Report of the two-phase simplex method implementation in MATLAB

吴嘉骜 21307130203

2023 年 5 月 10 日

1 Introduction

Linear programming is a mathematical technique used to optimize a linear objective function subject to constraints represented as linear equations or inequalities. The simplex method is a widely used algorithm for solving linear programming problems, developed by George Dantzig in 1947. Since then, the simplex method has been used in a variety of applications.

Purpose of the experiment

The purpose of this experiment is to understand the principles behind the simplex method and to demonstrate its implementation in MATLAB.

Environment of the experiment

The simplex method is implemented in MATLAB R2022a on Windows 11.

Brief overview of the report

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

2 Experiment design and methodology

Anti-cycling and pivoting rule

We choose the Bland's rule, namely the smallest subscript pivoting rule:
1. Find the smallest j for which the reduced cost $\overline{c}_j$ is negative and have the column A_j enter the basis.
2. Out of all variables x_i that are tied in the test for choosing an exiting variable, select the one with the smallest value of i.

Phase I

[Phase I.m] is the implementation of the first phase of simplex method, which finds the inital basic feasible solution.
Inputs:
A: $m \times n, m < n$, input matrix, not necessarily with full rank.
b: $m \times 1$ the right hand of equality constraints.
tol: the tolerance to set zero, such as 10^{-10}.
Outputs:
basis index: a row vector that consists of basic indices of the original A (after removing redundant rows).
status: feasible or infeasible
a flag which shows whether A is linearly dependent.
refreshed A, b if necessary.
Intermediate process:
1. Check if b is nonnegative, if not, multiply some of the constraints by -1.
2. Add artificial variables y to the problem, leading to new A and c.
3. Solve the auxiliary problem by simplex method (phase II).
4. If the optimal cost is positive, the original problem is infeasible, and we return.
5. If the optimal cost is zero, a feasible solution to the original problem has been found. Now check if there is any artificial variable in the final basis. If not, change the ith artificial variable into an original variable by finding a non-zero element in the ith row.
6. If we cannot find a non-zero element in the ith row, then the ith row is

redundant, and we delete it. Modify the flag which shows that A is linearly independent.
7. Repeat step 5 and 6 until all artificial variables are driven out of the basis. Return eliminated A, b.

Phase II

[Phase II.m] is the implementation of the second phase of simplex method, which is also used to update the tableau in phase I.
Inputs:
A: $m \times n, m < n, rank(A) = m$.
b: $m \times 1$ the right hand of equality constraints.
c: $n \times 1$ the cost coefficients of the problem.
basis index: a row vector that consists of basic indices of A.
tol: the tolerance to set zero, such as 10^{-10}.
Outputs:
optsol: optimal solution
bas index: a row vector that consists of basic indices of the optimal solution.
optval: optimal value
status: optimal, unbounded or exceeding max iteration
Intermediate process:
1. Form the initial tableau.
2. Check if the current tableau is optimal. If not, find the entering variable.
3. Check if the problem is unbounded by examining the entering column. If true, return the status and terminate. Set the optimal value to $-\infty$.
4. Find the exiting variable by ratio test (pivoting rule).
5. Update the basis index.
6. Update the tableau.
7. Repeat step 2 to 6 until the tableau is optimal or the maximum iteration is reached. Return the status and the optimal value and solution.

Main function

[main.m] is the full implementation of the two-phase simplex method.
Inputs:
A: $m \times n, m < n$, input matrix, not necessarily with full rank.
b: $m \times 1$ the right hand of equality constraints.
c: $n \times 1$ the cost coefficients of the problem.
tol: the tolerance to set zero, such as 10^{-10}.
Outputs:
optsol: optimal solution (if the problem is infeasible or unbounded, then it will be returned as $NULL$.)
B: the basis corresponding to the optimal solution.
optval: optimal value (if the problem is infeasible, it will be returned as $NULL$; if unbounded, it will be returned as $-\infty$.)
status: optimal, unbounded, infeasible or exceeding max iteration
Intermediate process:
1. Carry out phase I.
2. If the problem is infeasible, return the status and terminate.
3. If the problem is feasible, carry out phase II.
4. Return the status, optimal value and solution.
5. Print the results in the command window.

3 Numerical results

The tol is set to be 10^{-10}, and the maximum iteration is set to be 1000.

3.1 A simple case

Consider the following linear programming problem:

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

The output results are:

The problem status is optimal.

The optimal cost is -3.000000.

The optimal solution is: $[0, 0, 0.3333, 0, 2.0000]^T$.

The corresonding basis is:

$$B = \begin{bmatrix} 0 & 4 & 1 \\ 0 & -3 & 1 \\ 3 & 0 & 0 \end{bmatrix}.$$

3.2 A rank-deficiency case

Consider the following linear programming problem:

$$\begin{aligned} \textit{minimize} \quad & x_1 + x_2 + x_3 \\ \textit{subject to} \quad & x_1 + 2x_2 + 3x_3 = 3 \\ & -x_1 + 2x_2 + 6x_3 = 2 \\ & 4x_2 + 9x_3 = 5 \\ & 3x_3 + x_4 = 1 \\ & x_1, \ldots, x_4 \geq 0. \end{aligned}$$

The output results are:

A has linearly dependent rows.

