Advanced Matrix Structural Analysis
Final project report

Contents

1 Program without distributed load

1.1 My flowchart

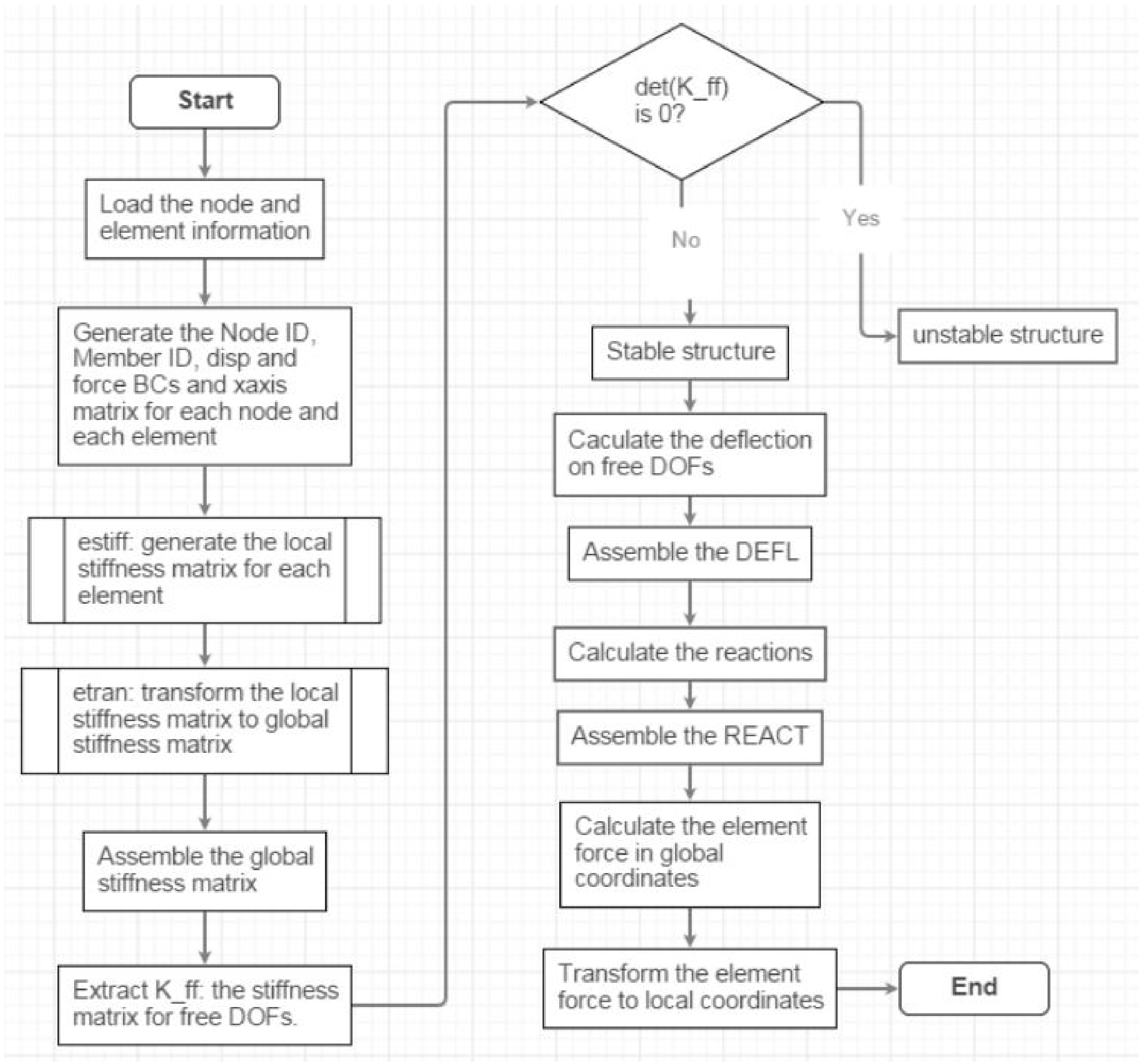
Start
Load the node and element information
Generate the Node ID, Member ID, disp and force BCs and xaxis matrix for each node and each element
estiff: generate the local stiffness matrix for each element
etran: transform the local stiffness matrix to global stiffness matrix
Assemble the global stiffness matrix
Extract K_ff: the stiffness matrix for free DOFs.
det(K_ff) is 0?
No
Yes
unstable structure
Stable structure
Caculate the deflection on free DOFs
Assemble the DEFL
Calculate the reactions
Assemble the REACT
Calculate the element force in global coordinates
Transform the element force to local coordinates
End

