Travail de Fin d'Etudes : Développement d'un modèle novateur de caractérisation du comportement précis d'assemblages de construction métallique et mixte acier-béton

Auteur : Mathieu, Julien
Promoteur(s) : Jaspart, Jean-Pierre
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil des constructions, à finalité spécialisée en "civil engineering"
Année académique : 2019-2020
URI/URL : http://hdl.handle.net/2268.2/8929

Data processing

```
% Generation of mechanical spring model for FINELG treatment, including
% group effects and components ductility

% Conducted as part of master thesis relating to the development of an
% innovative model for the precise characterization of steel and
   composite
% joints

% MATHIEU Julien

close all
clear
clc

%% General data
% Lines and columns of data_all must be verified if excel file is
   changed

% Name of FINELG input file
Name = 'Input_Finelg';
Ext = '.dat';

% Extraction of data from excel file
data_all = xlsread('Data.xlsx');

% Name of .txt file usefull for results processing
Name_txt = 'Loads_Dep';
Ext_txt = '.txt';

% Number of rows in tension
n_rows_t = data_all(10,1);
% Number of rows in compression
n_rows_c = data_all(11,1);
% Total number of rows
n_rows_tot = n_rows_t + n_rows_c;

% Total number of groups
n_gr_tot = n_rows_t*(n_rows_t-1)/2;
% Total number of demultiplication of the rows
n_demult = n_gr_tot-(n_rows_t-1)-(n_rows_t-2)+1;

% Location of the point of application of the axial load
CG_beam = data_all(12,1);

% Bending moment
M = data_all(1,1)*10^6;
% Axial force
N = data_all(2,1)*10^3;
% Transversal load
```

```
48 V = data_all(3,1)*10^3;
49
50 % Initial increment
51 incr_ini = data_all(4,1);
52 % Initial small axial force (sequences 1 and 3)
53 N1 = data_all(5,1)*10^3;
54 % Initial small bending moment (sequence 2)
55 M1 = data_all(6,1)*10^6;
56
57 % First line of table 'Components data' in 'data_all'
58 first_line_comp = 2;
59 % First column of table 'Components data' in 'data_all'
60 first_column_comp = 3;
61 % Last line of table 'Components data' in 'data_all'
62 last_line_comp = size(data_all,1);
63
64 % First line of table 'Groups data' in 'data_all'
65 first_line_gr = 2;
66 % First column of table 'Groups data' in 'data_all'
67 first_column_gr = 11;
68 % Last line of table 'Groups data'
69 last_line_gr = first_line_gr+n_gr_tot-1;
70
71 %% Rows general data
72
73 % Column 1 : Row number
74 % Column 2 : Distance between the bottom row and row i
75 % Column 3 : Row in tension(1)/compression(0)
76 % Column 4 : Number of groups in row i
77 % Column 5 : Number of groups seperating the row i (number of "cuts")
78 % Column 6 : Total number of elements in row i
79 % Column 7 : First element in row i
80 % Column 8 : Last element in row i
81 % Column 9 : Total number of nodes in row i
82 % Column 10 : First node in row i
83 % Column 11 : Last node in row i
84
85 data_rows = zeros(n_rows_tot,4);
86 nt = 0;
87 for i = 1:n_rows_tot
88
89     data_rows(i,1) = i;
90
91     for j = first_line_comp:last_line_comp
92         if data_all(j,first_column_comp) == i
93
94             data_rows(i,2) = data_all(j,first_column_comp+1);
95             data_rows(i,3) = data_all(j,first_column_comp+2);
96
97         end
98     end
```

```
99
100     if data_rows(i,3) == 1
101         nt = nt+1;
102         if nt == 1
103
104             data_rows(i,4) = n_rows_t-1;
105
106         elseif nt <= n_demult+1
107
108             n = 1;
109             while data_rows(i-n,3) == 0
110                 n = n+1;
111             end
112             data_rows(i,4) = data_rows(i-n,4)-1;
113
114         else
115
116             n = 1;
117             while data_rows(i-n,3) == 0
118                 n = n+1;
119             end
120             data_rows(i,4) = data_rows(i-n,4)-1;
121
122         end
123     end
124 end
125
126 % Number of groups considering row i in each zone
127 % Each zone j is composed of all equivalent springs next to groups j to
      i
128 % Number of zones = total number of rows in tension - 3 (only one
129 % equivalent spring in the two first rows and no group in last row)
130 for i = 1:n_rows_t
131     for j = 1:n_rows_t-2-1
132
133         if i <= 2+(j-1)
134
135             nb_groups_zone(i,j) = 0;
136
137         elseif i == 2+(j-1)+1
138
139             nb_groups_zone (i,j) = n_rows_t-i;
140
141         elseif i == 2+(j-1)+2
142
143             nb_groups_zone (i,j) = nb_groups_zone(i-1,j);
144
145         else
146
147             nb_groups_zone (i,j) = nb_groups_zone(i-1,j)-1;
148
```

```
        end
    end
end

% Total number of groups considering row i
n = 0;
for i = 1:n_rows_tot
    if data_rows(i,3) == 1
        n = n+1;
        data_rows(i,5) = sum(nb_groups_zone(n,:));
    end
end

n = 0;
for i = 1:n_rows_tot
    if data_rows(i,3) == 1
        n = n+1;

        if n <= 2
            data_rows(i,6) = data_rows(i,4)+1;
            data_rows(i,9) = data_rows(i,6)+1;
        else
            data_rows(i,6) = data_rows(i,4)+n_demult;
            data_rows(i,9) = data_rows(i,5)+n_demult+1;
        end

    else
        data_rows(i,6) = 1;
        data_rows(i,9) = 2;
    end

    data_rows(i,7) = sum(data_rows(1:i,6))-data_rows(i,6)+1;
    data_rows(i,8) = sum(data_rows(1:i,6));
    data_rows(i,10) = sum(data_rows(1:i,9))-data_rows(i,9)+1;
    data_rows(i,11) = sum(data_rows(1:i,9));

end

% Additional node if the last row is in tension
if data_rows(end,3) ~= 0
    data_rows(end,9) = data_rows(end,9)+1;
    data_rows(i,11) = data_rows(i,11)+1;
end

% Separation between rows in tension and rows in compression
nt = 0;
nc = 0;
for i = 1:n_rows_tot
    if data_rows(i,3) == 1

        nt = nt+1;
```

```
200         data_rows_t(nt,:) = data_rows(i,:);
201
202     else
203
204         nc = nc+1;
205         data_rows_c(nc,:) = data_rows(i,:);
206
207     end
208 end
209
210 % Number of nodes
211 n_nodes = data_rows(end,11)+1;
212
213 % Number of linear constrains in rows in tension 3 to n_rows_t
214 n = 0;
215 for i = 3:n_rows_t
216
217     n = n+1;
218     lin_const_int(n,1) = sum(1:1:data_rows_t(i,4));
219
220 end
221
222 % Total number of linear constrains
223 if data_rows(end,3) == 0
224     n_lin_const = sum(lin_const_int)+sum(data_rows_t(1:2,4))+(n_demult-1)*(n_rows_t
        -2)+n_rows_c+1;
225 else
226     n_lin_const = sum(lin_const_int)+sum(data_rows_t(1:2,4))+(n_demult-1)*(n_rows_t
        -2)+n_rows_c+2;
227 end
228
229 % Number of fictional nodes
230 n_fict_nodes = ceil(n_lin_const/3);
231
232 % Total number of nodes
233 n_nodes_tot = n_nodes + n_fict_nodes;
234
235
236 %% Nodes coordinates
237
238 coord = zeros(n_nodes_tot,5);
239
240 % FINELG input
241 % Nodes numbering
242 for i = 1:n_nodes_tot
243
244     coord(i,1) = i;
245
246 end
247
248 for i = 1:n_nodes
```

```
249
250     coord(i,2) = 0;
251     coord(i,3) = 0;
252     coord(i,5) = 0;
253
254 end
255
256 % Distance from the bottom row to the node
257 for i = 1:n_rows_tot
258     for j = data_rows(i,10):data_rows(i,11)
259
260         coord(j,4) = data_rows(i,2);
261
262     end
263 end
264
265 % Coordinates of the point of application of the axial load
266 coord(end-n_fict_nodes,4) = CG_beam;
267
268
269 %% Constitutive laws
270
271 % Constitutive laws of the components
272 for i = 1:n_rows_tot
273     n = 0;
274     for j = 1:last_line_comp-first_line_comp+1
275         if data_all(first_line_comp-1+j,first_column_comp) == i
276             n = n+1;
277
278             % Bi-linear law
279             if data_all(first_line_comp-1+j,first_column_comp+3) == 1
280
281                 Resistance(n,1,i) = data_all(first_line_comp-1+j,first_column_comp+5)
                        ;
282                 Rigidity(n,1,i) = min(data_all(first_line_comp-1+j,first_column_comp
                        +4),500000000);
283                 Delta(n,i) = data_all(first_line_comp-1+j,first_column_comp+6);
284
285                 % Tri-linear law
286             elseif data_all(first_line_comp-1+j,first_column_comp+3) == 2
287
288                 Resistance(n,2,i) = data_all(first_line_comp-1+j,first_column_comp+5)
