WRIGHT STATE UNIVERSITY
DEPARTMENT OF MECHANICAL AND MATERIALS ENGINEERING

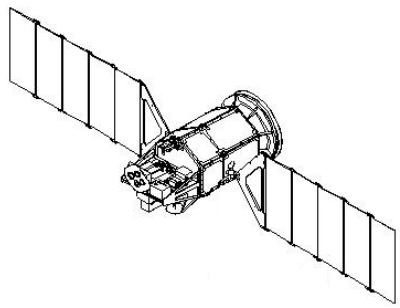
Design Optimization of 10-bar Truss Problem

Koorosh Gobal

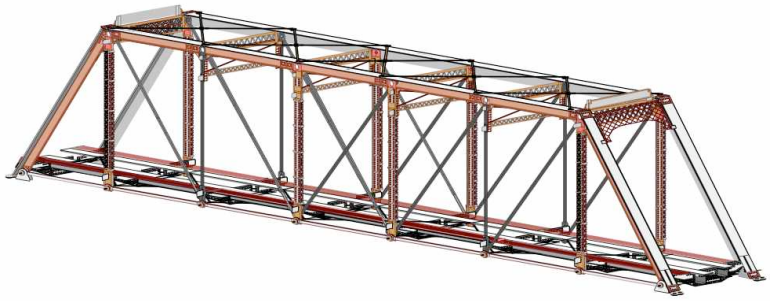
December 5, 2013

1 SYNOPSES

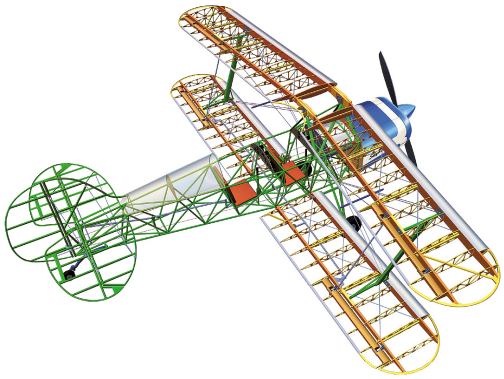
TRUSS is a structure comprising one or more triangular units constructed with straight members whose ends are connected at joints referred to as nodes. The truss structure is used in many design applications such as bridges, airplane wing boxes, and satellite's solar panels as shown in Figure 1.1. In many application, especially in aerospace, it is needed to design these truss structures such that they have mininum possible weights while they can carry the loads.



(a) Satelite's solar panel structure



(b) Bridge



(c) Airplane wing-box

Figure 1.1: Applications of truss structure.

The standard 10-bar truss consists of 10-hinged truss elements connected as shown in Figure 1.2. Nodes 5 and 6 are constrained in the plane of the truss. The truss is subjected to two vertical loads

applied at nodes 2 and 4. In this project the stress in truss members is calculated using finite element analysis. To do this an inhouse finite element solver is written in MATLAB. The code and its subroutines is included in the appendix. The structure needs to carry the loads without failing, therefore the stresses should not exceed the yield stress limit of the material.

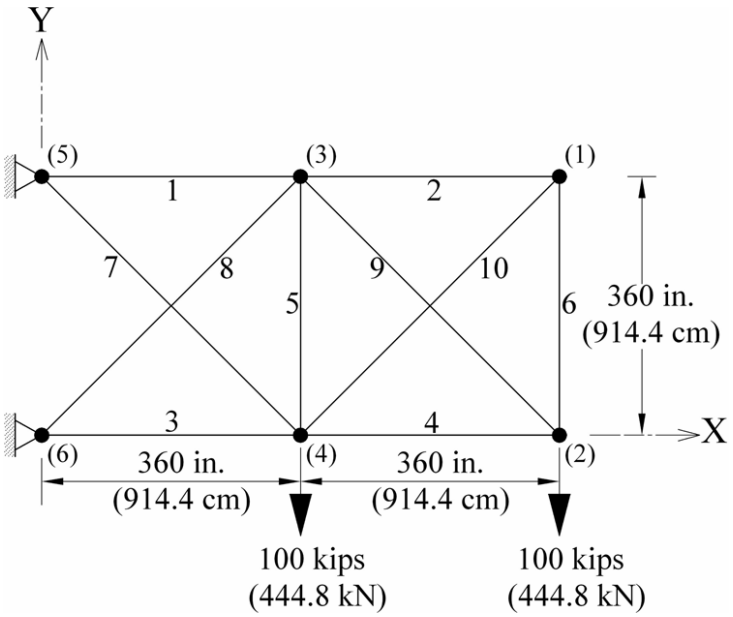


Figure 1.2: 10-bar truss problem.

2 Design Goals

The goal of this problem is to minimize the mass of the truss (M), varying only the cross sections of the truss members while the stress in members should not exceed the yield stress of the material.

