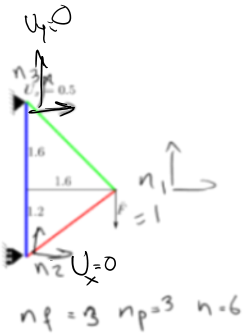
2023/11/20

Monday, November 20, 2023 11:07 AM

From last time

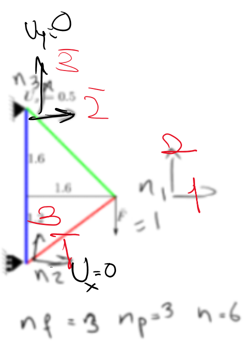


node	p	pos	v	f
1	0		0?	0
	0		0?	0
2	1		0.0	0 ?
	0		0?	-1
3	1		0.5	0 p
	1		0.0	0 ?

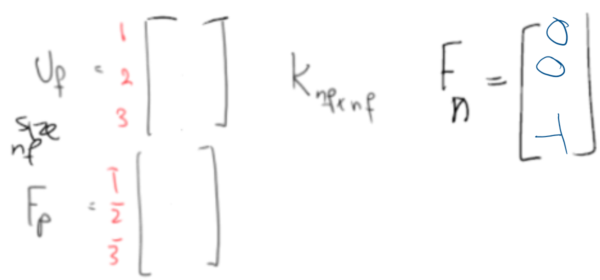
e	LEM	dof Map	dofs	fe

Steps 6 & 7

Step 6: dof positions; Step 7: Set $\mathbf{F}(\mathbf{F}_f)$



node	p	pos	v	f
1	0	1	0?	0
	0	2	0?	0
2	1	1	0.0	0 ?
	0	3	0?	-1
3	1	2	0.5	0 p
	1	3	0.0	0 ?



e	LEM	dof Map	dofs	fe

free counter ~~0~~ ~~1~~ ~~2~~ 3

prescribed counter ~~0~~ ~~1~~ ~~2~~ 3

in C++, Python dof positions start from 0

in Fortran, Matlab ← 1

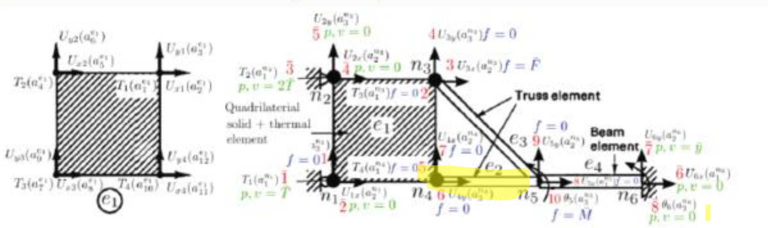
posf = 0, posp = 0
for n = 1:nNodes
  for dofi = 1: node(n).nndof num dof for node (n)
    if node(n).ndof(dofi).p == true prescribed dof
      posp = posp - 1;
      node(n).ndof(dofi).pos = posp;
  else free dof

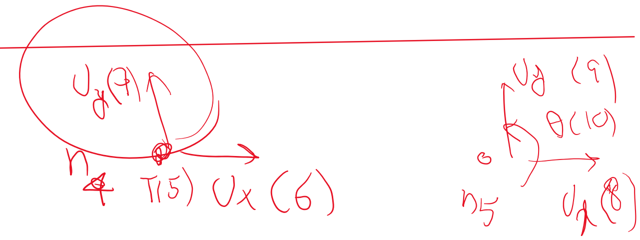
```
posf = 0, posp = 0
for n = 1:nNodes
   for dofi = 1: node(n).nndof num dof for node (n)
      if node(n).ndof(dofi).p == true prescribed dof
         posp = posp - 1;
         node(n).ndof(dofi).pos = posp;
      else free dof
         posf = posf + 1;
         node(n).ndof(dofi).pos = posf;
         F(posf) = node(n).ndof(dofi).f
      end
   end
end
```

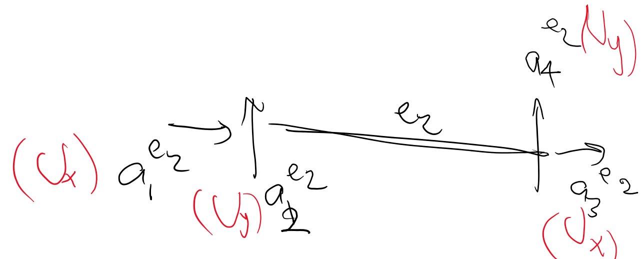
(in Fortran, Matlab → 1

Step 8:
Setting dofMap of elements

Step 8: Element dof maps M_t^e







$M(e_2) = [6, 7, 8, 9]$

- As mentioned, M_t^e is a vector of size n_{dof}^e that maps elements dofs to global positions.
- For element 1, dofs are ordered as (loop over nodes, then loop over dofs for the node):

$$a_t^e = [a_1^{e1} \quad a_2^{e1} \quad \cdots \quad a_{12}^{e1}]$$
$$= [\ T_1 \quad U_{x1} \quad U_{y1} \mid T_2 \quad U_{x2} \quad U_{y2} \mid T_3 \quad U_{x3} \quad U_{y3} \mid T_4 \quad U_{x4} \quad U_{y4}\ ]$$

