CEE532 Developing Software for Engineering Applications

PLANAR FRAME PROGRAM REPORT

DHARANIDHARAN ARUMUGAM

Fall 2023

Table of Contents

1.0 Problem Statement

To develop a C++ program to carry out planar frame analysis with the use of object-oriented programming.

2.0 Theory

Planar frame structures are composed of slender frame elements that can resist the loading through bending in addition to the axial resistance. Analysis of planar frame element using FEM approaches involves 1. Discretization of structures (node numbering and element numbering), 2. Setting up element equilibrium-compatibility equations, 3. Assembling structural stiffness matrices and forms global system of equations 4. Apply boundary conditions 5. Solving the equilibrium equations and 6. Find the nodal (displacements and support reactions) and element responses (stress, strain, and force in the members).

To solve for the nodal displacements, support reactions and member forces, for each frame member, force – displacement equations in the form of matrices are formed along the local axis of the member. The equations are then transformed to global direction and the system of equations are assembled. Fig. 1 shows the local and global coordinates system of a truss member.

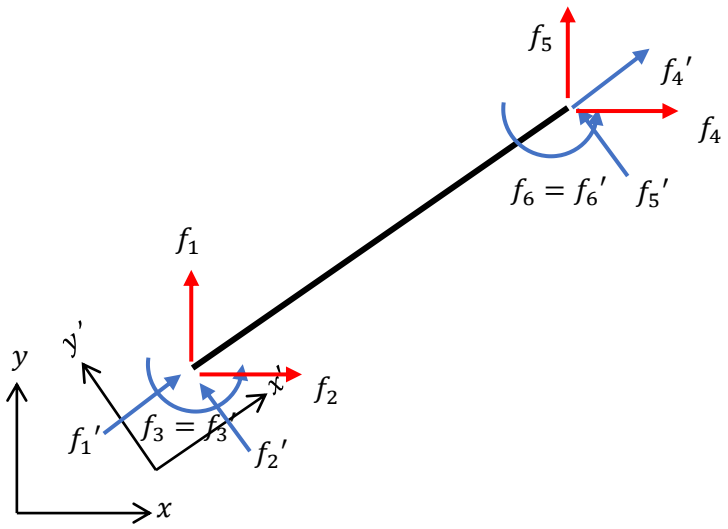


Fig. 2.1: Global and local coordinate system of a frame member

The following symbolic frame matrix equations capture the steps involved in transforming the forces and displacement to global system and vice versa.

$$k'_{6\times6}\, d'_{6\times1} - q'_{6\times1} = f'_{6\times1} \tag{2.1}$$

$$d'_{6\times1} = T_{6\times6}\, d_{6\times1} \tag{2.2}$$

$$f_{6\times1} = T'_{6\times6}\, f'_{6\times1} \tag{2.3}$$

$$k_{6\times6}\, d_{6\times1} = f_{6\times1} \tag{2.4}$$

$$k_{6\times6} = T'_{6\times6} k'_{6\times6}\, T_{6\times6} \tag{2.5}$$

The transformation matrix (T) in the above equations is given as,

$$T_{6\times6} = \begin{bmatrix} l & m & 0 & 0 & 0 & 0 \\ -m & l & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & l & m & 0 \\ 0 & 0 & 0 & -m & l & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

Here, l and m are direction cosines of the truss member.
The element stiffness matrix for an element in local coordinates is given by,

$$K_{6\times6} = \begin{bmatrix} AE/L & 0 & 0 & -AE/L & 0 & 0 \\ 0 & 12EI/L^3 & 6EI/L^2 & 0 & -12EI/L^3 & 6EI/L^2 \\ 0 & 6EI/L^2 & 4EI/L & 0 & -6EI/L^2 & 2EI/L \\ -AE/L & 0 & 0 & AE/L & 6EI/L^2 & 0 \\ 0 & -12EI/L^3 & -6EI/L^2 & 6EI/L^2 & 12EI/L^3 & -6EI/L^2 \\ 0 & 6EI/L^2 & 2EI/L & 0 & -6EI/L^2 & 4EI/L \end{bmatrix} \tag{2.7}$$

These element equations must be appropriately modified if there is a hinge in the beam[1]. Once the system equations have been set up, it can be solved through direct or iterative solvers. Generally, Cholesky decomposition technique is used and implemented the same in the current program. The algorithm and implementation of the Cholesky solver will be explained in the subsequent sections.

3.0 PROGRAM IMPLEMENTATION

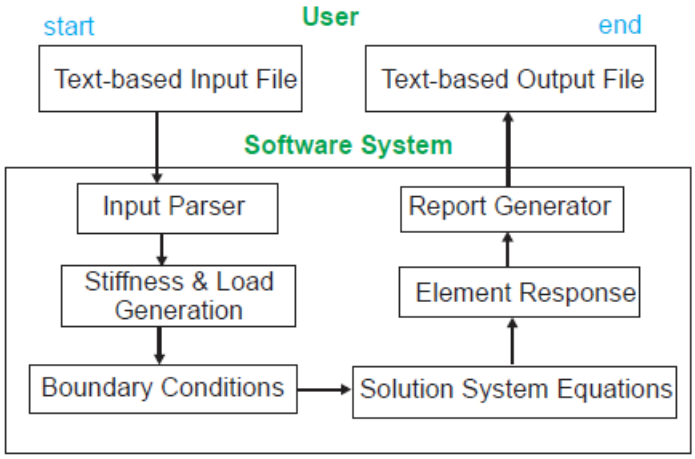


Fig. 3.1: Program flow for the space truss analysis (reproduced from [2])

The structure and flow of the program is illustrated in Fig. 1 which is based on the structure given in [2]. The program implementation requires several components: starting from parsing the input from the input text file, store the read data in the class objects, develop system stiffness matrix and compute nodal loads, apply boundary condition, solve system of equations using $\mathrm{LDL^T}$ decomposition, finding the element response such as stress, strain, and forces, and finally write the output to a text file in a specified format.

4.0 The CFRAME Class

The developed program is implemented with creating several class definitions; the relationship between major individual classes is shown in Fig. 4.1.

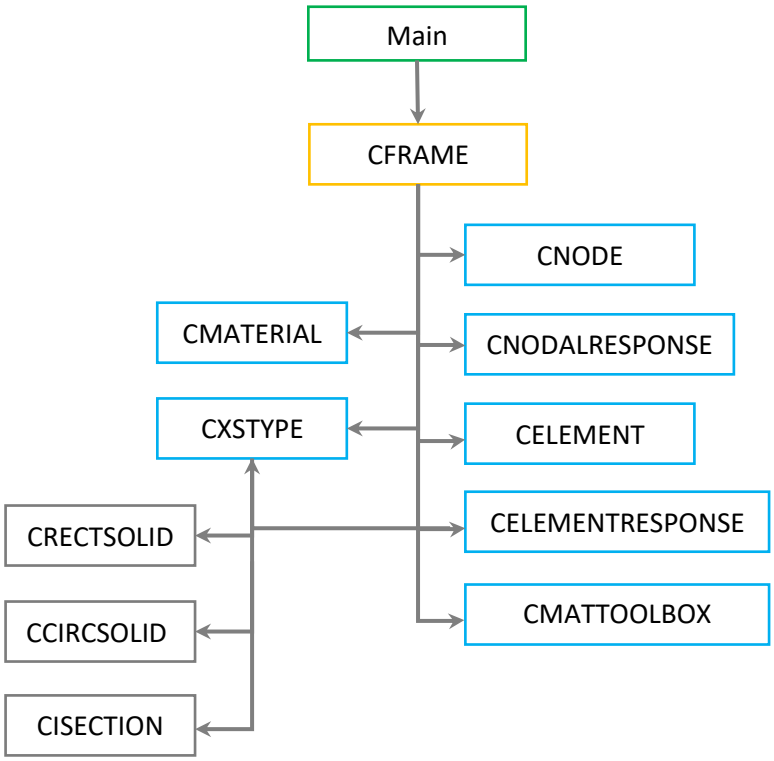


Figure 4.1: Class relationship of other major classes with CTRUSS class

The following table shows the summary of the functions involved with each of these major classes,