The constrains can be eliminated to:

$$A = \begin{bmatrix} 1 & 2 & 3 & 0 \\ -1 & 2 & 6 & 0 \\ 0 & 0 & 3 & 1 \end{bmatrix}, \; b = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}.$$

The problem status is optimal.
The optimal cost is 1.750000.
The optimal solution is: $[0.2500, 0.5000, 0, 1.0000]^T$.
The corresonding basis is:

$$B = \begin{bmatrix} 1 & 2 & 0 \\ -1 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

3.3 An infeasible case

Consider the following linear programming problem:

$$\begin{aligned} \textit{minimize} \quad & x_1 + 20x_2 + 85x_3 + 13x_4 + x_5 \\ \textit{subject to} \quad & x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 = 100 \\ & -x_1 - 2x_2 - 3x_3 - 55x_5 = 2 \\ & 9x_1 + 8x_2 + 7x_3 + 6x_4 + 5x_5 = 5 \\ & x_1, \ldots, x_5 \geq 0. \end{aligned}$$

The output results are:
The problem is infeasible!

3.4 An unbounded case

Consider the following linear programming problem:

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

The output results are:
The problem is unbounded from below.

3.5 random cases

Use MATLAB to randomly generate several linear programming problems with m constrains and n variables. The correctness can be verified by the solver CVX. Codes can be found in the appendix.

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 CVX.
For all cases of the experiment, including the randomly generated cases, the results are correct, which shows the robustness of the program.
The numerical examples are all simple, stable with a rather small scale, so the efficiency of the program is not tested.

4.2 Conclusion

This experiment preliminarily implements the simplex method for linear programming. The results are consistent with the theoretical analysis and the results of the MATLAB solver. This program is robust and can be used to solve small-scale linear programming problems. In all, the simplex method is a very useful method for solving linear programming problems.

4.3 Future work

More requirements such as solving the general form of linear programming problems, presenting historical information of the intermediate process, and so on, can be satisfied in the program. The code style and the efficiency of the program can be improved so as to accelerate the solving process. Additionally, the two-phase tableau implementation may not be a good way to tackle large-scale problems and can be replaced by other methods such as the revised simplex method.

5 References

Dimitris Bertsimas and John Tsitsiklis. 1997. Introduction to Linear Optimization (1st. ed.). Athena Scientific.

6 Appendix

A Phase I

```
% Phase I of the two-phase simplex method.
% input: A, b, tol
% A: m<n, full rank not a must
% output: bas_index, status, A_line_ind, rm
% rm: indices of redundant constraints that should be removed.
% status could be 0(feasible), 3(infeasible)

function [bas_index, status, A_ind, rm]=Phase_I(A,b,tol)
    [m,n]=size(A);
    A_ind = 1; %indicate the linear independence of A's rows
    r = 0; rm=[];

    % check if b is nonnegative
    for i=1:m
        if b(i) < 0
            A(i,:) = -A(i,:);
            b(i) = -b(i);
        end
    end

    % introduce artificial variables
    A = [A, eye(m)];
    c = [zeros(1,n), ones(1,m)]';
    bas_index = n+1:n+m;

    % solve the phase I problem (by means of phase II)
    [~, bas_index, optval, ~, T] = Phase_II(A, b, c, bas_index, tol);
    if optval > tol
        status = 3;
        return;
    else
        status = 0;
        % remove artificial variables
        if isempty(find(bas_index > n, 1))
            return;
        else
            for i=1:m
                if bas_index(i) > n
                    % find a non-zero element in the i-th row
                    for l=2:n+1
                        if T(i,l) ~= 0
                            bas_index(i) = l-1;
                            l = 0;
                            break;
```

```
                    end
                end
                if l ~= 0
                    r = r+1;
                    A_ind = 0;
                    % the i-th row should be removed
                    rm(r) = i;
                end
            end
        end
        bas_index(rm)=[];
    end
  end
end
```

