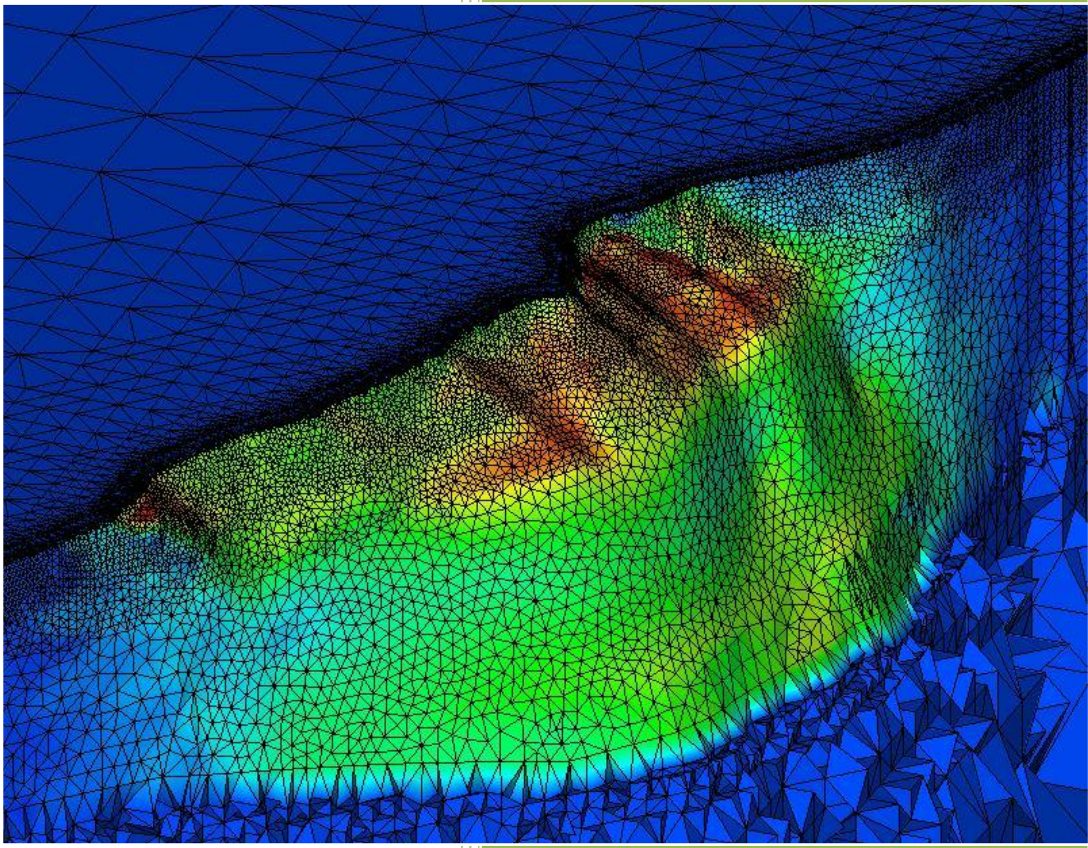
EngiSSol
Engineering Software Solutions

3D Frame Analysis Library Technical Notes & Examples



Static, Dynamic, Linear and Non-Linear 3D Structural Analysis for frames and shells

Contents

Introduction

The finite element method (FEM) (sometimes referred to as finite element analysis) is a numerical technique for finding approximate solutions of partial differential equations (PDE) as well as of integral equations. The solution approach is based either on eliminating the differential equation completely (steady state problems), or rendering the PDE into an approximating system of ordinary differential equations, which are then numerically integrated using standard techniques such as Euler's method, Runge-Kutta, etc.

In solving partial differential equations, the primary challenge is to create an equation that approximates the equation to be studied, but is numerically stable, meaning that errors in the input data and intermediate calculations do not accumulate and cause the resulting output to be meaningless. There are many ways of doing this, all with advantages and disadvantages. The Finite Element Method is a good choice for solving partial differential equations over complex domains (like cars and oil pipelines), when the domain changes (as during a solid state reaction with a moving boundary), when the desired precision varies over the entire domain, or when the solution lacks smoothness.

ENGISSOL, as a leader company in finite element programming, has launched many finite element libraries which are continuously enriched by new contemporary arithmetic techniques and optimized in order to come up to any complex engineering simulation. Among these libraries, ENGISSOL’s R&D department has created a commercial library which can perform 3D finite element analysis for frames and buildings very easily with great accuracy and reliability. This library has been developed in the modern programming environment of MS Visual Studio 2008 and is compatible with almost every programming interface. The integration of 3D Frame Analysis Library to a programming interface can result into a complete, high quality and competitive finite element application.

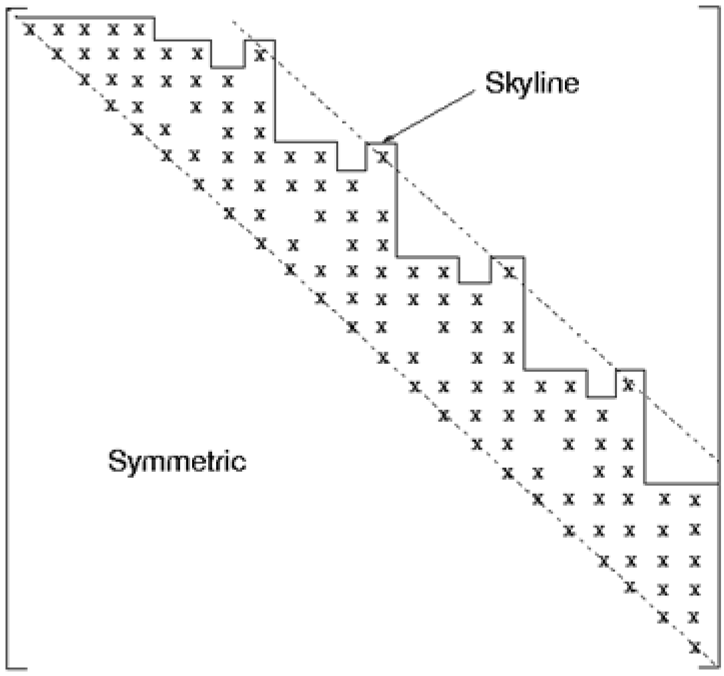
The scope of this paper is to provide theoretical and also practical information about the library’s assumptions, as well as a comprehensive description of the adapted methods and algorithms. Reference to finite element analysis theory will be made if necessary. In any case, the reader is advised to refer to a general finite element book in order to get familiar enough with the philosophy of the finite element method and particularly 3D Frame Analysis Library. Furthermore, reference to the library’s classes, objects, methods etc will be made if needed.

Basic theoretical background

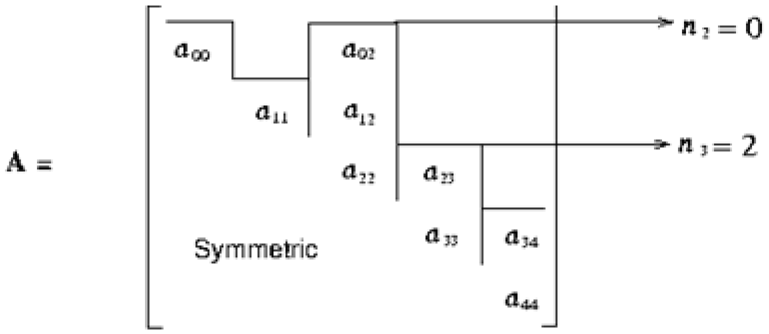
Skyline storage scheme

A skyline matrix, or a variable band matrix, is a form of a sparse matrix storage format for a square, banded (and typically symmetric) matrix that reduces the storage requirement of a matrix more than banded storage. In banded storage, all entries within a fixed distance from the diagonal (called half-bandwidth) are stored. In column oriented skyline storage, only the entries from the first nonzero entry to the last nonzero entry in each column are stored. There is also row oriented skyline storage, and, for symmetric matrices, only one triangle is usually stored.

Skyline storage has become very popular in the finite element codes for structural mechanics, because the skyline is preserved by Cholesky decomposition (a method of solving systems of linear equations with a symmetric, positive-definite matrix; all fill-in falls within the skyline), and systems of equations from finite elements have a relatively small skyline. In addition, the effort of coding skyline Cholesky is about same as for Cholesky for banded matrices.



An example of the skyline storage scheme follows in the next picture.



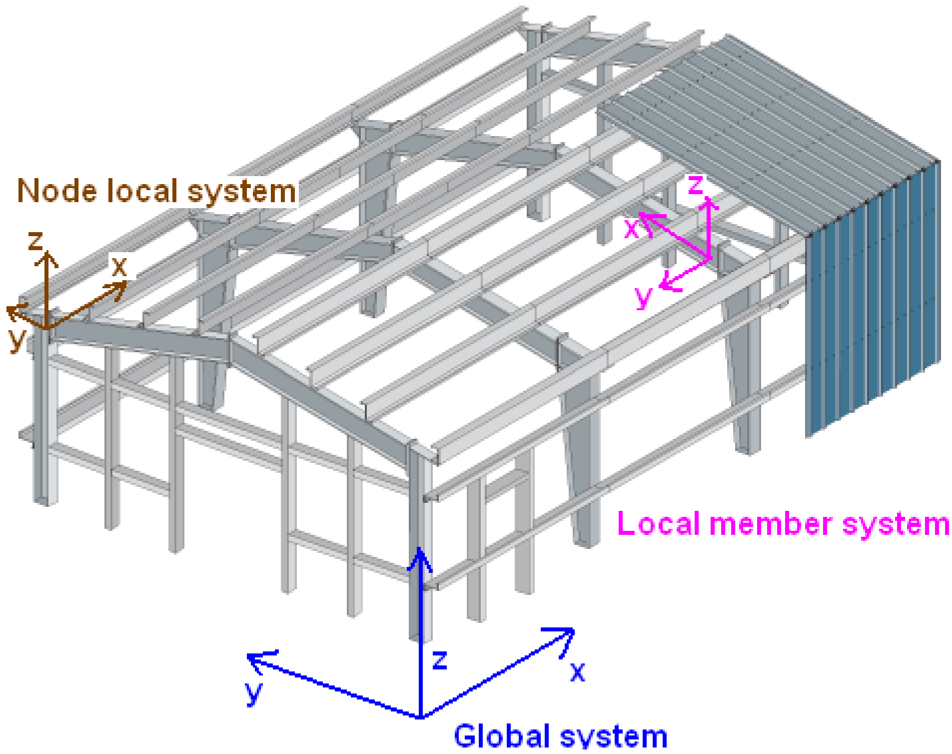
$$\bar{A} = \begin{array}{c} \begin{matrix} \bar{a}_0 & \bar{a}_1 & \bar{a}_2 & \bar{a}_3 & \bar{a}_4 & \bar{a}_5 & \bar{a}_6 & \bar{a}_7 & \bar{a}_8 \end{matrix} \\ \begin{bmatrix} a_{00} & a_{11} & a_{22} & a_{12} & a_{02} & a_{33} & a_{23} & a_{44} & a_{34} \end{bmatrix} \end{array}$$

$$MAX\bar{A} = \begin{bmatrix} 0 & 1 & 2 & 5 & 7 \end{bmatrix}$$

3D Frame Analysis Library uses this storage technique at all cases where symmetric and positive defined matrices are to be stored, in order to minimize computer memory usage and accelerate the solution speed as much as possible.

Coordinate systems

Three different coordinate systems are available in 3D Frame Analysis Library. The global system and two local ones, the element local and node local system. It has to be noted that these three coordinate systems result into more flexibility and ease in creating the structural model, since data as loads, boundary conditions etc can be defined at the desired system, whereas analysis results are obtained in each corresponding coordinate system.



Global system

The global coordinate system remains constant for each element, node and generally the complete model.

Element local system

For each element (frame etc), a local system is assigned by rotating the global one according to followings:

- Local x axis is defined from element's starting to its ending node
- Local y axis is defined by an auxiliary point that lies on the plane that is formed by the local x and y element axes
- Local z axis is defined as perpendicular to x and y local axes, so that a right hand side coordinate system is formed.

Node local system

Generally the local system of a node matches the global system unless otherwise defined. Local system of a node is defined the same way as the element local system.

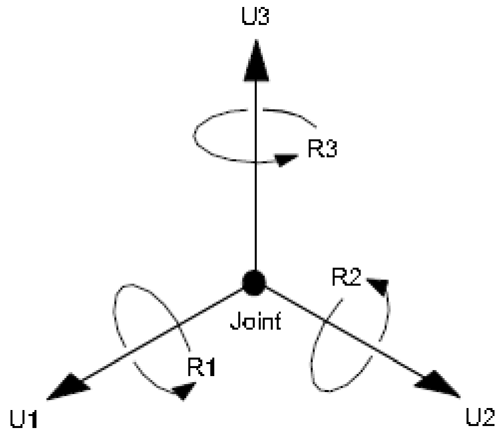
Model data and corresponding coordinate system

Nodal loads	Node local system

Nodal reactions	Node local system
Nodal displacements	Node local system
Prescribed displacements	Node local system
Member loads	Element local and global system (as specified)
Member end releases	Element local system
Response spectrum excitation (ground motion direction)	Global system
Member internal forces and displacements	Element local system
Diaphragm loads	Global system
Diaphragm displacements	Global system

Degrees of freedom

3D Frame Analysis Library features 6 degrees of freedom per node as indicated below. Each degree of freedom can be fully or partially (by springs) constrained. Furthermore, in case of frame elements, each set of degrees of freedom can be released unless a mechanism is formed. The ability of partial releases is also available in 3D Frame Analysis Library.



Load combination

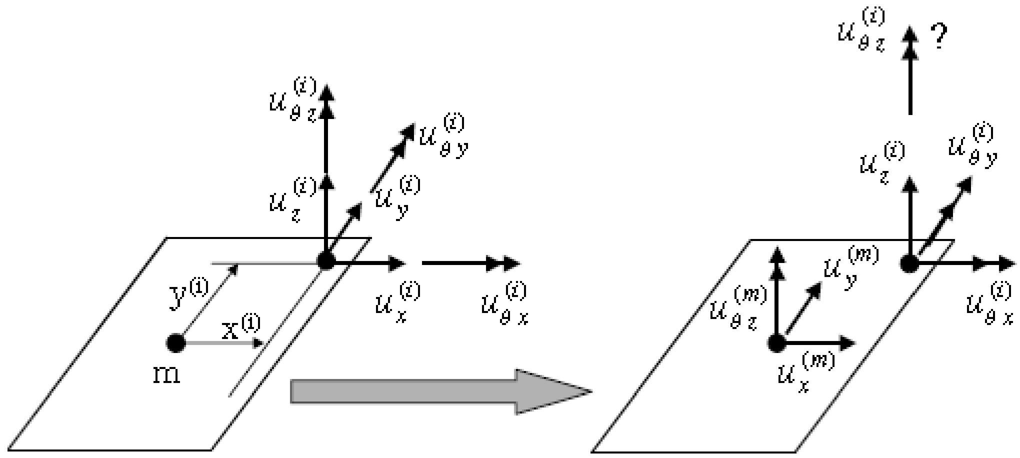
The following load combination types are supported in 3D Frame Analysis Library:

- **Linear Add**: All load case results are multiplied by their scale factor and added together.
- **Envelope**. A max/min Envelope of the defined load cases is evaluated for each frame output segment and object joint. The load cases that give the maximum and minimum components are used for this combo. Therefore the load Combo holds two values for each output segment and joint.
- **Absolute Add**: The absolute of the individual load case results are summed and positive and negative values are automatically produced for each output segment and joint.
- **SRSS**: The Square Root Sum of the Squares calculation is performed on the load cases and positive and negative values are automatically produced for each output segment and joint.
- **CQC**: The Complete Quadratic Combination is used in case of coupled modes combination. Modes are generally coupled in ordinary building structures so this method is used as an improvement on SRSS.

It should be noted that in case of Modal analysis, only one of the last two combination methods (SRSS, CQC) can be used, since the remaining do not have a meaning when combining dynamic modes.

Rigid diaphragm constrained

Many automated structural analysis computer programs use master-slave constraint options. However, in many cases the user's manual does not clearly define the mathematical constraint equations that are used within the program. To illustrate the various forms that this constraint option can take, let us consider the floor diaphragm system shown below.