The cost function is defined as the mass of the structure. The mass is calculated by multiplying the area of each element by its length and add it for all the elements. The constraints are selected as the allowable stress of the material, meaning the the stress value cannot exceed the allowable stress. This introduce *ten* constraints on the problem. These constraints are *nonlinear* with respect to cross-sectional area, simply because the definition of the stress, i.e. stress is equal to force divided by area. In this problem it is assumed that the allowable stress in both tensile and compression is the same. It is also needed to introduce bounds on the design variables, i.e. cross-sectional areas, since these cannot grow without bounds and also they cannot be smaller than zero.

3 Problem Formulation

As mentioned in section 2, the constraints are selected as the allowable stress in each element. In this project finite element analysis is used to calculate the stresses. In the following lines a brief review is given on the finite element formulation of the problem.

The first step when using the finite element method is to identify the individual elements which make up the structure. Once the elements are identified, the structure is disconnected at the nodes, the points which connect the different elements together. Each element is then analyzed individually to develop element stiffness equations.

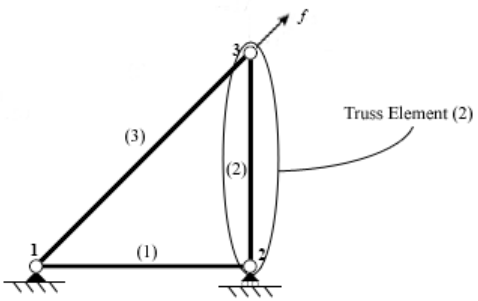


Figure 3.1: Individual element.

The forces and displacements are related through the element stiffness matrix which depends on the geometry and properties of the element as shown in Equation (3.1) where $[K]$ is the stiffness, $[U]$ is the displacement and $[F]$ is force matrix.

$$[K][U] = [F] \tag{3.1}$$

In this problem, truss elements are used to discretize the domain. Truss elements can only take axial loading and have two degrees of freedom at each end. A simple truss element is shown in Figure 3.2. In general case, the element makes an angle of "θ" with the x axis.

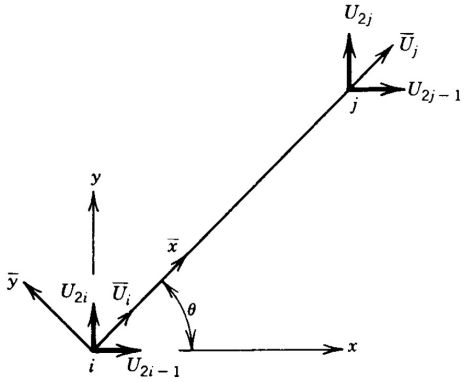


Figure 3.2: Truss element.

In order to solve Equation (3.1), the first step is do generate the stiffness matrix for truss element. For the general case of 2D element having angle θ with x axis the stiffness matrix can be written as

$$K = \frac{EA}{L}\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & cos(\theta) & -sin(\theta) \\ 0 & 0 & sin(\theta) & cos(\theta) \end{bmatrix}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} cos(\theta) & sin(\theta) & 0 & 0 \\ -sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & cos(\theta) & sin(\theta) \\ 0 & 0 & -sin(\theta) & cos(\theta) \end{bmatrix} \tag{3.2}$$

The stiffness matrix defined in Equation (3.2) is for a single element. This can be written for all the elements. The difference between these stiffness matrices are angle θ and length of the element, L. Once the individual element stiffness relations have been developed they must be assembled into the original structure. When merging these matrices together there are two rules that must be followed: compatibility of displacements and force equilibrium at each node.

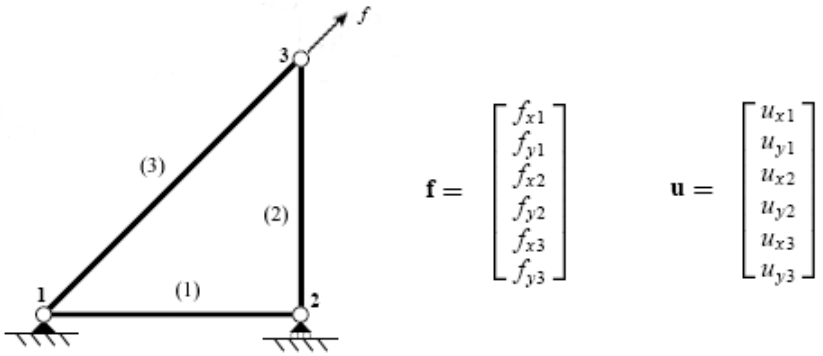


Figure 3.3: Global force and displacement.

The global displacement and force vectors each contain one entry for each degree of freedom in the structure. The element stiffness matrices are assembled together by expanding each matrix in conformation to the global displacement and load vectors. For example for element (1) in Figure 3.3 the stiffness matrix can be transferred from elemental to global as follows

$$k^1 = \frac{AE}{L}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow K^1 = \frac{AE}{L}\begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.3}$$

Finally, the global stiffness matrix is constructed by adding the individual global element stiffness matrices together.

