東南大學
SOUTHEAST UNIVERSITY

Optimization Methods

Programming Homework

Name 陈鲲龙

ID 08022311

College School of Automation

January 3, 2025

Contents

1 Assignment Description

- Define a quadratic programming (i.e., quadratic objective function with linear constraints) problem by yourself.
- Write the program of the Interior Point Method to solve the quadratic programming problem by Matlab, C, C++, or Python.
- Solve the quadratic programming problem by using the function "quadprog" in MATLAB.
- Compare the results of Steps 2 and 3 for different scales or parameters in your quadratic programming model.
- Write the report of this programming exercise.

2 Solution and Implementation

2.1 Problem Definition

In this assignment, the quadratic programming problem to be solved has the objective function defined as:

$$\min \quad x_1^2 + x_2^2 + x_1 + x_2$$

The constraints are:

$$\text{s.t.} \quad x_1 \geq 0, \quad x_2 \geq 0$$

The standard matrix form of the problem is as follows:

$$\min \quad \frac{1}{2}\mathbf{x}^T P\mathbf{x} + \mathbf{q}^T\mathbf{x}$$

The constraints can be written as:

$$\text{s.t.} \quad \mathbf{A}\mathbf{x} \geq \mathbf{b}$$

where:

$$P = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \quad \mathbf{q} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

The reason for choosing this problem is that its optimization result is clearly $(0, 0)$, which helps in verifying whether the method implementation and program results are correct.

2.2 Results of quadprog(MATLAB)

The result obtained by the MATLAB `quadprog` function, including the optimal objective value and the corresponding X, is as follows:

```
quadprog:

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
fval:
   2.5000e-10

x:
   1.0e-09 *

    0.1250
    0.1250
```

The following is the code:

```
clc,clear,close all

P = [2, 0; 0, 2];
q = [1; 1];
x = [1; 1];
A = [-1, 0; 0, -1];
b = [0; 0];

disp('quadprog: ');
```

```
[x, fval] = quadprog(P, q, A, b);
disp('fval: ');
disp(fval);
disp('x: ');
disp(x);
```

2.3 Implemented Interior Point Method in MATLAB

The result is as follows:

```
interior point method:
   6.0127e-09


   1.0e-08 *


    0.3006
    0.3006
```

Since the problem to be solved in this assignment is relatively simple, and a good starting point, such as (1, 1), can be provided manually, I have chosen this so-called "newton-barrier" method[1].

The details and the algorithm[1] are as follows:

"We have seen that the point $x^\star(t)$ is $\frac{m}{t}$-suboptimal, and that a certificate of this accuracy is provided by the dual feasible pair $\lambda^\star(t), \nu^\star(t)$. This suggests a very straightforward method for solving the original problem (11.1) with a guaranteed specified accuracy ϵ: We simply take $t = \frac{m}{\epsilon}$ and solve the equality constrained problem

$$\text{minimize} \quad \frac{m}{\epsilon} f_0(x) + \phi(x)$$

subject to

$$Ax = b$$

using Newton's method. This method could be called the unconstrained minimization method, since it allows us to solve the inequality constrained problem (11.1) to a guaranteed accuracy by solving an unconstrained, or linearly constrained, problem. Although this method can work well for small problems, good starting points, and moderate accuracy (i.e., ϵ not too small), it does not work well in other cases. As a result, it is rarely, if ever, used."

Algorithm 1 Barrier Method[1]

1: **Given:** strictly feasible x, $t := t(0) > 0$, $\mu > 1$, tolerance $\epsilon > 0$.
2: **repeat**
3: **Centering step:** Compute $x^\star(t)$ by minimizing $tf_0 + \varphi$, subject to $Ax = b$, starting at x.
4: **Update:** $x := x^\star(t)$.
5: **Stopping criterion:** Quit if $\frac{m}{t} < \epsilon$.
6: **Increase** t: $t := \mu t$.
7: **until** $\frac{m}{t} < \epsilon$

