

Advanced Matrix Structural Analysis
Final project report

Contents

1	Program without distributed load	2
1.1	My flowchart	2
1.2	main program	3
1.3	subroutine estiff	6
1.4	subroutine etran	7
1.5	The verification problem	8
1.5.1	My node and element definition for the verification problem	8
1.5.2	The full results for the verification problem	9
1.6	The question in the verification problem	10
1.6.1	Deflections at points c and d	10
1.6.2	Reactions at points a and b	10
1.6.3	Axial forces in members a-c, c-b, and c-d	10
2	Program with distributed load	11
2.1	main program	11
2.2	Problem 1 to verify the main program with distributed load	15
2.3	Problem 2 to verify the main program with distributed load	16

1 Program without distributed load

1.1 My flowchart

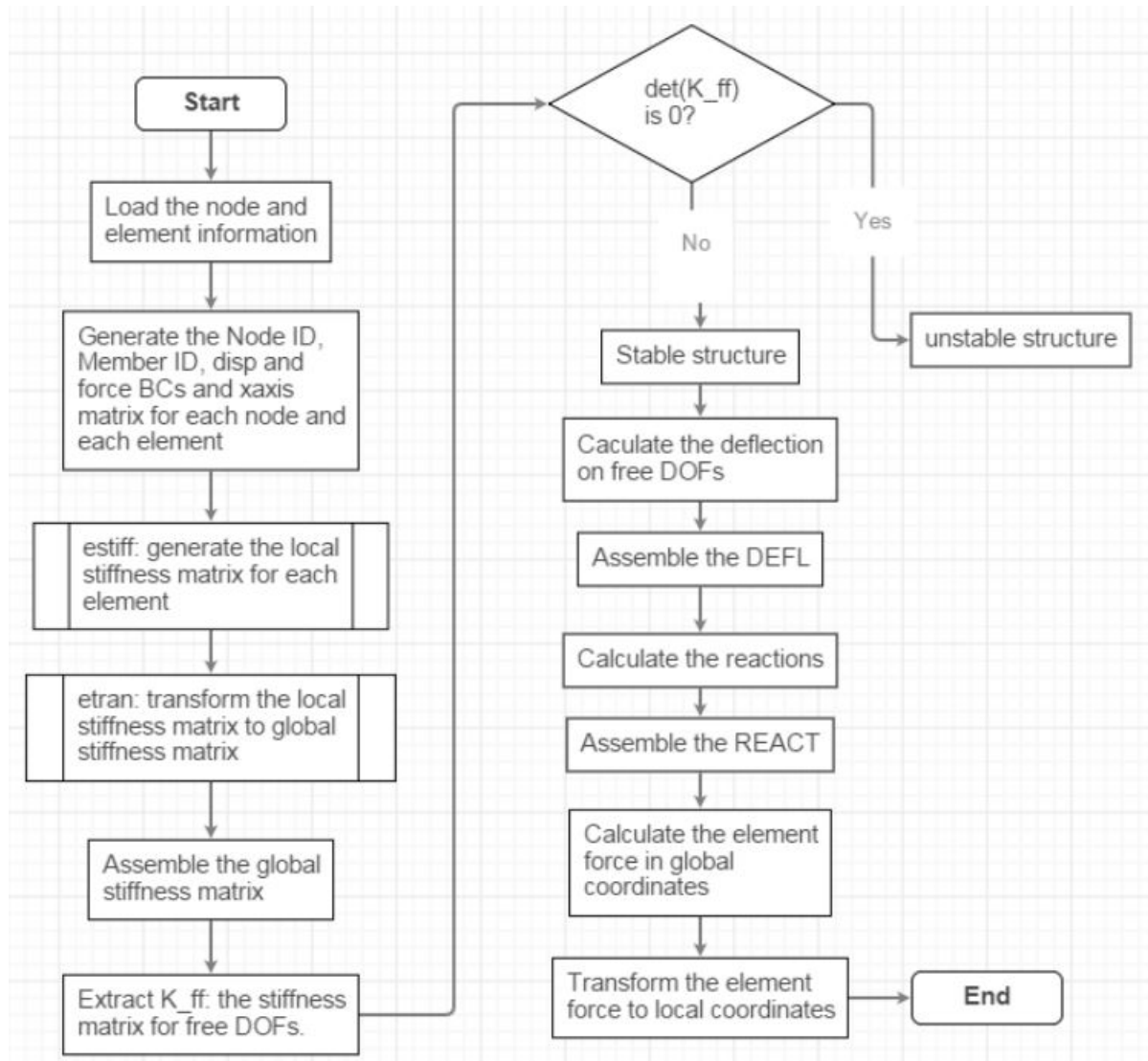


Figure 1: Flowchart

1.2 main program

```

1 clear
2 clc
3 % load the input information.
4 load nnodes.txt;
5 load coord.txt;
6 load concen.txt;
7 load fixity.txt;
8 load nele.txt;
9 load ends.txt;
10 load Area.txt;
11 load Izz.txt;
12 load Iyy.txt;
13 load J.txt;
14 load E.txt;
15 load v.txt;
16 load beta_ang.txt;
17
18 % Generate the node id.
19 % Each line represents the 6 dofs for one node.
20 node_id=zeros(nnodes,6);
21 for i=1:nnodes
22     for j=1:6
23         node_id(i,j)=(i-1)*6+j;
24     end
25 end
26
27 % Generate the member id.
28 % Each line represents the 12 DOFs for one element.
29 mem_id=zeros(nele,12);
30 for i=1:nele
31     mem_id(i,1:6)=node_id(ends(i,1),1:6);
32     mem_id(i,7:12)=node_id(ends(i,2),1:6);
33 end
34
35 % Generate the displacement boundary conditions.
36 % Each line in fixity represents the disp BC on 6 DOFs of one node.
37 % 0 represents that the DOF is fixed.
38 % NaN represents that the DOF is free.
39 fixity_tran=fixity';
40 D=fixity_tran(:);
41 free_dof=find(isnan(D));
42 fixed_dof=find(D==0);
43
44
45 % Generate the force boundary conditions.
46 % Each line in concen represents the force BC on 6 DOFs of one node.
47 % 0 represents that the DOF does not has external force.
48 % The number represents that the force values on the DOF.
49 concen_tran=concen';
50 P_total=concen_tran(:);