                        ;
289                 Resistance(n,1,i) = Resistance(n,2,i)*2/3;
290                 Rigidity(n,1,i) = min(data_all(first_line_comp-1+j,first_column_comp
                        +4),500000000);
291                 Rigidity(n,2,i) = (Resistance(n,2,i)-Resistance(n,1,i))/(Resistance(n,2,i)
                        /(Rigidity(n,1,i)/3)-Resistance(n,1,i)/Rigidity(n,1,i));
292                 Delta(n,i) = data_all(first_line_comp-1+j,first_column_comp+6);
293
294                 % Multi-linear law
```

```
295             elseif data_all(first_line_comp-1+j,first_column_comp+3) == 3
296
297                 Resistance(n,4,i) = data_all(first_line_comp-1+j,first_column_comp+5)
                        ;
298                 Resistance(n,1,i) = Resistance(n,4,i)*6/9;
299                 Resistance(n,2,i) = Resistance(n,4,i)*7/9;
300                 Resistance(n,3,i) = Resistance(n,4,i)*8/9;
301                 Rigidity(n,1,i) = min(data_all(first_line_comp-1+j,first_column_comp
                        +4),500000000);
302                 Rigidity(n,2,i) = (Resistance(n,2,i)-Resistance(n,1,i))/(Resistance(n,2,i)
                        /(Rigidity(n,1,i)/1.5)-Resistance(n,1,i)/Rigidity(n,1,i));
303                 Rigidity(n,3,i) = (Resistance(n,3,i)-Resistance(n,2,i))/(Resistance(n,3,i)
                        /(Rigidity(n,1,i)/2.2)-Resistance(n,1,i)/(Rigidity(n,1,i))/1.5);
304                 Rigidity(n,4,i) = (Resistance(n,4,i)-Resistance(n,3,i))/(Resistance(n,4,i)
                        /(Rigidity(n,1,i)/3)-Resistance(n,1,i)/(Rigidity(n,1,i))/2.2);
305                 Delta(n,i) = data_all(first_line_comp-1+j,first_column_comp+6);
306
307             end
308         end
309     end
310 end
311
312 % Constitutive laws of the groups
313 data_laws_gr(:,1) = data_all(first_line_gr:last_line_gr,first_column_gr);
314 data_laws_gr(:,2) = data_all(first_line_gr:last_line_gr,first_column_gr+1);
315 data_laws_gr(:,3) = data_all(first_line_gr:last_line_gr,first_column_gr+2);
316 data_laws_gr(:,4) = data_all(first_line_gr:last_line_gr,first_column_gr+3);
317
318 % Constitutive laws of the equivalent springs
319 n = 0;
320 for i = 1:n_rows_tot
321     clear temp
322     for j = 1:nnz(any(Resistance(:,:,i),1))
323         temp(size(Resistance,1)*(j-1)+1:size(Resistance,1)*j,1) = Resistance(:,j,i);
324     end
325     temp = sort(temp(temp>0));
326
327     K_i(:,1,i) = Rigidity(:,1,i);
328
329     for j = 1:size(temp)
330         [r,c] = find(Resistance(:,:,i)==temp(j,1));
331         if c ~= nnz(any(Resistance(:,:,i),1))
332
333             data_laws_spring(i,j*2) = temp(j,1);
334             K_i(:,j+1,i) = K_i(:,j,i);
335             K_i(r,j+1,i) = Rigidity(r,c+1,i);
336
337         else
338
339             data_laws_spring(i,j*2) = temp(j,1);
340
```

```
341             break
342         end
343     end
344
345     K_i(K_i==0)=NaN;
346     inv_K_i = 1./K_i;
347
348     for j = 1:nnz(any(inv_K_i(:,:,i),1))
349         data_laws_spring(i,j*2-1) = 1/nansum(inv_K_i(:,j,i));
350     end
351
352     if data_rows(i,3) == 1
353         n = n+1;
354         if n > 2
355             for j = 1:2:nnz(any(data_laws_spring(i,:),1))
356
357                 data_laws_spring(i,j) = data_laws_spring(i,j)*n_demult;
358
359             end
360         end
361     end
362
363     % Maximum displacement of the equivalent spring
364     [r_d,c_d] = find(Resistance(:,nnz(any(Resistance(:,:,i),1)),i)==data_laws_spring(i,nnz
            (any(data_laws_spring(i,:),1))));
365     n_r = 1;
366     Delta_spring(i,n_r) = data_laws_spring(i,2)/data_laws_spring(i,1);
367     for j = 3:2:nnz(any(data_laws_spring(i,:),1))-1
368         n_r = n_r+1;
369         Delta_spring(i,n_r) = Delta_spring(i,n_r-1)+(data_laws_spring(i,j+1)-
            data_laws_spring(i,j-1))/data_laws_spring(i,j);
370     end
371
372     Delta_max(i,1) = Delta_spring(i,nnz(any(Delta_spring(i,:),1)))+Delta(r_d,i);
373
374 end
375
376 % Maximum displacement of the groups
377 for i = 1:n_gr_tot
378
379    Delta_max(n_rows_tot+i,1) = data_laws_gr(i,3)/500000000 + data_laws_gr(i,4);
380
381 end
382
383 % Conversion to FINELG laws parameters
384 % Column 1 : Constitutive law number
385 % Column 2 : law type
386 % Column 3 to ... : law parameters
387
388 for i = 1:size(data_laws_spring,1)
389
```

```
390     meca(i,1) = i;
391     meca(i,2) = 11;
392     meca(i,3) = data_laws_spring(i,1);
393     meca(i,4) = 0;
394 
395     for j = 2:2:nnz(any(data_laws_spring(i,:),1))
396 
397         meca(i,j+3) = data_laws_spring(i,j);
398 
399     end
400 
401     if nnz(any(data_laws_spring(i,:),1)) > 2
402 
403         meca(i,6) = data_laws_spring(i,2)/data_laws_spring(i,1)+(data_laws_spring(i,4)
                -data_laws_spring(i,2))/(data_laws_spring(i,3));
404 
405         for j = 5:2:nnz(any(data_laws_spring(i,:),1))
406 
407             meca(i,j+3) = meca(i,j-2+3) + (data_laws_spring(i,j+1)-data_laws_spring(i
                    ,j-1))/(data_laws_spring(i,j));
408 
409         end
410 
411     end
412     meca(i,nnz(any(meca(i,:),1))+2) = 1;
413 end
414 
415 if size(meca,2) > 20
416     error('Equivalent spring constitutive law too nonlinear, please choose another component
            constitutive law type')
417 end
418 
419 % Groups laws
420 for i = 1:size(data_laws_gr,1)
421 
422     meca(end+1,1) = meca(end,1)+1;
423     meca(end,2) = 2;
424     meca(end,3) = 500000000;
425     meca(end,4) = 0;
426     meca(end,5) = data_laws_gr(i,3);
427     meca(end,6) = 1;
428 
429 end
430 
431 % Rigid material law
432 meca(end+1,1) = meca(end,1)+1;
433 meca(end,2) = 0;
434 meca(end,3) = 200000;
435 meca(end,4) = 3.0000E-01;
436 
437 % "Constitutive laws" for linear constrains
```

```
% Column 1 : law number
% Column 3 to ... : linear constrain parameters

% Linear constrains for groups 1 to i
meca(end+1,1) = meca(end,1)+1;
meca(end,2) = 0;
meca(end,3) = 1;
meca(end,4) = -1;

% Linear constrains for groups 3 => n_rows_t-1 to i
meca(end+1,1) = meca(end,1)+1;
meca(end,2) = 0;
meca(end,3) = 1;
meca(end,4) = -1;
meca(end,5) = -1;
meca(end,6) = 1;

% Linear constrains for groups 2 to i
for i = 2 : (n_rows_t-1)

    meca(end+1,1) = meca(end,1)+1;
    meca(end,2) = 0;
    meca(end,3) = data_rows_t(1,2);
    meca(end,4) = -data_rows_t(1,2);
    meca(end,5) = -(data_rows_t(2,2)-data_rows_t(2+(i-1),2));
    meca(end,6) = data_rows_t(2,2)-data_rows_t(2+(i-1),2);

end

% Linear constrains for the "Bernouilli element"
x = 0;
nt = 0;
if data_rows(end,3) == 1
    for i = 1:n_rows_tot
        if x < n_rows_c+1
            if data_rows(i,3) == 1
                nt = nt+1;
            end
            if nt < 2

                x = x+1;
                meca(end+1,1) = meca(end,1)+1;
                meca(end,2) = 0;
                meca(end,3) = data_rows(i+1,2);
                meca(end,4) = data_rows(1,2)-data_rows(i+1,2);
                meca(end,5) = -data_rows(1,2);

            else
                if data_rows(i+1,3)==1
                    continue
                else
```

```
489
490                     x = x+1;
491                     meca(end+1,1) = meca(end,1)+1;
492                     meca(end,2) = 0;
493                     meca(end,3) = data_rows(i+1,2);
494                     meca(end,4) = data_rows(1,2)-data_rows(i+1,2);
495                     meca(end,5) = -data_rows(1,2);
496
497                 end
498             end
499         end
500     end
501 else
502     for i = 1:n_rows_tot
503         if x < n_rows_c
504             if data_rows(i,3) == 1
505                 nt = nt+1;
506             end
507             if nt < 2
508
509                 x = x+1;
510                 meca(end+1,1) = meca(end,1)+1;
511                 meca(end,2) = 0;
512                 meca(end,3) = data_rows(i+1,2);
513                 meca(end,4) = data_rows(1,2)-data_rows(i+1,2);
514                 meca(end,5) = -data_rows(1,2);
515
516             else
517                 if data_rows(i+1,3)==1
518                     continue
519                 else
520
521                     x = x+1;