The following is the code:

```
function [x, fval, xlist, gaplist] = interior_point_qp(P, q, A, b, x, t, r, alpha,
beta, mu, epsion)
%INTERIOR_POINT_QP Solve a quadratic programming problem by interior point method

m = size(A, 1);
i = 1;
xlist = [];
gaplist = [];
while (true)
    [x, ~, nsteps] = newton_barrier(P, q, A, b, x, t, r, alpha, beta);
    xlist = [xlist, x];
    gaplist = [gaplist, m * mu / t * ones(1, nsteps)];

    if (m / t < epsion)
        fval = compute_value_barrier(P, q, A, b, x, t);
        return;
    end
    t = mu * t;
    i = i + 1;
end
end
```

Next, we will solve the following using Newton' s method [1]:

$$\min Q(x, r) = t \cdot f(x) - \sum_{i=1}^{m} \ln(-g_i(x))$$

where $f(x)$ is our objective function:

$$\min \quad \frac{1}{2}\mathbf{x}^T P\mathbf{x} + \mathbf{q}^T\mathbf{x}$$

and $-g_i(x)$ is:

$$b - A\mathbf{x}$$

The algorithm[1] are as follows:

Algorithm 2 Newton' s Method[1]

1: **Input:** A starting point $x \in \mathrm{dom} f$, tolerance $\epsilon > 0$

2: **Repeat**

3: 1. Compute the Newton step and decrement:

$$\Delta x_{nt} := -\nabla^2 f(x)^{-1}\nabla f(x); \quad \lambda_2 := \nabla f(x)^T \nabla^2 f(x)^{-1}\nabla f(x)$$

4: 2. Stopping criterion: Quit if $\frac{\lambda_2}{2} \leq \epsilon$

5: 3. Line search: Choose step size t by backtracking line search

6: 4. Update: $x := x + t\Delta x_{nt}$

The following is the code:

```
function [x, p, nsteps] = newton_barrier(P, q, A, b, x, t, r, alpha, beta)
%NEWTON_QP Solve a qradratic programming by newton's method
%   alpha: Armijo criterion for line search
%   beta: descent multiplier for line search
nsteps = 0;
while (true)
    grad2 = compute_grad2_barrier(P, q, A, b, x, t);
    grad = compute_grad_barrier(P, q, A, b, x, t);
    grad2_inv = inv(grad2);
    d = - grad2_inv * grad;
    lambda2 = grad' * grad2_inv * grad;
    if (lambda2 / 2 <= r)
        p = compute_value_barrier(P, q, A, b, x, t);
        return;
    end
    step = backtracking_armijo_barrier(P, q, A, b, x, t, d, alpha, beta);
    x = x + step .* d;
    nsteps = nsteps + 1;
end
end
```

Where the first derivative of the objective function is

$$P\mathbf{x} + \mathbf{q}$$

and the second derivative is

$$P$$

The following is the code:

```
function grad = compute_grad_qp(P, q, x)
    grad = P * x + q;
end

function grad2 = compute_grad2_qp(P, q, x)
    grad2 = P;
end
```

The gradient and Hessian of the barrier function[1] are

$$\nabla\phi(x) = \frac{1}{b - A^T x} Ax,$$

$$\nabla^2\phi(x) = \frac{1}{(b - A^T x)^2} AA^T.$$

Or, more compactly,

$$\nabla\phi(x) = A^T r,$$

$$\nabla^2\phi(x) = A^T \mathrm{diag}(r^2) A,$$

where the elements of $r \in \mathbb{R}^m$ are given by

$$r = \frac{1}{b - A^T x}.$$

Since x is strictly feasible, we have $r \succ 0$, so the Hessian of ϕ is nonsingular if and only if A has rank n.

