Bingwei Ge (40115253)

Project Design of Optimization Techniques

1. Problem Description

The problem requires to program an algorithm in Matlab using sequential unconstrained optimization to find the maximum volume rectangle R that it contained in a polytope P. The rectangle and the polytope are defined as follows:

$$R = \{x \in \Re^n \mid l \le x \le u\}$$
$$P = \{x \in \Re^n \mid Ax \le b\}$$

where l, u are lower and upper bound vectors respectively. A,b have appropriate dimensions and b > 0.

2. Derivation of Constraint Description

The constraint of the rectangle being contained in the polytope can be described by $A^+u - A^-l \le b$, where $A_{ij}^+ = x(0, A_{ij}), A_{ij}^- = x(0, -A_{ij})$ and use as an optimization functional $-\sum_i \log(u_i - l_i)$. The equations below wii show the derivation of this conclusion.

At first, we need to prove $P = \{x \in R^n | Ax \le b\} \Leftrightarrow P = \{x \in R^n | (A^+ - A^-)x \le b\}$ by dividing matrix A into two parts: the generalized positive part and negative part.

For the positive prat, every element A_{ij} in A is greater or equal to 0, so we get $A_{ij}^+ = \max(0, A_{ij}) = A_{ij}$. It is obvious that all $-A_{ij} \le 0$ in terms of this part. We can get $A_{ij}^- = \max(0, -A_{ij}) = 0$. So this part can be written as $A_{ij} = A_{ij} - 0 = A_{ij}^+ - A_{ij}^-$. Since the same situation occurs in the negative part, we can express A as $A_{ij} = A_{ij}^+ - A_{ij}^-$. We can replace the A in $P = \{x \in \Re^n \mid Ax \le b\}$ then we have $P = \{x \in \Re^n \mid (A^+ - A^-)x \le b\}$.

The second step is to prove $(A^+ - A^-)x \le b \Leftrightarrow A^+u - A^-l \le b$ when $l \le x \le u$. Firstly we need to assume that $A^+u - A^-l \le b$. Because A^+ and A^- are both greater or equal to 0, we can get $A^+x \le A^+u$, $-A^-x \le -A^-l$. Then we can see $(A^+ - A^-)x \le A^+u - A^-l \le b$. If we assume that $A^+u - A^-l > b$ and $(A^+ - A^-)x \le b$, we can infer that $A^+u - A^-l > (A^+ - A^-)x$, after transformation there is $A^+(u - x) + A^-(x - l) > 0$, but when x reach

the boundary u or l, $A^{-}(x-l)>0$ or $A^{+}(u-x)$ needs to be true, but A^{+} and A^{-} both greater or equal to 0. When they equal to 0, the equation above is not true. Finally we prove the two constraints at the beginning of this chapter are equivalent.

3. Analysis of Solving Algorithm

3.1 Assumption

We assume that the given half - space constraints must be able to form convex polytope in N dimensional space, which means that the target set decided by A and b cannot be open and it has the same dimension with the space (i.e. if the constraints form a plane in three dimension space, it is invalid). That is the only hypothesis we make in this project.

3.2 Solve the Vertexes

First of all, we need to find the vertexes of the polytope, which is not only important for determining the initial point of optimization, but also for plotting the final result. We can think the given A and b as the parameters of the n-dimensional system of linear equations. If we find the solution then we get the points. In N dimensional space we need N equations to get to a certain point if it exists. For example, two lines in the same plane may define a point.

Suppose the size of matrix A is 5x3, in other words, there are five equations in 2 dimensional space. We can get the number of combination of equations is C_5^2, then we find all feasible solutions of these combinations. Then we need to keep the points that satisfy the constraints and get rid of duplicate values. Finally we get the vertexes of polytope. Fig 1 shows how this algorithm works.

