Stanford University

Department of Civil &Environmental Engineering SEG

Advanced Structural Analysis Programming Project Report

Professor Gregory Deierlein

Nov 30th, 2015

Wenjin Situ

Liyi Wang

Table of Contents:

Summary of CEE 280 Programing Assignment Project

The objective of this designed program is to perform first order linear-elastic analysis of 3D structures with any geometry using the MASTAN2 post-processor. The finished program is able to solve the nodal displacements and reactions of the structure under concentrated nodal forces, loads along elements, support settlements, and recovered the element forces and displacements in local coordinates. Object-oriented programming methodology of Matlab language is employed throughout the project.

The whole program is divided into three classes to perform the required analysis. These three classes are **Node**, **Element**, and **Analysis**. The node class contains the properties related to the nodes of the created structural model. In this class, the nodal coordinates of the 3D structure and the assigned DOF of each node are constructed. The constructed properties are then saved with private methods to avoid access from outside of the class. However, as these properties will be used in the element class, public "Get" methods are written to extract the properties of the class so they can be called from outside of the node class.

The Element class contains all the required properties and methods related to the element level of the structure. Element properties such as section area, moment of inertia, Young's Modulus , etc. will be saved as private properties after the user inputs them into MASTAN2. Element length is computed using the nodal coordinates from the node class. Meanwhile, the local and global stiffness of the element are computed with the element properties and transformation matrix. Moreover, fixed end forces in local and global coordinates will also be computed. As usual, all the properties are saved within the element class with private methods. Public methods are then written to extract element stiffness matrix and fixed end forces for the use of structural analysis.

In the analysis class, nodes and elements are created as object vectors in the analysis class by calling the constructor methods written in the Node and Element classes. The global structural stiffness matrix of the 3D structural model is constructed by looping over the elements with the element Degree of Freedoms. The sparse built in function in Matlab is used to condense out the zeros in the structural global stiffness matrix to enhance the efficiency of the program. The ClassifyDOF method is written to classify the free, displacement, and support DOF of the fixity matrix. The concentrated load vector and the nodal fixed end force vectors are assembled by looping over the nodes and elements of the structure. With these two load vectors, the final load vector is obtained by subtracting the concentrated force vector from the fixed end force nodal vector. With the classified DOF, the structural stiffness matrix and the load vector are partitioned into several sub-matrices using the linear-indexing feature of Matlab. The nodal displacements of the structure are calculated with the partitioned Kff and Pf matrices. Similarly, the reactions of

the structure can also be computed with the partitioned stiffness matrices and the obtained nodal displacement vectors.
With the solved nodal displacements of the structure, we aim to recover the member forces and member displacements in the element level. This is achieved by the written a RecoverElementForce method. In this method, the nodal displacements of the structure are mapped into the local level of the members using the element DOF. The element forces will then be computed by the ComputeForces method written in the Element class. All the element forces and element displacements are recovered in this manner by a written for loop. The method RunAnalysis is created in the analysis class; all the analysis methods in the analysis class will be performed when this method is called.

The program includes features to exam the condition of the assembled structural global stiffness matrix, as well as to compute the error of the computed displacements. The method CheckKffMatrix is designed o fulfill the first task. In this method, a logical flag AFLAG is generated to indicate the condition of the Kff matrix. As Matlab is working with 16 significant digits, the designers of the program decide to limit the lost of digits to 12 so that at least 4 significant digits are available in the end of the analysis. The logical flag will display 'Unstable Structure' if more than 12 digits are lost. The method ComputeError is created to calculate the error of the calculation by back calculating the load vector. The load vector can be back calculated by using the solved displacements, the stiffness matrix, and the fixed end force vector. The error will be the difference between the back-calculated load vector and the defined load vector.

The final task of the program is to return all the solved values into MASTAN2. To achieve this, the method GetMastan2Returns is created in the Analysis class to return the solved deflection, reaction, element force, and the logical flag AFLAG value. A single file with the unchanged function name ud_3d1el is created for MASTAN2 to recognize it and run the written analysis codes. With the three classes and the ud_3d1el files replaced in the MASTAN2 directory, a linear-elastic analysis can be performed in the user-define option under the analysis drop-down menu of the MASTAN2 interface.

The team embraces the practice of object oriented programming for the following reasons. First, by dividing the programs into different classes and to create objects within the class, the team finds it easier to divide tasks between the teammates. Moreover, OOP is more organized than the usual Matlab step by step script; therefore, the team is more efficient to make changes of the program in the later interims, as well as easier to debug when writing the codes. As the program gets more and more complicated and includes more features, the team recognizes the benefits of writing the scripts in this style.

In conclusion, the team designed the program with three classes: Node, Element, and Analysis. Methods are written within each class with good programming practice: only the methods need to be called from outside of the class should be public, while the rest of the methods are restricted to be private. The Node and Element classes are mainly designed to construct the required material and geometry properties for the Analysis class, while the Analysis class is responsible to assemble the obtained properties passed by the previous two classes and performs linear elastic analysis of the structure. The team is pleased for the included features of the finished program as it can return most of the needed solutions for a first order analysis; however, possible features such as thermal loads analysis are not included in the program. With more time granted, the team will improve the features of the designed program for its better use.

Code Design Document Flowchart & Printed Listing

Node class

data properties:

- node_coord
- node_DOF

methods:

- Constructor
- AssignDOF()
- GetNodeDOF()

Constructor

Arguments

- node_coord
- node_number

Computes/Stores

- node_coord

Returns

- N/A

GetNodeCoord()

Arguments

- N/A

Computes/Stores

- N/A

Returns

- node_coord

AssignDOF()

Arguments

- node_number

Computes/Stores

- node_DOF

Returns

- N/A

GetNodeDOF()

Arguments

- N/A

Computes/Stores

- N/A

Returns

- node_DOF

Element class

data properties:

- element_nodes
- A
- Izz
- Iyy
- J
- Ayy
- Azz
- E
- v
- webdir
- w
- length
- TransformationMatrix
- LocalStiffnessMatrix
- GlobalStiffnessMatrix
- GlobalDisplacement
- element_DOF
- FeF_local
- FeF_global

methods:

- Constructor
- GetElementDOF()
- GetGlobalStiffnessMatrix ()
- GetFeF()
- ComputeForces()
- ComputeLength()
- ComputeTransformation Matrix()
- ComputeElasticStiffness Matrix()
- RetrieveDOF()
- ComputeFixedEndForces ()

Constructor

Arguments

- element_nodes
- A
- Izz
- Iyy
- J
- Ayy
- Azz
- E
- v
- webdir
- w

Computes/Stores

- element_nodes
- A
- Izz
- Iyy
- J
- Ayy
- Azz
- E
- v
- webdir
- w

Returns

- N/A

↓

ComputeLength()

