REFERENCES

Ali, N., Behdinan, K., Fawaz, Z. (2003). Applicability and viability of a GA based finite element analysis architecture for structural design optimization. Journal of Computer & Structures, 81: 2259–2271.

Bathe, J., (1996). Finite element procedures. New York : Prentice Hall.

Begg, D.W., Liu, X. (2000). On simultaneous optimization of smart structures - Part II: Algorithms and examples. Journal of Computer Methods Application Mechanics Engineering, 184: 25-37.

Chan, S.L. (2001). Review in non-linear behavior and design of steel structures. Journal of Constructional Steel Research, 57: 1217–1231.

Chen, S.Y. (2001). An approach for impact structure optimization using the robust genetic algorithm. Journal of Finite Elements in Analysis and Design, 37: 431-446.

Chen, S.Y., Rajan, S. D. (2000). Using Genetic Algorithm as An Automatic Structural Design Tool. Arizona State University.

Chen, W.F. (2000). Structural stability: from theory to practice. Journal of Engineering Structures, 22: 116–122.

Chen, W.F. , Lui, E.M. (1991). Stability Design of Steel Frames. Boca Raton : CRC Press.

Chen, W.F. , Sohal, I. (1995). Plastic Design and Second-Order Analysis of Steel Frames. New York : Springer-Verlag.

Choi, S.H., Kim, S.E. (2002). Optimal design of steel frame using practical nonlinear inelastic analysis. Journal of Engineering Structures, 24(9): 1189-1201.

Cook, R.D., Malkus, D.S., Plesha, M.E., Witt, R.J. (2002). Concept and Applications of Finite Element Analysis. New York : John Wiley & Sons.

Erbatur, F., Hasancebi, O., Tutuncu, I., Kihc, H. (2000). Optimal design of planar and space structures with genetic algorithms. Journal of Computer & Structures, 75: 209-224.

Fafitis, A. (2002). Nonlinear truss Analysis by One Matrix Invertion. Arizona State University, 15th ASCE Engineering Mechanics Division Conference.

Frangopol, D.M., Klisinski, M. (1989). Material Behavior and Optimum Design of Structural Systems. Journal of Structural Engineering, 115(5): 1054-1075.

Goldberg, D.E., (1989). Genetic algorithms in Search, Optimization & Machine Learning. Boston : Addition-Wesley.

Goto, Y., Miyashita, S. (1998). Classification System for Rigid and Semirigid Connections. Journal of Structural Engineering, 124(7): 750-757.

Jirásek, M., Bazant, Z.P. (2002). Inelastic Analysis of Structures. England : John Wiley & Sons.

Kameshki, E. S., Saka, M. P. (2001). Optimum design of nonlinear steel frames with semi-rigid connections using a genetic algorithm. Journal of Computer & Structures, 79(17): 1593-1604.

Kameshki, E. S., Saka, M. P. (2003). Genetic algorithm based optimum design of nonlinear planar steel frames with various semi-rigid connections. Journal of Constructional Steel Research, 59(1): 109-134.

Kemp, A.R. (2000). Simplified amplification factors representing material and geometric inelasticity in frame instability. Journal of Engineering Structures, 22: 1609–1619.

Kennedy, D.J.L., Albert, C., MacCrimmon, R.A. (2000). Inelastic incremental analysis of an industrial Pratt truss. Journal of Engineering Structures, 22: 146–154.

Kida, Y., Kawamura, H., Tani, A., Takizawa, A. (2000). Multi-Objective Optimization of Spatial Truss Structures by Genetic Algorithm. Forma, 15: 133–139.

Kim, S.E., Chen, W.F. (1999). Design guide for steel frames using advanced analysis program. Journal of Engineering Structures, 21(4): 352-364 .

Kim, S.E., Choi, S.H. (2001). Practical advanced analysis for semi-rigid space frames. International Journal of Solids and Structures, 38(50-51): 9111-9131.

Kim, S.E., Choi, S.H.,Kim, C.S. (2004). Automatic design of space steel frame using practical nonlinear analysis. Journal of Thin-Walled Structures, : In Press.

Kim, S.E., Kim. Y., Choi, S.H. (2001). Nonlinear analysis of 3-D steel frames. Journal of Thin-Walled Structures, 39(6): 445-461.

Kim, S.E., Lee, D.H. (2002). Second-order distributed plasticity analysis of space steel frames. Journal of Engineering Structures, 24(6): 735-744.

Kim, S.E., Park, M.H., Choi, S.H. (2001). Direct design of three-dimensional frames using practical advanced analysis. Journal of Engineering Structures, 23(11): 1491-1502 .

Kim, S.E., Park, M.H., Choi, S.H. (2001). Practical advanced analysis and design of three-dimensional truss bridges. Journal of Constructional Steel Research, 57(8): 907-923 .

Kwon, Y.D., Kwon, S.B., Jin, S.B., Kim, J.Y. (2003). Convergence enhanced genetic algorithm with successive zooming method for solving continuous optimization problems. Journal of Computer & Structures, 81: 1715–1725.

Liew, J.Y.R., Chen, W.F., Chen, H. (2000). Advanced inelastic analysis of frame structures. Journal of Constructional Steel Research, 55: 245–265.

Limsamphancharoen, N. (2003). Application of Dynamic Neighborhood Method in Damage Identification. Thammasat University, : 1-10.

Nukala, P.K., White, D.W. (2004). A mixed finite element for three-dimensional nonlinear analysis of steel frames. Journal of Computer Methods Application Mechanics Engineering, 193: 2507–2545.

Ovunc, B. A., Ren, T. (1996). Nonlinearities in the analysis of frames. Journal of Computer & Structures, 61(6): 1177-1 184.

Pezeshk, S., Camp, C.V. (2003). State of the Art on the Use of Genetic Algorithms in Design of Steel Structures. The University of Memphis.

Pezeshk, S., Camp, C.V. Chen, D. (2000). Design of Nonlinear Framed Structures Using Genetic Optimization. Journal of Structural Engineering, 126(3): 382-388.

Sakata, S., Ashida, F., Zako, M. (2002). Topology and detail geometry optimization for beam structures using homotopy modeling. Journal of Computer Methods Application Mechanics Engineering, 191: 4279–4293.

Teh, L.H. (2001). Cubic beam elements in practical analysis and design of steel frames. Journal of Engineering Structures, 23: 1243–1255 .

Turkkan, N. (2003). Discrete Optimization Of Structures Using A Floating Point Genetic Algorithm. Annual Conference of the Canadian Society for Civil Engineering, GCM 134: 1-8.

Yang, Y.B., Kuo, S.R. (1994). Theory & Analysis of Nonlinear Framed Structures. Singapore : Prentice Hall.

