Programming Exercise Report

Quadratic Programming with Interior Point Method

08023214 Zhangyunyixuan

December 21, 2025

Outline

Problem Definition

Solve the following quadratic programming problem:

$$\begin{aligned} \min_{x \in \mathbb{R}^4} \quad & \frac{1}{2} x^T Q x + c^T x, \\ \text{s.t.} \quad & a_1^T x \leq b_1, \\ & a_2^T x \leq b_2, \end{aligned}$$

Where:

$$Q = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 8 \end{pmatrix} \qquad c = \begin{pmatrix} -8 \\ -6 \\ -4 \\ -2 \end{pmatrix}$$

$$a_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad b_1 = 3 \qquad a_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad b_2 = 4$$

Step 1: Constructing the Barrier Function

The problem is transformed using a barrier function:

$$Q'(x, r) = \frac{1}{2}x^T Qx + c^T x - r\sum_{i=1}^{2} \ln(b_i - a_i^T x)$$

Step 1: Constructing the Barrier Function

The problem is transformed using a barrier function:

$$Q'(x, r) = \frac{1}{2}x^T Qx + c^T x - r \sum_{i=1}^{2} \ln(b_i - a_i^T x)$$

The gradient $\nabla_x Q'(x, r)$ is:

$$\frac{\partial Q'}{\partial x} = Qx + c + r \sum_{i=1}^{2} \frac{a_i}{b_i - a_i^T x}$$

Step 1: Constructing the Barrier Function

The problem is transformed using a barrier function:

$$Q'(x, r) = \frac{1}{2}x^T Qx + c^T x - r\sum_{i=1}^{2} \ln(b_i - a_i^T x)$$

The gradient $\nabla_x Q'(x, r)$ is:

$$\frac{\partial Q'}{\partial x} = Qx + c + r\sum_{i=1}^{2} \frac{a_i}{b_i - a_i^T x}$$

And the Hessian matrix $\mathbf{H}$ is:

$$\mathbf{H} = Q + r\sum_{i=1}^{2} \frac{a_i a_i^T}{(b_i - a_i^T x)^2}$$

The MATLAB Code for This Section

Define functions for $\nabla_x Q'(x, r)$ and $\mathbf{H}$:

```
Grad_Q_prime = @(x) (Q * x + c) - ...
    r_val * ( (-a1 / (b1 - a1' * x)) + ...
              (-a2 / (b2 - a2' * x)) );

Hessian_Q_prime = @(x) Q + r_val * ( ...
    (a1 * a1') / (b1 - a1' * x)^2 + ...
    (a2 * a2') / (b2 - a2' * x)^2 );
```

Step 2: Solving with Newton's Method

For a fixed parameter r, we find the central path by solving $\nabla_x Q'(x_k, r) = 0$.

1. **Calculate Newton step $\mathbf{p}_k$**

$$\mathbf{H}_k \mathbf{p}_k = -\nabla_x Q'(x_k, r)$$

2. **Update x:**

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$$

3. **Stopping Condition:**

$$\|\nabla_x Q'(x_k, r)\|_2 < \varepsilon$$

Step 3: The Path-Following Algorithm

The overall algorithm iteratively reduces r and solves for the new center.

```
for path_k = 1:MAX_PATH_ITER

    % Define Grad and Hessian with current r_val
    % ...

    % Solve for the central path
    x_center = newtonSolver(Grad_Q_prime, ...
               Hessian_Q_prime, x_k);

    if r_val < R_TOLERANCE
        x_opt = x_center;
        return;
    end

    % Decrease the barrier parameter
    r_val = mu * r_val;

    x_k = x_center;
end
```

Final Solution

The solution obtained by the Interior Point Method is:

$$x_{ipm} = \begin{pmatrix} 1.333 \\ 1.667 \\ 0.667 \\ 0.25 \end{pmatrix}$$

Final Solution

The solution obtained by the Interior Point Method is:

$$x_{ipm} = \begin{pmatrix} 1.333 \\ 1.667 \\ 0.667 \\ 0.25 \end{pmatrix}$$

This can be verified using MATLAB's ‘quadprog‘ function.

Result from ‘quadprog‘

The result from ‘quadprog‘ is identical, confirming our solution.

Experiment 1: Scaling the Problem Coefficients

We scale Q and c by a factor and compare performance.

```
Scale      | IPM Time (s)          | quadprog Time (s)     | IPM Obj Val           | quadprog Obj Val
------------------------------------------------------------------------------------------------------------
0.10       | 0.069796              | 0.015464              | -1.591667             | -1.591667
1.00       | 0.126931              | 0.015377              | -15.916667            | -15.916667
10.00      | 0.083019              | 0.012009              | -124.247078           | -159.166667
  - Warning: For scale 10.00  , the norm of solution difference is 1.525770e+00
100.00     | 0.174953              | 0.049813              | -1359.491383          | -1591.666667
  - Warning: For scale 100.00 , the norm of solution difference is 1.244120e+00
==========================================================================================
```

Figure: Initial Result: The method fails as scale increases.

Experiment 1: Scaling the Problem Coefficients

We scale Q and c by a factor and compare performance.

```
Scale      | IPM Time (s)         | quadprog Time (s)    | IPM Obj Val          | quadprog Obj Val
----------------------------------------------------------------------------------------------------------
0.10       | 0.069796             | 0.015464             | -1.591667            | -1.591667
1.00       | 0.126931             | 0.015377             | -15.916667           | -15.916667
10.00      | 0.083019             | 0.012009             | -124.247078          | -159.166667
  - Warning: For scale 10.00  , the norm of solution difference is 1.525770e+00
100.00     | 0.174953             | 0.049813             | -1359.491383         | -1591.666667
  - Warning: For scale 100.00 , the norm of solution difference is 1.244120e+00
==========================================================================================
```

Figure: Initial Result: The method fails as scale increases.

Reason: The initial barrier parameter r was fixed. It should be scaled along with the problem data.

Experiment 1: Corrected Results

After scaling r with the same factor, we get correct results.

Scale	IPM Time (s)	quadprog Time (s)	IPM Obj Val	quadprog Obj Val
0.10	0.060116	0.020203	-1.591667	-1.591667
1.00	0.148444	0.017543	-15.916667	-15.916667
10.00	0.107294	0.011425	-159.166667	-159.166667
100.00	0.114949	0.054408	-1591.666667	-1591.666667

Figure: Corrected Result: The method is now stable.

Experiment 2: Scaling the Problem Size

We compare run-time by increasing the number of variables (n) and constraints (m).

- $n = [10, 40, 100]$
- $m = 0.5 \times n$
- Compare run-time of our IPM vs. ‘quadprog‘.

Variables	Constraints	IPM Time (s)	quadprog Time (s)
10	5	0.000347	0.000817
40	20	0.035097	0.000835
100	50	0.073953	0.006258

Figure: Run-time comparison

Conclusion from Experiments

- **Correctness:** The implemented Interior Point Method is correct, but requires careful tuning of parameters like the initial barrier value r.
- **Performance:** Function ‘quadprog‘ is faster. As the problem size increases, the performance of the interior-point method is becoming better and better, showing that the interior-point method is efficient for large-scale problems.

Thank You!