In this problem the forces are applied on nodes 2 and 4 in the negative y direction. The force matrix is needed to be written in the global form. The boundary conditions are applied as zero displacement in both x and y directions for nodes 5 and 6.

After $[K]$ and $[F]$ matrices are known, Equation (3.1) can be used to calculate the nodal displacements of the structure. Later the stresses are calculated from the displacement fields.

4 Problem Statement

The general form of this optimization problem can be written as

$$\begin{aligned} &\text{Minimize } M(\bar{A}) \\ &\text{Subject to} \\ &\qquad |\sigma_i| \leq \sigma_{allowalbe} \qquad i = 1 \; to \; 10 \\ &\qquad 0.1 \leq \bar{A}_i \leq 50 \qquad\quad i = 1 \; to \; 10 \end{aligned} \tag{4.1}$$

In order to write the optimization problem in the standard form, all constraints need to be written in "≤" format. It should be noted that the lower limit value for cross-sectional area is selected as "0.1". This is because of FEA formulation. As shown in Equation (3.2) the cross-sectional area appears in the denominator, therefore, it cannot be equal to zero. The zero value of area causes singularities inside the finite element formulation and leads to numerical errors in inverting the stiffness matrix.

Moreover, since the magnitude of stress is larger compared to cross-sectional area by four order of magnitude, the constraints are normalized. This helps the optimization algorithm since all the constraints

will become in same order of magnitude. After all these modifications, the standard form of optimization problem can be written as

$$\begin{aligned}
&\text{Minimize } M(\bar{A})\\
&\text{Subject to}\\
&\frac{\sigma_i}{\sigma_{allowalbe}} - 1 \le 0 \qquad i = 1 \ to \ 10\\
&-\frac{\sigma_i}{\sigma_{allowalbe}} - 1 \le 0 \qquad i = 1 \ to \ 10\\
&1 - \frac{A_i}{0.1} \le 0 \qquad i = 1 \ to \ 10\\
&\frac{A_i}{50} - 1 \le 0 \qquad i = 1 \ to \ 10
\end{aligned} \tag{4.2}$$

Where σ_i is the stress in element i and A_i is the cross-sectional area of element i. In above formulation "M" represents the mass of the structure and can be calculate as follows

$$M(\bar{A}) = \sum_{1}^{10} \rho A_i L_i \tag{4.3}$$

5 Solution Approach

As mentioned in section 4, the optimization problem at hand is a constrained problem. The constraints are in nonlinear form therefore linear algorithems for constrained problems such as *SIMPLEX* cannot be used. For this problem the classical optimization methods cannot be utilized explicitly since the close form equations for the constraints does not exist. As a results surrogate models based on numerical gradients of constraints is utilized to generate sudo-closed form equations for calculating the constraints. In this project the optimization problem is solved using two different approaches:

1. Solution based on `fmincon` function in MATLAB.
2. Solution based on exterior penalty method.

The solution of `fmincon` function is used to validate the results from the penalty method approach.

5.1 Solution based on `fmincon` function

The `fmincon` function is used to find the minimum of a problem specified as

$$\begin{aligned}
&\text{Minimize } f(x)\\
&\text{Subject to}\\
&\qquad g(x) \le 0\\
&\qquad h(x) = 0\\
&\qquad lb \le x \le ub
\end{aligned} \tag{5.1}$$

$f(x)$, $g(x)$ and $h(x)$ returns vectors and can be nonlinear functions. `fmincon` attempts to find a constrained minimum of a scalar function of several variables starting with an initial estimate.

As shown in section 4, the problem has be written in standard form. Therefore `fmincon` can be used to solve the optimization problem. The side bounds also applied to the problem as mentioned in section 4. `fmincon` uses one of four algorithms: `active-set`, `interior-point`, `sqp`, or `trust-region-reflective` to solve the problem. For this project the *Sequential quadratic programming (SQP)* method has been utilized. The maximum number of allowable iteration for this problem is defined $1E5$ and the allowable tolerance on constraint violation is selected as $1E-5$.

Using the `fmincon` function the optimized solution of the problem is found. This result can be used to validate the penalty method that is used in the later section. The initial and final set of design variables is shown in Table 5.1. It should be noted that the weight of the structure is reduced from "2098.2" to "1593.2".