Figure 1: Flowchart

1.2 main program

```
clear
clc
% load the input information.
load nnodes.txt;
load coord.txt;
load concen.txt;
load fixity.txt;
load nele.txt;
load ends.txt;
load Area.txt;
load Izz.txt;
load Iyy.txt;
load J.txt;
load E.txt;
load v.txt;
load beta_ang.txt;

% Generate the node id.
% Each line represents the 6 dofs for one node.
node_id=zeros(nnodes,6);
for i=1:nnodes
    for j=1:6
        node_id(i,j)=(i-1)*6+j;
    end
end

% Generate the member id.
% Each line represents the 12 DOFs for one element.
mem_id=zeros(nele,12);
for i=1:nele
    mem_id(i,1:6)=node_id(ends(i,1),1:6);
    mem_id(i,7:12)=node_id(ends(i,2),1:6);
end

% Generate the displacement boundary conditions.
% Each line in fixity represents the disp BC on 6 DOFs of one node.
% 0 represents that the DOF is fixed.
% NaN represnts that the DOF is free.
fixity_tran=fixity';
D=fixity_tran(:);
free_dof=find(isnan(D));
fixed_dof=find(D==0);


% Generate the force boundary conditions.
% Each line in concen represents the force BC on 6 DOFs of one node.
% 0 represents that the DOF does not has external force.
% The number represnts that the force values on the DOF.
concen_tran=concen';
P_total=concen_tran(:);
P_free=P_total(free_dof);

```

```
% Generate the xaxis.
% xaxis will be used in the rotation matrix.
% The rotation matrix is to transform the stiffness matrix
% from global to local coordinate system.
for i=1:nele
    xaxis(i,:)=(coord(ends(i,2),:)-coord(ends(i,1),:));
    Length(i) = norm(xaxis(i,:));
    xaxisunit(i,:)=xaxis(i,:)/Length(i) ;
end

% Generate the local stiffness matrix for each element.
k_stack_local=zeros(nele,12,12);
gamma_stack_local=zeros(nele,12,12);
for i=1:nele
    k_stack_local(i,:,:)=estiff(Area(i),Izz(i),Iyy(i),J(i),E(i),v(i),Length(i));
    gamma_stack_local(i,:,:)=etran(beta_ang(i),xaxisunit(i,1:3));
end

 % Transform the local stiffness matrix to global stiffness matrix.
k_stack_global=zeros(nele,12,12);
for i=1:nele
    k_stack_global(i,:,:)=(squeeze(gamma_stack_local(i,:,:))')...
                          *squeeze(k_stack_local(i,:,:))...
                          *squeeze(gamma_stack_local(i,:,:));
end

% Assemble the global stiffness matrix together.
ndof=6*nnodes;
k_total=zeros(ndof,ndof);
for i=1:nele
    k_total(mem_id(i,1:6),mem_id(i,1:6))=...
        k_total(mem_id(i,1:6),mem_id(i,1:6))+squeeze(k_stack_global(i,1:6,1:6));
    k_total(mem_id(i,1:6),mem_id(i,7:12))=...
        k_total(mem_id(i,1:6),mem_id(i,7:12))+squeeze(k_stack_global(i,1:6,7:12));
    k_total(mem_id(i,7:12),mem_id(i,1:6))=...
        k_total(mem_id(i,7:12),mem_id(i,1:6))+squeeze(k_stack_global(i,7:12,1:6));
    k_total(mem_id(i,7:12),mem_id(i,7:12))=...
        k_total(mem_id(i,7:12),mem_id(i,7:12))+squeeze(k_stack_global(i,7:12,7:12));
end

% Extract the K_ff.
K_ff=k_total(free_dof,free_dof);
K_sf=k_total(fixed_dof,free_dof);


% C = rcond(A) returns an estimate for the reciprocal condition of A in 1-norm.
% If A is well conditioned, rcond(A) is near 1.0.
% If A is badly conditioned, rcond(A) is near 0.
if abs(det(K_ff))<(10^(-15))
    AFLAG=0;
    display('Unstable Structure.');
    display('Maybe you should fix more DOFs .');
else
    AFLAG=1;
    display('The structure is stable.');
```

```
end

if abs(rcond(K_ff))<(10^(-15))
    BFLAG=0;
    display('The matrix is badly conditioned. ');
    display('The results may have a large error. ');
    display('Some element may have a way bigger stiffness than others.');
    display('Maybe you need to check the units of element properties.');
else
    BFLAG=1;
    display('The matrix is well-conditioned.');
end

defl=K_ff\P_free;

% Put the deflection to its appropriate location.
% Include the fixed DOF.
defl_vector_total=zeros(ndof,1);
defl_vector_total(free_dof)=defl;

% Each line in DEFL' represents the 6 DOFs at each node.
DEFL=reshape(defl_vector_total,6,nnodes);
DEFL=DEFL';
display(DEFL)

% Generate the reaction.
react=K_sf*defl;

react_vector_total=zeros(ndof,1);
react_vector_total(fixed_dof)=react;
REACT=reshape(react_vector_total,6,nnodes);
REACT=REACT';
display(REACT)

% Calculate the element force.
% Transform the element from from global to local coordiante system.
ELE_FOR=zeros(nele,12);
for i=1:nele
    element_disp_global=zeros(1,12);
    element_disp_global(1:6)=DEFL(ends(i,1),1:6);
    element_disp_global(7:12)=DEFL(ends(i,2),1:6);
    element_disp_global=element_disp_global';
    element_force_local=squeeze(k_stack_local(i,:,:))...
                        *squeeze(gamma_stack_local(i,:,:))*element_disp_global;
    ELE_FOR(i,:)=element_force_local(:);
end
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.

