Report of interior methods (predictor-corrector algorithm) implementation in MATLAB

吴嘉骜 21307130203

2023 年 6 月 27 日

1 Introduction

Interior methods are a type of algorithms for solving linear programming problems. By the early 1990s, a subclass of interior-point methods known as *primal − dual methods* had distinguished themselves as the most efficient practical approaches, and proved to be strong competitors to the simplex method on large problems. This report will focus on a *practical predictor −corrector algorithm* which is a particular interior-point method proposed by Mehrotra.

Purpose of the experiment

The purpose of this experiment is to understand the principles behind the primal-dual method and to demonstrate the implementation of predictor-corrector algorithm in MATLAB.

Environment of the experiment

MATLAB R2022a on Windows 11.

Brief overview of the report

We will begin by describing the methodology used to implement the interior method in MATLAB and present the results obtained from the implementation.
Then, we will discuss the implications of the findings and the difficulites confronted with.
Finally, we will draw conclusions and provide recommendations for future improvement.

2 Experiment design and methodology

Find a starting point

$[x_0, \lambda_0, s_0]$ = starting_point(A, b, c) finds a starting point (x_0, λ_0, s_0).

Inputs:
A: $m \times n, m < n$, input matrix, with full row rank.
b: $m \times 1$ the right hand of equality constraints.
c: $n \times 1$ the cost coefficients of the problem.

Calculate the affine scaling direction

The affine scaling direction (Δx^{aff}, $\Delta \lambda^{aff}$, Δs^{aff}) is defined by:

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^{aff} \\ \Delta \lambda^{aff} \\ \Delta s^{aff} \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ -r_{xs} \end{bmatrix}$$

Here $r_{xs} = XSe$.
affine_direction(x, λ, s, A, b, c) solves the affine scaling direction, by transforming the above equation into an augmented system.

Calculate the step length

$[\alpha^{pri}, \alpha^{dual}]$ = step_length(x, s, Δx, Δs, η) calculates the step length.

Update step

$[x_{new}, \lambda_{new}, s_{new}]$ = update(x, λ, s, A, b, c, η) updates the current point. Calculate the step length, centering paremeter and the affine scaling direction as above. Then, we can update the current point and continue the iteration.

Main function

[optsol, optval, info] = main(A, b, c, option) is the full implementation of the interior method.
Inputs:
A: $m \times n, m < n$, input matrix, with full row rank.
b: $m \times 1$ the right hand of equality constraints.
c: $n \times 1$ the cost coefficients of the problem.
option: a struct that contains: [tol, η, $maxiter$]

Outputs:
optsol: optimal solution of the primal problem (if the problem is infeasible or unbounded, then it will be returned as $NULL$.)
optval: optimal value (if the problem is infeasible, it will be returned as $NULL$; if unbounded, it will be returned as ∞.)
info: a struct that contains: [status, iteration number].

Intermediate process:
1. Calculate the inital point (x_0, λ_0, s_0).
2. Check the current status: determine whether the problem is unbounded or infeasible by calculating infeasibilities r_b, r_c.
3. Iteration: calculate the affine scaling direction, step length and update

the current point.
4. Check the optimality condition: if the duality gap is small enough, return the optimal value and solution.
5. If not converged, return to step 3-5.
6. Print the results in the command window.

3 Numerical results

We will test the implementation on several linear programming problems. Note that the *Gurobi* solver (*linprog*) is used in every problem to check if the output is right.

3.1 A simple case

The *tol* is set to be 10^{-9}, $\eta = 0.95$ and the maximum iteration is set to be 1000.
Consider the following linear programming problem:

$$\begin{aligned} \textit{minimize} \quad & 2x_1 + 3x_2 + 3x_3 + x_4 - 2x_5 \\ \textit{subject to} \quad & x_1 + 3x_2 + 4x_4 + x_5 = 2 \\ & x_1 + 2x_2 - 3x_4 + x_5 = 2 \\ & -x_1 - 4x_2 + 3x_3 = 1 \\ & x_1, \ldots, x_5 \geq 0. \end{aligned}$$

Use the *Modified Cholesky Factorization*(*modChol*) to calculate the affine direction, the output results are:
————————Problem 1: Simple Case (ex3.17)————————
This problem has an optimal solution.
The optimal value is: -3.000000

Optimal solution found.

optimal value by Gurobi: -3.000000

3.2 An infeasible case

The *tol* is set to be 10^{-9}, $\eta = 0.95$ and the maximum iteration is set to be 1000.

Consider the following linear programming problem:

$$\begin{aligned} minimize \quad & 20x_2 + 13x_4 \\ subject\,to \quad & x_1 + 2x_2 + 2x_4 = 100 \\ & x_3 + 3x_4 = -2 \\ & 8x_2 + 6x_4 + x_5 = 5 \\ & x_1, \ldots, x_5 \geq 0. \end{aligned}$$

The output results are:

————————Problem 2: Infeasible Case————————

This problem is infeasible!