CNODE	CNODALRESPONSE	CELEMENT	CELEMENTRESPONSE
Store and retrieve: • node coordinate values • node forces • node fixities • thermal loads	Store and retrieve: • node displacements • support reactions	Store and retrieve: • element properties (length, direction cosines, E and CTE) • element connectivity	Store and retrieve: • element maximum internal forces • element maximum stresses • element nodal forces

CXSTYPE	CPROPERTY
Store and retrieve: • sectional properties (area, I, SM) • dimensions	Store and retrieve: • material property such as E, mu and CTE

In addition, CMATTOOLBOX is used to access the linear system of equations solver. In the subsequent sections, the functional prototypes of the major classes are presented.

4.1 CFRAME CLASS

The following functional prototypes shows the list of the functions performed by the CFRAME class,

```
void Banner (std::ostream& OF);
void PrepareIO (int argc, char *argv[]);
void Analyze ();
void TerminateProgram ();
void DisplayErrorMessage (CLocalErrorHandler::ERRORCODE);

// FEA-related functions
void ReadProblemSize ();
void ReadFrameModel ();
void ElementStiffness(const float fE, const float fA, const float fI,
                      const float fL, const double l, const double m,
                      const bool bLHinged, const bool bRHinged, CMatrix<double>& dMKe);
void ConstructSystemKandF ();
void Solve ();
void Response ();
void ComputeElementForces();
void ComputePeakStresses (const int nSections, const int ElNo,
                          const float fL, const float fA,
                          const float fS, const float fSF);
void ComputeSectionalLoads (const int ElNo, const float fL, const float fX,
                            float& fAF, float& fSF, float& fBM);
void MaxStresses (const float fA, const float fS, const float fSF,
                  const float fP, const float fV, const float fM,
                  float& fCS, float& fTS, float& fSS);
void CFrame::ObtainSupportReactions();
void CreateOutput ();
void PrintPropertyData();
void PrintNodalData();
void PrintElementData();
void PrintResponseData();
void PrintStatistics (CPrintTable& LoadTable, int nColumns, std::string strTag);
void PrintStatistics(CPrintTable& LoadTable, CVector<int> nVTags,
                     int nColumns, std::string strTag);

// modifier functions
void SetSize ();
void SetSystemSize();

// error handlers
void ErrorHandler (CLocalErrorHandler::ERRORCODE);        // gateway to local error handler
void ErrorHandler (CGlobalErrorHandler::ERRORCODE) const;// gateway to global error handler
void IOErrorHandler (ERRORCODE nCode) const;


void CFrame::Analyze ()
// -------------------------------------------------------------------------
// Function: Reads the input data and analyzes the frame
// Input:    none
// Output:   none
// -------------------------------------------------------------------------
{
        // read the problem size
        ReadProblemSize ();

        // set problem size
        SetSize ();

        // read nodal and element data
        ReadFrameModel ();

        // construct system equations
```

```
        ConstructSystemKandF ();


        // solve for the nodal displacements
        Solve ();

        // compute secondary unknowns
        Response ();

        // extract support reactions and find error norms
        ObtainSupportReactions();

        // create output file
        CreateOutput ();
}
```

4.2 CNODE CLASS

The following functional prototypes shows the list of the functions performed by the CNODE class,

```
// accessor functions
void GetCoords (CVector<float>& fVC) const;
void GetFixity (CVector<Fixity>& VFC) const;
float GetNodalTemperature() const;

// modifier functions
void SetCoords (const CVector<float>& fVC);
void SetFixity (const CVector<Fixity>& VFC);
void SetNodalTemperature(const float fdT);

// helper functions
float ToDistance(const CNode& Nd) const;
void DirectionCosines(const CNode& Nd, CVector<double>& dVDC) const;
```

4.3 CNODALRESPONSE CLASS

The following functional prototypes shows the list of the functions performed by the CNODALRESPONSE class,

```
// accessor functions
void GetDisplacements (CVector<float>& fVD) const;

void GetReactions (CVector<float>& fVR) const;

// modifier functions
void SetDisplacements (const CVector<float>& fVD);

void SetReactions (const CVector<float>& fVR);
```

4.4 CELEMENT CLASS

The following functional prototypes shows the list of the functions performed by the CELEMENT class,

```
// accessor functions
void GetENodes (int& nSN, int& nEN) const;
int  GetMatPropertyGroup () const;
CXSType* GetEPropertyGroup () const;
int  GetEPropertyGroupNo() const;
float GetLength() const;
void GetDirectionCosines (double& dl, double& dm) const;

// modifier functions
void SetENodes (const int nSN, const int nEN);
void SetEPropertyGroup (const int nEPG, CXSType*);
void SetMatPropertyGroup (const int);
void SetLength (const float);
void SetDirectionCosines (const double dl, const double dm);
```

4.5 CELEMENTRESPONSE CLASS

The following functional prototypes shows the list of the functions performed by the CELEMENTRESPONSE class,

```
// accessor functions
void GetForces (CVector<float>& fVFStartNode,
                CVector<float>& fVFEndNode) const;
void GetMaxStresses (float& fMaxCS, float& fMaxTS,
                     float& fMaxSS) const;
void GetMaxForces (float& fMaxP, float& fMaxV,
                   float& fMaxM) const;

// modifier functions
void SetForces (const CVector<float>& fVFStartNode,
                const CVector<float>& fVFEndNode);
void SetMaxStresses (const float fMaxCS, const float fMaxTS,
                     const float fMaxSS);
void SetMaxForces (const float fMaxP,const float fMaxV,
                   const float fMaxM);
```

4.6 CXSTYPE CLASS

The following functional prototypes shows the list of the functions performed by the CELEMENTRESPONSE class,

```
void GetProperties (float& fA, float& fIyy, float& fIzz);
void GetDimensions (CVector<float>&) const;
void GetSectionModuli (float& fSyy, float& fSzz);
void GetShearFactors (float& fSFy, float& fSFz);

virtual void ComputeProperties () = 0;
```

5.0 IMPORTANT PROGRAM IMPLEMENTATIONS

The developed program involves numerous functional implementations. In the current section, the algorithms and the corresponding program implementations are discussed.

5.1 STORING THE EFFECTIVE SYSTEM STIFFNESS MATRIX AND FORCES

In the current implementation, the system stiffness matrix is stored as full-size matrix with effective degrees of freedom size. The following algorithm is implemented for constructing the effective stiffness matrix and force vector.