```
function [stiffmatrix] = estiff(A,Izz,Iyy,J,E,v,Length)
      % A      -- Area
    % Izz   -- Second moment of area over axis zz
    % Iyy   -- Second moment of area over axis zz
    % J     -- torsion constant
    % E     -- Elastic modulus
    % v     -- Poisson's ratio
    % Length -- Element Length

   % write 4 (2 by 2 ) small matrix first and then put them together.
   KK1=diag([A/Length, 12*Izz/Length^3, 12*Iyy/Length^3, J/(2*(1+v)*Length), 4*Iyy/Length,
       4*Izz/Length]);
   KK1(2,6)=6*Izz/Length^2;
   KK1(6,2)=KK1(2,6);
   KK1(3,5)=-6*Iyy/Length^2;
   KK1(5,3)=KK1(3,5);

   KK2=diag([-A/Length, -12*Izz/Length^3, -12*Iyy/Length^3, -J/(2*(1+v)*Length), 2*Iyy/Length,
       2*Izz/Length]);
   KK2(2,6)=6*Izz/Length^2;
   KK2(6,2)=-KK2(2,6);
   KK2(3,5)=-6*Iyy/Length^2;
   KK2(5,3)=6*Iyy/Length^2;

   KK3=diag([-A/Length, -12*Izz/Length^3, -12*Iyy/Length^3, -J/(2*(1+v)*Length), 2*Iyy/Length,
       2*Izz/Length]);
   KK3(2,6)=-6*Izz/Length^2;
   KK3(6,2)=6*Izz/Length^2;
   KK3(3,5)=6*Iyy/Length^2;
   KK3(5,3)=-6*Iyy/Length^2;

   KK4=diag([A/Length, 12*Izz/Length^3, 12*Iyy/Length^3, J/(2*(1+v)*Length), 4*Iyy/Length,
       4*Izz/Length]);
   KK4(2,6)=-6*Izz/Length^2;
   KK4(6,2)=KK4(2,6);
   KK4(3,5)=6*Iyy/Length^2;
   KK4(5,3)=KK4(3,5);