Table 5.1: fmincon results.

element number	Initial area	optimized area	initial stress	optimized stress
1	5.0	7.94	39073	25000
2	5.0	0.1	8024.9	15533
3	5.0	8.06	-40927	-25000
4	5.0	3.94	-11975	-25000
5	5.0	0.1	7097.9	0
6	5.0	0.1	8024.9	15533
7	5.0	5.74	29595	25000
8	5.0	5.57	-26973	-25000
9	5.0	5.57	16935	25000
10	5.0	0.1	-11349	-21967

5.2 Solution based on exterior penalty method

Penalty methods are a certain class of algorithms for solving constrained optimization problems. A penalty method replaces a constrained optimization problem by a series of unconstrained problems whose solutions ideally converge to the solution of the original constrained problem. The unconstrained problems are formed by adding a term, called a penalty function, to the objective function that consists of a penalty parameter multiplied by a measure of violation of the constraints. The measure of violation is nonzero when the constraints are violated and is zero in the region where constraints are not violated. For example, the following constrained problem can be converted to a series of unconstrained problems.

$$\min f(x) \quad \text{subject to} \quad g_i(x) \leq 0 \tag{5.2}$$

This problem can be converted to

$$\min \Phi_k(x) = f(x) + r_k \sum \Omega(g_i(x)) \quad , \quad \Omega(g_i(x)) = \max(0, g_i(x))^2 \tag{5.3}$$

In above equation, $\Omega(g_i(x))$ is the *penalty function* while r_k are the *penalty coefficients*. In each iteration k of the method, we increase the penalty coefficient r_k, solve the unconstrained problem and use the solution as the initial guess for the next iteration. Solutions of the successive unconstrained problems will eventually converge to the solution of the original constrained problem.

In this project *step function* is used instead of $\Omega(x)$ function as defined in Equation (5.3). The constraint is violated and added to cost function as penalty if its value is grater than zero. For step function it is known that

$$u(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{5.4}$$

Therefore the penalty function can be written as follows. The value of penalty function is therefore equal to $c_i^2(x)$ when the constraint is violated and zero when it is satisfied.

$$\text{penalty function: } u(g_i(x)) \cdot g_i^2(x) \tag{5.5}$$

Doing so the constraint optimization is transfered to an unconstrained optimization that that be solved

using unconstrained optimization methods. The final form of optimization problem is written as

$$\begin{aligned}\min \Phi_k(x) = W(\bar{A}) + r_k \sum & u\left(\frac{\sigma_i}{\sigma_{allowable}} - 1\right) \cdot \left(\frac{\sigma_i}{\sigma_{allowable}} - 1\right)^2 \\ & + u\left(-\frac{\sigma_i}{\sigma_{allowable}} - 1\right) \cdot \left(-\frac{\sigma_i}{\sigma_{allowable}} - 1\right)^2 \\ & + u\left(1 - \frac{A_i}{0.1}\right) \cdot \left(1 - \frac{A_i}{0.1}\right)^2 \\ & + u\left(\frac{A_i}{50} - 1\right) \cdot \left(\frac{A_i}{50} - 1\right)^2\end{aligned} \tag{5.6}$$

The initial value of r_k is selected as 10 and it is multiplied by 10 in every iteration so that the solution of unconstrained problem will converge to the constrained one.

The unconstrained problem is separated to two different parts, direction finding and step size calculation. The direction can be found by calculating the gradient of constraints at the required point. The step size can be calculated using methods such as *equal interval search* or *golden section*. The constraints does not have a closed form solution, therefore the value of gradient is calculated using numerical methods. In this project the *finite different* is used to calculate the value of the gradient as shown in Equation (5.7).

$$\frac{\partial \Phi}{\partial A_i} = \frac{\Phi(A_i + \delta A_i) - \Phi(A_i)}{\delta A_i} \tag{5.7}$$

To calculate the value of $\frac{\partial \Phi}{\partial A_i}$ only area A_i is purturbed and all the other areas are fixed. The area is perturbed by 0.01 in all the optimization steps. As can be seen in order to calculate the gradient of Φ the finite element solver needs to be called 10 times at each point. This cause problems with the computational cost of finite element solver in considerable.

The gradient data is used to calculate the direction at which the function will decrease. After the direction is calculated the problem is changed to on variable line search problem. Since the finite difference is used to calculate the gradients, the gradient information is valid only near the point where it is calculated. Therefore it is decided to chose a fixed step size to be near the region were gradients are valid instead of solving the *one-dimensional search problem*. The results for exterior penalty method is shown in table 5.2. The weight of the structure is reduced from "2098.2" to "1593.2"in this method.

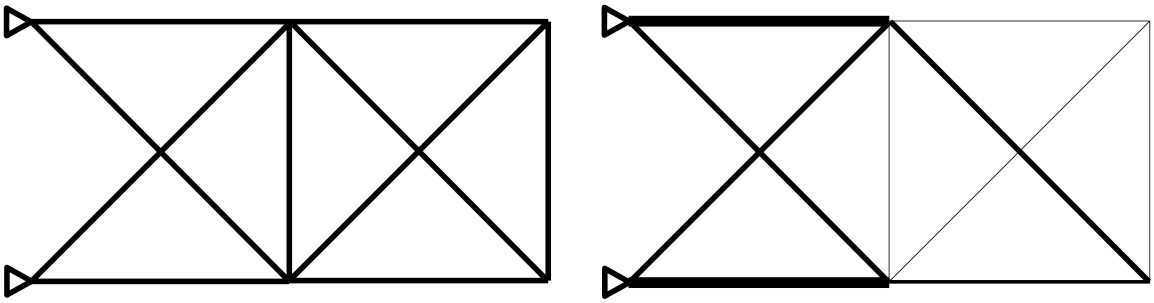
Table 5.2: Exterior penalty method results.