1. Loop through elements.
2. Obtain the element node number.
3. Obtain the element fixity conditions.
4. Find the element properties (area, MI, length, direction cosines).
5. Find element stiffness matrices and load vectors in global coordinates system.
6. Obtain the nodal displacements (needed for specified displacements)
7. Loop through rows indices of the element, $i = 1\ to\ 6$
8. Loop through column indices of the element , $j = 1\ to\ 6$
9. Find the DOFs of the full K matrix
10. Find the DOFs of the effective K matrix (i_A, j_A)
11. If the ADOF > 0 , $K_{eff}\ (i_A, j_A) \mathrel{+}= Ke(i,j)$
12. Else check there is any specified nonzero displacement (d)
13. If there is a specified nonzero displacement, $F_{eff}\ (j_A) \mathrel{-}= d\ Ke(i,j)$

Code:

```
void CFrame::ConstructSystemKandF()
// ------------------------------------------------------------------------
// Function: constructs the system stiffness matrix and force vector
// Input:    none
// Output:   none
// ------------------------------------------------------------------------
{
    int nSN, nEN, nMatGrp;
    CXSType* pXSGrp;
    float fE, fA, fL, fCTE, fI, fIyy, fIzz;
    double dl, dm;
    CVector<float> fVC1(NDIM), fVC2(NDIM);
    CVector<CNode::Fixity> VFC1(DOFPN), VFC2(DOFPN);
    CMatrix<double> dMKe(DOFPE, DOFPE);
    CMatToolBox<double> MTB;

    // loop throught the elements
    for (int ElNo = 1; ElNo <= m_nElements; ElNo++)
    {

        // ------------------------------------------------------------------
        //                  GETTING THE ELEMENT PROPERTIES
        // ------------------------------------------------------------------

        //Getting the numbers of the element nodes in nSN and nEN
        m_ElementData(ElNo).GetENodes(nSN, nEN);

        //Getting the fixities of the start and end node in VFC1 and VFC2 resp.
        m_NodalData(nSN).GetFixity(VFC1);
        m_NodalData(nEN).GetFixity(VFC2);

        //obtaining the pointer of the element c/s group
        pXSGrp = m_ElementData(ElNo).GetEPropertyGroup();
        pXSGrp->GetProperties(fA, fIyy, fIzz);

        //obtaining the members E and CTE based on  material group
        nMatGrp = m_ElementData(ElNo).GetMatPropertyGroup();
        fE = m_MaterialData(nMatGrp).GetYM();
        fCTE = m_MaterialData(nMatGrp).GetCTE();

        //finding the element length
        fL = m_ElementData(ElNo).GetLength();

        //Finding MI for the bending
        fI = fIzz;
        if (fIyy > fIzz) fI = fIyy;

        //finding the direction cosines
        m_ElementData(ElNo).GetDirectionCosines(dl, dm);

        //finding the boolean value indicating the left and right node is hinged or not
        bool bLH = (VFC1(3) == CNode::Fixity::HINGED ? true : false);
        bool bRH = (VFC2(3) == CNode::Fixity::HINGED ? true : false);


        // ------------------------------------------------------------------
        //       ASSEMBLING THE Effective K matrix and F vector
        // ------------------------------------------------------------------

        //finding the element stiffness matrix
        ElementStiffness(fE, fA, fI, fL, dl, dm, bLH, bRH, dMKe);

        CVector<float> fVD1(DOFPN), fVD2(DOFPN), fVD(DOFPN);
        m_NodalResponseData(nSN).GetDisplacements(fVD1);
        m_NodalResponseData(nEN).GetDisplacements(fVD2);
```

```
int nNi, nNj, nAi, nAj, DOFi, DOFj;
float fD;

//loop through the element degrees of freedom
for (int i = 1; i <= DOFPE; i++)
{
    // find the degrees of freedom
    nNi     = (i <= 3 ? nSN - 1 : nEN - 2);
    DOFi    = nNi * DOFPN + i;
    nAi     = m_SADOF(DOFi); // find the active dof no

    if (nAi > 0) // if the node active
    {
        // loop through the elements
        for (int j = 1; j <= DOFPE; j++)
        {
            nNj     = (j <= 3 ? nSN - 1 : nEN - 2);
            DOFj    = nNj * DOFPN + j;
            nAj     = m_SADOF(DOFj);

            // if the dof active
            if (nAj > 0)
            {
                m_SSM(nAi, nAj) += dMKe(i, j); //store the structural stiffness matrix
            }
            else
            {
                fD = (j <= 3 ? fVD1(j) : fVD2(j-DOFPN));
                if (abs(fD) > 0.0f)
                {
                    m_SNF(nAi) -= double(fD) * dMKe(i, j);  //store the structural nodal
                                                               forces
                }
            }
        }
        m_SNF(nAi) += double (m_ELG(i, ElNo)); //store the structural nodal forces
    }

}

std::string strElem = std::to_string(ElNo);
if (m_nDebugLevel == 1)
{
    MTB.PrintMatrixRowWise(dMKe, "Element Stiffness " + strElem, m_FileOutput);
}
```

5.2 FINDING THE RESPONSE

Computing the structural response involves finding the member forces in local and global directions, finding the maximum internal forces and internal stresses, and finding the support reactions.

1. Loop through elements.
2. Obtain the element fixity conditions, length and direction cosines.
3. Obtain the element nodal displacements in global coordinates.
4. Find the nodal displacements in local coordinates.
5. Recover the rotations if the hinges are present.
6. Use Equation 2.1 to obtain the element forces in local direction.
7. Find the element forces in global direction.
8. Find the maximum internal forces and stresses in the member.
9. Loop through degrees of freedom of the element (DOF_e)
10. Residual vector, $R(DOF_e) \leftarrow R(DOF_e) + f_e$

- Finding support reactions
  1. Loop through nodes
  2. Find the nodal degrees of freedom.
  3. Find the structural degrees of freedom (DOF)
  4. Check whether DOF is constrained($DOF_{constrained}$) or not DOF_{free}.
  5. Support reactions, $F_R = R(DOF_{constrained})$
  6. Find absolute error norm $\left\|R^*(DOF_{free})\right\|$
  7. Find relative error norm $\left\|R^*(DOF_{free})\right\| / \left\|F^*(DOF_{free})\right\|$