522                     meca(end+1,1) = meca(end,1)+1;
523                     meca(end,2) = 0;
524                     meca(end,3) = data_rows(i+1,2);
525                     meca(end,4) = data_rows(1,2)-data_rows(i+1,2);
526                     meca(end,5) = -data_rows(1,2);
527
528                 end
529             end
530         else
531             break
532         end
533     end
534 end
535
536 % Linear constrains for the "Bernouilli element" - Point of application
      of
537 % the axial load
538 meca(end+1,1) = meca(end,1)+1;
```

```
539 meca(end,2) = 0;
540 meca(end,3) = CG_beam;
541 meca(end,4) = data_rows(1,2)-CG_beam;
542 meca(end,5) = -data_rows(1,2);
543
544
545 %% Elements
546
547 % Column 1 : Element number
548 % Column 2 : Element identification number (FINELG)
549 % Column 3 : Constitutive law linked to the element
550 % Column 4 : Geometical parameter
551 % Column 5 : Fist node of the element
552 % Column 6 : Last node of the element
553 % Column 18 : row active in tension (1), compression (2) or both (0)
554 % Column 19 : identification of groups elements
555
556 % Groups and components
557 nt = 0;
558 n_delta = 0;
559 for i = 1:n_rows_tot
560     if data_rows(i,3) == 0
561         % Rows in compression
562         % Equivalent spring
563         for j = data_rows(i,7) : data_rows(i,8)
564
565             elements(j,1) = j;
566             elements(j,2) = 201;
567             elements(j,3) = i;
568             if j == 1
569                 elements(j,4) = 1;
570             else
571                 elements(j,4) = 0;
572             end
573             if i == 1
574                 elements(j,5) = j;
575             else
576                 if j == data_rows(i,7)
577                     elements(j,5) = data_rows(i,10);
578                 else
579                     elements(j,5) = elements(j-1,6);
580                 end
581             end
582             elements(j,6) = elements(j,5)+1;
583             elements(j,18) = 2;
584         end
585
586         % Identification of the node which will be usefull for the
587         % displacement verification
588         Delta_max(i,2) = elements(j,6);
589         % Negative displacement for equivalent spring in compression
```

```
590         Delta_max(i,1) = -Delta_max(i,1);
591 
592     else
593         % Rows in tension
594         nt = nt+1;
595         if nt == 1
596             % First row in tension
597             % Groups
598             n = 0;
599             for j = data_rows(i,7) : data_rows(i,4)
600 
601                 n = n+1;
602                 elements(j,1) = j;
603                 elements(j,2) = 201;
604                 elements(j,3) = n_rows_tot+(n_rows_t-1)-(j-data_rows(i,7));
605                 if j == 1
606                     elements(j,4) = 1;
607                 else
608                     elements(j,4) = 0;
609                 end
610                 if i == 1
611                     elements(j,5) = j;
612                 else
613                     if j == data_rows(i,7)
614                         elements(j,5) = data_rows(i,10);
615                     else
616                         elements(j,5) = elements(j-1,6);
617                     end
618                 end
619 
620                 elements(j,6) = elements(j,5)+1;
621 
622                 % Indentification of group element
623                 elements(j,19) = 1;
624 
625                 % Identification of the node which will be usefull for the
626                 % displacement verification
627                 Delta_max(n_rows_tot+(n_rows_t-1)-(n-1),2) = elements(j,6);
628 
629             end
630             % Equivalent spring
631             for j = data_rows(i,7)+data_rows(i,4) : data_rows(i,8)
632 
633                 elements(j,1) = j;
634                 elements(j,2) = 201;
635                 elements(j,3) = i;
636                 if j == 1
637                     elements(j,4) = 1;
638                 else
639                     elements(j,4) = 0;
640                 end
```

```
641                 if i == 1
642                     elements(j,5) = j;
643                 else
644                     if j == data_rows(i,7)
645                         elements(j,5) = data_rows(i,10);
646                     else
647                         elements(j,5) = elements(j-1,6);
648                     end
649                 end
650                 elements(j,6) = elements(j,5)+1;
651                 elements(j,18) = 1;
652 
653                 % Identification of the node which will be usefull for the
654                 % displacement verification
655                 Delta_max(i,2) = elements(j,6);
656 
657             end
658         elseif nt == 2
659             % Second row in tension
660             % Equivalent spring
661             for j = data_rows(i,7) : data_rows(i,8)-data_rows(i,4)
662 
663                 elements(j,1) = j;
664                 elements(j,2) = 201;
665                 elements(j,3) = i;
666                 elements(j,4) = 0;
667                 if j == data_rows(i,7)
668                     elements(j,5) = data_rows(i,10);
669                 else
670                     elements(j,5) = elements(j-1,6);
671                 end
672                 elements(j,6) = elements(j,5)+1;
673                 elements(j,18) = 1;
674 
675                 % Identification of the node which will be usefull for the
676                 % displacement verification
677                 Delta_max(i,2) = elements(j,6);
678 
679             end
680             % Groups
681             n = 0;
682             for j = data_rows(i,8)-data_rows(i,4)+1 : data_rows(i,8)
683 
684                 n = n+1;
685                 elements(j,1) = j;
686                 elements(j,2) = 201;
687                 elements(j,3) = n_rows_tot+(n_rows_t-1)+(j-(data_rows(i,8)-data_rows(
                        i,4)));
688                 elements(j,4) = 0;
689                 if j == data_rows(i,7)
690                     elements(j,5) = data_rows(i,10);
```

```
691                 else
692                     elements(j,5) = elements(j-1,6);
693                 end
694                 elements(j,6) = elements(j,5)+1;
695 
696                 % Indentification of group element
697                 elements(j,19) = 1;
698 
699                 % Identification of the node which will be usefull for the
700                 % displacement verification
701                 Delta_max(n_rows_tot+(n_rows_t-1)+n,2) = elements(j,6);
702 
703             end
704         else
705             % Other rows
706             for j = 1:size(nb_groups_zone,2)
707                 if nt > j+2
708                     % Equivalent springs in rows i > j+2 in each zone
709                     for k = data_rows(i,7) : data_rows(i,7)
710 
711                         elements(k,1) = k;
712                         elements(k,2) = 201;
713                         elements(k,3) = i;
714                         elements(k,4) = 0;
715                         elements(k,5) = data_rows(i,10);
716                         elements(k,6) = elements(k,5)+1;
717                         elements(k,18) = 1;
718 
719                     end
720                     for k = data_rows(i,7)+sum(data_rows_t(3:2+(j-1),4))+1 : data_rows(i
                        ,7)+sum(data_rows_t(3:2+(j-1),4))+nb_groups_zone(nt,j)
721 
722                         elements(k,1) = k;
723                         elements(k,2) = 201;
724                         elements(k,3) = i;
725                         elements(k,4) = 0;
726                         if k == data_rows(i,7)
727                             elements(k,5) = data_rows(i,10);
728                         else
729                             elements(k,5) = elements(k-1,6)+1;
730                         end
731                         elements(k,6) = elements(k,5)+1;
732                         elements(k,18) = 1;
733 
734                     end
735                     if nt > j+3
736                     for k = data_rows(i,7)+sum(data_rows_t(3:2+(j-1),4))+
