = M\bw;
    dx = delt(1:7);
    ds = delt(8:14);
    dy = delt(15:17);
    % Perform line search using Eq. (13.117)
    a1 = find_ax(x,dx);
    a2 = find_ax(s,ds);
    a3 = find_ay(A,c,y,dy);
    a = 0.75*min([a1 a2 a3]);
    % Update {x,y,s}
    x = x + a*dx;
    s = s + a*ds;
    y = y + a*dy;
    k = k + 1;
    % Evaluate duality gap
    gap = x'*s
    mu = gap/q;
 end
xs = y;
disp('solution point:')
xs
disp('minimized objective:')
norm([2 0; 0 1; 0 0]*xs(2:3)-[4; 2; 3])
disp('norm of xs to check solution feasibility:')
norm(xs(2:3))
format short
```

```
function a = find_ax(x,dx)
x = x(:);
dx = dx(:);
x1 = x(1:4);
x2 = x(5:7);
dx1 = dx(1:4);
dx2 = dx(5:7);
a1 = [1 zeros(1,1000)];
a2 = a1;
da = 0.001;
for i = 1:1000
    dai = i*da;
    xw1 = x1 + dai*dx1;
    a1(i+1) = xw1(1) - norm(xw1(2:4));
    xw2 = x2 + dai*dx2;
    a2(i+1) = xw2(1) - norm(xw2(2:3));
end
```

```
18    ind1 = find(a1 > 0);
19    ind2 = find(a2 > 0);
20    ind= min([ind1(end) ind2(end)]) - 1;
21    a = da*ind;

1     function a = find_ay(A,c,y,dy)
2     s = c + A'*y;
3     ds = A'*dy;
4     a = find_ax(s,ds);
```

(**c**) With δ changed from 0.1 to 1, there are two items in data set required by the algorithm that are affected:

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -4 \\ -2 \\ -3 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \text{ and } \boldsymbol{s}_0 = \begin{bmatrix} 6 \\ -4 \\ -2 \\ -3 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

With these changes, the MATLAB code given above applied to solve the problem. With 14 iterations, the algorithm converges to the solution

$$\boldsymbol{y}^* = \begin{bmatrix} 4.030379185505437 \\ 0.933344789412242 \\ 0.358981163143590 \end{bmatrix}$$

which yields a solution for the original LSQI problem as

$$\boldsymbol{x}^* = \begin{bmatrix} 0.933344789412242 \\ 0.358981163143590 \end{bmatrix}.$$

As expected, the minimized objective function was reduced to

$$\| \boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b} \|_2 = 4.030379160328916 .$$

The solution is also feasible as it satisfies the constraint

$$\| \boldsymbol{B}\boldsymbol{x}^* \|_2 = 0.999999985707454 < 1. \quad \blacksquare$$

Chapter 14 Algorithms for General Convex Problems

14.1

(a) First, note that $\partial\left(\|\boldsymbol{x}\|_1\right) = \sum_{i=1}^{n} \partial\left(|\, x_i \,|\right)$. For each *i*, it follows from Example 14.1 that

$$\partial |x_i| = \begin{cases} \boldsymbol{e}_i & \text{for } x_i > 0 \\ g\boldsymbol{e}_i \text{ with any } g \in [-1, 1] & \text{for } x_i = 0 \\ -\boldsymbol{e}_i & \text{for } x_i < 0 \end{cases}$$

where $\boldsymbol{e}_i$ is the ith canonical unit vector (whose its ith entry is one, and zero elsewhere). Hence

$$\partial\left(\|\boldsymbol{x}\|_1\right) = \sum_{i=1}^{n} \partial(|x_i|) = \sum_{i \in I_+(\boldsymbol{x})} \boldsymbol{e}_i - \sum_{i \in I_-(\boldsymbol{x})} \boldsymbol{e}_i + \sum_{i \in I_0(\boldsymbol{x})} [-\boldsymbol{e}_i, \boldsymbol{e}_i] \qquad \text{(S14.1)}$$

where $I_+(\boldsymbol{x}) = \{i : x_i > 0\}$, $I_-(\boldsymbol{x}) = \{i : x_i < 0\}$, and $I_0(\boldsymbol{x}) = \{i : x_i = 0\}$. The notation $[-\boldsymbol{e}_i, \boldsymbol{e}_i]$ in (S14.1) is understood as the set of all vectors of form $g\boldsymbol{e}_i$ for some scalar g in [–1, 1].

(b) We compute the subdifferential $\partial\left(\|\boldsymbol{x}\|_2\right)$ for two cases:

(i) If $\boldsymbol{x} \neq \boldsymbol{0}$, then $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2$ is differentiable and $\partial\left(\|\boldsymbol{x}\|_2\right) = \boldsymbol{x} / \|\boldsymbol{x}\|_2$.

(ii) If $\boldsymbol{x} = \boldsymbol{0}$, then $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2$ is nondifferentiable. To compute its subdifferential, note the well-known fact, which can be shown using Schwartz inequality, that

$$\|\boldsymbol{x}\|_2 = \max_{\|\boldsymbol{g}\|_2 \le 1} (\boldsymbol{g}^T \boldsymbol{x})$$

Hence for any $\boldsymbol{x}$ and $\boldsymbol{g}$ with $\|\boldsymbol{g}\|_2 \le 1$, we have $\|\boldsymbol{x}\|_2 \ge \boldsymbol{g}^T \boldsymbol{x}$, i.e., $\|\boldsymbol{x}\|_2 \ge \|\boldsymbol{0}\|_2 + \boldsymbol{g}^T(\boldsymbol{x} - \boldsymbol{0})$. Conversely, if there exists a vector $\boldsymbol{g}$ with $\|\boldsymbol{x}\|_2 \ge \boldsymbol{g}^T \boldsymbol{x}$ for any $\boldsymbol{x}$, then it is immediate that $\|\boldsymbol{g}\|_2 \le 1$. Therefore at $\boldsymbol{x} = \boldsymbol{0}$, we have $\partial\left(\|\boldsymbol{x}\|_2\right) = \{\boldsymbol{g} : \|\boldsymbol{g}\| \le 1\}$. ■

(c) The problem can be readily treated by expressing the function $h(\boldsymbol{x}) = \sum_{i=1}^{m} |\boldsymbol{a}_i^T \boldsymbol{x} - b_i|$ as a compound function of $h(\boldsymbol{x}) = f(\boldsymbol{y})$ with $f(\boldsymbol{y}) = \|\boldsymbol{y}\|_1$ and $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ where $\boldsymbol{A}$ is an m by n matrix with $\boldsymbol{a}_i^T$ as its ith row, and $\boldsymbol{b}$ is an m by 1 vector with b_i as its ith entry. Now we can use the result from parts (d) and (a), namely, $h(\boldsymbol{x}) = f(\boldsymbol{y})$ with $f(\boldsymbol{y}) = \|\boldsymbol{y}\|_1$ and $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ to obtain

$$\partial h(\boldsymbol{x}) = \sum_{i \in I_+(\boldsymbol{x})} \boldsymbol{a}_i - \sum_{i \in I_-(\boldsymbol{x})} \boldsymbol{a}_i + \sum_{i \in I_0(\boldsymbol{x})} \text{any } \boldsymbol{g}_i \in [-\boldsymbol{a}_i, \boldsymbol{a}_i]$$

where

$$I_+(\boldsymbol{x}) = \{i : \boldsymbol{a}_i^T \boldsymbol{x} - b_i > 0\},\ I_-(\boldsymbol{x}) = \{i : \boldsymbol{a}_i^T \boldsymbol{x} - b_i < 0\},\ I_0(\boldsymbol{x}) = \{i : \boldsymbol{a}_i^T \boldsymbol{x} - b_i = 0\}$$

and $\boldsymbol{g}_i \in [-\boldsymbol{a}_i, \boldsymbol{a}_i]$ is understood as $\boldsymbol{g}_i = t\boldsymbol{a}_i$ for some scalar t between –1 and 1. ■

(d) Note that $h(\boldsymbol{x}) = f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b})$ is a composite function of $f(\boldsymbol{y})$ with $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$. Since $f(\boldsymbol{y})$ is

convex, $h(\boldsymbol{x})$ is also convex. Hence we have

$$h(\tilde{\boldsymbol{x}}) \geq h(\boldsymbol{x}) + \partial^T h(\boldsymbol{x})(\tilde{\boldsymbol{x}} - \boldsymbol{x})$$

and

$$f(\tilde{\boldsymbol{y}}) \geq f(\boldsymbol{y}) + \partial^T f(\boldsymbol{y})(\tilde{\boldsymbol{y}} - \boldsymbol{y})$$

which, with $\tilde{\boldsymbol{y}} = \boldsymbol{A}\tilde{\boldsymbol{x}} + \boldsymbol{b}, \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ and hence $\tilde{\boldsymbol{y}} - \boldsymbol{y} = \boldsymbol{A}(\tilde{\boldsymbol{x}} - \boldsymbol{x})$, becomes

$$f(\boldsymbol{A}\tilde{\boldsymbol{x}} + \boldsymbol{b}) \geq f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) + \partial^T f(\boldsymbol{y})\boldsymbol{A}(\tilde{\boldsymbol{x}} - \boldsymbol{x})$$

This, by definition of function *h*, can be written as

$$h(\tilde{\boldsymbol{x}}) \geq h(\boldsymbol{x}) + \partial^T f(\boldsymbol{y})\boldsymbol{A}(\tilde{\boldsymbol{x}} - \boldsymbol{x})$$

where the second term on the right-hand side can be expressed as

$$\partial^T f(\boldsymbol{y})\boldsymbol{A}(\tilde{\boldsymbol{x}} - \boldsymbol{x}) = \left(\boldsymbol{A}^T \partial f(\boldsymbol{y})\right)^T (\tilde{\boldsymbol{x}} - \boldsymbol{x})$$

Therefore, we obtain

$$h(\tilde{\boldsymbol{x}}) \geq h(\boldsymbol{x}) + \left(\boldsymbol{A}^T \partial f(\boldsymbol{y})\right)^T (\tilde{\boldsymbol{x}} - \boldsymbol{x})$$

which implies that

$$\partial h(\boldsymbol{x}) = \boldsymbol{A}^T \partial f(\boldsymbol{y}) \quad \text{with} \quad \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \qquad \blacksquare$$

14.2 By definition any subgradients of $f(\boldsymbol{x})$ at $\boldsymbol{x}$ and $\boldsymbol{y}$, namely $\boldsymbol{g}_x$ and $\boldsymbol{g}_y$, obey

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{g}_x^T(\boldsymbol{y} - \boldsymbol{x})$$

and

$$f(\boldsymbol{x}) \geq f(\boldsymbol{y}) + \boldsymbol{g}_y^T(\boldsymbol{x} - \boldsymbol{y})$$

By adding up the two inequalities, we obtain

$$f(\boldsymbol{y}) + f(\boldsymbol{x}) \geq f(\boldsymbol{x}) + f(\boldsymbol{y}) + \boldsymbol{g}_x^T(\boldsymbol{y} - \boldsymbol{x}) + \boldsymbol{g}_y^T(\boldsymbol{x} - \boldsymbol{y})$$

This yields

$$\boldsymbol{g}_x^T(\boldsymbol{y} - \boldsymbol{x}) + \boldsymbol{g}_y^T(\boldsymbol{x} - \boldsymbol{y}) \leq 0$$

which obviously implies that

$$\left(\boldsymbol{g}_x - \boldsymbol{g}_y\right)^T \left(\boldsymbol{x} - \boldsymbol{y}\right) \geq 0 \qquad \blacksquare$$

14.3 Since $f(\boldsymbol{x})$ is convex, we have

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \partial f(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x}) \qquad \text{(S14.2)}$$

Recall *epigraph* is defined by

$$\text{epi}(f) = \{(\boldsymbol{y},t) : \boldsymbol{y} \in \text{dom}(f) \text{ and } t \geq f(\boldsymbol{y})\}$$

hence for any $(\boldsymbol{y},t) \in \text{epi}(f)$ we have $t \geq f(\boldsymbol{y})$ which in conjunction with (S14.2) implies that

$$t \geq f(\boldsymbol{x}) + \partial f(\boldsymbol{x})^T(\boldsymbol{y}-\boldsymbol{x}) \quad \text{i.e.} \quad \partial f(\boldsymbol{x})^T(\boldsymbol{y}-\boldsymbol{x}) - (t - f(\boldsymbol{x})) \leq 0$$

which immediately yields

$$\begin{bmatrix} \partial f(\boldsymbol{x}) \\ -1 \end{bmatrix}^T \left(\begin{bmatrix} \boldsymbol{y} \\ t \end{bmatrix} - \begin{bmatrix} \boldsymbol{x} \\ f(\boldsymbol{x}) \end{bmatrix} \right) \leq 0 \quad \forall\, (\boldsymbol{y},t) \in \text{epi}(f) \qquad \blacksquare$$

14.4

(a) With $C = \{x : a \leq x \leq b\}$, by inspection it is immediate that

$$P_C(z) = \begin{cases} a & \text{if } z \leq a \\ z & \text{if } a < z < b \\ b & \text{if } z \geq b \end{cases}$$

(**b**) By definition the problem at hand can be formulated as the solution of the convex problem

$$\begin{aligned} &\text{minimize} \quad \left\| \boldsymbol{x} - \begin{bmatrix} 2 \\ 1 \end{bmatrix} \right\|_2 \\ &\text{subject to:} \quad \frac{(x_1-4)^2}{9} + \frac{(x_2-5)^2}{4} \leq 1 \end{aligned}$$

Running the code below using **CVX**

```
z = [2 1]';
cvx_begin
  variable x(2,1)
  minimize(norm(x - z))
  subject to
  (x(1)-4)^2/9 + (x(2)-5)^2/4 <= 1
cvx_end
```

yields $\boldsymbol{x}^* = \begin{bmatrix} 2.70250403 \\ 3.19673085 \end{bmatrix}$. $\blacksquare$

14.5

(**a**) By definition the problem at hand can be formulated as the solution of the convex problem

$$\begin{aligned} &\text{minimize} \quad \|\boldsymbol{x} - \boldsymbol{z}\|_2 \\ &\text{subject to:} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \end{aligned}$$

(**b**) The optimization problem in part (a) is equivalent to

$$\begin{aligned} &\text{minimize} \quad \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{z}\|_2^2 \\ &\text{subject to:} \quad \boldsymbol{A}\boldsymbol{x}=\boldsymbol{b} \end{aligned}$$

whose KKT conditions are given by

$$\boldsymbol{x}-\boldsymbol{z}+\boldsymbol{A}^T\boldsymbol{\lambda}=\boldsymbol{0} \text{ and } \boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$$

Solving these equations for $\boldsymbol{x}$, we obtain

$$\boldsymbol{x}^*=\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{b}+(\boldsymbol{I}-\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A})\boldsymbol{z}$$

Note that the first term of the projection is the least-square solution of $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$, and the second term of the projection is the orthogonal projection of $\boldsymbol{z}$ onto the null space of $\boldsymbol{A}$. ■

(**c**) By applying the formula obtained from part (b) to the given data, we obtain the projection as

$$\boldsymbol{x}^*=\begin{bmatrix}1.42857143\\0.14285714\\0.71428571\end{bmatrix} \qquad \blacksquare$$

14.6

Note: there are two typos in the problem statement: (1) in part (a), $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$ should be $\boldsymbol{A}\boldsymbol{x}\le\boldsymbol{b}$; (2) in part (b), $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$ should be $\boldsymbol{A}\boldsymbol{x}\le\boldsymbol{b}$.

(a) Given a point $\boldsymbol{z}$, the problem of finding $P_C(\boldsymbol{z})$ can be formulated as the constrained convex problem

$$\begin{aligned} &\text{minimize} \quad \|\boldsymbol{x}-\boldsymbol{z}\|_2 \\ &\text{subject to:} \quad \boldsymbol{A}\boldsymbol{x}\le\boldsymbol{b} \end{aligned}$$

(b) Using **CVX** with the data given above, we obtain

$$P_C(\boldsymbol{z})=\begin{bmatrix}1.6\\0.2\\2\\2\end{bmatrix}$$

CVX code

```
A = [-1 0 0 0; 0 -1 0 0; 1 2 0 0; 0 0 0 -1; 0 0 -1 -1; 0 0 1 2];
b = [0 0 2 -2 -3 6]';
z = [2 1 4 3]';
cvx_begin
  variable x(4,1)
  minimize(norm(x-z))
  subject to
```