   % Put them together.
   stiffmatrix=E*[KK1 KK2;KK3 KK4];

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.

```
function [gamma] = etran(beta_ang,xaxis)
   % Make the transformation matrix by two steps.
   % Step 1: take xaxis by matrix StepOne.
   % Step 2: rotate xaxis by matrix StepTwo.

   % Initialize the matrix StepOne and StepTwo.
   StepOne=zeros(3);
   StepTwo=zeros(3);

   % StepOne starts!!!
   %  Deal with the special case when xaxis is parallel to y direction.
   if (xaxis(1)==0 & xaxis(3)==0)
      StepOne(1,:)=xaxis;
      StepOne(2,:)=[-1,0,0];
      StepOne(3,:)=cross(StepOne(1,:),StepOne(2,:));

   %  Deal with the normal case
   else
      StepOne(1,:)=xaxis;
      % Change to unit vector
      StepOne(1,:)=StepOne(1,:)/norm(StepOne(1,:));
      yaxis=[0 1 0];
      StepOne(3,:)=cross(StepOne(1,:),yaxis) ;
      StepOne(3,:)=StepOne(3,:)/norm(StepOne(3,:));
      StepOne(2,:)=cross(StepOne(3,:),StepOne(1,:));
      StepOne(2,:)=StepOne(2,:)/norm(StepOne(2,:));
   end
   % % StepTwo starts!!!
   StepTwo(1,1)=1;
   StepTwo(2,2)=cos(beta_ang);
   StepTwo(3,3)=cos(beta_ang);
   StepTwo(2,3)=sin(beta_ang);
   StepTwo(3,2)=-sin(beta_ang);

   % The final gamma:
   gammaPart=StepTwo*StepOne;

   % To 12 by 12
   gamma=zeros(12);
   for i=1:4
      gamma(3*i-2:3*i,3*i-2:3*i)=gammaPart;
   end

end
```