Code:

```
void CFrame::Response()
// -------------------------------------------------------------------------
// Function: computes the element nodal forces and support reactions
// Input:    none
// Output:   none
// -------------------------------------------------------------------------
{
    int nSN, nEN, nMatGrp;
    CXSType* pXSGrp;
    float fE, fA, fL, fCTE;
    float fIyy, fIzz, fI, fSyy, fSzz, fS;
    float fSFy, fSFz, fSF;

    CVector<float> fVD1(DOFPN), fVD2(DOFPN);
    CVector<float> fVDL1(DOFPN), fVDL2(DOFPN);
    CVector<CNode::Fixity> VFC1(DOFPN), VFC2(DOFPN);
    CVector<float> fVEF1(DOFPN), fVEF2(DOFPN);
    CVector<double> fVEFG1(DOFPN), fVEFG2(DOFPN);


    for (int ElNo = 1; ElNo <= m_nElements; ElNo++)
    {
        // ---------------------------------------------------------------
        //                  GETTING THE ELEMENT PROPERTIES
        // ---------------------------------------------------------------

        //Getting the numbers of the element nodes in nSN and nEN
        m_ElementData(ElNo).GetENodes(nSN, nEN);

        //obtaining the pointer of the element c/s group
        pXSGrp = m_ElementData(ElNo).GetEPropertyGroup();
        pXSGrp->GetProperties(fA, fIyy, fIzz);

        //obtaining the members E and CTE based on  material group
        nMatGrp = m_ElementData(ElNo).GetMatPropertyGroup();
        fE = m_MaterialData(nMatGrp).GetYM();
        fCTE = m_MaterialData(nMatGrp).GetCTE();

        //finding the element length
        fL = m_ElementData(ElNo).GetLength();

        //Finding MI for the bending
        fI = fIzz;
        if (fIyy > fIzz) fI = fIyy;

        //finding the direction cosines
        double dl, dm;
        m_ElementData(ElNo).GetDirectionCosines(dl, dm);

        m_NodalResponseData(nSN).GetDisplacements(fVD1);
        m_NodalResponseData(nEN).GetDisplacements(fVD2);
        float fl = float(dl), fm = float(dm);
```

```
m_NodalData(nSN).GetFixity(VFC1);
m_NodalData(nEN).GetFixity(VFC2);

//finding the boolean value indicating the left and right node is hinged or not
bool bLH = (VFC1(3) == CNode::Fixity::HINGED ? true : false);
bool bRH = (VFC2(3) == CNode::Fixity::HINGED ? true : false);

// element displacements in local direction
fVDL1(1) = fl * fVD1(1) + fm * fVD1(2);
fVDL1(2) = -fm * fVD1(1) + fl * fVD1(2);
fVDL2(1) = fl * fVD2(1) + fm * fVD2(2);
fVDL2(2) = -fm * fVD2(1) + fl * fVD2(2);

// modify the element forces for the hinge
if (!bLH && !bRH)
{
    fVDL1(3) = fVD1(3);
    fVDL2(3) = fVD2(3);
}
else if (bLH && !bRH)
{
    fVDL2(3) = fVD2(3);
    fVDL1(3) = (1.5f/fL)*(-fVDL1(2) + fVDL2(2)) - 0.50f*fVDL2(3)
             + (fL/(4.0f*fE*fI))* m_ELL(3, ElNo);
}
else if (bRH && !bLH)
{
    fVDL1(3) = fVD1(3);
    fVDL2(3) = (1.5f / fL) * (-fVDL1(2) + fVDL2(2)) - 0.50f * fVDL1(3)
             + (fL / (4.0f * fE * fI)) * m_ELL(6, ElNo);
}
else
{
    fVDL1(3) = (1 / fL) * (-fVDL1(2) + fVDL2(2)) + (fL / (6.0f * fE * fI))
               * (2 * m_ELL(3, ElNo) - m_ELL(6, ElNo));
    fVDL2(3) = (1 / fL) * (-fVDL1(2) + fVDL2(2)) + (fL / (6.0f * fE * fI))
               * (2 * m_ELL(6, ElNo) - m_ELL(3, ElNo));
}

// stiffness terms
float ka = fA * fE / fL;
float ks = 12.0f * fE * fI / pow(fL, 3.0f);
float km = 4.0f * fE * fI / fL;
float kms = 6.0f * fE * fI / pow(fL, 2.0f);


// element forces
fVEF1(1) = ka * (fVDL1(1) - fVDL2(1));
fVEF1(2) = ks * (fVDL1(2) - fVDL2(2)) + kms * (fVDL1(3) + fVDL2(3));
fVEF1(3) = kms * (fVDL1(2) - fVDL2(2)) + km * (fVDL1(3) + 0.5f * fVDL2(3));
fVEF2(1) = -fVEF1(1);
fVEF2(2) = -fVEF1(2);
fVEF2(3) = kms * (fVDL1(2) - fVDL2(2)) + km * (0.50f * fVDL1(3) + fVDL2(3));

for (int i = 1; i <= DOFPN; i++)
    fVEF1(i) -= m_ELL(i, ElNo);
for (int i = 1; i <= DOFPN; i++)
    fVEF2(i) -= m_ELL(i + DOFPN, ElNo);

m_ElementResponseData(ElNo).SetForces(fVEF1, fVEF2);

//finding the element forces in global direction
fVEFG1(1) = fl * fVEF1(1) - fm * fVEF1(2);
fVEFG1(2) = fm * fVEF1(1) + fl * fVEF1(2);
fVEFG1(3) = fVEF1(3);
fVEFG2(1) = fl * fVEF2(1) - fm * fVEF2(2);
fVEFG2(2) = fm * fVEF2(1) + fl * fVEF2(2);
```

```
            fVEFG2(3) = fVEF2(3);

            // find the residual vector
            int DOF;
            for (int i = 1; i <= DOFPN; i++)
            {
                DOF = (nSN - 1) * DOFPN + i;
                m_fVR(DOF) += fVEFG1(i);
            }
            for (int i = 1; i <= DOFPN; i++)
            {
                DOF = (nEN - 1) * DOFPN + i;
                m_fVR(DOF) += fVEFG2(i);
            }

            pXSGrp->GetSectionModuli(fSyy, fSzz);
            pXSGrp->GetShearFactors(fSFy, fSFz);

            //Finding MI for the bending
            fI = fIzz; fS = fSzz; fSF = fSFy;
            if (fIyy > fIzz)
            {
                fI = fIyy;
                fS = fSyy;
                fSF = fSFz;
            }

            int nSections = 50;
            //Finding the peak stresses
            ComputePeakStresses(nSections, ElNo, fL, fA, fS, fSF);

        }

}
```

- Finding the maximum internal forces and internal stresses
  1. Find equally spaced points of investigation by the specified number of sections input.
  2. Loop through each section
  3. Set $MaxP = abs(f_1'), MaxV = abs(f_2')\ and\ MaxM = abs(f_3')$
  4. Find the maximum compressive ($f_{c,max}$), tensile stress ($f_{t,max}$), and shear stress (τ_{max}), at that section with the following equations 5.1 to 5.3 for forces f_1', f_2', f_3'
  5. Compute the axial (P), shear (V) and moment (M) due to loads from the left.
  6. Obtain forces at the section, $Ps = P + f_1', Vs = V + f_2', Ms = M + f_1' - xf_2'$
  7. Compute stresses correspond to these forces.
  8. If $abs(P) > MaxP$, $MaxP = abs(P)$, If $abs(V) > MaxV$, $MaxV = abs(V)$, If $abs(M) > MaxM$, $MaxM = abs(M)$,
  9. If $f_c > f_{c,max}, f_{c,max} = f_c$, If $f_t > f_{t,max}, f_{t,max} = f_t$, If $\tau > \tau_{max}, \tau_{max} = \tau$
  10.