Arguments

- N/A

Computes/Stores

- length

Returns

- N/A

GetElementDOF()

Arguments

- N/A

Computes/Stores

- N/A

Returns

- element_DOF

GetGlobalStiffness Matrix()

Arguments

- N/A

Computes/Stores

- N/A

Returns

- GlobalStiffness Matrix

GetFeF()

Arguments

- N/A

Computes/Stores

- N/A

Returns

- FeF_global

ComputeForces()

Arguments

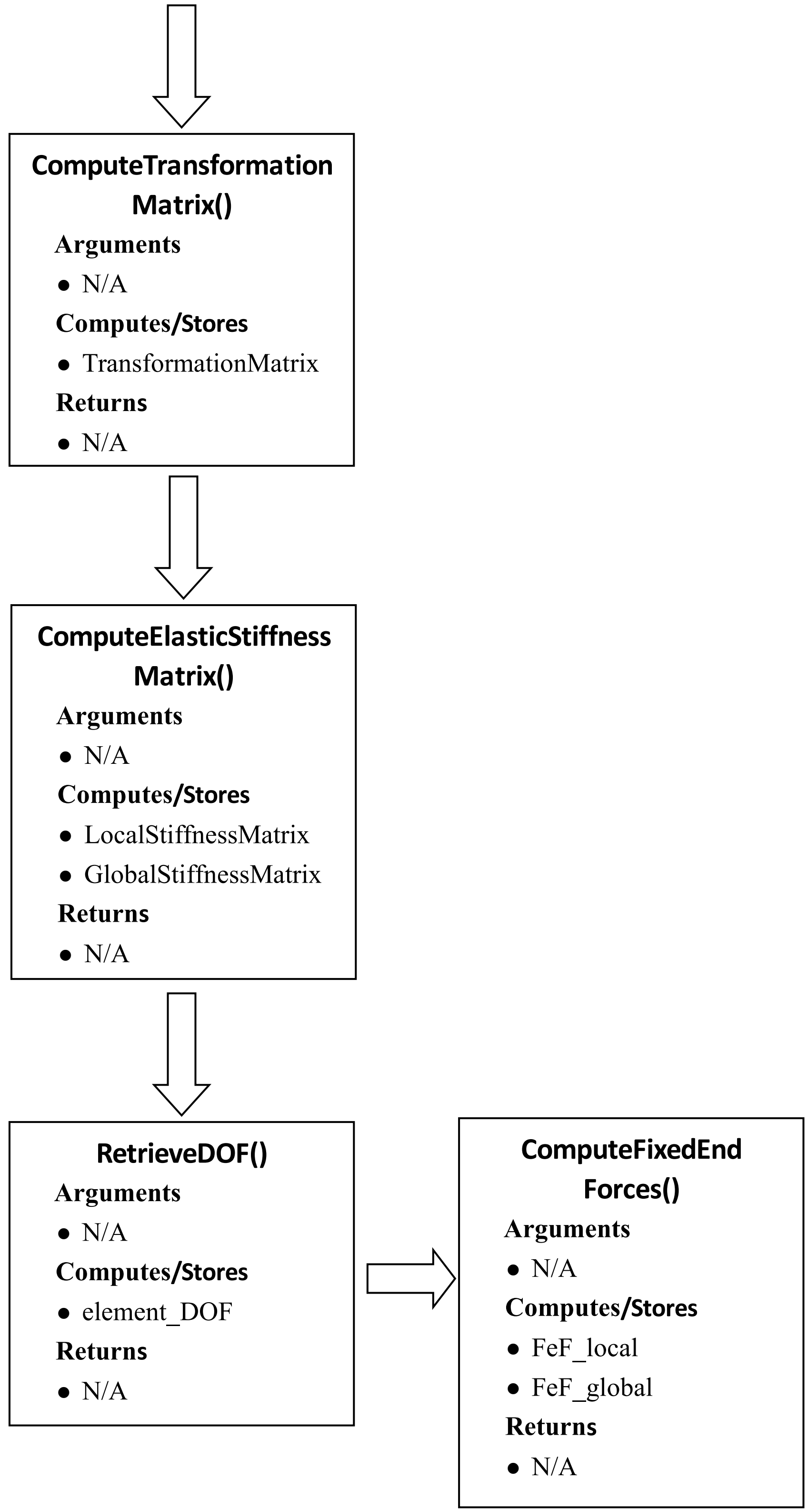
- displacement

Computes/Stores

- ElementForces

Returns

- ElementForces


ComputeTransformation Matrix()
Arguments
• N/A
Computes/Stores
• TransformationMatrix
Returns
• N/A
ComputeElasticStiffness Matrix()
Arguments
• N/A
Computes/Stores
• LocalStiffnessMatrix
• GlobalStiffnessMatrix
Returns
• N/A
RetrieveDOF()
Arguments
• N/A
Computes/Stores
• element_DOF
Returns
• N/A
ComputeFixedEnd Forces()
Arguments
• N/A
Computes/Stores
• FeF_local
• FeF_global
Returns
• N/A

Analysis class

data properties:

- nnodes
- coord
- fixity
- concen
- nele
- ends
- A
- Izz
- Iyy
- J
- Ayy
- Azz
- E
- v
- webdir
- w
- NodeVector
- ElementVector
- StiffnessMatrix
- LoadVector
- free_dof
- disp_dof
- supp_dof
- ConditionNumber
- LostDigits
- Kff
- Ksf
- Kfn
- Knn
- Kns
- Kss
- Pf
- Feff
- Fefs
- Fefn
- Delf
- DEFL
- REACT
- ELE_FOR
- AFLAG

Constructor

Arguments

- nnodes
- coord
- fixity
- concen
- nele
- ends
- A
- Izz
- Iyy
- J
- Ayy
- Azz
- E
- v
- webdir
- w

Computes/Stores

- nnodes
- coord
- fixity
- concen
- nele
- ends
- A
- Izz
- Iyy
- J
- Ayy
- Azz
- E
- v
- webdir
- w

Returns

- N/A

RunAnalysis()

Arguments

- N/A

Computes/Stores

- N/A

Returns

- N/A

CreateStiffness Matrix()

Arguments

- N/A

Computes/Stores

- StiffnessMatrix

Returns

- N/A

ClassifyDOF()

Arguments

- N/A

Computes/Stores

- free_dof
- supp_dof
- disp_dof

Returns

- N/A

methods:

- Constructor
- RunAnalysis()
- GetMastan2Returns()
- CreateNodes()
- CreateElements()
- CreateStiffnessMatrix()
- CreateLoadVector()
- ClassifyDOF()
- ComputeStiffnessSub Matrices()
- CheckKffMatrix()
- ComputeDisplacements Reactions()
- RecoverElementForces()
- ComputeError()

CreateNodes()