APPENDIX

TenBarTruss.m

```
% Geometric and forces initial data of the truss
% The given problem
%---------------------------
%  1. Nodal coordinates
%---------------------------
nDOFs = 2;                          % Number of DOFs per node
nNode = 6;                          % Total number of nodes
sdof = nDOFs*nNode;                 % Total DOFs = nDOFs x No.of Nodes

Node = zeros(nNode,nDOFs);

%      X axis           Y axis
Node(1,1)=0.0;    Node(1,2)=0.0;      % coordinate of Node #1
Node(2,1)=0.0;    Node(2,2)=100.0;    % coordinate of Node #2
Node(3,1)=100.0;  Node(3,2)=0.0;      % coordinate of Node #3
Node(4,1)=100.0;  Node(4,2)=100.0;    % coordinate of Node #4
Node(5,1)=200.0;  Node(5,2)=0.0;      % coordinate of Node #5
Node(6,1)=200.0;  Node(6,2)=100.0;    % coordinate of Node #6

%-----------------------------
%  2. Nodal connectivity
%-----------------------------
nNODs = 2;                            % Number of nodes per element
nElem = 10;                           % Total number of elements
ndoe = nDOFs*nNODs;                   % Number of DOFs per element

Element = zeros(nElem,nNODs);

%starting node      ending node
Element(1,1)=1;   Element(1,2)=3;     % element #1  (node1,node3)
Element(2,1)=3;   Element(2,2)=5;     % element #2  (node3,node5)
Element(3,1)=5;   Element(3,2)=6;     % element #3  (node5,node6)
Element(4,1)=6;   Element(4,2)=4;     % element #4  (node6,node4)
Element(5,1)=4;   Element(5,2)=2;     % element #5  (node4,node4)
Element(6,1)=3;   Element(6,2)=4;     % element #6  (node3,node4)
Element(7,1)=4;   Element(7,2)=1;     % element #7  (node4,node1)
Element(8,1)=2;   Element(8,2)=3;     % element #8  (node2,node3)
Element(9,1)=3;   Element(9,2)=6;     % element #9  (node3,node6)
Element(10,1)=4;  Element(10,2)=5;    % element #10(node4,node5)

%------------------------------
%  3. Element's length calculation
%------------------------------

Length=zeros(nElem,1);  % length vector for all elements

for i=1:nElem           % loop for every element

nd(1)=Element(i,1);     % 1st connected node for the (iel)-th element
nd(2)=Element(i,2);     % 2nd connected node for the (iel)-th element

x1=Node(nd(1),1); y1=Node(nd(1),2);       % coordinate of 1st node
x2=Node(nd(2),1); y2=Node(nd(2),2);       % coordinate of 2nd node

Length(i)=sqrt((x2-x1)^2+(y2-y1)^2);      % the length of element(i)

end

%------------------------------
```

```
%  4. Applied restraints
%----------------------------

Bcdof(1)=1;       % 1st dof (horizontal displ) is constrained
Bcval(1)=0;       % whose described value is 0
Bcdof(2)=2;       % 2nd dof (vertical displ) is constrained
Bcval(2)=0;       % whose described value is 0
Bcdof(3)=3;       % 3rd dof (horizontal displ) is constrained
Bcval(3)=0;       % whose described value is 0
Bcdof(4)=4;       % 4th dof (horizontal displ) is constrained
Bcval(4)=0;       % whose described value is 0

%----------------------------
%  5. Applied nodal force
%----------------------------

Force=zeros(sdof,1);% system force vector
Force(6) = -2268.0; % 3rd node (6th dof) has 5 kips in downward
direction
Force(10)= -2268.0; % 5th node(10th dof) has 5 kips in downward
direction
                    % 1kips = 10^3pounds = 453.6 kgf
Dfactor = 1.2;      % Default value for self weight load
Lfactor = 1.6;      % Default value for static point loads
rho = 7.849e-3;     % Unit in kg/cm^3

%----------------------------
%  6. Miscellaneous
%----------------------------
Emodulus = 2038.9019e3; % elastic modulus (ksc)
YieldStr = 2531.0507;   % minimum yield stress (ksc)
maxdisp = 2.00;         % maximum displacement (cm)

%----------------------------
%  7. Section Indices
%----------------------------
%   The lists of available sections and its radius of gyration

%   The possible cross-sectional areas for each member

%   The smallest value of section
%   Equal angle, JIS G 3192 (cm^2)
Sections =
[1.427 1.727 2.336 3.755 3.492 4.302 3.892 4.802 5.644 4.692 ...
 5.802 6.367 7.527 9.761 8.127 8.727 12.69 16.56 9.327 10.55];

%   The radius of gyration (cm) appropriate to its sections
Gyration =
[0.747 0.908 1.230 1.200 1.360 1.360 1.530 1.520 1.500 1.850 ...
 1.840 1.990 1.980 1.940 2.140 2.300 2.250 2.220 2.460 2.770];

%     = size(Sections,1);    % Get the number of rows
nsect = size(Sections,2);    % Get the number of columns

coeficient = (sqrt(YieldStr/Emodulus))/3.14159;

lamdacoef  = zeros(1,nsect);
for i=1:nsect
    lambdacoef(i) = coeficient/Gyration(i);
end
```

```
%-----------------------------
%  8. Input Database
%-----------------------------
Data{1} = nDOFs;    % Number of DOFs per node
Data{2} = nNode;    % Total number of nodes
Data{3} = Node;     % The nodal coordinates vector
Data{4} = nNODs;    % Number of nodes per element
Data{5} = nElem;    % Total number of elements
Data{6} = Element;  % The nodal connectivity vector
Data{7} = sdof;     % Total DOFs = nDOFs x No.of Nodes
Data{8} = ndoe;     % No. of DOFs/element = nDOFs x nNODs
Data{9} = Length;   % The length vector for all elements
Data{10}= Bcdof;    % a vector containing dofs associated with BC
Data{11}= Bcval;    % a vector containing BC values associated with
Bcdof
Data{12}= Emodulus; % elastic modulus (psi)
Data{13}= Force;    % Applied nodal force
Data{14}= YieldStr; % minimum yield stress
Data{15}= nsect;    % The number of section values
Data{16}= Sections; % The lists of availabe cross-sectional areas
Data{17}=lambdacoef;% Prepared data for the calculation of slederness
ratio
Data{18}= maxdisp;  % maximum allowable displacement
Data{19}= Dfactor;  % Default value for self weight load
Data{20}= Lfactor;  % Default value for static point loads
Data{21}= rho;      % Unit in kg/cm^3
```

FEM_Nonlinear.m

```
%-----------------------------------------------------------------------%
% Nonlinear Analysis of Planar Steel Trusses
% Author: TRAN Tien-Dac
%       Graduate student, Department of Civil Engineering
%       Chulalongkorn University
% Last change: May 19, 2005
%
% Problem description
%   Find the deflection, stress and load ratio of the given truss
problem.
%-----------------------------------------------------------------------%

clear;

global Data;          % Declare the global variables
Data = cell(21,1);    % (1)nDOFs       Number of DOFs per node
                      % (2)nNode       Number of nodes
                      % (3)Node        The nodal coordinates vector
                      % (4)nNODs       Number of nodes per element
                      % (5)nElem       Number of elements
                      % (6)Element     The nodal connectivity vector
                      % (7)sdof        Total system DOFs = nDOFs x nNode
                      % (8)ndoe        No. of DOFs/element
                      % (9)Length      The element's length vector
                      %(10)Bcdof       The constrained-DOFs vector
                      %(11)Bcval       whose described value is 0
                      %(12)Emodulus    Elastic modulus
                      %(13)Force       The system force vector
                      %{14}YieldStr    The minimum yield stress
                      %(15)nsect       The number of section values
                      %{16}Sections    The lists of availabe sections
                      %{17}lamdacoef Prepared for the slenderness ratio
                      %{18}maxdisp     maximum allowable displacement
                      %(19)Dfactor     Default for self weight load
                      %(20)Lfactor     Default for live load loads
                      %(21)rho         Unit in kg/cm^3

TenBarTruss;                           % Input the initial data of the truss

Solution = 10*ones(nElem,1);           % Assume an abitrary section indices
Stop     = 0;                          % stopping criterion

[Score,Vol,Disp,Stress,history] = CMainFEM(Solution,Data,Stop);

%---------------------------
%  output phase
%---------------------------

BDisplayStress(Data,Disp,Stress);
BDisplayHistory(history);
```