$$f_c = \min(-P/A - |M|/Z, 0) \tag{5.1}$$

$$f_t = \max(-P/A + |M|/Z, 0) \tag{5.2}$$

$$\tau = V/SF \tag{5.3}$$

Here, initially, $s = P + f_1', Vs = V + f_2', Ms = M + f_1' - xf_2'$

```
void CFrame::ComputePeakStresses (const int nSections, const int ElNo,
```

```
                                const float fL, const float fA,
                                const float fS, const float fSF)
// ------------------------------------------------------------------------
// Function: computes the peak stress values in a member
// Input:    nSections - number of sections to analyze, ElNo - Element No, fL - length
//           fA - Area, fI - MOI, fS - Section Modulus, fSF - shear factor
// Output:   none
// ------------------------------------------------------------------------
{
    float fP, fV, fM;
    float fCS, fTS, fSS;
    float fMaxCS, fMaxTS, fMaxSS;
    float fMaxP, fMaxV, fMaxM;
    float fSpacing;

    CVector<float> fVFL(DOFPN), fVFR(DOFPN);

    m_ElementResponseData(ElNo).GetForces(fVFL, fVFR);

    fP = fVFL(1), fV = fVFL(2), fM = fVFL(3);

    MaxStresses(fA, fS, fSF, fP, fV, fM, fCS, fTS, fSS);

    fMaxCS = fCS, fMaxTS = fTS, fMaxSS = fSS;
    fMaxP = abs(fP); fMaxV = abs(fV); fMaxM = abs(fM);


    fSpacing = fL / float(nSections);

    float fX;
    for (int SNo = 2; SNo <= nSections + 1; SNo++)
    {
        fX = float(SNo - 1) * fSpacing;

        ComputeSectionalLoads(ElNo, fL, fX, fP, fV, fM);

        fP += fVFL(1), fV += fVFL(2); fM += fVFL(3);
        fM -= fVFL(2) * fX;

        if (abs(fP) > fMaxP)
            fMaxP = abs(fP);
        if (abs(fV) > fMaxV)
            fMaxV = abs(fV);
        if (abs(fM) > fMaxM)
            fMaxM = abs(fM);

        MaxStresses(fA, fS, fSF, fP, fV, fM, fCS, fTS, fSS);

        if (fCS > fMaxCS)
            fMaxCS = fCS;
        if (fTS > fMaxTS)
            fMaxTS = fTS;
        if (fSS > fMaxSS)
            fMaxSS = fSS;
    }


    m_ElementResponseData(ElNo).SetMaxStresses(fMaxCS, fMaxTS, fMaxSS);
    m_ElementResponseData(ElNo).SetMaxForces(fMaxP, fMaxV, fMaxM);
}


void CFrame::MaxStresses (const float fA, const float fS, const float fSF,
                          const float fP, const float fV, const float fM,
                          float& fCS, float& fTS, float& fSS)
//------------------------------------------------------------------------
//Function: computes maximum stress values at a section
//Input:    fA - Area, fS - Section Modulus, fP - Axial force
//          fV - Shear Force, fM - Moment
```

```
//Output:   none
//--------------------------------------------------------------------------
{
    fCS = abs(std::min((-fP / fA) - (fabs(fM) / fS), 0.0f));
    fTS = std::max((-fP / fA) + (fabs(fM) / fS), 0.0f);
    fSS = fabs(fV) / fSF;
    }
```

5.3 System of Equations solving using LDL^T decomposition

Theory: Solving through LDLT decomposition involves three stages (refer to Eq. 5.0),

1. Factorization of A into L and D
2. Forward substitution to find y
3. Backward substitution to find x

$$A = LDL^T \\ Ly = b \\ Ux = y \tag{5.4}$$

Algorithm: Eqn. (5.4) is implemented as given in [1] as follows in two phases,

Phase 1: LDL^T decomposition

Step 1: Check whether the matrix dimensions are consistent.
If the matrix is 1 by 1 simply find $x = b/A$

Step 2: Loop through, $i = 1,2, \dots n$

Step 3: Set $a_{ii} = a_{ii} - \sum_{k=1}^{i-1} a_{ik}^2 a_{kk}$. If $a_{ii} < \epsilon, stop$. The matrix is not positive definite.

Step 4: Loop through, $j = i + 1, \dots, n$. Set $a_{ji} = (a_{ji} - \sum_{k=1}^{i-1} l_{jk} a_{kk} l_{ik})/a_{ii}$

Step 5: End loop through j

Phase 2: Forward and backward substitution

Step 1: Loop through , $i = 1,2, \dots, n$ and solve for y_i

Step 2: Loop through , $i = n, n-1, \dots 1$ and solve for x_i

Code:

```
template <class T>
void CMatToolBox<T>::LDLTFactorization (CMatrix<T>& A,
                                        T TOL)
// ============================================================
// Function: carries out LDL(T) factorization of matrix A
//           A is replaced with L and D. A is a symmetric matrix.
//    Input: matrix A and tolerance value to detect singular A
//   Output: matrix A
// ============================================================
{
    // Getting the column dimension of the matrix A
    int n = A.GetColumns();

    // Checking whether the A matrix is square
    if (n != A.GetRows())
        ErrorHandler(Error::MATERR_LDLTFACTORIZATION);

    // check if the matrix is trivial 1 by 1
    if (n == 1)
        return; //if it is return the original matrix

    // Step 2
    for (int i = 1; i <= n; i++)
    {
        // Step 3
        for (int k = 1; k <= i - 1; k++)
        {
            A(i,i) -= A(i, k) * A(i, k) * A(k, k);
```

```
        }

        if (A(i, i) <= TOL)
            ErrorHandler(Error::MATERR_NOTPOSDEFMATRIX)
        // Step 3
        for (int j = i + 1; j <= n; j++)
        {
            for (int k = 1; k <= i - 1; k++)
            {
                A(j, i) -= A(j, k) * A(k, k) * A(i, k);
            }
            A(j, i) /= A(i, i);
        }
    }
    double Ops = static_cast<double> ((pow(n,3)-n)/6);

    m_dDOP += static_cast<double>(n*(n-1)/2);  // number of divisional operations
    m_dMOP += 2*Ops;  // number of multiplication operations
    m_dASOP += Ops; // number of subratction operations (half of multiplication operations)

}
template <class T>
void CMatToolBox<T>::LDLTSolve (const CMatrix<T>& A,
                                CVector<T>& x,
                                const CVector<T>& b)
// ============================================================================
// Function: carries out forward and backward substitution so as to
//           solve A x = b. A contains L and D terms.
//    Input: matrix A, vectors x and b
//   Output: vector x
// ============================================================================
{
    // Getting the column dimension of the matrix A
    int n = A.GetColumns();
    // Checking whether the A matrix is square
    if (n != A.GetRows() || n != x.GetSize() || n != b.GetSize())
        ErrorHandler(Error::MATERR_LDLTSOLVE);

    //  if the matrix is 1 by 1
    if (n == 1)
    {
        x(1) = b(1) / A(1, 1);
        m_dDOP += static_cast<double>(1);
        return;
    }
    // x initially contains b
    x = b;
    // forward substiution
    for (int i = 2; i <= n; i++)
    {
        for (int j = 1; j <= i - 1; j++)
        {
            x(i) -= A(i, j) * x(j);
        }
    }
    // backward substiution
    for (int i = n; i >= 1; i--)
    {
        x(i) /= A(i, i);

        for (int j = i + 1; j <= n; j++)
        {
            x(i) -= A(j, i) * x(j);
        }
    }
    m_dDOP += static_cast<double>(n);  // number of divisional operations
    m_dMOP += static_cast<double>(n * (n - 1));  // number of multiplication operations
    m_dASOP += static_cast<double>(n * (n - 1)); // number of subratction operations (same
as multiplication operations)
```

6.0 VALIDATION THROUGH TEST CASES

Several test cases are used to validate the obtained results by investigating the residual error, comparing it with simple standard cases analytical solution and comparing it with ASU FRAME PROGRAM [3] and applying equilibrium checks.