Arguments
- N/A

Computes/Stores
- NodeVector

Returns
- N/A

CreateElements()

Arguments
- N/A

Computes/Stores
- ElementVector

Returns
- N/A

GetMastan2 Returns()

Arguments
- N/A

Computes/Stores
- N/A

Returns
- DEFL
- REACT
- ELE_FOR
- AFLAG

CreateLoadVector()

Arguments
- N/A

Computes/Stores
- Pf
- Feff
- Fefs
- Fefn
- LoadVector

Returns
- N/A

ComputeStiffnessSub Matrices()

Arguments
- N/A

Computes/Stores
- Kff
- Ksf
- Kfn
- Knn
- Kss
- Kns

Returns
- N/A

CreateLoadVector()

Arguments
- N/A

Computes/Stores
- ConditionNumber
- LostDigits
- AFLAG

Returns
- N/A

ComputeDisplacementsReactions()

Arguments
- N/A

Computes/Stores
- Delf
- DEFL
- REACT

Returns
- N/A

RecoverElementForces()

Arguments
- N/A

Computes/Stores
- ELE_FOR

Returns
- N/A

RecoverElementForces()

Arguments
- N/A

Computes/Stores
- Error

Returns
- N/A

Matlab Source Code:

WSLW_Node

```
classdef WSLW_Node < handle
% Node class for a 3-dimensional framed structure
    % Private properties go here
    properties (Access = private)
        % 3x1 vector containing the x, y, and z coordinates of the node
        node_coord
        node_DOF
    end
    % Public methods go here
    methods (Access = public)
        %% Constructor
        %    Arguments
        %      node_coord:  3x1 vector containing the x, y, and z
coordinates of the node
        %      node_number: serial number of this node
        function self = WSLW_Node(node_coord,node_number)
            self.node_coord = node_coord;
            self.AssignDOF(node_number); %assign DOF based on the node
number
        end
        %% Get Node Coordinates
        %  Return "node_coord"
        function node_coord = GetNodeCoord(self)
            node_coord = self.node_coord;
        end
        %% Get Node DOF
        %  Return "node_DOF"
        function node_DOF = GetNodeDOF(self)
            node_DOF = self.node_DOF;
        end
    end
    % Private methods go here
    methods (Access = private)
  %% Assign numbers to the degrees of freedom of nodes
        function AssignDOF(self,node_number)
            self.node_DOF = 6*(node_number-1)+(1:6)';
        end
    end
end
```

WSLW_Element

```
classdef WSLW_Element < handle

 % Element class for a 3-dimensional framed structure
 % Private properties go here
    properties (Access = private)
        element_nodes
        A
        Izz
        Iyy
        J
        Ayy
        Azz
        E
        v
        webdir
        w
        length
        TransformationMatrix
        LocalStiffnessMatrix
        GlobalStiffnessMatrix
        GlobalDisplacement
        element_DOF
        FeF_local
        FeF_global
    end

    % Public methods go here
    methods (Access = public)
   %% Constructor
        function self =
WSLW_Element(element_nodes,A,Izz,Iyy,J,Ayy,Azz,...
    E,v,webdir,w) %beta_ang
            self.A=A;
            self.Iyy=Iyy;
            self.Izz=Izz;
            self.J=J;
            self.Ayy=Ayy;
            self.Azz=Azz;
            self.E=E;
            self.v=v;
```

```
            self.webdir=webdir;
            self.w=w;
            self.element_nodes=element_nodes;
            self.ComputeLength();
            self.ComputeTransformationMatrix();
            self.ComputeElasticStiffnessMatrix();
            self.RetrieveDOF();
            self.ComputeFixedEndForces();
        end
    %% Get Element DOF
        %  Return "element_DOF"
        function element_DOF = GetElementDOF(self)
            element_DOF = self.element_DOF;
        end
    %% Get Global Stiffness Matrix
        %  Return "GlobalStiffnessMatrix"
        function GlobalStiffnessMatrix =
GetGlobalStiffnessMatrix(self)
            GlobalStiffnessMatrix = self.GlobalStiffnessMatrix;
        end
   %% Get Fixed End Forces in Global Coordinates
         %  Return "FeF_global"
        function FeF_global = GetFeF(self)
            FeF_global = self.FeF_global;
        end
   %% Compute the end forces of the element
         % Get the element displacements and return the element forces
        function ElementForces = ComputeForces(self,displacement)
                 ElementForces =
self.LocalStiffnessMatrix*(self.TransformationMatrix*displacement)
+self.FeF_local;
        end
    end

    % Private methods go here
    methods (Access = private)
  %% Compute the element's length
        function ComputeLength(self)

node_coordinates=[self.element_nodes(1).GetNodeCoord();self.eleme
nt_nodes(2).GetNodeCoord()];
```

```
self.length=sqrt((node_coordinates(4)-node_coordinates(1))^2+...

(node_coordinates(5)-node_coordinates(2))^2+(node_coordinates(6)-
node_coordinates(3))^2);
        end

  %% Compute the element's geometric transformation matrix
        function ComputeTransformationMatrix(self)

node_coordinates=[self.element_nodes(1).GetNodeCoord();self.eleme
nt_nodes(2).GetNodeCoord()];

x_vect=[(node_coordinates(4)-node_coordinates(1)),(node_coordinat
es(5)...

-node_coordinates(2)),(node_coordinates(6)-node_coordinates(3))]/
self.length;
            z_vect = cross(x_vect,self.webdir);
            gama=[x_vect;self.webdir;z_vect];

self.TransformationMatrix=[gama,zeros(3),zeros(3),zeros(3);
                                  zeros(3),gama,zeros(3),zeros(3);
                                  zeros(3),zeros(3),gama,zeros(3);

zeros(3),zeros(3),zeros(3),gama];
        end

  %% Compute the element's elastic stiffness matrix in local and global
coordinates
        function ComputeElasticStiffnessMatrix(self)
            G=self.E/(2*(1+self.v));
            nzz=self.E*self.Izz/(self.Ayy*G); %Factor of shear
deformation along z-z axial
            nyy=self.E*self.Iyy/(self.Azz*G); %Factor of shear
deformation along y-y axial

Flex_zz1=self.Izz/(self.length*(self.length^2/12+nzz)); %replace
12Izz/L^3
            Flex_zz2=self.Izz/(2*(self.length^2/12+nzz)); %replace
6Izz/L^2
```

```
Flex_zz3=self.Izz*(self.length^2/3+nzz)/(self.length*(self.length
^2/12+nzz));
           %replace 4Izz/L

Flex_zz4=self.Izz*(self.length^2/6-nzz)/(self.length*(self.length
^2/12+nzz));
           %replace 2Izz/L

Flex_yy1=self.Iyy/(self.length*(self.length^2/12+nyy)); %replace
12Iyy/L^3
          Flex_yy2=self.Iyy/(2*(self.length^2/12+nyy));%replace
6Iyy/L^2

Flex_yy3=self.Iyy*(self.length^2/3+nyy)/(self.length*(self.length
^2/12+nyy));
          %replace 4Iyy/L

Flex_yy4=self.Iyy*(self.length^2/6-nyy)/(self.length*(self.length
^2/12+nyy));
           %replace 2Iyy/L
       self.LocalStiffnessMatrix=self.E*[self.A/self.length 0 0 0 0
0 -self.A/self.length 0 0 0 0 0;
           0 Flex_zz1 0 0 0 Flex_zz2,...
           0 -Flex_zz1 0 0 0 Flex_zz2;
           0 0 Flex_yy1 0 -Flex_yy2 0,...
           0 0 -Flex_yy1 0 -Flex_yy2 0;
           0 0 0 self.J/(2*(1+self.v)*self.length) 0 0,...
           0 0 0 -self.J/(2*(1+self.v)*self.length) 0 0;
           0 0 -Flex_yy2 0 Flex_yy3 0,...
           0 0 Flex_yy2 0 Flex_yy4 0;
           0 Flex_zz2 0 0 0 Flex_zz3,...
           0 -Flex_zz2 0 0 0 Flex_zz4;
           -self.A/self.length 0 0 0 0 0 self.A/self.length 0 0 0 0
0;
           0 -Flex_zz1 0 0 0 -Flex_zz2,...
           0 Flex_zz1 0 0 0 -Flex_zz2;
           0 0 -Flex_yy1 0 Flex_yy2 0,...
           0 0 Flex_yy1 0 Flex_yy2 0;
           0 0 0 -self.J/(2*(1+self.v)*self.length) 0 0,...
           0 0 0 self.J/(2*(1+self.v)*self.length) 0 0;
```

```
            0 0 -Flex_yy2 0 Flex_yy4 0,...
            0 0 Flex_yy2 0 Flex_yy3 0;
            0 Flex_zz2 0 0 0 Flex_zz4,...
            0 -Flex_zz2 0 0 0 Flex_zz3];

self.GlobalStiffnessMatrix=transpose(self.TransformationMatrix)*..
.

self.LocalStiffnessMatrix*self.TransformationMatrix;
        end
  %% Retrieve the DOF of the nodes at both ends
        function RetrieveDOF(self)

self.element_DOF=[self.element_nodes(1).GetNodeDOF();self.element
_nodes(2).GetNodeDOF();];
        end
  %% Compute the fixed end forces
        function ComputeFixedEndForces(self)
          %construct local FEF

FeF_x=[-self.w(1)*self.length/2;0;0;0;0;0;-self.w(1)*self.length/
2;0;0;0;0;0];

FeF_y=[0;-self.w(2)*self.length/2;0;0;0;-self.w(2)*self.length^2/
12;0;...

-self.w(2)*self.length/2;0;0;0;self.w(2)*self.length^2/12];

FeF_z=[0;0;-self.w(3)*self.length/2;0;self.w(3)*self.length^2/12;
0;0;0;...

-self.w(3)*self.length/2;0;-self.w(3)*self.length^2/12;0];
         self.FeF_local=FeF_x+FeF_y+FeF_z;
         %convert FEF from local to global coordinate

self.FeF_global=transpose(self.TransformationMatrix)*self.FeF_loc
al;
        end
    end
end
```