No feasible solution found.

Linprog stopped because no point satisfies the constraints.

optimal value by Gurobi:

3.3 An unbounded case

The *tol* is set to be 10^{-9}, $\eta = 0.95$ and the maximum iteration is set to be 1000.

Consider the following linear programming problem:

$$\begin{aligned} minimize \quad & -2x_1 - 3x_3 + 5x_4 + x_5 + 3x_6 \\ subject\,to \quad & x_1 - x_2 + x_3 + x_4 - x_6 = 5 \\ & -2x_1 + x_2 + x_5 = 3 \\ & x_2 - 2x_3 + 2x_4 + x_6 = 7 \\ & x_1, \ldots, x_6 \geq 0. \end{aligned}$$

The output results are:

————————Problem 3: Unbounded Case————————

This problem is unbounded!

Problem is unbounded.

optimal value by Gurobi:

3.4 Random case

Use MATLAB to randomly generate several linear programming problems with m constrains and n variables.
For instance, we use seed 07130203 to generate the random numbers, and set $m = 200, n = 300$.
The *tol* is set to be 10^{-8}, $\eta = 0.9$, and the maximum iteration is set to be 10000.
The output results are:
————————Problem 4: Random Case————————
This problem is infeasible!
No feasible solution found.
Linprog stopped because no point satisfies the constraints.
optimal value by Gurobi:

3.5 Dataset: lp_capri

Download the dataset *lp_capri* from the website: https://sparse.tamu.edu/LPnetlib.
The *tol* is set to be 10^{-8}, $\eta = 0.9$, and the maximum iteration is set to be 1000.
Use *linsolv* to calculate the affine direction, the output results are:
————————Problem 5: lp_capri————————
This problem has an optimal solution.
The optimal value is: 904.296955

Optimal solution found.
optimal value by Gurobi: 904.296954

4 Discussion

4.1 Analysis of the results

The results are consistent with the theoretical analysis, and are also consistent with the results of the MATLAB solver *linprog*. But there are not

without tricky problems.

First and foremost, the calculation of affine scaling direction is mostly by applying Cholesky factorization to AD^2A^T, but the matrix can be ill-conditioned or singular. Ill conditioning of this system is often observed during the final stages of the algorithm, when the elements of the diagonal weighting matrix D^2 take on both huge and tiny values. The Cholesky technique may encounter diagonal elements that are very small, zero or (because of roundoff error) slightly negative.
For example, in Problem 1, at iteration number 8, x is:
7.28307265146803e-10
1.66028402521730e-10
0.333333333722446
1.04899713893160e-10
1.99999999936688
However, in iteration 9, the elements in x suddenly become rather huge, and the problem is determined as unbounded:
3.65446008508403e+24
1.43543218869267e+43
1.91390958492356e+43
1.46393113200396e+44
1.44799303737779e+34
So the program may not be stable.
To solve the problem, we implement *modChol*, which skips a step of the factorization, setting the component of $\Delta\lambda$ that corresponds to the faulty diagonal element to zero. With this approach, we successfully solve the problem.

Second, when testing with rather large problems such as Problem 5, *modChol* is rather slow, perhaps because our implementation is not efficient enough. So we change back into the original *linsolv*, the MATLAB solver, and the program runs much faster with correct results.

Third, for unbounded and infeasible cases, our criterion to determine the status has not been well tested, and there is still obscurity. We only know for sure that when presented with such two cases, the algorithm typically diverges, with the infeasibilities r_b^k and r_c^k and/or the duality measure μ_k going to ∞. But the rigorous approach to clarify the exact status out of the two is still not clear. As a preliminary idea, we can determine the status ahead of time by other methods, such as the simplex method.

In all, the program is can work for feasible cases with finite optimal value by choosing proper equation solvers. But for other cases, the program may not be stable and efficient enough.

4.2 Conclusion

This experiment preliminarily implements the primal dual simplex method (predictor−corrector algorithm) for linear programming. The results are consistent with the theoretical analysis and the results of the MATLAB solver. This program can be used to solve linear programming problems efficiently.

4.3 Future work

A detailed analysis of the stability of the program can be conducted, and with more knowledge of the algorithm, a more robust interior method can be implemented. More requirements such as handling the rank deficient constraints can be satisfied in the program. We can also practice other interior methods such as the path following method, to compare the efficiency and stability of different methods. The code style and the efficiency of the program can be improved so as to accelerate the solving process.

5 References

Jorge Nocedal, Stephen J. Wright, Numerical Optimization, 2006.
S. J. Wright, Primal-Dual Interior-Point Methods, SIAM Publications, Philadel-

phia, PA, 1997.
Modified Cholesky factorizations in interior-point algorithms for linear programming, S. J. WRIGHT, SIAM Journal on Optimization, 9 (1999).