```
  A*x <= b;
cvx_end
```

■

14.7 The first quadratic function is convex with Lipschitz continuous gradient with $L = 4$ because its Hessian $\boldsymbol{H}_1$ is positive semidefinite with largest eigenvalue 4; but it is not a strongly convex because its Hessian is not positive definite; The second quadratic function is not convex because its Hessian $\boldsymbol{H}_2$ contains a negative eigenvalue –0.01; The third quadratic function is convex with Lipschitz continuous gradient with $L = 4.005$ because its Hessian $\boldsymbol{H}_1$ is positive semidefinite with largest eigenvalue 4.005; and it is also a strongly convex because its Hessian is positive definite. ■

14.8

Note: there is a typo in part (a), line 4, where $\boldsymbol{y}$ should be $\boldsymbol{u}$.

(**a**) From Fig. 14.7 (see p. 496), it is intuitively clear that given $\boldsymbol{u}$, point

$$\boldsymbol{x} = \sup_{\boldsymbol{x} \in \mathrm{dom}(f)} \left(\boldsymbol{u}^T \boldsymbol{x} - f(\boldsymbol{x}) \right)$$

is where the gradient $\nabla f(\boldsymbol{x}) = \boldsymbol{u}$. In other words, the inverse mapping $\nabla^{-1} f(\boldsymbol{u})$ is equal to that point $\boldsymbol{x}$, i.e., $\nabla^{-1} f(\boldsymbol{u}) = \sup_{\boldsymbol{x} \in \mathrm{dom}(f)} \left(\boldsymbol{u}^T \boldsymbol{x} - \nabla^{-1} f(\boldsymbol{x}) \right)$.

(**b**) From part (a) we know that at $\boldsymbol{x} = \sup_{\boldsymbol{x} \in \mathrm{dom}(f)} \left(\boldsymbol{u}^T \boldsymbol{x} - f(\boldsymbol{x}) \right)$ we have $\boldsymbol{u} = \nabla f(\boldsymbol{x})$. By property (*e*) of conjugate functions (see p.498), we see that $\boldsymbol{u} = \nabla f(\boldsymbol{x})$ implies $\boldsymbol{x} = \nabla f^*(\boldsymbol{u})$. Hence we can write

$$\nabla f^*(\boldsymbol{u}) = \sup_{\boldsymbol{x} \in \mathrm{dom}(f)} \left(\boldsymbol{u}^T \boldsymbol{x} - \nabla^{-1} f(\boldsymbol{x}) \right)$$

This in conjunction with the result from part (a) that $\nabla^{-1} f(\boldsymbol{u}) = \sup_{\boldsymbol{x} \in \mathrm{dom}(f)} \left(\boldsymbol{u}^T \boldsymbol{x} - \nabla^{-1} f(\boldsymbol{x}) \right)$ yields

$$\nabla f^*(\boldsymbol{u}) = \nabla^{-1} f(\boldsymbol{u}).$$

(**c**) The Lagrange dual of the problem is defined by

$$q(\boldsymbol{\lambda}) = \inf_{\boldsymbol{x}} \left(f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{x} \right)$$

which can be written as

$$q(\boldsymbol{\lambda}) = -\sup_{\boldsymbol{x}} \left(-f(\boldsymbol{x}) - \boldsymbol{\lambda}^T \boldsymbol{x} \right) = -\sup_{\boldsymbol{x}} \left((-\boldsymbol{\lambda})^T \boldsymbol{x} - f(\boldsymbol{x}) \right) = -f^*(-\boldsymbol{\lambda}).$$

(**d**) The Lagrange dual of the problem is defined by

$$q(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{x}} \left\{ f(\boldsymbol{x}) + \boldsymbol{\lambda}^T (\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d}) + \boldsymbol{\mu}^T (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \right\}$$

which can be written as

$$
\begin{aligned}
q(\boldsymbol{\lambda},\boldsymbol{\mu}) &= -\boldsymbol{b}^T\boldsymbol{\mu} - \boldsymbol{d}^T\boldsymbol{\lambda} + \inf_{\boldsymbol{x}}\left\{f(\boldsymbol{x}) + \boldsymbol{\lambda}^T\boldsymbol{C}\boldsymbol{x} + \boldsymbol{\mu}^T\boldsymbol{A}\boldsymbol{x}\right\} \\
&= -\boldsymbol{b}^T\boldsymbol{\mu} - \boldsymbol{d}^T\boldsymbol{\lambda} - \sup_{\boldsymbol{x}}\left\{(-\boldsymbol{A}^T\boldsymbol{\mu} - \boldsymbol{C}^T\boldsymbol{\lambda})^T\boldsymbol{x} - f(\boldsymbol{x})\right\} \\
&= -\boldsymbol{b}^T\boldsymbol{\mu} - \boldsymbol{d}^T\boldsymbol{\lambda} - f^*(-\boldsymbol{A}^T\boldsymbol{\mu} - \boldsymbol{C}^T\boldsymbol{\lambda})
\end{aligned}
$$

■

14.9

(**a**) Let $\boldsymbol{x}^*$ be a solution of (P14.3), namely $\boldsymbol{x}^*$ minimizes $f(\boldsymbol{x})$ among those satisfying $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and $c_j(\boldsymbol{x}) \le 0$ for $j = 1,2,\ldots,q$. Because $c_j(\boldsymbol{x}^*) \le 0$ for $j = 1,2,\ldots,q$, we have

$$
f(\boldsymbol{x}^*) + \sum_{i=1}^{q} I_-(c_j(\boldsymbol{x}^*)) = f(\boldsymbol{x}^*) \ .
$$

Therefore, $\boldsymbol{x}^*$ minimizes the objective function in (P14.4a) among those satisfying $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. This implies that $\boldsymbol{x}^*$ is also a solution of (P14.4). Conversely, let $\tilde{\boldsymbol{x}}$ be a solution of (P14.4). Thus it satisfies $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. In addition, it must satisfy $c_j(\boldsymbol{x}) \le 0$ for $1 \le j \le q$ because violating any of the inequality constraints would lead the objective function to the value of infinity which would contradict the assumption that $\tilde{\boldsymbol{x}}$ minimizes the objective function in (P14.4a). Furthermore, because point $\tilde{\boldsymbol{x}}$ makes the 2nd term of the objective function zero, it actually minimizes $f(\boldsymbol{x})$ among those satisfying $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and $c_j(\boldsymbol{x}) \le 0$ for $1 \le j \le q$, hence $\tilde{\boldsymbol{x}}$ also solves problem (P14.3). Therefore, the two problems are equivalent.

(**b**) Function $F(\boldsymbol{x})$ is a composite function that is the positive combination of $f(\boldsymbol{x})$, which is convex, and the sum $\sum_{j=1}^{q} I_-(c_j(\boldsymbol{x}))$. Hence $F(\boldsymbol{x})$ is convex if we can show each $I_-(c_j(\boldsymbol{x}))$ is convex. To this end, let $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ be two points in the domain of $c_j(\boldsymbol{x})$ and note that $c_j(\boldsymbol{x})$ is convex which implies that

$$
c_j(\alpha\boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2) \le \alpha c_j(\boldsymbol{x}_1) + (1-\alpha)c_j(\boldsymbol{x}_2) \quad \text{for any} \quad \alpha \in (0,1) \qquad \text{(S14.3)}
$$

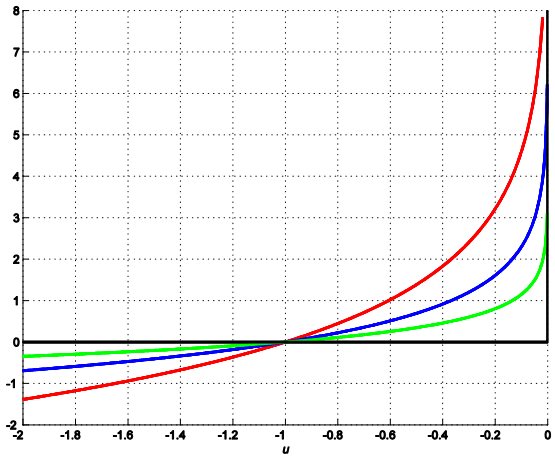
Now if $c_j(\alpha\boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2) > 0$, then at least one of the terms on the right-hand side of (S14.3) is also positive, and hence by definition

$$
I_-(c_j(\alpha\boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2)) \le \alpha I_-(c_j(\boldsymbol{x}_1)) + (1-\alpha)I_-(c_j(\boldsymbol{x}_2)) \qquad \text{(S14.4)}
$$

holds. If $c_j(\alpha\boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2) \le 0$, then $I_-(c_j(\alpha\boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2)) = 0$ and (S14.1) also hold because $I_-(u)$ is always nonnegative. This means that every $I_-(c_j(\boldsymbol{x}))$ is convex and so is $F(\boldsymbol{x})$. ■

14.10

(**a**), (**c**), and (**d**): See the figure below.



In the figure, function $I_{-}(u)$ is shown in black, function $h(u) = -(1/\tau)\log(-u)$ for τ = 0.5, 1, and 2 over [–2, 0) are shown in red, blue and green, respectively. It is observed that function $h(u)$ gets smoother while approximating $I_{-}(u)$ as the value of τ decreases, and its approximation accuracy improves as τ increases. And $h(u)$ approaches $I_{-}(u)$ as τ goes to infinity.

(**b**) For a fixed $\tau > 0$, the second-order derivative of $h(u)$ is given by $h''(u) = \dfrac{u^2}{\tau} \geq 0$ and hence $h(u)$ is convex.

(**e**) The observations made above suggest that we use the smooth convex formulation

$$\begin{aligned} &\text{minimize} \quad f(\boldsymbol{x}) \; - \; \frac{1}{\tau}\sum_{j=1}^{q}\log(-c_j(\boldsymbol{x})) \\ &\text{subject to:} \;\; \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \end{aligned} \tag{S14.5}$$

to approximate the problem in (P14.4). It can be readily verified that if the original problem, i.e.

$$\begin{aligned} &\text{minimize} \quad f(\boldsymbol{x}) \\ &\text{subject to:} \;\; \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ &\qquad\qquad c_j(\boldsymbol{x}) \leq 0 \;\; \text{for} \;\; j = 1, 2, \ldots, q \end{aligned}$$

is a CP problem, so is problem (S14.5). ■

14.11

To apply the Newton barrier algorithm, we apply Newton algorithm to the problem

$$\text{minimize} \;\; F_\tau(\boldsymbol{x}) = \tau \cdot \frac{\| \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \|_2^2}{\boldsymbol{c}^T\boldsymbol{x} + d} - \log\left(\boldsymbol{c}^T\boldsymbol{x} + d - \delta\right)$$

with τ initially fixed to τ = 1 in the 1st round and increased to $\tau = \gamma\tau$ with γ = 10 in

subsequent rounds. To apply Newton algorithm, we compute the gradient $\nabla F_\tau(\boldsymbol{x})$ and Hessian $\nabla^2 F_\tau(\boldsymbol{x})$ as follows:

$$\nabla F_\tau(\boldsymbol{x}) = \tau\left(-\frac{\| \boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}\|_2^2\, \boldsymbol{c}}{(\boldsymbol{c}^T\boldsymbol{x}+d)^2} + \frac{2\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b})}{\boldsymbol{c}^T\boldsymbol{x}+d}\right) - \frac{\boldsymbol{c}}{\boldsymbol{c}^T\boldsymbol{x}+d-\delta}$$

$$\nabla^2 F_\tau(\boldsymbol{x}) = \tau\left(\boldsymbol{H}_1 - (\boldsymbol{H}_2 + \boldsymbol{H}_2^T) + \boldsymbol{H}_3\right) + \boldsymbol{H}_4$$

where

$$\boldsymbol{H}_1 = \left(\frac{2\| \boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}\|_2^2}{(\boldsymbol{c}^T\boldsymbol{x}+d)^3}\right)\boldsymbol{c}\boldsymbol{c}^T$$

$$\boldsymbol{H}_2 = \left(\frac{2}{(\boldsymbol{c}^T\boldsymbol{x}+d)^2}\right)\boldsymbol{c}(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b})^T\boldsymbol{A}$$

$$\boldsymbol{H}_3 = \left(\frac{2}{\boldsymbol{c}^T\boldsymbol{x}+d}\right)\boldsymbol{A}^T\boldsymbol{A}$$

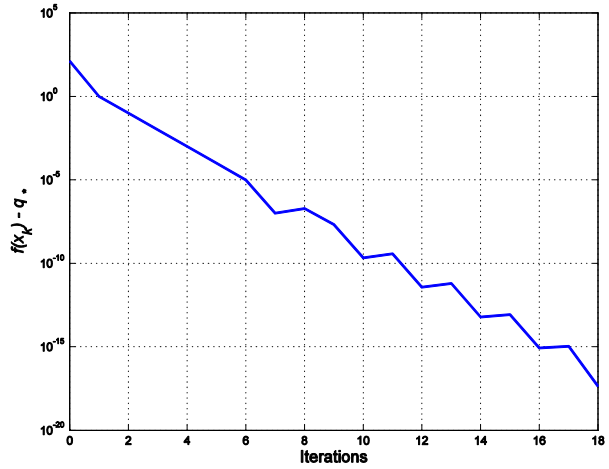
$$\boldsymbol{H}_4 = \left(\frac{1}{(\boldsymbol{c}^T\boldsymbol{x}+d-\delta)^2}\right)\boldsymbol{c}\boldsymbol{c}^T$$

To implement the algorithm, we start with a strictly feasible initial point $\boldsymbol{x}_0$ which can be any point satisfying $\boldsymbol{c}^T\boldsymbol{x}_0 + d > \delta$. For the problem at hand the line search step in Newton algorithm is carried out as follows.

Let $\boldsymbol{x}_k$ be the present iterate which is strictly feasible, i.e., $\boldsymbol{c}^T\boldsymbol{x}_k + d > \delta$. We compute Newton direction $\boldsymbol{d}_k$ as $\boldsymbol{d}_k = -\nabla^2 F_\tau(\boldsymbol{x}_k)\cdot\nabla F_\tau(\boldsymbol{x}_k)$ and update $\boldsymbol{x}_k$ to $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{d}_k$ where α_k is determined by ensuring the strict feasibility of $\boldsymbol{x}_{k+1}$ and is given by

$$\alpha_k = \begin{cases} 0.99\times\min\{1, (\delta - \boldsymbol{c}^T\boldsymbol{x}_k - d)/\boldsymbol{c}^T\boldsymbol{d}_k\} & \text{if } \boldsymbol{c}^T\boldsymbol{d}_k < 0 \\ 1 & \text{if } \boldsymbol{c}^T\boldsymbol{d}_k \geq 0 \end{cases}$$

With $\boldsymbol{x}_0 = [4 \quad -2]^T$ and initial τ = 1, a total of 18 rounds of Newton barrier iterations were carried out that led to practically the perfect global minimizer $\boldsymbol{x}^* = [0.004 \quad 1.006]^T$ at which the original objective function assumes the value $q_* = 0.004$. The figure below shows the profile of $f(\boldsymbol{x}_k) - q_*$ versus Newton barrier iterations. ■



MATLAB code

```
% Example: [xs,f,fs] = prob14_11_newton([4 -2]',1e-6,18);
function [xs,f,fs] = prob14_11_newton(x0,epsi,K)
A = [2 -1; 0 1]; b = [1 -1]'; c = [1 1]'; d = -1; dt = 0.01;
t = 1;
xk = [t; x0];
k = 0;
qs = 0.004;
f(1) = (norm(A*x0+b)^2)/(c'*x0+d) - qs;
while k < K,
  er = 1;
  while er > epsi;
    gk = g14_11(xk);
    Hk = h14_11(xk);
    dk = -Hk\gk;
    xw = xk(2:3);
    q = c'*dk;
    if q >= 0,
       ak = 1;
    else
       ak = 0.99*min((dt-d-c'*xw)/q,1);
    end
    adk = ak*dk;
    xk = [xk(1); xk(2:3)+adk];
    er = norm(adk);
  end
  xk(1) = 10*xk(1);
  k = k + 1;
  f(k+1) = (norm(A*xw+b)^2)/(c'*xw+d) - qs;
end
disp('Solution point:')
xs = xw
disp('Objective function at solution:')
fs = (norm(A*xs+b)^2)/(c'*xs+d)
disp('Satisfaction of the constraint:')
c'*xs + d - dt
figure(1)
semilogy(0:1:K,f,'-b','linewidth', 1.5)
grid
xlabel('Iterations','fontsize',14)
ylabel('\itf(x_k) - q_*','fontsize',14)

function g = g14_11(x)
t = x(1);
xw = x(2:3);
```

```
A = [2 -1; 0 1]; b = [1 -1]'; c = [1 1]'; d = -1; dt = 0.01;
f0 = c'*xw + d;
f1 = A*xw + b;
f2 = norm(f1)^2;
g1 = (-f2/f0^2)*c;
g2 = 2*A'*f1/f0;
g3 = c/(f0-dt);
g = t*(g1+g2) - g3;

function H = h14_11(x)
t = x(1);
xw = x(2:3);
A = [2 -1; 0 1]; b = [1 -1]'; c = [1 1]'; d = -1; dt = 0.01;
f0 = c'*xw + d;
f1 = A*xw + b;
f2 = norm(f1)^2;
H1 = (2*f2/f0^3)*(c*c');
H2 = (2/f0^2)*c*(f1'*A);
H3 = (2/f0)*(A'*A);
H4 = (1/(f0-dt)^2)*(c*c');
H = t*(H1 - (H2+H2') + H3) + H4;
```

14.12

(a) For a fixed $\boldsymbol{\theta}_2$, problem (P14.8) is equivalent to

$$\text{minimize} \quad \tfrac{1}{2}\|\boldsymbol{\theta}_1 - \boldsymbol{T}_1(\boldsymbol{u} - \boldsymbol{T}_2^T\boldsymbol{\theta}_2)\|_2^2 + \mu\|\boldsymbol{\theta}_1\|_1 \tag{S14.6}$$

It follows immediately that the global minimizer of (S14.6) is given by (P14.9).

(b) For a fixed $\boldsymbol{\theta}_1$, problem (P14.8) is equivalent to

$$\text{minimize} \quad \tfrac{1}{2}\|\boldsymbol{\theta}_2 - \boldsymbol{T}_2(\boldsymbol{u} - \boldsymbol{T}_1^T\boldsymbol{\theta}_1)\|_2^2 + \mu\|\boldsymbol{\theta}_2\|_1 \tag{S14.7}$$

It follows immediately that the global minimizer of (S14.2) is given by (P14.10).

(c) Based on (P14.9) and (P14.10), a natural iterative scheme for the problem in (P14.8) is to update the current pair $\{\boldsymbol{\theta}_1^{(k)}, \boldsymbol{\theta}_2^{(k)}\}$ to $\{\boldsymbol{\theta}_1^{(k+1)}, \boldsymbol{\theta}_2^{(k+1)}\}$ by

$$\boldsymbol{\theta}_1^{(k+1)} = \mathcal{S}_\mu\left(\boldsymbol{T}_1(\boldsymbol{u} - \boldsymbol{T}_2^T\boldsymbol{\theta}_2^{(k)})\right) \tag{S14.8a}$$

then

$$\boldsymbol{\theta}_2^{(k+1)} = \mathcal{S}_\mu\left(\boldsymbol{T}_2(\boldsymbol{u} - \boldsymbol{T}_1^T\boldsymbol{\theta}_1^{(k+1)})\right) \tag{S14.8b}$$

Another iterative scheme is given by

$$\boldsymbol{\theta}_1^{(k+1)} = \mathcal{S}_\mu\left(\boldsymbol{T}_1(\boldsymbol{u} - \boldsymbol{T}_2^T\boldsymbol{\theta}_2^{(k)})\right) \tag{S14.9a}$$

and

$$\boldsymbol{\theta}_2^{(k+1)} = \mathcal{S}_\mu\left(\boldsymbol{T}_2(\boldsymbol{u} - \boldsymbol{T}_1^T\boldsymbol{\theta}_1^{(k)})\right) \tag{S14.9b}$$

As we can see below, the second scheme allows a compact and nice formula for the iterations. If we denote

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{T}_1 \\ \boldsymbol{T}_2 \end{bmatrix}, \quad \boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \end{bmatrix}$$

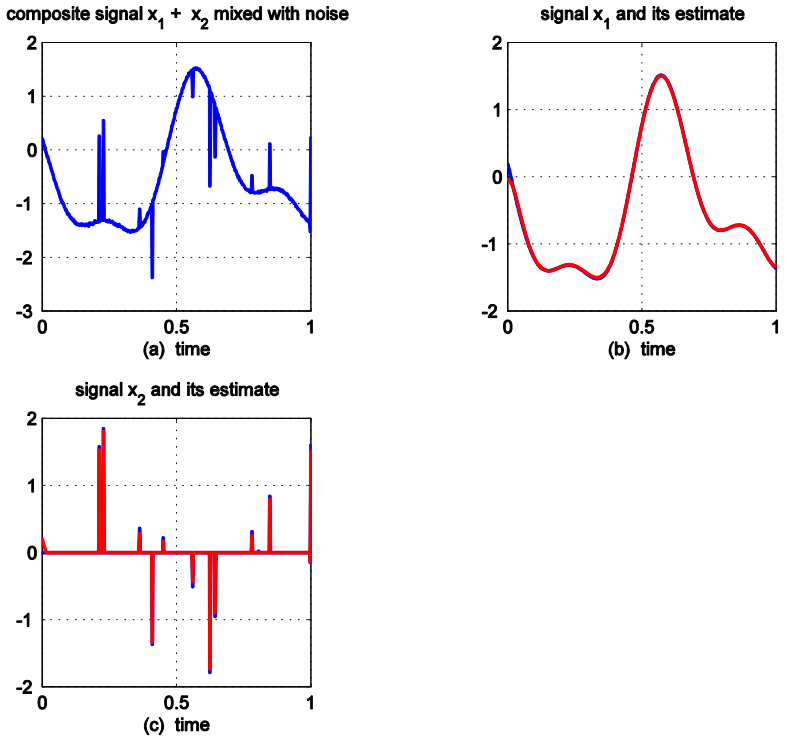
then the two vectors on the right-hand side of (S14.9) can be written together as

$$\begin{bmatrix} \boldsymbol{T}_1(\boldsymbol{u}-\boldsymbol{T}_2^T\boldsymbol{\theta}_2^{(k)}) \\ \boldsymbol{T}_2(\boldsymbol{u}-\boldsymbol{T}_1^T\boldsymbol{\theta}_1^{(k)}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{T}_1(\boldsymbol{u}-\boldsymbol{T}^T\boldsymbol{\theta}^{(k)}+\boldsymbol{T}_1^T\boldsymbol{\theta}_1^{(k)}) \\ \boldsymbol{T}_2(\boldsymbol{u}-\boldsymbol{T}^T\boldsymbol{\theta}^{(k)}+\boldsymbol{T}_2^T\boldsymbol{\theta}_2^{(k)}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{T}_1(\boldsymbol{u}-\boldsymbol{T}^T\boldsymbol{\theta}^{(k)})+\boldsymbol{\theta}_1^{(k)} \\ \boldsymbol{T}_2(\boldsymbol{u}-\boldsymbol{T}^T\boldsymbol{\theta}^{(k)})+\boldsymbol{\theta}_2^{(k)} \end{bmatrix} = \boldsymbol{T}(\boldsymbol{u}-\boldsymbol{T}^T\boldsymbol{\theta}^{(k)})+\boldsymbol{\theta}^{(k)}$$

hence the iteration scheme (S14.9) becomes

$$\boldsymbol{\theta}^{(k+1)} = S_\mu\left(\boldsymbol{T}(\boldsymbol{u}-\boldsymbol{T}^T\boldsymbol{\theta}^{(k)})+\boldsymbol{\theta}^{(k)}\right) \qquad \text{(S14.10)}$$

The iterative approach based on (S14.8a-b) was found to be more efficient than that based on (S14.10). The algorithm converges after 7 iterations. The MATLAB code implementing the first approach is enclosed below. With $\boldsymbol{\theta}_2^{(0)} = \mathbf{0}$ and 31 iterations, the results obtained are shown in the figure below.