AMain.m

```
%-------------------------------------------------------------------%
% Sizing Optimization of Nonlinear Planar Steel Trusses
% Using Genetic Algorithms
% Author: TRAN Tien-Dac
%       Graduate student, Department of Civil Engineering
%       Chulalongkorn University
% Last change: May 19, 2005

% Problem description
%   Find the optimization solution of the given truss problem.
%-------------------------------------------------------------------%

clear;

global Data;           % Declare the global variables
Data = cell(21,1);     % (1)nDOFs       Number of DOFs per node
                       % (2)nNode       Number of nodes
                       % (3)Node        The nodal coordinates vector
                       % (4)nNODs       Number of nodes per element
                       % (5)nElem       Number of elements
                       % (6)Element     The nodal connectivity vector
                       % (7)sdof        Total system DOFs = nDOFs x nNode
                       % (8)ndoe        No. of DOFs/element
                       % (9)Length      The element's length vector
                       %(10)Bcdof       The constrained-DOFs vector
                       %(11)Bcval       whose described value is 0
                       %(12)Emodulus    Elastic modulus
                       %(13)Force       The system force vector
                       %{14}YieldStr    The minimum yield stress
                       %(15)nsect       The number of section values
                       %{16}Sections    The lists of availabe sections
                       %{17}lamdacoef Prepared for the slenderness ratio
                       %{18}maxdisp     maximum allowable displacement
                       %(19)Dfactor     Default for self weight load
                       %(20)Lfactor     Default for live load loads
                       %(21)rho         Unit in kg/cm^3

TenBarTruss;                    % Input the initial data of the truss

%% Required functions

% The creation function will return a cell array of size
PopulationSize.
% GA_create.m

% The custom crossover function takes a cell array, the population,
% and returns a cell array, the children that result from the
% crossover.
% GA_crossover.m

% The custom mutation function takes an individual, and returns a
% mutated ordered set.
% GA_mutate.m

% The fitness function
% Bfitness.m

FitnessFcn = @(x) BFitness(x,Data);
```

```
%% Genetic Algorithm options setup

% Firstly, create an options structure to indicate a custom data type
% and the population range.
nsect=Data{15};
options = gaoptimset('PopulationType',
'custom','PopInitRange',[1;nsect]);

% Secondly, declare the custom creation, crossover, mutation, that
% we have created, as well as setting some stopping conditions.
options = gaoptimset(options,'CreationFcn',@GA_create, ...
                    'CrossoverFcn',@GA_crossover, ...
                    'MutationFcn',@GA_mutate, ...
                    'Generations',400,'PopulationSize',40, ...
                    'StallGenLimit',3000, ...
                    'StallTimeLimit',3000,'Vectorized','on', ...
                    'PlotFcns', @gaplotbestf,'PlotInterval',2);

% Finally, call the genetic algorithm with the problem information.
NumOfVar = Data{5};
[x,fval,reason,output] = ga(FitnessFcn,NumOfVar,options)

% Re-analysis the GA_optimize solution
solution = x{1};
[FScore,FVol,FDisp,FStress,FHistory] = CMainFEM(solution,Data,1)
BDisplayTruss(Data,solution);
BDisplayStress(Data,FDisp,FStress);
BDisplayHistory(FHistory);
```

GA_create.m

```
function pop = GA_create(NVARS,FitnessFcn,options)
% Creates a population of permutations
%------------------------------------------------------------
% pop = GA_create(NVARS,FitnessFcn,options)
%
% Variable Description:
%   pop = creates a population of permutations with a length of
NVARS.
%   NVARS = Number of variables
%   FitnessFcn = Fitness function
%   options = Options structure used by the GA
%------------------------------------------------------------

totalPopulationSize = sum(options.PopulationSize);
n = NVARS;
pop = cell(totalPopulationSize,1);  % Create an empty column vector
for i = 1:totalPopulationSize
    for j = 1:n
        k = randperm(20);            % nsect = 20
        pop{i}(j) = k(1);            % Fill a random order to cell i
of pop
    end
end
```

GA_crossover.m

```
function xoverKids  = GA_crossover(parents,options,NVARS, ...
    FitnessFcn,thisScore,thisPopulation)
% Custom crossover function
%------------------------------------------------------------
% xoverKids  = GA_crossover(parents,options,NVARS, ...
%
FitnessFcn,thisScore,thisPopulation)
%
% Variable Description:
%   xoverKids = crossovers PARENTS to produce the children XOVERKIDS.
%   parents = Parents chosen by the selection function
%   options = Options structure created from GAOPTIMSET
%   NVARS = Number of variables
%   FitnessFcn = Fitness function
%   thisScore = Vector of scores of the current population
%   thisPopulation = Matrix of individuals in the current population
%------------------------------------------------------------

nKids = length(parents)/2;
xoverKids = cell(nKids,1);
index = 1;

for i=1:nKids
    parent = thisPopulation{parents(index)};
    index = index + 2;

    % Flip a section of parent1.
    p1 = ceil((length(parent) -1) * rand);
    p2 = p1 + ceil((length(parent) - p1- 1) * rand);
    child = parent;
    child(p1:p2) = fliplr(child(p1:p2));
    xoverKids{i} = child;
end
```

GA_mutate.m

```
function mutationChildren = GA_mutate(parents,options,NVARS, ...
    FitnessFcn, state, thisScore,thisPopulation,mutationRate)
% Custom mutation function
%----------------------------------------------------------
% mutationChildren = GA_mutate(parents,options,NVARS, ...
%                 FitnessFcn, state,
thisScore,thisPopulation,mutationRate)
%
% Variable Description:
%   mutationChildren = mutate the parents to produce mutated children
%   options = Options structure created from GAOPTIMSET
%   NVARS = Number of variables
%   FitnessFcn = Fitness function
%   state = State structure used by the GA solver
%   thisScore = Vector of scores of the current population
%   thisPopulation = Matrix of individuals in the current population
%   mutationRate = Rate of mutation
%----------------------------------------------------------

% Here we swap two elements of the permutation
mutationChildren = cell(length(parents),1);
for i=1:length(parents)
    parent = thisPopulation{parents(i)};
    p = ceil(length(parent) * rand(1,2));
    child = parent;
    child(p(1)) = parent(p(2));
    child(p(2)) = parent(p(1));
    mutationChildren{i} = child;
end
```

BFitness.m

```
function scores = BFitness(x,Data)
% Calculate the fitness of a generation.
%------------------------------------------------------------
% scores = BFitness(x,Data)
%
% Variable Description:
% every solution x{j} is a row vector ( 1 x num.of.variables ).
% it means x(j) is a row vector of ( 1 x num.of.elements.in.the.truss
)
% each value in vector x(j) varies from 1 to nsect
(num.of.sect.available)
% this number is the order of its cross sectional area accordingly.
%------------------------------------------------------------

scores = zeros(size(x,1),1);    % create a column vector for a
generation
for j = 1:size(x,1)             % x = [population size x 1] of cells
    p = x{j};                   % consider the solution x{j}
    score = CMainFEM(p,Data,1);
    scores(j) = score;
end

% Export the results
[fid,msg]=fopen('peak.txt','a');
avr = mean(scores);
count = fprintf(fid,'%12.3f \n',avr);
status = fclose(fid);
```