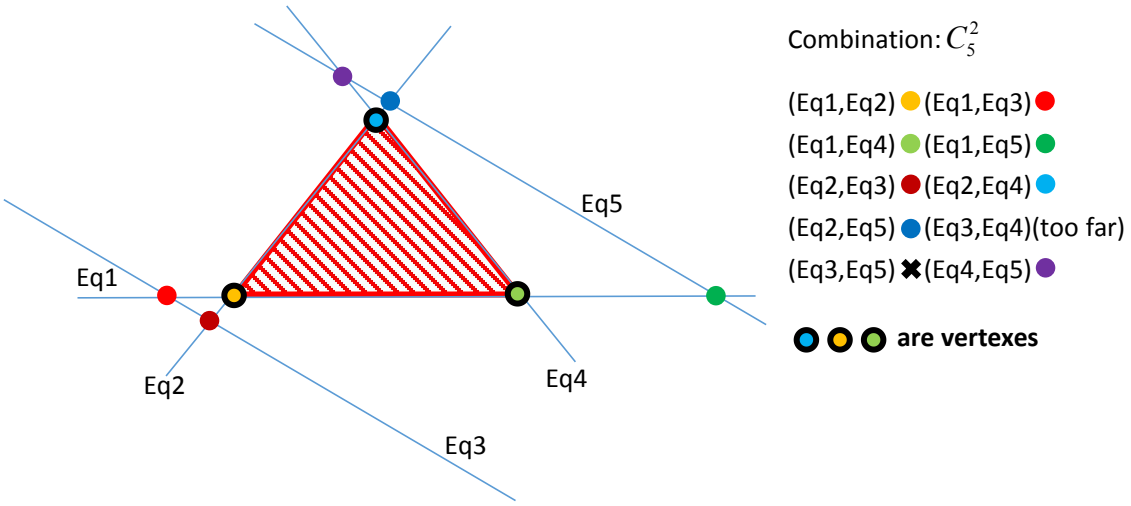

Fig.1 how to find the vertexes of the polytope

3.3 Determine Initial Points

Then we will talk about how to choose the start points. First we need to find the center of the polygon by $X = [x_1 \quad x_2 \quad \dots \quad x_n]^T$, where $x_i = \frac{1}{n}\sum_{j=1}^{n} x_{ij}$. The center point minus a small number could be initial low point of rectangle and the center point plus the same number could be the up point of the rectangle. But when the optimal rectangle is quite small, the low and up points may be out of the feasible area. So we need to judge the generated points satisfy the constrains or not. If the points are not feasible, the program can make the number smaller until the points is in the feasible space.

Fig.2 explains how to choose the initial points. Once the initial points are selected, this problem can be optimized by Newton's method combined with Interior-point method.

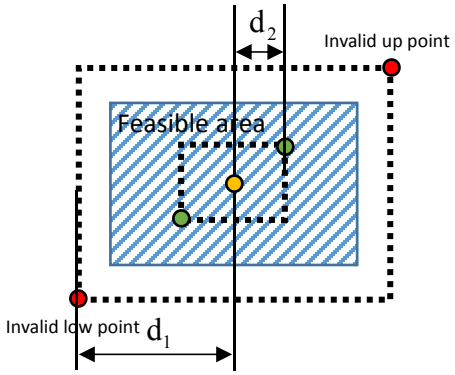

Fig.2 how to choose initial points

3.4 Description of Optimization Algorithm

The optimization algorithm itself is described in the lecture note[1], so we will not go into details here. But program implementation of the algorithm needs more attention.

In order to realize the derivation and Hessian matrix of any objective function as well as constraint functions, we use the symbolic calculation in Matlab, and then we can replace the variable with values to get the result. However, the result of the operation of all symbol variables is stored as text, which means the efficiency of the program is very low when the calculation result is complicated. Therefore, we have to give up some precision and use the ***vpa function*** [2] to convert the text result to floating with finite number of digits, so that we compute the result faster. It is a kind of balance.

Fig.3 demonstrates the results of symbolic calculation(take testing A and b for example), such as objective function(G), penalty function(phi), optimization function(f), the derivation and Hessian matrix of f.

```
G =

 U2 - L1 - 3
 U1 + U2 - 3
         -L2
      U2 - 2


phi =

- log(3 - U2 - U1) - log(2 - U2) - log(L2) - log(L1 - U2 + 3)


f =

- log(U1 - L1) - log(U2 - L2)


F =

- log(3 - U2 - U1) - log(2 - U2) - log(L2) - 100*log(U1 - L1) - 100*log(U2 - L2) - log(L1 - U2 + 3)


F1 =

[ - 100/(L1 - U1) - 1/(L1 - U2 + 3), - 1/L2 - 100/(L2 - U2), 100/(L1 - U1) - 1/(U1 + U2 - 3), 100/(L2 - U2) - 1/(U1 + U2 - 3) - 1/(U2 - 2) + 1/(L1 - U2 + 3)]


F2 =

[ 100/(L1 - U1)^2 + 1/(L1 - U2 + 3)^2,                      0,                 -100/(L1 - U1)^2,                                                    -1/(L1 - U2 + 3)^2]
[                                   0, 1/L2^2 + 100/(L2 - U2)^2,                                0,                                                     -100/(L2 - U2)^2]
[                    -100/(L1 - U1)^2,                      0, 1/(U1 + U2 - 3)^2 + 100/(L1 - U1)^2,                                                  1/(U1 + U2 - 3)^2]
[                  -1/(L1 - U2 + 3)^2,       -100/(L2 - U2)^2,                1/(U1 + U2 - 3)^2, 1/(U2 - 2)^2 + 1/(U1 + U2 - 3)^2 + 100/(L2 - U2)^2 + 1/(L1 - U2 + 3)^2]
```