```
% Input:
% (x1,x2): two signals with different features.
%  T1 and T2: two orthonormal bases.
% (st,sig): Gaussian white noise with standard deviation
%           sig. Noise is to be generated with initial
%           set to st.
% mu: regularization parameter \mu.
% K: number of iterations to be performed.
% Output:
% x1s: estimate of signal x1
% x2s: estimate of signal x2
% u: noise contaminated signal u = x1 + x2 + w
```

```
% Example:
% t = 0:1/511:1;
% x1 = -0.9*sin(2*pi*1.2*t)+0.6*cos(2*pi*1.8*t)-0.5*sin(2*pi*3.1*t)-0.4;
% x2 = zeros(512,1);
% rand('state',14); r = randperm(512);
% randn('state',28); x2(r(1:13)) = randn(13,1);
% T1 = dctmtx(512); T2 = eye(512);
function [x1s,x2s,u] = prob14_12a(x1,x2,T1,T2,st,sig,mu,K)
T1t = T1';
T2t = T2';
x1 = x1(:);
x2 = x2(:);
n = length(x1);
randn('state',st)
w = sig*randn(n,1);
u = x1 + x2 + w;
a2s = zeros(n,1);
k = 0;
while k < K,
   ck1 = T1*(u - T2t*a2s);
   a1s = max(0,abs(ck1)-mu).*sign(ck1);
   ck2 = T2*(u - T1t*a1s);
   a2s = max(0,abs(ck2)-mu).*sign(ck2);
   k = k + 1;
end
x1s = T1'*a1s;
x2s = T2'*a2s;
t = 0:1/(n-1):1;
figure(1)
subplot(221)
plot(t,u,'b-','linewidth',1.4)
axis square
grid
xlabel('(a)  time')
title('composite signal x_1+x_2 mixed with noise')
subplot(222)
plot(t,x1,'b-',t,x1s,'r-','linewidth',1.4)
axis square
grid
title('signal x_1 and its estimate')
xlabel('(b)  time')
subplot(223)
plot(t,x2,'b-',t,x2s,'r-','linewidth',1.4)
axis square
grid
title('signal x_2 and its estimate')
```

```
xlabel('(c)  time')
disp('average estimation error for signal x1:')
norm(x1-x1s)/sqrt(n)
disp('average estimation error for signal x2:')
norm(x2-x2s)/sqrt(n)
```

The average 2-norm estimation errors were found to be $\| \tilde{\boldsymbol{x}}_1 - \boldsymbol{x}_1 \|_2 / \sqrt{512}$ = 0.0159 and $\| \tilde{\boldsymbol{x}}_2 - \boldsymbol{x}_2 \|_2 / \sqrt{512}$ = 0.0144. ■

4.13

Note: There is a typo in the statement of part (b): In the line defining set $\mathcal{S}_1$, the first constraint should be $-x_1 \le 0$.

(a) By applying scaled ADMM to the problem in (P14.11), we obtain

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}} \left[I_{S_1}(\boldsymbol{x}) + \tfrac{\alpha}{2} \| \boldsymbol{x} - (\boldsymbol{y}_k - \boldsymbol{\nu}_k) \|_2^2 \right] \\
\boldsymbol{y}_{k+1} &= \arg\min_{\boldsymbol{y}} \left[I_{S_2}(\boldsymbol{y}) + \tfrac{\alpha}{2} \| \boldsymbol{y} - (\boldsymbol{x}_{k+1} + \boldsymbol{\nu}_k) \|_2^2 \right] \\
\boldsymbol{\nu}_{k+1} &= \boldsymbol{\nu}_k + \boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}
\end{aligned}$$

We see that the $\boldsymbol{x}$- and $\boldsymbol{y}$-minimization steps are merely projection of vectors $\boldsymbol{y}_k - \boldsymbol{\nu}_k$ and $\boldsymbol{x}_{k+1} + \boldsymbol{\nu}_k$ onto sets $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Therefore the ADMM iterations for (P14.11) can be expressed as

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= P_{S_1}(\boldsymbol{y}_k - \boldsymbol{\nu}_k) \\
\boldsymbol{y}_{k+1} &= P_{S_2}(\boldsymbol{x}_{k+1} + \boldsymbol{\nu}_k) \\
\boldsymbol{\nu}_{k+1} &= \boldsymbol{\nu}_k + \boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}
\end{aligned} \qquad \text{(S14.11a-c)}$$

(b) Under the circumstances, the $\boldsymbol{x}$-minimization in (S14.11a) is carried out by solving the convex problem

$$\begin{aligned}
\text{minimize} \quad & \| \boldsymbol{x} - (\boldsymbol{y}_k - \boldsymbol{\nu}_k) \|_2 \\
\text{subject to:} \quad & -x_1 \le 0 \\
& -x_2 \le 0 \\
& x_1 + 2x_2 - 2 \le 0
\end{aligned}$$

and the $\boldsymbol{y}$-minimization in (S14.11b) is carried out by solving the convex problem

$$\begin{aligned}
\text{minimize} \quad & \| \boldsymbol{y} - (\boldsymbol{x}_{k+1} + \boldsymbol{\nu}_k) \|_2 \\
\text{subject to:} \quad & 2y_1 - y_2 - 1.6 \le 0 \\
& -2y_1 - y_2 + 2.4 \le 0 \\
& y_2 - 3 \le 0
\end{aligned}$$

Since the problem requires to find a point in the intersection without optimality conditions, the ADMM iterations were terminated as soon as a point that satisfies the six constraints (which

define sets S_1 and S_2) was identified. With initial points $\boldsymbol{y}_0 = [0 \quad 0]'$ and $\boldsymbol{v}_0 = [0 \quad 0]'$, it took 5 ADMM iterations to yield a point $\boldsymbol{x}^* = [1.04 \quad 0.48]^T$ which lies in the intersection of sets S_1 and S_2.

MATLAB Code

```
% Input:
% A1, b1 that define set S1;
% A2, b2 that define set S2.
% Output:
% X: columns of X are iterates xk;
% xs: last column of X which lies in the intersection of sets S1 and s2.
% k: number of ADMM iterations used.
% Example: A1 = [-1 0; 0 -1; 1 2]; b1 = [0 0 2]';
% A2 = [2 -1; -2 -1; 0 1]; b2 = [1.6, -2.4, 3]';
% [X,xs,k] = prob4_13(A1,b1,A2,b2);
function  [X,xs,k] = prob4_13(A1,b1,A2,b2)
X = [];
yk = [0 0]';
vk = yk;
flag = 0;
k = 0;
while flag == 0
  sk = yk - vk;
  cvx_begin quiet
    variable x(2,1)
    minimize(norm(x-sk))
    subject to
    A1*x <= b1;
  cvx_end
  X = [X x];
  xk1 = x;
  e1 = A1*xk1 - b1;
  e2 = A2*xk1 - b2;
  ek = max([e1; e2]);
  if ek <= 0
     flag = 1;
     k = k + 1;
     break
  end
  tk = xk1 + vk;
  cvx_begin quiet
    variable y(2,1)
    minimize(norm(y-tk))
    subject to
```

```
    A2*y <= b2;
  cvx_end
  yk1 = y;
  wk = xk1 - yk1;
  vk1 = vk + wk;
  k = k + 1;
  yk = yk1;
  vk = vk1;
end
xs = X(:,end)
```

■

4.14

(a) Suppose each $\mathcal{S}_i$ is a set in Euclidean space R^n and we define $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_K$, then the product set $\mathcal{S}$ is an Euclidean space of dimension Kn in which a point $\boldsymbol{x}$ assumes the form $(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_K)$. Now if we define $\mathcal{R} = \{(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_K) : \boldsymbol{x}_1 = \boldsymbol{x}_2 = \cdots = \boldsymbol{x}_K\}$, then the problem of finding a point in the intersection of K convex sets $\{\mathcal{S}_i, i = 1, 2, \ldots, K\}$ becomes the problem of finding an point in the intersection of convex sets $\mathcal{S}$ and $\mathcal{R}$, which can be solved by the ADMM-based method developed in Problem 14.13, namely,

$$\begin{aligned} \boldsymbol{x}_{k+1} &= P_S(\boldsymbol{y}_k - \boldsymbol{\nu}_k) \\ \boldsymbol{y}_{k+1} &= P_R(\boldsymbol{x}_{k+1} + \boldsymbol{\nu}_k) \\ \boldsymbol{\nu}_{k+1} &= \boldsymbol{\nu}_k + \boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1} \end{aligned} \qquad \text{(S14.12a-c)}$$

The projection of a point $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_K)$ onto set $\mathcal{S}$ is given by

$$P_S(\boldsymbol{x}) = (P_{S_1}(\boldsymbol{x}_1), P_{S_2}(\boldsymbol{x}_2), \cdots, P_{S_K}(\boldsymbol{x}_K)) \qquad \text{(S14.13)}$$

and the projection of a point $\boldsymbol{z} = (\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_K)$ onto set $\mathcal{R}$ can be obtained by solving the convex problem

$$\begin{aligned} &\text{minimize} \quad \| \boldsymbol{x} - \boldsymbol{z} \|_2 \\ &\text{subject to:} \quad \boldsymbol{x} \in \mathcal{R} \end{aligned}$$

whose solution is given by $\boldsymbol{x}^* = (\bar{\boldsymbol{x}}, \bar{\boldsymbol{x}}, \cdots, \bar{\boldsymbol{x}})$ with $\bar{\boldsymbol{x}} = \frac{1}{K} \sum_{i=1}^{K} \boldsymbol{z}_i$.

(b) We begin by writing $\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}$ as $\boldsymbol{a}_i^T \boldsymbol{x} \le b_i$ for $i = 1, 2, \ldots, q$. Define $\mathcal{S}_i = \{\boldsymbol{x} : \boldsymbol{a}_i^T \boldsymbol{x} \le b_i\}$, the problem of finding a feasible point of $\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}$ becomes that of finding a point in the intersection of sets $\{\mathcal{S}_i, i = 1, 2, \ldots, q\}$.

The $\boldsymbol{x}$-minimization step in (S14.12a) is carried out via (S14.13) where, based on the result of Prob. 14.5(b), each $P_{S_i}(\boldsymbol{x}_i)$ is computed using

$$P_{S_i}(\boldsymbol{x}_i) = \begin{cases} \boldsymbol{x}_i & \text{if } \boldsymbol{a}_i^T \boldsymbol{x}_i \le b_i \\ \dfrac{b_i}{\boldsymbol{a}_i^T \boldsymbol{a}_i}\boldsymbol{a}_i + \left(\boldsymbol{I} - \dfrac{1}{\boldsymbol{a}_i^T \boldsymbol{a}_i}\boldsymbol{a}_i \boldsymbol{a}_i^T \right) \boldsymbol{x}_i & \text{if } \boldsymbol{a}_i^T \boldsymbol{x}_i > b_i \end{cases}$$

With initial points **`y0 = zeros(24,1)`** and **`v0 = zeros(24,1)`**, it took 14 ADMM iterations to converge to a solution point whose first 4 components are given by

$$\boldsymbol{x}^* = \begin{bmatrix} 0 \\ 0 \\ 0.968647067513139 \\ 2.120992601412950 \end{bmatrix}$$

It can readily verified that $\boldsymbol{x}^*$ lies in the intersection of sets $\{S_i, \text{I} = 1, 2, \ldots, 6\}$ as it satisfies $\boldsymbol{A}\boldsymbol{x}^* \le \boldsymbol{b}$.

MATLAB Code

```
% A = [-1 0 0 0; 0 -1 0 0; 1 2 0 0; 0 0 0 -1; 0 0 -1 -1; 0 0 1 2];
% b = [0 0 2 -2 -3 6]';
% Example: [xs,r] = prob4_12b_573(A,b,14);
function  [xs,r] = prob4_12b_573(A,b,K)
[q,n] = size(A);
I = eye(n);
N = q*n;
yk = zeros(N,1);
vk = yk;
k = 0;
r = zeros(1,K);
while k < K,
  sk = yk - vk;
  xk1 = zeros(N,1);
  for i = 1:q,
     bi = b(i);
     ai = A(i,:)';
     si = sk((i-1)*n+1:i*n);
     if ai'*si <= bi,
        xk1((i-1)*n+1:i*n) = si;
     else
        ti = 1/(ai'*ai);
        xw = (bi*ti)*ai + (I - (ti*ai)*ai')*si;
        xk1((i-1)*n+1:i*n) = xw;
     end
  end
```

```
  zk = xk1 + vk;
  xb = zeros(n,1);
  for i = 1:q,
     xb = xb + zk((i-1)*n+1:i*n);
  end
  xb = xb/q;
  yk1 = zeros(N,1);
  for i = 1:q,
     yk1((i-1)*n+1:i*n) = xb;
  end
  wk = xk1 - yk1;
  vk1 = vk + wk;
  k = k + 1;
  r(k) = norm(wk);
  yk = yk1;
  vk = vk1;
end
format long
disp('Solution point:')
xs = xk1(1:n)
disp('Constraints atisfaction:')
A*xs - b
figure(1)
kt = 1:1:K;
semilogy(kt,r,'linewidth',1.5)
grid
axis square
axis([1 K 1e-18 10e0])
xlabel('Iterations')
ylabel('Primal residue')
```

(c) Evidently, the problem can be solved using the method from part (b) with vector $\boldsymbol{b}-\varepsilon\boldsymbol{e}$ replacing $\boldsymbol{b}$, where $\boldsymbol{e}$ is the all-one vector and $\varepsilon>0$ is a small scalar to ensure strict feasibility. Running the code used in part (b) with $\boldsymbol{b}$ replaced by $\boldsymbol{b}-10^{-3}\boldsymbol{e}$, it took 14 ADMM iterations to obtain the solution point whose first 4 components are given by

$$\boldsymbol{x}^{*}=\begin{bmatrix}0.001517975537266\\0.001517975537266\\0.969667756053377\\2.121308444572764\end{bmatrix}$$

which is strictly feasible for $\boldsymbol{A}\boldsymbol{x}\le\boldsymbol{b}$ because

$$\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b} = \begin{bmatrix} -0.001517975537266 \\ -0.001517975537266 \\ -1.995446073388203 \\ -0.121308444572763 \\ -0.090976200626140 \\ -0.787715354801096 \end{bmatrix} < \boldsymbol{0}\,. \qquad \blacksquare$$

4.15 The Lagrangian of the objective in (P14.12) is given by

$$L(\boldsymbol{x}_1,...,\boldsymbol{x}_K,\boldsymbol{y},\boldsymbol{\lambda}) = \sum_{i=1}^{K}\Big[\, f_i(\boldsymbol{x}_i) + \boldsymbol{\lambda}_i^T(\boldsymbol{x}_i - \boldsymbol{y}) + \tfrac{\alpha}{2}\,\|\,\boldsymbol{x}_i - \boldsymbol{y}\,\|_2^2\Big]$$

from which the standard ADMM iterations can be deduced as

$$\begin{aligned}
\boldsymbol{x}_i^{(k+1)} &= \arg\min_{\boldsymbol{x}_i}\Big[\, f_i(\boldsymbol{x}_i) + (\boldsymbol{\lambda}_i^{(k)})^T(\boldsymbol{x}_i - \boldsymbol{y}^{(k)}) + \tfrac{\alpha}{2}\,\|\,\boldsymbol{x}_i - \boldsymbol{y}^{(k)}\,\|_2^2\Big] \quad \text{for } i = 1,...,K \\
\boldsymbol{y}^{(k+1)} &= \arg\min_{\boldsymbol{y}}\left[\sum_{i=1}^{K}\Big[-(\boldsymbol{\lambda}_i^{(k)})^T\boldsymbol{y} + \tfrac{\alpha}{2}\,\|\,\boldsymbol{y} - \boldsymbol{x}_i^{(k+1)}\,\|_2^2\Big]\right] \\
\boldsymbol{\lambda}_i^{(k+1)} &= \boldsymbol{\lambda}_i^{(k)} + \alpha(\boldsymbol{x}_i^{(k+1)} - \boldsymbol{y}^{(k+1)}) \quad \text{for } i = 1,...,K
\end{aligned}$$

where $\boldsymbol{y}$-minimization step can be carried out in closed-form as

$$\boldsymbol{y}^{(k+1)} = \frac{1}{K}\sum_{i=1}^{K}\Big(\boldsymbol{x}_i^{(k+1)} + \tfrac{1}{\alpha}\boldsymbol{\lambda}_i^{(k)}\Big)$$

Hence the ADMM iterations assume the form

$$\begin{aligned}
\boldsymbol{x}_i^{(k+1)} &= \arg\min_{\boldsymbol{x}_i}\Big[\, f_i(\boldsymbol{x}_i) + (\boldsymbol{\lambda}_i^{(k)})^T(\boldsymbol{x}_i - \boldsymbol{y}^{(k)}) + \tfrac{\alpha}{2}\,\|\,\boldsymbol{x}_i - \boldsymbol{y}^{(k)}\,\|_2^2\Big] \quad \text{for } i = 1,...,K \\
\boldsymbol{y}^{(k+1)} &= \frac{1}{K}\sum_{i=1}^{K}\Big(\boldsymbol{x}_i^{(k+1)} + \tfrac{1}{\alpha}\boldsymbol{\lambda}_i^{(k)}\Big) \qquad \text{(S14.14a-c)} \\
\boldsymbol{\lambda}_i^{(k+1)} &= \boldsymbol{\lambda}_i^{(k)} + \alpha(\boldsymbol{x}_i^{(k+1)} - \boldsymbol{y}^{(k+1)}) \quad \text{for } i = 1,...,K
\end{aligned}$$

The ADMM iterations in (S14.14) can be simplified by *averaging* the updates as follows. By defining the average updates

$$\overline{\boldsymbol{x}}^{(k)} = \frac{1}{K}\sum_{i=1}^{K}\boldsymbol{x}_i^{(k)} \quad \text{and} \quad \overline{\boldsymbol{\lambda}}^{(k)} = \frac{1}{K}\sum_{i=1}^{K}\boldsymbol{\lambda}_i^{(k)}$$

Eq. (S14.14b) becomes

$$\boldsymbol{y}^{(k+1)} = \overline{\boldsymbol{x}}^{(k+1)} + \tfrac{1}{\alpha}\overline{\boldsymbol{\lambda}}^{(k)} \qquad \text{(S14.15)}$$

By averaging Eq. (S14.14c), we obtain

$$\overline{\boldsymbol{\lambda}}^{(k+1)} = \overline{\boldsymbol{\lambda}}^{(k)} + \alpha(\overline{\boldsymbol{x}}^{(k+1)} - \boldsymbol{y}^{(k+1)}) \qquad \text{(S14.16)}$$

Substituting (S14.15) into (S14.16) yields $\overline{\boldsymbol{\lambda}}^{(k+1)} = \boldsymbol{0}$ which in turn leads (S14.15) to

$$\boldsymbol{y}^{(k+1)} = \bar{\boldsymbol{x}}^{(k+1)}$$

Consequently, the ADMM iterations in (S14.14) are simplified to

$$\boldsymbol{x}_i^{(k+1)} = \arg\min_{\boldsymbol{x}_i}\left[f_i(\boldsymbol{x}_i) + (\boldsymbol{\lambda}_i^{(k)})^T(\boldsymbol{x}_i - \bar{\boldsymbol{x}}^{(k)}) + \tfrac{\alpha}{2}\| \boldsymbol{x}_i - \bar{\boldsymbol{x}}^{(k)} \|_2^2 \right] \quad \text{for } i = 1,...,K$$

$$\boldsymbol{\lambda}_i^{(k+1)} = \boldsymbol{\lambda}_i^{(k)} + \alpha(\boldsymbol{x}_i^{(k+1)} - \bar{\boldsymbol{x}}^{(k+1)}) \quad \text{for } i = 1,...,K$$

■

Chapter 15 Algorithms for General Nonconvex Problems

15.1 The MATLAB code solving the problem is given below.

```
% input:
% x0: an initial point in the domain of f(x)
% tau: positive scalar tau (typically tau is a large)
% roh: initial value of roh the defines trust region |x_i| <= roh
% K: number of iterations (the number of sub-problems to be solved).
% output:
% xs: local minimizer obtained.
% fs: objective function at xs.
% X: trajectory of the iterates.
% Example: [xs,fs,X] = prob15_1([1 1]',1e5,0.1,20);
function [xs,fs,X] = prob15_1(x0,tau,roh,K)
xk = x0(:);
e = ones(2,1);
k = 0;
X = xk;
f = log(1+xk(1)^2) + xk(2);
r = roh;
a = 0.1; bs = 1.1; bf = 0.5;
while k < K
    xk1 = xk(1);
    xk2 = xk(2);
    gf1 = 2*xk1/(1+xk1^2);
    gf2 = 1;
    gf = [gf1 gf2]';
    hw1 = max([1-xk1^2 0]);
    h11 = 2*hw1/(1+xk1^2)^2;
    H = [h11 0; 0 0];
    ak = (1+xk1^2)^2 + xk2^2 - 4;
    ga1 = 4*xk1*(1+xk1^2);
    ga2 = 2*xk2;
    ga = [ga1 ga2]';
    F1 = f(k+1);
    F2 = abs(ak);
    F = F1 + tau*F2;
    cvx_begin quiet
```

```
      variable x(2,1);
      u = abs(ak + ga'*(x-xk));
      minimize(0.5*(x-xk)'*H*(x-xk)+gf'*(x-xk)+tau*u);
      subject to
      -r*e + xk <= x <= r*e + xk;
    cvx_end
    xt = x;
    xt1 = xt(1); xt2 = xt(2);
    Ft1 = log(1+xt1^2) + xt2;
    t2 = (1+xt1^2)^2 + xt2^2 - 4;
    Ft2 = abs(t2);
    Ft = Ft1 + tau*Ft2;
    Ftt = abs(ak + ga'*(xt-xk));
    Ftt = 0.5*(xt-xk)'*H*(xt-xk)+gf'*(xt-xk)+tau*Ftt;
    dt = F - Ftt;
    dtt = F - Ft;
    if dt >= a*dtt
       xk = xt;
       r = bs*r;
    elseif dt < a*dtt,
       r = bf*r;
    end
    fk = log(1+xk(1)^2) + xk(2);
    f = [f fk];
    X = [X xk];
    k = k + 1;
end
xs = xk;
fs = log(1+xs(1)^2) + xs(2);figure(1)
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel({'Iteration'},'fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
axis square
% axis([0 k -0.5 2.5])
disp('satisfaction of equality constraint:')
(1+xs(1)^2)^2 + xs(2)^2 - 4
```

With $\tau = 10^5$ and $\rho = 0.1$, it took the algorithm 20 iterations to converge to the solution

$$\boldsymbol{x}^* = \begin{bmatrix} 0.001087661796966 \\ -1.732055222534393 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = -1.732054039526909$ and the

satisfaction of the equality constraint is measured by the value

$$|a(\boldsymbol{x}^*)| = 1.765992643587566 \times 10^{-5}.$$

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 1.693147180559945$ and it violates the constraint as $|a(\boldsymbol{x}_0)| = 1$. ■

15.2 The MATLAB code solving the problem is given below.