A. Typical Joint "I" on Floor System in x-y Plane   *B. Master-Slave Constraints*

The diaphragm, or the physical floor system in the real structure, can have any number of columns and beams connected to it. At the end of each member, at the diaphragm level, six degrees of freedom exist for a three-dimensional structure before introduction of constraints. Field measurements have verified for a large number of building-type structures that the in plane deformations in the floor systems are small compared to the inter-story horizontal displacements. Hence, it has become common practice to assume that the in-plane motion of all points on the floor diaphragm move as a rigid body. Therefore, the in-plane displacements of the diaphragm can be expressed in terms of two displacements, (m) $u_x^{(m)}$ and $u_y^{(m)}$, and a rotation about the z-axis, $u_{z\theta}^{(m)}$. In the case of static loading, the location of the master node (m) can be at any location on the diaphragm. However, for the case of dynamic earthquake loading, the master node must be located at the center of mass of each floor if a diagonal mass matrix is to be used. 3D Frame Analysis Library automatically calculates the location of the master node based on the center of mass of the constraint nodes. As a result of this rigid diaphragm approximation, the following compatibility equations must be satisfied for joints attached to the diaphragm:

$u_x^{(i)} = u_x^{(m)} - y^{(i)}\, u_{z\theta}^{(m)}$

$u_y^{(i)} = u_Y^{(m)} + x^{(i)}\, u_{z\theta}^{(m)}$

$u_{z\theta}^{(i)} = u_{z\theta}^{(m)}$

Or in matrix form, the displacement transformation is:

$$\begin{bmatrix} u_x^{(i)} \\ u_y^{(i)} \\ u_{\theta z}^{(i)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -y^{(i)} \\ 0 & 1 & x^{(i)} \\ 0 & 0 & u_{\theta z}^{(i)} \end{bmatrix} \begin{bmatrix} u_x^{(m)} \\ u_y^{(m)} \\ u_{\theta z}^{(m)} \end{bmatrix} \text{ or, } \mathbf{u}^{(i)} = \mathbf{T}^{(i)}\,\mathbf{u}^{(m)}$$

If displacements are eliminated by the application of constraint equations, the loads associated with those displacements must also be transformed to the master node. From simple statics the loads applied at joint "i" can be moved to the master node "m" by the following equilibrium equations:

$R_x^{(mi)} = R_x^{(i)}$

$R_y^{(mi)} = R_y^{(i)}$

$R_{\theta z}^{(mi)} = R_{\theta z}^{(i)} - y^{(i)} R_x^{(i)} + x^{(i)} R_y^{(i)}$

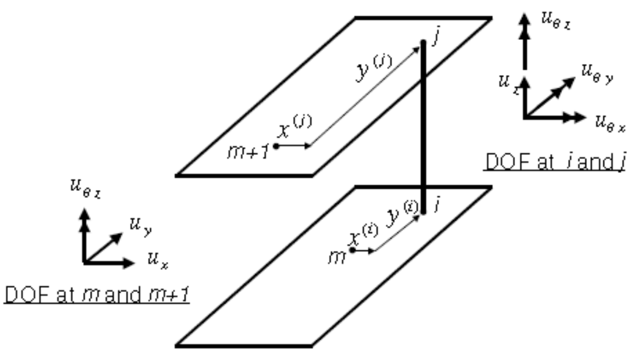
Or in matrix form the load transformation is:

$$\begin{bmatrix} R_x^{(mi)} \\ R_y^{(mi)} \\ R_{\theta z}^{(mi)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -y^{(i)} & x^{(i)} & 1 \end{bmatrix} \begin{bmatrix} R_x^{(i)} \\ R_y^{(i)} \\ R_{\theta z}^{(i)} \end{bmatrix} \text{ Or, } \mathbf{R}^{(mi)} = \mathbf{T}^{(i)^T}\mathbf{R}^{(i)}$$

Again, one notes that the force transformation matrix is the transpose of the displacement transformation matrix. The total load applied at the master point will be the sum of the contributions from all slave nodes, or:

$$R^{(m)} = \sum_i R^{(mi)} = \sum T^{(i)^T} R^{(i)}$$

Now, consider a vertical column connected between joint i at level m and joint j at level m+1, as shown below. Note that the location of the master node can be different for each level.



It is apparent that the displacement transformation matrix for the column is given by

$$
\begin{bmatrix} u_x^{(i)} \\ u_y^{(i)} \\ u_z^{(i)} \\ u_{\theta x}^{(i)} \\ u_{\theta y}^{(i)} \\ u_{\theta z}^{(i)} \\ u_x^{(j)} \\ u_y^{(j)} \\ u_z^{(j)} \\ u_{\theta x}^{(j)} \\ u_{\theta y}^{(j)} \\ u_{\theta z}^{(j)} \end{bmatrix}
=
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & -y^{(i)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & x^{(i)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -y^{(j)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & x^{(j)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix} u_x^{(m)} \\ u_y^{(m)} \\ u_z^{(i)} \\ u_{\theta x}^{(i)} \\ u_{\theta y}^{(i)} \\ u_{\theta z}^{(i)} \\ u_{\theta z}^{(m)} \\ u_x^{(m+1)} \\ u_y^{(m+1)} \\ u_z^{(i)} \\ u_{\theta x}^{(i)} \\ u_{\theta y}^{(i)} \\ u_{\theta z}^{(i)} \\ u_{\theta z}^{(m+1)} \end{bmatrix}
$$

Or in symbolic form:

D = **B u**

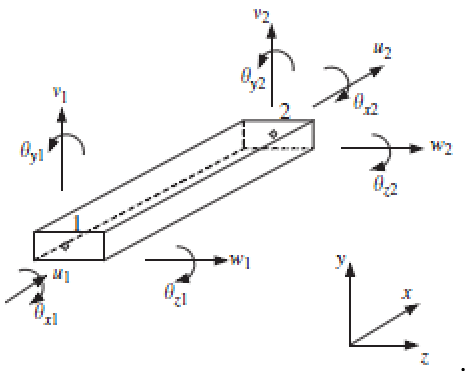
The displacement transformation matrix is 12 by 14 if the z-rotations are retained as independent displacements. The new 14 by 14 stiffness matrix, with respect to the master and slave reference systems at both levels, is given by:

K = $\mathbf{B}^{\mathbf{T}}$ **k B**,

where k is the initial 12 by 12 global stiffness matrix for the column.

The frame element

The approach used to develop the two-dimensional frame elements can be used to develop the three-dimensional frame elements as well. The only difference is that there are more DOFs at a node in a 3D frame element than there are in a 2D frame element. There are altogether six DOFs at a node in a 3D frame element: three translational displacements in the x, y and z directions, and three rotations with respect to the x, y and z axes. Therefore, for an element with two nodes, there are altogether twelve DOFs, as shown in Figure below.



Equations in Local Coordinate System

The element displacement vector for a frame element in space can be written as.

$$\mathbf{d}_e = \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \\ d_8 \\ d_9 \\ d_{10} \\ d_{11} \\ d_{12} \end{Bmatrix} = \begin{Bmatrix} u_1 \\ v_1 \\ w_1 \\ \theta_{x1} \\ \theta_{y1} \\ \theta_{z1} \\ u_2 \\ v_2 \\ w_2 \\ \theta_{x2} \\ \theta_{y2} \\ \theta_{z2} \end{Bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} u_1 \\ v_1 \\ w_1 \\ \theta_{x1} \\ \theta_{y1} \\ \theta_{z1} \end{matrix}}\right\} \text{displacement components at node 1} \\ \left.\vphantom{\begin{matrix} u_2 \\ v_2 \\ w_2 \\ \theta_{x2} \\ \theta_{y2} \\ \theta_{z2} \end{matrix}}\right\} \text{displacement components at node 2} \end{matrix}$$

The element matrices can be obtained by a similar process of obtaining the matrices of the truss element in space and that of beam elements, and adding them together. Because of the huge matrices involved, the details will not be shown herein, but the stiffness matrix is listed here as follows, and can be easily confirmed simply by inspection:

$$
\mathbf{k}_e = \begin{array}{c}
\begin{array}{cccccccccccc}
u_1 & v_1 & w_1 & \theta_{x1} & \theta_{y1} & \theta_{z1} & u_2 & v_2 & w_2 & \theta_{x2} & \theta_{y2} & \theta_{z2} \\
\uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow
\end{array} \\
\begin{bmatrix}
\frac{AE}{2a} & 0 & 0 & 0 & 0 & 0 & \frac{-AE}{2a} & 0 & 0 & 0 & 0 & 0 \\
 & \frac{3EI_z}{2a^3} & 0 & 0 & 0 & \frac{3EI_z}{2a^2} & 0 & \frac{-3EI_z}{2a^3} & 0 & 0 & 0 & \frac{3EI_z}{2a^2} \\
 & & \frac{3EI_y}{2a^3} & 0 & \frac{-3EI_y}{2a^2} & 0 & 0 & 0 & \frac{-3EI_y}{2a^3} & 0 & \frac{-3EI_y}{2a^2} & 0 \\
 & & & \frac{GJ}{2a} & 0 & 0 & 0 & 0 & 0 & \frac{-GJ}{2a} & 0 & 0 \\
 & & & & \frac{2EI_y}{a} & 0 & 0 & 0 & \frac{3EI_y}{2a^2} & 0 & \frac{EI_y}{a} & 0 \\
 & & & & & \frac{2EI_z}{a} & 0 & \frac{-3EI_z}{2a^2} & 0 & 0 & 0 & \frac{EI_z}{a} \\
 & & & & & & \frac{AE}{2a} & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & \frac{3EI_z}{2a^3} & 0 & 0 & 0 & \frac{-3EI_z}{2a^2} \\
 & & sy. & & & & & & \frac{3EI_y}{2a^3} & 0 & \frac{3EI_y}{2a^2} & 0 \\
 & & & & & & & & & \frac{GJ}{2a} & 0 & 0 \\
 & & & & & & & & & & \frac{2EI_y}{a} & 0 \\
 & & & & & & & & & & & \frac{2EI_z}{a}
\end{bmatrix}
\end{array}
$$

where I_y and I_z are the second moment of area (or moment of inertia) of the cross-section of the beam with respect to the y and z axes, respectively. Note that the fourth DOF is related to the torsional deformation. The development of a torsional element of a bar is very much the same as that for a truss element. The only difference is that the axial deformation is replaced by the torsional angular deformation, and axial force is replaced by torque. Therefore, in the resultant stiffness matrix, the element tensile stiffness AE/l_e is replaced by the element torsional stiffness GJ/l_e, where G is the shear modules and J is the polar moment of inertia of the cross-section of the bar. The mass matrix is also shown as follows:

$$
\mathbf{m}_e = \frac{\rho A a}{105}\begin{bmatrix}
70 & 0 & 0 & 0 & 0 & 0 & 35 & 0 & 0 & 0 & 0 & 0 \\
 & 78 & 0 & 0 & 0 & 22a & 0 & 27 & 0 & 0 & 0 & -13a \\
 & & 78 & 0 & -22a & 0 & 0 & 0 & 27 & 0 & 13a & 0 \\
 & & & 70r_x^2 & 0 & 0 & 0 & 0 & 0 & -35r_x^2 & 0 & 0 \\
 & & & & 8a^2 & 0 & 0 & 0 & -13a & 0 & -6a^2 & 0 \\
 & & & & & 8a^2 & 0 & 13a & 0 & 0 & 0 & -6a^2 \\
 & & & & & & 70 & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & 78 & 0 & 0 & 0 & -22a \\
 & & & & & & & & 78 & 0 & 22a & 0 \\
 & & & sy. & & & & & & 70r_x^2 & 0 & 0 \\
 & & & & & & & & & & 8a^2 & 0 \\
 & & & & & & & & & & & 8a^2
\end{bmatrix}
$$

Where

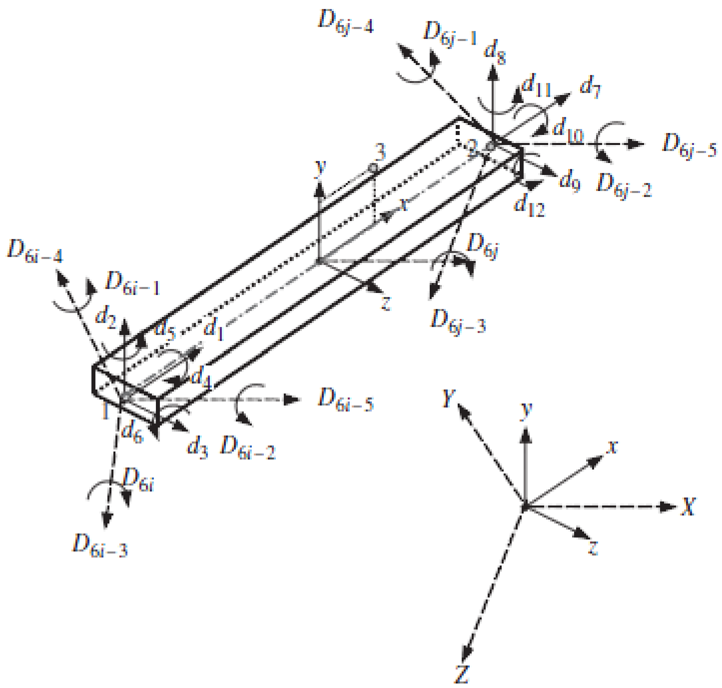
$$
{r_x}^2 = \frac{I_x}{A}
$$

in which Ix is the second moment of area (or moment of inertia) of the cross-section of the beam with respect to the x axis.

Equations in Global Coordinate System

Having known the element matrices in the local coordinate system, the next thing to do is to transform the element matrices into the global coordinate system to account for the differences in orientation of all the local coordinate systems that are attached on individual frame members.

Assume that the local nodes 1 and 2 of the element correspond to global nodes i and j , respectively. The displacement at a local node should have three translational components in the x, y and z directions, and three rotational components with respect to the x, y and z axes. They are numbered sequentially by d1–d12 corresponding to the physical deformations as defined by Eq. (6.16). The displacement at a global node should also have three translational components in the X, Y and Z directions, and three rotational components with respect to the X, Y and Z axes. They are numbered sequentially by $D6_{i-5}$,$D6_{i-4}$, . . . , and D6i for the i_{th} node, as shown in Figure below. The same sign convention applies to node j. The coordinate transformation gives the relationship between the displacement vector de based on the local coordinate system and the displacement vector De for the same element but based on the global coordinate system:



d_e = T D_e, where

$$\mathbf{D}_e = \begin{Bmatrix} D_{6i-5} \\ D_{6i-4} \\ D_{6i-3} \\ D_{6i-2} \\ D_{6i-1} \\ D_{6i} \\ D_{6j-5} \\ D_{6j-4} \\ D_{6j-3} \\ D_{6j-2} \\ D_{6j-1} \\ D_{6j} \end{Bmatrix}$$

and T is the transformation matrix for the truss element given by

$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}_3 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{T}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T}_3 \end{bmatrix}$$

in which

$$\mathbf{T}_3 = \begin{bmatrix} l_x & m_x & n_x \\ l_y & m_y & n_y \\ l_z & m_z & n_z \end{bmatrix}$$

where l_k, m_k and n_k (k = x, y, z) are direction cosines defined by

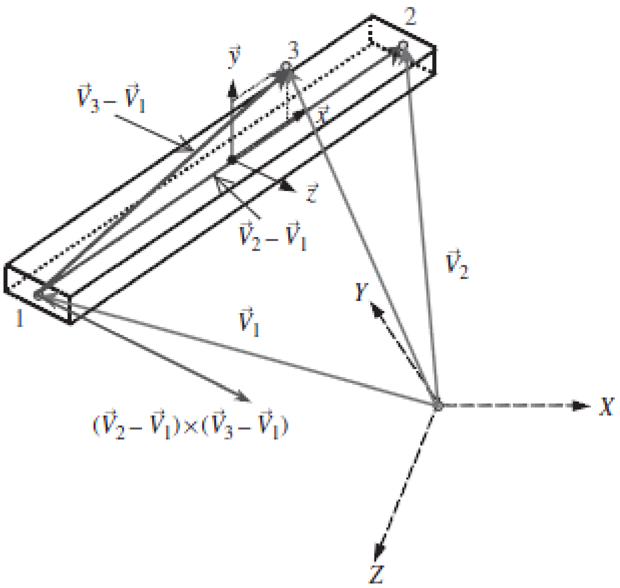
$l_x = \cos(x,X)$, $m_x = \cos(x, Y)$, $n_x = \cos(x,Z)$

$l_y = \cos(y,X)$, $m_y = \cos(y, Y)$, $n_y = \cos(y,Z)$

$l_z = \cos(z,X)$, $m_z = \cos(z, Y)$, $n_z = \cos(z,Z)$

To define these direction cosines, the position and the three-dimensional orientation of the frame element have to be defined first. With nodes 1 and 2, the location of the element is fixed on the local coordinate frame, and the orientation of the element has also been fixed in the x direction. However, the local coordinate frame can still rotate about the axis of the beam. One more additional point in the local coordinate has to be defined. This point can be chosen anywhere in the local x– Y plane, but not on the x-axis. Therefore, node 3 is chosen, as shown in Figure 6.6.The position vectors _ V1, _ V2 and _ V3 can be expressed as

$$\vec{V}_1 = X_1\vec{X} + Y_1\vec{Y} + Z_1\vec{Z}$$
$$\vec{V}_2 = X_2\vec{X} + Y_2\vec{Y} + Z_2\vec{Z}$$
$$\vec{V}_3 = X_3\vec{X} + Y_3\vec{Y} + Z_3\vec{Z}$$



where X_k, Y_k and Z_k (k = 1, 2, 3) are the coordinates for node k, and $\vec{X}$, $\vec{Y}$, $\vec{Z}$ are unit vectors along X, Y and Z axes. We now define

$$\left.\begin{aligned} X_{kl} &= X_k - X_l \\ Y_{kl} &= Y_k - Y_l \\ Z_{kl} &= Z_k - Z_l \end{aligned}\right\} \quad k, l = 1, 2, 3$$

Vectors ($\vec{V_2}$-$\vec{V_1}$) and ($\vec{V_3}$-$\vec{V_1}$) can thus be obtained using above equations as follows:

$\vec{V_2}$-$\vec{V_1}$ = $X_{21}\vec{X}$ + $Y_{21}\vec{Y}$ + $Z_{21}\vec{Z}$

$\vec{V_3}$-$\vec{V_1}$ = $X_{31}\vec{X}$ + $Y_{31}\vec{Y}$ + $Z_{31}\vec{Z}$

The length of the frame element can be obtained by

$$l_e = 2a = \left|\vec{V_2} - \vec{V_1}\right| = \sqrt{X_{21}^2 + Y_{21}^2 + Z_{21}^2}$$