Therefore, the first and second derivative of grad-barrier function can be written as follows:

$$t(P\mathbf{x} + \mathbf{q}) + A^T r$$

$$tP + A^T \mathrm{diag}(r^2) A$$

The following is the code:

```
function grad = compute_grad_barrier(P, q, A, b, x, t)
    grad = compute_grad_qp(P, q, x);
    residual = b - A * x;
    grad = t * grad + A' * (1 ./ residual);
end

function grad2 = compute_grad2_barrier(P, q, A, b, x, t)
    grad2 = compute_grad2_qp(P, q, x);
    residual = b - A * x;
    r = 1 ./ (residual .^ 2);
    grad2 = t .* grad2 + A' * diag(r) * A;
end
```

Where, the Backtracking line search[1] is:

"Most line searches used in practice are inexact: the step length is chosen to approximately minimize f along the ray $\{x + t\Delta x \mid t \geq 0\}$, or even to just reduce f "enough". Many inexact line search methods have been proposed. One inexact line search method that is very simple and quite effective is called backtracking line search. It depends on two constants α, β with $0 < \alpha < 0.5$, $0 < \beta < 1$."

Algorithm 3 Backtracking Line Search[1]

1: **Input:** Given a descent direction Δx for f at $x \in \mathrm{dom} f$, $\alpha \in (0, 0.5)$, $\beta \in (0, 1)$
2: $t := 1$
3: **while** $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^T \Delta x$ **do**
4: $\quad t := \beta t$
5: **end while**

The following is the code:

```
function step = backtracking_armijo_barrier(P, q, A, b, x0, t, d, alpha, beta)
% d: search direction
step = 1;

while (true)
    delta_x = step .* d;
    x = x0 + delta_x;

    fval_x0 = compute_value_barrier(P, q, A, b, x0, t);
```

```
    grad_x0 = compute_grad_barrier(P, q, A, b, x0, t);
    fval_x = compute_value_barrier(P, q, A, b, x, t);
    if (fval_x <= fval_x0 + alpha * grad_x0' * delta_x)
        return;
    end
    step = beta * step;
end
end
```

What's more, the optimization process is shown in Figure 1 and Figure 2:

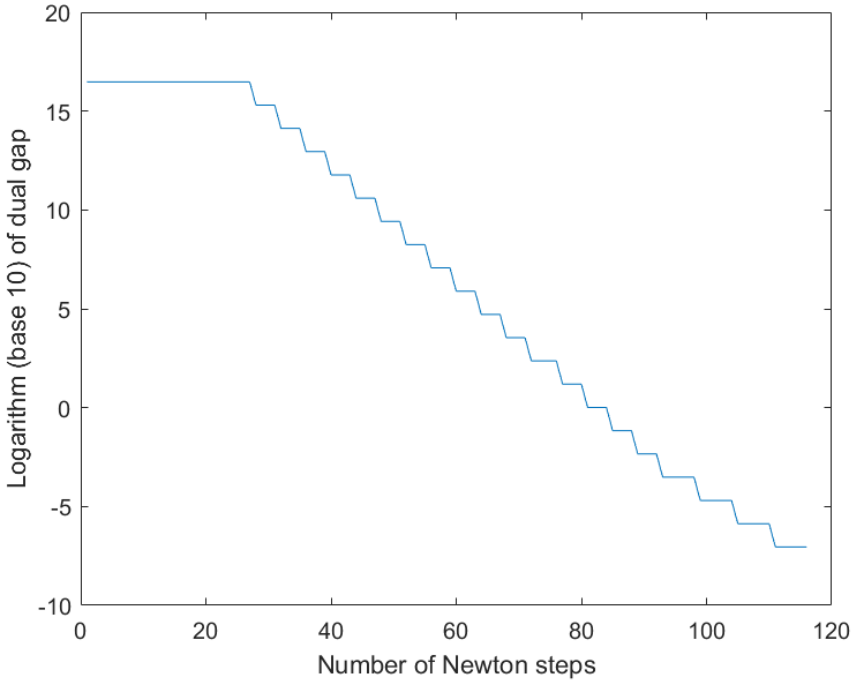

Figure 1: The logarithm (base 10) of the dual gap as a function of the number of Newton steps.