```
% input:
% [x0,lam0]: initial point
% tau: weight in merit function for constraint violation
% mu: weight for penalty term
% rou: bound on \delta
% epsi: tolerance for convergence
% output:
% xs: local minimizer obtained.
% fs: objective function at xs.
% k: Number of iterations performed.
% X: trajectory of the iterates.
% Example: [xs,fs,k,X] = prob15_2([1 1]',1,20,1,0.1,1e-6);
function [xs,fs,k,X] = prob15_2(x0,lam0,eta,tau,rou,epsi)
xk = x0(:);
lamk = lam0;
Zk = eye(2,2);
k = 0;
X = xk;
f = log(1+xk(1)^2) + xk(2);
err = 1;
while err >= epsi
    xk1 = xk(1);
    xk2 = xk(2);
    gf1 = 2*xk1/(1+xk1^2);
    gf = [gf1, 1]';
    ak = (1+xk1^2)^2 + xk2^2 - 4;
    ga1 = 4*xk1*(1+xk1^2);
    ga2 = 2*xk2;
    Ae = [ga1, ga2];
    cvx_begin quiet
      variable dt(2,1)
      variable u(1,1)
      variable v(1,1)
      variable r(1,1)
      dual variable y1
      minimize(0.5*dt'*Zk*dt + gf'*dt + eta*(u+v+r));
      subject to
```

```
  y1:ak + Ae*dt == u - v;
  norm(dt) <= rou + r;
  u >= 0;
  v >= 0;
  r >= 0;
cvx_end
nuvw = norm([u, v, r]);
if nuvw <= 1e-6
   dtk = dt;
   lam_qp = y1;
else
   ak = ak - u + v;
   rou_t = rou + r;
   cvx_begin quiet
     variable dt(2,1)
     dual variable y1
     minimize(0.5*dt'*Zk*dt + gf'*dt);
     subject to
     y1: Ae*dt == -ak;
     norm(dt) <= rou_t;
   cvx_end
   dtk = dt;
   lam_qp = y1;
end
dlam = lam_qp - lamk;
Kt = 40;
alf = 0:1/(Kt-1):1;
psi = zeros(Kt,1);
for i = 1:Kt,
    alfi = alf(i);
    xw = xk + alfi*dtk;
    x1 = xw(1);
    x2 = xw(2);
    t1 = log(1+x1^2) + x2;
    t2 = abs((1 + x1^2)^2 + x2^2 - 4);
    psi(i) = t1 + tau*t2;
end
[~,ind] = min(psi);
% alfk = (ind - 1)/(Kt-1);
alfk = alf(ind);
xnew = xk + alfk*dtk;
lamk1 = lamk + alfk*dlam;
x1 = xnew(1);
x2 = xnew(2);
gf1 = 2*x1/(1+x1^2);
gfnew = [gf1, 1]';
```

```
    ga1 = 4*x1*(1+x1^2);
    ga2 = 2*x2;
    Aenew = [ga1, ga2];
    gamk = (gfnew - gf) + (Aenew - Ae)'*lamk1;
    ct1 = dtk'*gamk;
    ct2 = dtk'*Zk*dtk;
    if ct1 >= 0.2*ct2,
       thet = 1;
    else
       thet = (0.8*ct2)/(ct2 - ct1);
    end
    ht = Zk*dtk;
    ek = thet*gamk + (1-thet)*ht;
    Zk = Zk + (ek*ek')/(dtk'*ek) - (ht*ht')/ct2;
    err = norm([(xnew-xk);(lamk1-lamk)]);
    xk = xnew;
    lamk = lamk1;
    fk = log(1+xk(1)^2) + xk(2);
    f = [f fk];
    X = [X xk];
    k = k + 1;
end
xs = xk;
fs = fk;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel({'Iteration';'(b)'},'fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
axis square
% axis([0 k -1 3])
disp('satisfaction of equality constraint:')
(1+xs(1)^2)^2 + xs(2)^2 - 4
```

With $\lambda_0 = 1, \eta = 20, \tau = 1, \rho = 0.1,$ and $\varepsilon = 10^{-6}$, it took the algorithm 48 iterations to converge to the solution

$$\boldsymbol{x}^* = \begin{bmatrix} -0.000857400644365 \\ -1.732233777615760 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = -1.732233042480166$ and the satisfaction of the equality constraint is measured by the value

$$|a(\boldsymbol{x}^*)| = 6.353305852382363 \times 10^{-4}.$$

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 1.693147180559945$ and it violates the constraint as $a(\boldsymbol{x}_0) = 1$. ■

15.3 The MATLAB code solving the problem is given below.

```
% input:
% x0: an initial point in the domain of f(x)
% tau: positive scalar tau (typically tau is a large)
% roh: initial value of roh the defines trust region |x_i| <= roh
% K: number of iterations (the number of convex sub-problems to be solved).
% output:
% xs: local minimizer obtained.
% fs: objective function at xs.
% X: trajectory of the iterates.
% Example:
% x0 = [3 2 3]'; [xs,fs,X] = prob15_3(x0,200,0.1,25);
function [xs,fs,X] = prob15_3(x0,tau,roh,K)
xk = x0(:);
e = ones(3,1);
k = 0;
X = xk;
x1 = xk(1);
x2 = xk(2);
x3 = xk(3);
f = x1^4 + 2*x2^4 + x3^4 - (x1^2)*(x2^2) - (x1^2)*(x3^2);
r = roh;
a = 0.1; bs = 1.1; bf = 0.5;
while k < K
    x1 = xk(1);
    x2 = xk(2);
    x3 = xk(3);
    gf1 = 4*x1^3 - 2*x1*(x2^2) - 2*x1*(x3^2);
    gf2 = 8*x2^3 - 2*x2*(x1^2);
    gf3 = 4*x3^3 - 2*x3*(x1^2);
    gf = [gf1 gf2 gf3]';
    h11 = 12*x1^2 - 2*x2^2 - 2*x3^2;
    h12 = -4*x1*x2;
    h13 = -4*x1*x3;
    h22 = 24*x2^2 - 2*x1^2;
    h33 = 12*x3^2 - 2*x1^2;
    Hw = [h11 h12 h13; h12 h22 0; h13 0 h33];
    [V,Dw] = eig(Hw);
    dw = diag(Dw);
    dww = max(dw,1e-4);
    Dww = diag(dww);
```

```
    H = V*Dww*V';
    a1k = x1^4 + x2^4 + x3^4 - 25;
    g11 = 4*x1^3;
    g12 = 4*x2^3;
    g13 = 4*x3^3;
    g1 = [g11 g12 g13]';
    a2k = 8*x1^2 + 14*x2^2 + 7*x3^2 - 56;
    g21 = 16*x1;
    g22 = 28*x2;
    g23 = 14*x3;
    g2 = [g21 g22 g23]';
    F1 = f(k+1);
    F2 = abs(a1k);
    F3 = abs(a2k);
    F = F1 + tau*(F2+F3);
    cvx_begin quiet
      variable x(3,1)
      u1 = abs(a1k + g1'*(x-xk));
      u2 = abs(a2k + g2'*(x-xk));
      minimize(0.5*(x-xk)'*H*(x-xk)+gf'*(x-xk)+tau*(u1+u2));
      subject to
      -r*e + xk <= x <= r*e + xk;
    cvx_end
    xt = x;
    xt1 = xt(1); xt2 = xt(2); xt3 = xt(3);
    Ft1 = xt1^4 + 2*xt2^4 + xt3^4 - (xt1^2)*(xt2^2) - (xt1^2)*(xt3^2);
    Ft2 = abs(xt1^4 + xt2^4 + xt3^4 - 25);
    Ft3 = abs(8*xt1^2 + 14*xt2^2 + 7*xt3^2 - 56);
    Ft = Ft1 + tau*(Ft2+Ft3);
    Ftt1 = abs(a1k + g1'*(xt-xk));
    Ftt2 = abs(a2k + g2'*(xt-xk));
    Ftt = 0.5*(xt-xk)'*H*(xt-xk)+gf'*(xt-xk)+tau*(Ftt1+Ftt2);
    dt = F - Ftt;
    dtt = F - Ft;
    if dt >= a*dtt
       xk = xt;
       r = bs*r;
    elseif dt < a*dtt,
       r = bf*r;
    end
    fk = xk(1)^4 + 2*xk(2)^4 + xk(3)^4 - (xk(1)^2)*(xk(2)^2) - (xk(1)^2)*(xk(3)^2);
    f = [f fk];
    X = [X xk];
    k = k + 1;
end
xs = xk;
```

```
fs = xs(1)^4 + 2*xs(2)^4 + xs(3)^4 - (xs(1)^2)*(xs(2)^2) -
(xs(1)^2)*(xs(3)^2);
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel('Iteration','fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
axis square
axis([0 k 0 80])
disp('satisfaction of equality 1st constraint:')
xs(1)^4 + xs(2)^4 + xs(3)^4 - 25
disp('satisfaction of equality 2nd constraint:')
8*xs(1)^2 + 14*xs(2)^2 + 7*xs(3)^2 - 56
```

With $\tau = 200$ and $\rho = 0.1$, it took the algorithm 25 iterations to converge to the solution

$$\boldsymbol{x}^* = \begin{bmatrix} 1.879636265248005 \\ 0.464099848584489 \\ 1.879220913985726 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = 11.808614467977174$. The two equality constraints are approximately satisfied in the sense of

$$|a_1(\boldsymbol{x}^*)| = 1.924661319208099 \times 10^{-9} \text{ and } |a_2(\boldsymbol{x}^*)| = 7.370744015133823 \times 10^{-10}.$$

For comparison, the objective at the initial point $\boldsymbol{x}_0 = \begin{bmatrix} 3 & 2 & 3 \end{bmatrix}^T$ assumes the value $f(\boldsymbol{x}_0) = 77$ and it violates both constraints. ■

15.4 The MATLAB code solving the problem is given below.

```
% input:
% [x0,lam0]: initial point
% tau: weight in merit function for constraint violation
% mu: weight for penalty term
% rou: bound on \delta
% epsi: tolerance for convergence
% output:
% xs: local minimizer obtained.
% fs: objective function at xs.
% k: number of iterations performed.
% X: trajectory of the iterates.
% Example: [xs,fs,k,X] = prob15_4([3 2 3]',[1 1]',20,1,0.5,1e-6);
function [xs,fs,k,X] = prob15_4(x0,lam0,eta,tau,rou,epsi)
xk = x0(:);
```

```
lam1k = lam0(1);
lam2k = lam0(2);
Zk = eye(3,3);
k = 0;
X = xk;
f = xk(1)^4 + 2*xk(2)^4 + xk(3)^4 - (xk(1)^2)*(xk(2)^2) -
(xk(1)^2)*(xk(3)^2);
err = 1;
while err >= epsi
    xk1 = xk(1);
    xk2 = xk(2);
    xk3 = xk(3);
    gf1 = 4*xk1^3 - 2*xk1*(xk2^2) - 2*xk1*(xk3^2);
    gf2 = 8*xk2^3 - 2*(xk1^2)*xk2;
    gf3 = 4*xk3^3 - 2*(xk1^2)*xk3;
    gf = [gf1, gf2, gf3]';
    ak1 = xk1^4 + xk2^4 + xk3^4 - 25;
    ak2 = 8*xk1^2 + 14*xk2^2 + 7*xk3^2 - 56;
    ga11 = 4*xk1^3;
    ga12 = 4*xk2^3;
    ga13 = 4*xk3^3;
    ga21 = 16*xk1;
    ga22 = 28*xk2;
    ga23 = 14*xk3;
    Ae1 = [ga11, ga12, ga13];
    Ae2 = [ga21, ga22, ga23];
    cvx_begin quiet
      variable dt(3,1)
      variable u1(1,1)
      variable v1(1,1)
      variable u2(1,1)
      variable v2(1,1)
      variable r(1,1)
      dual variable y1
      dual variable y2
      minimize(0.5*dt'*Zk*dt + gf'*dt + eta*(u1+v1+u2+v2+r));
      subject to
      y1:ak1 + Ae1*dt == u1 - v1;
      y2:ak2 + Ae2*dt == u2 - v2;
      norm(dt) <= rou + r;
      u1 >= 0;
      v1 >= 0;
      u2 >= 0;
      v2 >= 0;
      r >= 0;
    cvx_end
```

```
nuvw = norm([u1, v1, u2, v2, r]);
if nuvw <= 1e-6
   dtk = dt;
   lam1_qp = y1;
   lam2_qp = y2;
else
   ak1 = ak1 - u1 + v1;
   ak2 = ak2 - u2 + v2;
   rou_t = rou + r;
   cvx_begin quiet
     variable dt(3,1)
     dual variable y1
     dual variable y2
     minimize(0.5*dt'*Zk*dt + gf'*dt);
     subject to
     y1: Ae1*dt == -ak1;
     y2: Ae2*dt == -ak2;
     norm(dt) <= rou_t;
   cvx_end
   dtk = dt;
   lam1_qp = y1;
   lam2_qp = y2;
end
dlam1 = lam1_qp - lam1k;
dlam2 = lam2_qp - lam2k;
Kt = 40;
alf = 0:1/(Kt-1):1;
psi = zeros(Kt,1);
for i = 1:Kt
    alfi = alf(i);
    xw = xk + alfi*dtk;
    x1 = xw(1);
    x2 = xw(2);
    x3 = xw(3);
    t1 = x1^4 + 2*x2^4 + x3^4 - (x1^2)*(x2^2) - (x1^2)*(x3^2);
    t2 = abs(x1^4 + x2^4 + x3^4 - 25);
    t3 = abs(8*x1^2 + 14*x2^2 + 7*x3^2 - 56);
    psi(i) = t1 + tau*(t2+t3);
end
[~,ind] = min(psi);
alfk = alf(ind);
xnew = xk + alfk*dtk;
lam1k1= lam1k + alfk*dlam1;
lam2k1= lam2k + alfk*dlam2;
x1 = xnew(1);
x2 = xnew(2);
```

```
    x3 = xnew(3);
    gf1 = 4*x1^3 - 2*x1*(x2^2) - 2*x1*(x3^2);
    gf2 = 8*x2^3 - 2*(x1^2)*x2;
    gf3 = 4*x3^3 - 2*(x1^2)*x3;
    gfnew = [gf1, gf2, gf3]';
    ga11 = 4*x1^3;
    ga12 = 4*x2^3;
    ga13 = 4*x3^3;
    ga21 = 16*x1;
    ga22 = 28*x2;
    ga23 = 14*x3;
    Ae1new = [ga11, ga12, ga13];
    Ae2new = [ga21, ga22, ga23];
    gamk = (gfnew - gf) + (Ae1new - Ae1)'*lam1k1 + (Ae2new - Ae2)'*lam2k1;
    ct1 = dtk'*gamk;
    ct2 = dtk'*Zk*dtk;
    if ct1 >= 0.2*ct2
       thet = 1;
    else
       thet = (0.8*ct2)/(ct2 - ct1);
    end
    ht = Zk*dtk;
    ek = thet*gamk + (1-thet)*ht;
    Zk = Zk + (ek*ek')/(dtk'*ek) - (ht*ht')/ct2;
    err = norm([(xnew-xk);(lam1k1-lam1k);(lam2k1-lam2k)]);
    xk = xnew;
    lam1k = lam1k1;
    lam2k = lam2k1;
    fk = xk(1)^4 + 2*xk(2)^4 + xk(3)^4 - (xk(1)^2)*(xk(2)^2) - (xk(1)^2)*(xk(3)^2);
    f = [f fk];
    X = [X xk];
    k = k + 1;
end
xs = xk;
fs = fk;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel({'Iteration';'(b)'},'fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
axis square
% axis([0 k -1 3])
disp('satisfaction of 1st equality constraint:')
xs(1)^4 + xs(2)^4 + xs(3)^4 - 25
disp('satisfaction of 2nd equality constraint:')
```

```
8*xs(1)^2 + 14*xs(2)^2 + 7*xs(3)^2 - 56
```

With $\boldsymbol{\lambda}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \eta = 20, \tau = 1, \rho = 0.5,$ and $\varepsilon = 10^{-6}$, it took the algorithm 40 iterations to converge to a solution

$$\boldsymbol{x}^* = \begin{bmatrix} 1.879654078011085 \\ -0.464094550644822 \\ 1.879203168858130 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = 11.808614461066718$. The two equality constraints are approximately satisfied in the sense that

$$|a_1(\boldsymbol{x}^*)| = 5.115907697472721 \times 10^{-11} \text{ and } |a_2(\boldsymbol{x}^*)| = 7.024425485724350 \times 10^{-11}.$$

For comparison, the objective at the initial point $\boldsymbol{x}_0 = \begin{bmatrix} 3 & 2 & 3 \end{bmatrix}^T$ assumes the value $f(\boldsymbol{x}_0) = 77$ and it violates both constraints. ■

15.5 The MATLAB code solving the problem is given below.

```
% input:
% x0: an initial point in the domain of f(x)
% tau: positive scalar tau (typically tau is a large)
% roh: initial value of roh the defines trust region |x_i| <= roh
% K: number of iterations (number of convex sub-problems to be solved).
% output:
% xs: local minimizer obtained.
% fs: objective function at xs.
% X: trajectory of the iterates.
% Example:
% x0 = [-1.71 1.59 1.82 -0.763 -0.763]';
% [xs,fs,X] = prob15_5(x0,1e4,0.001,30);
function [xs,fs,X] = prob15_5(x0,tau,roh,K)
xk = x0(:);
e = ones(5,1);
k = 0;
X = xk;
x1 = xk(1);
x2 = xk(2);
x3 = xk(3);
x4 = xk(4);
x5 = xk(5);
f = exp(x1*x2*x3*x4*x5) - 0.5*(x1^3+x2^3+1)^2;
r = roh;
a = 0.1; bs = 1.1; bf = 0.5;
```

```
while k < K
    x1 = xk(1); x2 = xk(2); x3 = xk(3); x4 = xk(4); x5 = xk(5);
    x12 = x1*x2; x23 = x2*x3; x45 = x4*x5;
    x123 = x12*x3; x124 = x12*x4; x234 = x23*x4; x345 = x3*x45;
    x1234 = x1*x234; x1235 = x123*x5; x2345 = x2*x345; x1345 = x1*x345; x1245 = x124*x5;
    x12345 = x1234*x5; x33 = x1^3 + x2^3 + 1;
    ew = exp(x12345);
    gf1 = x2345*ew - 3*(x1^2)*x33;
    gf2 = x1345*ew - 3*(x2^2)*x33;
    gf3 = x1245*ew;
    gf4 = x1235*ew;
    gf5 = x1234*ew;
    gf = [gf1 gf2 gf3 gf4 gf5]';
    Hw = h_prob15_5(xk);
    [V,Dw] = eig(Hw);
    dw = diag(Dw);
    dww = max(dw,1e-2);
    Dww = diag(dww);
    H = V*Dww*V';
    a1k = x1^2 + x2^2 + x3^2 + x4^2 + x5^2 - 11;
    g1 = 2*xk;
    a2k = x2*x3 - 5*x4*x5;
    g2 = [0, x3, x2, -5*x5, -5*x4]';
    a3k = x1^3 + x2^3 + 1;
    g31 = 3*x1^2;
    g32 = 3*x2^2;
    g3 = [g31 g32 0 0 0]';
    F1 = f(k+1);
    F2 = abs(a1k);
    F3 = abs(a2k);
    F4 = abs(a3k);
    F = F1 + tau*(F2+F3+F4);
    cvx_begin quiet
      variable x(5,1)
      u1 = abs(a1k + g1'*(x-xk));
      u2 = abs(a2k + g2'*(x-xk));
      u3 = abs(a3k + g3'*(x-xk));
      minimize(0.5*(x-xk)'*H*(x-xk)+gf'*(x-xk)+tau*(u1+u2+u3));
      subject to
      -r*e + xk <= x <= r*e + xk;
    cvx_end
    xt = x;
    xt1 = xt(1); xt2 = xt(2); xt3 = xt(3); xt4 = xt(4); xt5 = xt(5);
    Ft1 = exp(xt1*xt2*xt3*xt4*xt5) - 0.5*(xt1^3+xt2^3+1)^2;
    Ft2 = abs(xt1^2 + xt2^2 + xt3^2 + xt4^2 + xt5^2 - 11);
    Ft3 = abs(xt2*xt3 - 5*xt4*xt5);
```

```
    Ft4 = abs(xt1^3 + xt2^3 + 1);
    Ft = Ft1 + tau*(Ft2+Ft3+Ft4);
    Ftt1 = abs(a1k + g1'*(xt-xk));
    Ftt2 = abs(a2k + g2'*(xt-xk));
    Ftt3 = abs(a3k + g3'*(xt-xk));
    Ftt = 0.5*(xt-xk)'*H*(xt-xk)+gf'*(xt-xk)+tau*(Ftt1+Ftt2+Ftt3);
    dt = F - Ftt;
    dtt = F - Ft;
    if dt >= a*dtt
       xk = xt;
       r = bs*r;
    elseif dt < a*dtt
       r = bf*r;
    end
    fk = exp(xk(1)*xk(2)*xk(3)*xk(4)*xk(5)) - 0.5*(xk(1)^3+xk(2)^3+1)^2;
    f = [f fk];
    X = [X xk];
    k = k + 1;
end
xs = xk;
fs = exp(xs(1)*xs(2)*xs(3)*xs(4)*xs(5)) - 0.5*(xs(1)^3+xs(2)^3+1)^2;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel('Iteration','fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
axis square
% axis([0 k 0 80])
disp('satisfaction of equality 1st constraint:')
xs(1)^2 + xs(2)^2 + xs(3)^2 + xs(4)^2 + xs(5)^2 - 11
disp('satisfaction of equality 2nd constraint:')
xs(2)*xs(3) - 5*xs(4)*xs(5)
disp('satisfaction of equality 3rd constraint:')
xs(1)^3 + xs(2)^3 + 1
```

With $\tau = 10^4$ and $\rho = 0.001$, it took the algorithm 30 iterations to converge to the solution

$$\boldsymbol{x}^* = \begin{bmatrix} -1.791598380790630 \\ 1.681072746136753 \\ 1.917056605728193 \\ -0.800982940913075 \\ -0.804685802998241 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = 0.024198952996234$. The three

equality constraints are approximately satisfied in the sense of

$$|a_1(\boldsymbol{x}^*)| = 2.927860215606870\times 10^{-5},$$

$$|a_2(\boldsymbol{x}^*)| = 1.360770844360815\times 10^{-5},$$

$$|a_3(\boldsymbol{x}^*)| = 4.118129420582761\times 10^{-6}.$$

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 0.055900151864083$ and it violates the first constraint as $|a_1(\boldsymbol{x}^*)| = 1.071062$. ■

15.6 The MATLAB code solving the problem is given below.