WSLW_Analysis

```
classdef WSLW_Analysis < handle

    properties (Access = private)
        nnodes
        coord
        fixity
        concen
        nele
        ends
        A
        Ayy
        Azz
        Iyy
        Izz
        J
        E
        v
        webdir
        w
        NodeVector
        ElementVector
        StiffnessMatrix
        LoadVector  % sum of concentrated forces and negative fixed
end forces
        free_dof
        disp_dof
        supp_dof
        ConditionNumber
        LostDigits
        Kff
        Ksf
        Kfn
        Knn
        Kns
        Kss
        Pf     % concentrated forces
        Feff
```

```
        Fefs
        Fefn
        Delf    %displacements of free nodes
        DEFL    %nodal displacement in the form of vectors whose number
is nnodes
        REACT   %nodal reaction vector
        ELE_FOR %element internal force vector
        AFLAG   %indicator of whether the analysis is successful
    end

    % Public methods go here
    methods (Access = public)
  %%   Constructor
        function self = WSLW_Analysis(nnodes, coord, fixity, concen,
nele, ends, A, ...
    Ayy, Azz, Iyy, Izz, J, E, v, webdir, w)
            self.nnodes=nnodes;
            self.coord=coord;
            self.fixity=fixity;
            self.concen=concen;
            self.nele=nele;
            self.ends=ends;
            self.A=A;
            self.Ayy=Ayy;
            self.Azz=Azz;
            self.Iyy=Iyy;
            self.Izz=Izz;
            self.J=J;
            self.E=E;
            self.v=v;
            self.webdir=webdir;
            self.w=w;
            self.CreateNodes();    %use this fuction to create a vector
of node objects
            self.CreateElements(); %use this fuction to create a vector
of element objects

        end
  %% Run Analysis
  % use this fuction to call all the fuctions of analyzing the structure
  % from assembling the stiffness matrix to calculating
```

```
  % displacements,reactions and element internal forces
        function RunAnalysis(self)
          self.CreateStiffnessMatrix();
          self.ClassifyDOF();
          self.CreateLoadVector();
          self.ComputeStiffnessSubMatrices();
          self.CheckKffMatrix();
           if self.AFLAG==0 %The structure is unstable, stop the
program
              return;
           end
          self.ComputeDisplacementsReactions();
          self.RecoverElementForces();
          self.ComputeError();
        end
     %% Get the values to return to Mastan2
     % This function is to return the values that Mastan2 need to do
     % pro-precessing
         function [DEFL, REACT, ELE_FOR, AFLAG] =
GetMastan2Returns(self)
                DEFL=self.DEFL;
                REACT=self.REACT;
                ELE_FOR=self.ELE_FOR;
                AFLAG=self.AFLAG;
         end
    end

   % Private methods go here
    methods  (Access=private)
     %% Create  Nodes
        function CreateNodes(self)
           self.NodeVector=WSLW_Node.empty; %access Node class
           for i=1:self.nnodes

self.NodeVector(i,1)=WSLW_Node((self.coord(i,1:3))',i);%create a
object vector for Node
           end
        end
     %% Create Elements
        function CreateElements(self)
           self.ElementVector=WSLW_Element.empty; %access Element
```

```
class
            for i=1:self.nele

EndNodes=[self.NodeVector(self.ends(i,1));self.NodeVector(self.en
ds(i,2))];

self.ElementVector(i,1)=WSLW_Element(EndNodes,self.A(i),self.Izz(
i),self.Iyy(i),self.J(i),...

self.Ayy(i),self.Azz(i),self.E(i),self.v(i),self.webdir(i,1:3),se
lf.w(i,1:3));%create a object vector for Element
            end
        end
         %% Create Stiffness Matrix
        function CreateStiffnessMatrix(self)
            number_of_DOF=6*self.nnodes;

self.StiffnessMatrix=zeros(number_of_DOF,number_of_DOF);%initiate
stiffness matrix
            for i=1:self.nele   %loop over the elements to assemble the
stiffness matrix
               elememt_dof=self.ElementVector(i).GetElementDOF();

self.StiffnessMatrix(elememt_dof,elememt_dof)=self.StiffnessMatri
x(elememt_dof,elememt_dof)...
                   +self.ElementVector(i).GetGlobalStiffnessMatrix;
            end
            self.StiffnessMatrix=sparse(self.StiffnessMatrix); %to
save memory
        end
        %% Create Load Vector
        function CreateLoadVector(self)
            ConcentratedForce=zeros(6*self.nnodes,1);
            for i=1:self.nnodes  % loop over the nodes to assemble the
concentrated forces
               node_DOF=self.NodeVector(i).GetNodeDOF();

ConcentratedForce(node_DOF)=(self.concen(i,1:6))';%substitute
input node forces into a vector form
            end
            self.Pf=ConcentratedForce(self.free_dof);
```

```
            FeF=zeros(6*self.nnodes,1);%initiate FEF vector
            for i=1:self.nele % loop over the elements to assemle the
fixed end forces
                elememt_dof= self.ElementVector(i).GetElementDOF();

FeF(elememt_dof)=FeF(elememt_dof)+self.ElementVector(i).GetFeF();
%map from element level
            end
            %partition FEF vector
            self.Feff=FeF(self.free_dof);
            self.Fefs=FeF(self.disp_dof);
            self.Fefn=FeF(self.supp_dof);
            self.LoadVector=ConcentratedForce-FeF;
             % sum of concentrated forces and negative fixed end forces
        end
        %% classify DOF
        function ClassifyDOF(self)
            fix_t=self.fixity';
            self.free_dof=find(isnan(fix_t)); %free DOF
            self.supp_dof=find(fix_t==0);   %support DOF
            self.disp_dof=find(fix_t~=0 & ~isnan(fix_t)); %displaced
DOF
        end
            %% Extracting Kff Ksf Kfn
              % get different parts of the complete stiffness matrix
            function ComputeStiffnessSubMatrices(self)
                f=self.free_dof;
                n=self.disp_dof;
                s=self.supp_dof;
                %partition Stiffness Matrix
                self.Kff = self.StiffnessMatrix(f,f);
                self.Ksf = self.StiffnessMatrix(s,f);
                self.Kfn = self.StiffnessMatrix(f,n);
                self.Knn = self.StiffnessMatrix(n,n);
                self.Kss = self.StiffnessMatrix(s,s);
                self.Kns = self.StiffnessMatrix(n,s);
            end
            %% Condition of Kff Matrix
            %      AFLAG          ==  logical flag to indicate if a
successful
%                            analysis has been completed
```

```
%                            AFLAG = 1     Successful
%                            AFLAG = 0     Unstable Structure
%                            AFLAG = inf   No analysis code available
          function  CheckKffMatrix(self)
              self.ConditionNumber=cond(self.Kff); %condition
number
              self.LostDigits=log10(cond(self.Kff)); %lost of
digits
              disp('conditional number');
              disp(self.ConditionNumber);
              disp('lost of significant digits');
              disp(self.LostDigits);

              if self.LostDigits>=12 %unstable if lost more than 12
digits
                  self.AFLAG=0;
                  disp('Unstable Structure')
              else
                  self.AFLAG=1;
                  disp('Stable Structure')
              end

          end
        %%  Compute Nodal Displacements and Reactions
            function ComputeDisplacementsReactions(self)
                fix_t=self.fixity'; %transpose fixity matrix to use
linear indexing
                Deln = fix_t(self.disp_dof);%support displacement
                self.Delf = self.Kff \ (self.Pf -
self.Kfn*Deln-self.Feff); %calculate displacement in free DOF

                DEFL_t = zeros (6,self.nnodes);
                DEFL_t(self.free_dof) = self.Delf;%free displacement
                DEFL_t(self.disp_dof)
=DEFL_t(self.disp_dof)+Deln;%plus support displacement
                self.DEFL=DEFL_t';

                REACT_t = zeros (6,self.nnodes);
                REACT_t(self.disp_dof) =
self.Kfn'*self.Delf+self.Knn*Deln-self.LoadVector(self.disp_dof);
%reaction in displaced DOF
```