BDisplayTruss.m

```
function outputstring = BDisplayTruss(Data,Solution)
% Display to screen the geometric figure of the truss
%------------------------------------------------------------
% BDisplayTruss(Data,Solution)
%
% Variable Description:
%   Data = the global variables
%   Solution = the set of section indices for all members
%------------------------------------------------------------
nNode = Data{2};     Node = Data{3};
nElem = Data{5};     Element = Data{6};
sdof = Data{7};      BC = Data{10};
Force = Data{13};    Sect = Data{16};

figure

%------------------------------
%  1. Plot the bars
%------------------------------
Area = Sect(Solution);

for e1=1:nElem
   node1=Node(Element(e1,1),:);
   node2=Node(Element(e1,2),:);
   shift=2*ones(1,2);
   midpoint = (node1+node2)/2;
   if node1(1)==node2(1) shift(2)=0;
   elseif node1(2)==node2(2) shift(1)=0;
   else
       midpoint = (2*node1+node2)/3;
       nodetmp = node1 - node2;
       if nodetmp(1)*nodetmp(2)>0 shift(1)=-shift(1); end
   end
   hold on
   plot([node1(1);node2(1)],[node1(2);node2(2)],'b','LineWidth',2);
        text(midpoint(1)+shift(1),midpoint(2)+shift(2), ...
        [num2str(Area(e1)),'cm2'], ...
        'Color','b','FontSize',12);
end

%------------------------------
%  2. Add node number
%------------------------------
for n1=1:nNode

text(Node(n1,1),Node(n1,2),int2str(n1),'Color','magenta','FontSize',1
6);
end

%------------------------------
%  3. Add boundary constraints
%------------------------------
nBC = size(BC,2);                  % Get the number of constrainted-DOFs
shift = -5;
for n1=1:nBC,
    nID = floor((BC(n1)+1)/2);     % Get the nodeID corresponding to the
DOF
    coord = Node(nID,:);           % Get the node's coordinates
    if mod(BC(n1),2)==1            % horizontal displacement constraint
```

```
        coord(1) = coord(1)+shift;
        sign = '>';
     else                          % vertical displacement constraint
        coord(2) = coord(2)+shift;
        sign = '^';
     end
     plot(coord(1),coord(2),sign,'MarkerEdgeColor','red',...
         'MarkerFaceColor','green','MarkerSize',15);
end

%-----------------------------
%  4. Add point loads
%-----------------------------
for n1=1:sdof,
    if Force(n1)~=0
        nID = floor((n1+1)/2);   % Get the nodeID corresponding to the
DOF
        coord = Node(nID,:);     % Get the node's coordinates
        if mod(n1,2)==1          % horizontal point load
            if Force(n1)>0
                value = '\rightarrow';
            else
                value = '\leftarrow';
            end
        else                     % vertical point load
            if Force(n1)>0
                value = '\uparrow';
            else
                value = '\downarrow';
            end
        end
        text(coord(1),coord(2),value,'Color','red','FontSize',30);
    end
end

%-----------------------------
%  5. Add titles
%-----------------------------
title(['The ',int2str(nElem),'-bar truss
problem'],'Color','black','FontSize',16)
axis equal
```

BDisplayStress.m

```
function outputstring = BDisplayStress(Data,Disp,Stress)
% Display the stress results (axial force) of the truss
%-------------------------------------------------------------
% BDisplayStress(Data,Disp,Stress)
%
% Variable Description:
%   Data = the global variables
%   Disp = the column vector of global displacements
%   Stress = the column vector of global stresses
%-------------------------------------------------------------

nNode = Data{2};     Node = Data{3};
nElem = Data{5};     Element = Data{6};      sdof = Data{7};
BC = Data{10};       Force = Data{13};       maxdisp = Data{18};

num=1:1:sdof;
displ=[num' Disp]              % print displacements

numm=1:1:nElem;
stresses=[numm' Stress]        % print stresses

figure
%------------------------------
%  1. Plot the bars
%------------------------------

for el=1:nElem
   node1=Node(Element(el,1),:);
   node2=Node(Element(el,2),:);
   shift=2*ones(1,2);
   midpoint = (node1+node2)/2;
   if node1(1)==node2(1) shift(2)=0;
   elseif node1(2)==node2(2) shift(1)=0;
   else
       midpoint = (2*node1+node2)/3;
       nodetmp = node1 - node2;
       if nodetmp(1)*nodetmp(2)>0 shift(1)=-shift(1); end
   end
   hold on
   plot([node1(1);node2(1)],[node1(2);node2(2)],'k', ...
                                  'LineWidth',1,'LineStyle','-.');
   text(midpoint(1)+shift(1),midpoint(2)+shift(2),int2str(el), ...
                                  'Color','k','FontSize',12);
end

%------------------------------
%  2. Update the deformed shape
%------------------------------
if max(Disp)<maxdisp
    ratio = 2.0;
else
    ratio = 1.0;
end

text(1,-90,['Displacement magnifier = ',
                                  num2str(ratio)],'FontSize',12);

for i=1:nNode
    Node(i,1)=Node(i,1)+ratio*Disp(i*2-1);
```

```
     Node(i,2)=Node(i,2)+ratio*Disp(i*2);
 end

 %----------------------------
 %  3. Plot the deformed shape
 %----------------------------

 for e1=1:nElem
    node1=Node(Element(e1,1),:);
    node2=Node(Element(e1,2),:);

    shift = 2*ones(1,2);

    midpoint = (node1+node2)/2;

    if node1(1)==node2(1)
        shift(2)=0;
    elseif node1(2)==node2(2)
        shift(1)=0;
    else
        midpoint = (2*node1+node2)/3;
        nodetmp = node1 - node2;
        if nodetmp(1)*nodetmp(2)>0
            shift(1)=-shift(1);
        end
    end

    hold on

    if Stress(e1)<0
        colour = 'red';
        width = 3;
        style = '-';
    else
        colour = 'blue';
        width = 2;
        style = '--';
    end

    plot([node1(1);node2(1)],[node1(2);node2(2)],colour, ...
                              'LineWidth',width,'LineStyle',style);
    text(midpoint(1)+shift(1),midpoint(2)+shift(2), ...

num2str(round(Stress(e1))),'Color',colour,'FontSize',12);
end

%------------------------------
%  4. Add boundary constraints
%------------------------------
nBC = size(BC,2);                  % Get the number of constrainted-DOFs
shift = -5;
for n1=1:nBC,
    nID = floor((BC(n1)+1)/2);    % Get the nodeID corresponding to DOF
    coord = Node(nID,:);          % Get the node's coordinates
    if mod(BC(n1),2)==1           % horizontal displacement constraint
       coord(1) = coord(1)+shift;
       sign = '>';
    else                          % vertical displacement constraint
       coord(2) = coord(2)+shift;
       sign = '^';
    end
```

```
    plot(coord(1),coord(2),sign,'MarkerEdgeColor','red',...
                        'MarkerFaceColor','green','MarkerSize',15);
end

%------------------------------
%  5. Add point loads
%------------------------------
for n1=1:sdof,
    if Force(n1)~=0
        nID = floor((n1+1)/2);  % Get the nodeID corresponding to DOF
        coord = Node(nID,:);    % Get the node's coordinates
        if mod(n1,2)==1         % horizontal point load
            if Force(n1)>0
                value = '\rightarrow';
            else
                value = '\leftarrow';
            end
        else                    % vertical point load
            if Force(n1)>0
                value = '\uparrow';
            else
                value = '\downarrow';
            end
        end
        text(coord(1),coord(2),value,'Color','red','FontSize',30);
    end
end

%------------------------------
%  6. Add titles
%------------------------------
title(['The ',int2str(nElem),'-bar truss problem'], ...

'Color','black','FontSize',16)
axis equal
```