The unit vector along x-axis can thus be expressed as

$$\vec{x} = \frac{(\vec{V_2} - \vec{V_1})}{\left|\vec{V_2} - \vec{V_1}\right|} = \frac{X_{21}}{2a}\vec{X} + \frac{Y_{21}}{2a}\vec{Y} + \frac{Z_{21}}{2a}\vec{Z}$$

Therefore, the direction cosines in the x direction are given as

$$l_x = \cos(x, X) = \vec{x}\,\vec{X} = \frac{X_{21}}{2a}$$

$$m_x = \cos(x, Y) = \vec{x}\,\vec{Y} = \frac{Y_{21}}{2a}$$

$$n_x = \cos(x, Z) = \vec{x}\,\vec{Z} = \frac{Z_{21}}{2a}$$

It now can be seen that the direction of z axis can be defined by the cross product of vectors $(\vec{V_2} - \vec{V_1})$ and $(\vec{V_3} - \vec{V_1})$. Hence a unit vector along z axis can be expressed as:

$$\vec{z} = \frac{(\vec{V_2} - \vec{V_1})\text{x}(\vec{V_3} - \vec{V_1})}{\left|(\vec{V_2} - \vec{V_1})\text{x}(\vec{V_3} - \vec{V_1})\right|}$$

Since y axis is perpendicular to both x axis and z axis, the unit vector along y axis can be obtained by cross product

$$\vec{y} = \vec{z}\,\text{x}\,\vec{x}$$

Using the transformation matrix, T, the matrices for space frame elements in the global coordinate system can be obtained as:

$$\mathbf{K_e = T^T\, k_e\, T}$$

$$\mathbf{M_e = T^T\, m_e\, T}$$

$$\mathbf{F_e = T^T\, f_e}$$

Frame element end releases

Including member loading in equation

$$\mathbf{f}_{IJ} = \mathbf{k}_{IJ}\, \mathbf{u}_{IJ},$$

the twelve equilibrium equations in the local IJ reference system can be written as

$$\mathbf{F} = \mathbf{ku} + \mathbf{r}$$

If one end of the member has a hinge, or other type of release that causes the corresponding force to be equal to zero, above equation requires modification. A typical equation is of the following form:

$$f_n = \sum_{j=1}^{12} k_{nj}\, u_j + r_n$$

If we know a specific value of f_n is zero because of a release, the corresponding displacement u_n can be written as:

$$u_n = \sum_{j=1}^{n-1} \frac{k_{nj}}{k_{nm}} u_j + \sum_{j=n+1}^{12} \frac{k_{nj}}{k_{nm}} u_j + r_n$$

Therefore, by substitution of last equation into the other eleven equilibrium equations, the unknown un can be eliminated and the corresponding row and column set to zero. Or:

$$\overline{f_{IJ}} = \overline{k_{IJ}} u_{IJ} + \overline{r_{IJ}}$$

The terms $f_n = r_n = 0$ and the new stiffness and load terms are equal to:

$$\overline{k_{IJ}} = k_{ij} - k_{in} \frac{k_{nj}}{k_{nn}}$$

$$\overline{r_i} = r_i - r_n \frac{k_{ni}}{k_{nn}}$$

This procedure can be repeatedly applied to the element equilibrium equations for all releases. After the other displacements associated with the element have been found from a solution of the global equilibrium equations, the displacements associated with the releases can be calculated from Equation (4.31) in reverse order from the order in which the displacements were eliminated. The repeated application of these simple numerical equations is defined as static condensation or partial Gauss elimination.

Unstable End Releases

In 3D Frame Analysis Library, any combination of end releases may be specified for a Frame element provided that the element remains stable; this assures that all load applied to the element is transferred to the rest of the structure. The following sets of releases are unstable, either alone or in combination, and are not permitted.

- Releasing U1 at both end
- Releasing U2 at both ends
- Releasing U3 at both ends
- Releasing R1 at both ends
- Releasing R2 at both ends and U3 at either end
- Releasing R3 at both ends and U2 at either end

Introduction to Dynamic Analysis

The dynamic force equilibrium Equation can be written in the following form as a set of N_d second order differential equations:

$$M\ddot{u}(t) + C\dot{u}(t) + Ku(t) = F(t) = \sum_{j=1}^{J} f_j g(t)_j$$

All possible types of time-dependent loading, including wind, wave and seismic, can be represented by a sum of "J" space vectors f_j , which are not a function of time, and J time functions $g(t)_j$.

For the dynamic solution of arbitrary structural systems, however, the elimination of the massless displacement is, in general, not numerically efficient because the stiffness matrix loses its sparsity. Therefore, 3D Frame Analysis Library does not use static condensation to retain the sparseness of the stiffness matrix.

The fundamental mathematical method that is used to solve the equilibrity equations is the separation of variables. This approach assumes the solution can be expressed in the following form:

$\mathbf{u}(t) = \Phi\, \mathbf{Y}(t)$

Where Φ is an "N_d by N" matrix containing N spatial vectors that are not a function of time, and Y(t) is a vector containing N functions of time.

Before solution, we require that the space functions satisfy the following mass and stiffness orthogonality conditions:

$\Phi^T \mathbf{M}\, \Phi = I$

$\Phi^T \mathbf{K}\, \Phi = \Omega^2$

where I is a diagonal unit matrix and Ω^2 is a diagonal matrix in which the diagonal terms are ω_n^2 .The term ω_n has the units of radians per second and may or may not be a free vibration frequencies. It should be noted that the fundamentals of mathematics place no restrictions on those vectors, other than the orthogonality properties. Each space function vector, ϕ_n, is always normalized so that the Generalized Mass is equal to one, or $\phi_n^T M \phi_n = 1.0$.

The above equations yield to:

$$I\ddot{Y}(t) + d\dot{Y}(t) + \Omega^2 Y(t) = \sum_{j=1}^{J} p_j g(t)_j$$

where $p_j = \Phi^T f_j$ are defined as the modal participation factors for load function j. The term p_{nj} is associated with the n_{th} mode. Note that there is one set of "N" modal participation factors for each spatial load condition fj . For all real structures, the "N by N" matrix d is not diagonal; however, to uncouple the modal equations, it is necessary to assume classical damping where there is no coupling between modes. Therefore, the diagonal terms of the modal damping are defined by:

$d_{nn} = 2\ \zeta_n\ \omega_n$

where ζ_n is defined as the ratio of the damping in mode n to the critical damping of the model. A typical uncoupled modal equation for linear structural systems is of the following form:

$$\ddot{y}(t)_n + 2\zeta_n\omega_n\dot{y}(t)_n + {\omega_n}^2 y(t)_n = \sum_{j=1}^{J} p_{nj}g(t)_j$$

For three-dimensional seismic motion, this equation can be written as:

$$\ddot{y}(t)_n + 2\zeta_n\omega_n\dot{y}(t)_n + {\omega_n}^2 y(t)_n = p_{nx}\ddot{u}(t)_{gx} + p_{ny}\ddot{u}(t)_{gy} + p_{nz}\ddot{u}(t)_{gz}$$

where the three-directional modal participation factors, or in this case earthquake excitation factors, are defined by $p_{nj} = -\Phi_n^T M_j$ in which j is equal to x, y or z and n is the mode number.

Response Spectrum Analysis

The maximum modal displacement for a structural model can now be calculated for a typical mode n with period T_n and corresponding spectrum response value $S(\omega_n)$. The maximum modal response associated with period T_n is given by:

$$Y(T_n)_{MAX} = S(\omega_n) / \omega_n^2$$

The maximum modal displacement response of the structural model is calculated from:

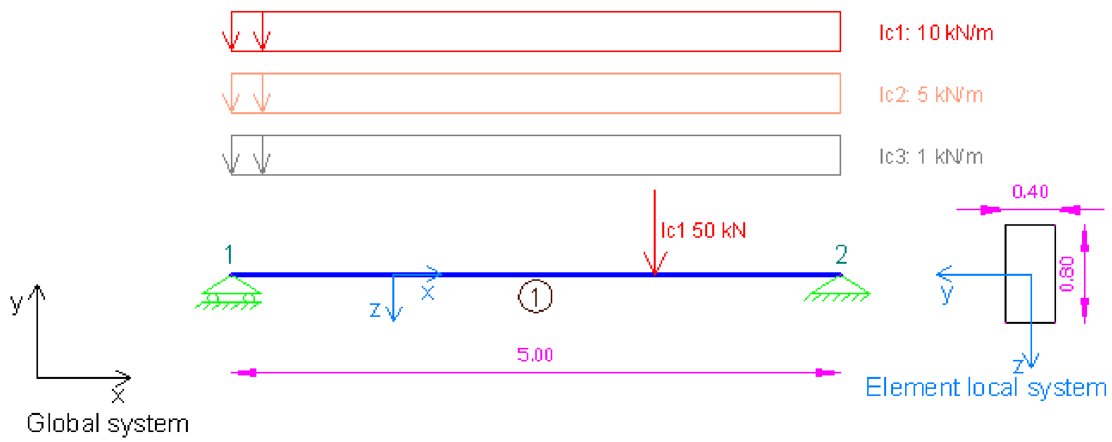
$$u_n = y(T_n)_{MAX}\, \phi_n$$

The corresponding internal modal forces, f_{kn}, are calculated from standard matrix structural analysis using the same equations as required in static analysis.

Example problems for 3D Frame Analysis Library

At this section some characteristic primer problems will be presented in order to comprehensively demonstrate the basic features of 3D Frame Analysis Library. The reader is advised to refer to the corresponding Visual Studio project to see in action how each example is implemented and analyzed with 3D Frame Analysis Library.

Example 1: Load case and combination definitions



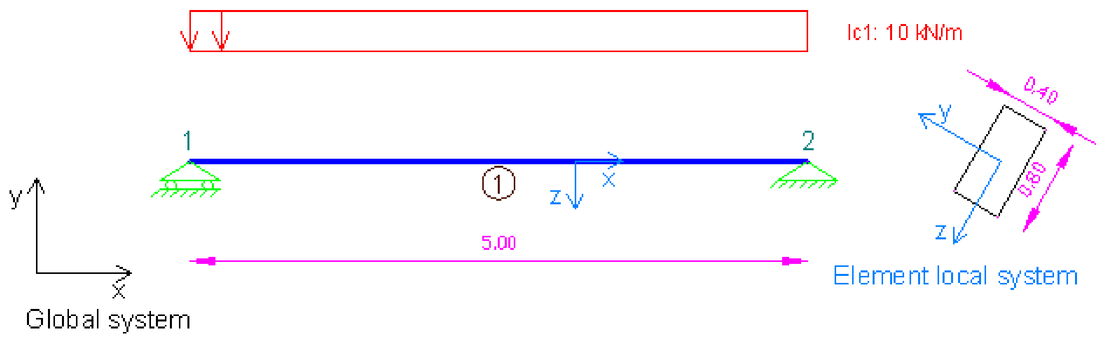

```
//New model definition
Model Model = new Model();
Model.LicenseInfo = LicenseInfo;

//-------MATERIAL DEFINITION-------

//Create a new material for concrete
Material matConcrete = new Material();
matConcrete.Name = "Concrete";//Material name
matConcrete.Density = 2.5;//density in mass units/m3, for example tn/m3
matConcrete.G = 11538461;//shear modulus
matConcrete.E = 30000000;//elasticity modulus

//-------SECTIONS DEFINITION-------

//Create a new beam section of dimensions 40cmx80xm
FrameElementSection secBeam40_80 = new FrameElementSection();
secBeam40_80.Name = "Beam40/80";//section name
secBeam40_80.A = 0.4 * 0.8;//section area
secBeam40_80.Iy = 0.4 * 0.8 * 0.8 * 0.8 / 12;//inertia moment about local y axis
secBeam40_80.Iz = 0.8 * 0.4 * 0.4 * 0.4 / 12;//inertia moment about local z axis
secBeam40_80.It = 0.0117248;//torsional constant
secBeam40_80.b = 0.40;//section width
secBeam40_80.h = 0.80;//section height

//-------MODEL GEOMETRY AND LOADS DEFINITION-------

//Create node n1
Frame3D.SuperNode n1 = new Frame3D.SuperNode(1, 0, 0, 0);
```

```
            n1.dof1constraint = true;//delete
            n1.dof2constraint = true;//translational constraint in
direction y at local system of node
            n1.dof3constraint = true;//translational constraint in
direction z at local system of node
            n1.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n1.dof5constraint = true;//rotational constraint in
direction y at local system of node
            Model.InputNodes.Add(n1);

            //Create node n2
            Frame3D.SuperNode n2 = new Frame3D.SuperNode(2, 5, 0, 0);
            n2.dof1constraint = true;//translational constraint in
direction x at local system of node
            n2.dof2constraint = true;//translational constraint in
direction y at local system of node
            n2.dof3constraint = true;//translational constraint in
direction z at local system of node
            n2.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n2.dof5constraint = true;//rotational constraint in
direction y at local system of node
            Model.InputNodes.Add(n2);

            //Create frame element 1
            //Note that the 4th argument specifies the auxiliary
point that lies in the xy plane that is formed by the x and y axes in
the local element system
            FrameSuperElement el1 = new FrameSuperElement(1, n1, n2,
new Geometry.XYZ(0, 0, 1), matConcrete, secBeam40_80, new
MemberReleases(), new MemberReleases(), false, false);

            LinearLoadCaseForSuperFrameElement lc1 = new
LinearLoadCaseForSuperFrameElement("lc1", LoadCaseType.DEAD);
            lc1.UniformLoad.UniformLoadsY.Add(new
FrameSuperUniformLoad(0, 1, -10, -10,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Global));
            lc1.PointLoad.PointLoadsY.Add(new SuperPointLoad(3.5, -
50, LoadDefinitionFromStartingNode.Absolutely,
LoadCordinateSystem.Global));
            el1.LinearLoadCasesList.Add(lc1);

            LinearLoadCaseForSuperFrameElement lc2 = new
LinearLoadCaseForSuperFrameElement("lc2", LoadCaseType.LIVE);
            lc2.UniformLoad.UniformLoadsY.Add(new
FrameSuperUniformLoad(0, 1, -5, -5,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Global));
            el1.LinearLoadCasesList.Add(lc2);

            LinearLoadCaseForSuperFrameElement lc3 = new
LinearLoadCaseForSuperFrameElement("lc3", LoadCaseType.LIVE);
            lc3.UniformLoad.UniformLoadsY.Add(new
FrameSuperUniformLoad(0, 1, -1, -1,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Global));
            el1.LinearLoadCasesList.Add(lc3);

            Model.InputFiniteElements.Add(el1);
```

```
            //-------SOLUTION PHASE-------
            Model.Solve();

            //-------OBTAIN RESULTS-------
            double[] Min, Max;//The combination results will be saved
in these arrays
            //Note that the definition of two arrays for minimum and
maximum combination results is required
            //For combination type "ADD", Min and Max values are
always equal

            //Reactions (All are defined in the node local system)
            //Rections for load case lc1
            n1.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n1_Rty_lc1 = Max[1];
            n2.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n2_Rty_lc1 = Max[1];

            //Rections for load case lc2
            n1.GetReactionsForLoadCase("lc2", out Min, out Max, 0);
            double n1_Rty_lc2 = Max[1];
            n2.GetReactionsForLoadCase("lc2", out Min, out Max, 0);
            double n2_Rty_lc2 = Max[1];

            //Rections for load case lc13
            n1.GetReactionsForLoadCase("lc3", out Min, out Max, 0);
            double n1_Rty_lc3 = Max[1];
            n2.GetReactionsForLoadCase("lc3", out Min, out Max, 0);
            double n2_Rty_lc3 = Max[1];

            //Node Displacements (All are defined in the node local
system)
            //Note that constained degrees of freedom have zero
displacements
            n1.GetNodalDisplacementsForLoadCase("lc1", out Min, out
Max, 0);
            double[] n1_Disp = Max;
            n2.GetNodalDisplacementsForLoadCase("lc1", out Min, out
Max, 0);
            double[] n2_Disp = Max;

            //Element internal forces and displacements
            el1.GetInternalForcesForLoadCase(0, "lc1", out Min, out
Max, 0); //Internal forces at the start of the member
            double[] forces_along_member_left = Max;
            el1.GetInternalForcesForLoadCase(2.5, "lc1", out Min, out
Max, 0);//Internal forces at the middle of the member
            double[] forces_along_member_middle = Max;
            el1.GetInternalForcesForLoadCase(5, "lc1", out Min, out
Max, 0);//Internal forces at the end of the member
            double[] forces_along_member_right = Max;

            el1.GetDisplacementsForLoadCase(0, "lc1", out Min, out
Max, 0); //Internal displacements at the start of the member
            double[] disps_along_member_left = Max;
            el1.GetDisplacementsForLoadCase(2.5, "lc1", out Min, out
Max, 0);//Internal displacements at the middle of the member
            double[] disps_along_member_middle = Max;
            el1.GetDisplacementsForLoadCase(5, "lc1", out Min, out
Max, 0);//Internal displacements at the end of the member
```

```
            double[] disps_along_member_right = Max;

            //Creation of a load combination
            //Note that load combinations can also be defined after
analysis has been completed
            //A load combination for 2.00 lc1 - 0.5 lc2 is created,
as follows:
            LoadCombination LCombo = new
LoadCombination("combination", ComboType.ADD);
            LCombo.Items.Add(new LoadCaseWithFactor("lc1", 2));
            LCombo.Items.Add(new LoadCaseWithFactor("lc2", -0.5));

            //All result data can be now obtained for the combination
in the same way as for the load cases
            //for example, get first node reactions:
            n1.GetReactionsForLoadCombination(LCombo, out Min, out
Max);
```