element number	Initial area	optimized area	initial stress	optimized stress
1	5.0	7.94	39073	25000
2	5.0	0.1	8024.9	15541
3	5.0	8.06	-40927	-25000
4	5.0	3.94	-11975	-25000
5	5.0	0.1	7097.9	-1.614
6	5.0	0.1	8024.9	15549
7	5.0	5.74	29595	24999
8	5.0	5.57	-26973	-25000
9	5.0	5.57	16935	25000
10	5.0	0.1	-11349	-21956

6 RESULTS AND SUMMARY

As shown in tables 5.1 and 5.2 the results of both methods agrees with each other. It is also interesting to look at the stress results of the optimized structure.

As shown in Table 5.2 the stresses for element 1, 3, 4, 7, 8, 9 are at the bound of allowable stress constraint. This means that all these constraints are active and the optimization problem is well defined. Looking at the cross-sectional area data of the optimized structure, it is seen that the cross-sectional areas of elements 2, 5, 6 and 10 are at their lower bounds. This can be interpreted as these elements are not contributing and can be removed from the structure. This is shown in Figure 6.1.



(a) Structure before optimization

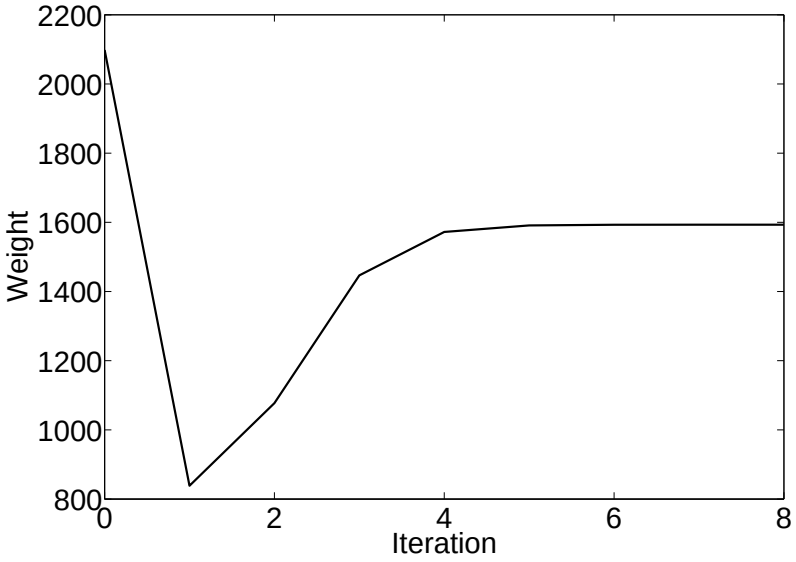
(b) Structure after optimization

Figure 6.1: Optimization results.

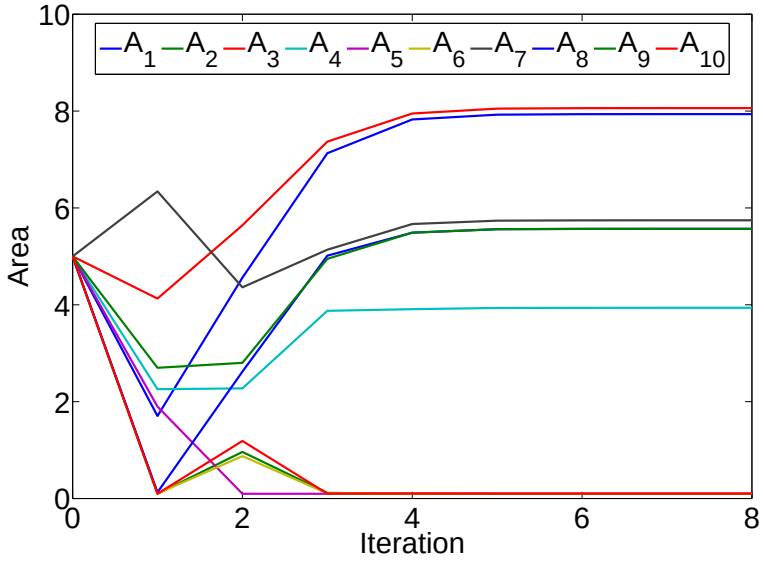
When using exterior penalty method to optimize the problem, the problem is transferred to a unconstrained problem. This removes some complexities involved in solving the optimization problem but can also introduce convergence issues. Also when using the penalty function method the initial value of penalty coefficient, r, and how it increase should be carefully selected. Otherwise the optimizer might not be able to converge to the optimized results. If to much penalty is introduced by selecting high r value at the initial stage of optimization, the solution might not converge to the actual solution. Therefore it is always better to start with low values for r and then gradually increase it.