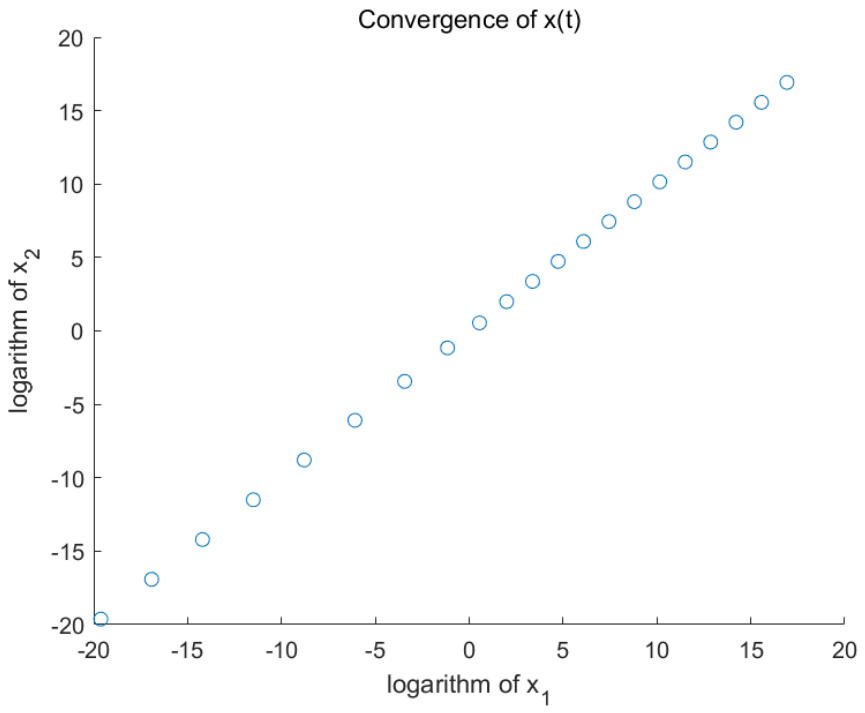

Figure 2: Convergence of $x(t)$, represented as the logarithm of x_1 and x_2.

2.4 Compare the results for different scales or parameters

The parameters used in Newton's method are summarized in the table below:

Parameter	Description
r	Tolerance for stopping criterion
α	Step size reduction factor
β	Step size factor for backtracking

Table 1: Parameters for Newton's Method

The parameters used in the interior point method are summarized in the table below:

Parameter	Description
m	Number of constraints (rows in matrix A)
ϵ	Small tolerance for stopping criterion
μ	Centrality parameter
t	Initial value for the barrier parameter

Table 2: Parameters for Interior Point Method

2.4.1 Impact of α and β in Backtracking Line Search

In **Backtracking Line Search**, the parameters α and β play a crucial role in controlling the step size adjustment process, which, in turn, affects the convergence speed and stability of the optimization algorithm.

1. Parameter α

Meaning: α is a constant used to control the reduction of the search step. It determines how much the new step size should reduce in order to guarantee a sufficient decrease in the objective function value. Typically, $0 < \alpha < 0.5$, and commonly chosen values are $\alpha \approx 0.2$ to $\alpha \approx 0.4$.

Impact:

- **Smaller α:** If α is too small (e.g., close to 0), it results in very small step sizes. This makes the backtracking line search overly conservative, causing the algorithm to shrink the step size too much, which can slow down the convergence speed, as Figure 3.