Example 2: Element local coordinate system (skew member)

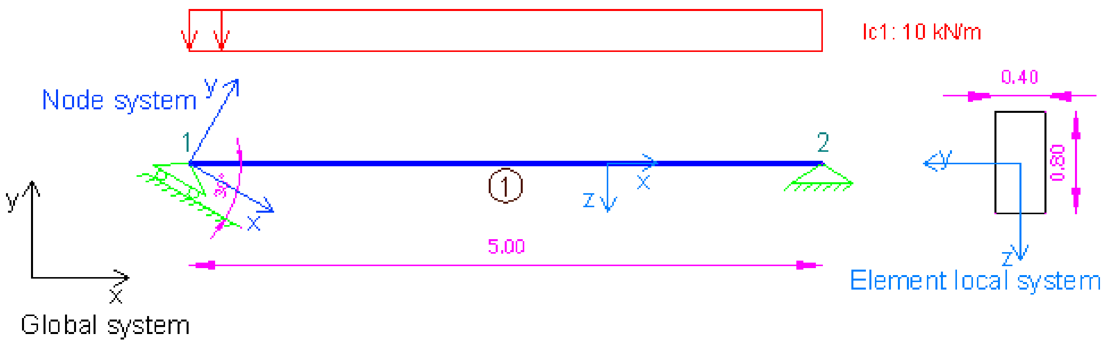



```
            //New model definition
            Model Model = new Model();
            Model.LicenseInfo = LicenseInfo;

            //-------MATERIAL DEFINITION-------

            //Create a new material for concrete
            Material matConcrete = new Material();
            matConcrete.Name = "Concrete";//Material name
            matConcrete.Density = 2.5;//density in mass units/m3, for
example tn/m3
            matConcrete.G = 11538461;//shear modulus
            matConcrete.E = 30000000;//elasticity modulus

            //-------SECTIONS DEFINITION-------

            //Create a new beam section of dimensions 40cmx80xm
            FrameElementSection secBeam40_80 = new
FrameElementSection();
            secBeam40_80.Name = "Beam40/80";//section name
            secBeam40_80.A = 0.4 * 0.8;//section area
            secBeam40_80.Iy = 0.4 * 0.8 * 0.8 * 0.8 / 12;//inertia
moment about local y axis
            secBeam40_80.Iz = 0.8 * 0.4 * 0.4 * 0.4 / 12;//inertia
moment about local z axis
            secBeam40_80.It = 0.0117248;//torsional constant
            secBeam40_80.b = 0.40;//section height
            secBeam40_80.h = 0.80;//section height

            //-------MODEL GEOMETRY AND LOADS DEFINITION-------

            //Create node n1
            Frame3D.SuperNode n1 = new Frame3D.SuperNode(1, 0, 0, 0);
            n1.dof1constraint = true;//translational constraint in
direction y at local system of node
            n1.dof2constraint = true;//translational constraint in
direction y at local system of node
            n1.dof3constraint = true;//translational constraint in
direction z at local system of node
            n1.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n1.dof5constraint = true;//rotational constraint in
direction y at local system of node
```

```
            n1.dof6constraint = true;//rotational constraint in
direction y at local system of node
            Model.InputNodes.Add(n1);

            //Create node n2
            Frame3D.SuperNode n2 = new Frame3D.SuperNode(2, 5, 0, 0);
            n2.dof1constraint = true;//translational constraint in
direction x at local system of node
            n2.dof2constraint = true;//translational constraint in
direction y at local system of node
            n2.dof3constraint = true;//translational constraint in
direction z at local system of node
            n2.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n2.dof5constraint = true;//rotational constraint in
direction y at local system of node
            n2.dof6constraint = true;//rotational constraint in
direction y at local system of node
            Model.InputNodes.Add(n2);

            //Create frame element 1
            //Note the definition of the auxiliary point:
Geometry.XYZ(0, Math.Tan(30/180*Math.PI), 1)
            //It shows that the frame will be inserted properly
(rotated about its longitudinal axis)
            FrameSuperElement el1 = new FrameSuperElement(1, n1, n2,
new Geometry.XYZ(0, 0, 1), matConcrete, secBeam40_80, new
MemberReleases(), new MemberReleases(), false, false);

            LinearLoadCaseForSuperFrameElement lc1 = new
LinearLoadCaseForSuperFrameElement("lc1", LoadCaseType.DEAD);
            lc1.UniformLoad.UniformLoadsY.Add(new
FrameSuperUniformLoad(0, 1, -10, -10,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Global));
            el1.LinearLoadCasesList.Add(lc1);

            Model.InputFiniteElements.Add(el1);


            //-------SOLUTION PHASE-------
            Model.Solve();


            //-------OBTAIN RESULTS-------
            double[] Min, Max;

            //Reactions
            //Rections for load case lc1
            n1.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double[] n1_R_lc1 = Max;
            n2.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double[] n2_R_lc1 = Max;

            //Note that element forces are now different, shear force
acts on both y and z directions in element local system
            el1.GetInternalForcesForLoadCase(0, "lc1", out Min, out
Max, 0);//Internal forces at the start of the member
            double[] forces_along_member_left = Max;
            el1.GetInternalForcesForLoadCase(2.5, "lc1", out Min, out
Max, 0);//Internal forces at the middle of the member
```

```
            double[] forces_along_member_middle = Max;
            el1.GetInternalForcesForLoadCase(5, "lc1", out Min, out
Max, 0);//Internal forces at the end of the member
            double[] forces_along_member_right = Max;

            el1.GetDisplacementsForLoadCase(0, "lc1", out Min, out
Max, 0);//Internal displacements at the start of the member
            double[] disps_along_member_left = Max;
            el1.GetDisplacementsForLoadCase(2.5, "lc1", out Min, out
Max, 0);//Internal displacements at the middle of the member
            double[] disps_along_member_middle = Max;
            el1.GetDisplacementsForLoadCase(5, "lc1", out Min, out
Max, 0);//Internal displacements at the end of the member
            double[] disps_along_member_right = Max;        }
```

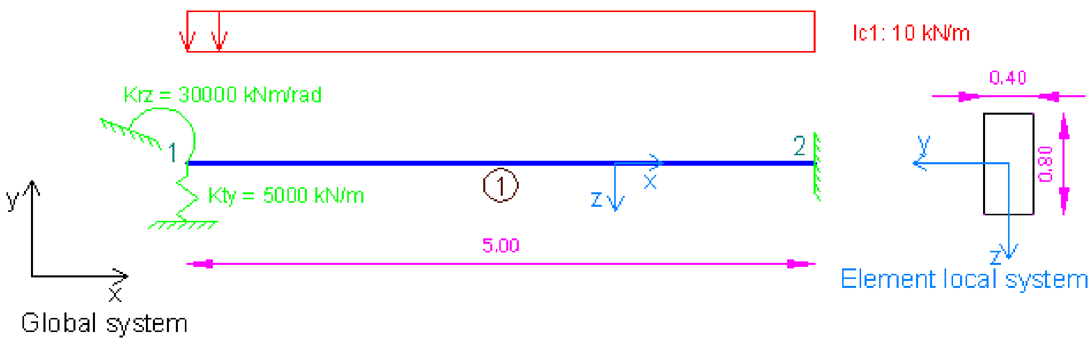
Example 3: Node local coordinate system (skew support)




```
//New model definition
Model Model = new Model();
Model.LicenseInfo = LicenseInfo;

//-------MATERIAL DEFINITION-------

//Create a new material for concrete
Material matConcrete = new Material();
matConcrete.Name = "Concrete";//Material name
matConcrete.Density = 2.5;//density in mass units/m3, for
example tn/m3
matConcrete.G = 11538461;//shear modulus
matConcrete.E = 30000000;//elasticity modulus

//-------SECTIONS DEFINITION-------

//Create a new beam section of dimensions 40cmx80xm
FrameElementSection secBeam40_80 = new
FrameElementSection();
secBeam40_80.Name = "Beam40/80";//section name
secBeam40_80.A = 0.4 * 0.8;//section area
secBeam40_80.Iy = 0.4 * 0.8 * 0.8 * 0.8 / 12;//inertia
moment about local y axis
secBeam40_80.Iz = 0.8 * 0.4 * 0.4 * 0.4 / 12;//inertia
moment about local z axis
secBeam40_80.It = 0.0117248;//torsional constant
secBeam40_80.b = 0.40;//section height
secBeam40_80.h = 0.80;//section height

//-------MODEL GEOMETRY AND LOADS DEFINITION-------

//Create node n1, the local coordinate system of the node
is assigned, which means that it is different from the default global
system.
//In order to define the new system, a new
LocalCoordinateSystem is passed in the corresponding constructor of
SuperNode object
//The first two point of this constructor define the
local x axis of the node system and the third one defines the
coordinates of an auxiliary
//point that lies in local XY plane
Frame3D.SuperNode n1 = new Frame3D.SuperNode(1, 0, 0, 0,
new LocalCoordinateSystem(new Geometry.XYZ(0, 0, 0), new
```

```
Geometry.XYZ(1, Math.Tan(-Math.PI / 6), 0), new Geometry.XYZ(1,
Math.Tan(60.0 / 180 * Math.PI), 0)));
            n1.dof2constraint = true;//translational constraint in
direction y at local system (which was defined previously) of node
            n1.dof3constraint = true;//translational constraint in
direction z at local system of node
            n1.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n1.dof5constraint = true;//rotational constraint in
direction y at local system of node
            Model.InputNodes.Add(n1);

            //Create node n2
            Frame3D.SuperNode n2 = new Frame3D.SuperNode(2, 5, 0, 0);
            n2.dof1constraint = true;//translational constraint in
direction x at local system of node
            n2.dof2constraint = true;//translational constraint in
direction y at local system of node
            n2.dof3constraint = true;//translational constraint in
direction z at local system of node
            n2.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n2.dof5constraint = true;//rotational constraint in
direction y at local system of node
            Model.InputNodes.Add(n2);

            //Create frame element 1
            FrameSuperElement el1 = new FrameSuperElement(1, n1, n2,
new Geometry.XYZ(0, 0, 1), matConcrete, secBeam40_80, new
MemberReleases(), new MemberReleases(), false, false);

            LinearLoadCaseForSuperFrameElement lc1 = new
LinearLoadCaseForSuperFrameElement("lc1", LoadCaseType.DEAD);
            lc1.UniformLoad.UniformLoadsY.Add(new
FrameSuperUniformLoad(0, 1, -10, -10,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Global));
            el1.LinearLoadCasesList.Add(lc1);

            Model.InputFiniteElements.Add(el1);

            //-------SOLUTION PHASE-------
            Model.Solve();

            //-------OBTAIN RESULTS-------
            double[] Min, Max;

            //Support reactions (Note that they are defined in the
node local system)
            n1.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n1_Rty_lc1 = Max[1];
            n2.GetReactionsForLoadCase("lc1", out Min, out Max,
0);//Axial force is acting on the element because of the skew support
at node 1
            double n2_Rtx_lc1 = Max[0];
            n2.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n2_Rty_lc1 = Max[1];
```

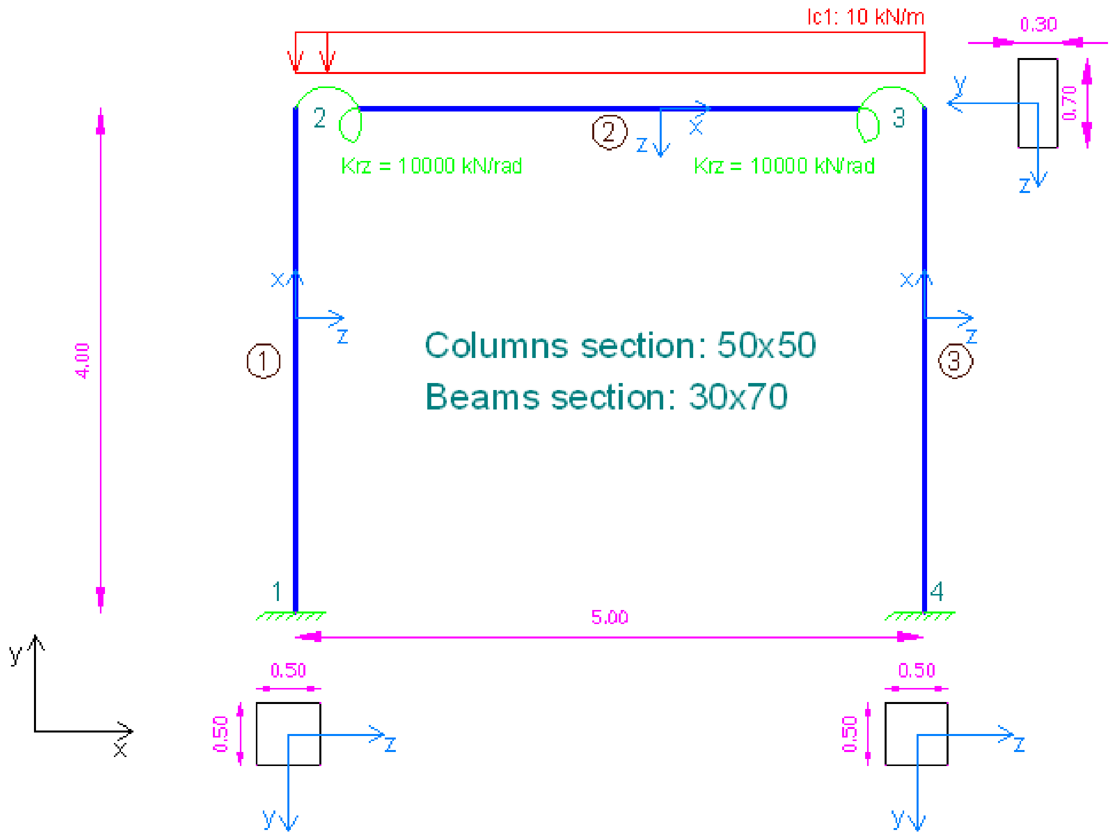
Example 4: Spring supports




```
		//New model definition
		Model Model = new Model();
		Model.LicenseInfo = LicenseInfo;

		//-------MATERIAL DEFINITION-------

		//Create a new material for concrete
		Material matConcrete = new Material();
		matConcrete.Name = "Concrete";//Material name
		matConcrete.Density = 2.5;//density in mass units/m3, for
example tn/m3
		matConcrete.G = 11538461;//shear modulus
		matConcrete.E = 30000000;//elasticity modulus

		//-------SECTIONS DEFINITION-------

		//Create a new beam section of dimensions 40cmx80xm
		FrameElementSection secBeam40_80 = new
FrameElementSection();
		secBeam40_80.Name = "Beam40/80";//section name
		secBeam40_80.A = 0.4 * 0.8;//section area
		secBeam40_80.Iy = 0.4 * 0.8 * 0.8 * 0.8 / 12;//inertia
moment about local y axis
		secBeam40_80.Iz = 0.8 * 0.4 * 0.4 * 0.4 / 12;//inertia
moment about local z axis
		secBeam40_80.It = 0.0117248;//torsional constant
		secBeam40_80.b = 0.40;//section height
		secBeam40_80.h = 0.80;//section height

		//-------MODEL GEOMETRY AND LOADS DEFINITION-------

		//Create node n1
		Frame3D.SuperNode n1 = new Frame3D.SuperNode(1, 0, 0, 0);
		n1.dof3constraint = true;//translational constraint in
direction z at local system of node
		n1.dof4constraint = true;//rotational constraint in
direction x at local system of node
		n1.dof5constraint = true;//rotational constraint in
direction y at local system of node
		n1.Kdof2 = 5000;//Translational spring constant of
partial support at y direction of local node system (units:
force/length, for example kN/m)
```

```
            n1.Kdof6 = 30000;//Rotational spring constant of partial
support about z direction of local node system (units:
moment/rotation, for example kNm/rad)
            Model.InputNodes.Add(n1);

            //Create node n2
            Frame3D.SuperNode n2 = new Frame3D.SuperNode(2, 5, 0, 0);
            n2.dof1constraint = true;//translational constraint in
direction x at local system of node
            n2.dof2constraint = true;//translational constraint in
direction y at local system of node
            n2.dof3constraint = true;//translational constraint in
direction z at local system of node
            n2.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n2.dof5constraint = true;//rotational constraint in
direction y at local system of node
            n2.dof6constraint = true;//rotational constraint in
direction z at local system of node
            Model.InputNodes.Add(n2);

            //Create frame element 1
            FrameSuperElement el1 = new FrameSuperElement(1, n1, n2,
new Geometry.XYZ(0, 0, 1), matConcrete, secBeam40_80, new
MemberReleases(), new MemberReleases(), false, false);

            LinearLoadCaseForSuperFrameElement lc1 = new
LinearLoadCaseForSuperFrameElement("lc1", LoadCaseType.DEAD);
            lc1.UniformLoad.UniformLoadsY.Add(new
FrameSuperUniformLoad(0, 1, -10, -10,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Global));
            el1.LinearLoadCasesList.Add(lc1);

            Model.InputFiniteElements.Add(el1);

            //-------SOLUTION PHASE-------
            Model.Solve();

            //-------OBTAIN RESULTS-------
            double[] Min, Max;

            //Spring reactions can be obtained from the corresponding
Method, as follows
            //Spring reactions, as node reactions, as reported in the
node local system
            n1.GetSpringReactionsForLoadCase("lc1", out Min, out Max,
0);
            double n1_Rty_lc1 = Max[1];
            n1.GetSpringReactionsForLoadCase("lc1", out Min, out Max,
0);
            double n1_Rrz_lc1 = Max[5];
            n2.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n2_Rty_lc1 = Max[1];
            n2.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n2_Rzz_lc1 = Max[5];
```