```
% input:
% [x0,lam0]: initial point
% tau: weight in merit function for constraint violation
% mu: weight for penalty term
% rou: bound on \delta
% epsi: tolerance for convergence
% output:
% xs: local minimizer obtained.
% fs: objective function at xs.
% k: number of iterations performed.
% X: trajectory of the iterates.
% Example:  x0 = [-1.71 1.59 1.82 -0.763 -0.763]';
% [xs,fs,k,X] = prob15_6(x0,ones(5,1),20,1,1,1e-6);
function [xs,fs,k,X] = prob15_6(x0,lam0,eta,tau,rou,epsi)
xk = x0(:);
lam1k = lam0(1);
lam2k = lam0(2);
lam3k = lam0(3);
Zk = eye(5,5);
k = 0;
X = xk;
f = exp(xk(1)*xk(2)*xk(3)*xk(4)*xk(5)) - 0.5*(xk(1)^3+xk(2)^3+1)^2;
err = 1;
while err >= epsi
    x1 = xk(1); x2 = xk(2); x3 = xk(3); x4 = xk(4); x5 = xk(5);
    x12 = x1*x2; x23 = x2*x3; x45 = x4*x5;
    x123 = x12*x3; x124 = x12*x4; x234 = x23*x4; x345 = x3*x45;
    x1234 = x1*x234; x1235 = x123*x5; x2345 = x2*x345; x1345 = x1*x345; x1245 = x124*x5;
    x12345 = x1234*x5; x33 = x1^3 + x2^3 + 1;
    ew = exp(x12345);
    gf1 = x2345*ew - 3*(x1^2)*x33;
    gf2 = x1345*ew - 3*(x2^2)*x33;
```

```
gf3 = x1245*ew;
gf4 = x1235*ew;
gf5 = x1234*ew;
gf = [gf1 gf2 gf3 gf4 gf5]';
a1k = x1^2 + x2^2 + x3^2 + x4^2 + x5^2 - 11;
Ae1 = 2*xk';
a2k = x2*x3 - 5*x4*x5;
Ae2 = [0, x3, x2, -5*x5, -5*x4];
a3k = x1^3 + x2^3 + 1;
g31 = 3*x1^2;
g32 = 3*x2^2;
Ae3 = [g31 g32 0 0 0];
cvx_begin quiet
  variable dt(5,1)
  variable u1(1,1)
  variable v1(1,1)
  variable u2(1,1)
  variable v2(1,1)
  variable u3(1,1)
  variable v3(1,1)
  variable r(1,1)
  dual variable y1
  dual variable y2
  dual variable y3
  minimize(0.5*dt'*Zk*dt + gf'*dt + eta*(u1+v1+u2+v2+u3+v3+r));
  subject to
  y1:a1k + Ae1*dt == u1 - v1;
  y2:a2k + Ae2*dt == u2 - v2;
  y3:a3k + Ae3*dt == u3 - v3;
  norm(dt) <= rou + r;
  u1 >= 0;
  v1 >= 0;
  u2 >= 0;
  v2 >= 0;
  u3 >= 0;
  v3 >= 0;
  r >= 0;
cvx_end
nuvw = norm([u1, v1, u2, v2, u3, v3, r]);
if nuvw <= 1e-6
   dtk = dt;
   lam1_qp = y1;
   lam2_qp = y2;
   lam3_qp = y3;
else
   a1k = a1k - u1 + v1;
```

```
   a2k = a2k - u2 + v2;
   a3k = a3k - u3 + v3;
   rou_t = rou + r;
   cvx_begin quiet
     variable dt(5,1)
     dual variable y1
     dual variable y2
     dual variable y3
     minimize(0.5*dt'*Zk*dt + gf'*dt);
     subject to
     y1: Ae1*dt == -a1k;
     y2: Ae2*dt == -a2k;
     y3: Ae3*dt == -a3k;
     norm(dt) <= rou_t;
   cvx_end
   dtk = dt;
   lam1_qp = y1;
   lam2_qp = y2;
   lam3_qp = y3;
end
dlam1 = lam1_qp - lam1k;
dlam2 = lam2_qp - lam2k;
dlam3 = lam3_qp - lam3k;
Kt = 40;
alf = 0:1/(Kt-1):1;
psi = zeros(Kt,1);
for i = 1:Kt
    alfi = alf(i);
    xw = xk + alfi*dtk;
    x1 = xw(1); x2 = xw(2); x3 = xw(3); x4 = xw(4); x5 = xw(5);
    t1 = exp(x1*x2*x3*x4*x5) - 0.5*(x1^3+x2^3+1)^2;
    t2 = abs(x1^2 + x2^2 + x3^2 + x4^2 + x5^2 - 11);
    t3 = abs(x2*x3 - 5*x4*x5);
    t4 = abs(x1^3 + x2^3 + 1);
    psi(i) = t1 + tau*(t2+t3+t4);
end
[~,ind] = min(psi);
alfk = alf(ind);
xnew = xk + alfk*dtk;
lam1k1= lam1k + alfk*dlam1;
lam2k1= lam2k + alfk*dlam2;
lam3k1= lam3k + alfk*dlam3;
x1 = xnew(1); x2 = xnew(2); x3 = xnew(3); x4 = xnew(4); x5 = xnew(5);
x12 = x1*x2; x23 = x2*x3; x45 = x4*x5;
x123 = x12*x3; x124 = x12*x4; x234 = x23*x4; x345 = x3*x45;
x1234 = x1*x234; x1235 = x123*x5; x2345 = x2*x345; x1345 = x1*x345; x1245 = x124*x5;
```

```
    x12345 = x1234*x5; x33 = x1^3 + x2^3 + 1;
    ew = exp(x12345);
    gf1 = x2345*ew - 3*(x1^2)*x33;
    gf2 = x1345*ew - 3*(x2^2)*x33;
    gf3 = x1245*ew;
    gf4 = x1235*ew;
    gf5 = x1234*ew;
    gfnew = [gf1 gf2 gf3 gf4 gf5]';
    Ae1new = 2*xnew';
    Ae2new = [0, x3, x2, -5*x5, -5*x4];
    g31 = 3*x1^2;
    g32 = 3*x2^2;
    Ae3new = [g31 g32 0 0 0];
    gamk = (gfnew - gf) + (Ae1new - Ae1)'*lam1k1 + (Ae2new - Ae2)'*lam2k1 + (Ae3new - Ae3)'*lam3k1;
    ct1 = dtk'*gamk;
    ct2 = dtk'*Zk*dtk;
    if ct1 >= 0.2*ct2
       thet = 1;
    else
       thet = (0.8*ct2)/(ct2 - ct1);
    end
    ht = Zk*dtk;
    ek = thet*gamk + (1-thet)*ht;
    Zk = Zk + (ek*ek')/(dtk'*ek) - (ht*ht')/ct2;
    err = norm([(xnew-xk);(lam1k1-lam1k);(lam2k1-lam2k);(lam3k1-lam3k)]);
    xk = xnew;
    lam1k = lam1k1;
    lam2k = lam2k1;
    lam3k = lam3k1;
    fk = exp(xk(1)*xk(2)*xk(3)*xk(4)*xk(5)) - 0.5*(xk(1)^3+xk(2)^3+1)^2;
    f = [f fk];
    X = [X xk];
    k = k + 1;
end
xs = xk;
fs = fk;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel({'Iteration';'(b)'},'fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
axis square
disp('satisfaction of 1st equality constraint:')
xs(1)^2 + xs(2)^2 + xs(3)^2 + xs(4)^2 + xs(5)^2 - 11
disp('satisfaction of 2nd equality constraint:')
```

```
xs(2)*xs(3) - 5*xs(4)*xs(5)
disp('satisfaction of 3rd equality constraint:')
xs(1)^3 + xs(2)^3 + 1
```

With $\boldsymbol{\lambda}_0 = \texttt{ones(5,1)}, \eta = 20, \tau = 1, \rho = 1,$ and $\varepsilon = 10^{-6}$, it took the algorithm 6 iterations to converge to a solution

$$\boldsymbol{x}^* = \begin{bmatrix} -1.791693753813294 \\ 1.681180586691481 \\ 1.916877305407775 \\ -0.802822138645124 \\ -0.802822138643495 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = 0.024199071051650$. The three equality constraints are approximately satisfied in the sense that

$$|a_1(\boldsymbol{x}^*)| = 4.910410389413755 \times 10^{-8},$$

$$|a_2(\boldsymbol{x}^*)| = 1.856608333739018 \times 10^{-8},$$

$$|a_3(\boldsymbol{x}^*)| = 9.850473325911935 \times 10^{-9}.$$

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 0.055900151864083$ and it violates the first constraint as $|a_1(\boldsymbol{x}^*)| = 1.071062$. ■

15.7

(**a**) Note that the constraint defines a convex feasible region, but the objective function is not convex. To apply Algorithm 15.7, we write the objective function as $f(\boldsymbol{x}) = u_0(\boldsymbol{x}) - v_0(\boldsymbol{x})$ where both $u_0(\boldsymbol{x}) = (x_1 - x_2)^2 = \boldsymbol{x}^T \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \boldsymbol{x}$ and $v_0(\boldsymbol{x}) = x_1^2 + x_2^2 = \boldsymbol{x}^T \boldsymbol{x}$ are convex. It follows that

$$\hat{v}_0(\boldsymbol{x}, \boldsymbol{x}_k) = \boldsymbol{x}_k^T \boldsymbol{x}_k + 2\boldsymbol{x}_k^T (\boldsymbol{x} - \boldsymbol{x}_k)$$

and the CP problem in Eq. (15.27) becomes

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{x}^T \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \boldsymbol{x} - 2\boldsymbol{x}_k^T (\boldsymbol{x} - \boldsymbol{x}_k) \\ \text{subject to:} \quad & 0.2x_1^2 + 0.4x_2^2 - 1 \le 0 \end{aligned}$$

The MATLAB code that applies Algorithm 15.7 to solve the problem at hand is given below.

```
% input:
% x0: initial point
```

```
% epsi: convergence tolerance
% output:
% xs: solution obtained.
% fs: objective function at xs.
% k: numver of iterations performed.
% Examples:
% [xs,fs,k] = prob15_7([0 0]',1e-6);
% [xs,fs,k] = prob15_7([-2 -2]',1e-6);
function [xs,fs,k] = prob15_7(x0,epsi)
H = [1 -1; -1 1];
xk = x0;
fk = -2*xk(1)*xk(2);
f = fk;
err = 1;
k = 0;
while err >= epsi
    cvx_begin quiet
      variable x(2,1)
      minimize(x'*H*x - 2*xk'*(x-xk));
      subject to
      0.2*x(1)^2 + 0.4*x(2)^2 - 1 <= 0;
    cvx_end
    xnew = x;
    fnew = -2*xnew(1)*xnew(2);
    err = abs((fnew-fk)/fk);
    xk = xnew;
    fk = fnew;
    f = [f fk];
    k = k + 1;
    current = [k fk err]
end
xs = xk;
fs = fk;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-')
xlabel({'Iteration'},'fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itX,Y}{\rm)}','fontsize',14)
grid
disp('satisfaction of the inequality constraint:')
0.2*xs(1)^2 + 0.4*xs(2)^2 - 1
```

With $\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $\varepsilon = 10^{-6}$, it took the algorithm 12 iterations to converge to the solution

$$\boldsymbol{x}^* = \begin{bmatrix} -1.580961095193760 \\ -1.118159648412428 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = -3.535533804711164$. The inequality constraint is satisfied as $c_1(\boldsymbol{x}^*) = -3.361627642917142 \times 10^{-9} < 0$.

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 0$ while it satisfies the constraint as $c_1(\boldsymbol{x}_0) = -1 < 0$.

(**b**) The MATLAB code used in part (a) also applies. With $\boldsymbol{x}_0 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$ and $\varepsilon = 10^{-6}$, it took the algorithm 12 iterations to converge to the solution

$$\boldsymbol{x}^* = \begin{bmatrix} -1.580923979863377 \\ -1.118185886401130 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = -3.535533763512667$. The inequality constraint is satisfied as $c_1(\boldsymbol{x}^*) = -3.359915012879355 \times 10^{-9} < 0$.

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = -8$, but it violates the constraint as $c_1(\boldsymbol{x}_0) = 1.4 > 0$. ■

15.8 Note that the objective function $f(\boldsymbol{x})$ and constraint function $c_1(\boldsymbol{x})$ are convex, but constraint function $c_2(\boldsymbol{x})$ is not. To apply Algorithm 15.8, we write $c_2(\boldsymbol{x})$ as $c_2(\boldsymbol{x}) = u_2(\boldsymbol{x}) - v_2(\boldsymbol{x})$ where $u_2(\boldsymbol{x}) = x_1$ and $v_2(\boldsymbol{x}) = 2x_2^2$ are convex. It follows that

$$\hat{v}_2(\boldsymbol{x}, \boldsymbol{x}_k) = 2(x_2^{(k)})^2 + 4x_2^{(k)}(x_2 - x_2^{(k)})$$

and the CP problem in Eq. (15.31) becomes

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) = (x_1 - 1)^2 + (x_2 - 0.5)^2 + \tau_k(s(1) + s(2)) \\ \text{subject to:} \quad & 2x_1^2 - x_2 \le s(1) \\ & x_1 \le 2(x_2^{(k)})^2 + 4x_2^{(k)}(x_2 - x_2^{(k)}) + s(2) \\ & s(1) \ge 0, s(2) \ge 0 \end{aligned}$$

The MATLAB code that applies Algorithm 15.8 to solve the problem at hand is given below.

```
% input:
% x0: initial point.
```

```
% t0: initial value of parameter tau
% tmax: maximum value of tau
% mu: parameter mu
% epsi: convergence tolerance
% output:
% xs: local minimizer obtained
% fs: objective function at xs
% k: number of iterations performed
% Example: [xs,fs,k] = prob15_8([0.2 -0.2]',1,1e3,1.5,1e-9);
function [xs,fs,k] = prob15_8(x0,t0,tmax,mu,epsi)
xk = x0(:);
tk = t0;
fk = (xk(1) - 1)^2 + (xk(2) - 0.5)^2;
f = fk;
X = xk;
k = 0;
err = 1;
while err >= epsi
    cvx_begin quiet
      variable x(2,1)
      variable s(2,1)
      u = x - [1, 0.5]';
      minimize(u'*u + tk*sum(s))
      subject to
      2*x(1)^2 - x(2) <= s(1);
      x(1) <= 2*xk(2)^2  + 4*xk(2)*(x(2) - xk(2)) + s(2);
      s >= 0;
    cvx_end
    xnew = x;
    fnew = (xnew(1) - 1)^2 + (xnew(2) - 0.5)^2;
    err = abs((fnew - fk)/fk);
    xk = xnew;
    fk = fnew;
    f = [f fk];
    X = [X x];
    tk = min([mu*tk tmax]);
    k = k + 1;
end
xs = xk;
fs = (xs(1) - 1)^2 + (xs(2) - 0.5)^2;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel('Iteration','fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
```

```
axis square
disp('satisfaction of 1st inequality constraint:')
2*xs(1)^2 - xs(2)
disp('satisfaction of 2nd inequality constraint:')
xs(1) - 2*xs(2)^2
```

With $\boldsymbol{x}_0 = \begin{bmatrix} 0.2 \\ -0.2 \end{bmatrix}$, $\tau_0 = 1$, $\tau_{\max} = 10^3$, $\mu = 1.5$, and $\varepsilon = 10^{-9}$, it took the algorithm 5 iterations to converge to the solution

$$\boldsymbol{x}^* = \begin{bmatrix} 0.582685505564819 \\ 0.679044797472119 \end{bmatrix}$$

at which $f(\boldsymbol{x}^*) = 0.206208426767523$. The two inequality constraints are satisfied as $c_1(\boldsymbol{x}^*) = -6.814612207861614 \times 10^{-10}$ and $c_2(\boldsymbol{x}^*) = -0.339518168383082$.

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 1.13$, and it violates both constraints as $c_1(\boldsymbol{x}_0) = 0.28$ and $c_2(\boldsymbol{x}_0) = 0.12$. ■

15.9 Note that constraint function $c_2(\boldsymbol{x})$ is convex, but the objective function $f(\boldsymbol{x})$ and constraint function $c_1(\boldsymbol{x})$ are not. To apply Algorithm 15.8, we write $f(\boldsymbol{x})$ as $f(\boldsymbol{x}) = u_0(\boldsymbol{x}) - v_0(\boldsymbol{x})$ where $u_0(\boldsymbol{x}) = x_1^2 + 4x_1x_2 + 4x_2^2$, $v_0(\boldsymbol{x}) = x_2^2$ and write $c_1(\boldsymbol{x})$ as $c_1(\boldsymbol{x}) = u_1(\boldsymbol{x}) - v_1(\boldsymbol{x})$ where $u_1(\boldsymbol{x}) = -x_2$ and $v_1(\boldsymbol{x}) = x_1^2$. It follows that

$$\hat{v}_0(\boldsymbol{x}, \boldsymbol{x}_k) = (x_2^{(k)})^2 + 2x_2^{(k)}(x_2 - x_2^{(k)})$$

and

$$\hat{v}_1(\boldsymbol{x}, \boldsymbol{x}_k) = (x_1^{(k)})^2 + 2x_1^{(k)}(x_1 - x_1^{(k)})$$

and the CP problem in Eq. (15.31) becomes

$$\begin{aligned} \text{minimize} \quad & x_1^2 + 4x_1x_2 + 4x_2^2 - 2x_2^{(k)}x_2 + \tau_k(s(1) + s(2)) \\ \text{subject to:} \quad & -x_2 - (x_1^{(k)})^2 - 2x_1^{(k)}(x_1 - x_1^{(k)}) \le s(1) \\ & x_1^2 + x_2^2 - 1 \le s(2) \\ & s(1) \ge 0, s(2) \ge 0 \end{aligned}$$

The MATLAB code that applies Algorithm 15.8 to solve the problem at hand is given below.