51 P_free=P_total(free_dof);
52

```

```

53 % Generate the xaxis.
54 % xaxis will be used in the rotation matrix.
55 % The rotation matrix is to transform the stiffness matrix
56 % from global to local coordinate system.
57 for i=1:nele
58     xaxis(i,:)=(coord(ends(i,2),:)-coord(ends(i,1),:));
59     Length(i) = norm(xaxis(i,:));
60     xaxisunit(i,:)=xaxis(i,+)/Length(i) ;
61 end
62
63 % Generate the local stiffness matrix for each element.
64 k_stack_local=zeros(nele,12,12);
65 gamma_stack_local=zeros(nele,12,12);
66 for i=1:nele
67     k_stack_local(i,:,:)=estiff(Area(i),Izz(i),Iyy(i),J(i),E(i),v(i),Length(i));
68     gamma_stack_local(i,:,:)=etran(beta_ang(i),xaxisunit(i,1:3));
69 end
70
71 % Transform the local stiffness matrix to global stiffness matrix.
72 k_stack_global=zeros(nele,12,12);
73 for i=1:nele
74     k_stack_global(i,:,:)=(squeeze(gamma_stack_local(i,:,:))'')...
75         *squeeze(k_stack_local(i,:,:))...
76         *squeeze(gamma_stack_local(i,:,:));
77 end
78
79 % Assemble the global stiffness matrix together.
80 ndof=6*nnodes;
81 k_total=zeros(ndof,ndof);
82 for i=1:nele
83     k_total(mem_id(i,1:6),mem_id(i,1:6))=...
84         k_total(mem_id(i,1:6),mem_id(i,1:6))+squeeze(k_stack_global(i,1:6,1:6));
85     k_total(mem_id(i,1:6),mem_id(i,7:12))=...
86         k_total(mem_id(i,1:6),mem_id(i,7:12))+squeeze(k_stack_global(i,1:6,7:12));
87     k_total(mem_id(i,7:12),mem_id(i,1:6))=...
88         k_total(mem_id(i,7:12),mem_id(i,1:6))+squeeze(k_stack_global(i,7:12,1:6));
89     k_total(mem_id(i,7:12),mem_id(i,7:12))=...
90         k_total(mem_id(i,7:12),mem_id(i,7:12))+squeeze(k_stack_global(i,7:12,7:12));
91 end
92
93 % Extract the K_ff.
94 K_ff=k_total(free_dof,free_dof);
95 K_sf=k_total(fixed_dof,free_dof);
96
97
98 % C = rcond(A) returns an estimate for the reciprocal condition of A in 1-norm.
99 % If A is well conditioned, rcond(A) is near 1.0.
100 % If A is badly conditioned, rcond(A) is near 0.
101 if abs(det(K_ff))<(10^(-15))
102     AFLAG=0;
103     display('Unstable Structure. ');
104     display('Maybe you should fix more DOFs . ');
105 else
106     AFLAG=1;
107     display('The structure is stable. ');