Another issue when using exterior penalty method is that the solution might remain in the infeasible region if the optimizer reach its maximum number of function evaluation. In the exterior penalty method, the optimizer tries to reach the optimum feasible solution by moving in the infeasible reagion. Therefore, if maximum number of iterations is reached, the optimizer will return an infeasible answer. To handle this issue other methods such as *interior penalty methods* can be used. In that case, the solution always remain in the feasible region.

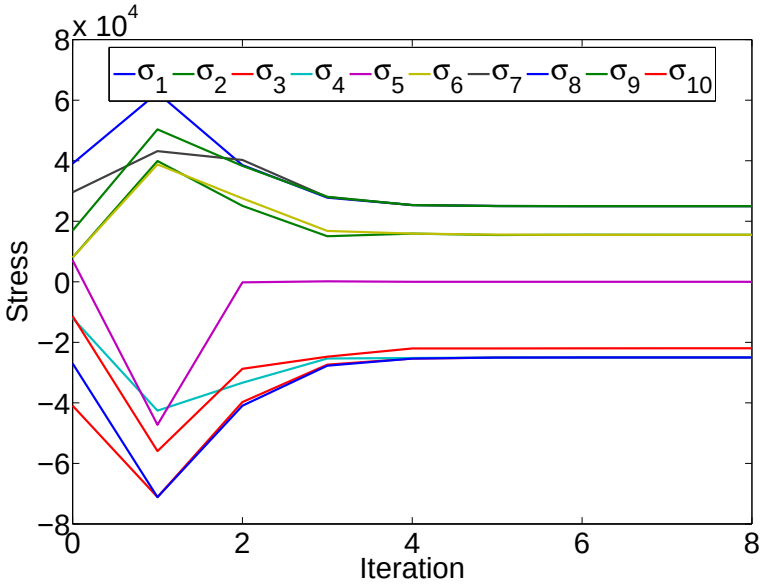
In this project the stopping criteria is introduced on the constraint. Hence, if the stresses in members is not changing more that $1E-3$ in any two subsequent steps, then the convergence criteria is satisfied. The plots for convergence history of stress in elements and cross-sectional areas is shown in Figure 6.2. The x axis represent the *penalty coefficient* in every step which can be considered as iteration step.



(a) Cost function iteration history



(b) Cross-sectional area iteration history



(c) Stress iteration history

Figure 6.2: Optimization convergence history.

7 APPENDIX

7.1 FINITE ELEMENT SOLVER

7.1.1 FEA SOLVER

```
function [output] = FEA(area,output_request)
% DEFINIG NODES AND ELEMENTS
node = [720 360;720 0;360 360;360 0;0 360;0 0];
element = [5 3;3 1;6 4;4 2;3 4;1 2;5 4;6 3;3 2;4 1];

% MECHANICAL PROPERTIES OF ELEMENTS
youngMS = 1E+7;
DOF = 2;
alpha = 7.3;
deltaT = 0;

% DEFINE NODAL BOUNDARY CONDITIONS
% DISPALCEMENT(deltaU) = [NODE_NUMBER X_DISPLACEMENT Y_DISPLACEMENT]
% FORCE = [NODE_NUMBER F_X F_Y]
deltaU = [6 0 0;5 0 0];
force = [2 0 -100000;4 0 -100000];

numNode = size(node,1);
numEl = size(element,1);
youngM = youngMS .* ones(numEl,1);

% CALCULATING LENGTH OF EACH ELEMENT AND ALSO ITS ANGLE TO HORIZION
elLength = elementLength(node,element,numNode,numEl);
elAngle = elementAngle(node,element,numNode,numEl);

% CALCULATE GLOBAL STIFFNESS MATRIX
gsMat = GSM(node,element,numNode,numEl,youngM,area,elLength,elAngle,DOF,deltaU);

% CALCULATE THE THERMAL FORCE IN GLOBAL FORM
F_thermal = zeros(numNode * DOF,1);
for i=1:numEl
    phi = elAngle(i);
    l2gMat = zeros(numNode * DOF,numNode * DOF);
    nodeStart = element(i,1);
    nodeEnd = element(i,2);

    f_thermal = [-youngMS*alpha*deltaT*area(i) 0 youngMS*alpha*deltaT*area(i) 0]';
    rMat = [cosd(phi) -sind(phi) 0 0;...
    sind(phi) cosd(phi) 0 0;...
    0 0 cosd(phi) -sind(phi);...
    0 0 sind(phi) cosd(phi)];
    f_thermal = rMat * f_thermal;

    l2gMat(2 * nodeStart - 1,2 * nodeStart - 1) = 1;
    l2gMat(2 * nodeStart,2 * nodeStart) = 1;
    l2gMat(2 * nodeEnd - 1,2 * nodeEnd - 1) = 1;
    l2gMat(2 * nodeEnd,2 * nodeEnd) = 1;
    l2gMat( :, ~any(l2gMat,1) ) = [];  %columns
```

```
    f_thermal = l2gMat * f_thermal;
    F_thermal = F_thermal + f_thermal;
end

% PUT THE FORCE MATRIX IN GLOBAL FORM
F = [];
for i=1:size(force,1)
    F = [F;force(i,2:3)'];
end
tMat = zeros(size(gsMat,1),size(F,1));
for i=1:size(force,1)
    tMat(force(i,1)*2-1,2*i-1) = 1;
    tMat(force(i,1)*2,2*i) = 1;
end
F = tMat*F;
F = F + F_thermal(1:length(F),1);

% STRUCTURAL SOLVER
% [nodeDef nodeR] = feaSolve(gsMat,deltaU,force);
nodeDef = inv(gsMat) * F;
U = nodeDef;
nodeDef = [nodeDef;zeros(4,1)];

% CALCULATE ELEMENT STRESSES
elStress = feaStress(node,element,numEl,nodeDef,elLength,youngM,area,elAngle);

if (strcmp(output_request,'stress'))
    output = elStress;
elseif (strcmp(output_request,'displacement'))
        output = nodeDef;
end
```