1.5 The verification problem

1.5.1 My node and element definition for the verification problem

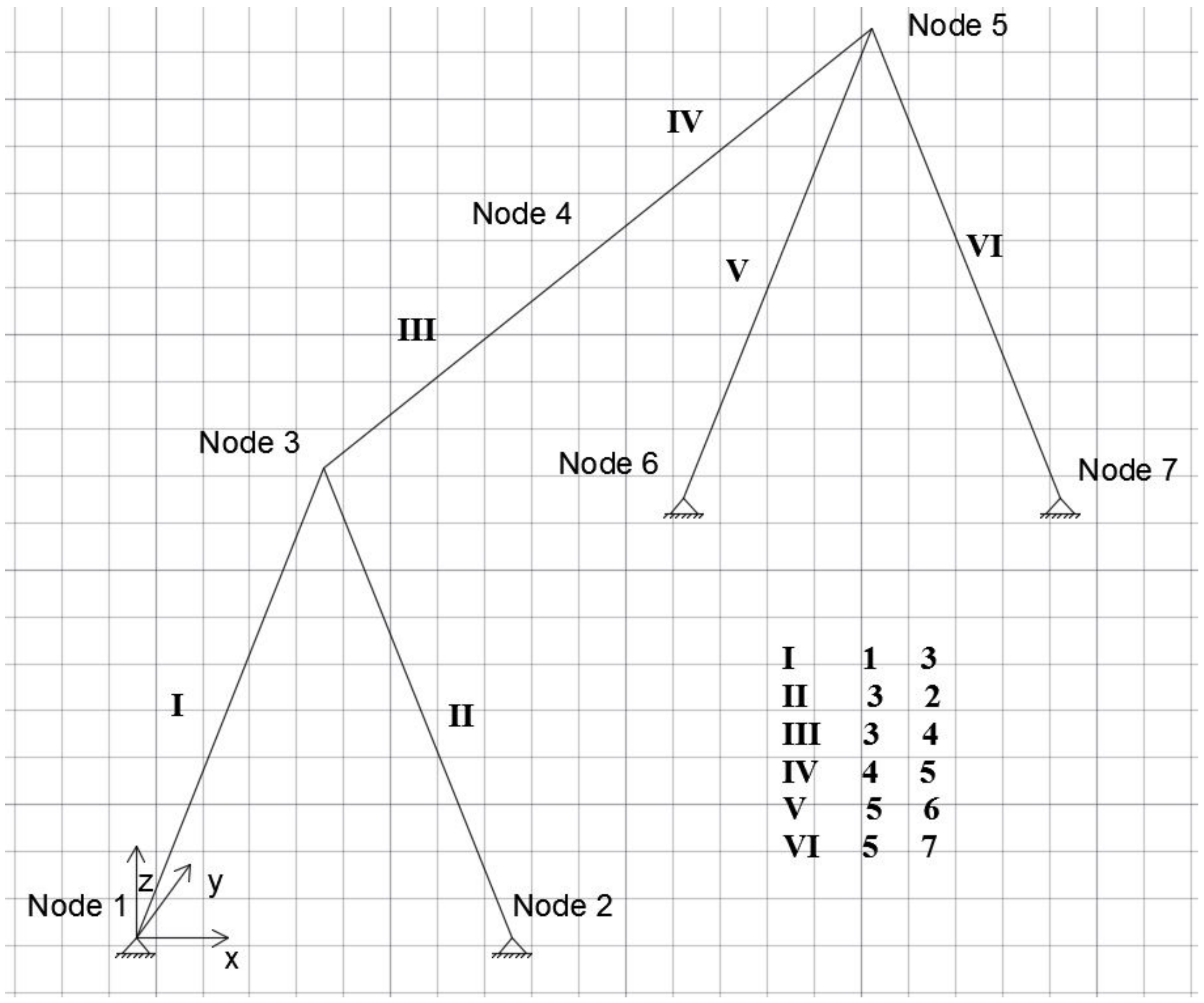

Figure 2: The node and element definition

1.5.2 The full results for the verification problem

```
The structure is stable.
The matrix is well-conditioned.

DEFL =

        0         0         0    0.0008    0.0000   -0.0013
        0         0         0    0.0008   -0.0000    0.0013
  -0.0000    0.0026   -0.0123   -0.0026   -0.0000   -0.0000
   0.0000   -0.0000   -4.4649   -0.0000   -0.0000    0.0000
  -0.0000   -0.0026   -0.0123    0.0026   -0.0000    0.0000
        0         0         0   -0.0008    0.0000    0.0013
        0         0         0   -0.0008   -0.0000   -0.0013


REACT =

   0.4498    0.2505    1.1250         0         0         0
  -0.4498    0.2505    1.1250         0         0         0
        0         0         0         0         0         0
        0         0         0         0         0         0
        0         0         0         0         0         0
   0.4498   -0.2505    1.1250         0         0         0
  -0.4498   -0.2505    1.1250         0         0         0


ELE_FOR =

   1.21   0.25   0.00   0.00      0.00     0.00  -1.21  -0.25  -0.00  -0.00     -0.48  674.55
   1.21  -0.25  -0.00  -0.00      0.48  -674.55  -1.21   0.25   0.00   0.00     -0.00    0.00
   0.50   0.00   2.25  -0.00  -1252.61     0.00  -0.50  -0.00  -2.25   0.00  -2122.39    0.00
   0.50  -0.00  -2.25  -0.00   2122.39    -0.00  -0.50   0.00   2.25   0.00   1252.61   -0.00
   1.21   0.25   0.00      0     -0.48   674.55  -1.21  -0.25  -0.00      0      0.00   -0.00
   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.

```
clear
clc
% load the input information.
load nnodes.txt;
load coord.txt;
load concen.txt;
load fixity.txt;
load nele.txt;
load ends.txt;
load Area.txt;
load Izz.txt;
load Iyy.txt;
load J.txt;
load E.txt;
load v.txt;
load beta_ang.txt;


% Generate the node id.
% Each line represents the 6 dofs for one node.
node_id=zeros(nnodes,6);
for i=1:nnodes
    for j=1:6
        node_id(i,j)=(i-1)*6+j;
    end
end

% Generate the member id.
% Each line represents the 12 DOFs for one element.
mem_id=zeros(nele,12);
for i=1:nele
    mem_id(i,1:6)=node_id(ends(i,1),1:6);
    mem_id(i,7:12)=node_id(ends(i,2),1:6);
end