```

```

108 end
109
110 if abs(rcond(K_ff))<(10^(-15))
111     BFLAG=0;
112     display('The matrix is badly conditioned. ');
113     display('The results may have a large error. ');
114     display('Some element may have a way bigger stiffness than others. ');
115     display('Maybe you need to check the units of element properties. ');
116 else
117     BFLAG=1;
118     display('The matrix is well-conditioned. ');
119 end
120
121 defl=K_ff\P_free;
122
123 % Put the deflection to its appropriate location.
124 % Include the fixed DOF.
125 defl_vector_total=zeros(ndof,1);
126 defl_vector_total(free_dof)=defl;
127
128 % Each line in DEFL' represents the 6 DOFs at each node.
129 DEFL=reshape(defl_vector_total,6,nnodes);
130 DEFL=DEFL';
131 display(DEFL)
132
133 % Generate the reaction.
134 react=K_sf*defl;
135
136 react_vector_total=zeros(ndof,1);
137 react_vector_total(fixed_dof)=react;
138 REACT=reshape(react_vector_total,6,nnodes);
139 REACT=REACT';
140 display(REACT)
141
142 % Calculate the element force.
143 % Transform the element from from global to local coordiante system.
144 ELE_FOR=zeros(nele,12);
145 for i=1:nele
146     element_disp_global=zeros(1,12);
147     element_disp_global(1:6)=DEFL(ends(i,1),1:6);
148     element_disp_global(7:12)=DEFL(ends(i,2),1:6);
149     element_disp_global=element_disp_global';
150     element_force_local=squeeze(k_stack_local(i,:,:))...
151         *squeeze(gamma_stack_local(i,:,:))*element_disp_global;
152     ELE_FOR(i,:)=element_force_local(:);
153 end
154 display(ELE_FOR)

```

1.3 subroutine estiff

This subroutine will generate the local stiffness matrix for each element. The input arguments is listed in the source code comments. The output is the stiffness matrix for this element in local coordinate system.

```

1 function [stiffmatrix] = estiff(A,Izz,Iyy,J,E,v,Length)
2     % A      -- Area
3     % Izz    -- Second moment of area over axis zz
4     % Iyy    -- Second moment of area over axis yy
5     % J      -- torsion constant
6     % E      -- Elastic modulus
7     % v      -- Poisson's ratio
8     % Length -- Element Length
9
10    % write 4 (2 by 2) small matrix first and then put them together.
11    KK1=diag([A/Length, 12*Izz/Length^3, 12*Iyy/Length^3, J/(2*(1+v)*Length), 4*Iyy/Length,
12             4*Izz/Length]);
13    KK1(2,6)=6*Izz/Length^2;
14    KK1(6,2)=KK1(2,6);
15    KK1(3,5)=-6*Iyy/Length^2;
16    KK1(5,3)=KK1(3,5);
17
18    KK2=diag([-A/Length, -12*Izz/Length^3, -12*Iyy/Length^3, -J/(2*(1+v)*Length), 2*Iyy/Length,
19             2*Izz/Length]);
20    KK2(2,6)=6*Izz/Length^2;
21    KK2(6,2)=-KK2(2,6);
22    KK2(3,5)=-6*Iyy/Length^2;
23    KK2(5,3)=6*Iyy/Length^2;
24
25    KK3=diag([-A/Length, -12*Izz/Length^3, -12*Iyy/Length^3, -J/(2*(1+v)*Length), 2*Iyy/Length,
26             2*Izz/Length]);
27    KK3(2,6)=-6*Izz/Length^2;
28    KK3(6,2)=6*Izz/Length^2;
29    KK3(3,5)=6*Iyy/Length^2;
30    KK3(5,3)=-6*Iyy/Length^2;
31
32    KK4=diag([A/Length, 12*Izz/Length^3, 12*Iyy/Length^3, J/(2*(1+v)*Length), 4*Iyy/Length,
33             4*Izz/Length]);
34    KK4(2,6)=-6*Izz/Length^2;
35    KK4(6,2)=KK4(2,6);
36    KK4(3,5)=6*Iyy/Length^2;
37    KK4(5,3)=KK4(3,5);
38
39    % Put them together.
40    stiffmatrix=E*[KK1 KK2;KK3 KK4];
41
42 end