```
% input:
% x0: initial point.
% t0: initial value of parameter tau
```

```
% tmax: maximum value of tau
% mu: parameter mu
% epsi: convergence tolerance
% output:
% xs: local minimizer obtained
% fs: objective function at xs
% k: number of iterations performed
% Example: [xs,fs,k] = prob15_9([-0.5 4]',5,1e3,1.5,1e-9);
function [xs,fs,k] = prob15_9(x0,t0,tmax,mu,epsi)
xk = x0(:);
tk = t0;
fk = xk(1)^2 + 4*xk(1)*xk(2) + 3*xk(2)^2;
f = fk;
X = xk;
k = 0;
err = 1;
H = [1 2; 2 4];
while err >= epsi
    cvx_begin quiet
      variable x(2,1)
      variable s(2,1)
      v = [0, 2*xk(2)]';
      minimize(x'*H*x - v'*x + tk*sum(s))
      subject to
      -x(2) - xk(1)^2 - [2*xk(1) 0]*(x - xk) <= s(1);
      x(1)^2 + x(2)^2 - 1 <= s(2);
      s >= 0;
    cvx_end
    xnew = x;
    fnew = xnew(1)^2 + 4*xnew(1)*xnew(2) + 3*xnew(2)^2;
    err = abs((fnew - fk)/fk);
    xk = xnew;
    fk = fnew;
    f = [f fk];
    X = [X x];
    tk = min([mu*tk tmax]);
    k = k + 1;
end
xs = xk;
fs = xs(1)^2 + 4*xs(1)*xs(2) + 3*xs(2)^2;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel('Iteration','fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
```

```
axis square
disp('satisfaction of 1st inequality constraint:')
-xs(1)^2 + xs(2)
disp('satisfaction of 2nd inequality constraint:')
xs(1)^2 + xs(2)^2 - 1
```

With $\boldsymbol{x}_0 = \begin{bmatrix} -0.5 \\ 4 \end{bmatrix}$, $\tau_0 = 5$, $\tau_{\max} = 10^3$, $\mu = 1.5$, and $\varepsilon = 10^{-9}$, it took the algorithm 16 iterations to converge to the solution

$$\boldsymbol{x}^* = \begin{bmatrix} -0.850646391873034 \\ 0.525738238616289 \end{bmatrix}$$

at which $f(\boldsymbol{x}^*) = -0.236067972357836$. The two inequality constraints are satisfied as $c_1(\boldsymbol{x}^*) = -0.197861045390321$ and $c_2(\boldsymbol{x}^*) = -2.045003077988383 \times 10^{-8}$.

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 40.25$, and it violates both constraints as $c_1(\boldsymbol{x}_0) = 3.75$ and $c_2(\boldsymbol{x}_0) = 15.25$. ■

15.10 The Lagrangian of the QP problem in Eq. (15.16) is given by

$$L(\boldsymbol{\delta}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{\delta}^T \boldsymbol{Z}_k \boldsymbol{\delta} + \boldsymbol{\delta}^T \boldsymbol{g}_k + \boldsymbol{\lambda}^T (\boldsymbol{A}_{ek}\boldsymbol{\delta} + \boldsymbol{a}_k) + \boldsymbol{\mu}^T (\boldsymbol{A}_{ik}\boldsymbol{\delta} + \boldsymbol{c}_k)$$

which allows us to write the KKT conditions of the problem as

$$\boldsymbol{A}_{ek}\boldsymbol{\delta} = -\boldsymbol{a}_k$$

$$\boldsymbol{A}_{ik}\boldsymbol{\delta} \le -\boldsymbol{c}_k$$

$$\nabla_{\boldsymbol{\delta}} L(\boldsymbol{\delta}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \boldsymbol{Z}_k \boldsymbol{\delta} + \boldsymbol{g}_k + {\boldsymbol{A}_{ek}}^T \boldsymbol{\lambda} + {\boldsymbol{A}_{ik}}^T \boldsymbol{\mu} = \boldsymbol{0}$$

$$\boldsymbol{\mu} \ge \boldsymbol{0}$$

$$(\boldsymbol{\mu})_j (\boldsymbol{A}_{ik}\boldsymbol{\delta} + \boldsymbol{c}_k)_j = 0 \quad \text{for } j = 1, 2, \ldots, q$$

The solution of the above equations, $\{\boldsymbol{\delta}_k, \boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1}\}$, matches with that of Eq. (15.14). ■

15.11 The MATLAB code solving the problem is given below.

```
% input:
% [x0,mu0]: initial point
% tau: weight in merit function for constraint violation
% mu: weight for penalty term
% rou: bound on \delta
% epsi: tolerance for convergence
```

```
% output:
% xs: local minimizer obtained.
% fs: objective function at xs.
% k: number of iterations performed.
% X: trajectory of the iterates.
% Example:[xs,fs,k,X] = prob15_11([15 5]',[1 1 1]',20,1,1,1e-6);
function [xs,fs,k,X] = prob15_11(x0,mu0,eta,tau,rou,epsi)
xk = x0(:);
mu1k = mu0(1);
mu2k = mu0(2);
mu3k = mu0(3);
Zk = eye(2,2);
k = 0;
X = xk;
f = 0.01*xk(1)^2 + xk(2)^2;
err = 1;
while err >= epsi
    xk1 = xk(1);
    xk2 = xk(2);
    gf1 = 0.02*xk1;
    gf2 = 2*xk2;
    gf = [gf1, gf2]';
    c1k = xk1*xk2 - 25;
    Ac1 = [xk2, xk1];
    c2k = -xk1^2 - xk2^2 + 25;
    Ac2 = -2*xk';
    c3k = -xk1 + 2;
    Ac3 = [-1, 0];
    cvx_begin quiet
      variable dt(2,1)
      variable u(1,1)
      variable v(1,1)
      variable w(1,1)
      variable r(1,1)
      dual variable y1
      dual variable y2
      dual variable y3
      minimize(0.5*dt'*Zk*dt + gf'*dt + eta*(u+v+w+r));
      subject to
      y1:c1k + Ac1*dt <= u;
      y2:c2k + Ac2*dt <= v;
      y3:c3k + Ac3*dt <= w;
      norm(dt) <= rou + r;
      u >= 0;
      v >= 0;
      w >= 0;
```

```
  r >= 0;
cvx_end
nuvw = norm([u, v, w, r]);
if nuvw <= 1e-6
   dtk = dt;
   mu1_qp = y1;
   mu2_qp = y2;
   mu3_qp = y3;
else
   c1k = c1k - u;
   c2k = c2k - v;
   c3k = c3k - w;
   rou_t = rou + r;
   cvx_begin quiet
     variable dt(2,1)
     dual variable y1
     dual variable y2
     dual variable y3
     minimize(0.5*dt'*Zk*dt + gf'*dt);
     subject to
     y1: Ac1*dt <= -c1k;
     y2: Ac2*dt <= -c2k;
     y3: Ac3*dt <= -c3k;
     norm(dt) <= rou_t;
   cvx_end
   dtk = dt;
   mu1_qp = y1;
   mu2_qp = y2;
   mu3_qp = y3;
end
dmu1 = mu1_qp - mu1k;
dmu2 = mu2_qp - mu2k;
dmu3 = mu3_qp - mu3k;
Kt = 40;
alf = 0:1/(Kt-1):1;
psi = zeros(Kt,1);
for i = 1:Kt
    alfi = alf(i);
    xw = xk + alfi*dtk;
    x1 = xw(1);
    x2 = xw(2);
    t1 = 0.01*x1^2 + x2^2;
    t2 = max(0,x1*x2-25);
    t3 = max(0,-x1^2-x2^2+25);
    t4 = max(0,-x1+2);
    psi(i) = t1 + tau*(t2+t3+t4);
```

```
    end
    [~,ind] = min(psi);
    alfk = (ind - 1)/(Kt-1);
    xnew = xk + alfk*dtk;
    mu1k1 = mu1k + alfk*dmu1;
    mu2k1 = mu2k + alfk*dmu2;
    mu3k1 = mu3k + alfk*dmu3;
    x1 = xnew(1);
    x2 = xnew(2);
    gf1 = 0.02*x1;
    gf2 = 2*x2;
    gfnew = [gf1, gf2]';
    Ac1new = [x2, x1];
    Ac2new = -2*xnew';
    Ac3new = [-1, 0];
    gamk = (gfnew-gf)+(Ac1new-Ac1)'*mu1k1+(Ac2new-Ac2)'*mu2k1+(Ac3new-Ac3)'*mu3k1;
    ct1 = dtk'*gamk;
    ct2 = dtk'*Zk*dtk;
    if ct1 >= 0.2*ct2
       thet = 1;
    else
       thet = (0.8*ct2)/(ct2 - ct1);
    end
    ht = Zk*dtk;
    ek = thet*gamk + (1-thet)*ht;
    Zk = Zk + (ek*ek')/(dtk'*ek) - (ht*ht')/ct2;
    err = norm([(xnew-xk);(mu1k1-mu1k);(mu2k1-mu2k);(mu3k1-mu3k)]);
    xk = xnew;
    mu1k = mu1k1;
    mu2k = mu2k1;
    mu3k = mu3k1;
    fk = 0.01*xk(1)^2 + xk(2)^2;
    f = [f fk];
    X = [X xk];
    k = k + 1;
end
xs = xk;
fs = fk;
figure(1)
subplot(121)
set(gca,'fontsize',13,'fontname','times')
plot(X(1,:),X(2,:),'ko')
hold on
plot(X(1,:),X(2,:),'kx')
plot(X(1,:),X(2,:),'k.')
plot(X(1,:),X(2,:),'k-','linewidth',1.4)
```

```
hold off
grid
xlabel({'\itx_{\rm1}';'\rm(a)'},'fontsize',14,'fontname','times')
ylabel({'\itx_{\rm2}'},'fontsize',14,'fontname','times')
axis square
axis([-1.6 -0.8 -1.5 1.5])
subplot(122)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel({'Iteration';'(b)'},'fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
axis square
axis([0 k -1 3])
disp('satisfaction of 1st inequality constraint:')
xs(1)*xs(2) - 25
disp('satisfaction of 2nd inequality constraint:')
-xs(1)^2 - xs(2)^2 + 25
disp('satisfaction of 3rd inequality constraint:')
-xs(1) + 2
```

With $\boldsymbol{x}_0 = \begin{bmatrix} 15 \\ 5 \end{bmatrix}, \boldsymbol{\mu}_0 = \texttt{ones(3,1)}, \eta = 20, \tau = 1, \rho = 1,$ and $\varepsilon = 10^{-6}$, it took the algorithm 18 iterations to converge to a solution

$$\boldsymbol{x}^* = \begin{bmatrix} 5.000000004475676 \\ -0.000001739720579 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = 0.250000000450594$. The three inequality constraints are satisfied as

$$c_1(\boldsymbol{x}^*) = -25.000008698602905,$$

$$c_2(\boldsymbol{x}^*) = -4.475978698792460 \times 10^{-8},$$

$$c_3(\boldsymbol{x}^*) = -3.000000004475676.$$

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 27.25$ and is infeasible as it violates the 1st constraint. ■

15.12 Note the objective function $f(\boldsymbol{x})$ and constraint function $c_3(\boldsymbol{x})$ are convex, but $c_1(\boldsymbol{x})$ and $c_2(\boldsymbol{x})$ are not. To apply Algorithm 15.8, we write $c_1(\boldsymbol{x})$ as $c_1(\boldsymbol{x}) = u_1(\boldsymbol{x}) - v_1(\boldsymbol{x})$ where

$$u_1(\boldsymbol{x}) = 0.5x_1^2 + x_1x_2 + 0.5x_2^2 - 25 \text{ and } v_1(\boldsymbol{x}) = 0.5x_1^2 + 0.5x_2^2,$$

and write $c_2(\boldsymbol{x})$ as $c_2(\boldsymbol{x}) = u_2(\boldsymbol{x}) - v_2(\boldsymbol{x})$ where $u_2(\boldsymbol{x}) = 25$ and $v_2(\boldsymbol{x}) = x_1^2 + x_2^2$.

It follows that

$$\hat{v}_1(\boldsymbol{x}, \boldsymbol{x}_k) = 0.5\boldsymbol{x}_k^T\boldsymbol{x}_k + \boldsymbol{x}_k^T(\boldsymbol{x} - \boldsymbol{x}_k)$$

and

$$\hat{v}_2(\boldsymbol{x}, \boldsymbol{x}_k) = \boldsymbol{x}_k^T\boldsymbol{x}_k + 2\boldsymbol{x}_k^T(\boldsymbol{x} - \boldsymbol{x}_k)$$

and the CP problem in Eq. (15.31) becomes

$$\begin{aligned}
\text{minimize} \quad & 0.01x_1^2 + x_2^2 + \tau_k(s(1) + s(2)) \\
\text{subject to:} \quad & 0.5x_1^2 + x_1x_2 + 0.5x_2^2 - 25 \le 0.5\boldsymbol{x}_k^T\boldsymbol{x}_k + \boldsymbol{x}_k^T(\boldsymbol{x} - \boldsymbol{x}_k) + s(1) \\
& 25 \le \boldsymbol{x}_k^T\boldsymbol{x}_k + 2\boldsymbol{x}_k^T(\boldsymbol{x} - \boldsymbol{x}_k) + s(2) \\
& s(1) \ge 0, s(2) \ge 0
\end{aligned}$$

The MATLAB code that applies Algorithm 15.8 to solve the problem at hand is given below.

```
% input:
% x0: initial point.
% t0: initial value of parameter tau
% tmax: maximum value of tau
% mu: parameter mu
% epsi: convergence tolerance
% output:
% xs: local minimizer obtained
% fs: objective function at xs
% k: number of iterations performed
% Example: [xs,fs,k] = prob15_12([15 5]',1,1e3,1.5,1e-9);
function [xs,fs,k] = prob15_12(x0,t0,tmax,mu,epsi)
xk = x0(:);
tk = t0;
fk = 0.01*xk(1)^2 + xk(2)^2;
f = fk;
X = xk;
k = 0;
err = 1;
H = [0.01 0; 0 1];
H1 = 0.5*[1 1; 1 1];
while err >= epsi
    cvx_begin quiet
      variable x(2,1)
      variable s(3,1)
      minimize(x'*H*x + tk*sum(s))
      subject to
```

```
      x'*H1*x - 25 <= 0.5*xk'*xk + xk'*(x-xk) + s(1);
      25 <= xk'*xk + 2*xk'*(x-xk) + s(2);
      -x(1) + 2 <= s(3);
      s >= 0;
    cvx_end
    xnew = x;
    fnew = 0.01*xnew(1)^2 + xnew(2)^2;
    err = abs((fnew - fk)/fk);
    xk = xnew;
    fk = fnew;
    f = [f fk];
    X = [X x];
    tk = min([mu*tk tmax]);
    k = k + 1;
end
xs = xk;
fs = 0.01*xs(1)^2 + xs(2)^2;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel('Iteration','fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
axis square
disp('satisfaction of 1st inequality constraint:')
xs(1)*xs(2) - 25
disp('satisfaction of 2nd inequality constraint:')
-xs(1)^2 - xs(2)^2 + 25
disp('satisfaction of 3rd inequality constraint:')
-xs(1) + 2
```

With $\tau_0 = 1, \tau_{\max} = 10^3, \mu = 1.5,$ and $\varepsilon = 10^{-9}$, it took the algorithm 7 iterations to converge to the solution

$$\boldsymbol{x}^* = \begin{bmatrix} 5.000000006976924 \\ 0.000005524351692 \end{bmatrix}$$

at which $f(\boldsymbol{x}^*) = 0.250000000728211$. The solution satisfies all three constraints as

$$c_1(\boldsymbol{x}^*) = -24.999972378241498,$$

$$c_2(\boldsymbol{x}^*) = -6.979975708532038 \times 10^{-8},$$

$$c_3(\boldsymbol{x}^*) = -3.000000006976924.$$

For comparison, the objective at the initial point $\boldsymbol{x}_0 = \begin{bmatrix} 15 & 5 \end{bmatrix}^T$ assumes the value $f(\boldsymbol{x}_0) = 27.25$ and is infeasible as it violates the 1st constraint. ■

5.13 The MATLAB code solving the problem is given below.

```
% input:
% [x0,lam0,mu0]: initial point
% tau: weight in merit function for constraint violation
% mu: weight for penalty term
% rou: bound on \delta
% epsi: tolerance for convergence
% output:
% xs: local minimizer obtained.
% fs: objective function at xs.
% k: number of iterations performed.
% X: trajectory of the iterates.
% Example:
% [xs,fs,k,X] = prob15_13([6 -2]',1,[1 1 1 1]',20,20,1,1e-6);
function [xs,fs,k,X] = prob15_13(x0,lam0,mu0,eta,tau,rou,epsi)
A = [0, -1; 0, 1; -2, 0; 2, 0];
b = [-1, 5, -2, 4]';
xk = x0(:);
lamk = lam0;
muk = mu0(:);
Zk = eye(2,2);
k = 0;
X = xk;
f = 2*xk(1) + xk(2)^2;
err = 1;
while err >= epsi,
    xk1 = xk(1);
    xk2 = xk(2);
    gf = [2, 2*xk2]';
    ak = 4*xk1^2 + xk2^2 - 9;
    Ae = [8*xk1, 2*xk2];
    ck = A*xk - b;
    cvx_begin quiet
      variable dt(2,1)
      variable u(1,1)
      variable v(1,1)
      variable w(4,1)
      variable r(1,1)
      dual variable y1
      dual variable y2
```

```
  minimize(0.5*dt'*Zk*dt + gf'*dt + eta*(u+v+sum(w)+r));
  subject to
  y1:ak + Ae*dt == u - v;
  y2:ck + A*dt <= w;
  norm(dt) <= rou + r;
  u >= 0;
  v >= 0;
  w >= 0;
  r >= 0;
cvx_end
nuvw = norm([u, v, w', r]);
if nuvw <= 1e-6,
   dtk = dt;
   lam_qp = y1;
   mu_qp = y2;
else
   ak = ak - u + v;
   ck = ck - w;
   rou_t = rou + r;
   cvx_begin quiet
     variable dt(2,1)
     dual variable y1
     dual variable y2
     minimize(0.5*dt'*Zk*dt + gf'*dt);
     subject to
     y1: Ae*dt == -ak;
     y2: A*dt <= -ck;
     norm(dt) <= rou_t;
   cvx_end
   dtk = dt;
   lam_qp = y1;
   mu_qp = y2;
end
dlam = lam_qp - lamk;
dmu = mu_qp - muk;
Kt = 40;
alf = 0:1/(Kt-1):1;
psi = zeros(Kt,1);
for i = 1:Kt,
    alfi = alf(i);
    xw = xk + alfi*dtk;
    x1 = xw(1);
    x2 = xw(2);
    t1 = 2*x1 + x2^2;
    t2 = abs(4*x1^2 + x2^2 - 9);
    t3 = max(0,A*xw-b);
```

```
        psi(i) = t1 + tau*(t2+sum(t3));
    end
    [~,ind] = min(psi);
    alfk = alf(ind);
    xnew = xk + alfk*dtk;
    lamk1 = lamk + alfk*dlam;
    muk1 = muk + alfk*dmu;
    x1 = xnew(1);
    x2 = xnew(2);
    gfnew = [2, 2*x2]';
    Aenew = [8*x1, 2*x2];
    gamk = (gfnew - gf) + (Aenew - Ae)'*lamk1 ;
    ct1 = dtk'*gamk;
    ct2 = dtk'*Zk*dtk;
    if ct1 >= 0.2*ct2
       thet = 1;
    else
       thet = (0.8*ct2)/(ct2 - ct1);
    end
    ht = Zk*dtk;
    ek = thet*gamk + (1-thet)*ht;
    Zk = Zk + (ek*ek')/(dtk'*ek) - (ht*ht')/ct2;
    err = norm([(xnew-xk);(lamk1-lamk);(muk1-muk)]);
    xk = xnew;
    lamk = lamk1;
    muk = muk1;
    fk = 2*xk(1) + xk(2)^2;
    f = [f fk];
    X = [X xk];
    k = k + 1;
end
xs = xk;
fs = fk;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel({'Iteration';'(b)'},'fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
axis square
disp('satisfaction of equality constraint:')
4*xs(1)^2 + xs(2)^2 - 9
disp('satisfaction of inequality constraint:')
A*xs - b
```

With $\boldsymbol{x}_0 = \begin{bmatrix} 6 \\ -2 \end{bmatrix}, \lambda_0 = 1, \boldsymbol{\mu}_0 = \texttt{ones(4,1)}, \eta = 20, \tau = 20, \rho = 1,$ and $\varepsilon = 10^{-6}$, it took the algorithm 7 iterations to converge to a solution

$$\boldsymbol{x}^* = \begin{bmatrix} 1.414213562303672 \\ 1.000000000392651 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = 3.828427125392646$. Satisfaction of the equality and inequality constraints are evaluated in terms of

$$| a_1(\boldsymbol{x}^*) | = 1.278976924368180 \times 10^{-13},$$

$$\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b} = \begin{bmatrix} -0.000000000392651 \\ -3.999999999607350 \\ -0.828427124607344 \\ -1.171572875392656 \end{bmatrix}.$$

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 16$ and is infeasible as

$$| a_1(\boldsymbol{x}_0) | = 139 \quad \text{and} \quad \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b} = \begin{bmatrix} 3 \\ -7 \\ -10 \\ 8 \end{bmatrix} \quad \blacksquare$$

15.14 Note the objective function $f(\boldsymbol{x})$ and linear constraint function $\boldsymbol{Ax} - \boldsymbol{b}$ are convex, but the equality constraint function $a_1(\boldsymbol{x})$ is nonlinear, hence this is a nonconvex problem. Also note that $a_1(\boldsymbol{x}) = 0$ is equivalent to $-a_1(\boldsymbol{x}) \le 0$ and $a_1(\boldsymbol{x}) \le 0$ where $a_1(\boldsymbol{x})$ is convex but $-a_1(\boldsymbol{x})$ is not. To apply Algorithm 15.8, we write $-a_1(\boldsymbol{x}) = u_1(\boldsymbol{x}) - v_1(\boldsymbol{x})$ where

$$u_1(\boldsymbol{x}) = 9 \quad \text{and} \quad v_1(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{H}_1 \boldsymbol{x} \quad \text{with} \quad \boldsymbol{H}_1 = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}.$$

It follows that

$$\hat{v}_1(\boldsymbol{x}, \boldsymbol{x}_k) = \boldsymbol{x}_k^T \boldsymbol{H}_1 \boldsymbol{x}_k + 2\boldsymbol{x}_k^T \boldsymbol{H}_1 (\boldsymbol{x} - \boldsymbol{x}_k)$$

and the CP problem in Eq. (15.31) becomes

$$
\begin{aligned}
\text{minimize} \quad & 2x_1 + x_2^2 + \tau_k \sum_{i=1}^{6} s(i) \\
\text{subject to:} \quad & 9 \le \boldsymbol{x}_k^T \boldsymbol{H}_1 \boldsymbol{x}_k + 2\boldsymbol{x}_k^T \boldsymbol{H}_1 (\boldsymbol{x} - \boldsymbol{x}_k) + s(1) \\
& \boldsymbol{x}^T \boldsymbol{H}_1 \boldsymbol{x} - 9 \le s(2) \\
& \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b} + s(3:6) \\
& s(i) \ge 0 \ \text{ for } \ i = 1, 2, ..., 6
\end{aligned}
$$

The MATLAB code that applies Algorithm 15.8 to solve the problem at hand is given below.