% Generate the displacement boundary conditions.
% Each line in fixity represents the disp BC on 6 DOFs of one node.
% 0 represents that the DOF is fixed.
% NaN represnts that the DOF is free.
fixity_tran=fixity';
D=fixity_tran(:);

free_dof=find(isnan(D));
fixed_dof=find(D==0);



% Generate the force boundary conditions.
% Each line in concen represents the force BC on 6 DOFs of one node.
% 0 represents that the DOF does not has external force.
% The number represnts that the force values on the DOF.
concen_tran=concen';
P_total=concen_tran(:);
P_free=P_total(free_dof);

% Generate the xaxis.
% xaxis will be used in the rotation matrix.
% The rotation matrix is to transform the stiffness matrix
% from global to local coordinate system.
for i=1:nele
    xaxis(i,:)=(coord(ends(i,2),:)-coord(ends(i,1),:));
    Length(i) = norm(xaxis(i,:));
    xaxisunit(i,:)=xaxis(i,:)/Length(i) ;
end


% Generate the local stiffness matrix for each element.
k_stack_local=zeros(nele,12,12);
gamma_stack_local=zeros(nele,12,12);
for i=1:nele
    k_stack_local(i,:,:)=estiff(Area(i),Izz(i),Iyy(i),J(i),E(i),v(i),Length(i));
    gamma_stack_local(i,:,:)=etran(beta_ang(i),xaxisunit(i,1:3));
end

 % Transform the local stiffness matrix to global stiffness matrix.
k_stack_global=zeros(nele,12,12);
for i=1:nele
    k_stack_global(i,:,:)=(squeeze(gamma_stack_local(i,:,:))')...
                         *squeeze(k_stack_local(i,:,:))...
                         *squeeze(gamma_stack_local(i,:,:));
end


% Assemble the global stiffness matrix together.
ndof=6*nnodes;
k_total=zeros(ndof,ndof);
for i=1:nele
    k_total(mem_id(i,1:6),mem_id(i,1:6))=...
        k_total(mem_id(i,1:6),mem_id(i,1:6))+squeeze(k_stack_global(i,1:6,1:6));
    k_total(mem_id(i,1:6),mem_id(i,7:12))=...
        k_total(mem_id(i,1:6),mem_id(i,7:12))+squeeze(k_stack_global(i,1:6,7:12));
    k_total(mem_id(i,7:12),mem_id(i,1:6))=...
        k_total(mem_id(i,7:12),mem_id(i,1:6))+squeeze(k_stack_global(i,7:12,1:6));
    k_total(mem_id(i,7:12),mem_id(i,7:12))=...
        k_total(mem_id(i,7:12),mem_id(i,7:12))+squeeze(k_stack_global(i,7:12,7:12));
end


% Extract the K_ff.
K_ff=k_total(free_dof,free_dof);
K_sf=k_total(fixed_dof,free_dof);


% C = rcond(A) returns an estimate for the reciprocal condition of A in 1-norm.
```

```matlab
% If A is well conditioned, rcond(A) is near 1.0.
% If A is badly conditioned, rcond(A) is near 0.
if abs(det(K_ff))<(10^(-15))
    AFLAG=0;
    display('Unstable Structure.');
    display('Maybe you should fix more DOFs .');
else
    AFLAG=1;
    display('The structure is stable.');
end

if abs(rcond(K_ff))<(10^(-15))
    BFLAG=0;
    display('The matrix is badly conditioned. ');
    display('The results may have a large error. ');
    display('Some element may have a way bigger stiffness than others.');
    display('Maybe you need to check the units of element properties.');
else
    BFLAG=1;
    display('The matrix is well-conditioned.');
end


% Input the distributed load.
load distributedLoad.txt;
Fequiva_stack_local=zeros(nele,12);
for i=1:nele
    qx=distributedLoad(i,1);
    qy=distributedLoad(i,2);
    qz=distributedLoad(i,3);
    Fequiva_stack_local(i,:)=[-qx*Length(i)/2; -qy*Length(i)/2; -qz*Length(i)/2; ...
                        0;         qz*Length(i)^2/12;  -qy*Length(i)^2/12; ...
                -qx*Length(i)/2;     -qy*Length(i)/2;     -qz*Length(i)/2; ...
                        0;     -qz*Length(i)^2/12;   qy*Length(i)^2/12];
end