```

1.4 subroutine etran

The beta angle is the rotation angle over the x-axis.

And xaxis is the vector to represent the xaxis in the global coordinate system.

The output is the 12 by 12 transform matrix.

```

1 function [gamma] = etran(beta_ang,xaxis)
2     % Make the transformation matrix by two steps.
3     % Step 1: take xaxis by matrix StepOne.
4     % Step 2: rotate xaxis by matrix StepTwo.
5
6     % Initialize the matrix StepOne and StepTwo.
7     StepOne=zeros(3);
8     StepTwo=zeros(3);
9
10    % StepOne starts!!!
11    % Deal with the special case when xaxis is parallel to y direction.
12    if (xaxis(1)==0 & xaxis(3)==0)
13        StepOne(1,:)=xaxis;
14        StepOne(2,:)=[-1,0,0];
15        StepOne(3,:)=cross(StepOne(1,:),StepOne(2,:));
16
17    % Deal with the normal case
18    else
19        StepOne(1,:)=xaxis;
20        % Change to unit vector
21        StepOne(1,:)=StepOne(1,:)/norm(StepOne(1,:));
22        yaxis=[0 1 0];
23        StepOne(3,:)=cross(StepOne(1,:),yaxis) ;
24        StepOne(3,:)=StepOne(3,:)/norm(StepOne(3,:));
25        StepOne(2,:)=cross(StepOne(3,:),StepOne(1,:));
26        StepOne(2,:)=StepOne(2,:)/norm(StepOne(2,:));
27    end
28    % % StepTwo starts!!!
29    StepTwo(1,1)=1;
30    StepTwo(2,2)=cos(beta_ang);
31    StepTwo(3,3)=cos(beta_ang);
32    StepTwo(2,3)=sin(beta_ang);
33    StepTwo(3,2)=-sin(beta_ang);
34
35    % The final gamma:
36    gammaPart=StepTwo*StepOne;
37
38    % To 12 by 12
39    gamma=zeros(12);
40    for i=1:4
41        gamma(3*i-2:3*i,3*i-2:3*i)=gammaPart;
42    end
43
44 end