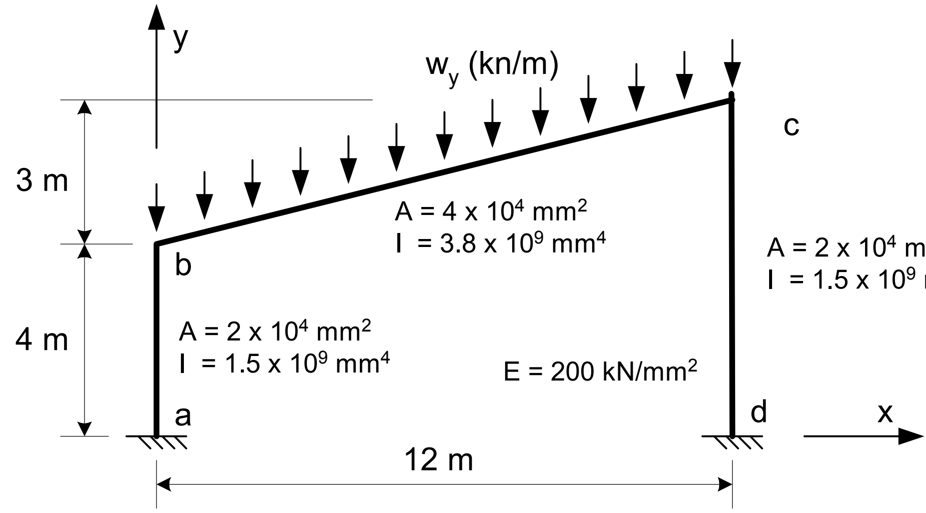
```
                REACT_t(self.supp_dof) =
self.Ksf*self.Delf+self.Kns'*Deln-self.LoadVector(self.supp_dof);
%reaction in support DOF
                self.REACT=REACT_t';
            end
        %%  Recover Element Internal Forces
            function RecoverElementForces(self)
                %initiate displacement and element force vectors
                displacement=zeros(12,1);
                self.ELE_FOR=zeros(self.nele,12);
                for i=1:self.nele
                    element_dof=
self.ElementVector(i).GetElementDOF();
                    DEFL_t=self.DEFL';
                    displacement= DEFL_t(element_dof); %map element
displacement using element dof
                    %call computeforce function in element class to
                    %compute local element force
                    ElementForces =
self.ElementVector(i,1).ComputeForces(displacement);
                    self.ELE_FOR(i,:)=ElementForces';
                end
            end

       %%  Compute load vector using the computed displacements
            function  ComputeError(self)
                DEFL_t=self.DEFL';
                Displacement=zeros(6*self.nnodes,1);% make
calculated displacement as a column vector
                Displacement(:)=DEFL_t(:);

BackLoadVector=self.StiffnessMatrix*Displacement;%back calculate
load vector with obtained displacement

Error=BackLoadVector(self.free_dof)-self.LoadVector(self.free_dof)
;  %calculate error
                disp('Error');
                disp(Error);
            end
    end
end
```