- **Test case set 1 (Simple Cantilever Beam, units N, m – different loads)**

  Cantilevers with three different load cases are analyzed and compared with standard results. The results are approximately similar. Table 6.1 shows the comparison and Fig. 6.1 shows the results obtained.
  - Length of the beam = 3
  - Rectangular cross section: 0.34641× 0.028867
  - Area = 1e-2
  - Moment of Inertia = 1e-4
  - Youngs' Modululs = 2e11

Table 6.1.a: Comparison of the tip vertical displacement

Loading case	Analytical	Program
1. Concentrated load the free end – 5000 (↓)	$PL^3/3EI = 0.00225$	0.00225
2. Uniformly distributed load 3000 (↓)	$wL^4/8EI = 0.00152$	0.00152
3. Concentrated Moment 1000 (↻)	$ML^2/2EI = 0.000125$	0.000125

Table 6.1.b: Comparison of the support reactions

Loading case	Analytical	Program
1	$F_x = 0, F_y = 5000 \uparrow, M_z = 15000 \circlearrowleft$	$F_x = 0, F_y = 5000 \uparrow, M_z = 15000 \circlearrowleft$
2	$F_x = 0, F_y = 9000 \uparrow, M_z = 13500 \circlearrowleft$	$F_x = 0, F_y = 9000 \uparrow, M_z = 13500 \circlearrowleft$
3	$F_x = 0, F_y = 0, M_z = 1000 \circlearrowleft$	$F_x = 0, F_y = 0, M_z = 1000 \circlearrowleft$

SUPPORT(NODAL) REACTIONS

Node	X-Reaction	Y-Reaction	RZ-Reaction
1	0	5000	15000

Fig 6.1.a : Support reactions of Concentrated load the free end – 5000 (↓)

SUPPORT(NODAL) REACTIONS

Node	X-Reaction	Y-Reaction	RZ-Reaction
1	0	9000	13500

Fig 6.1.b : Support reactions of Uniformly distributed load 3000 (↓)

SUPPORT(NODAL) REACTIONS

Node	X-Reaction	Y-Reaction	RZ-Reaction
1	0	0.00012207	1000

Fig 6.1.c : Support reactions of Concentrated Moment 1000 (↻)

- **Test case set 2 (Simple Cantilever Beam, units N, m – different sections)**

  Cantilevers with three different cross sections are analyzed and compared with standard results. The results are approximately similar. Table 6.2 shows the comparison and Fig. 6.2 shows the results obtained.

  - Length of the beam = 3
  - Loading Type: Concentrated load the free end – 5000 (↓)
  - Case 1: Rectangular cross section
    - Height = 0.34641 and width = 0.028867
  - Case 2: Circular cross section
    - Radius = 0.1
  - Case 3: I section
    - Web ht. = 0.4, Web tk. = 0.1, Flange wd. = 0.2 and Flange tk. = 0.1

Table 6.2: Comparison of the shear and normal stress

Case	Analytical (MPa)	Program
Rectangular	$\sigma = 25.95$ and $\tau = 0.75$	$\sigma = 25.95$ and $\tau = 0.75$
Circular	$\sigma = 19.10$ and $\tau = 0.212$	$\sigma = 19.10$ and $\tau = 0.212$
I Section	$\sigma = 1.467$ and $\tau = 0.114$	$\sigma = 1.467$ and $\tau = 0.114$

MAX. ELEMENT STRESSES

Element	Compression	Tension	Shear
1	2.59812e+07	2.59812e+07	750014

Fig 6.2.a : Maximum stresses – Rectangular case

MAX. ELEMENT STRESSES

Element	Compression	Tension	Shear
1	1.90986e+07	1.90986e+07	212207

Fig 6.2.b : Maximum stresses – Circular case

MAX. ELEMENT STRESSES

Element	Compression	Tension	Shear
1	1.46739e+06	1.46739e+06	114130

Fig 6.2.c : Maximum stresses – I Section case

- **Test case set 3 (Simply supported beam with different loading conditions)**

  Simply supported beams with three different loading conditions are analyzed and compared with standard results. The results are approximately similar. Table 6.3 shows the comparison and Fig. 6.3 shows the results obtained. Though the table shows only non-zero reactions, the zero reaction values can be seen in Fig. 6.3.

  - Length of the beam = 3
  - Rectangular cross section: 0.34641× 0.028867
  - Area = 1e-2
  - Moment of Inertia = 1e-4
  - Youngs' Modululs = 2e11
  - Loading cases
    1. Distributed load : wL = 3000 ↓ and wR = 5000 ↓
    2. Concentrated load : a = 1 (distance from left end) and W = 5000 ↓
    3. Concentrated moment : a = 1 (distance from left end) and M = 1000 ↻

Table 6.3: Comparison of the support reactions

Loading case	Analytical	Program
1	$F_{yL} = 5500 \uparrow, F_{yR} = 6500 \uparrow$	$F_{yL} = 5500 \uparrow, F_{yR} = 6500 \uparrow$
2	$F_{yL} = 3333 \uparrow, F_{yR} = 1667 \uparrow$	$F_{yL} = 3333 \uparrow, F_{yR} = 1667 \uparrow$
3	$F_{yL} = 333.3 \downarrow, F_{yR} = 333.3 \uparrow$	$F_{yL} = 333.3 \downarrow, F_{yR} = 333.3 \uparrow$

SUPPORT(NODAL) REACTIONS

Node	X-Reaction	Y-Reaction	RZ-Reaction
1	0	5500	
2		6500	

Fig 6.3.a : Support Reactions – Distributed load

SUPPORT(NODAL) REACTIONS

Node	X-Reaction	Y-Reaction	RZ-Reaction
1	0	3333.33	
2		1666.67	

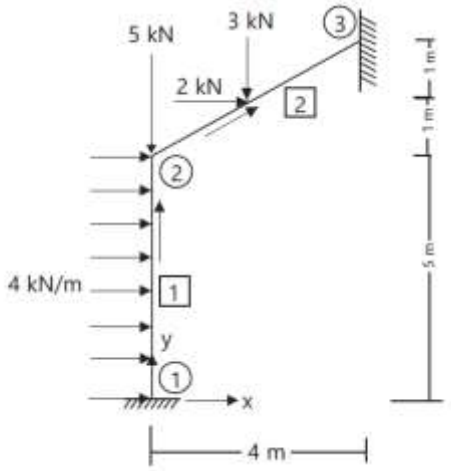
Fig 6.3.b : Support Reactions – Concentrated Load

MAX. ELEMENT STRESSES

Element	Compression	Tension	Shear
1	1.46739e+06	1.46739e+06	114130

Fig 6.3.c : Support Reactions – Concentrated Moment

- **Test case 4 (Frame)**



The frame shown above is analyzed and compared with the ASU FRAME program[2]. Following are the additional details of the frame.