                        nb_groups_zone(nt,j)+1 : data_rows(i,7)+sum(data_rows_t(3:2+(j
                        -1),4))+data_rows_t(j+2,4)
737 
738                         elements(k,1) = k;
```

```
739                     elements(k,2) = 201;
740                     elements(k,3) = i;
741                     elements(k,4) = 0;
742                     elements(k,5) = elements(k-1,6);
743                     elements(k,6) = elements(k,5)+1;
744                     elements(k,18) = 1;
745
746                 end
747                 end
748             elseif nt == j+2
749                 % Rows and zones including groups j+2 to i
750                 % Equivalent springs
751                 for k = data_rows(i,7)+sum(data_rows_t(3:nt,4))-data_rows(i,4) : 2 :
                        data_rows(i,7)+sum(data_rows_t(3:nt,4))-data_rows(i,4)+2*
                        data_rows(i,4)
752
753                     elements(k,1) = k;
754                     elements(k,2) = 201;
755                     elements(k,3) = i;
756                     elements(k,4) = 0;
757
758                     if k == data_rows(i,7)
759                         elements(k,5) = data_rows(i,10);
760                     elseif k ~= data_rows(i,7) && k == data_rows(i,7)+sum(
                            data_rows_t(3:nt,4))-data_rows(i,4)
761                         elements(k,5) = elements(k-1,6)+1;
762                     else
763                         elements(k,5) = elements(k-2,6)+1;
764                     end
765                     elements(k,6) = elements(k,5)+1;
766                     elements(k,18) = 1;
767
768                 end
769                 % Groups
770                 n = 0;
771                 for k = data_rows(i,7)+sum(data_rows_t(3:nt,4))-data_rows(i,4)+1 : 2
                        : data_rows(i,7)+sum(data_rows_t(3:nt,4))-data_rows(i,4)+2*
                        data_rows(i,4)-1
772
773                     elements(k,1) = k;
774                     elements(k,2) = 201;
775                     elements(k,3) = n_rows_tot+sum(data_rows(1:i,4))-(k-(k-n));
776                     elements(k,4) = 0;
777                     elements(k,5) = elements(k-1,6);
778                     elements(k,6) = elements(k,5)+1;
779                     n = n+1;
780                     % Indentification of group element
781                     elements(k,19) = 1;
782
783                     % Identification of the node which will be usefull
                            for the
```

```
784                     % displacement verification
785                     n_delta = n_delta+1;
786                     Delta_max(n_rows_tot+(n_rows_t-1)+(n_rows_t-2)+n_delta,2) =
                            elements(k,6);
787
788                 end
789             else
790                 % Springs after the groups (j+2) to i
791                 % Equivalent springs
792                 for k = data_rows(i,7)+sum(data_rows_t(3:nt,4))+data_rows(i,4) :
                        data_rows(i,8)
793
794                     elements(k,1) = k;
795                     elements(k,2) = 201;
796                     elements(k,3) = i;
797                     elements(k,4) = 0;
798                     elements(k,5) = elements(k-1,6);
799                     elements(k,6) = elements(k,5)+1;
800                     elements(k,18) = 1;
801
802                 end
803             end
804         end
805     end
806
807     % Identification of the node which will be usefull for the
808     % displacement verification
809     if nt > 2 && nt < n_rows_t
810         Delta_max(i,2) = data_rows_t(nt,11);
811     elseif nt == n_rows_t
812         if data_rows(end,3)==0
813             Delta_max(i,2) = data_rows_t(nt,11);
814         else
815             Delta_max(i,2) = data_rows_t(nt,11)-1;
816         end
817     end
818
819 end
820 end
821
822 % Elements connecting embedded supports
823 nt = 0;
824 n = 0;
825 for i = 1:n_rows_tot
826     if n < n_rows_c
827         if data_rows(i,3) == 1
828             nt = nt+1;
829             if nt == 1
830
831                 n=n+1;
832                 elements(end+1,1) = elements(end,1)+1;
```

```
833                 elements(end,2) = 33;
834                 elements(end,3) = n_rows_tot+n_gr_tot+1;
835                 if i == 1
836                     elements(end,4) = 1;
837                 else
838                     elements(end,4) = 0;
839                 end
840                 elements(end,5) = data_rows(i,10);
841                 for j = i+1:n_rows_tot
842                     if data_rows(j,3) == 0
843                         elements(end,6) = data_rows(j,10);
844                         break
845                     end
846                 end
847 
848             else
849                 continue
850             end
851         else
852 
853             n=n+1;
854             elements(end+1,1) = elements(end,1)+1;
855             elements(end,2) = 33;
856             elements(end,3) = n_rows_tot+n_gr_tot+1;
857             if i == 1
858                 elements(end,4) = 1;
859             else
860                 elements(end,4) = 0;
861             end
862             elements(end,5) = data_rows(i,10);
863             for j = i+1:n_rows_tot
864                 if nt >= 1
865                     if data_rows(j,3) == 0
866 
867                         elements(end,6) = data_rows(j,10);
868                         break
869 
870                     end
871                 else
872 
873                     elements(end,6) = data_rows(j,10);
874                     break
875 
876                 end
877             end
878         end
879     end
880 end
881 
882 % Linear constrains elements
883 % Column 1 : element number
```

```
884 % Column 2 : element indentification number
885 % Column 3 : "constitutive law" of the linear constrain
886 % Column 5 : node linked to the linear constrain
887 % Column 6 to ... : linear constrain parameters
888
889 % Linear constrains for groups 1 to i
890 n = 0;
891 for i = 1:data_rows_t(1,4)
892
893     elements(end+1,1) = elements(end,1)+1;
894     elements(end,2) = 221;
895     elements(end,3) = n_rows_tot+n_gr_tot+2;
896     if i == 1
897         elements(end,4) = 1;
898     else
899         elements(end,4) = 0;
900     end
901     if n < 3
902         elements(end,5) = coord(data_rows(end,11),1)+1+ceil(i/3);
903     else
904         elements(end,5) = elements(end-1,5)+1;
905     end
906     elements(end,6) = data_rows_t(1,10)+i;
907     elements(end,7) = data_rows_t(end-(i-1),10);
908     elements(end,8) = 0;
909     elements(end,9) = 0;
910     elements(end,10) = 0;
911     n = n+1;
912     if n == 4
913         n = 1;
914     end
915     elements(end,11) = n;
916     elements(end,12) = 1;
917     elements(end,13) = 1;
918
919 end
920
921 % Linear constrains of groups 3 => n_rows_t to i
922 for i = 3:n_rows_t-1
923     for j = 1:nnz(any(nb_groups_zone(i,:),1))
924         n_lc = 0;
925         if i == j+2
926             for k = sum(data_rows_t(3:2+(j-1),4)) : 2 : sum(data_rows_t(3:2+(j-1),4))+(
                    data_rows_t(i,4)-1)*2
927
928                 elements(end+1,1) = elements(end,1)+1;
929                 elements(end,2) = 221;
930                 elements(end,3) = n_rows_tot+n_gr_tot+3;
931                 elements(end,4) = 0;
932                 if n < 3
933                     elements(end,5) = elements(end-1,5);
```

```
934                 else
935                     elements(end,5) = elements(end-1,5)+1;
936                 end
937                 elements(end,6) = elements(k+data_rows_t(i,7),6);
938                 elements(end,7) = elements((k-n_lc)+data_rows_t(i+1,7),6);
939                 elements(end,8) = elements(end,6)+1;
940                 elements(end,9) = elements(end,7)+1;
941                 elements(end,10) = 0;
942                 elements(end,11) = 0;
943                 elements(end,12) = 0;
944                 n = n+1;
945                 if n == 4
946                     n = 1;
947                 end
948                 elements(end,13) = n;