Example 5: Partial (semi-rigid) member releases

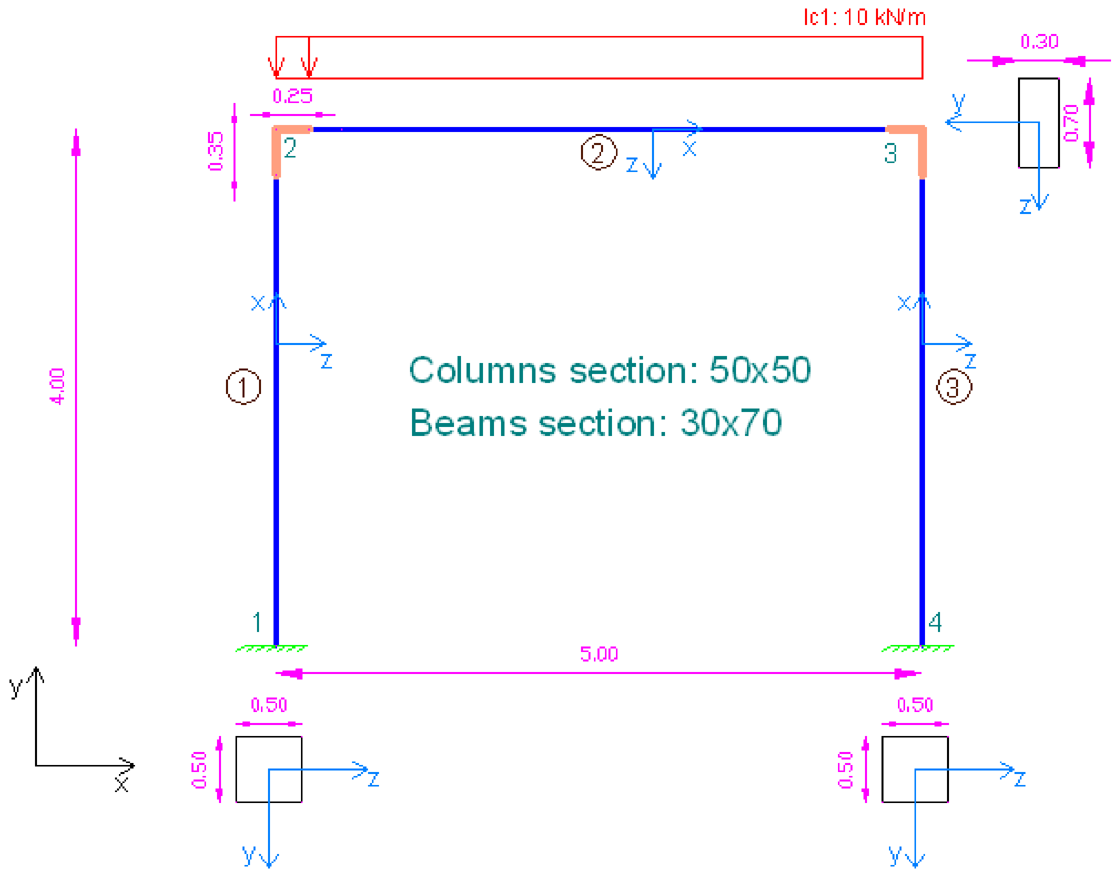



```
//New model definition
Model Model = new Model();
Model.LicenseInfo = LicenseInfo;

//-------MATERIAL DEFINITION-------

//Create a new material for concrete
Material matConcrete = new Material();
matConcrete.Name = "Concrete";//Material name
matConcrete.Density = 2.5;//density in mass units/m3, for
example tn/m3
matConcrete.G = 11538461;//shear modulus
matConcrete.E = 30000000;//elasticity modulus

//-------SECTIONS DEFINITION-------

//Create a new beam section of dimensions 30cmx70xm
FrameElementSection secBeam30_70 = new
FrameElementSection();
secBeam30_70.Name = "Beam30/70";//section name
secBeam30_70.A = 0.3 * 0.7;//section area
secBeam30_70.Iy = 0.3 * 0.7 * 0.7 * 0.7 / 12;//inertia
moment about local y axis
secBeam30_70.Iz = 0.8 * 0.3 * 0.3 * 0.3 / 12;//inertia
moment about local z axis
secBeam30_70.It = 4.347e-3;//torsional constant
secBeam30_70.b = 0.30;//section height
secBeam30_70.h = 0.70;//section height
```

```
            //Create a new beam section of dimensions 50cmx50xm
            FrameElementSection secColumn50_50 = new
FrameElementSection();
            secColumn50_50.Name = "Column50/50";  //section name
            secColumn50_50.A = 0.5 * 0.5;//section area
            secColumn50_50.Iy = 0.5 * 0.5 * 0.5 * 0.5 / 12;//inertia
moment about local y axis
            secColumn50_50.Iz = 0.5 * 0.5 * 0.5 * 0.5 / 12;//inertia
moment about local z axis
            secColumn50_50.It = 8.8125e-3;
            secColumn50_50.b = 0.50;//section height
            secColumn50_50.h = 0.50;//section height

            //-------MODEL GEOMETRY AND LOADS DEFINITION-------

            //Create node n1
            Frame3D.SuperNode n1 = new Frame3D.SuperNode(1, 0, 0, 0);
            n1.dof1constraint = true;//translational constraint in
direction x at local system of node
            n1.dof2constraint = true;//translational constraint in
direction y at local system of node
            n1.dof3constraint = true;//translational constraint in
direction z at local system of node
            n1.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n1.dof5constraint = true;//rotational constraint in
direction y at local system of node
            n1.dof6constraint = true;//rotational constraint in
direction z at local system of node
            Model.InputNodes.Add(n1);

            //Create node n2
            Frame3D.SuperNode n2 = new Frame3D.SuperNode(2, 0, 4, 0);
            Model.InputNodes.Add(n2);

            //Create node n3
            Frame3D.SuperNode n3 = new Frame3D.SuperNode(3, 5, 4, 0);
            Model.InputNodes.Add(n3);

            //Create node n4
            Frame3D.SuperNode n4 = new Frame3D.SuperNode(4, 5, 0, 0);
            n4.dof1constraint = true;//translational constraint in
direction x at local system of node
            n4.dof2constraint = true;//translational constraint in
direction y at local system of node
            n4.dof3constraint = true;//translational constraint in
direction z at local system of node
            n4.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n4.dof5constraint = true;//rotational constraint in
direction y at local system of node
            n4.dof6constraint = true;//rotational constraint in
direction z at local system of node
            Model.InputNodes.Add(n4);

            //Create frame element 1
            FrameSuperElement el1 = new FrameSuperElement(1, n1, n2,
new Geometry.XYZ(0, 0, 1), matConcrete, secColumn50_50, new
MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el1);
```

```
            //Create a MemberRelases object. Release are defined in
element local coordinate system.
            MemberReleases PartialRelease = new MemberReleases();
            PartialRelease.Name = "Partial bending release";//Name of
the object
            PartialRelease.rz = true;//Release the rotational degree
of freedom about z axis (in element local coordinate system)
            PartialRelease.krz = 10000;//Assign a spring stiffness
(units in moment/rotations, for example kNm/rad)
            //Note that the corresponding degree of freedom should be
first released in order to define afterwards a partial stiffness
constant
            //In case of full release we should have given
PartialRelease.krz = 0;

            //Create frame element 2. Note that the proper release
object (Partial Releases is passed in the constructor)
            FrameSuperElement el2 = new FrameSuperElement(2, n2, n3,
new Geometry.XYZ(0, 4, 1), matConcrete, secBeam30_70, PartialRelease,
PartialRelease, false, false);
            LinearLoadCaseForSuperFrameElement lc1 = new
LinearLoadCaseForSuperFrameElement("lc1", LoadCaseType.DEAD);
            lc1.UniformLoad.UniformLoadsY.Add(new
FrameSuperUniformLoad(0, 1, -10, -10,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Global));
            el2.LinearLoadCasesList.Add(lc1);
            Model.InputFiniteElements.Add(el2);

            //Create frame element 3
            FrameSuperElement el3 = new FrameSuperElement(3, n4, n3,
new Geometry.XYZ(5, 0, 1), matConcrete, secColumn50_50, new
MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el3);

            //-------SOLUTION PHASE-------
            Model.Solve();

            //-------OBTAIN RESULTS-------
            double[] Min, Max;

            //Support reactions
            n1.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n1_Rtx_lc1 = Max[0];
            n1.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n1_Rty_lc1 = Max[1];
            n4.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n4_Rtx_lc1 = Max[0];
            n4.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n4_Rty_lc1 = Max[1];

            //Rotations at nodes 2 and 3 (in local node system)
            n2.GetNodalDisplacementsForLoadCase("lc1", out Min, out
Max, 0);//negative rotation
            double n2_Rrz_lc1 = Max[5];//negative rotation
            n3.GetNodalDisplacementsForLoadCase("lc1", out Min, out
Max, 0);//the same rotation, but positive
            double n3_Rrz_lc1 = Max[5];//the same rotation, but
positive
```

Example 6: Rigid offsets

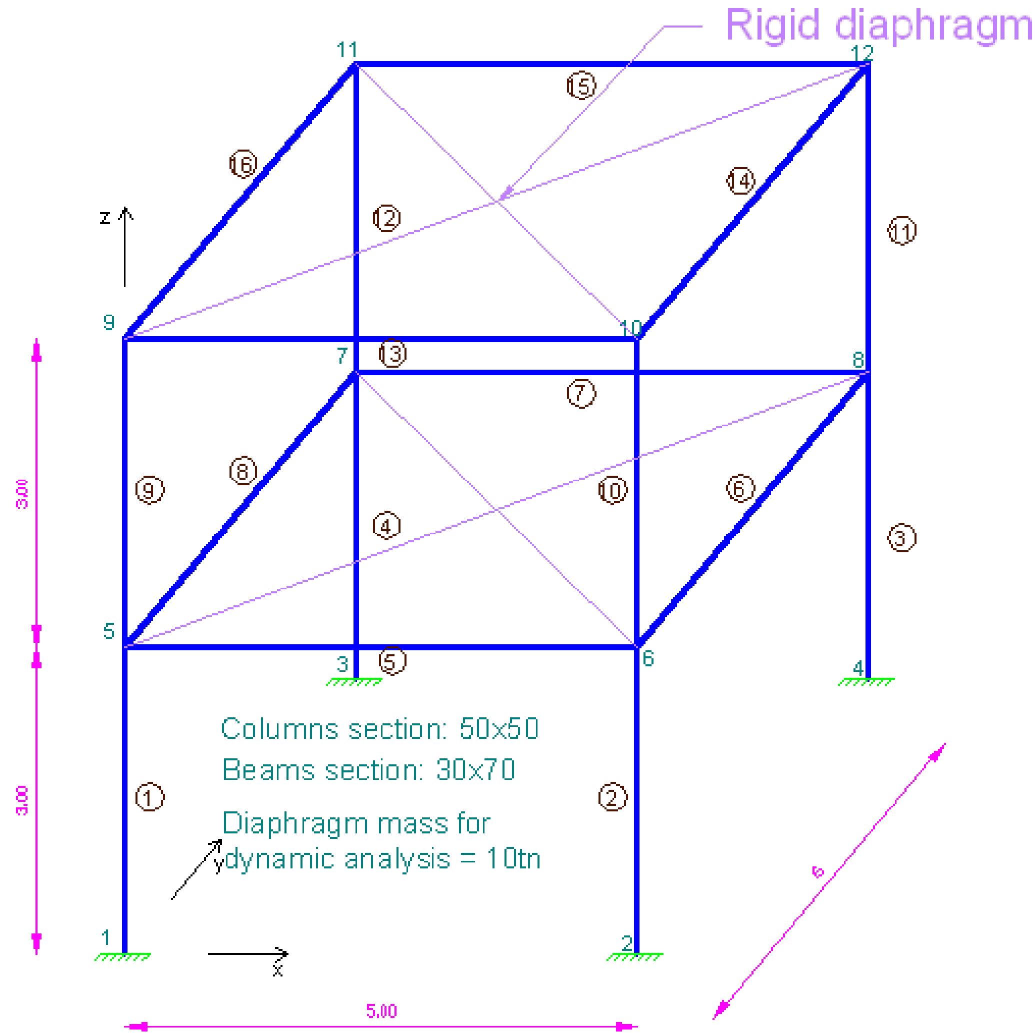


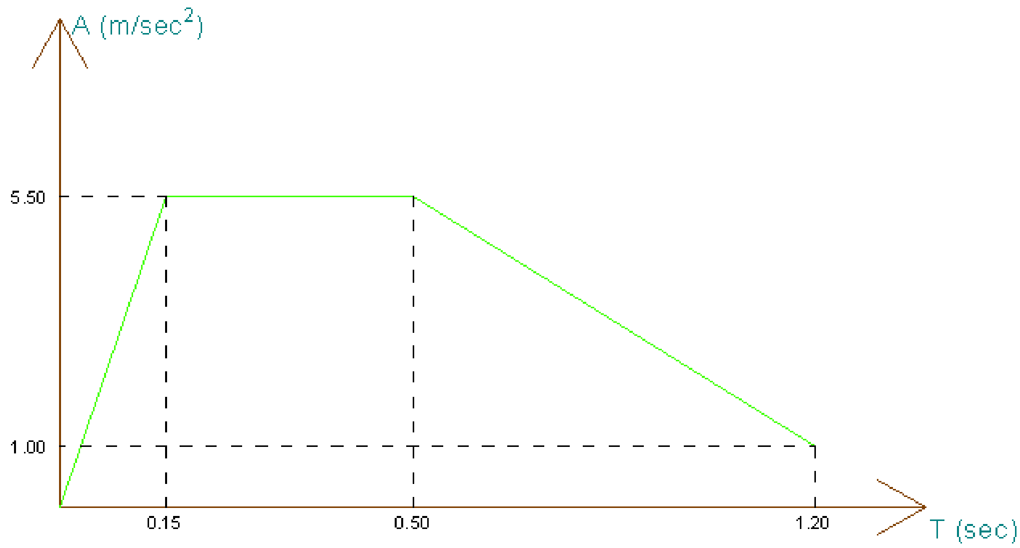

```
            //New model definition
            Model Model = new Model();
            Model.LicenseInfo = LicenseInfo;

            //-------MATERIAL DEFINITION-------

            //Create a new material for concrete
            Material matConcrete = new Material();
            matConcrete.Name = "Concrete";//Material name
            matConcrete.Density = 2.5;//density in mass units/m3, for
example tn/m3
            matConcrete.G = 11538461;//shear modulus
            matConcrete.E = 30000000;//elasticity modulus

            //-------SECTIONS DEFINITION-------

            //Create a new beam section of dimensions 30cmx70xm
            FrameElementSection secBeam30_70 = new
FrameElementSection();
            secBeam30_70.Name = "Beam30/70";//section name
            secBeam30_70.A = 0.3 * 0.7;//section area
            secBeam30_70.Iy = 0.3 * 0.7 * 0.7 * 0.7 / 12;//inertia
moment about local y axis
            secBeam30_70.Iz = 0.7 * 0.3 * 0.3 * 0.3 / 12;//inertia
moment about local z axis
            secBeam30_70.It = 4.347e-3;//torsional constant
            secBeam30_70.b = 0.30;//section height
```

```
            secBeam30_70.h = 0.70;//section height

            //Create a new beam section of dimensions 50cmx50xm
            FrameElementSection secColumn50_50 = new
FrameElementSection();
            secColumn50_50.Name = "Column50/50";  //section name
            secColumn50_50.A = 0.5 * 0.5;//section area
            secColumn50_50.Iy = 0.5 * 0.5 * 0.5 * 0.5 / 12;//inertia
moment about local y axis
            secColumn50_50.Iz = 0.5 * 0.5 * 0.5 * 0.5 / 12;//inertia
moment about local z axis
            secColumn50_50.It = 8.8125e-3;
            secColumn50_50.b = 0.50;//section height
            secColumn50_50.h = 0.50;//section height

            //-------MODEL GEOMETRY AND LOADS DEFINITION-------

            //Create node n1
            Frame3D.SuperNode n1 = new Frame3D.SuperNode(1, 0, 0, 0);
            n1.dof1constraint = true;//translational constraint in
direction x at local system of node
            n1.dof2constraint = true;//translational constraint in
direction y at local system of node
            n1.dof3constraint = true;//translational constraint in
direction z at local system of node
            n1.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n1.dof5constraint = true;//rotational constraint in
direction y at local system of node
            n1.dof6constraint = true;//rotational constraint in
direction z at local system of node
            Model.InputNodes.Add(n1);

            //Create node n2
            Frame3D.SuperNode n2 = new Frame3D.SuperNode(2, 0, 4, 0);
            Model.InputNodes.Add(n2);

            //Create node n3
            Frame3D.SuperNode n3 = new Frame3D.SuperNode(3, 5, 4, 0);
            Model.InputNodes.Add(n3);

            //Create node n4
            Frame3D.SuperNode n4 = new Frame3D.SuperNode(4, 5, 0, 0);
            n4.dof1constraint = true;//translational constraint in
direction x at local system of node
            n4.dof2constraint = true;//translational constraint in
direction y at local system of node
            n4.dof3constraint = true;//translational constraint in
direction z at local system of node
            n4.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n4.dof5constraint = true;//rotational constraint in
direction y at local system of node
            n4.dof6constraint = true;//rotational constraint in
direction z at local system of node
            Model.InputNodes.Add(n4);

            //Create frame element 1
            FrameSuperElement el1 = new FrameSuperElement(1, n1, n2,
new Geometry.XYZ(0, 0, 1), matConcrete, secColumn50_50, new
MemberReleases(), new MemberReleases(), false, false);
```

```
            el1.RigidOffsetEndDx = 0.35;
            Model.InputFiniteElements.Add(el1);

            //Create frame element 2. Note that the proper release
object (Partial Releases is passed in the constructor)
            FrameSuperElement el2 = new FrameSuperElement(2, n2, n3,
new Geometry.XYZ(0, 4, 1), matConcrete, secBeam30_70, new
MemberReleases(), new MemberReleases(), false, false);
            el2.RigidOffsetStartDx = 0.25;
            el2.RigidOffsetEndDx = 0.25;
            LinearLoadCaseForSuperFrameElement lc1 = new
LinearLoadCaseForSuperFrameElement("lc1", LoadCaseType.DEAD);
            lc1.UniformLoad.UniformLoadsY.Add(new
FrameSuperUniformLoad(0, 1, -10, -10,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Global));
            el2.LinearLoadCasesList.Add(lc1);
            Model.InputFiniteElements.Add(el2);

            //Create frame element 3
            FrameSuperElement el3 = new FrameSuperElement(3, n4, n3,
new Geometry.XYZ(5, 0, 1), matConcrete, secColumn50_50, new
MemberReleases(), new MemberReleases(), false, false);
            el3.RigidOffsetEndDx = 0.35;
            Model.InputFiniteElements.Add(el3);

            //-------SOLUTION PHASE-------
            Model.Solve();

            //-------OBTAIN RESULTS-------
            double[] Min, Max;

            //Support reactions
            n1.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n1_Rtx_lc1 = Max[0];
            n1.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n1_Rty_lc1 = Max[1];
            n4.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n4_Rtx_lc1 = Max[0];
            n4.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n4_Rty_lc1 = Max[1];

            //Rotations at nodes 2 and 3 (in local node system)
            n2.GetNodalDisplacementsForLoadCase("lc1", out Min, out
Max, 0); //negative rotation
            double n2_Rrz_lc1 = Max[5];//negative rotation
            n3.GetNodalDisplacementsForLoadCase("lc1", out Min, out
Max, 0); ;//the same rotation, but positive
            double n3_Rrz_lc1 = Max[5];//the same rotation, but
positive
```