Fig.3 The results of symbolic calculation

Step size selection for Newton's method is another important problem. when we use the fixed step size, if it is small, the convergence rate of the algorithm in the early stage will be slow. If the step size is big, the new point generated by the algorithm may beyond the feasible domain. So we choose the variable step size, If a new point is detected that does not meet the constraint, the step size is updated by exponential decay and the next point is recalculated.

About the plotting, If we have the vertexes of the polytope we can compute the convex hull, which is used to form the image of the polytope.

4. The Test Results

We used two-dimensional and three-dimensional constraints to test the program, respectively. The fig.4 below show the final result.

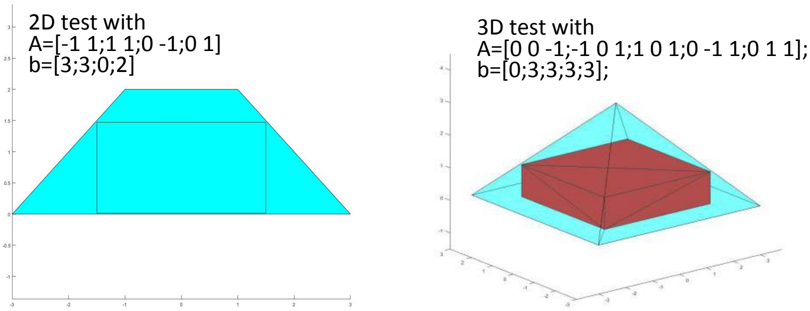

Fig.4 The final test results

5. Reference List

[1] Luis Rodrigues, "Optimization – Lecture 8 Numerical Algorithms." [Online]. Available: https://moodle.concordia.ca/moodle/pluginfile.php/3790881/mod_resource/content/1/Optimization_Lecture8.pdf. [Accessed: 23-Nov-2019].

[2] mycoolcn, "Matlab control the calculation precision using the digits(A) and vpa(B) function." [Online]. Available: https://blog.csdn.net/mycoolcn/article/details/77397291. [Accessed: 23-Nov-2019].

[3] Sophia_Dz, "MATLAB permutation combination function --nchoosek." [Online]. Available: https://blog.csdn.net/facetosea1/article/details/83573155. [Accessed: 23-Nov-2019].

[4] gcponly, "Matlab symbolic operation." [Online]. Available: https://wenku.baidu.com/view/e488e5decc7931b764ce15c4.html. [Accessed: 23-Nov-2019].

[5] JiYuNan, "Matlab optimization - Newton method." [Online]. Available: https://wenku.baidu.com/view/41c0e0ed85254b35eefdc8d376eeaeaad0f3165b.html. [Accessed: 23-Nov-2019].

[6] Nazanin Hashemi-A, "Optimization." [Online]. Available: https://github.com/nhashemia/Optimization/blob/master/maxvolrectangle.m. [Accessed: 23-Nov-2019].