949                 elements(end,14) = 1;
950                 elements(end,15) = 1;
951                 elements(end,16) = 1;
952                 elements(end,17) = 1;
953                 n_lc = n_lc+1;
954
955             end
956         else
957             for k = sum(data_rows_t(3:2+(j-1),4)) : sum(data_rows_t(3:2+(j-1),4))+
                    data_rows_t(i,4)-1
958
959                 elements(end+1,1) = elements(end,1)+1;
960                 elements(end,2) = 221;
961                 elements(end,3) = n_rows_tot+n_gr_tot+3;
962                 elements(end,4) = 0;
963                 if n < 3
964                     elements(end,5) = elements(end-1,5);
965                 else
966                     elements(end,5) = elements(end-1,5)+1;
967                 end
968                 elements(end,6) = elements(k+data_rows_t(i,7),6);
969                 elements(end,7) = elements(k+data_rows_t(i+1,7),6);
970                 elements(end,8) = elements(end,6)+1;
971                 elements(end,9) = elements(end,7)+1;
972                 elements(end,10) = 0;
973                 elements(end,11) = 0;
974                 elements(end,12) = 0;
975                 n = n+1;
976                 if n == 4
977                     n = 1;
978                 end
979                 elements(end,13) = n;
980                 elements(end,14) = 1;
981                 elements(end,15) = 1;
982                 elements(end,16) = 1;
983                 elements(end,17) = 1;
```

```
            end
        end
    end
end

% Linear constrains for groups 2 to i
for i = 1:data_rows_t(2,4)

    elements(end+1,1) = elements(end,1)+1;
    elements(end,2) = 221;
    elements(end,3) = elements(end-1,3)+1;
    elements(end,4) = 0;
    if n < 3
        elements(end,5) = elements(end-1,5);
    else
        elements(end,5) = elements(end-1,5)+1;
    end
    elements(end,6) = data_rows_t(2,10)+i;
    elements(end,7) = data_rows_t(2+i,11);
    elements(end,8) = data_rows(1,11);
    elements(end,9) = data_rows(end,11);
    elements(end,10) = 0;
    elements(end,11) = 0;
    elements(end,12) = 0;
    n = n+1;
    if n == 4
        n = 1;
    end
    elements(end,13) = n;
    elements(end,14) = 1;
    elements(end,15) = 1;
    elements(end,16) = 1;
    elements(end,17) = 1;

end

if data_rows(end,3) == 1
    elements(end,7) = data_rows_t(end,11)-1;
end

% Linear constrains of the "Bernouilli element"
nt = 0;
x = 0;
if data_rows(end,3) == 1
    for i = 1:n_rows_tot
        if x < n_rows_c+1
            if data_rows(i,3) == 1
                nt = nt+1;
            end
            if nt < 2

```

```
                elements(end+1,1) = elements(end,1)+1;
                elements(end,2) = 221;
                elements(end,3) = elements(end-1,3)+1;
                elements(end,4) = 0;
                if n < 3
                    elements(end,5) = elements(end-1,5);
                else
                    elements(end,5) = elements(end-1,5)+1;
                end
                elements(end,6) = data_rows(1,11);
                elements(end,7) = data_rows(end,11);
                elements(end,8) = data_rows(i+1,11);
                elements(end,9) = 0;
                elements(end,10) = 0;
                elements(end,11) = 0;
                n = n+1;
                if n == 4
                    n = 1;
                end
                elements(end,12) = n;
                elements(end,13) = 1;
                elements(end,14) = 1;
                elements(end,15) = 1;
                x=x+1;

            else
                if data_rows(i+1,3)==1
                    continue
                else
                    elements(end+1,1) = elements(end,1)+1;
                    elements(end,2) = 221;
                    elements(end,3) = elements(end-1,3)+1;
                    elements(end,4) = 0;
                    if n < 3
                        elements(end,5) = elements(end-1,5);
                    else
                        elements(end,5) = elements(end-1,5)+1;
                    end
                    elements(end,6) = data_rows(1,11);
                    elements(end,7) = data_rows(end,11);
                    elements(end,8) = data_rows(i+1,11);
                    elements(end,9) = 0;
                    elements(end,10) = 0;
                    elements(end,11) = 0;
                    n = n+1;
                    if n == 4
                        n = 1;
                    end
                    elements(end,12) = n;
                    elements(end,13) = 1;
                    elements(end,14) = 1;
```

```
                    elements(end,15) = 1;
                    x = x+1;

                end
            end
        end
    end
else
    for i = 1:n_rows_tot
        if x < n_rows_c
            if data_rows(i,3) == 1
                nt = nt+1;
            end
            if nt < 2

                elements(end+1,1) = elements(end,1)+1;
                elements(end,2) = 221;
                elements(end,3) = elements(end-1,3)+1;
                elements(end,4) = 0;
                if n < 3
                    elements(end,5) = elements(end-1,5);
                else
                    elements(end,5) = elements(end-1,5)+1;
                end
                elements(end,6) = data_rows(1,11);
                elements(end,7) = data_rows(end,11);
                elements(end,8) = data_rows(i+1,11);
                elements(end,9) = 0;
                elements(end,10) = 0;
                elements(end,11) = 0;
                n = n+1;
                if n == 4
                    n = 1;
                end
                elements(end,12) = n;
                elements(end,13) = 1;
                elements(end,14) = 1;
                elements(end,15) = 1;
                x=x+1;

            else
                if data_rows(i+1,3)==1
                    continue
                else
                    elements(end+1,1) = elements(end,1)+1;
                    elements(end,2) = 221;
                    elements(end,3) = elements(end-1,3)+1;
                    elements(end,4) = 0;
                    if n < 3
                        elements(end,5) = elements(end-1,5);
                    else
```

```
                        elements(end,5) = elements(end-1,5)+1;
                    end
                    elements(end,6) = data_rows(1,11);
                    elements(end,7) = data_rows(end,11);
                    elements(end,8) = data_rows(i+1,11);
                    elements(end,9) = 0;
                    elements(end,10) = 0;
                    elements(end,11) = 0;
                    n = n+1;
                    if n == 4
                        n = 1;
                    end
                    elements(end,12) = n;
                    elements(end,13) = 1;
                    elements(end,14) = 1;
                    elements(end,15) = 1;
                    x = x+1;

                end
            end
        end
    end
end

% Linear constrain for "Bernouilli element" - Axial force application
    node
elements(end+1,1) = elements(end,1)+1;
elements(end,2) = 221;
elements(end,3) = elements(end-1,3)+1;
elements(end,4) = 0;
if n < 3
    elements(end,5) = elements(end-1,5);
else
    elements(end,5) = elements(end-1,5)+1;
end
elements(end,6) = data_rows(1,11);
elements(end,7) = data_rows(end,11);
elements(end,8) = coord(data_rows(end,11)+1,1);
elements(end,9) = 0;
elements(end,10) = 0;
elements(end,11) = 0;
n = n+1;
if n == 4
    n = 1;
end
elements(end,12) = n;
elements(end,13) = 1;
elements(end,14) = 1;
elements(end,15) = 1;

```

```
% Linear constrains for equalization of the displacements of all
    aquivalent
% springs of row i
for i = 3:n_rows_t
    for j = data_rows_t(i,7) : data_rows_t(i,8)-1
        if elements(j,19) == 0

            elements(end+1,1) = elements(end,1)+1;
            elements(end,2) = 221;
            elements(end,3) = n_rows_tot+n_gr_tot+3;
            elements(end,4) = 0;
            if n < 3
                elements(end,5) = elements(end-1,5);
            else
                elements(end,5) = elements(end-1,5)+1;