BDisplayHistory.m

```
function BDisplayHistory(history)
% Display the history behavior of the truss
%------------------------------------------------------------
% BDisplayHistory(history)
%
% Variable Description:
%   history = the vector of load ratio of all members
%------------------------------------------------------------
figure
nElem = size(history,1);
list = sort(history);
num = [1:nElem];
his = plot(num,list,'LineWidth',2);
grid on;

%------------------------------
%  Add titles
%------------------------------
title(['The behavior of the',int2str(nElem),'-bar truss problem'],...
                                    'Color','black','FontSize',16)
xlabel('number of members fail to yielding/buckling','FontSize',14)
ylabel('load ratio','FontSize',14)

%------------------------------
%  Add values
%------------------------------
% Get the plotted data
x = get(his,'XData');
y = get(his,'YData');

% Find the index of the min and max
imin = find(min(y) == y);
imax = find(max(y) == y);

text(x(imin(1)),y(imin(1)),['\lambda_m_i_n = ', num2str(y(imin(1)))],
    'VerticalAlignment','top','HorizontalAlignment','left', ...
                                                      'FontSize',14)
text(x(imax(1)),y(imax(1)),['\lambda_m_a_x = ', num2str(y(imax(1)))],
    'VerticalAlignment','bottom','HorizontalAlignment','right',...
                                                      'FontSize',14)
```

CMainFEM.m

```
function
[Score,Vol,Disp,Stress,history] = CMainFEM(Solution,Data,StopIf)
% FEM Program solving the static 2-d truss structure
%------------------------------------------------------------
% [Score,Vol,Disp,Stress,history] = CMainFEM(Solution,Data,StopIf)
%
% Variable Description:
%   Score = the fitness value for the solution under consideration
%   Vol = the total volume for the solution under consideration
%   Disp = the column vector of global displacements
%   Stress = the column vector of global stresses
%   history = the vector of load ratio of all members
%   Solution = the set of section indices for all members
%   Data = the global variables
%   StopIf = The condition to stop the nonlinear analysis
%   StopIf = 0 : stop when the truss nearly collapse : for analysis
%   StopIf = 1 : stop when the maximum displacements is reached:forGA
%------------------------------------------------------------

%---------------------------
%  convertion of input data
%---------------------------
nDOFs = Data{1};    nNode = Data{2};    Node  = Data{3};
nElem = Data{5};    sdof  = Data{7};    ndoe  = Data{8};
Len   = Data{9};    Bcdof = Data{10};   Bcval = Data{11};
E     = Data{12};   Yield = Data{14};   LambdaCoef = Data{17};

%---------------------------
%  get the limits
%---------------------------

YieldLim  = 0.9*Yield;            % the yield limit of tension members
BuckleLim = zeros(nElem,1);       % the buckle limit of compression
members
slender = Len.*LambdaCoef(Solution)';  % the slenderness ratio vector

for i=1:nElem
    if slender(i)<=1.5
        BuckleLim(i) = 0.85*Yield*0.658^(slender(i)^2);
    else
        BuckleLim(i) = 0.85*Yield*0.877/(slender(i)^2);
    end
end

%---------------------------
%  initial phase
%---------------------------
NodeTmp = zeros(nNode,nDOFs);% temporary vector of global coordinates
Disp    = zeros(sdof,1);     % column vector of global displacements
disptmp = zeros(sdof,1);     % temporary vector of global disp.
dispinc = zeros(sdof,1);     % column vector of disp. increments

Stress    = zeros(nElem,1); % create column vector of global stresses
stresstmp = zeros(nElem,1); % temporary vector of global stresses
stressinc = zeros(nElem,1); % column vector of stresses increments

LengthTmp = zeros(nElem,1);
EtTmp     = zeros(nElem,1);
history   = zeros(nElem,1); % limit load-ratio vector of all elements
```

```
loadratio = 0.0;            % the load coefficient at current step
loadinc   = 0.04;           % the initial value of load increment
dForce    = zeros(sdof,1);  % create column vector of Force increment

nsteps    = 0;              % the total number of analysis steps
redo      = 0;              % assign the initial value of loop

Force = zeros(sdof,1);
Force = DCombineForces(Solution,Data);

%--------------------------
%  the trial run
%--------------------------
Et = E*ones(nElem,1);

kk = DKglobal(Solution,Data,Len,Et,Stress);
dForce = loadinc*Force;
[kk,dForce]= DapplyBC(kk,dForce,Bcdof,Bcval);
dispinc = kk\dForce;
stressinc = DPostProcess(Solution,Data,Len,Et,dispinc);

%--------------------------
%  get the tangent modulus
%--------------------------

ratioP = 0.0;
ratioE = 0.0;

for i=1:nElem                  % loop for all elements

    if slender(i)<=(1.5)
        ratioP = 0.658^(slender(i)^2);
    else
        ratioP = 0.877/(slender(i)^2);
    end

    if ratioP <= (0.39)
        ratioE = 1.0;
    else
        ratioE = -2.7243*ratioP*log(ratioP);
    end

    if stressinc(i)<0        % the member is in compression
        Et(i) = ratioE*E;
    end
end

%--------------------------
%  start the loops
%--------------------------

while (redo<3)                 % not all members reach to
yielding/buckling

    nsteps = nsteps + 1;

    % Forming global stiffness matrix
    kk     = DKglobal(Solution,Data,Len,Et,Stress);

    dForce = loadinc*Force;                    % determine a load step
```

```
    [kk,dForce]= DapplyBC(kk,dForce,Bcdof,Bcval);   % apply BC

    dispinc   = kk\dForce;                         % solve the equation

    Len = DUpdateGeometry(Data,dispinc);
    stressinc = DPostProcess(Solution,Data,Len,Et,dispinc);

    % get the total disp. and stresses upto the current step
    [Disp,Stress] =
            DupdateResults(Disp,Stress,dispinc,stressinc,history);

    % check these results whether satisfy the constraints
    historytmp = zeros(nElem,1);    % the temporary history vector
    [historytmp,ultimate] =
                  Dchecking(Solution,Data,Disp,Stress,StopIf, ...
                                      YieldLim,BuckleLim,loadratio);
    if ultimate
        loadratio = loadratio - loadinc;
        loadinc = loadinc/2;
        redo = redo+1;         % Totally 3 times of redo for all loops

        % return the values before this step
        nsteps = nsteps - 1;
        [Disp,Stress] =
            DRedoResults(Disp,Stress,dispinc,stressinc,history);
        Len = DRecoverGeometry(Data,dispinc);

    else
       % these values are accepted after running this step successful
        history = DUpdateHistory(history,historytmp);

    end
    loadratio = loadratio + loadinc;
end

loadratio = loadratio - loadinc;
history = DFinalHistory(history,loadratio);

%---------------------------
% calculate the penalty ratio by multiple linear segment approach
%---------------------------

[Score,Vol] = DPenalty(Solution,Data,Disp,history);
```