% Transform the equivalent node force from local to global coordinate system.
Fequiva_stack_global=zeros(nele,12);
for i=1:nele
    Fequiva_stack_global(i,:)=squeeze(gamma_stack_local(i,:,:))'*Fequiva_stack_local(i,:)';
end
% Assembly the equivalent node force
Fequiva_global=zeros(ndof,1);
for i=1:nele
    Fequiva_global(mem_id(i,1:12),1)=Fequiva_global(mem_id(i,1:12),1)+Fequiva_stack_global(i,:)';
end

Fequiva_free=Fequiva_global(free_dof);
Fequiva_fixed=Fequiva_global(fixed_dof);
%


defl=K_ff\(P_free+Fequiva_free);


```

```
% Put the deflection to its appropriate location.
% Include the fixed DOF.
defl_vector_total=zeros(ndof,1);
defl_vector_total(free_dof)=defl;

% Each line in DEFL' represents the 6 DOFs at each node.
DEFL=reshape(defl_vector_total,6,nnodes);
DEFL=DEFL';
display(DEFL)


% Generate the reaction.
react=K_sf*defl;

react_vector_total=zeros(ndof,1);
react_vector_total(fixed_dof)=react;
REACT=reshape(react_vector_total,6,nnodes);
REACT=REACT';
display(REACT)


% Calculate the element force.
% Transform the element from from global to local coordiante system.
ELE_FOR=zeros(nele,12);
for i=1:nele
    element_disp_global=zeros(1,12);
    element_disp_global(1:6)=DEFL(ends(i,1),1:6);
    element_disp_global(7:12)=DEFL(ends(i,2),1:6);
    element_disp_global=element_disp_global';
    element_force_local=squeeze(k_stack_local(i,:,:))...
                    *squeeze(gamma_stack_local(i,:,:))*element_disp_global;
    for j=1:12
        % When we have distributed load,
        % we should add the equivalent node force to element force.
        ELE_FOR(i,j)=element_force_local(j)+Fequiva_stack_local(i,j);
    end
end
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, Izz=Iyy=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.

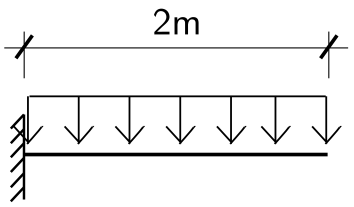

Figure 3: Problem 1 to verify the distributed load

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

$$\triangle = \frac{qL^4}{8EI} = 2 \tag{1}$$

The rotation angle is

$$\theta = \frac{qL^3}{6EI} = -1.33 \tag{2}$$

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.

```
The structure is stable.
The matrix is well-conditioned.

DEFL =

         0         0         0         0         0         0
         0         0    2.0000         0   -1.3333         0


REACT =

         0         0   -1.0000         0    1.6667         0
         0         0         0         0         0         0


ELE_FOR =

         0         0   -0.0000         0    1.3333         0
         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.

```
The structure is stable.
The matrix is well-conditioned.

DEFL =

        0        0        0        0        0        0
        0        0   0.0238        0  -0.0014        0
        0        0   0.0347        0   0.0013        0
        0        0        0        0   0.0012        0
        0        0        0        0        0        0


REACT =

        0        0  -2.9297        0  71.6146        0
        0        0        0        0        0        0
        0        0        0        0        0        0
        0        0  -2.5000        0        0        0
        0        0   0.4297        0  14.3229        0


ELE_FOR =

        0   0  -2.9297  0   71.6146  0  0  0   2.9297  0    1.6276  0
        0   0   2.0703  0  -43.2943  0  0  0   2.9297  0   64.7786  0
        0   0   2.0703  0  -23.1120  0  0  0  -2.0703  0  -28.6458  0
        0   0  -0.4297  0   28.6458  0  0  0   0.4297  0   14.3229  0
```