```

1.5 The verification problem

1.5.1 My node and element definition for the verification problem

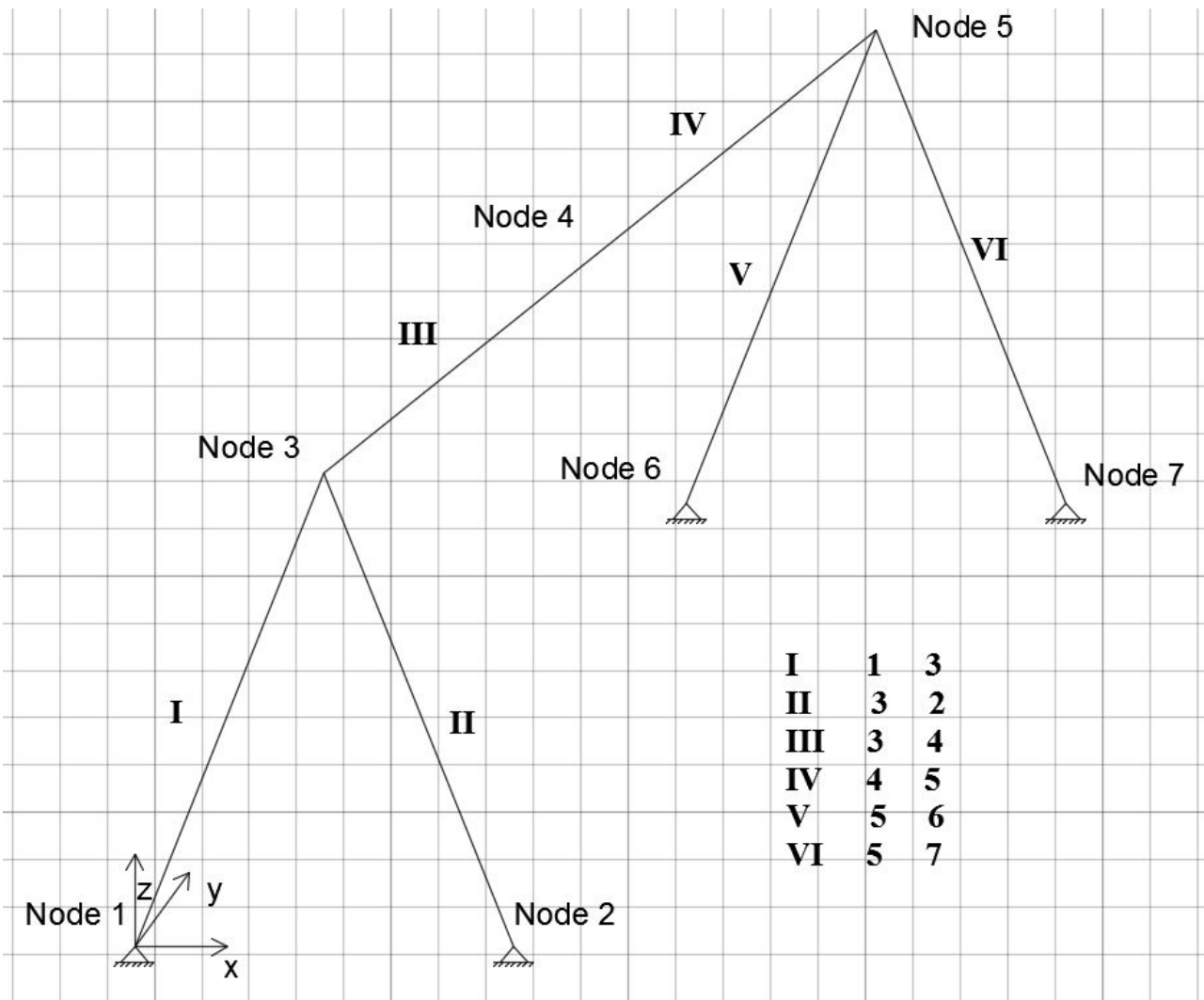


Figure 2: The node and element definition

1.5.2 The full results for the verification problem

```

1 The structure is stable.
2 The matrix is well-conditioned.
3
4 DEFL =
5
6      0      0      0  0.0008  0.0000 -0.0013
7      0      0      0  0.0008 -0.0000  0.0013
8    -0.0000  0.0026 -0.0123 -0.0026 -0.0000 -0.0000
9      0.0000 -0.0000 -4.4649 -0.0000 -0.0000  0.0000
10    -0.0000 -0.0026 -0.0123  0.0026 -0.0000  0.0000
11      0      0      0 -0.0008  0.0000  0.0013
12      0      0      0 -0.0008 -0.0000 -0.0013
13
14
15 REACT =
16
17    0.4498  0.2505  1.1250      0      0      0
18   -0.4498  0.2505  1.1250      0      0      0
19      0      0      0      0      0      0
20      0      0      0      0      0      0
21      0      0      0      0      0      0
22    0.4498 -0.2505  1.1250      0      0      0
23   -0.4498 -0.2505  1.1250      0      0      0
24
25
26 ELE_FOR =
27
28    1.21  0.25  0.00  0.00      0.00      0.00 -1.21 -0.25 -0.00 -0.00      -0.48 674.55
29    1.21 -0.25 -0.00 -0.00      0.48 -674.55 -1.21  0.25  0.00  0.00      -0.00  0.00
30    0.50  0.00  2.25 -0.00 -1252.61      0.00 -0.50 -0.00 -2.25  0.00 -2122.39  0.00
31    0.50 -0.00 -2.25 -0.00  2122.39      -0.00 -0.50  0.00  2.25  0.00  1252.61 -0.00
32    1.21  0.25  0.00      0      -0.48 674.55 -1.21 -0.25 -0.00      0      0.00 -0.00
33    1.21  0.25 -0.00 -0.00      0.48 674.55 -1.21 -0.25  0.00  0.00      -0.00 -0.00