DCombineForces.m

```
function Force = DCombineForces(Solution,Data)
% Combine the self-weight load and the static point loads
%------------------------------------------------------------
% Force = DCombineForces(Solution,Data)
%
% Variable Description:
%   Solution = the set of section indices for all members
%   Data = the given input data of the truss problem
%------------------------------------------------------------

%--------------------------
%  convertion of input data
%--------------------------

nElem = Data{5};          Element = Data{6};       sdof = Data{7};
Length = Data{9};         LForce = Data{13};       Sections = Data{16};
Dfactor = Data{19};       Lfactor = Data{20};      rho = Data{21};

DForce=zeros(sdof,1);     % dead load vector

%--------------------------
%  loop for elements
%--------------------------
for iel=1:nElem     % loop for the total number of elements

% 1st connected node for the (iel)-th element
    nd(1)=Element(iel,1);
% 2nd connected node for the (iel)-th element
    nd(2)=Element(iel,2);
% DOF's number in y-axis at the starting node
    yDOFa = 2*nd(1);
% DOF's number in y-axis at the ending node
    yDOFb = 2*nd(2);
    weight = -rho*Length(iel)*Sections(Solution(iel));

    DForce(yDOFa) = DForce(yDOFa)+ weight/2;
    DForce(yDOFb) = DForce(yDOFb)+ weight/2;
end

Force = Dfactor*DForce + Lfactor*LForce;
```

DKglobal.m

```
function kk = DKglobal(Solution,Data,Length,Et,Stress)
% Forming the global stiffness matrix for the truss structure.
%-----------------------------------------------------------
% kk = DKglobal(Solution,Data,Length,Et,Stress)
%
% Variable descriptions
%   kk = system stiffness matrix
%   Solution = the section indices vector of every element
%   Data = the given input data of the truss problem.
%   Length = length vector for all elements
%   Et = the tangent modulus vector for all (compression) elements
%   Stress = the column vector of global stresses
%----------------------------------------------------------------------

%---------------------------
%  convertion of input data
%---------------------------
nDOFs = Data{1};                                   Node = Data{3};
nNODs = Data{4};      nElem = Data{5};     Element = Data{6};
sdof = Data{7};       ndoe = Data{8};
Section = Data{16};

%----------------------------
%  initialization to zero
%----------------------------
index=zeros(ndoe,1);      % index vector
k=zeros(ndoe,ndoe);       % element stiffness matrix, local
kk=zeros(sdof,sdof);      % system stiffness matrix, global

%--------------------------
%  loop for elements
%--------------------------
for iel=1:nElem     % loop for the total number of elements
% 1st connected node for the (iel)-th element
nd(1)=Element(iel,1);
% 2nd connected node for the (iel)-th element
nd(2)=Element(iel,2);
x1=Node(nd(1),1); y1=Node(nd(1),2);      % coordinate of 1st node
x2=Node(nd(2),1); y2=Node(nd(2),2);      % coordinate of 2nd node

if (x2-x1)==0;
if y2>y1;
   beta=2*atan(1);        % angle between local and global axes
else
   beta=-2*atan(1);
end
else
beta=atan((y2-y1)/(x2-x1));
end

el = Et(iel);                     % extract tangent modulus for member iel
leng = Length(iel);               % extract element length for member iel
area = Section(Solution(iel));    % extract sections for member
axial = Stress(iel);              % the axial stress of the element

index=EassignDOF(nd,nNODs,nDOFs);  % extract system dofs for element

% create the local Inelastic stiffness matrix
ke = EIstiffmat(el,leng,area,beta);
```

```
% create the local Geometric stiffness matrix
kg = EGstiffmat(axial,area,leng);

% combine the two local matrices
k = ke + kg;

kk=Eassembly(kk,k,index);          % assemble into system matrix

end
```

DApplyBC.m

```
function [kk,ff]=DapplyBC(kk,ff,bcdof,bcval)
% Apply constraints to matrix equation [kk]{du}={df}
%-------------------------------------------------------------
% [kk,ff]=DapplyBC(kk,ff,bcdof,bcval)
%
% Variable Description:
%     kk - system matrix before applying constraints
%     ff - system vector before applying constraints
%     bcdof - a vector containging constrained d.o.f
%     bcval - a vector containing contained value
%
%     For example, there are constraints at d.o.f=2 and 10
%     and their constrained values are 0.0 and 2.5,
%     respectively.  Then, bcdof(1)=2 and bcdof(2)=10; and
%     bcval(1)=1.0 and bcval(2)=2.5.
%-------------------------------------------------------------

 n=length(bcdof);
 sdof=size(kk);

 for i=1:n
    c=bcdof(i);
    for j=1:sdof
       kk(c,j)=0;
    end

    kk(c,c)=1;
    ff(c)=bcval(i);
 end
```

DPostProcess.m

```
function [StressInc] = DPostProcess(Solution,Data,Length,Et,DispInc)
% Post processing of FEA - stresses increment calculation
%------------------------------------------------------------
% [StressInc] = DPostProcess(Solution,Data,Length,Et,DispInc)
%
% Variable Description:

%   StressInc = increment stress vector for every element
%   Solution = the section indices vector of every element
%   Data = the given input data of the truss problem.
%   Length = length vector for all elements
%   Et = the tangent modulus vector for all (compression) elements
%   DispInc = the disp. increment after the current step
%------------------------------------------------------------

%---------------------------
%  convertion of input data
%---------------------------
nDOFs = Data{1};                               Node = Data{3};
nNODs = Data{4};    nElem = Data{5};          Element = Data{6};
sdof = Data{7};     ndoe = Data{8};
Section = Data{16};

%----------------------------
%  initialization to zero
%----------------------------
index=zeros(ndoe,1);          % index vector
elforce=zeros(ndoe,1);        % element force vector
eldisp=zeros(ndoe,1);         % element nodal displacement vector
k=zeros(ndoe,ndoe);           % element stiffness matrix, local
kk=zeros(sdof,sdof);          % system stiffness matrix, global
StressInc=zeros(nElem,1);     % stress vector for every element

%---------------------------------------------------
%  post computation for stress calculation
%---------------------------------------------------

for iel=1:nElem          % loop for the total number of elements

leng=Length(iel);        % element length
% 1st connected node for the (iel)-th element
nd(1)=Element(iel,1);
% 2nd connected node for the (iel)-th element
nd(2)=Element(iel,2);
x1=Node(nd(1),1); y1=Node(nd(1),2);     % coordinate of 1st node
x2=Node(nd(2),1); y2=Node(nd(2),2);     % coordinate of 2nd node

if (x2-x1)==0;
beta=2*atan(1);        % angle between local and global axes
else
beta=atan((y2-y1)/(x2-x1));
end

el = Et(iel);                   % extract tangent modulus for member iel
leng = Length(iel);             % extract element length for member iel
area = Section(Solution(iel)); % extract section for member

index=EassignDOF(nd,nNODs,nDOFs);  % extract system dofs for element
```

```
k=Estiffmass(el,leng,area,0,beta,1); % compute element matrix

for i=1:(ndoe)                        % extract displacements
eldisp(i)=DispInc(index(i));          % (iel)-th element
end

elforce=k*eldisp;                   % element force vector
StressInc(iel)=sqrt(elforce(1)^2+elforce(2)^2)/area; % stress
calculation

if ((x2-x1)*elforce(3)) < 0;
StressInc(iel)=-StressInc(iel);
end

end
```