6. Appendix

6.1 The Function Code

```
function [low,up,volume]=maxvolrectangle(A,b)
% Calculate the maximum inner rectangle of the polygon,
% where A and b are the semi-space constraints forming convex polygons or
% polyhedrons.
[row,col] = size(A);
combination = nchoosek(1:row,col); % Find the combinations of rows in matrix A
[m,n] = size(combination);
points =[];
for i = 1:m
    A_star = A(combination(i,:),:); % Extract the number of rows corresponding to the combination label
    if rank(A_star)~= n % Check there is a solution or not
        continue; % If no solution,next combination
    else
        x = A_star\b(combination(i,:)); % If there is solution,give the answer
        if all(A*x<=b) % If the point is inside the polygon, it is the boundary point of the polygon
            points =[points x];
        end
    end
end
points =(unique(points','rows'))'; % Remove duplicate boundary points
[row_s,col_s]=size(points);
if row_s >= col_s
    error('The provided constraints cannot form a polygon or polyhedron')
```

```
end
% Compute A- and A+
Ap = zeros(row,col);
Am = zeros(row,col);
for i = 1:row
    for j = 1:col
        Ap(i,j)=max(A(i,j),0);
        Am(i,j)=max(-A(i,j),0);
    end
end
% Define general variables
K=100;
delta = 1e-10;% Inner loop closure condition
epsilon =1e-8;% Outer loop closure condition
% Significant digits is 10
digits(10)
% Define symbol variable
L=sym('L',[row_s 1]);
U=sym('U',[row_s 1]);
% Give the initial value
center = mean(points,2); % Calculate the center of the polygon to determine the position of the initial point
init = 1; % The initial point parameter ensures that the initial point is inside the polygon
X0 = [center-init;center+init];
while ~(all(A*X0(1:row_s)< b) && all(A*X0(row_s+1:2*row_s)< b))
    init = 0.9*init;
    X0 = [center-init;center+init];
end
% Construct penalty function
G = Ap*U-Am*L-b;
phi = sum(-log(-G));

% Function to be optimized
f = sum(-log(U-L));
F = K*f + phi;
% Solve for the gradient and the hessian matrix
F1 = jacobian(F,[L.',U.']);
F2 = jacobian(F1,[L.',U.']);
% Newton method and interior point method iterative optimal value X0
while(1)
    t=1;
    while(1)
        % The values of the gradient and the hessian matrix are computed numerically
        F1_X = vpa(subs(F1,[L.',U.'],X0'));
        F2_X = vpa(subs(F2,[L.',U.'],X0'));
```

```
            % Calculate the next point
            X1 = X0 - t*(F2_X\(F1_X'));
            % Determine whether the next point is a feasible solution,
            % if not, update step size by exponential attenuation and the attenuation rate of 0.9
            while ~(all(A*X1(1:row_s)< b) && all(A*X1(row_s+1:2*row_s)< b))
                t=t*0.9;
                X1 = X0 - t*(F2_X\(F1_X'));
            end
            % Calculate the function values of the last feasible point and the current feasible point
            f0 = vpa(subs(f,[L.',U.'],X0'));
            f1 = vpa(subs(f,[L.',U.'],X1'));
            % Compare the value of Euclidean norm with last value
            n = norm(f1-f0);
            % If the condition is met to end the current loop, otherwise the current optimal solution is set to a new X0
            if n < delta
                break;
            else
                X0=X1;
            end
        end
        % Determine whether the outer loop meets the end condition, otherwise update the value of K and then calculate
        if row/K < epsilon
            break;
        else
            K=K*10;
        end
    end
    low = double(X0(1:row_s));
    up = double(X0(row_s+1:2*row_s));
    % plotting
    hold on
    axis equal
    if row_s == 2
        patch(points(1,:),points(2,:),'c')
        pos =double([X0(1:2)' (X0(3:4)-X0(1:2))']);
        rectangle('Position',pos)
        volume = prod(double(X0(row_s+1:2*row_s)-X0(1:row_s)));
    elseif row_s == 3
        view(3) % Open 3D view
        P1 = points';
        K1 = convhulln(P1); % Calculate the convex hull
        patch('Vertice',P1,'Faces',K1,'FaceColor','c','FaceAlpha',.3) % Construct the polytope
        % Build therectangular vertexes
        P2=[ X0(1),X0(1),X0(1),X0(1),X0(4),X0(4),X0(4),X0(4);
```

```
            X0(2),X0(2),X0(5),X0(5),X0(2),X0(2),X0(5),X0(5);
            X0(3),X0(6),X0(3),X0(6),X0(3),X0(6),X0(3),X0(6)]';
        P2 = double(P2);
        K2 = convhulln(P2); % Calculate the convex hull
        patch('Vertice',P2,'Faces',K2,'FaceColor','r') % Building cuboid
        volume = prod(double(X0(row_s+1:2*row_s)-X0(1:row_s)));
    else
        disp("High-dimensional data cannot draw images")
    end
```

6.1 The Test Code

```
clear
clc
%Test using two-dimensional constraints
A=[-1 1;1 1;0 -1;0 1];
b=[3;3;0;2];
%Test using three-dimensional constraints
% A=[0 0 -1;-1 0 1;1 0 1;0 -1 1;0 1 1];
% b=[0;3;3;3;3];
[L,U,V]=maxvolrectangle(A,b);
```