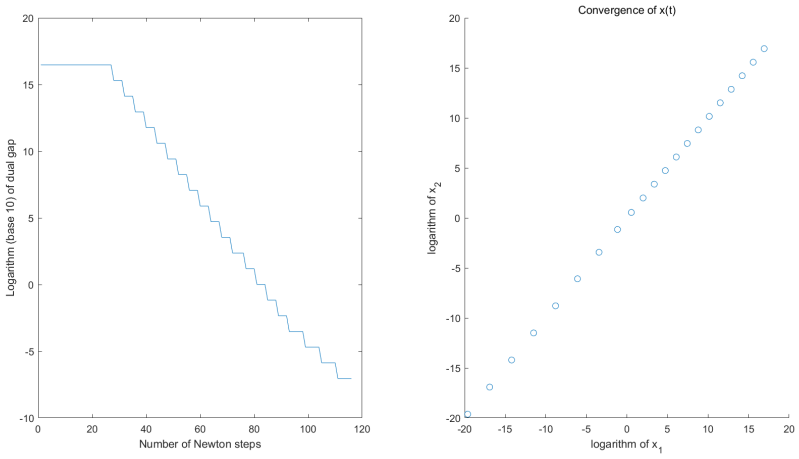

Figure 3: $\alpha = 0.0001$

whose result is:

```
interior point method:
   6.0127e-09

   1.0e-08 *

    0.3006
    0.3006
```

- **Larger α:** If α is too large, it may cause the step size to increase too quickly and fail to sufficiently decrease the objective function value during each iteration. This can lead to instability in convergence or even non-convergence of the algorithm, as Figure 4.

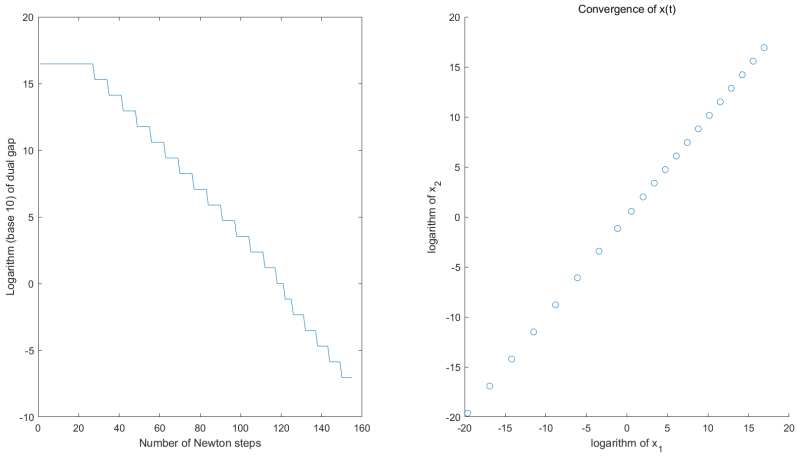

Figure 4: $\alpha = 0.5$

whose result is:

```
interior point method:
   6.0248e-09

   1.0e-08 *

    0.3012
    0.3012
```

2. Parameter β

Meaning: β is the step size reduction factor, controlling how quickly the step size decreases during each backtracking iteration. It is typically chosen in the range $0 < \beta < 1$, with common values being $\beta \approx 0.5$ or $\beta \approx 0.8$.

Impact:

- **Smaller β:** When β is smaller (e.g., close to 0.5), the step size reduces quickly, leading to faster shrinking of the step size. This can cause the algorithm to prematurely stop searching, resulting in suboptimal solutions, as Figure 5.

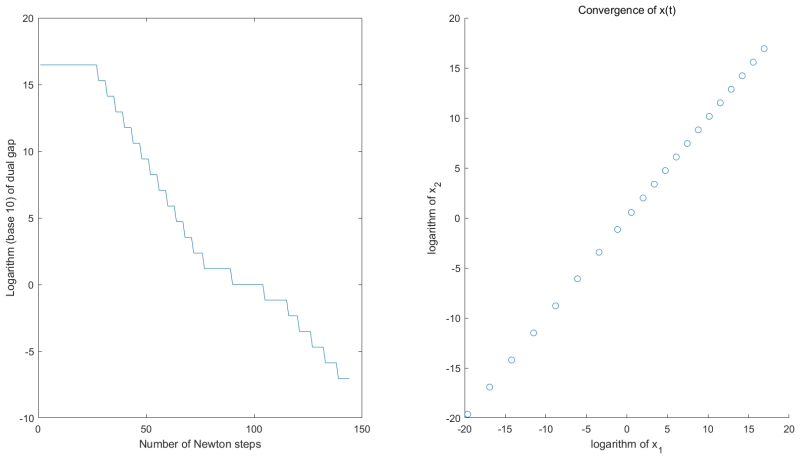

Figure 5: $\beta = 0.1$

whose result is:

```
interior point method:
   6.0099e-09

   1.0e-08 *

    0.3005
    0.3005
```

- **Larger β:** If β is larger (e.g., close to 1), the step size reduces more slowly, requiring more backtracking steps to find an appropriate step size. This can slow down convergence, especially in the early stages of optimization, requiring more computational resources and time to complete each optimization step, as Figure 6.