            end
            elements(end,6) = elements(j,6);
            elements(end,7) = elements(j,5);
            elements(end,8) = elements(data_rows_t(i,8),6);
            elements(end,9) = elements(data_rows_t(i,8),6)-1;
            elements(end,10) = 0;
            elements(end,11) = 0;
            elements(end,12) = 0;
            n = n+1;
            if n == 4
                n = 1;
            end
            elements(end,13) = n;
            elements(end,14) = 1;
            elements(end,15) = 1;
            elements(end,16) = 1;
            elements(end,17) = 1;

        end
    end
end

%% Supports

nt = 0;
n = 0;
nb_col = 0;
nb_row = 1;

% Embedded supports
for i = 1:n_gr_tot
    if n <= n_rows_c
        if data_rows(i,3) == 1
            nt = nt+1;
            if nt == 1

```

```
1236                n=n+1;
1237                if nb_col == 16
1238                    nb_row = nb_row+1;
1239                    nb_col = 1;
1240                else
1241                    nb_col = nb_col+1;
1242                end
1243                supports(nb_row,1) = 1110000;
1244                supports(nb_row,2) = 0;
1245                supports(nb_row,3) = 0;
1246                supports(nb_row,3+nb_col) = data_rows(i,10);
1247
1248            else
1249                continue
1250            end
1251        else
1252
1253            n=n+1;
1254            if nb_col == 16
1255                nb_row = nb_row+1;
1256                nb_col = 1;
1257            else
1258                nb_col = nb_col+1;
1259            end
1260            supports(nb_row,1) = 1110000;
1261            supports(nb_row,2) = 0;
1262            supports(nb_row,3) = 0;
1263            supports(nb_row,3+nb_col) = data_rows(i,10);
1264
1265        end
1266    end
1267end
1268
1269nb_col = 0;
1270nb_row = nb_row+1;
1271
1272% Springs internal supports (not part of a group or an embedded support
    )
1273for i = 3:n_rows_t
1274    if i < n_rows_t
1275        for j = data_rows_t(i,7)+sum(data_rows_t(3:i-1,4))+data_rows_t(i,4)*2 :
            data_rows_t(i,8)-1
1276
1277            if nb_col == 16
1278                nb_row = nb_row+1;
1279                nb_col = 1;
1280            else
1281                nb_col = nb_col+1;
1282            end
1283            supports(nb_row,1) = 0110000;
1284            supports(nb_row,2) = 0;
```

```
1285            supports(nb_row,3) = 0;
1286            supports(nb_row,3+nb_col) = elements(j,6);
1287
1288        end
1289    end
1290        for j = 1 : i-4
1291            if i >= j+4
1292            for k = data_rows_t(i,7)+sum(data_rows_t(3:2+(j-1)))+nb_groups_zone(i,j) :
                    data_rows_t(i,7)+sum(data_rows_t(3:2+j,4))-1
1293
1294                if nb_col == 16
1295                    nb_row = nb_row+1;
1296                    nb_col = 1;
1297                else
1298                    nb_col = nb_col+1;
1299                end
1300                supports(nb_row,1) = 0110000;
1301                supports(nb_row,2) = 0;
1302                supports(nb_row,3) = 0;
1303                supports(nb_row,3+nb_col) = elements(k,6);
1304
1305            end
1306            end
1307        end
1308 end
1309
1310 % Groups supports
1311 a = 0;
1312 for i = 1:size(supports,1)
1313     for j = 4:size(supports,2)
1314         if supports(i,j) ~= 0
1315             a = a+1;
1316             nodes(a,1) = supports(i,j);
1317             nodes = sort(nodes);
1318         end
1319     end
1320 end
1321
1322 nb_col = 0;
1323 if nnz(any(supports,2)) == nb_row
1324     nb_row = nb_row+1;
1325 end
1326
1327 for i = 1:size(nodes,1)-1
1328     if nodes(i+1,1) ~= nodes(i,1)+1
1329         n = nodes(i,1)+1;
1330         while n ~= nodes(i+1,1)
1331
1332             if nb_col == 16
1333                 nb_row = nb_row+1;
1334                 nb_col = 1;
```

```
            else
                nb_col = nb_col+1;
            end
            supports(nb_row,1) = 0100000;
            supports(nb_row,2) = 0;
            supports(nb_row,3) = 0;
            supports(nb_row,3+nb_col) = n;
            n = n+1;

        end
    end
end

if nodes(end,1)+1 ~= data_rows(end,11)

    if nb_col == 16
        nb_row = nb_row+1;
        nb_col = 1;
    else
        nb_col = nb_col+1;
    end
    supports(nb_row,1) = 0100000;
    supports(nb_row,2) = 0;
    supports(nb_row,3) = 0;
    supports(nb_row,3+nb_col) = data_rows(end,11)-1;

    if nb_col == 16
        nb_row = nb_row+1;
        nb_col = 1;
    else
        nb_col = nb_col+1;
    end
    supports(nb_row,1) = 0100000;
    supports(nb_row,2) = 0;
    supports(nb_row,3) = 0;
    supports(nb_row,3+nb_col) = data_rows(end,11);

else

    if nb_col == 16
        nb_row = nb_row+1;
        nb_col = 1;
    else
        nb_col = nb_col+1;
    end
    supports(nb_row,1) = 0100000;
    supports(nb_row,2) = 0;
    supports(nb_row,3) = 0;
    supports(nb_row,3+nb_col) = data_rows(end,11);

end
```

```
1386
1387 % Point of application of the axial load
1388 if nb_col == 16
1389     nb_row = nb_row+1;
1390     nb_col = 1;
1391 else
1392     nb_col = nb_col+1;
1393 end
1394 supports(nb_row,1) = 0100000;
1395 supports(nb_row,2) = 0;
1396 supports(nb_row,3) = 0;
1397 supports(nb_row,3+nb_col) = data_rows(end,11)+1;
1398
1399
1400 %% Sequences
1401
1402 % Number of sequences
1403 if M == 0
1404     % N only
1405     nb_seq = 1;
1406 else
1407     % M only and other MN
1408     nb_seq = 4;
1409 end
1410
1411 % FINELG input
1412 for i = 1:nb_seq
1413     sequences(i,1) = 1;
1414     sequences(i,2) = 1;
1415     for j = 3:6
1416         if j-2 == i
1417             sequences(i,j) = 1;
1418         else
1419             sequences(i,j) = 0;
1420         end
1421     end
1422 end
1423
1424
1425 %% Loading
1426
1427 % Number of loads
1428 if M == 0
1429     % N only
1430     nb_loads = 1;
1431 else
1432     % M only and other MN
1433     nb_loads = 7;
1434 end
1435
1436 % FINELG input
```

```
for i = 1:nb_loads

    loading(i,1) = i;
    loading(i,2) = 0;
    loading(i,4) = 0;

end
for i = nb_loads+1:nb_loads*2

    loading(i,2) = 1;
    loading(i,3) = i-nb_loads;
    loading(i,4) = 0;

end

if M == 0

    loading(nb_loads,3) = N;
    loading(nb_loads*2,1) = 1;
    loading(nb_loads*2,5) = data_rows(end,11)+1;

else
    loading(1,3) = N1;
    if M < 0
        loading(2,3) = M1/data_rows(1,2);
        loading(3,3) = -M1/data_rows(1,2);
    elseif M > 0
        loading(2,3) = -M1/data_rows(1,2);
        loading(3,3) = M1/data_rows(1,2);
    end
    loading(4,3) = -N1;
    loading(5,3) = -M/data_rows(1,2);
    loading(6,3) = M/data_rows(1,2);
    loading(7,3) = N;

    loading(8,1) = 1;
    loading(9,1) = 2;
    loading(10,1) = 2;
    loading(11,1) = 3;
    loading(12,1) = 4;
    loading(13,1) = 4;
    loading(14,1) = 4;

    loading(8,5) = data_rows(end,11)+1;
    loading(9,5) = data_rows(1,11);
    loading(10,5) = data_rows(end,11);
    loading(11,5) = data_rows(end,11)+1;
    loading(12,5) = data_rows(1,11);
    loading(13,5) = data_rows(end,11);
    loading(14,5) = data_rows(end,11)+1;
end
```

```
1488
1489
1490  %% Control
1491
1492  control(1,1) = 0;
1493  control(1,2) = 0;
1494  control(1,3) = 0;
1495  control(1,4) = 23;
1496  control(1,5) = 1200060;
1497  control(1,6) = 0;
1498  control(1,7) = 1;
1499  control(1,8) = 0;
1500  control(1,9) = 0;
1501  control(1,10) = 0;
1502  control(1,11) = 0;
1503
1504  for i = 2:1+nb_seq
1505
1506      control(i,1) = 0;
1507      control(i,2) = 0;
1508      control(i,3) = 500;
1509      control(i,4) = 1;
1510      control(i,5) = 99999999;
1511
1512  end
1513
1514  control(end,1) = 1;
1515
1516  %% Data writing
1517
1518  FileName = (fullfile([Name, Ext]));
1519  nb_files = 0;
1520
1521  if  exist(FileName,'file')
1522
1523      Dir = dir(fullfile([Name, '*', Ext]));
1524      nb_files = size(Dir,1);
1525      FileName = fullfile([Name, sprintf('%d', nb_files), Ext]);
1526
1527  end
1528
1529  fid = fopen(FileName,'w');
1530
1531  % Version FINELG
1532  fprintf(fid,'%s%6s%8s\n','FINELG','103','12');
1533
1534  % Control