```

1.6 The question in the verification problem

1.6.1 Deflections at points c and d

Deflections at point c in global coordinate system (unit: mm for displacement and unitless for θ):

u	v	w	θ_x	θ_y	θ_z
-0.0000	0.0026	-0.0123	-0.0026	-0.0000	-0.0000

Deflections at point d in global coordinate system (unit: mm for displacement and unitless for θ):

u	v	w	θ_x	θ_y	θ_z
0.0000	-0.0000	-4.4649	-0.0000	-0.0000	0.0000

1.6.2 Reactions at points a and b

Reactions at points a in global coordinate system (unit: kN for force and $N \cdot m$ for moment):

F_x	F_y	F_z	M_x	M_y	M_z
0.4498	0.2505	1.1250	0	0	0

Reactions at points b in global coordinate system (unit: kN for force and $N \cdot m$ for moment):

F_x	F_y	F_z	M_x	M_y	M_z
-0.4498	0.2505	1.1250	0	0	0

1.6.3 Axial forces in members a-c, c-b, and c-d

Axial forces in members a-c is 1.21 kN.

Axial forces in members c-b is 1.21 kN.

Axial forces in members c-d is 0.50 kN.

2 Program with distributed load

2.1 main program

The main program is here. The subroutines are the same.

```

1 clear
2 clc
3 % load the input information.
4 load nnodes.txt;
5 load coord.txt;
6 load concen.txt;
7 load fixity.txt;
8 load nele.txt;
9 load ends.txt;
10 load Area.txt;
11 load Izz.txt;
12 load Iyy.txt;
13 load J.txt;
14 load E.txt;
15 load v.txt;
16 load beta_ang.txt;
17
18
19 % Generate the node id.
20 % Each line represents the 6 dofs for one node.
21 node_id=zeros(nnodes,6);
22 for i=1:nnodes
23     for j=1:6
24         node_id(i,j)=(i-1)*6+j;
25     end
26 end
27
28 % Generate the member id.
29 % Each line represents the 12 DOFs for one element.
30 mem_id=zeros(nele,12);
31 for i=1:nele
32     mem_id(i,1:6)=node_id(ends(i,1),1:6);
33     mem_id(i,7:12)=node_id(ends(i,2),1:6);
34 end
35
36 % Generate the displacement boundary conditions.
37 % Each line in fixity represents the disp BC on 6 DOFs of one node.
38 % 0 represents that the DOF is fixed.
39 % NaN represnts that the DOF is free.
40 fixity_tran=fixity';
41 D=fixity_tran(:);
42
43 free_dof=find(isnan(D));
44 fixed_dof=find(D==0);
45
46
47
48 % Generate the force boundary conditions.
49 % Each line in concen represents the force BC on 6 DOFs of one node.