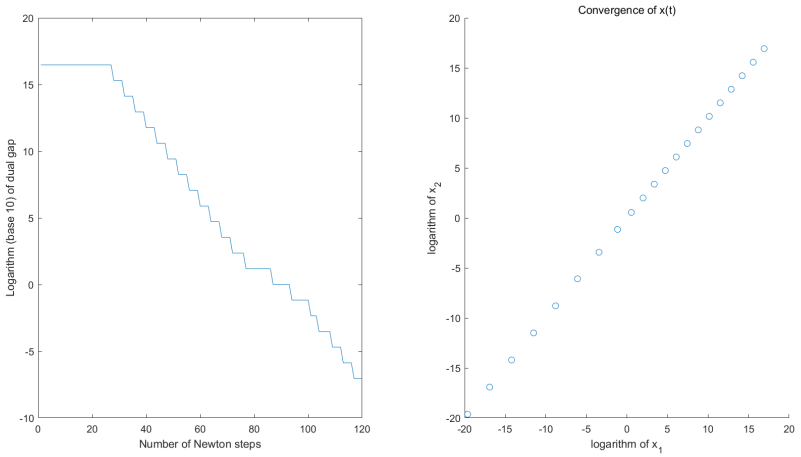

Figure 6: $\beta = 0.9$

whose result is:

```
interior point method:
   6.0038e-09

   1.0e-08 *

    0.3002
    0.3002
```

Combined Effect:

Balanced Selection: α and β need to be chosen in a balanced way for optimal performance. Ideally, α should be small enough to ensure each iteration reduces the objective function value, while β should be large enough to prevent excessive backtracking, but not too large to slow down the step size reduction process.

Impact on Optimization Speed:

- Smaller α values make the algorithm more cautious in adjusting the step size, which may lead to slower convergence.
- Larger β values slow down the reduction of the step size, which may increase the number of backtracking steps and thus result in slower convergence.

Conclusion:

- **Smaller α and β** values make the backtracking line search more conservative, resulting in slower convergence.

- **Larger α and β** values may lead to faster step size adjustments, which could result in unstable convergence or failure to converge.

Usually, the optimal values for α and β need to be tuned experimentally for the specific problem at hand.

2.4.2 Impact of μ in Interior Point Method

In the **Interior Point Method**, the parameter μ plays an important role in controlling the convergence behavior of the algorithm. The value of μ influences the adjustment of the barrier parameter, which regulates the trade-off between optimizing the objective function and staying within the feasible region.

1. Parameter μ

Meaning: μ is a parameter that controls the update of the barrier term in the interior point method. It is typically initialized as a small positive value and is updated at each iteration. The value of μ determines the balance between the objective function and the barrier term. Larger values of μ result in stronger penalties for violating the constraints, and smaller values of μ allow the solution to get closer to the boundary of the feasible region.

Impact:

- **Smaller μ:** If μ is too small, the penalty for violating constraints becomes weak, and the algorithm may make significant progress towards the optimal solution, but at the cost of violating feasibility. This can lead to a faster reduction in the objective function, but at the risk of non-feasibility in early iterations, as shown in Figure 7.