```
% input:
% x0: initial point.
% t0: initial value of parameter tau
% tmax: maximum value of tau
% mu: parameter mu
% epsi: convergence tolerance
% output:
% xs: local minimizer obtained
% fs: objective function at xs
% k: number of iterations performed
% Example: [xs,fs,k] = prob15_14([6 -2]',1,1e3,1.5,1e-6);
function [xs,fs,k] = prob15_14(x0,t0,tmax,mu,epsi)
A = [0, -1; 0, 1; -2, 0; 2, 0];
b = [-1, 5, -2, 4]';
xk = x0(:);
tk = t0;
fk = 2*xk(1) + xk(2)^2;
f = fk;
X = xk;
k = 0;
err = 1;
H1 = [4 0; 0 1];
while err >= epsi
    cvx_begin quiet
      variable x(2,1)
      variable s(6,1)
      minimize(2*x(1) + x(2)^2 + tk*sum(s))
      subject to
      x'*H1*x - 9 <= s(1);
      9 <= xk'*H1*xk + [8*xk(1), 2*xk(2)]*(x - xk) + s(2);
      A*x <= b + [s(3) s(4) s(5) s(6)]';
      s >= 0;
    cvx_end
    xnew = x;
    fnew = 2*xnew(1) + xnew(2)^2;
    err = abs((fnew - fk)/fk);
    xk = xnew;
```

```
    fk = fnew;
    f = [f fk];
    X = [X x];
    tk = min([mu*tk tmax]);
    k = k + 1;
end
s
xs = xk;
fs = 2*xs(1) + xs(2)^2;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel('Iteration','fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
axis square
disp('satisfaction of the equality constraint:')
4*xs(1)^2 + xs(2)^2 - 9
disp('satisfaction of inequality constraints:')
A*xs - b
```

With $\boldsymbol{x}_0 = \begin{bmatrix} 6 \\ -2 \end{bmatrix}, \tau_0 = 1, \tau_{\max} = 10^3, \mu = 1.5,$ and $\varepsilon = 10^{-6}$, it took the algorithm 7 iterations to converge to a solution

$$\boldsymbol{x}^* = \begin{bmatrix} 1.414213562308050 \\ 1.000000000258204 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = 3.828427125132507$. Satisfaction of the equality and inequality constraints are evaluated in terms of

$$|a_1(\boldsymbol{x})| = 2.194973092173313 \times 10^{-10},$$

$$\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b} = \begin{bmatrix} -0.000000000258204 \\ -3.999999999741796 \\ -0.828427124616099 \\ -1.171572875383901 \end{bmatrix}.$$

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 16$ and is infeasible as

$$|a_1(\boldsymbol{x}_0)| = 139 \quad \text{and} \quad \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b} = \begin{bmatrix} 3 \\ -7 \\ -10 \\ 8 \end{bmatrix} \quad \blacksquare$$

15.15 The MATLAB code solving the problem is given below.

```
% input:
% [x0,lam0,mu0]: initial point
% tau: weight in merit function for constraint violation
% mu: weight for penalty term
% rou: bound on \delta
% epsi: tolerance for convergence
% output:
% xs: local minimizer obtained.
% fs: objective function at xs.
% k: number of iterations performed.
% X: trajectory of the iterates.
% Example: [xs,fs,k,X] = prob15_15([-1 -2]',1,[1 1 1 1]',20,20,1,1e-6);
function [xs,fs,k,X] = prob15_15(x0,lam0,mu0,eta,tau,rou,epsi)
A = [-1, 0; 1, 0; 0, -1; 0, 1];
b = [-2, 5, -1, 5]';
xk = x0(:);
lamk = lam0;
muk = mu0(:);
Zk = eye(2,2);
k = 0;
X = xk;
f = 2*xk(1)^2 + 3*xk(2)^2;
err = 1;
while err >= epsi
    xk1 = xk(1);
    xk2 = xk(2);
    gf = [4*xk1, 6*xk2]';
    ak = xk1^2 + xk2^2 - 16;
    Ae = 2*xk';
    ck = A*xk - b;
    cvx_begin quiet
      variable dt(2,1)
      variable u(1,1)
      variable v(1,1)
      variable w(4,1)
      variable r(1,1)
      dual variable y1
      dual variable y2
      minimize(0.5*dt'*Zk*dt + gf'*dt + eta*(u+v+sum(w)+r));
      subject to
      y1:ak + Ae*dt == u - v;
      y2:ck + A*dt <= w;
      norm(dt) <= rou + r;
      u >= 0;
```

```
  v >= 0;
  w >= 0;
  r >= 0;
cvx_end
nuvw = norm([u, v, w', r]);
if nuvw <= 1e-6
   dtk = dt;
   lam_qp = y1;
   mu_qp = y2;
else
   ak = ak - u + v;
   ck = ck - w;
   rou_t = rou + r;
   cvx_begin quiet
     variable dt(2,1)
     dual variable y1
     dual variable y2
     minimize(0.5*dt'*Zk*dt + gf'*dt);
     subject to
     y1: Ae*dt == -ak;
     y2: A*dt <= -ck;
     norm(dt) <= rou_t;
   cvx_end
   dtk = dt;
   lam_qp = y1;
   mu_qp = y2;
end
dlam = lam_qp - lamk;
dmu = mu_qp - muk;
Kt = 40;
alf = 0:1/(Kt-1):1;
psi = zeros(Kt,1);
for i = 1:Kt
    alfi = alf(i);
    xw = xk + alfi*dtk;
    x1 = xw(1);
    x2 = xw(2);
    t1 = 2*x1^2 + 3*x2^2;
    t2 = abs(x1^2 + x2^2 - 16);
    t3 = max(0,A*xw-b);
    psi(i) = t1 + tau*(t2+sum(t3));
end
[~,ind] = min(psi);
alfk = alf(ind);
xnew = xk + alfk*dtk;
lamk1 = lamk + alfk*dlam;
```

```
    muk1 = muk + alfk*dmu;
    x1 = xnew(1);
    x2 = xnew(2);
    gfnew = [4*x1, 6*x2]';
    Aenew = 2*xnew';
    gamk = (gfnew - gf) + (Aenew - Ae)'*lamk1 ;
    ct1 = dtk'*gamk;
    ct2 = dtk'*Zk*dtk;
    if ct1 >= 0.2*ct2
       thet = 1;
    else
       thet = (0.8*ct2)/(ct2 - ct1);
    end
    ht = Zk*dtk;
    ek = thet*gamk + (1-thet)*ht;
    Zk = Zk + (ek*ek')/(dtk'*ek) - (ht*ht')/ct2;
    err = norm([(xnew-xk);(lamk1-lamk);(muk1-muk)]);
    xk = xnew;
    lamk = lamk1;
    muk = muk1;
    fk = 2*xk(1)^2 + 3*xk(2)^2;
    f = [f fk];
    X = [X xk];
    k = k + 1;
end
xs = xk;
fs = fk;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel({'Iteration';'(b)'},'fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
axis square
disp('satisfaction of equality constraint:')
xs(1)^2 + xs(2)^2 - 16
disp('satisfaction of inequality constraint:')
A*xs - b
```

With $\boldsymbol{x}_0 = \begin{bmatrix} -1 \\ -2 \end{bmatrix}, \lambda_0 = 1, \boldsymbol{\mu}_0 = \texttt{ones(4,1)}, \eta = 20, \tau = 20, \rho = 1,$ and $\varepsilon = 10^{-6}$, it took the algorithm 18 iterations to converge to a solution

$$\boldsymbol{x}^* = \begin{bmatrix} 3.872983346191314 \\ 1.000000000061651 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = 33.000000000120437$. Satisfaction of the equality and inequality constraints are evaluated in terms of

$$|a_1(\boldsymbol{x}^*)| = 1.431743612556602 \times 10^{-12},$$

$$\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b} = \begin{bmatrix} -1.872983346191314 \\ -1.127016653808686 \\ -0.000000000061651 \\ -3.999999999938349 \end{bmatrix}.$$

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 14$ and is infeasible as

$$|a_1(\boldsymbol{x}_0)| = 11 \quad \text{and} \quad \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b} = \begin{bmatrix} 3 \\ -6 \\ 3 \\ -7 \end{bmatrix} \quad \blacksquare$$

15.16 Note the objective function $f(\boldsymbol{x})$ and linear constraint function $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ are convex, but the equality constraint function $a_1(\boldsymbol{x})$ is nonlinear, hence this is a nonconvex problem. Also note that $a_1(\boldsymbol{x}) = 0$ is equivalent to $-a_1(\boldsymbol{x}) \le 0$ and $a_1(\boldsymbol{x}) \le 0$ where $a_1(\boldsymbol{x})$ is convex but $-a_1(\boldsymbol{x})$ is not. To apply Algorithm 15.8, we write $-a_1(\boldsymbol{x}) = u_1(\boldsymbol{x}) - v_1(\boldsymbol{x})$ where

$$u_1(\boldsymbol{x}) = 16 \quad \text{and} \quad v_1(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x}.$$

It follows that

$$\hat{v}_1(\boldsymbol{x}, \boldsymbol{x}_k) = \boldsymbol{x}_k^T\boldsymbol{x}_k + 2\boldsymbol{x}_k^T(\boldsymbol{x} - \boldsymbol{x}_k)$$

and the CP problem in Eq. (15.31) becomes

$$\begin{aligned} \text{minimize} \quad & 2x_1 + 3x_2^2 + \tau_k \sum_{i=1}^{6} s(i) \\ \text{subject to:} \quad & 16 \le \boldsymbol{x}_k^T\boldsymbol{x}_k + 2\boldsymbol{x}_k^T(\boldsymbol{x} - \boldsymbol{x}_k) + s(1) \\ & \boldsymbol{x}^T\boldsymbol{x} - 16 \le s(2) \\ & \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b} + s(3:6) \\ & s(i) \ge 0 \quad \text{for} \quad i = 1, 2, ..., 6 \end{aligned}$$

The MATLAB code that applies Algorithm 15.8 to solve the problem at hand is given below.

```
% input:
% x0: initial point.
% t0: initial value of parameter tau
% tmax: maximum value of tau
```

```
% mu: parameter mu
% epsi: convergence tolerance
% output:
% xs: local minimizer obtained
% fs: objective function at xs
% k: number of iterations performed
% Example: [xs,fs,k] = prob15_16([-1 -2]',1,1e3,1.5,1e-9)
function [xs,fs,k] = prob15_16(x0,t0,tmax,mu,epsi)
A = [-1, 0; 1, 0; 0, -1; 0, 1];
b = [-2, 5, -1, 5]';
xk = x0(:);
tk = t0;
fk = 2*xk(1) + xk(2)^2;
f = fk;
X = xk;
k = 0;
err = 1;
while err >= epsi
    cvx_begin quiet
      variable x(2,1)
      variable s(6,1)
      minimize(2*x(1)^2 + 3*x(2)^2 + tk*sum(s))
      subject to
      16 <= xk'*xk + 2*xk'*(x - xk) + s(1);
      x'*x - 16 <= s(2);
      A*x <= b + [s(3) s(4) s(5) s(6)]';
      s >= 0;
    cvx_end
    xnew = x;
    fnew = 2*xnew(1)^2 + 3*xnew(2)^2;
    err = abs((fnew - fk)/fk);
    xk = xnew;
    fk = fnew;
    f = [f fk];
    X = [X x];
    tk = min([mu*tk tmax]);
    k = k + 1;
end
xs = xk;
fs = 2*xs(1)^2 + 3*xs(2)^2;
figure(1)
set(gca,'fontsize',13,'fontname','times')
plot(0:1:k,f,'k-','linewidth',1.4)
xlabel('Iteration','fontsize',14,'fontname','times')
ylabel('{\itf}{\rm(}{\itx}{\rm)}','fontsize',14)
grid
```

```
axis square
disp('satisfaction of the equality constraint:')
xs'*xs - 16
disp('satisfaction of inequality constraints:')
A*xs - b
```

With $\boldsymbol{x}_0 = \begin{bmatrix} -1 \\ -2 \end{bmatrix}, \tau_0 = 1, \tau_{\max} = 10^3, \mu = 1.5,$ and $\varepsilon = 10^{-9}$, it took the algorithm 12 iterations to converge to a solution

$$\boldsymbol{x}^* = \begin{bmatrix} 3.872983342021446 \\ 1.000000016398318 \end{bmatrix}$$

at which the objective function assumes the value $f(\boldsymbol{x}^*) = 33.000000033541127$. Satisfaction of the equality and inequality constraints are evaluated in terms of

$$|a_1(\boldsymbol{x}_0)| = 3.722426811236801 \times 10^{-10},$$

$$\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b} = \begin{bmatrix} -1.872983342021446 \\ -1.127016657978554 \\ -0.000000016398318 \\ -3.999999983601682 \end{bmatrix}.$$

For comparison, the objective at the initial point assumes the value $f(\boldsymbol{x}_0) = 14$ and is infeasible as

$$|a_1(\boldsymbol{x}_0)| = 11 \text{ and } \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b} = \begin{bmatrix} 3 \\ -6 \\ 3 \\ -7 \end{bmatrix} \quad \blacksquare$$

15.17 The problem at hand is equivalent to

$$\text{minimize } \tfrac{1}{2} \| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \|_2^2 + I_C(\boldsymbol{x})$$

where $I_C(\boldsymbol{x})$ is an indicator function and $C = \{\boldsymbol{x} : x_i \in \{1, -1\} \text{ for } i = 1, 2, ..., n\}$. The problem is in turn equivalent to the constrained problem

$$\begin{aligned} &\text{minimize } \tfrac{1}{2} \| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \|_2^2 + I_C(\boldsymbol{y}) \\ &\text{subject to: } \boldsymbol{x} - \boldsymbol{y} = \boldsymbol{0} \end{aligned}$$

which fits into the ADMM formulation in Eq. (15.35) where $f(\boldsymbol{x}) = \tfrac{1}{2} \| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \|_2^2$. As a result, the ADMM iterations in Eqs. (15.36a-c) can be specified to

$$\boldsymbol{x}_{k+1} = \left(\alpha \boldsymbol{I} + \boldsymbol{A}^T \boldsymbol{A} \right)^{-1} \left(\alpha (\boldsymbol{y}_k - \boldsymbol{v}_k) + \boldsymbol{A}^T \boldsymbol{b} \right)$$

$$\boldsymbol{y}_{k+1} = P_C(\boldsymbol{x}_{k+1} + \boldsymbol{v}_k)$$

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}$$

where P_C denotes pointwise projection

$$P_C(\boldsymbol{z}) = \boldsymbol{q} \text{ with } q_i = \begin{cases} 1 & \text{if } z_i > 0 \\ -1 & \text{if } z_i < 0 \end{cases}.$$

Note that the quantities $\left(\alpha \boldsymbol{I} + \boldsymbol{A}^T \boldsymbol{A}\right)^{-1}$ and $\boldsymbol{A}^T \boldsymbol{b}$ required to evaluated $\boldsymbol{x}_{k+1}$ need to compute only once for all iterations.

The MATLAB code that applies Algorithm 15.9 to solve the problem at hand is given below.

```
% Example: [xs,x,r,d,ki] = prob15_17(50,12,17,6,0.0001,20);
function [xs,x,r,d,ki] = prob15_17(m,n,st1,st2,a,K)
% Data preparation
randn('state',st1)
A = randn(m,n);
randn('state',st2)
b = 5*randn(m,1);
Ai = inv(a*eye(n)+A'*A);
atb = A'*b;
% Find the global solution
X = get_x(n);
N = 2^n;
xs = X(:,1);
fs = norm(A*xs - b);
for i = 1:(N-1)
    xwi = X(:,i+1);
    fwi = norm(A*xwi - b);
    if fwi < fs
        xs = xwi;
        fs = fwi;
    end
end
% ADMM iterations
yk = zeros(n,1);
vk = zeros(n,1);
ki = 0;
d = 1;
r = 1;
dk = 1;
rk = 1;
while ki < K
    tk = yk - vk;
```

```
    xk1 = Ai*(atb + a*tk);
    xvk = xk1 + vk;
    yk1 = sign(xvk);
    vk1 = vk + xk1 - yk1;
    dk = a*norm(yk1-yk);
    d(ki+1) = dk;
    rk = norm(xk1-yk1);
    r(ki+1) = rk;
    yk = yk1;
    vk = vk1;
    ki = ki + 1;
end
tk = yk - vk;
xk1 = Ai*(atb + a*tk);
x = sign(xk1);
[xs x]
t = 1:1:n;
figure(1)
subplot(121)
set(gca,'fontsize',13,'fontname','times')
stem(t,x,'.k')
axis square
axis([0 n -1.5 1.5])
xlabel({'Component index';'\rm(a)'},'fontsize',14,'fontname','times')
ylabel('{\bf\itx}*','fontsize',14,'fontname','times')
grid
subplot(122)
set(gca,'fontsize',13,'fontname','times')
stem(t,xs,'.k')
axis square
axis([0 n -1.5 1.5])
xlabel({'Component index';'\rm(b)'},'fontsize',14,'fontname','times')
ylabel('{\bf\itx}_s','fontsize',14,'fontname','times')
grid
figure(2)
kt = 1:1:ki;
subplot(121)
set(gca,'fontsize',13,'fontname','times')
semilogy(kt,r,'-k','linewidth',1.5)
axis square
axis([1 ki 1e-4 1e1])
xlabel({'Iteration';'(a)'},'fontsize',14,'fontname','times')
ylabel({'||{\bf\itr}_k||'},'fontsize',14,'fontname','times')
grid
subplot(122)
set(gca,'fontsize',13,'fontname','times')
```

```
plot(kt,d,'-k','linewidth',1.5)
axis square
axis([1 ki -0.01 0.025])
xlabel({'Iteration';'(b)'},'fontsize',14,'fontname','times')
ylabel({'||{\bf\itd}_k||'},'fontsize',14,'fontname','times')
grid

function X = get_x(n)
X = [1 -1];
for i = 1:n-1
   ni = size(X)*[0 1]';
   for j = 1:ni
      W(:,2*j-1) = [X(:,j); 1];
      W(:,2*j) = [X(:,j); -1];
   end
   X = W;
   clear W
end
```

With $\alpha = 10^{-4}$, it took the algorithm 20 iterations to converge to the solution

$$\boldsymbol{x}^* = \begin{bmatrix} 1 & -1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 \end{bmatrix}^T.$$

The solution obtained happens to be globally optimal as it is identical to the solution obtained by an exhaustive search with much heavier computational burdens. ■

Chapter 16 Applications of Constrained Optimization

16.6

(**a**) Recall that an FIR filter of order *N* is characterized by its transfer function

$$H(z) = \sum_{n=0}^{N} h_n z^{-n}.$$

We follow the solution idea outlined in Sec. 16.2.4 to convert the problem in Example 16.2 which is given by Eq. (16.28), namely,

$$\begin{aligned} &\text{minimize} && \delta \\ &\text{subject to:} && W(\omega)\,|H(e^{j\omega}) - H_d(\omega)| \le \delta \ \text{ for } \ \omega \in \Omega \end{aligned}$$

into the SOCP formulation

$$\begin{aligned} &\text{minimize} && \boldsymbol{c}^T \boldsymbol{x} \\ &\text{subject to:} && \| \boldsymbol{A}_i \boldsymbol{x} + \boldsymbol{c}_i \|_2 \le \boldsymbol{c}^T \boldsymbol{x} \ \text{ for } \ i = 1, 2, ..., M \end{aligned}$$

where Ω is a frequency region of interest over the positive half of the baseband $[0, \pi]$, $W(\omega)$ is a given weighting function,

$$\boldsymbol{x} = \begin{bmatrix} \delta \\ h_1 \\ \vdots \\ h_N \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \boldsymbol{A}_i = \begin{bmatrix} \mathbf{0} & \boldsymbol{c}_w^T \\ \mathbf{0} & \boldsymbol{s}_w^T \end{bmatrix}, \boldsymbol{c}_i = \begin{bmatrix} -H_{rw}(\omega_i) \\ H_{iw}(\omega_i) \end{bmatrix}$$

with $\boldsymbol{c}_w = W(\omega_i)\boldsymbol{c}(\omega_i)$, $\boldsymbol{s}_w = W(\omega_i)\boldsymbol{s}(\omega_i)$, $H_{rw}(\omega_i) = W(\omega_i)H_r(\omega_i)$, $H_{iw}(\omega_i) = W(\omega_i)H_i(\omega_i)$,

$$\boldsymbol{c}(\omega_i) = \begin{bmatrix} 1 \\ \cos\omega_i \\ \vdots \\ \cos N\omega_i \end{bmatrix} \text{ and } \boldsymbol{s}(\omega_i) = \begin{bmatrix} 0 \\ \sin\omega_i \\ \vdots \\ \sin N\omega_i \end{bmatrix}.$$

We follow the design specifications described in Example 16.2 to design a minimax lowpass FIR filter of order 84 with group delay D = 42, passband edge $\omega_p = 0.45\pi$ and stopband edge $\omega_a = 0.5\pi$. To this end, we uniformly place M = 300 sample frequencies in the normalized frequency range $\Omega = [0, 0.45\pi] \cup [0.5\pi, \pi]$. The weight function assumes the form described in Eq. (16.1) with $\gamma = 1.5$.

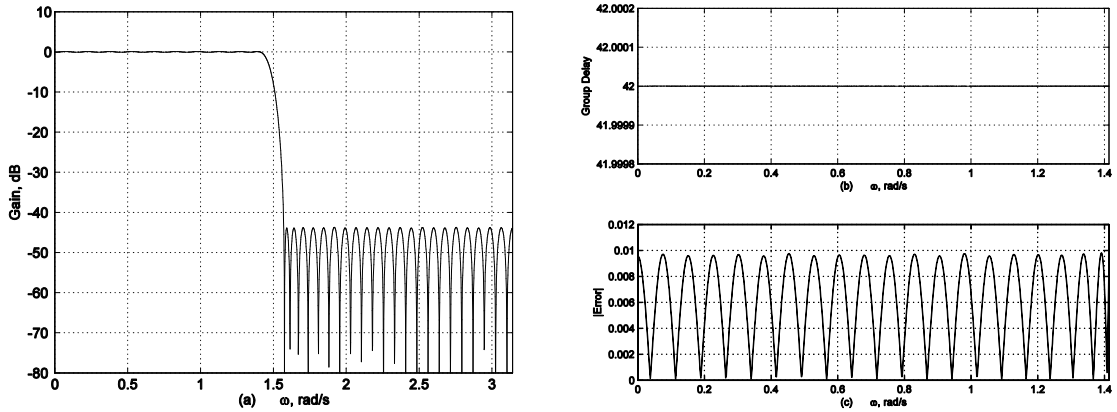
The MATLAB code implementing the design is given below.