```

```

50 % 0 represents that the DOF does not has external force.
51 % The number represents that the force values on the DOF.
52 concen_tran=concen';
53 P_total=concen_tran(:);
54 P_free=P_total(free_dof);
55
56 % Generate the xaxis.
57 % xaxis will be used in the rotation matrix.
58 % The rotation matrix is to transform the stiffness matrix
59 % from global to local coordinate system.
60 for i=1:nele
61     xaxis(i,:)=(coord(ends(i,2),:)-coord(ends(i,1),:));
62     Length(i) = norm(xaxis(i,:));
63     xaxisunit(i,:)=xaxis(i,+)/Length(i) ;
64 end
65
66 % Generate the local stiffness matrix for each element.
67 k_stack_local=zeros(nele,12,12);
68 gamma_stack_local=zeros(nele,12,12);
69 for i=1:nele
70     k_stack_local(i,:,:)=estiff(Area(i),Izz(i),Iyy(i),J(i),E(i),v(i),Length(i));
71     gamma_stack_local(i,:,:)=etran(beta_ang(i),xaxisunit(i,1:3));
72 end
73
74 % Transform the local stiffness matrix to global stiffness matrix.
75 k_stack_global=zeros(nele,12,12);
76 for i=1:nele
77     k_stack_global(i,:,:)=(squeeze(gamma_stack_local(i,:,:))'...
78         *squeeze(k_stack_local(i,:,:))...
79         *squeeze(gamma_stack_local(i,:,:));
80 end
81
82 % Assemble the global stiffness matrix together.
83 ndof=6*nnodes;
84 k_total=zeros(ndof,ndof);
85 for i=1:nele
86     k_total(mem_id(i,1:6),mem_id(i,1:6))=...
87         k_total(mem_id(i,1:6),mem_id(i,1:6))+squeeze(k_stack_global(i,1:6,1:6));
88     k_total(mem_id(i,1:6),mem_id(i,7:12))=...
89         k_total(mem_id(i,1:6),mem_id(i,7:12))+squeeze(k_stack_global(i,1:6,7:12));
90     k_total(mem_id(i,7:12),mem_id(i,1:6))=...
91         k_total(mem_id(i,7:12),mem_id(i,1:6))+squeeze(k_stack_global(i,7:12,1:6));
92     k_total(mem_id(i,7:12),mem_id(i,7:12))=...
93         k_total(mem_id(i,7:12),mem_id(i,7:12))+squeeze(k_stack_global(i,7:12,7:12));
94 end
95
96 % Extract the K_ff.
97 K_ff=k_total(free_dof,free_dof);
98 K_sf=k_total(fixed_dof,free_dof);
99
100 % C = rcond(A) returns an estimate for the reciprocal condition of A in 1-norm.

```

```

105 % If A is well conditioned, rcond(A) is near 1.0.
106 % If A is badly conditioned, rcond(A) is near 0.
107 if abs(det(K_ff))<(10^(-15))
108     AFLAG=0;
109     display('Unstable Structure. ');
110     display('Maybe you should fix more DOFs .');
111 else
112     AFLAG=1;
113     display('The structure is stable. ');
114 end
115
116 if abs(rcond(K_ff))<(10^(-15))
117     BFLAG=0;
118     display('The matrix is badly conditioned. ');
119     display('The results may have a large error. ');
120     display('Some element may have a way bigger stiffness than others. ');
121     display('Maybe you need to check the units of element properties. ');
122 else
123     BFLAG=1;
124     display('The matrix is well-conditioned. ');
125 end
126
127 % Input the distributed load.
128 load distributedLoad.txt;
129 Fequiva_stack_local=zeros(nele,12);
130 for i=1:nele
131     qx=distributedLoad(i,1);
132     qy=distributedLoad(i,2);
133     qz=distributedLoad(i,3);
134     Fequiva_stack_local(i,:)=[-qx*Length(i)/2; -qy*Length(i)/2; -qz*Length(i)/2; ...
135                               0;          qz*Length(i)^2/12; -qy*Length(i)^2/12; ...
136                               -qx*Length(i)/2; -qy*Length(i)/2; -qz*Length(i)/2; ...
137                               0;          -qz*Length(i)^2/12;  qy*Length(i)^2/12];
138 end
139
140 % Transform the equivalent node force from local to global coordinate system.
141 Fequiva_stack_global=zeros(nele,12);
142 for i=1:nele
143     Fequiva_stack_global(i,:)=squeeze(gamma_stack_local(i,:,:))'*Fequiva_stack_local(i,:);
144 end
145 % Assembly the equivalent node force
146 Fequiva_global=zeros(ndof,1);
147 for i=1:nele
148     Fequiva_global(mem_id(i,1:12),1)=Fequiva_global(mem_id(i,1:12),1)+Fequiva_stack_global(i,:);
149 end
150
151 Fequiva_free=Fequiva_global(free_dof);
152 Fequiva_fixed=Fequiva_global(fixed_dof);
153 %
154
155
156 defl=K_ff\ (P_free+Fequiva_free);
157
158
159