Example 7: Simple 3D building with rigid floor diaphragms and Response Spectrum Analysis





Response spectrum definition

```
            //New model definition
            Model Model = new Model();
            Model.LicenseInfo = LicenseInfo;

            //-------MATERIAL DEFINITION-------

            //Create a new material for concrete
            Material matConcrete = new Material();
            matConcrete.Name = "Concrete";//Material name
            matConcrete.Density = 2.5;//density in mass units/m3, for
example tn/m3
            matConcrete.G = 11538461;//shear modulus
            matConcrete.E = 30000000;//elasticity modulus

            //-------SECTIONS DEFINITION-------

            //Create a new beam section of dimensions 30cmx70xm
            FrameElementSection secBeam30_70 = new
FrameElementSection();
            secBeam30_70.Name = "Beam30/70";//section name
            secBeam30_70.A = 0.3 * 0.7;//section area
            secBeam30_70.Iy = 0.3 * 0.7 * 0.7 * 0.7 / 12;//inertia
moment about local y axis
            secBeam30_70.Iz = 0.8 * 0.3 * 0.3 * 0.3 / 12;//inertia
moment about local z axis
            secBeam30_70.It = 4.347e-3;//torsional constant
            secBeam30_70.b = 0.30;//section height
            secBeam30_70.h = 0.70;//section height

            //Create a new beam section of dimensions 50cmx50xm
            FrameElementSection secColumn50_50 = new
FrameElementSection();
            secColumn50_50.Name = "Column50/50";  //section name
            secColumn50_50.A = 0.5 * 0.5;//section area
            secColumn50_50.Iy = 0.5 * 0.5 * 0.5 * 0.5 / 12;//inertia
moment about local y axis
            secColumn50_50.Iz = 0.5 * 0.5 * 0.5 * 0.5 / 12;//inertia
moment about local z axis
            secColumn50_50.It = 8.8125e-3;
            secColumn50_50.b = 0.50;//section height
            secColumn50_50.h = 0.50;//section height

            //-------MODEL GEOMETRY AND LOADS DEFINITION-------

            //Create node n1
            Frame3D.SuperNode n1 = new Frame3D.SuperNode(1, 0, 0, 0);
            n1.dof1constraint = true;//translational constraint in
direction x at local system of node
            n1.dof2constraint = true;//translational constraint in
direction y at local system of node
            n1.dof3constraint = true;//translational constraint in
direction z at local system of node
            n1.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n1.dof5constraint = true;//rotational constraint in
direction y at local system of node
            n1.dof6constraint = true;//rotational constraint in
direction z at local system of node
            Model.InputNodes.Add(n1);

            //Create node n2
```

```
            Frame3D.SuperNode n2 = new Frame3D.SuperNode(2, 5, 0, 0);
            n2.dof1constraint = true;//translational constraint in
direction x at local system of node
            n2.dof2constraint = true;//translational constraint in
direction y at local system of node
            n2.dof3constraint = true;//translational constraint in
direction z at local system of node
            n2.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n2.dof5constraint = true;//rotational constraint in
direction y at local system of node
            n2.dof6constraint = true;//rotational constraint in
direction z at local system of node
            Model.InputNodes.Add(n2);

            //Create node n3
            Frame3D.SuperNode n3 = new Frame3D.SuperNode(3, 0, 6, 0);
            n3.dof1constraint = true;//translational constraint in
direction x at local system of node
            n3.dof2constraint = true;//translational constraint in
direction y at local system of node
            n3.dof3constraint = true;//translational constraint in
direction z at local system of node
            n3.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n3.dof5constraint = true;//rotational constraint in
direction y at local system of node
            n3.dof6constraint = true;//rotational constraint in
direction z at local system of node
            Model.InputNodes.Add(n3);

            //Create node n4
            Frame3D.SuperNode n4 = new Frame3D.SuperNode(4, 5, 6, 0);
            n4.dof1constraint = true;//translational constraint in
direction x at local system of node
            n4.dof2constraint = true;//translational constraint in
direction y at local system of node
            n4.dof3constraint = true;//translational constraint in
direction z at local system of node
            n4.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n4.dof5constraint = true;//rotational constraint in
direction y at local system of node
            n4.dof6constraint = true;//rotational constraint in
direction z at local system of node
            Model.InputNodes.Add(n4);

            //Create node n5
            Frame3D.SuperNode n5 = new Frame3D.SuperNode(5, 0, 0, 3);
            Model.InputNodes.Add(n5);

            //Create node n6
            Frame3D.SuperNode n6 = new Frame3D.SuperNode(6, 5, 0, 3);
            Model.InputNodes.Add(n6);

            //Create node n7
            Frame3D.SuperNode n7 = new Frame3D.SuperNode(7, 0, 6, 3);
            Model.InputNodes.Add(n7);

            //Create node n8
            Frame3D.SuperNode n8 = new Frame3D.SuperNode(8, 5, 6, 3);
```

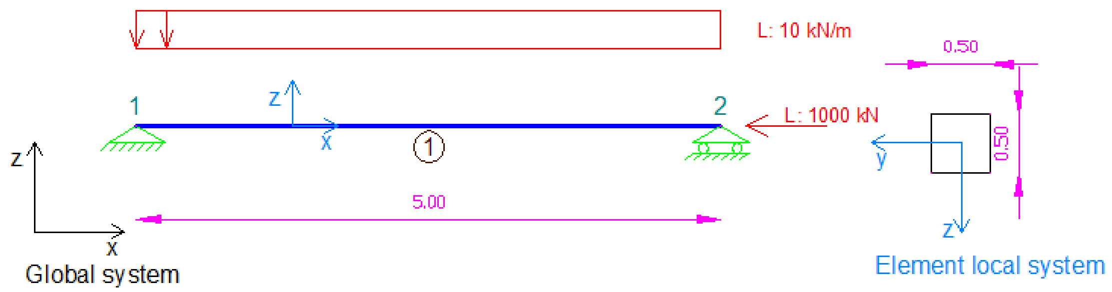
```
            Model.InputNodes.Add(n8);

            //Create node n9
            Frame3D.SuperNode n9 = new Frame3D.SuperNode(9, 0, 0, 6);
            Model.InputNodes.Add(n9);

            //Create node n10
            Frame3D.SuperNode n10 = new Frame3D.SuperNode(10, 5, 0,
6);
            Model.InputNodes.Add(n10);

            //Create node n11
            Frame3D.SuperNode n11 = new Frame3D.SuperNode(11, 0, 6,
6);
            Model.InputNodes.Add(n11);

            //Create node n12
            Frame3D.SuperNode n12 = new Frame3D.SuperNode(12, 5, 6,
6);
            Model.InputNodes.Add(n12);


            //Create frame elements (Note the definition of the
auxiliary point which is different for each frame in order to
correclty place it
            //It is reminded that auxiliary point is only only used
to define the rotation of the frame element about its longitudinal
axis
            //This point should not belong to the longitudinal axis
of the element. In such case, arithmetic errors would occur.


            //Create first story columns

            FrameSuperElement el1 = new FrameSuperElement(1, n1, n5,
new Geometry.XYZ(0, 1, 0), matConcrete, secColumn50_50, new
MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el1);
            FrameSuperElement el2 = new FrameSuperElement(2, n2, n6,
new Geometry.XYZ(5, 1, 0), matConcrete, secColumn50_50, new
MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el2);
            FrameSuperElement el3 = new FrameSuperElement(3, n4, n8,
new Geometry.XYZ(5, 7, 0), matConcrete, secColumn50_50, new
MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el3);
            FrameSuperElement el4 = new FrameSuperElement(4, n3, n7,
new Geometry.XYZ(0, 7, 0), matConcrete, secColumn50_50, new
MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el4);


            //Create first story beams

            FrameSuperElement el5 = new FrameSuperElement(5, n5, n6,
new Geometry.XYZ(0, 1, 3), matConcrete, secColumn50_50, new
MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el5);
            FrameSuperElement el6 = new FrameSuperElement(6, n6, n8,
new Geometry.XYZ(4, 0, 3), matConcrete, secColumn50_50, new
MemberReleases(), new MemberReleases(), false, false);
```

```
            Model.InputFiniteElements.Add(el6);
            FrameSuperElement el7 = new FrameSuperElement(7, n7, n8, new Geometry.XYZ(0, 7, 3), matConcrete, secColumn50_50, new MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el7);
            FrameSuperElement el8 = new FrameSuperElement(8, n5, n7, new Geometry.XYZ(-1, 0, 3), matConcrete, secColumn50_50, new MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el8);

            //Create second story columns

            FrameSuperElement el13 = new FrameSuperElement(13, n9, n10, new Geometry.XYZ(0, 1, 3), matConcrete, secColumn50_50, new MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el13);
            FrameSuperElement el14 = new FrameSuperElement(14, n10, n12, new Geometry.XYZ(4, 0, 3), matConcrete, secColumn50_50, new MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el14);
            FrameSuperElement el15 = new FrameSuperElement(15, n11, n12, new Geometry.XYZ(0, 7, 3), matConcrete, secColumn50_50, new MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el15);
            FrameSuperElement el16 = new FrameSuperElement(16, n9, n11, new Geometry.XYZ(-1, 0, 3), matConcrete, secColumn50_50, new MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el16);

            //Create second story beams

            FrameSuperElement el9 = new FrameSuperElement(9, n5, n9, new Geometry.XYZ(0, 1, 3), matConcrete, secColumn50_50, new MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el9);
            FrameSuperElement el10 = new FrameSuperElement(10, n6, n10, new Geometry.XYZ(5, 1, 3), matConcrete, secColumn50_50, new MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el10);
            FrameSuperElement el11 = new FrameSuperElement(11, n8, n12, new Geometry.XYZ(5, 7, 3), matConcrete, secColumn50_50, new MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el11);
            FrameSuperElement el12 = new FrameSuperElement(12, n7, n11, new Geometry.XYZ(0, 7, 3), matConcrete, secColumn50_50, new MemberReleases(), new MemberReleases(), false, false);
            Model.InputFiniteElements.Add(el12);

            //Create a list of Geometry.XY objects with the boundary points of the floor diaphragms
            //A polygon is then internally defined and all points that liew it it or on its edges will be assumed to be restarined by the diaphragm
            List<Geometry.XY> Pts = new List<Geometry.XY>();
            //Points should be given anti-clockwise
            //Points are given in plan view (x-y)
            //Points (if more than 3) should lie on the same plane
            Pts.Add(new Geometry.XY(0, 0));
            Pts.Add(new Geometry.XY(5, 0));
            Pts.Add(new Geometry.XY(5, 6));
            Pts.Add(new Geometry.XY(0, 6));
```

```
            //Floor diaphragm definition. Note that the 3rd argument
specifies the z-coordinate of diaphragm
            //Floor diaphragm is defined in xy plane only. Global Z
axis is always perpendicular to the plane that the diaphragm points
define
            FloorDiaphragm fd1 = new FloorDiaphragm(1, Pts, 3);
            //Create a load case than specifies the mass source for
the diaphragm
            //This load case only defines the mass for the diaphragm
and is only needed in dynamic analysis.
            //Vertical static loads are not taken into account from
this load case.
            //If a diaphragm is loaded, the corresponding mass load
case should be assigned. Then the diaphragm mass will be considered
for the dynamic analysis
            //A load case for the perimetric beams should then
manually be created, which will distribute the diaphragm loads to the
frames. This is not made automatically by the library
            LinearLoadCaseForFloorDiaphragm mass_fd1 = new
LinearLoadCaseForFloorDiaphragm("mass source", LoadCaseType.DEAD);
            mass_fd1.pz = 5;//units in force/area, for example kN/m2,
positive direction = gravity
            fd1.LinearLoadCasesList.Add(mass_fd1);
            Model.FloorDiaphragms.Add(fd1);
            //Similarily create a floor diaphragm for second story
            FloorDiaphragm fd2 = new FloorDiaphragm(2, Pts, 6);
            LinearLoadCaseForFloorDiaphragm mass_fd2 = new
LinearLoadCaseForFloorDiaphragm("mass source", LoadCaseType.DEAD);
            mass_fd2.pz = 2;//units in force/area, for example kN/m2,
positive direction = gravity
            fd2.LinearLoadCasesList.Add(mass_fd2);
            Model.FloorDiaphragms.Add(fd2);

            //Define a load combination for the mass for the
diaphragms (for example DEAD+0.5LIVE etc)
            LoadCombination MassCombo = new LoadCombination("mass
combo", ComboType.ADD);
            MassCombo.Items.Add(new LoadCaseWithFactor("mass source",
1.0));
            Model.MassSourceCombination = MassCombo;

            //Specify how mass is going to be calculated
            GeneralData.IncludeAdditionalMassesInMassSource = true;
            GeneralData.IncludeLoadsInMassSource = true;
            GeneralData.IncludeSelfWeightInMassSource = true;

            //Create a response spectrum function
            ResponseSpectrumFunction RSFunction = new
ResponseSpectrumFunction("RS function");
            RSFunction.RS_T = new double[] { 0, 0.15, 0.50, 1.20
};//T (time) values of point of the spectrum (in sec)
            RSFunction.RS_A = new double[] { 0, 5.5, 5.5, 1.0 };//A
(spectral acceleration) values of points in spectrum (in length/sec2,
for example m/sec2)

            //Create a response spectrum case and specify the
application direction and the modal combination rule (SRSS or CQC)
            ResponseSpectrumCase RSCase = new
ResponseSpectrumCase("RScase", GroundMotionDirection.UX,
ModeComboType.CQC);
```

```
            RSCase.DiaphragmEccentricityRatio = 0.05;//Specify
diaphragm eccentricity ratio (usually 5%-10%). This value will
produce a torsional about the global Z coordinate at the center of
mass of each diaphragm.
            RSCase.RSFunction = RSFunction;//Assign the previously
defined response spectrum
            Model.ResponseSpectrumCases.Add(RSCase);//Add to model

            Model.NrOfModesToFind = 6;

            //-------SOLUTION PHASE-------
            Model.Solve();

            //-------OBTAIN RESULTS-------
            //Effective mass ratio calculation:
            double Effmx = Model.TotalEffectiveMassUX;//mass excited
in x direction
            double Effmy = Model.TotalEffectiveMassUY;//mass excited
in y direction
            double Massmx = Model.TotalMassUX;//total lateral mass in
x direction
            double Massmy = Model.TotalMassUY;//total lateral mass in
y direction
            double ratio_mass_x = Effmx / Massmx;//>90% of the total
mass is excited by the response spectrum analysis
            double ratio_mass_y = Effmy / Massmy;//>90% of the total
mass is excited by the response spectrum analysis

            //Reactions (Note that all results are now envelopes
beacuse they came from a dynamic analysis)
            double[] Min1, Max1;
            double[] Min2, Max2;
            double[] Min3, Max3;
            double[] Min4, Max4;
            n1.GetReactionsForLoadCase(RSCase.name, out Min1, out
Max1, 0);
            n2.GetReactionsForLoadCase(RSCase.name, out Min2, out
Max2, 0);
            n3.GetReactionsForLoadCase(RSCase.name, out Min3, out
Max3, 0);
            n4.GetReactionsForLoadCase(RSCase.name, out Min4, out
Max4, 0);

            //Modal information
            double[,] Modes = Model.Modes;//each rows represents each
degree of freedom, each column represents the corresponding modal
displacements

            //Periods
            double[] Periods = Model.Periods;//each entry represents
the period of the corresponding node

            //Element 2 (el2) internal forces for response spectrum
case
            double[] Min, Max;
            el2.GetInternalForcesForLoadCase(0, "RScase", out Min,
out Max, 0);
```