```
% Input:
% N: order of the FIR filter
% D: desired group delay in passband
% omi_p: passband edge
% omi_a: stopband edge
% g: weight \gamma in stopband
% M: number of frequency grids used
% Output:
% h: impulse response of the FIR filter obtained
% copt: minimum ripple achieved in the passband and stopband
% Example: [h,copt] = prob16_6a(84,42,0.45,0.5,1.5,300);
function [h,copt] = prob16_6a(N,D,omip,omia,g,M)
Mp = ceil(M*omip/(omip+1-omia));
Ma = M - Mp;
W = [ones(Mp,1); g*ones(Ma,1)];
omip1 = pi*omip;
omia1 = pi*omia;
omi1 = 0:omip1/(Mp-1):omip1;
omi2 = omia1:(pi-omia1)/(Ma-1):pi;
omi1 = omi1(:);
omi2 = omi2(:);
d1 = D*omi1;
```

```
omi = [omi1; omi2];
Hr = zeros(M,1);
Hi = Hr;
Hr(1:Mp) = cos(d1);
Hi(1:Mp) = -sin(d1);
Hrw = W.*Hr;
Hiw = W.*Hi;
t = 0:1:N;
t = t(:)';
z = zeros(N+1,1);
c = [1; z];
cvx_begin quiet
variable x(N+2,1)
minimize(c'*x)
subject to
for i = 1:M
   Ai = [0 W(i)*cos(omi(i)*t); 0 W(i)*sin(omi(i)*t)];
   ci = [-Hrw(i); Hiw(i)];
   norm(Ai*x+ci) <= c'*x;
end
cvx_end
copt = x(1);
h = x(2:N+2);
w = 0:pi/2047:pi;
H = freqz(h,1,w);
fp = 0:omip1/1023:omip1;
Hp = freqz(h,1,fp);
erp = max(abs(abs(Hp)-1));
fa = omia1:(pi-omia1)/1023:pi;
Ha = freqz(h,1,fa);
era = -max(20*log10(abs(Ha)));
disp('maximum passband ripple is:')
erp
disp('minimum stopband attenuation(dB) is:')
era
f = 0:pi/2047:pi;
figure(1)
plot(f,20*log10(abs(H)))
axis([0 pi -80 10])
grid
get(gca)
get(gcf)
set(gca,'fontsize',14)
xlabel('(a)     {\it\omega}, rad/s','fontsize',14)
ylabel('Gain, dB','fontsize',14)
figure(2)
```

```
gd = -1023*diff(phase(Hp))/omip1;
ft1 = 0:omip1/1022:omip1;
subplot(211)
plot(ft1,gd)
grid
axis([0 omip1 D-0.0002 D+0.0002])
xlabel('(b)     {\it\omega}, rad/s')
ylabel('Group Delay')
subplot(212)
Hr0 = cos(D*fp(:));
Hi0 = sin(D*fp(:));
Hd = Hr0 - sqrt(-1)*Hi0;
ep = abs(Hd - Hp(:));
plot(fp,ep)
grid
axis([0 omip1 0 0.012])
xlabel('(c)     {\it\omega}, rad/s')
ylabel('|Error|')
```

The figures below depict the amplitude and phase responses of the FIR filter designed. On comparing with the design results obtained from Example 16.2 and Fig. 16.2, the two designs are found to be practically identical.



(**b**) Recall that an IIR filter is characterized by its transfer function

$$H(z) = \frac{A(z)}{B(z)} = \frac{\sum_{i=0}^{N} a_i z^{-i}}{1 + \sum_{i=1}^{K} b_i z^{-i}}$$

We follow the solution idea outlined in Sec. 16.2.4 to replace the problem formulation in Example 16.3 which is given by Eq. (16.23), namely,

$$\begin{aligned} &\text{minimize} && \delta \\ &\text{subject to:} && \frac{W^2(\omega)}{|B_{k-1}(\omega)|^2} |A(\omega) - B(\omega)H_d(\omega)|^2 \le \delta \quad \text{for} \quad \omega \in \Omega \\ & && B(z) \ne 0 \quad \text{for} \quad |z| \ge 1 \end{aligned}$$

with

$$\begin{aligned}&\text{minimize} && \delta \\ &\text{subject to:} && W_k(\omega_i)\,|\,A(\omega_i)-B(\omega_i)H_d(\omega_i)\,|\le\delta \ \text{ for } \ \{\omega_i, i=1,...,M\}\subset\Omega \\ & && B(z)\neq 0 \ \text{ for } \ |\,z\,|\ge 1\end{aligned}$$

where $W_k(\omega_i)=\dfrac{W(\omega_i)}{|\,B_{k-1}(\omega_i)\,|}$.

Now we convert the first set of constraints of the above problem into a set of SOCP constraints in a procedure describe below. Let $\boldsymbol{a}=\begin{bmatrix}a_0 & a_1 & \cdots & a_N\end{bmatrix}^T$ and $\boldsymbol{b}=\begin{bmatrix}b_1 & b_2 & \cdots & b_K\end{bmatrix}^T$, and define

variable $\boldsymbol{x}=\begin{bmatrix}\delta\\ \boldsymbol{a}\\ \boldsymbol{b}\end{bmatrix}$. If we let $H_d(\omega)=H_r(\omega)+jH_i(\omega)$ and denote $H_{rw}(\omega)=W_k(\omega)H_r(\omega)$ and $H_{iw}(\omega)=W_k(\omega)H_i(\omega)$, then

$$W_k(\omega_i)A(\omega_i)=\boldsymbol{a}^T\boldsymbol{c}_{1w}(\omega_i)-j\boldsymbol{a}^T\boldsymbol{s}_{1w}(\omega_i),$$

$$W_k(\omega_i)H_d(\omega_i)B(\omega_i)=H_{rw}(\omega_i)+\boldsymbol{b}^T\left(H_r(\omega_i)\boldsymbol{c}_{2w}(\omega_i)+H_i(\omega_i)\boldsymbol{s}_{2w}(\omega_i)\right)+j\left(H_{iw}(\omega_i)+\boldsymbol{b}^T\left(H_i(\omega_i)\boldsymbol{c}_{2w}(\omega_i)-H_r(\omega_i)\boldsymbol{s}_{2w}(\omega_i)\right)\right).$$

where

$$\boldsymbol{c}_{1w}(\omega_i)=W_k(\omega_i)\begin{bmatrix}1\\ \cos\omega_i\\ \vdots\\ \cos N\omega_i\end{bmatrix},\quad \boldsymbol{s}_{1w}(\omega_i)=W_k(\omega_i)\begin{bmatrix}0\\ \sin\omega_i\\ \vdots\\ \sin N\omega_i\end{bmatrix},\quad \boldsymbol{c}_{2w}(\omega_i)=W_k(\omega_i)\begin{bmatrix}\cos\omega_i\\ \cos 2\omega_i\\ \vdots\\ \cos K\omega_i\end{bmatrix},\ \boldsymbol{s}_{2w}(\omega_i)=W_k(\omega_i)\begin{bmatrix}\sin\omega_i\\ \sin 2\omega_i\\ \vdots\\ \sin K\omega_i\end{bmatrix}.$$

It follows that

$$\begin{aligned}&W_k(\omega_i)\,|\,A(\omega_i)-B(\omega_i)H_d(\omega_i)\,|\\ &=\left\{\left[\boldsymbol{a}^T\boldsymbol{c}_{1w}(\omega_i)-\boldsymbol{b}^T\left(H_r(\omega_i)\boldsymbol{c}_{2w}(\omega_i)+H_i(\omega_i)\boldsymbol{s}_{2w}(\omega_i)\right)-H_{rw}(\omega_i)\right]^2+\left[\boldsymbol{a}^T\boldsymbol{s}_{1w}(\omega_i)+\boldsymbol{b}^T\left(H_i(\omega_i)\boldsymbol{c}_{2w}(\omega_i)-H_r(\omega_i)\boldsymbol{s}_{2w}(\omega_i)\right)+H_{iw}(\omega_i)\right]^2\right\}^{1/2}\\ &=\|\,\boldsymbol{A}_i\boldsymbol{x}+\boldsymbol{c}_i\,\|_2\end{aligned}$$

where

$$\boldsymbol{A}_i=\begin{bmatrix}0 & \boldsymbol{c}_{1w}^T(\omega_i) & -H_r(\omega_i)\boldsymbol{c}_{2w}^T(\omega_i)-H_i(\omega_i)\boldsymbol{s}_{2w}^T(\omega_i)\\ 0 & \boldsymbol{s}_{1w}^T(\omega_i) & H_i(\omega_i)\boldsymbol{c}_{2w}^T(\omega_i)-H_r(\omega_i)\boldsymbol{s}_{2w}^T(\omega_i)\end{bmatrix}\ \text{ and }\ \boldsymbol{c}_i=\begin{bmatrix}-H_{rw}(\omega_i)\\ H_{iw}(\omega_i)\end{bmatrix},$$

hence the first set of the constraints becomes

$$\|\,\boldsymbol{A}_i\boldsymbol{x}+\boldsymbol{c}_i\,\|_2\le\boldsymbol{c}^T\boldsymbol{x}\ \text{ for }\ i=1,2,...,M$$

where $\boldsymbol{c} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$.

The second constraint, i.e. $B(z) \neq 0$ for $|z| \geq 1$, is to assure the filter's stability. It is known [6, 7] that an IIR filter is stable if $\text{Re}[B(\omega)] > 0$ for $\omega \in [0, \pi]$. Based on this, we replace the second constraint with the set of linear constaints

$$1 + \boldsymbol{b}^T \boldsymbol{c}_2(\hat{\omega}_l) \geq \eta \quad \text{for } l = 1, 2, \ldots, L$$

where $\{\hat{\omega}_1, \hat{\omega}_2, \ldots, \hat{\omega}_L\} \subset [0, \pi]$, η is a small and positive scalar, and $\boldsymbol{c}_2(\omega) = \begin{bmatrix} \cos\omega \\ \cos 2\omega \\ \vdots \\ \cos K\omega \end{bmatrix}$.

We have now derived an SOCP formulation for the design problem as

$$\begin{aligned} &\text{minimize} && \boldsymbol{c}^T \boldsymbol{x} \\ &\text{subject to:} && \| \boldsymbol{A}_i \boldsymbol{x} + \boldsymbol{c}_i \|_2 \leq \boldsymbol{c}^T \boldsymbol{x} \quad \text{for } i = 1, 2, \ldots, M \\ & && 1 + \boldsymbol{b}^T \boldsymbol{c}_2(\hat{\omega}_l) \geq \eta \quad \text{for } l = 1, 2, \ldots, L \end{aligned}$$

We follow the design specifications described in Example 16.3 to design a stable minimax lowpass IIR filter of order (N, K) = (12, 6) with passband edge $\omega_p = 0.5\pi$, stopband edge $\omega_a = 0.6\pi$, and passband group delay D = 9. For this we uniformly place M = 240 frequency grids in $\Omega = [0, 0.5\pi] \cup [0.6\pi, \pi]$. For the stability constraints, L = 100 frequency grids are used and constant η is set to 0.05. The weight function assumes the form of Eq. (16.1) with $\gamma = 1$.

The MATLAB code implementing the design is given below.

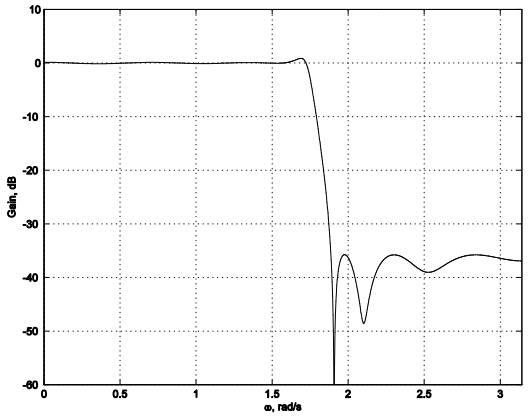
```
% Input:
% n: order of the IIR filter to be designed
% r: order of the denominator polynomial
% D: passband group delay
% fp: normalized passband edge betweem 0 and 1
% fa: normalized stopband edge between 0 and 1
% w: weight for the stopband
% M: total number of frequency grids in passnad and stopband
% L: number of frequency grids for stability constraints.
% eta: constant required in stability constraints.
% iter: number of iterative CVX runs.
% Output:
% as: optimal numerator polynomial
```

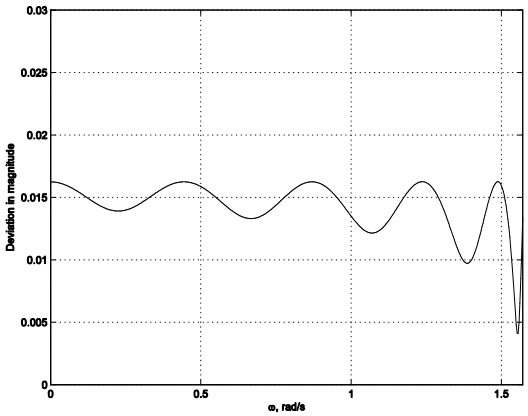
```
% bs: optimal denominator polynomial.
% dts: optimal upper bound.
% Example:
% [as,bs,dts] = prob16_6b(12,6,9,0.5,0.6,1,240,100,0.05,10);
function [as,bs,dts] = prob16_6b(N,K,D,fp,fa,w,M,L,eta,iter)
Mp = ceil(M*fp/(fp+1-fa));
Ma = M - Mp;
omip = 0:fp*pi/(Mp-1):fp*pi;
omia = fa*pi:(1-fa)*pi/(Ma-1):pi;
omi = [omip omia];
W = [ones(1,Mp) w*ones(1,Ma)];
D1 = 0:1:N;
D1 = D1(:);
D2 = 1:1:K;
D2 = D2(:);
Hr = zeros(M,1);
Hi = Hr;
d1 = D*omip;
Hr(1:Mp) = cos(d1);
Hi(1:Mp) = -sin(d1);
Hrw = W.*Hr;
Hiw = W.*Hi;
C1 = zeros(N+1,M); S1 = C1;
C2 = zeros(K,M); S2 = C2;
for i = 1:M
   C1(:,i) = cos(omi(i)*D1);
   S1(:,i) = sin(omi(i)*D1);
   C2(:,i) = cos(omi(i)*D2);
   S2(:,i) = sin(omi(i)*D2);
end
fh = 0:pi/(L-1):pi;
Ch = zeros(K,L);
t = 1:1:K;
t = t(:);
for i = 1:L
    Ch(:,i) = cos(fh(i)*t);
end
k = 0;
Wk = W;
while k < iter
cvx_begin
   variable a(N+1,1)
   variable b(K,1)
   variable dt(1,1)
   minimize(dt)
   subject to
```

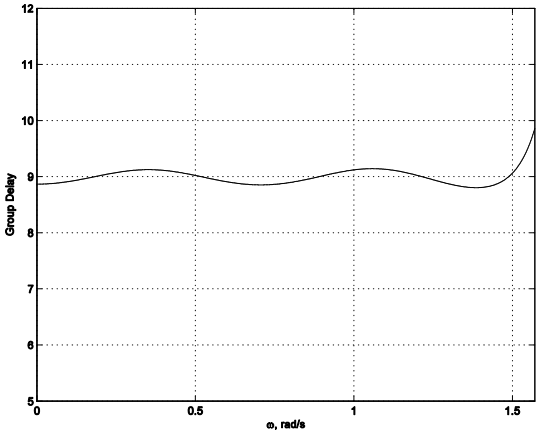
```
   x = [a; b];
   for i = 1:M
      c1 = Wk(i)*C1(:,i);
      s1 = Wk(i)*S1(:,i);
      c2 = Wk(i)*C2(:,i);
      s2 = Wk(i)*S2(:,i);
   ai1 = [c1' -Hr(i)*c2'-Hi(i)*s2'];
   ai2 = [s1' Hi(i)*c2'-Hr(i)*s2'];
   Ai = [ai1; ai2];
   ci = Wk(i)*[-Hr(i); Hi(i)];
   norm(Ai*x + ci) <= dt;
   end
   for i = 1:L
      1 + Ch(:,i)'*b >= eta;
   end
cvx_end
bk = b;
Wk = zeros(1,M);
for i = 1:M
    Bkr = 1 + bk'*C2(:,i);
    Bki = bk'*S2(:,i);
    Bk = [Bkr;Bki];
    Wk(i) = W(i)/norm(Bk);
end
k = k + 1;
end
as = a;
bs = [1;b];
dts = dt;
j = sqrt(-1);
disp('maximum modulus of the poles is')
modulus_max = max(abs(roots(bs)))
[h,ff] = freqz(as,bs,1024);
h_s = exp(-j*D*ff);
ff = 0:0.5/1023:0.5;
figure(1)
fs = 2*pi*ff;
mg = 20*log10(abs(h));
plot(fs,mg,'-');
axis([0 pi -60 10])
grid
xlabel('\omega, rad/s')
ylabel('Gain, dB')
figure(2)
w0 = floor(fp*1024);
ee = h_s(1:w0) - h(1:w0);
```

```
disp('max passband deviation')
max(abs(ee))
ff1 = 0:(0.5*fp)/(w0-1):0.5*fp;
ffs1 = 2*pi*ff1;
plot(ffs1,abs(ee),'-')
axis([0 pi*fp 0 0.03])
grid
xlabel('\omega, rad/s')
ylabel('Deviation in magnitude')
w1 = ceil(1024*fa);
mga = mg(w1:1024);
disp('minimum stopband attenuation')
-max(mga)
figure(3)
phx = 0.5*unwrap(angle(h))/pi;
php = phx(1:w0);
gdp = -2046*diff(php);
disp('maximum ripple in group delay:')
max(gdp-D)-min(gdp-D)
ff3 = 0:(0.5*fp)/(w0-2):0.5*fp;
ffs3 =2*pi*ff3;
plot(ffs3,gdp,'-')
xlabel('\omega, rad/s')
ylabel('Group Delay')
axis([0 pi*fp 5 12])
grid
```

It took 10 runs of the SOCP minimizaation, in each run the current iterate is used to update the weight $W_k(\omega) = \dfrac{W(\omega)}{|B_{k-1}(\omega)|}$. The IIR filter obtained is stable with the largest pole magnitude being 0.9445. It achieves maximum passband error 0.0163, minimum stopband attenuation 35.1585 dB, and maximum ripple of passband group delay 1.0575. This performance turns out to be comparable with that obtained from Example 16.3. The figures below depict the amplitude and phase responses of the IIR filter designed.







■

This is the last page of the manual.