DUpdateGeometry.m

```
function Length = DUpdateGeometry(Data,Disp)
% Update the diplacements into the current coordinates
% to get the current member's length vector
%------------------------------------------------------------
% Length = DUpdateGeometry(Data,Disp)
%
% Variable Description:
%   Length = (nElem x 1) vector, the member's length vector
%   Data = the global variables
%   Node = (nNode x nDOFs) matrix, the current coordinates
%   Disp = (sdof x 1) vector, the displacements vector
%------------------------------------------------------------

%--------------------------
%  convertion of input data
%--------------------------
nDOFs = Data{1};    nNode = Data{2};        Node = Data{3};
nElem = Data{5};    Element = Data{6};

%--------------------------
%  update the geometry
%--------------------------
CurrentNode = zeros(nNode,nDOFs);
for i=1:nNode
    CurrentNode(i,1)=Node(i,1)+Disp(i*2-1);
    CurrentNode(i,2)=Node(i,2)+Disp(i*2);
end

%--------------------------
%  update the length
%--------------------------
for i=1:nElem             % loop for the total number of elements

nd(1)=Element(i,1);       % 1st connected node for the (iel)-th element
nd(2)=Element(i,2);       % 2nd connected node for the (iel)-th element
% coord. of 1st node
x1=CurrentNode(nd(1),1); y1=CurrentNode(nd(1),2);
% coord. of 2nd node
x2=CurrentNode(nd(2),1); y2=CurrentNode(nd(2),2);
Length(i)=sqrt((x2-x1)^2+(y2-y1)^2);                % element length

end
```

DUpdateResuls.m

```
function
[Disp,Stress] = DupdateResults(Disp,Stress,dispinc,stressinc,history)
% Update the step results to the total results
%-----------------------------------------------------------
% [Disp,Stress] =
DupdateResults(Disp,Stress,dispinc,stressinc,history)
%
% Variable Description:
%   Disp = the column vector of global displacements
%   Stress = the column vector of global stresses
%   dispinc = the column vector of displacement increments
%   stressinc = the column vector of stresses increments
%   history = the vector of load ratio of all members
%-----------------------------------------------------------

Disp = Disp + dispinc;

nElem = size(Stress,1);
% The summation is neglected for the members which reached to
failures
for i=1:nElem
    if (history(i)==0)
        Stress(i) = Stress(i) + stressinc(i);
    end
end
```

DChecking.m

```
function [His,Ult] = Dchecking(Sol,Data,Disp,Stress,Stop, ...
                               YieldLim,BuckleLim,LR)
% Record the load-ratio at which a member reach to yielding/buckling
% Check whether all members reach to yielding/buckling
%-------------------------------------------------------------
% [His,Ult] =
Dchecking(Sol,Data,Disp,Stress,Stop,YieldLim,BuckleLim,LR)
%
% Variable Description:
%   His = (or History) store the limit load-ratio of all elements
%   Ult = (or Ultimate) boolean variable
%   Sol = (or Solution) the section indices vector of every element
%   Data = the given input data of the truss problem
%   Disp = the column vector of global displacements
%   Stress = the column vector of global stresses
%   YieldLim = Yield limit
%   BuckleLim = Buckle limits
%   LR = (or Load Ratio) the current value at the current step
%   Stop = The condition to stop the nonlinear analysis
%   Stop = 0 : stop when the truss nearly collapse : for analysis
%   Stop = 1 : stop when the maximum displacements is reached :for GA
%-------------------------------------------------------------

%--------------------------
%  convertion of input data
%--------------------------

nElem = Data{5};
maxdisp = Data{18};

%--------------------------
%  History
%--------------------------
His = zeros(nElem,1);

for i=1:nElem
    if Stress(i)>=0  % Member in tension

        if Stress(i)>YieldLim
            His(i)=LR;
        end

    else             % Member in compression

        if abs(Stress(i))>BuckleLim(i)
            His(i)=LR;
        end
    end
end

%--------------------------
%  Ultimate
%--------------------------
% If Stop = 0, Ultimate status means the truss nearly collapse
% If Stop = 1, The truss reaches to maximum displacement

if Stop==1      % check for maximum displacement
    if max(abs(Disp))>maxdisp
        Ult = true;
```

```
    else
        Ult = false;
    end
elseif Stop==0  % check for maximum load ratio
    Failure = His > 0.0;
    if sum(Failure) == nElem
        Ult = true;      % all members reach to yielding/buckling
    else
        Ult = false;     % NOT all members reach to yielding/buckling
    end
end
```

DRedoResults.m

```
function
[Disp,Stress] = DRedoResults(Disp,Stress,dispinc,stressinc,history)
% Recover the total results
%------------------------------------------------------------
% [Disp,Stress] = DRedoResults(Disp,Stress,dispinc,stressinc,history)
%
% Variable Description:
%   Disp = the column vector of global displacements
%   Stress = the column vector of global stresses
%   dispinc = the column vector of displacement increments
%   stressinc = the column vector of stresses increments
%   history = the vector of load ratio of all members
%------------------------------------------------------------

Disp = Disp - dispinc;

nElem = size(Stress,1);
% The subtraction is neglected for the members which reached to
failures
for i=1:nElem
    if (history(i)==0)
        Stress(i) = Stress(i) - stressinc(i);
    end
end
```

DRecoverGeometry.m

```
function Length = DRecoverGeometry(Data,Disp)
% Recover the diplacements from the current coordinates
%------------------------------------------------------------
% Length = DRecoverGeometry(Data,Disp)
%
% Variable Description:
%   Length = (nElem x 1) vector, the member's length vector
%   Data = the global variables
%   Node = (nNode x nDOFs) matrix, the current coordinates
%   Disp = (sdof x 1) vector, the displacements vector
%------------------------------------------------------------

%--------------------------
%  convertion of input data
%--------------------------
nDOFs = Data{1};    nNode = Data{2};       Node = Data{3};
nElem = Data{5};    Element = Data{6};

%--------------------------
%  update the geometry
%--------------------------
CurrentNode = zeros(nNode,nDOFs);
for i=1:nNode
    CurrentNode(i,1)=Node(i,1)-Disp(i*2-1);
    CurrentNode(i,2)=Node(i,2)-Disp(i*2);
end

%--------------------------
%  update the length
%--------------------------
for i=1:nElem             % loop for the total number of elements

nd(1)=Element(i,1);       % 1st connected node for the (iel)-th element
nd(2)=Element(i,2);       % 2nd connected node for the (iel)-th element

% coord. of 1st node
x1=CurrentNode(nd(1),1); y1=CurrentNode(nd(1),2);
% coord. of 2nd node
x2=CurrentNode(nd(2),1); y2=CurrentNode(nd(2),2);

Length(i)=sqrt((x2-x1)^2+(y2-y1)^2);               % element length

end
```