- Circular cross section – 0.1m radius
- Youngs' Modululs = 2e11

➢ Solver Performance

```
                    SOLVER PERFORMANCE
-------------------------------------------------------------------
          Solver Type          : Cholesky Solver
          Absolute Error Norm  : 0.000977
          Relative Error Norm  : 0.000000
```

➢ Validation

Fig. 6.4.a: Comparison of the Nodal Displacements

My Program

```
                    NODAL DISPLACEMENTS
-------------------------------------------------------------------
Node      X-Displacement      Y-Displacement      Z-Rotation
-------------------------------------------------------------------
2         1.44784e-05         -1.01541e-05        0.000238653
```

ASU Frame

```
-----------------------------------------------------
Node          X Disp          Y Disp          Z Rot
                ( m )           ( m )          (rad)
-----------------------------------------------------
   1                0               0              0
   2      1.44781e-05    -1.01539e-05    0.000238666
   3                0               0              0
```

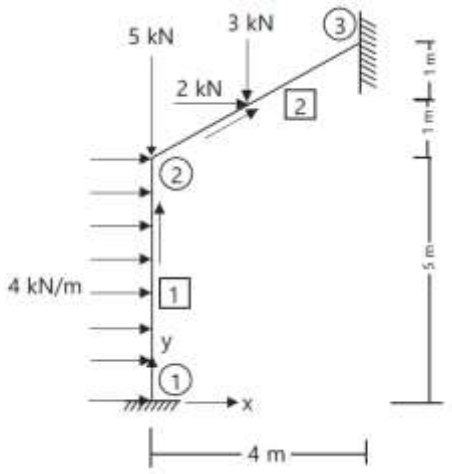
Fig. 6.4.b: Comparison of the Support Reactions

My Program	SUPPORT(NODAL) REACTIONS Node X-Reaction Y-Reaction RZ-Reaction 1 -10921.5 12760 9887.42 3 -11078.5 -4760.03 -396.755
ASU Frame	Node X Reaction (N) Y Reaction (N) Z Reaction (N - m) 1 -10921.6 12759.7 9887.5 3 -11078.1 -4760.28 -396.322

Fig. 6.4.c: Comparison of the maximum member stresses

My Program	MAX. ELEMENT STRESSES Element Compression Tension Shear 1 1.29952e+07 1.21829e+07 463524 2 7.09131e+06 6.35344e+06 122264
ASU Frame	Elem Compressive Stress (N / m ^2) Tensile Stress (N / m ^2) Shear Stress (N / m ^2) Sigcr (N / m ^2) 1 1.29953e+07 1.2183e+07 463526 1.97392e+08 2 7.0911e+06 6.35325e+06 122254 2.4674e+08

- **Test case 5 (Frame)**



The frame shown above is analyzed and compared with the ASU FRAME program[2]. Following are the additional details of the frame.

- Circular cross section – 0.1m radius
- Youngs' Modululs = 2e11
- With support settlement 1e-3 at node 1

- Solver Performance

```
                    SOLVER PERFORMANCE
-------------------------------------------------------------------
                Solver Type          : Cholesky Solver
                Absolute Error Norm  : 0.043391
                Relative Error Norm  : 0.000000
```

- Validation

Fig. 6.5.a: Comparison of the Nodal Displacements

My Program

```
                        NODAL DISPLACEMENTS
-------------------------------------------------------------------
Node      X-Displacement      Y-Displacement      Z-Rotation
-------------------------------------------------------------------
2         0.000511665         -0.00100811         0.000365634
```

ASU Frame

```
-----------------------------------------------------
 Node          X Disp          Y Disp          Z Rot
                ( m )           ( m )          (rad)
-----------------------------------------------------
    1               0          -0.001              0
    2     0.000511665     -0.00100811     0.000365646
    3               0               0              0
```

Fig. 6.5.b: Comparison of the Support Reactions

My Program

```
                        SUPPORT(NODAL) REACTIONS
-------------------------------------------------------------------
Node      X-Reaction          Y-Reaction          RZ-Reaction
-------------------------------------------------------------------
1         -12150              10187.8             12559.6
3         -9850.02            -2187.8             -4758.8
```

ASU Frame

```
-------------------------------------------------
 Node     X Reaction     Y Reaction     Z Reaction
                ( N )          ( N )      ( N - m )
-------------------------------------------------
    1        -12150        10187.5        12559.7
    3       -9849.71       -2188.05       -4758.36
```

Fig. 6.5.c: Comparison of the maximum member stresses

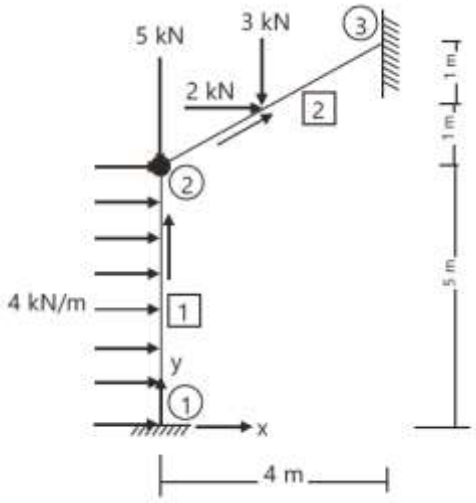
My Program

```
                        MAX. ELEMENT STRESSES
-------------------------------------------------------------------
Element        Compression          Tension             Shear
-------------------------------------------------------------------
1              1.63157e+07          1.56671e+07         515661
2              6.37067e+06          5.74751e+06         103906
```

ASU Frame

```
-------------------------------------------------------------------------
Elem   Compressive      Tensile       Shear          Sigcr
         Stress         Stress       Stress
       ( N / m ^2)   ( N / m ^2)   ( N / m ^2)    ( N / m ^2)
-------------------------------------------------------------------------
   1   1.63158e+07   1.56672e+07        515663    1.97392e+08
   2   6.37011e+06   5.74696e+06        103891    2.4674e+08
```

- **Test case 6 (Frame with hinge)**



The frame shown above is analyzed and compared with the ASU FRAME program[2]. Following are the additional details of the frame.

- I – Section : Web ht. = 0.5, Web tk. = 0.05, Flange wd. = 0.25 and Flange tk. = 0.04
- Youngs' Modululs = 2e11

➢ Solver Performance

```
                    SOLVER PERFORMANCE
-------------------------------------------------------------------
          Solver Type          : Cholesky Solver
          Absolute Error Norm  : 0.001092
          Relative Error Norm  : 0.000000
```

➢ Validation

Fig. 6.6.a: Comparison of the Nodal Displacements

My Program

```
                        NODAL DISPLACEMENTS
-------------------------------------------------------------------
Node      X-Displacement      Y-Displacement      Z-Rotation
-------------------------------------------------------------------
2         7.74972e-06         -5.46614e-06        0
```

ASU Frame

```
-------------------------------------------------------
 Node          X Disp          Y Disp          Z Rot
                ( m )           ( m )           (rad)
-------------------------------------------------------
    1               0               0               0
    2     7.74961e-06    -5.46602e-06               -
    3               0               0               0
```