1535  fprintf(fid,'%s\n','CTRL');
1536  fprintf(fid,'%4s%4s\n','N','MM');
1537  for i = 1:3
1538      fprintf(fid,'%4.0f',control(1,i));
```

```
1539 end
1540 for i = 4:5
1541     fprintf(fid,'%8.0f',control(1,i));
1542 end
1543 for i = 6:size(control,2)
1544     fprintf(fid,'%4.0f',control(1,i));
1545 end
1546 fprintf(fid,'\n%8s\n','NONL');
1547 for i = 2:size(control,1)
1548     fprintf(fid,'%8s%4.0f\n','SEQP',i-1);
1549     fprintf(fid,'%4.0f%4.0f%4.0f%24.0f%12.0f\n\n',control(i,1:5));
1550 end
1551 fprintf(fid,'%s\n','CTRL_END');
1552
1553 % Loading sequences
1554 for i = 1:size(sequences,1)-1
1555     fprintf(fid,'%s%8.0f\n','SEQP',i);
1556     fprintf(fid,'%8s\n%4.0f%12.3f%8.3f%8.3f%8.3f%8.3f\n','COMB',sequences(i,1:6));
1557     fprintf(fid,'%8s\n%4.0f%4.0f\n','INCR',110,-4);
1558     fprintf(fid,'%8s\n%8.3f\n','CREM',0.1);
1559     fprintf(fid,'%8s\n%4.0f%4.0f%4.0f\n','MOPA',1,60,-4);
1560     fprintf(fid,'%8s\n%4.0f\n%4.0f\n','NODC',data_rows(end,11)+1,1);
1561     fprintf(fid,'%s%4.0f\n','SEQP_END',i);
1562 end
1563 fprintf(fid,'%s%8.0f\n','SEQP',nb_seq);
1564 fprintf(fid,'%8s\n%4.0f%12.3f%8.3f%8.3f%8.3f%8.3f\n','COMB',sequences(end,1:6));
1565 fprintf(fid,'%8s\n%4.0f%4.0f%4.0f\n','INCR',-4,139,4);
1566 fprintf(fid,'%8s\n%8.3f\n','CREM',incr_ini);
1567 fprintf(fid,'%8s\n%4.0f%4.0f%4.0f\n','MOPA',1,60,-4);
1568 fprintf(fid,'%8s\n%4.0f\n%4.0f\n','NODC',data_rows(end,11)+1,1);
1569 fprintf(fid,'%s%4.0f\n','SEQP_END',nb_seq);
1570
1571 % Meca
1572 fprintf(fid,'%s','MECA');
1573 for i = 1:n_rows_tot+n_gr_tot
1574
1575     nb_lines = ceil((nnz(any(meca(i,3:end),1))+1)/6);
1576     nb_terms = -((nb_lines-1)*8-nnz(any(meca(i,:),1))+1);
1577
1578     if nnz(any(meca(i,:),1))+1 <= 8
1579         fprintf(fid,'\n%4.0f%4.0f%12.2f%12.8f%12.2f%12.8f%12.2f%12.8f',meca(i,1:nnz(any(
                meca(i,:),1))+1));
1580     else
1581         fprintf(fid,'\n%4.0f%4.0f%12.2f%12.8f%12.2f%12.8f%12.2f%12.8f',meca(i,1:8));
1582         for j = 2:nb_lines-1
1583             fprintf(fid,'\n%4.0f%16.2f%12.8f%12.2f%12.8f%12.2f%12.8f',-meca(i,1),meca(i
                    ,(8+(j-2)*6+1):(8+(j-1)*6)));
1584         end
1585         fprintf(fid,'\n%4.0f%16.2f%12.8f%12.2f%12.8f%12.2f%12.8f',-meca(i,1),meca(i,(8+(
                nb_lines-2)*6+1):nnz(any(meca(i,:),1))+1));
1586     end
```

```
1587 end
1588 for i = n_rows_tot+n_gr_tot+1:size(meca,1)
1589     fprintf(fid,'\n%4.0f%4.0f%12.3f%12.3f%12.3f%12.3f',meca(i,1:nnz(any(meca(i,:),1))+1));
1590 end
1591 fprintf(fid,'\n%s\n','MECA_END');
1592
1593 % Geometry
1594 fprintf(fid,'%s\n','GEOM');
1595 fprintf(fid,'%4.0f%4.0f%12.2f%12.0f\n',1,31,90,67500);
1596 fprintf(fid,'%s\n','GEOM_END');
1597
1598 % Coordinates
1599 fprintf(fid,'%s\n','COOR');
1600 for i = 1:data_rows(end,11)+1
1601     fprintf(fid,'%4.0f%4.0f%12.3f%12.3f%12.3f\n',coord(i,1:5));
1602 end
1603 for i = data_rows(end,11)+2:size(coord,1)
1604     fprintf(fid,'%4.0f\n',coord(i,1));
1605 end
1606
1607 % Elements
1608 fprintf(fid,'%s\n','ELEM');
1609 for i = 1:data_rows(end,8)+n_rows_c
1610     if i == 1 && elements(i,18) ~= 0
1611         fprintf(fid,'%4.0f%4.0f%4.0f%4.0f%4.0f%4.0f%44s%5.0f\n',elements(i,1:6),'S',
            elements(i,18));
1612     elseif i == 1 && elements(i,18) == 0
1613         fprintf(fid,'%4.0f%4.0f%4.0f%4.0f%4.0f%4.0f%44s\n',elements(i,1:6),'S');
1614     elseif elements(i,18) == 0
1615         fprintf(fid,'%4.0f%4.0f%4.0f%4.0f%4.0f%4.0f\n',elements(i,1:6));
1616     else
1617         fprintf(fid,'%4.0f%4.0f%4.0f%4.0f%4.0f%4.0f%49.0f\n',elements(i,1:6),elements(i,18)
            );
1618     end
1619 end
1620 for i = data_rows(end,8)+n_rows_c+1:data_rows(end,8)+n_rows_c+data_rows_t(1,4)
1621     for j = 1:12
1622         fprintf(fid,'%4.0f',elements(i,j));
1623     end
1624     fprintf(fid,'%4.0f\n',elements(i,13));
1625 end
1626 for i = data_rows(end,8)+n_rows_c+data_rows_t(1,4)+1:data_rows(end,8)+n_rows_c+
    data_rows_t(1,4)+data_rows_t(2,4)+sum(lin_const_int)
1627     for j = 1:16
1628         fprintf(fid,'%4.0f',elements(i,j));
1629     end
1630     fprintf(fid,'%3.0f\n%20.0f\n',elements(i,17),1);
1631 end
1632 if data_rows(end,3) == 0
1633 for i = data_rows(end,8)+n_rows_c+data_rows_t(1,4)+data_rows_t(2,4)+sum(
    lin_const_int)+1:data_rows(end,8)+n_rows_c+data_rows_t(1,4)+data_rows_t(2,4)+
```

```
    sum(lin_const_int)+n_rows_c+1
1634    for j = 1:14
1635        fprintf(fid,'%4.0f',elements(i,j));
1636    end
1637    fprintf(fid,'%4.0f\n',elements(i,15));
1638 end
1639 for i = data_rows(end,8)+n_rows_c+data_rows_t(1,4)+data_rows_t(2,4)+sum(
    lin_const_int)+n_rows_c+2:size(elements,1)
1640    for j = 1:16
1641        fprintf(fid,'%4.0f',elements(i,j));
1642    end
1643    fprintf(fid,'%3.0f\n%20.0f\n',elements(i,17),1);
1644 end
1645 else
1646 for i = data_rows(end,8)+n_rows_c+data_rows_t(1,4)+data_rows_t(2,4)+sum(
    lin_const_int)+1:data_rows(end,8)+n_rows_c+data_rows_t(1,4)+data_rows_t(2,4)+
    sum(lin_const_int)+n_rows_c+2
1647    for j = 1:14
1648        fprintf(fid,'%4.0f',elements(i,j));
1649    end
1650    fprintf(fid,'%4.0f\n',elements(i,15));
1651 end
1652 for i = data_rows(end,8)+n_rows_c+data_rows_t(1,4)+data_rows_t(2,4)+sum(
    lin_const_int)+n_rows_c+3:size(elements,1)
1653    for j = 1:16
1654        fprintf(fid,'%4.0f',elements(i,j));
1655    end
1656    fprintf(fid,'%3.0f\n%20.0f\n',elements(i,17),1);
1657 end
1658 end
1659
1660 fprintf(fid,'%s\n','RENU');
1661 fprintf(fid,'%s\n','ELEM_END');
1662
1663 % Loading
1664 fprintf(fid,'%s\n','CHAR');
1665 for i = 1:nb_loads
1666    fprintf(fid,'%4.0f%4.0f%12.0f%12.0f\n',loading(i,1:4));
1667 end
1668 fprintf(fid,'%7s\n','CAS');
1669 for i = nb_loads+1:size(loading,1)
1670    for j = 1:size(loading,2)-1
1671        fprintf(fid,'%4.0f',loading(i,j));
1672    end
1673    fprintf(fid,'%4.0f\n',loading(i,5));
1674 end
1675 fprintf(fid,'%s\n','CHAR_END');
1676
1677 % Supports
1678 fprintf(fid,'%8s\n','APPU');
1679 for i = 1:size(supports,1)
```

```
1680      fprintf(fid,'%8.0f',supports(i,1));
1681      for j = 2:size(supports,2)-1
1682          fprintf(fid,'%4.0f',supports(i,j));
1683      end
1684      fprintf(fid,'%4.0f\n',supports(i,size(supports,2)));
1685  end
1686  fprintf(fid,'%s\n','END');
1687
1688  fclose(fid);
1689
1690
1691  if nb_files > 0
1692
1693      FileName_txt = fullfile([Name_txt, sprintf('%d', nb_files), Ext_txt]);
1694
1695  else
1696
1697      FileName_txt = (fullfile([Name_txt, Ext_txt]));
1698
1699  end
1700
1701  % Text file for the verification of the displacement + loads in results
        processing
1702
1703  fid = fopen(FileName_txt,'w');
1704
1705  fprintf(fid,'%20.3f%20.3f%20.3f\n',M,N,V);
1706  for i = 1:size(Delta_max,1)
1707      fprintf(fid,'%20.6f%20.0f\n',Delta_max(i,1),Delta_max(i,2));
1708  end
1709
1710  fclose(fid);
```

Results processing

```
% Post treatment of FINELG output data - Ductility considered

% Conducted as part of master thesis relating to the development of an
% innovative model for the precise characterization of steel and
   composite
% joints

% MATHIEU Julien

close all
clear
clc

%% Results extraction

% Extraction of shear resistance data from excel file
data_all = xlsread('Data.xlsx');

% First line/column of shear resistance table
First_line_table = 2;
First_column_table = 16;

% Number of rows in tension/compression
n_rows_t = data_all(10,1);
n_rows_c = data_all(11,1);
n_rows_tot = n_rows_t+n_rows_c;

% Shear resistance data
for i = First_line_table : First_line_table+n_rows_t-1

      shear_resistance(i-(First_line_table-1),1) = data_all(i,First_column_table
          +1);
      shear_resistance(i-(First_line_table-1),2) = data_all(i,First_column_table
          +2);
      shear_resistance(i-(First_line_table-1),3) = data_all(i,First_column_table
          +4);

end

% Name of .TPS file
Name = 'Input_Finelg';
Ext = '.TPS';
FileName = (fullfile([Name, Ext]));

% Name of loads/max displacment file
Name_txt = 'Loads_Dep';
Ext_txt = '.txt';
FileName_txt = (fullfile([Name_txt, Ext_txt]));

```

```
% Name of displacements file
Name_Delta = 'Input_Finelg';