DUpdateHistory.m

```
function history = DUpdateHistory(history,now)
% Update the limit load-ratio vector of all elements
%------------------------------------------------------------
% history = DUpdateHistory(history,now)
%
% Variable Description:
% history = [nElem x 1] vector, store values of the greatest load-
% -ratio at which the particular member start to reach the failure
% status.
% now = [nElem x 1] vector, store the same value of the load-ratio
% at a arbitrary step to all the failed members upto that step
%------------------------------------------------------------

for i=1:size(history,1)
    if ((history(i)==0) & (now(i)~=0))
        history(i)=now(i);
    end
end
```

DFinalHistory.m

```
function history = DFinalHistory(history,loadratio)
% assign the greatest loadratio value to the last zero-value in
history
%-----------------------------------------------------------
% history = DFinalHistory(history,loadratio)
%
% Variable Description:
%   history = the vector of load ratio of all members
%   loadratio = the load coefficient at current step
%-----------------------------------------------------------

for i=1:size(history,1)
    if (history(i)==0)
        history(i)=loadratio;
    end
end
```

DPenalty.m

```
function [score,volume] = DPenalty(Solution,Data,disp,history)
% Calculate the penalty ratio by multiple linear segment approach
%-----------------------------------------------------------
% [score,volume] = DPenalty(Solution,Data,disp,history)
%
% Variable Description:
%   Score = the fitness value for the solution under consideration
%   Volume = the total volume for the solution under consideration
%   Solution = the set of section indices for all members
%   Data = the global variables
%   Disp = the column vector of global displacements
%   history = the vector of load ratio of all members
%-----------------------------------------------------------

%--------------------------
%  convertion of input data
%--------------------------

nElem = Data{5};           % Total number of elements
sdof = Data{7};            % total system dofs = nnode x ndof
Length = Data{9};          % Length of elements
Section = Data{16};        % The lists of availabe cross-sectional areas
MaxD = Data{18};           % maximum allowable dissplacement

%---------------------------------------------------------------
%  loop for nodes, check for the maximum allowable displacement
%---------------------------------------------------------------
PenalD = 1.0;              % coefficient ki = 1 for all nodes
for inode = 1:2:sdof
    dx = disp(inode,1); dy = disp(inode+1,1);
    if dx>MaxD
        PenalD = PenalD*(dx/MaxD);
    end
    if dy>MaxD
        PenalD = PenalD*(dy/MaxD);
    end
end

%---------------------------------------------------------------
%  consider all elements, check for the load-ratio in vector history
%---------------------------------------------------------------
PenalS = EPenalHistory(history);

%-------------------------
%  total values of penalties
%-------------------------
penalty = PenalD*PenalS;
volume = sum(Section(Solution).*Length');
score = penalty*volume;
```

Eassembly.m

```
function [kk]=Eassembly(kk,k,index)
% Assembly of element matrices into the system matrix
%------------------------------------------------------------
% [kk]=Eassembly(kk,k,index)
%
% Variable Description:
%     kk - system matrix
%     k  - element matrix
%     index - d.o.f. vector associated with an element
%------------------------------------------------------------

 edof = length(index);
 for i=1:edof
   ii=index(i);
     for j=1:edof
       jj=index(j);
         kk(ii,jj)=kk(ii,jj)+k(i,j);
     end
 end
```

EassignDOF.m

```
function [index]=EassignDOF(nd,nnel,ndof)
% Compute system dofs associated with each element
%-----------------------------------------------------------
% [index]=EassignDOF(nd,nnel,ndof)
%
% Variable Description:
%     index - system dof vector associated with element "iel"
%     nd - element number whose system dofs are to be determined
%     nnel - number of nodes per element
%     ndof - number of dofs per node
%-----------------------------------------------------------

 edof = nnel*ndof;
   k=0;
   for i=1:nnel
     start = (nd(i)-1)*ndof;
       for j=1:ndof
         k=k+1;
         index(k)=start+j;
       end
   end
```

EGstiffmat.m

```
function kg = EGstiffmat(axial,area,leng)
% Create the local Geometric Stiffness for the 2-d truss element
% nodal dof {u_1 v_1 u_2 v_2}
%-----------------------------------------------------------
% kg = EGstiffmat(axial,area,leng)
%
% Variable Description:
%     kg - element geometric stiffness matrix (size of 4x4)
%     axial - the axial stress of the element
%     area - area of truss cross-section
%     leng - element length
%-----------------------------------------------------------

% stiffness matrix

 c = axial*area/leng;

 kg = c*[ 1.0   0.0  -1.0   0.0;...
          0.0   1.0   0.0  -1.0;...
         -1.0   0.0   1.0   0.0;...
          0.0  -1.0   0.0   1.0];
```

EIstiffmat.m

```
function ki = EIstiffmat(el,leng,area,beta)
% Create the local Inelastic Stiffness for the 2-d truss element
% nodal dof {u_1 v_1 u_2 v_2}
%-----------------------------------------------------------
% ki = EIstiffmat(el,leng,area,beta)
%
% Variable Description:
%     ki - element inelastic stiffness matrix (size of 4x4)
%     el - elastic modulus
%     leng - element length
%     area - area of truss cross-section
%     beta - angle between the local and global axes
%            positive if the local axis is in the ccw direction from
%            the global axis
%-----------------------------------------------------------

% stiffness matrix

 c = cos(beta);
 s = sin(beta);
 ki = (area*el/leng)*[ c*c   c*s  -c*c  -c*s;...
                      c*s   s*s  -c*s  -s*s;...
                     -c*c  -c*s   c*c   c*s;...
                     -c*s  -s*s   c*s   s*s];
```

EPenalHistory.m

```
function coeff = EPenalHistory(history)
% find the penalty value of the load ratio history
%------------------------------------------------------------
% coeff = EPenalHistory(history)
%
% Variable Description:
%   coeff = the output value
%   history = the vector of load ratio of all members
%------------------------------------------------------------

num = size(history,1);
if num>3
    num=3;
end

his = sort(history);

for i=1:num

    lambda = his(i);

    if lambda<=1.1
        tmp = (lambda-1.1)^2+1;

    else
        tmp = (((lambda-1.1)^2)/2)+1;

    end

    results(i) = tmp;

end

coeff = prod(results);
```

VITA

Mr. Tien-Dac TRAN
2B Bui-Minh-Truc Residential Area, ward 5, district 8, Ho-Chi-Minh city, Vietnam.
Home phone: +848 8502679 Email: tiendac@yahoo.com, tiendac@gmail.com

Education

May 2005	Master of Engineering in Civil Engineering, Chulalongkorn University, Thailand. Dissertation: Sizing Optimization of Nonlinear Planar Steel Trusses Using Genetic Algorithm. Thesis advisor: Assit.Prof. Thanyawat POTHISIRI, Ph.D.
January 2000	Bachelor of Engineering in Civil Engineering, Ho-Chi-Minh city University of Technology, Vietnam (Silver Medal of Graduation). Dissertation: Saigon Metropolitan Tower – The EXSAP's software Thesis advisor: Assoc.Prof. Cong-Thanh BUI, Ph.D.

Teaching Experience

Sep 2002	Lecturer, Department of Civil Engineering, Ho-Chi-Minh city University of Technology, Vietnam.
April 2000	Teaching Assistant, Department of Civil Engineering, Ho-Chi-Minh city University of Technology, Vietnam.

Honors and Awards (nationwide level)

2003	Medal "The Knight in IT".
2000	First prize of "Students With Scientific Researches" award, First prize of "Technological Creativity – Vifotec" award, medal of "Creative Youth",etc.