```

```

160 % Put the deflection to its appropriate location.
161 % Include the fixed DOF.
162 defl_vector_total=zeros(ndof,1);
163 defl_vector_total(free_dof)=defl;
164
165 % Each line in DEFL' represents the 6 DOFs at each node.
166 DEFL=reshape(defl_vector_total,6,nnodes);
167 DEFL=DEFL';
168 display(DEFL)
169
170
171 % Generate the reaction.
172 react=K_sf*defl;
173
174 react_vector_total=zeros(ndof,1);
175 react_vector_total(fixed_dof)=react;
176 REACT=reshape(react_vector_total,6,nnodes);
177 REACT=REACT';
178 display(REACT)
179
180
181 % Calculate the element force.
182 % Transform the element from from global to local coordiante system.
183 ELE_FOR=zeros(nele,12);
184 for i=1:nele
185     element_disp_global=zeros(1,12);
186     element_disp_global(1:6)=DEFL(ends(i,1),1:6);
187     element_disp_global(7:12)=DEFL(ends(i,2),1:6);
188     element_disp_global=element_disp_global';
189     element_force_local=squeeze(k_stack_local(i,:,:))...
190         *squeeze(gamma_stack_local(i,:,:))*element_disp_global;
191     for j=1:12
192         % When we have distributed load,
193         % we should add the equivalent node force to element force.
194         ELE_FOR(i,j)=element_force_local(j)+Fequiva_stack_local(i,j);
195     end
196 end
197 display(ELE_FOR)

```

To verify my program for distributed load, two problems are tested.

2.2 Problem 1 to verify the main program with distributed load

Problem 1 is very simple. It is a one-element cantilever. We can solve this problem by hand.

As shown in the figure below, $E=1$, $I_{zz}=I_{yy}=1$, $J=2$, $A=6$, $L=2$ and the distributed load is $q=1$. Since we just want to verify the program, the parameters here are unitless to be simple.

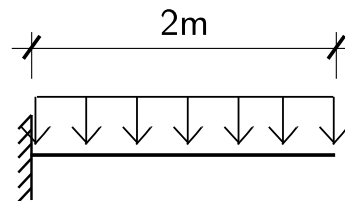


Figure 3: Problem 1 to verify the distributed load

According to the mechanics of materials, we know the vertical displacement at the free end is

$$\Delta = \frac{qL^4}{8EI} = 2 \quad (1)$$

The rotation angle is

$$\theta = \frac{qL^3}{6EI} = -1.33 \quad (2)$$

Then, I run my code with the distributed load. I get the same results below. The code and input for this example were emailed to professor in the folder "final Project Distributed Load for Cantilever". Just run "main" in the folder will give this result.

```

1 The structure is stable.
2 The matrix is well-conditioned.
3
4 DEFL =
5
6     0     0     0     0     0     0
7     0     0  2.0000     0 -1.3333     0
8
9
10 REACT =
11
12     0     0 -1.0000     0  1.6667     0
13     0     0     0     0     0     0
14
15
16 ELE_FOR =
17
18     0     0 -0.0000     0  1.3333     0
19     0     0  2.0000     0  0.6667     0

```

2.3 Problem 2 to verify the main program with distributed load

Problem 2 is the HW#3 Problem#1.

We already got the results by hand (with two elements) that the rotation at Point B is 0.0012.

I run my final project code with five elements and I got the same results below.

The 4th line in "DEFL" represents the displacement of the 4th node, which is Point B in the problem. We can see the rotation θ_y is 0.0012, the same result. The code and input for this example were emailed to professor in the folder "final Project Distributed Load for HW3Prob1". Just run "main" in the folder will give this result.

```

1 The structure is stable.
2 The matrix is well-conditioned.
3
4 DEFL =
5
6      0      0      0      0      0      0
7      0      0  0.0238      0 -0.0014      0
8      0      0  0.0347      0  0.0013      0
9      0      0      0      0  0.0012      0
10     0      0      0      0      0      0
11
12
13 REACT =
14
15      0      0 -2.9297      0  71.6146      0
16      0      0      0      0      0      0
17      0      0      0      0      0      0
18      0      0 -2.5000      0      0      0
19      0      0  0.4297      0  14.3229      0
20
21
22 ELE_FOR =
23
24      0  0 -2.9297  0  71.6146  0  0  0  2.9297  0  1.6276  0
25      0  0  2.0703  0 -43.2943  0  0  0  2.9297  0  64.7786  0
26      0  0  2.0703  0 -23.1120  0  0  0 -2.0703  0 -28.6458  0
27      0  0 -0.4297  0  28.6458  0  0  0  0.4297  0  14.3229  0

```