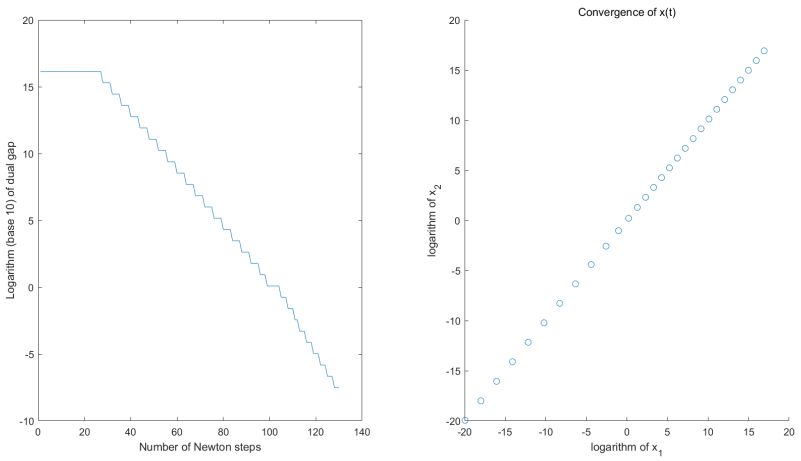

Figure 7: $\mu = 7$

`interior point method:`

```
4.3469e-09

1.0e-08 *

 0.2173
 0.2173
```

- **Larger μ:** If μ is too large, the penalty for violating constraints becomes stronger, and the algorithm may take more conservative steps to ensure feasibility. This slows down the progress of reducing the objective function value in the early iterations but ensures the solution remains feasible, as shown in Figure 8.

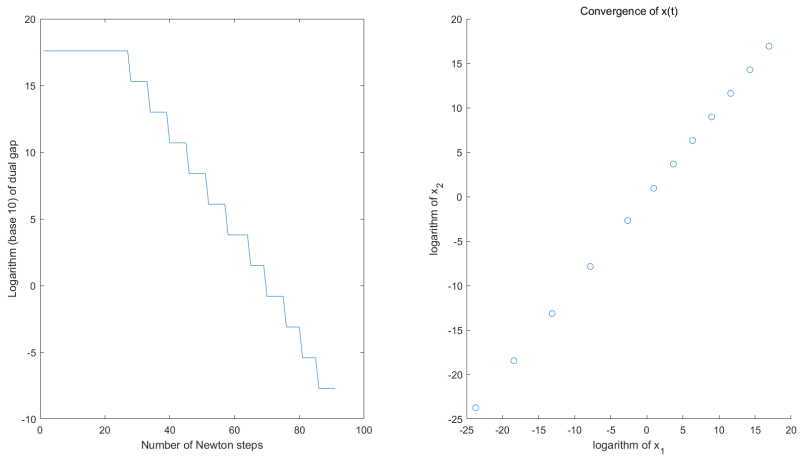

Figure 8: $\mu = 200$

```
interior point method:
9.7636e-11

1.0e-10 *

 0.4882
 0.4882
```

Combined Effect:

Balanced Selection: μ should be chosen carefully for a balanced trade-off between convergence and feasibility. A smaller value of μ may accelerate the objective function reduction but could lead to a less feasible solution, whereas a larger μ enforces feasibility at the cost of slower convergence.

Impact on Optimization Speed:

- Smaller values of μ allow for faster progress in terms of the objective function value, but they can result in violations of the constraints during intermediate steps.
- Larger values of μ result in slower objective function reduction, but ensure the solution stays feasible throughout the optimization process.

Conclusion:

- **Smaller** μ values make the interior point method more aggressive in reducing the objective function, leading to faster convergence, but they may violate feasibility in earlier iterations.
- **Larger** μ values lead to slower convergence but guarantee that the solution remains feasible during the optimization process.

Typically, the optimal value for μ needs to be tuned experimentally to balance between fast convergence and maintaining feasibility, depending on the specific problem at hand.

2.5 Conclusions

In this assignment, with reference to the textbook Convex Optimization by Boyd, I programmed and implemented the interior point method to solve a basic and simple quadratic programming problem. The process also involved methods such as Newton' s method and Backtracking Line Search. Overall, within the framework of the algorithm, I was able to obtain results that are generally close to the correct results provided by quadprog in MATLAB. However, subtle adjustments of parameters may affect the accuracy or computational speed of the results.

References

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.