ud_3d1el

```
%%Node Class: contains properties related to nodes, such as node
%%coordinate, node DOF

%%Element Class: contains properties related to elements. Element
stiffness
%%matrix, element length, element force, element fixed end forces are
%%computed in this class.

%%Analysis Class: assembles imformation from the previous two
classes,
%%performs linear analysis, solve for displacements, reactions, and
recover
%%element forces. It also checks the condition of stiffness matrix
and
%%computes the error of the solved displacements. The analyzed data
get to
%%return to MASTAN2
function [DEFL,REACT,ELE_FOR,AFLAG] = ud_3d1el(...

nnodes,coord,concen,fixity,nele,ends,A,Izz,Iyy,J,Cw,Zzz,Zyy,Ayy,A
zz,...
       E,v,Fy,YldSurf,Wt,webdir,beta_ang,w,thermal,truss,anatype)

   analysis=WSLW_Analysis(nnodes, coord, fixity, concen, nele, ends,
A, Ayy, Azz, Iyy, Izz, J, E, v, ...
       webdir, w); %compute analysis calculation in the analysis class

   analysis.RunAnalysis();%run analysis by calling the runanalysis
method in the analysis class
   [DEFL, REACT, ELE_FOR, AFLAG] =
analysis.GetMastan2Returns();%returned computed solutions to
MASTAN2
end
```

Verification Problems

Verification Problem 1:



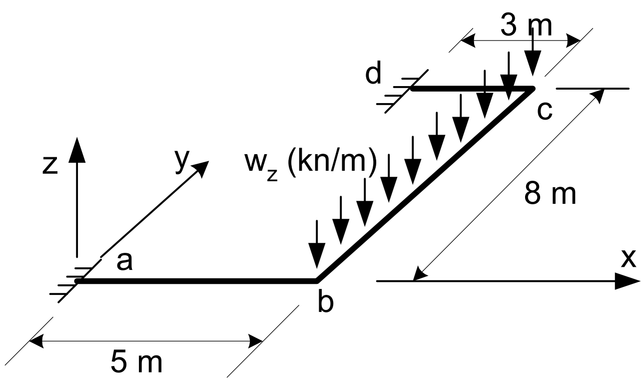
Notes:

1) **The load W_y = 15 kN/m is a vertical distributed load along the length of the member, which you will need to convert to equivalent amounts of distributed load in the local x' and y' axis of the member.**
2) **$I = I_{yy} = I_{zz}$**
3) **J = 5 x 10^9 mm^4**

Report the following information:

- **Deflections at point b (Δx, Δy, θz)**
- **Reactions at point a (Fx, Fy, Mz)**
- **Condition number (estimate of loss in significant digits) of *K_{ff}***
- **Sketch of bending moment diagram showing numeric values at member ends and midspan of b-c.**

Verification Problem 2:



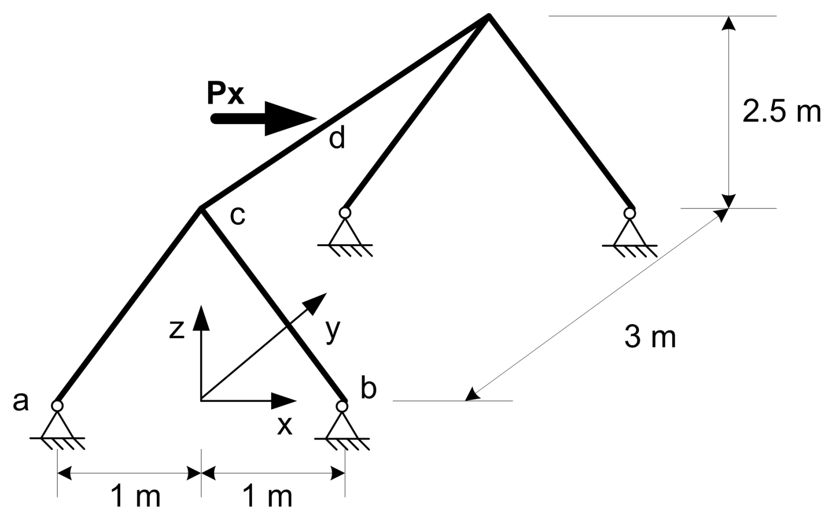
Notes:

1) **The structure consists of a horizontal grid of rectangular tubular members measuring 100 x 300 mm square. The members are all oriented with their tall dimension parallel to the global z-axis (vertical direction). The tubular members have the following properties: A = 11,000 mm^2, Imajor = 1.06 x 10^8 mm^4, Iminor = 1.74 x 10^7 mm^4 , J = 5.29 x 10^7 mm^4**
2) **Members are steel with E = 200 kN/mm^2 and ν = 0.3.**
3) **The load W_z = 5 kN/m is a vertical distributed load along the length of the member.**

Report the following information:

- **Deflections at point b (Δx, Δy, Δz, θz, θy, θz)**
- **Reactions at point a (Fx, Fy, Fz, Mx, My, Mz)**
- **Value of torsion (Mx') in member a-b.**
- **Condition number (estimate of loss in significant digits) of K_{ff}**
- **Sketch diagram of major axis bending for each member with key numerical values indicated.**

Verification Problem 3 – Swing Set:



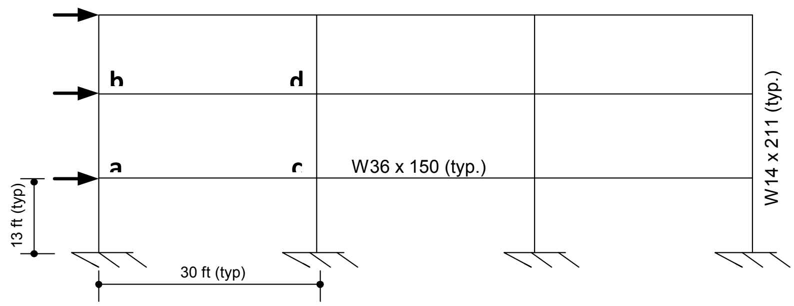
Notes:

1) **The structure is built with round 75 mm diameter tubular members have the following properties: A = 1,430 mm^2, I = 1.26 x 10^6 mm^4, J = 2.52 x 10^6 mm^4**
2) **Members are steel with E = 200 kN/mm^2 and ν = 0.3.**
3) **The load P_x = - 4.5 kN (opposite direction of that shown in the figure)**

Report the following information:

- **Deflections at point d (Δx, Δy, Δz, θz, θy, θz)**
- **Reactions at points a and b (Fx, Fy, Fz, Mx, My, Mz)**
- **Condition number (estimate of loss in significant digits) of K_{ff}**
- **Axial forces in members a-c, c-b, and c-d.**

Verification Problem 4:



Notes:

1) **Members have the following properties:**

   W36 x 150: A=44.2 in^2, I = 9,040 in^4, A_{web} = 22.4 in^2

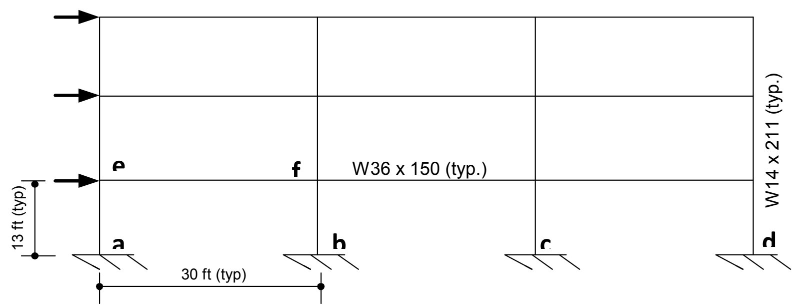
   W14 x 211: A=62.0 in^2, I = 2,660 in^4, A_{web} = 15.7 in^2

2) **Members are steel with E = 30,000 k/in^2 and ν = 0.3.**
3) **The applied lateral load at each floor is Px = 9.5 kips**
4) **Base your analysis on centerline dimensions (i.e., ignoring finite joint size effects).**

Perform two lateral load analyses, one in which shear deformations are included and one in which they are excluded. Report the following information for each analysis

- **Lateral deflections at each floor at the second column from the left (Δx1, Δx2, Δx3)**
- **The maximum moments in column a-b and beam b-d.**
- **Condition number (estimate of loss in significant digits) of K_{ff}**
- **What is the percentage change in lateral deflections due to the shear deformations?**
- **What is the percentage change in the *maximum* beam and column moments due to shear deformations?**

Verification Problem 5:



Notes:

1) **This is the same structure as for Problem 4.**
2) **For this problem, do NOT include member shear deformations.**

Perform an analysis where you apply a vertical settlement of Δ = -1 inch to the support at point b. Report the following information from this analysis.

- **Base reactions at points a, b, c, and d (Fx, Fy, M)**
- **Shear and moments in beam e-f (V, Me, Mf).**

FILL OUT THE BLANKS AND INCLUDE THESE SHEETS (or equivalents) WITH YOUR FINAL REPORT.

Verification Problem 1

Deflections at points b and c

point b [WRITE UNITS]

	MASTAN results	your results
Δx	0.745 mm	0.745 mm
Δy	-0.09817 mm	-0.09817 mm
θz	-0.0006226 rad	-0.0006226 rad

Reactions at point a

point a [WRITE UNITS]

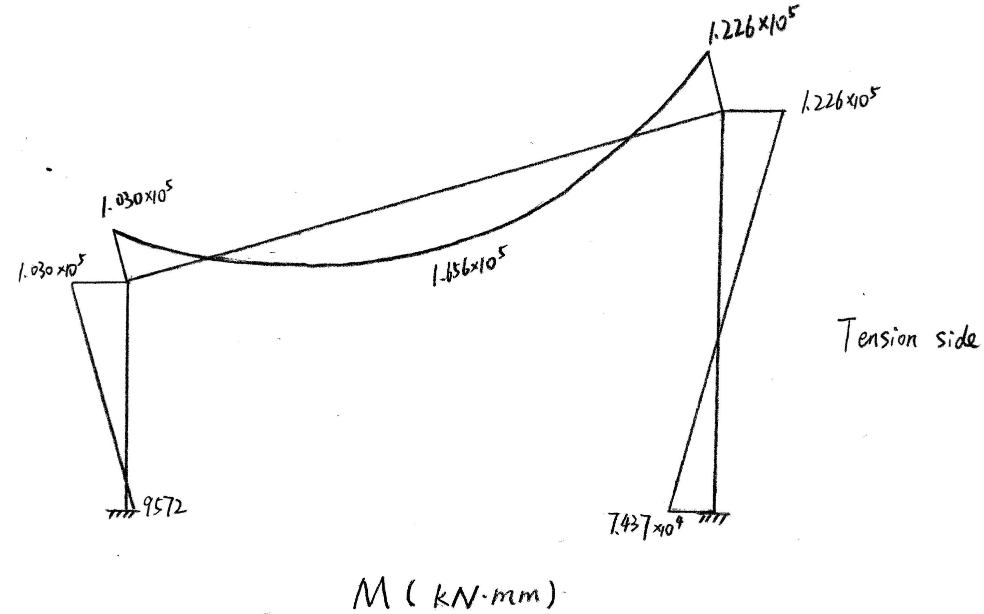
	MASTAN results	your results
Fx	28.13 kN	28.13 kN
Fy	98.17 kN	98.17 kN
Mz	-9572 kN-mm	-9572 kN-mm

point d [WRITE UNITS]

	MASTAN results	your results
Fx	-28.13 kN	-28.13 kN
Fy	87.37 kN	87.37 kN
Mz	74370 kN-mm	74370 kN-mm

Log10 of condition number of Kff: ____8.3124______

Sketch of bending moment diagram



Verification Problem 2

Deflections at point b

point b [WRITE UNITS]

	MASTAN results	your results
Δx	0	0
Δy	0	0
Δz	-35.5 mm	-35.5 mm
θx	-0.001078 rad	-0.001078 rad
θy	0.01048 rad	0.01048 rad
θz	0	0

Reactions at point a

point a [WRITE UNITS]