Ext_Delta = '.DE3';
FileName_Delta = (fullfile([Name_Delta, Ext_Delta]));


%% MN diagram

% Number of .TPS/loads files
if exist(FileName,'file')

    Dir = dir(fullfile([Name, '*', Ext]));
    nb_files = size(Dir,1);

else

    nb_files = 1;

end

% Values of final load multipliers
for i = 1:nb_files

    % Name of output files
    if i > 1

        FileName = fullfile([Name, sprintf('%d', i-1), Ext]);
        FileName_txt = fullfile([Name_txt, sprintf('%d', i-1), Ext_txt]);
        FileName_Delta = fullfile([Name, sprintf('%d', i-1), Ext_Delta]);

    end

    % Extraction of results
    % TPS
    fid = fopen(FileName,'r');

    Results = textscan(fid,repmat('%s',[1,25]),'Delimiter','*');

    fclose(fid);

    % Loads and displacement
    fid = fopen(FileName_txt,'r');

    Loads_Delta = textscan(fid,'%f%f%f');

    fclose(fid);

    % Convertion to numbers
    Mult = Results{1,4};
    Ajust = Results{1,3};
    Step = Results{1,2};
```

```
97
98      Ajust(cellfun(@isempty,Ajust)) = [];
99      Mult(cellfun(@isempty,Mult)) = [];
100     Step(cellfun(@isempty,Step)) = [];
101
102     Ajust = str2double(Ajust(2:end,:));
103     Mult = cell2mat(Mult(2:end,:));
104     Increment = Mult(:,7:end-2);
105     Mult = Mult(:,1:end-9);
106     Step = str2double(Step(2:end,:));
107
108     for j = 1:size(Mult,1)
109
110         Mult_all(j,i) = str2double(sscanf(Mult(j,:),'%s'));
111
112     end
113
114     Loads_Delta = cell2mat(Loads_Delta);
115     % Imposed loads
116     Loads_all(i,1) = Loads_Delta(1,1);
117     Loads_all(i,2) = Loads_Delta(1,2);
118     Loads_all(i,3) = Loads_Delta(1,3);
119
120     % Maximum displacement
121     Delta_max = Loads_Delta(2:end,1:2);
122
123     % Extraction of displacement of all nodes at each step
124     [DEP,NENR,LIB,NOMAX,FREQ] = lect_DE3(FileName_Delta,'C:');
125
126     n = 1;
127     Delta_all = zeros(size(DEP,1)/3,size(DEP,2));
128
129     for j = 1:size(DEP,1)
130         if j == 3*(n-1)+1
131
132             Delta_all(n,:) = DEP(j,:);
133             n = n+1;
134
135         end
136     end
137
138     % Displacement of each component located in the last zone
139     Delta_spring = zeros(size(Delta_max,1),size(Delta_all,2));
140     for j = 1:size(Delta_max,1)
141         for k = 1:size(Delta_all,1)
142             if k == Delta_max(j,2)
143                 Delta_spring(j,:) = Delta_all(k,:)-Delta_all(k-1,:);
144             end
145         end
146     end
147
```

```
148     % Negative displacement in components in tension = 0
149     % Positive displacement in components in compression = 0
150     for j = 1:size(Delta_spring,1)
151         for k = 1:size(Delta_spring,2)
152             if Delta_max(j,1) < 0 && Delta_spring(j,k) > 0
153 
154                 Delta_spring(j,k) = 0;
155 
156             end
157             if Delta_max(j,1) > 0 && Delta_spring(j,k) < 0
158 
159                 Delta_spring(j,k) = 0;
160 
161             end
162         end
163     end
164 
165     % Verification of the displacements
166      n_steps = nnz(any(Mult_all(:,i),2));
167     leave = 0;
168     for j = 1:size(Delta_spring,2)
169         for k = 1:size(Delta_spring,1)
170             if abs(Delta_spring(k,j)) > abs(Delta_max(k,1))
171 
172                 n_steps = nnz(any(Mult_all(:,i),2))-(size(Delta_all,2)-(j-1));
173                 n_steps_delta = j-1;
174                 leave = 1;
175 
176             end
177             if leave == 1
178                 break
179             end
180         end
181         if leave == 1
182             break
183         end
184     end
185 
186     % Value of the load multiplier
187 
188     Multip(i,1) = Mult_all(n_steps,i);
189 
190     % Value of bending moment and axial force at failure
191     MN(i,1:2) = Loads_all(i,1:2)*Multip(i,1);
192 
193     % Final step number
194 
195     fprintf('%s %s %s %d %s\n','Final step of',FileName,'=',Step(end,1),'(max = 40)')
196     fprintf('%s %s %s %s\n','Final increment of',FileName,'=',Increment(end,:))
197 
198     % Shear resistance
```

```
199     n = 0;
200     if n_steps == nnz(any(Mult_all(:,i),2))
201         for j = 1:n_rows_tot
202             if Delta_max(j,1) < 0 && Delta_spring(j,end) <= 0
203                 n = n+1;
204             elseif Delta_max(j,1) >= 0 && Delta_spring(j,end) == 0
205 
206                 shear_resistance_bolts(j-n,1) = shear_resistance(j-n,1);
207 
208             elseif Delta_max(j,1) >= 0 && Delta_spring(j,end) > 0
209 
210                 shear_resistance_bolts(j-n,1) = shear_resistance(j-n,1)*0.29;
211 
212             end
213         end
214     else
215         for j = 1:n_rows_tot
216             if Delta_max(j,1) < 0 && Delta_spring(j,n_steps_delta) <= 0
217                 n = n+1;
218                 continue
219             elseif Delta_max(j,1) >= 0 && Delta_spring(j,n_steps_delta) == 0
220 
221                 shear_resistance_bolts(j-n,1) = shear_resistance(j-n,1);
222 
223             elseif Delta_max(j,1) >= 0 && Delta_spring(j,n_steps_delta) > 0
224 
225                 shear_resistance_bolts(j-n,1) = shear_resistance(j-n,1)*0.29;
226 
227             end
228         end
229     end
230 
231     for j = 1:n_rows_t
232         if shear_resistance_bolts(j,1) <= shear_resistance(j,2)
233             resistance_bolts(j,1) = shear_resistance_bolts(j,1);
234         else
235             resistance_bolts(j,1) = shear_resistance(j,2);
236         end
237     end
238 
239     Total_resistance = sum(resistance_bolts);
240 
241     for j = 1:sum(not(isnan(shear_resistance(:,3))))
242         if Total_resistance >= shear_resistance(j,3)
243 
244             Total_resistance = shear_resistance(j,3);
245 
246         else
247             continue
248         end
249     end
```

```
250
251     if Loads_Delta(1,3) <= Total_resistance
252
253         fprintf('%s %d%s %s %d%s %s\n\n','Shear resistance :',Loads_Delta(1,3),'N','<=',
                Total_resistance,'N','OK')
254
255     else
256
257         fprintf('%s %d%s %s %d%s %s\n\n','Shear resistance :',Loads_Delta(1,3),'N','>',
                Total_resistance,'N','!NOT OK!')
258
259     end
260 end
261
262 % Values of the four peaks of the MN diagram
263 [Mpos_max,loc(1,1)] = max(MN(:,1));
264 [Npos_max,loc(2,1)] = max(MN(:,2));
265 [Mneg_max,loc(3,1)] = min(MN(:,1));
266 [Nneg_max,loc(4,1)] = min(MN(:,2));
267
268 MN1_max = MN(loc(1,1),:);
269 MN2_max = MN(loc(2,1),:);
270 MN3_max = MN(loc(3,1),:);
271 MN4_max = MN(loc(4,1),:);
272
273
274 % Division of the MN diagram in four zones
275 n1 = 0;
276 n2 = 0;
277 n3 = 0;
278 n4 = 0;
279 for i = 1:size(MN,1)
280     if MN(i,1) <= MN1_max(1,1) && MN(i,2) >= MN1_max(1,2) && MN(i,1) > MN2_max(1,1) &&
            MN(i,2) < MN2_max(1,2)
281
282         n1 = n1+1;
283         M1(n1,1) = MN(i,1)/10^6;
284         N1(n1,1) = MN(i,2)/10^3;
285
286     elseif MN(i,1) <= MN2_max(1,1) && MN(i,2) <= MN2_max(1,2) && MN(i,1) > MN3_max(1,1)
            && MN(i,2) > MN3_max(1,2)
287
288         n2 = n2+1;
289         M2(n2,1) = MN(i,1)/10^6;
290         N2(n2,1) = MN(i,2)/10^3;
291
292     elseif MN(i,1) >= MN3_max(1,1) && MN(i,2) <= MN3_max(1,2) && MN(i,1) < MN4_max(1,1)
            && MN(i,2) > MN4_max(1,2)
293
294         n3 = n3+1;
295         M3(n3,1) = MN(i,1)/10^6;
```

```
296         N3(n3,1) = MN(i,2)/10^3;
297
298     elseif MN(i,1) >= MN4_max(1,1) && MN(i,2) >= MN4_max(1,2) && MN(i,1) < MN1_max(1,1)
            && MN(i,2) < MN1_max(1,2)
299
300         n4 = n4+1;
301         M4(n4,1) = MN(i,1)/10^6;
302         N4(n4,1) = MN(i,2)/10^3;
303
304     end
305 end
306
307 % Sorting of the MN values in the considered zone
308 M1 = sort(M1,'descend');
309 N1 = sort(N1,'ascend');
310 M2 = sort(M2,'descend');
311 N2 = sort(N2,'descend');
312 M3 = sort(M3,'ascend');
313 N3 = sort(N3,'descend');
314 M4 = sort(M4,'ascend');
315 N4 = sort(N4,'ascend');
316
317 % Plot
318 M = [M1' M2' M3' M4' M1(1,1) '];
319 N = [N1' N2' N3' N4' N1(1,1) '];
320
321 plot(M,N,'red')
322 hold on
323 Analytique;
324 grid on
```

To not consider ductility, lines 167 to 184 must be deleted.