Fig. 6.6.b: Comparison of the Support Reactions

My Program

```
                        SUPPORT(NODAL) REACTIONS
-------------------------------------------------------------------
Node      X-Reaction          Y-Reaction          RZ-Reaction
-------------------------------------------------------------------
1         -12573.7            9839.05             12868.5
3         -9426.28            -1839.04            -3496.62
```

ASU Frame

```
-----------------------------------------------
 Node    X Reaction    Y Reaction    Z Reaction
                ( N )         ( N )     ( N - m )
-----------------------------------------------
    1      -12573.7       9838.84       12868.5
    3      -9426.01      -1839.38      -3496.1
```

Fig. 6.6.c: Comparison of the maximum member stresses

My Program

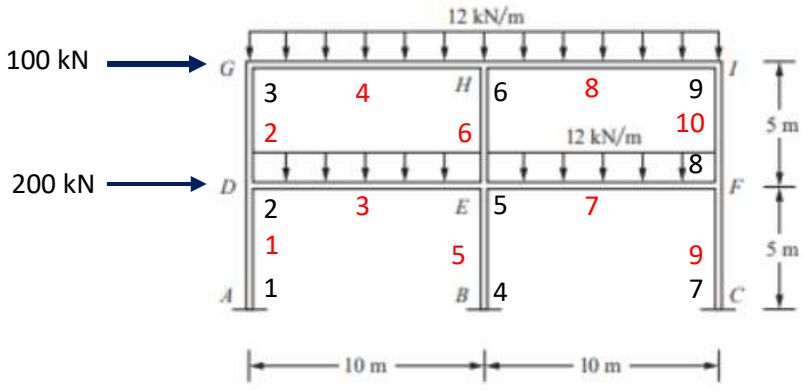
MAX. ELEMENT STRESSES

Element	Compression	Tension	Shear
1	2.10201e+06	1.66471e+06	540958
2	717379	306109	110598

ASU Frame

Elem	Compressive Stress (N / m ^2)	Tensile Stress (N / m ^2)	Shear Stress (N / m ^2)	Sigcr (N / m ^2)
1	2.102e+06	1.66472e+06	540958	3.47673e+09
2	717301	306035	110580	4.34592e+09

- **Test case set 7 (Two story and Two bay portal frame)**



The portal frame shown above is analyzed and compared with the ASU FRAME program[2]. Following are the additional details of the frame.

- Columns
  - I – Section : Web ht. = 0.5, Web tk. = 0.05, Flange wd. = 0.25, Flange tk. = 0.04
  - Youngs' Modululs = 2e11
- Beams
  - Rectangular : 0.5 × 0.25
  - Youngs' Modululs = 1e11

➢ Solver Performance

SOLVER PERFORMANCE

```
Solver Type          : Cholesky Solver
Absolute Error Norm  : 1.149394
Relative Error Norm  : 0.000003
```

- Validation

Fig. 6.7.a: Comparison of the Nodal Displacements

My Program

NODAL DISPLACEMENTS

Node	X-Displacement	Y-Displacement	Z-Rotation
2	0.00534545	-3.66118e-05	-0.00122497
3	0.0103743	-5.8413e-05	-0.00085529
5	0.00526016	-0.000137984	-0.000929115
6	0.0102891	-0.000207796	-0.00050309
8	0.0052202	-9.20706e-05	-0.00102206
9	0.0102479	-0.000133791	-0.000448291
Min	0.0052202	-0.000207796	-0.00122497
Node	8	6	2
Max	0.0103743	-3.66118e-05	-0.000448291
Node	3	2	9

ASU Frame

Node	X Disp (m)	Y Disp (m)	Z Rot (rad)
1	0	0	0
2	0.00534545	-3.66118e-05	-0.00122497
3	0.0103743	-5.8413e-05	-0.00085529
4	0	0	0
5	0.00526016	-0.000137984	-0.000929115
6	0.0102891	-0.000207796	-0.00050309
7	0	0	0
8	0.00522019	-9.20706e-05	-0.00102206
9	0.0102479	-0.000133791	-0.000448291
Min	0	-0.000207796	-0.00122497
Node	1	6	2
Max	0.0103743	0	0
Node	3	1	1

Fig. 6.7.b: Comparison of the Support Reactions

My Program

SUPPORT(NODAL) REACTIONS

Node	X-Reaction	Y-Reaction	RZ-Reaction
1	-86856.7	65901.3	314233
4	-111752	248372	353021
7	-101391	165727	334487

ASU Frame

Node	X Reaction (N)	Y Reaction (N)	Z Reaction (N - m)
1	-86856.7	65901.3	314233
4	-111752	248372	353021
7	-101391	165727	334487

Fig. 6.7.c: Comparison of the maximum member stresses

My Program

MAX. ELEMENT STRESSES

Element	Compression	Tension	Shear
1	4.74537e+07	4.45248e+07	3.73683e+06
2	7.55184e+06	5.80774e+06	281199
3	2.57168e+07	2.40111e+07	1.12009e+06
4	1.95356e+07	1.7831e+07	969094
5	5.71854e+07	4.61466e+07	4.8079e+06
6	2.78962e+07	2.23112e+07	2.37075e+06
7	2.49347e+07	2.41353e+07	1.08757e+06
8	1.71208e+07	1.62979e+07	901159
9	5.26362e+07	4.52706e+07	4.36216e+06
10	2.71426e+07	2.38049e+07	2.21275e+06
Min	7.55184e+06	5.80774e+06	281199
Element	2	2	2
Max	5.71854e+07	4.61466e+07	4.8079e+06
Element	5	5	5

ASU Frame

Elem	Compressive Stress (N / m ^2)	Tensile Stress (N / m ^2)	Shear Stress (N / m ^2)	Sigcr (N / m ^2)
1	4.74537e+07	4.45248e+07	3.73683e+06	3.47673e+09
2	7.55183e+06	5.80773e+06	281200	3.47673e+09
3	2.57168e+07	2.40111e+07	1.12009e+06	2.05617e+08
4	1.95356e+07	1.7831e+07	969094	2.05617e+08
5	5.71854e+07	4.61466e+07	4.8079e+06	3.47673e+09
6	2.78962e+07	2.23112e+07	2.37075e+06	3.47673e+09
7	2.49347e+07	2.41353e+07	1.08757e+06	2.05617e+08
8	1.71208e+07	1.62979e+07	901159	2.05617e+08
9	5.26362e+07	4.52706e+07	4.36216e+06	3.47673e+09
10	2.71426e+07	2.38049e+07	2.21274e+06	3.47673e+09
Min	7.55183e+06	5.80773e+06	281200	2.05617e+08
Elem	2	2	2	3
Max	5.71854e+07	4.61466e+07	4.8079e+06	3.47673e+09
Elem	5	5	5	1

7.0 Concluding Remarks

A C++ program is developed to analyze space frames. The system K matrix is stored as effective DOF size. The program can also analyze frames with internal hinges. The global system of equations solved with LDL^T decomposition. The results are validated by comparing with ASU Frame program, equilibrium checks and residual error check. The relative residual error norms are in the range of 10^{-5}.

References

[1] Rajan S D, Object-Oriented Numerical Methods via C++, Second Edition: July 2021.
[2] Rajan S D, Intermediate Structural Analysis and Design, Second Edition: December 2018.
[3] Rajan S D, GS – USA Program Ver. 10.19.