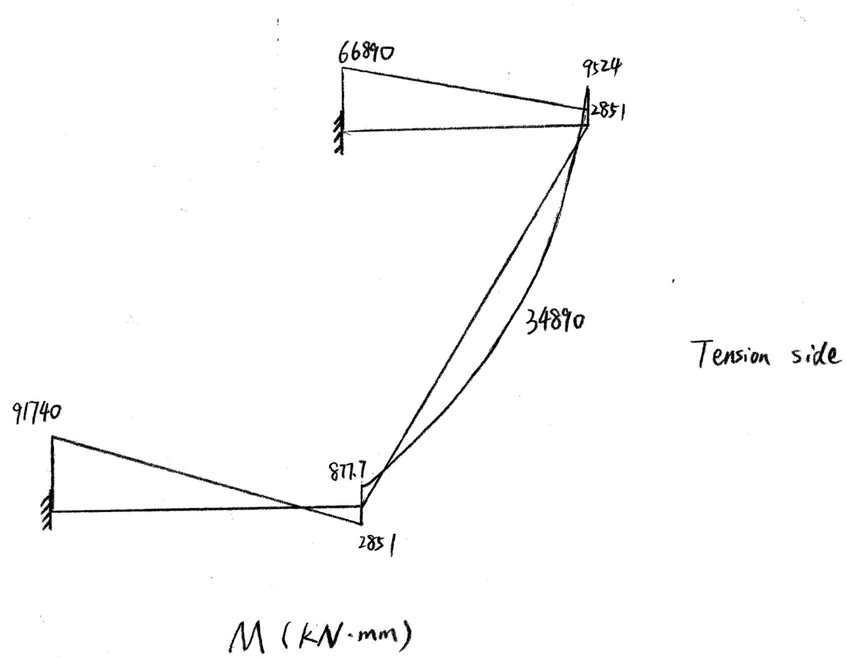
	MASTAN results	your results
Fx	0	0
Fy	0	0
Fz	18.92 kN	18.92 kN
Mx	877.7 kN-mm	877.7 kN-mm
My	-91740 kN-mm	-91740 kN-mm
Mz	0	0

Value of torsion Mx'

Member:	MASTAN results	your results
a-b	-877.7 kN-mm	-877.7 kN-mm

Log10 of condition number of Kff : ____7.7790_____

Sketch of bending moment diagram



Verification Problem 3

Deflections at point d

point d

	MASTAN results	your results
Δx	-7.744 mm	-7.744 mm
Δy	2.025E-15 mm	7.207E-16 mm
Δz	-7.998E-17 mm	6.419E-17 mm
θx	2.458E-19 rad	9.902E-21 rad
θy	-2.648E-5 rad	-2.648E-5 rad
θz	1.331E-19 rad	4.394E-19 rad

Reactions at points a and b

point a

	MASTAN results	your results
Fx	1.125 kN	1.125 kN
Fy	0.2662 kN	0.2662 kN
Fz	2.813 kN	2.813 kN
Mx	0	0
My	0	0
Mz	0	0

point b

	MASTAN results	your results
Fx	1.125 kN	1.125 kN
Fy	-0.2662 kN	-0.2662 kN
Fz	-2.813 kN	-2.813 kN
Mx	0	0
My	0	0
Mz	0	0

Log10 of condition number of Kff: ____ 7.9140_______

Axial forces

	MASTAN results	your results
a-c	-3.029 kN	-3.029 kN
c-b	3.029 kN	3.029 kN
c-d	0	0

Verification Problem 4

a) include shear deformation

Lateral deflections at each floor level

	MASTAN results	your results
Δx3 (Roof)	0.1107 in	0.1107 in
Δx2 (3F)	0.0884 in	0.0884 in
Δx1 (2F)	0.04627 in	0.04627 in

The maximum moments

	MASTAN results	your results
column a-b	324.6 in-kips	324.6 in-kips
beam b-d	429.5 in-kips	429.5 in-kips

Log10 of condition number of Kff: _____   7.4557________

b) exclude shear deformation

Lateral deflections at each floor level

	MASTAN results	your results
Δx3 (Roof)	0.09485 in	0.09485 in
Δx2 (3F)	0.07561 in	0.07561 in
Δx1 (2F)	0.0392 in	0.0392 in

The maximum moments

	MASTAN results	your results
column a-b	316.2 in-kips	316.2 in-kips
beam b-d	422.4 in-kips	422.4 in-kips

Log10 of condition number of Kff: _______7.5137_________

The percentage change in lateral deflection change

[= 100 * (Δx_include - Δx_exclude)/ Δx_include]

	your results
Δx3 (Roof)	**14.32%**
Δx2 (3F)	**14.47%**
Δx1 (2F)	**15.28%**

The percentage change in the maximum moments

[= 100 * (M_include - M_exclude)/ M_include]

	your results
column a-b	**2.59%**
beam b-d	**1.65%**

Verification Problem 5

Base reactions at points a b c and d

point a

	MASTAN results	your results
Fx	22.76 kips	22.76 kips
Fy	125 kips	125 kips
Mz	-844.6 in-kips	-844.6 in-kips

point b

	MASTAN results	your results
Fx	-11.9 kips	-11.9 kips
Fy	-285.2 kips	-285.2 kips
Mz	982.2 in-kips	982.2 in-kips

point c

	MASTAN results	your results
Fx	-34.1 kips	-34.1 kips
Fy	178.8 kips	178.8 kips
Mz	2160 in-kips	2160 in-kips

point d

	MASTAN results	your results
Fx	-5.27 kips	-5.27 kips
Fy	-18.61 kips	-18.61 kips
Mz	654 in-kips	654 in-kips

Shear and moments in beam e-f

	MASTAN results	your results
V	43.17 kips	43.17 kips
Me	-6480 in-kips	-6480 in-kips
Mf	9062 in-kips	9062 in-kips

Team Responsibility Summary

The team works in a highly efficient and collaborative manner. Before each interim, the team first meets and discusses the objectives of the assignment. Each team member then is responsible for writing the assigned methods on his own. The team then meets again to assemble the finished codes and test the results with diagnostic tools or MASTAN2. The team will also read through each other's codes checking for programming styles and efficiency. The team concludes that both members improve their programing techniques a lot through this assignment.

The documentation works are also divided equally. The team uses Google drive to share finished documents so that each member will know the progress of the project. The table below summarizes the responsibility divided among the team.

Table 1. Summary of work and responsibility divided

	Task	Team member
Interim 1	Methods in Node Class	Wenjin
	Methods in Element Class	Liyi
	Debug & Testing	Wenjin & Liyi
Interim 2	Extending Node Class	Liyi
	Extending Element Class	Wenjin
	Defining Analysis Class	Liyi
	Debug & Testing	Wenjin & Liyi
	First draft of code design document	Wenjin
Interim 3	Extending Analysis Class	Wenjin & Liyi
	Methods for recovering element force	Wenjin & Liyi
	Debug & Testing	Liyi
	Method that returns computed data to Mastan2	Liyi
Final Submission	2-3 pages short summary of the program	Wenjin
	Code design document flowchart	Liyi
	Listing of program and commented ud_3dlel file	Wenjin
	Verification problems	Liyi
	Work responsibility summary	Wenjin
	Self-assessment of effort	Wenjin & Liyi