Example 8: Beam under uniform and large axial load (P-Δ effect)

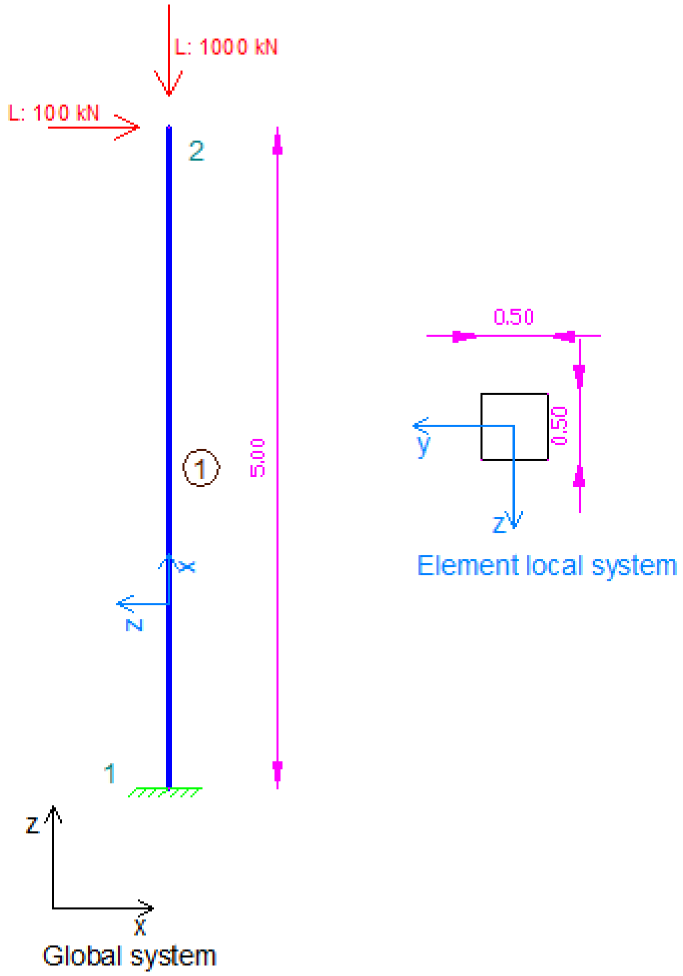



```
            //New model definition
            Model Model = new Model();
            Model.LicenseInfo = LicenseInfo;

            //-------MATERIAL DEFINITION-------

            //Create a new material for concrete
            Material matConcrete = new Material();
            matConcrete.Name = "Concrete";//Material name
            matConcrete.Density = 2.5;//density in mass units/m3, for
example tn/m3
            matConcrete.G = 11538461;//shear modulus
            matConcrete.E = 30000000;//elasticity modulus

            //-------SECTIONS DEFINITION-------

            //Create a new beam section of dimensions 50cmx50xm
            FrameElementSection secBeam50_50 = new
FrameElementSection();
            secBeam50_50.Name = "Beam50/50";//section name
            secBeam50_50.A = 0.5 * 0.5;//section area
            secBeam50_50.Iy = 0.5 * 0.5 * 0.5 * 0.5 / 12;//inertia
moment about local y axis
            secBeam50_50.Iz = 0.5 * 0.5 * 0.5 * 0.5 / 12;//inertia
moment about local z axis
            secBeam50_50.It = 4.347e-3;//torsional constant
            secBeam50_50.b = 0.5;//section height
            secBeam50_50.h = 0.5;//section height

            //-------MODEL GEOMETRY AND LOADS DEFINITION-------

            //First node creation
            Frame3D.SuperNode n1 = new Frame3D.SuperNode(1, 0, 0, 0);
            //Application of supports (fixed conditions out of plane)
            n1.dof1constraint = true;
            n1.dof2constraint = true;
            n1.dof3constraint = true;
            n1.dof4constraint = true;
            n1.dof5constraint = false;
            n1.dof6constraint = true;
            Model.InputNodes.Add(n1);

            //Second node creation
            Frame3D.SuperNode n2 = new Frame3D.SuperNode(2, 5, 0, 0);
            //Application of supports (fixed conditions out of plane)
            n2.dof1constraint = false;
            n2.dof2constraint = true;
```

```
            n2.dof3constraint = true;
            n2.dof4constraint = true;
            n2.dof5constraint = false;
            n2.dof6constraint = true;
            //Load case creation for horizontal load acting at right
node
            LinearLoadCaseForSuperNode L = new
LinearLoadCaseForSuperNode("L", LoadCaseType.OTHER);
            L.Px = -1000;
            n2.LinearLoadCasesList.Add(L);
            Model.InputNodes.Add(n2);

            //Frame element creation
            FrameSuperElement el1 = new FrameSuperElement(1, n1, n2,
new Geometry.XYZ(0, 1, 0), matConcrete, secBeam50_50, new
MemberReleases(), new MemberReleases(), false, false);
            //Load case creation for uniform vertical load on frame
element
            LinearLoadCaseForSuperFrameElement load1 = new
LinearLoadCaseForSuperFrameElement("L", LoadCaseType.OTHER);
            load1.UniformLoad.UniformLoadsZ.Add(new
FrameSuperUniformLoad(0, 1, -10, -10,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Local));
            el1.LinearLoadCasesList.Add(load1);
            Model.InputFiniteElements.Add(el1);

            //Creation of a geometric non linear case that includes
all load cases defined as "L"
            GeometricNonLinearCase NLcase = new
GeometricNonLinearCase("NL");
            //Analysis parameters:
            NLcase.LoadSteps = 50;//50 load steps
            NLcase.IterationsPerLoadStep = 30;//maximum 30 iteration
per load step
            NLcase.ConvergenceTolerance = 1e-12;//convergence
tolerance in terms of force
            //It will include the loads that have been defined as "L"
            NLcase.InputLoadCombo = new LoadCombination("NLcase
loads", ComboType.ADD);
            NLcase.InputLoadCombo.Items.Add(new
LoadCaseWithFactor("L", 1));
            //Definition of stiffness matrix update mode
            NLcase.UpdateStiffnessMethod =
GeometricNonLinearCase.UpdateStiffnessMatrixMethod.AfterEachIteration
InLoadStep;
            NLcase.SaveResultsAtEachLoadStep = true;//Results will be
saved at all intermediate load steps
            Model.GeometricNonLinearCases.Add(NLcase);

            //-------SOLUTION PHASE-------
            el1.Section.StiffnessModifiers.AMod = 0.1;

            Model.Solve();

            //-------OBTAIN RESULTS-------
            double[] Min, Max;
            for (int loadStep = 1; loadStep <= NLcase.LoadSteps;
loadStep++)
            {
```

```
                //Get deflection at the middle of the beam at each
load step
                el1.GetDisplacementsForLoadCase(2.5, "NL", out Min,
out Max, loadStep);
                double Deflection = Min[2];

                //Get bending moment at the middle of the beam at
each load step
                el1.GetInternalForcesForLoadCase(2.5, "NL", out Min,
out Max, loadStep);
                double SpanMoment = Min[4];
```

Example 9: Column under shear and large axial load (P-Δ effect)

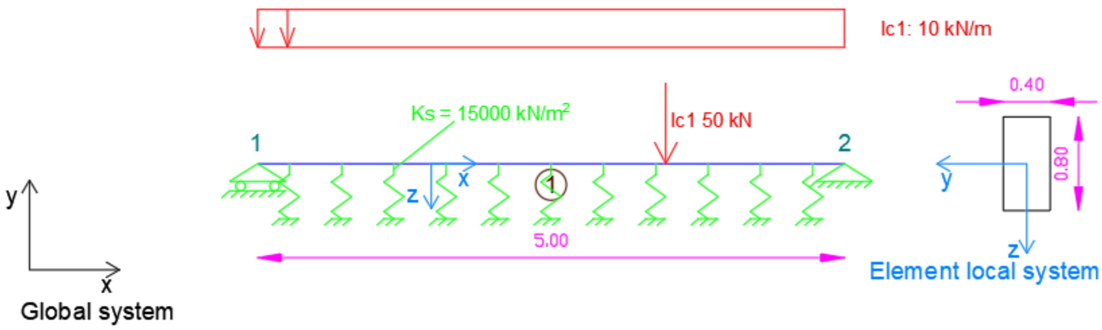



```
            //New model definition
            Model Model = new Model();
            Model.LicenseInfo = LicenseInfo;

            //-------MATERIAL DEFINITION-------

            //Create a new material for concrete
            Material matConcrete = new Material();
            matConcrete.Name = "Concrete";//Material name
            matConcrete.Density = 2.5;//density in mass units/m3, for
example tn/m3
            matConcrete.G = 11538461;//shear modulus
            matConcrete.E = 30000000;//elasticity modulus

            //-------SECTIONS DEFINITION-------

            //Create a new column section of dimensions 50cmx50xm
            FrameElementSection secCol050_50 = new
FrameElementSection();
            secCol050_50.Name = "Column50/50";//section name
            secCol050_50.A = 0.5 * 0.5;//section area
            secCol050_50.Iy = 0.5 * 0.5 * 0.5 * 0.5 / 12;//inertia
moment about local y axis
```

```
            secCol050_50.Iz = 0.5 * 0.5 * 0.5 * 0.5 / 12;//inertia
moment about local z axis
            secCol050_50.It = 4.347e-3;//torsional constant
            secCol050_50.b = 0.5;//section height
            secCol050_50.h = 0.5;//section height

            //-------MODEL GEOMETRY AND LOADS DEFINITION-------

            //First node creation
            Frame3D.SuperNode n1 = new Frame3D.SuperNode(1, 0, 0, 0);
            //Application of supports (fixed conditions out of plane)
            n1.dof1constraint = true;
            n1.dof2constraint = true;
            n1.dof3constraint = true;
            n1.dof4constraint = true;
            n1.dof5constraint = true;
            n1.dof6constraint = true;
            Model.InputNodes.Add(n1);

            //Second node creation
            Frame3D.SuperNode n2 = new Frame3D.SuperNode(2, 0, 0, 5);
            //Application of supports (fixed conditions out of plane)
            n2.dof1constraint = false;
            n2.dof2constraint = true;
            n2.dof3constraint = false;
            n2.dof4constraint = true;
            n2.dof5constraint = false;
            n2.dof6constraint = true;

            //Load case creation for horizontal and vertical load
acting at top node
            LinearLoadCaseForSuperNode L = new
LinearLoadCaseForSuperNode("L", LoadCaseType.OTHER);
            L.Px = 100;
            L.Pz = -1000;
            n2.LinearLoadCasesList.Add(L);
            Model.InputNodes.Add(n2);

            FrameSuperElement el1 = new FrameSuperElement(1, n1, n2,
new Geometry.XYZ(0, 1, 0), matConcrete, secCol050_50, new
MemberReleases(), new MemberReleases(), false, false);

            Model.InputFiniteElements.Add(el1);

            //Creation of a geometric non linear case
            GeometricNonLinearCase NLcase = new
GeometricNonLinearCase("NL");
            //Analysis parameters:
            NLcase.LoadSteps = 50;//50 load steps
            NLcase.IterationsPerLoadStep = 30;//maximum 30 iteration
per load step
            NLcase.ConvergenceTolerance = 1e-12;//convergence
tolerance in terms of force
            //It will include the loads that have been defined as "L"
            NLcase.InputLoadCombo = new LoadCombination("NLcase
loads", ComboType.ADD);
            NLcase.InputLoadCombo.Items.Add(new
LoadCaseWithFactor("L", 1));
            //Definition of stiffness matrix update mode
```

```
            NLcase.UpdateStiffnessMethod =
GeometricNonLinearCase.UpdateStiffnessMatrixMethod.AfterEachIteration
InLoadStep;
            NLcase.SaveResultsAtEachLoadStep = true;//Results will be
saved at all intermediate load steps
            Model.GeometricNonLinearCases.Add(NLcase);

            //-------SOLUTION PHASE-------
            Model.Solve();

            //-------OBTAIN RESULTS-------
            double[] Min, Max;
            for (int loadStep = 1; loadStep <= NLcase.LoadSteps;
loadStep++)
            {
                //Get horizontal displacement of top node of the
column at each load step
                n2.GetNodalDisplacementsForLoadCase("NL", out Min,
out Max, loadStep);
                double horDisplacement = Min[0];

                //Get bending moment at the base of the column at
each load step
                el1.GetInternalForcesForLoadCase(0, "NL", out Min,
out Max, loadStep);
                double BaseMoment = Min[4];
            }
```