7.1.2 CALCULATE ELEMENTS LENGTH

```
function [el_Length] = elementLength (node,element,numNode,numEl)
el_Length = zeros(numEl,1);

for i=1:numEl
    pointStart = element(i,1);
    pointEnd = element(i,2);
    x1 = node(pointStart,1);
    y1 = node(pointStart,2);
    x2 = node(pointEnd,1);
    y2 = node(pointEnd,2);
    el_Length(i,1) = sqrt(power(x2-x1,2) + power(y2-y1,2));
end
```

7.1.3 CALCULATE ELEMENTS ANGLE

```
function [el_Angle] = elementAngle (node,element,numNode,numEl)
el_Angle = zeros(numEl,1);

for i=1:numEl
    pointStart = element(i,1);
    pointEnd = element(i,2);
```

```
    x1 = node(pointStart,1);
    y1 = node(pointStart,2);
    x2 = node(pointEnd,1);
    y2 = node(pointEnd,2);
    el_Angle(i,1) = atand((y2-y1)/(x2-x1));
end
```

7.1.4 Assemble global stiffness matrix

```
function globalStiffnessMatrix = GSM (node,element,numNode,numEl,...
youngM,area,elLength,elAngle,DOF,deltaU)
globalStiffnessMatrix = zeros(numNode * DOF,numNode * DOF);
elStifness = zeros(numEl,1);

for i=1:numEl
    elStifness(i) = youngM(i) * area(i) / elLength(i);
end

for i=1:numEl
    phi = elAngle(i);
    l2gMat = zeros(numNode * DOF,numNode * DOF);
    nodeStart = element(i,1);
    nodeEnd = element(i,2);

    rMat = [cosd(phi) sind(phi) 0 0;...
    -sind(phi) cosd(phi) 0 0;...
    0 0 cosd(phi) sind(phi);...
    0 0 -sind(phi) cosd(phi)];
    elsMat = rMat' * (elStifness(i) * [1 0 -1 0;0 0 0 0;-1 0 1 0;0 0 0 0]) * rMat;

    l2gMat(2 * nodeStart - 1,2 * nodeStart - 1) = 1;
    l2gMat(2 * nodeStart,2 * nodeStart) = 1;
    l2gMat(2 * nodeEnd - 1,2 * nodeEnd - 1) = 1;
    l2gMat(2 * nodeEnd,2 * nodeEnd) = 1;

    l2gMat( :, ~any(l2gMat,1) ) = [];  %columns

    elsMat = l2gMat * elsMat * l2gMat';
    globalStiffnessMatrix = globalStiffnessMatrix + elsMat;
end

tMat = eye(size(globalStiffnessMatrix,1));
for i=1:size(deltaU,1)
    tMat(deltaU(i,1)*2-1,deltaU(i,1)*2-1) = 0;
    tMat(deltaU(i,1)*2,deltaU(i,1)*2) = 0;
end
tMat( :, ~any(tMat,1) ) = [];
globalStiffnessMatrix = tMat'*globalStiffnessMatrix*tMat;
```

7.1.5 Calculate stress from displacement data

```
function [elStress] = feaStress(node,element,numEl,nodeDef,elLength,youngM,area,elAngle)
stress = zeros(numEl,1);

sigma = [];
```

```
for i=1:numEl
    startNode = element(i,1);
    endNode = element(i,2);
    sigma = [sigma;...
        youngM(i) / elLength(i) * ...
        [cosd(elAngle(i)) sind(elAngle(i)) -cosd(elAngle(i)) -sind(elAngle(i))] * ...
        [nodeDef(2*startNode-1);...
        nodeDef(2*startNode);...
        nodeDef(2*endNode-1,1);...
        nodeDef(2*endNode)]];
end
elStress = -sigma;
```