B Phase II

```
% Phase II of the two-phase simplex method. (also core of the simplex method)
% rank(A)=m with known basis index
% input: A, b, c, bas_index, tol
% output: optsol, bas_index, optval, status, T
% status could be 1(optimal), 2(unbounded), 4(maxiteration)

function [optsol, bas_index, optval, status, T] = Phase_II(A, b, c, bas_index, tol)
    [m,n] = size(A);
    B = A(:, bas_index);
    c_b = c(bas_index);
    maxiter = 10000;

    %form the initial tableau
    T3 = B\b;
    T4 = B\A;
    T1 = -c_b'*T3;
    T2 = c' - c_b'*T4;
    T0 = [T1,T2];
    T = [T3,T4];

    %iterate
    for k = 1:maxiter
        % check if the current basis is optimal
        if all(T0(2:end) >= -tol)
            status = 1;
            optval = -T0(1);
            optsol = zeros(n,1);
            B = A(:, bas_index);
            x_b = B\b;
            optsol(bas_index) = x_b;
            return;
        end

        % find the entering variable
        for j = 2:n+1
            if T0(j) < -tol
                enter = j;
                break;
            end
        end

        % check if the problem is unbounded
        if all(T(:,enter) <= tol)
            status = 2;
```

```
            optval = '-inf';
            optsol = [];
            bas_index = [];
            return;
        end

        % find the leaving variable
        ratio = '0'; % the ratio of the leaving variable and the pivot element
        for i = 1:m
            if T(i,enter) > tol
                temp = T(i,1)/T(i,enter);
                if ratio == '0'
                    ratio = temp;
                    exit = i;
                else
                    if temp < ratio
                        ratio = temp;
                        exit = i;
                    end
                end
            end
        end

        % update the basis
        bas_index(exit) = enter - 1;

        % update the tableau
        T(exit,:) = T(exit,:)/T(exit,enter);
        for i = 1:m
            if i ~= exit
                T(i,:) = T(i,:) - T(exit,:)*T(i,enter);
            end
        end
        T0 = T0 - T(exit,:)*T0(enter);
    end
    if k == maxiter
        status = 4;
    end
end
```

C main

```
% full implementation of the two-phase simplex method
% input: A, b, c, tol
% output: optsol, B, optval, status
% status: feasible=0, optimal=1, unbounded=2, infeasible=3, maxiteration=4
% B: basis

function [optsol, B, optval, status] = main(A,b,c,tol)

    % Phase I
    [bas_index, status, A_ind, rm] = Phase_I(A,b,tol);
    if status == 3
        optsol = [];
        B = [];
        optval = [];
        fprintf('The problem is infeasible!\n');
        return
    end

    % report redundancy
    if A_ind == 0
        fprintf("A has linearly dependent rows.\n")
        A(rm,:) = [];
        b(rm) = [];
        fprintf("The constrains can be eliminated to:\n")
        A,b
    end

    % Phase II
    [optsol, bas_index, optval, status, ~] = Phase_II(A, b, c, bas_index, tol);
    B = A(:, sort(bas_index));

    % result of the problem
    switch status
        case 1
            fprintf("The problem status is optimal.\n");
        case 2
            fprintf("The problem is unbounded from below.\n");
        case 3
            fprintf("The problem is infeasible.\n")
        otherwise
            fprintf("Exceed the max iteration.\n")
    end

    % output the optimal case
```

```
    if status == 1
        fprintf('The optimal cost is %f.\n', optval);
        fprintf('The optimal solution is:\n');
        optsol
        fprintf('The corresonding basis is:\n');
        B
    end
end
```

D test

```
% test with several cases

%% -------------------Problem 1-------------------
clc,clear;
tol = 1e-10;
A = [1,3,0,4,1;1,2,0,-3,1;-1,-4,3,0,0];
b = [2;2;1];
c = [2;3;3;1;-2];
fprintf('-----------------Problem 1: Simple Case (ex3.17)----------------\n\n')
main(A,b,c,tol);

%% -------------------Problem 2-------------------
clc,clear;
tol = 1e-10;
A = [1,2,3,0;-1,2,6,0;0,4,9,0;0,0,3,1];
b = [3;2;5;1];
c = [1;1;1;0];
fprintf('-----------------Problem 2: Rank-deficiency Case (eg3.9)----------------\n\n')
main(A,b,c,tol);

%% -------------------Problem 3-------------------
clc,clear;
tol = 1e-10;
A = [1,2,3,4,5;-1,-2,-3,0,-55;9,8,7,6,5];
b = [100;2;5];
c = [1;20;85;13;1];
fprintf('\n-----------------Problem 3: Infeasible Case----------------\n\n')
main(A,b,c,tol);

%% -------------------Problem 4-------------------
clc,clear;
tol = 1e-10;
A = [1,-1,1,1,0,-1;-2,1,0,0,1,0;0,1,-2,2,0,1];
b = [5;3;7];
c = [0;-2;-3;5;1;3];
fprintf('\n-----------------Problem 4: Unbounded Case----------------\n\n')
main(A,b,c,tol);

%% -------------------Problem 5-------------------
clc,clear;
tol = 1e-10;
m = 5;
n = 15;
A = randn(m, n);
```

```
b = randn(m, 1);
c = randn(n, 1);
fprintf('\n-----------------Problem 5: Random Case-----------------\n\n')
main(A,b,c,tol);

cvx_begin
    variable x(15)
    minimize(c'*x)
    subject to
        A*x==b;
        x>=0;
cvx_end
```