Example 10: Frame on Winkler springs

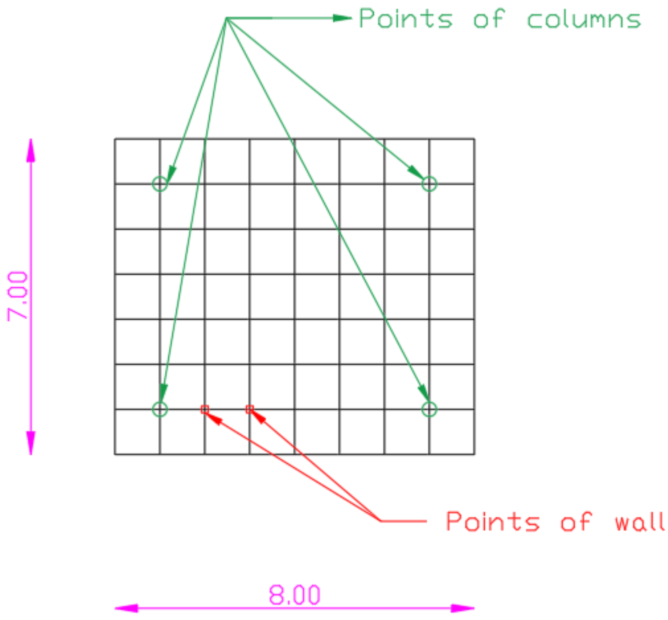


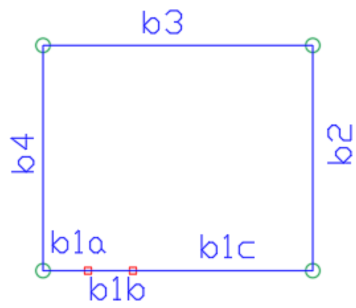

```
            //New model definition
            Model Model = new Model();
            Model.LicenseInfo = LicenseInfo;

            //-------MATERIAL DEFINITION-------

            //Create a new material for concrete
            Material matConcrete = new Material();
            matConcrete.Name = "Concrete";//Material name
            matConcrete.Density = 2.5;//density in mass units/m3, for
example tn/m3
            matConcrete.G = 11538461;//shear modulus
            matConcrete.E = 30000000;//elasticity modulus

            //-------SECTIONS DEFINITION-------

            //Create a new beam section of dimensions 40cmx80xm
            FrameElementSection secBeam40_80 = new
FrameElementSection();
            secBeam40_80.Name = "Beam40/80";//section name
            secBeam40_80.A = 0.4 * 0.8;//section area
            secBeam40_80.Iy = 0.4 * 0.8 * 0.8 * 0.8 / 12;//inertia
moment about local y axis
            secBeam40_80.Iz = 0.8 * 0.4 * 0.4 * 0.4 / 12;//inertia
moment about local z axis
            secBeam40_80.It = 0.0117248;//torsional constant
            secBeam40_80.b = 0.40;//section width
            secBeam40_80.h = 0.80;//section height

            //-------MODEL GEOMETRY AND LOADS DEFINITION-------

            //Create node n1
            Frame3D.SuperNode n1 = new Frame3D.SuperNode(1, 0, 0, 0);
            n1.dof1constraint = true;//delete
            n1.dof2constraint = true;//translational constraint in
direction y at local system of node
            n1.dof3constraint = true;//translational constraint in
direction z at local system of node
            n1.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n1.dof5constraint = true;//rotational constraint in
direction y at local system of node
            Model.InputNodes.Add(n1);

            //Create node n2
```

```
            Frame3D.SuperNode n2 = new Frame3D.SuperNode(2, 5, 0, 0);
            n2.dof1constraint = true;//translational constraint in
direction x at local system of node
            n2.dof2constraint = true;//translational constraint in
direction y at local system of node
            n2.dof3constraint = true;//translational constraint in
direction z at local system of node
            n2.dof4constraint = true;//rotational constraint in
direction x at local system of node
            n2.dof5constraint = true;//rotational constraint in
direction y at local system of node
            Model.InputNodes.Add(n2);

            //Create frame element 1
            //Note that the 4th argument specifies the auxiliary
point that lies in the xy plane that is formed by the x and y axes in
the local element system
            FrameSuperElement el1 = new FrameSuperElement(1, n1, n2,
new Geometry.XYZ(0, 0, 1), matConcrete, secBeam40_80, new
MemberReleases(), new MemberReleases(), false, false);
            el1.WinklerStiffnessZ = 15000;//Winkler spring constant
on local Z frame axis (units: force/length/length)

            LinearLoadCaseForSuperFrameElement lc1 = new
LinearLoadCaseForSuperFrameElement("lc1", LoadCaseType.DEAD);
            lc1.UniformLoad.UniformLoadsY.Add(new
FrameSuperUniformLoad(0, 1, -10, -10,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Global));
            lc1.PointLoad.PointLoadsY.Add(new SuperPointLoad(3.5, -
50, LoadDefinitionFromStartingNode.Absolutely,
LoadCordinateSystem.Global));
            el1.LinearLoadCasesList.Add(lc1);

            Model.InputFiniteElements.Add(el1);

            //-------SOLUTION PHASE-------
            Model.Solve();

            //-------OBTAIN RESULTS-------
            double[] Min, Max;//The combination results will be saved
in these arrays
            //Note that the definition of two arrays for minimum and
maximum combination results is required
            //For combination type "ADD", Min and Max values are
always equal

            //Reactions (All are defined in the node local system)
            //Rections for load case lc1
            n1.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n1_Rty_lc1 = Max[1];
            n2.GetReactionsForLoadCase("lc1", out Min, out Max, 0);
            double n2_Rty_lc1 = Max[1];

            //Node Displacements (All are defined in the node local
system)
            //Note that constained degrees of freedom have zero
displacements
            n1.GetNodalDisplacementsForLoadCase("lc1", out Min, out
Max, 0);
            double[] n1_Disp = Max;
```

```
            n2.GetNodalDisplacementsForLoadCase("lc1", out Min, out
Max, 0);
            double[] n2_Disp = Max;

            //Element internal forces and displacements
            el1.GetInternalForcesForLoadCase(0, "lc1", out Min, out
Max, 0); //Internal forces at the start of the member
            double[] forces_along_member_left = Max;
            el1.GetInternalForcesForLoadCase(2.5, "lc1", out Min, out
Max, 0);//Internal forces at the middle of the member
            double[] forces_along_member_middle = Max;
            el1.GetInternalForcesForLoadCase(5, "lc1", out Min, out
Max, 0);//Internal forces at the end of the member
            double[] forces_along_member_right = Max;

            el1.GetDisplacementsForLoadCase(0, "lc1", out Min, out
Max, 0); //Internal displacements at the start of the member
            double[] disps_along_member_left = Max;
            el1.GetDisplacementsForLoadCase(2.5, "lc1", out Min, out
Max, 0);//Internal displacements at the middle of the member
            double[] disps_along_member_middle = Max;
            el1.GetDisplacementsForLoadCase(5, "lc1", out Min, out
Max, 0);//Internal displacements at the end of the member
            double[] disps_along_member_right = Max;

            el1.GetWinklerSpringReactionsForLoadCase(0, "lc1", out
Min, out Max, 0); //Soil reaction at the start of the member (units
in force: length)
            double[] soil_reaction_left = Max;
            el1.GetWinklerSpringReactionsForLoadCase(2.5, "lc1", out
Min, out Max, 0); //Soil reaction at the middle of the member (units
in force: length)
            double[] soil_reaction_middle = Max;
            el1.GetWinklerSpringReactionsForLoadCase(5, "lc1", out
Min, out Max, 0); //Soil reaction at the end of the member (units in
force: length)
            double[] soil_reaction_right = Max;
```

Example 11: Single storey building with shell elements (wall, foundation slab and superstructure slab)





Story 1 (Beam schedule)

Foundation

```
            //New model definition
            Model Model = new Model();
            Model.LicenseInfo = LicenseInfo;

            //Create a new material for concrete
            Material matConcrete = new Material();
            matConcrete.Name = "Concrete";//Material name
            matConcrete.Density = 2.5;//density in mass units/m3, for
example tn/m3
            matConcrete.G = 30000000 / 2.6;//shear modulus
            matConcrete.E = 30000000;//elasticity modulus

            //Create a cross section for frames
            FrameElementSection ConcreteSection = new
FrameElementSection();
            ConcreteSection.Name = "Concrete section";//section name
            ConcreteSection.A = 0.4 * 0.8;//section area
            ConcreteSection.Iy = 0.4 * 0.8 * 0.8 * 0.8 / 12;//inertia
moment about local y axis
            ConcreteSection.Iz = 0.8 * 0.4 * 0.4 * 0.4 / 12;//inertia
moment about local z axis
            ConcreteSection.It = 0.0117248;//torsional constant
            ConcreteSection.b = 0.40;//section height
            ConcreteSection.h = 0.80;//section height

            Frame3D.SuperNode n1, n2, n3, n4;
            ShellSuperElementTriangular ssel1;
            ShellSuperElementTriangular ssel2;

            double soilStiffnessPerArea = 5000;

            int i_nodes = 1;
```

```
            //Create foundation nodes (these nodes will be used to
create the shell elements of the foundation slab)
            for (double x = 0; x <= 7; x += 1.0)
            {
                for (double y = 0; y <= 8; y += 1.0)
                {
                    Frame3D.SuperNode n = new
Frame3D.SuperNode(i_nodes++, x, y, 0);
                    n.dof1constraint = true;
                    n.dof2constraint = true;
                    n.Kdof3 = soilStiffnessPerArea;
                    n.dof6constraint = true;
                    Model.InputNodes.Add(n);
                }
            }

            //The foundation slab elements are created below. The
thickness of the foundation is 0.5 m.
            int i_elements = 1;
            for (double y = 1; y <= 8; y += 1.0)
            {
                for (double x = 1; x <= 7; x += 1.0)
                {
                    double y12 = y - 1;
                    double y34 = y;

                    double x14 = x - 1;
                    double x23 = x;

                    n1 = Model.InputNodes.Find(p => p.x == x14 && p.y
== y12 && p.z == 0);
                    n2 = Model.InputNodes.Find(p => p.x == x23 && p.y
== y12 && p.z == 0);
                    n3 = Model.InputNodes.Find(p => p.x == x23 && p.y
== y34 && p.z == 0);
                    n4 = Model.InputNodes.Find(p => p.x == x14 && p.y
== y34 && p.z == 0);

                    //ssel = new ShellSuperElement(i_elements++, n1,
n2, n3, n4, matConcrete);
                    ssel1 = new
ShellSuperElementTriangular(i_elements++, n1, n2, n3, matConcrete);
                    ssel1.Thickness = 0.5;
                    ssel1.Spring_Z_local_Stiffness_per_Area =
soilStiffnessPerArea;
                    Model.InputFiniteElements.Add(ssel1);
                    ssel2 = new
ShellSuperElementTriangular(i_elements++, n1, n3, n4, matConcrete);
                    ssel2.Thickness = 0.5;
                    ssel2.Spring_Z_local_Stiffness_per_Area =
soilStiffnessPerArea;
                    Model.InputFiniteElements.Add(ssel2);
                }
            }

            //The nodes of the story are created
            Frame3D.SuperNode ns1 = new Frame3D.SuperNode(i_nodes++,
1, 1, 3);
            Model.InputNodes.Add(ns1);
```

```
            Frame3D.SuperNode ns2 = new Frame3D.SuperNode(i_nodes++,
6, 1, 3);
            Model.InputNodes.Add(ns2);

            Frame3D.SuperNode ns3 = new Frame3D.SuperNode(i_nodes++,
6, 7, 3);
            Model.InputNodes.Add(ns3);

            Frame3D.SuperNode ns4 = new Frame3D.SuperNode(i_nodes++,
1, 7, 3);
            Model.InputNodes.Add(ns4);


            //The following two nodes belong to the vertical wall at
elevation z=1.00 and z=2.00
            Frame3D.SuperNode nsw1_z1 = new
Frame3D.SuperNode(i_nodes++, 2, 1, 1);
            Model.InputNodes.Add(nsw1_z1);
            Frame3D.SuperNode nsw2_z1 = new
Frame3D.SuperNode(i_nodes++, 3, 1, 1);
            Model.InputNodes.Add(nsw2_z1);

            Frame3D.SuperNode nsw1_z2 = new
Frame3D.SuperNode(i_nodes++, 2, 1, 2);
            Model.InputNodes.Add(nsw1_z2);
            Frame3D.SuperNode nsw2_z2 = new
Frame3D.SuperNode(i_nodes++, 3, 1, 2);
            Model.InputNodes.Add(nsw2_z2);


            //The shell elements of the wall are created. (the wall
has a thickness of 0.25 m)
            n1 = Model.InputNodes.Find(p => p.x == 2 && p.y == 1 &&
p.z == 0);
            n2 = Model.InputNodes.Find(p => p.x == 3 && p.y == 1 &&
p.z == 0);
            n3 = Model.InputNodes.Find(p => p.x == 2 && p.y == 1 &&
p.z == 1);
            n4 = Model.InputNodes.Find(p => p.x == 3 && p.y == 1 &&
p.z == 1);
            ssel1 = new ShellSuperElementTriangular(i_elements++, n1,
n2, n3, matConcrete);
            ssel1.Thickness = 0.25;
            Model.InputFiniteElements.Add(ssel1);
            ssel2 = new ShellSuperElementTriangular(i_elements++, n1,
n3, n4, matConcrete);
            ssel2.Thickness = 0.25;
            Model.InputFiniteElements.Add(ssel2);

            n1 = Model.InputNodes.Find(p => p.x == 2 && p.y == 1 &&
p.z == 1);
            n2 = Model.InputNodes.Find(p => p.x == 3 && p.y == 1 &&
p.z == 1);
            n3 = Model.InputNodes.Find(p => p.x == 2 && p.y == 1 &&
p.z == 2);
            n4 = Model.InputNodes.Find(p => p.x == 3 && p.y == 1 &&
p.z == 2);
            ssel1 = new ShellSuperElementTriangular(i_elements++, n1,
n2, n3, matConcrete);
            ssel1.Thickness = 0.25;
            Model.InputFiniteElements.Add(ssel1);
```

```
            ssel2 = new ShellSuperElementTriangular(i_elements++, n1,
n3, n4, matConcrete);
            ssel2.Thickness = 0.25;
            Model.InputFiniteElements.Add(ssel2);

            n1 = Model.InputNodes.Find(p => p.x == 2 && p.y == 1 &&
p.z == 2);
            n2 = Model.InputNodes.Find(p => p.x == 3 && p.y == 1 &&
p.z == 2);
            //The upper wall joints are created here:
            n3 = new SuperNode(i_nodes++, 2, 1, 3);
            LinearLoadCaseForSuperNode llcsn = new
LinearLoadCaseForSuperNode("lc1", LoadCaseType.DEAD);
            llcsn.Pz = -100;
            n3.LinearLoadCasesList.Add(llcsn);

            Model.InputNodes.Add(n3);
            n4 = new SuperNode(i_nodes++, 3, 1, 3);
            Model.InputNodes.Add(n4);
            ssel1 = new ShellSuperElementTriangular(i_elements++, n1,
n2, n3, matConcrete);
            ssel1.Thickness = 0.25;
            Model.InputFiniteElements.Add(ssel1);
            ssel2 = new ShellSuperElementTriangular(i_elements++, n1,
n3, n4, matConcrete);
            ssel2.Thickness = 0.25;
            Model.InputFiniteElements.Add(ssel2);

            //The columns are created below
            Frame3D.FrameSuperElement c1 = new
FrameSuperElement(i_elements++, Model.InputNodes[10], ns1, new
Geometry.XYZ(1, 2, 0), matConcrete, ConcreteSection, new
MemberReleases(), new MemberReleases(), true, false);
            Model.InputFiniteElements.Add(c1);

            Frame3D.FrameSuperElement c2 = new
FrameSuperElement(i_elements++, Model.InputNodes[55], ns2, new
Geometry.XYZ(6, 2, 0), matConcrete, ConcreteSection, new
MemberReleases(), new MemberReleases(), true, false);
            Model.InputFiniteElements.Add(c2);

            Frame3D.FrameSuperElement c3 = new
FrameSuperElement(i_elements++, Model.InputNodes[61], ns3, new
Geometry.XYZ(6, 8, 0), matConcrete, ConcreteSection, new
MemberReleases(), new MemberReleases(), true, false);
            Model.InputFiniteElements.Add(c3);

            Frame3D.FrameSuperElement c4 = new
FrameSuperElement(i_elements++, Model.InputNodes[16], ns4, new
Geometry.XYZ(1, 8, 0), matConcrete, ConcreteSection, new
MemberReleases(), new MemberReleases(), true, false);
            Model.InputFiniteElements.Add(c4);

            //Create beams
            LinearLoadCaseForSuperFrameElement beamLoad = new
LinearLoadCaseForSuperFrameElement("lc1", LoadCaseType.DEAD);
            beamLoad.UniformLoad.UniformLoadsZ.Add(new
FrameSuperUniformLoad(0, 1, -10, -10,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Global));
```

```
            //Beam starts from column c1 and ends at column c2. It
also intersects the wall. Thus it is modelled using 3 parts.
            Frame3D.FrameSuperElement b1a = new
FrameSuperElement(i_elements++, ns1, n3, new Geometry.XYZ(1, 2, 3),
matConcrete, ConcreteSection, new MemberReleases(), new
MemberReleases(), true, false);
            b1a.LinearLoadCasesList.Add(beamLoad);
            Model.InputFiniteElements.Add(b1a);
            Frame3D.FrameSuperElement b1b = new
FrameSuperElement(i_elements++, n3, n4, new Geometry.XYZ(1, 2, 3),
matConcrete, ConcreteSection, new MemberReleases(), new
MemberReleases(), true, false);
            b1b.LinearLoadCasesList.Add(beamLoad);
            Model.InputFiniteElements.Add(b1b);
            Frame3D.FrameSuperElement b1c = new
FrameSuperElement(i_elements++, n4, ns2, new Geometry.XYZ(1, 2, 3),
matConcrete, ConcreteSection, new MemberReleases(), new
MemberReleases(), true, false);
            b1c.LinearLoadCasesList.Add(beamLoad);
            Model.InputFiniteElements.Add(b1c);

            Frame3D.FrameSuperElement b2 = new
FrameSuperElement(i_elements++, ns2, ns3, new Geometry.XYZ(5, 0, 3),
matConcrete, ConcreteSection, new MemberReleases(), new
MemberReleases(), true, false);
            b2.LinearLoadCasesList.Add(beamLoad);
            Model.InputFiniteElements.Add(b2);

            Frame3D.FrameSuperElement b3 = new
FrameSuperElement(i_elements++, ns3, ns4, new Geometry.XYZ(6, 8, 3),
matConcrete, ConcreteSection, new MemberReleases(), new
MemberReleases(), true, false);
            b3.LinearLoadCasesList.Add(beamLoad);
            Model.InputFiniteElements.Add(b3);

            Frame3D.FrameSuperElement b4 = new
FrameSuperElement(i_elements++, ns4, ns1, new Geometry.XYZ(0, 7, 3),
matConcrete, ConcreteSection, new MemberReleases(), new
MemberReleases(), true, false);
            b4.LinearLoadCasesList.Add(beamLoad);
            Model.InputFiniteElements.Add(b4);

            //Add loads to beams
            LinearLoadCaseForSuperFrameElement llcsfe = new
LinearLoadCaseForSuperFrameElement("lc1", LoadCaseType.DEAD);
            llcsfe.UniformLoad.UniformLoadsZ.Add(new
FrameSuperUniformLoad(0, 1, -10, -10,
LoadDefinitionFromStartingNode.Relatively,
LoadCordinateSystem.Global));
            b1a.LinearLoadCasesList.Add(llcsfe);
            b1b.LinearLoadCasesList.Add(llcsfe);
            b1c.LinearLoadCasesList.Add(llcsfe);
            b2.LinearLoadCasesList.Add(llcsfe);
            b3.LinearLoadCasesList.Add(llcsfe);
            b4.LinearLoadCasesList.Add(llcsfe);

            Model.Solve();

            //-------OBTAIN RESULTS-------
            double[] Min, Max;
```

```
            //Beam forces
            b1a.GetInternalForcesForLoadCase(0, "lc1", out Min, out
Max, 1);

            //Foundation deflection at wall base x=3, y=1, z=0
            SuperNode nf = Model.InputNodes.Find(p => p.x == 3 && p.y
== 1 && p.z == 0);
            nf.GetNodalDisplacementsForLoadCase("lc1", out Min, out
Max, 1);

            //Estimation of soil stress at wall base
            double soilStress = -soilStiffnessPerArea * Min[2];


            //Get stresses at the base of wall (stresses of sheels
are reported in the local coordinate system!!)

((ShellSuperElementTriangular)Model.InputFiniteElements.Find(p =>
((ShellSuperElementTriangular)p).Node3.z ==
1)).GetInternalStressesForLoadCase(0, 0, 0.125,
CoordinateSystem.Local, 0, "lc1", out Min, out Max, 1);

            //Get forces at selected node (Fx,Fy,Fz,Mx,My,Mz are
reported on the local or global coordinate system)

((ShellSuperElementTriangular)Model.InputFiniteElements.Find(p =>
((ShellSuperElementTriangular)p).Node3.z ==
1)).GetInternalForcesForLoadCase(ShellTriangularResultsLocation.Point
1, CoordinateSystem.Global, "lc1", out Min, out Max, 1);
```