7.2 Cost and constraint function definition

7.2.1 Cost function

```
function f = costFunc(x)
densityS = 0.1;
L = [360.0000;360.0000;360.0000;360.0000;360.0000;360.0000;...
509.1169;509.1169;509.1169;509.1169];
f = densityS * (x(1)*L(1) + x(2)*L(2) + x(3)*L(3) + x(4)*L(4) + x(5)*L(5) + ...
    x(6)*L(6) + x(7)*L(7) + x(8)*L(8) + x(9)*L(9) + x(10)*L(10));
```

7.2.2 Constrains

```
function [c ce] = CSTR(area)

displacement = FEA(area,'displacement');
stress = FEA(area,'stress');

sigma_allowable = 25000;
c = [stress./sigma_allowable-1;-stress./sigma_allowable-1];
ce = [];
```

7.3 Optimization solvers

7.3.1 Optimization based on `fmincon`

```
clc;
clear;
format short g;
% ------------------------------------------------------------------------- %
area = 5 .* ones(10,1);

A = [];
b = [];
lb = 0.1 * ones(10,1);
ub = 50 * ones(10,1);

options=optimset('Algorithm','sqp','MaxIter',1E5,'MaxFunEvals',1E5,'TolCon',1E-5);
x0 = area;
[x fval] = fmincon(@costFunc,x0,A,b,[],[],lb,ub,@CSTR,options)
oldCostFunctionValue = costFunc(area);
[area x]
[FEA(area,'stress') FEA(x,'stress')]
```

```
[FEA(area,'displacement') FEA(x,'displacement')]
[costFunc(area) costFunc(x)]
[CSTR(area) CSTR(x)]
```

7.3.2 Optimization based on penalty methods - based on `fminsearch`

```
clc;
% clear;
format short g;
close all;
% ---------------------------------------------------------------------- %
area = 5 .* ones(10,1);
comparison_of_stresses = [FEA(area,'stress')];
comparison_of_area = [area];
comparison_of_cost_function = [costFunc(area)];
R = [0]
for i=1:100
    r = 1*10^i
    phi = @(area) costFunc(area)...
     + sum(r * (u(CSTR(area)).*CSTR(area).^2))...
     + sum(r * (u(bound(area)).*bound(area).^2));
    x = fminsearch(phi, area);
    R = [R r]
    comparison_of_stresses = [comparison_of_stresses FEA(x,'stress')]
    comparison_of_area = [comparison_of_area x]
    comparison_of_cost_function = [comparison_of_cost_function costFunc(x)]
    if max(abs((x - area)./area)) < 1E-3
        area = x;
        break;
    end
    area = x;
end
clc
r
optimized_area = area
optimized_stress = FEA(area,'stress')
optimized_weight = costFunc(area)

font = 40;
RR = 0:1:8;
figure(1)
plot(RR,comparison_of_cost_function,'LineWidth',3,'color','k')
set(gca,'FontSize',font)
xlabel('Iteration')
ylabel('Weight')
figure(2)
plot(RR,comparison_of_area,'LineWidth',3)
set(gca,'FontSize',font)
xlabel('Iteration')
ylabel('Area')
legend('A_1','A_2','A_3','A_4','A_5','A_6','A_7','A_8','A_9','A_{10}');
figure(3)
plot(RR,comparison_of_stresses,'LineWidth',3)
set(gca,'FontSize',font)
```

```
xlabel('Iteration')
ylabel('Stress')
legend('\sigma_1','\sigma_2','\sigma_3',...
'\sigma_4','\sigma_5','\sigma_6',...
'\sigma_7','\sigma_8','\sigma_9','\sigma_{10}')
```

7.3.3 Optimization based on penalty methods

```
clc;
clear;
format short g;
% ------------------------------------------------------------------------ %
area = 5 .* ones(10,1);
area_step = 1E-3;
AREA = area;


for ri=0:100
    r = 10^ri;
    phi = @(area) costFunc(area) + ...
    sum(r * (u(CSTR(area)).*CSTR(area).^2)) + ...
    sum(r * (u(bound(area)).*bound(area).^2));
    gradient_phi = zeros(10,1);
    for i=1:10
        selection_mat = zeros(10,1);
        selection_mat(i) = 1;
        dArea = area_step * selection_mat .* ones(10,1);
        dArea_scalar = dArea(i);
        gradient_phi(i) = (phi(area + dArea) - phi(area)) / dArea_scalar;
    end
    d = -gradient_phi
    alpha = 1E-3;
    area_new = area + alpha * d;
    [CSTR(area) CSTR(area_new)];
    if max(abs(CSTR(area)-CSTR(area_new))) < 1E-3
        area = area_new;
        break;
    end
    area = area_new;
end
```