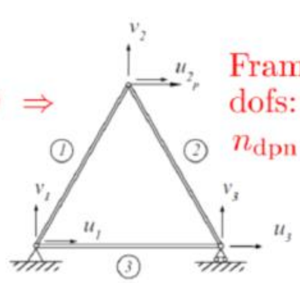
- We need to map these dofs to global dofs and have their position in M_t^e vector. For example, 1st dof of node 1 ($a_1^{e1} = T_1$) is mapped to first dof of n_3 which has position 2.
- 2nd dof of node 3 ($a_8^{e1} = U_{y2}$) is mapped to 2nd dof of n_1 which has position $\bar{2}(-2)$.
- The map for element e_1 is:

$$\mathrm{M}^{e1}{}_t = [\ 2 \quad 3 \quad 4 \quad \bar{3} \quad \bar{4} \quad \bar{5} \quad \bar{1} \quad \bar{2} \quad 1 \quad 5 \quad 6 \quad 7\ ]$$

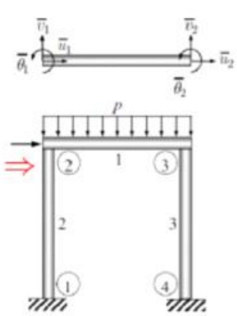
Not needed for the term project

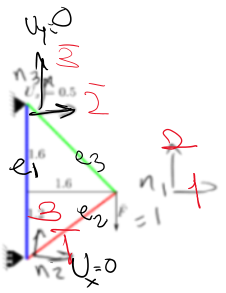
Step 8: Element dof maps M_t^e: Simplified limited case

Truss
dofs: $u, v \Rightarrow$
$n_{\mathrm{dpn}} = 2$



Frame
dofs: $u, v, \theta \Rightarrow$
$n_{\mathrm{dpn}} = 3$





$n_f = 3 \quad n_p = 3 \quad n = 6$

node	p	pos	v	f
1	0	1	0?	0
	0	2	0?	0
2	1	-1	0.0	0 ?
	0	3	0?	-1
3	1	-2	0.5	0 ?
	1	-3	0.0	0 ?

$U_f = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} \\ \\ \\ \end{bmatrix}$ size n_f $\quad K_{n_f \times n_f} \quad F_n = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$

$F_p = \begin{matrix} -1 \\ -2 \\ -3 \end{matrix} \begin{bmatrix} \\ \\ \\ \end{bmatrix}$

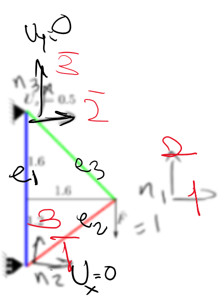
e	LEM	dofMap	dofs	fe
1	[2,3]	[-1 3 -2 -3]		
2	[2,1]	[-1 3 1 2]		
3	[3,1]	[-2 -3 1 2]		

Be careful with indices stored in dofMap, in C++ I subtract 1 from prescribed dofs. For example, for e2 without the subtraction, I would have had

0 2 0 1, this becomes -1, 2 0 1
Similarly element 1 will have dofMap -1, 2, -2, -3

Step 9: Set element dofs $\mathbf{a}^e$

needed for $f_D^e = K^e a^e$



$n_f = 3 \quad n_p = 3 \quad n = 6$

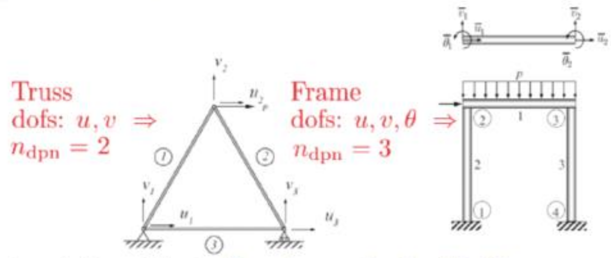
node	p	pos	v	f
1	0	1	0?	0
	0	2	0?	0
2	1	-1	0.0	0 ?
	0	3	0?	-1
3	1	-2	0.5	0 ?
	1	-3	0.0	0 ?

$U_f = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} \\ \\ \\ \end{bmatrix}$ size n_f $\quad K_{n_f \times n_f} \quad F_n = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$

$F_p = \begin{matrix} -1 \\ -2 \\ -3 \end{matrix} \begin{bmatrix} \\ \\ \\ \end{bmatrix}$

e	LEM	dofMap	dofs	fe
1	[2,3]	[-1 3 -2 -3]	[0.0 0 .5 0]	
2	[2,1]	[-1 3 1 2]	[0.0 0 0 0]	
3	[3,1]	[-2 -3 1 2]	[.5 0 0 0]	

Step 9: Set element dofs $\mathbf{a}^e$: Simplified limited case



- Similar to steps 1, 2, and 8, step 9 can be greatly simplified if we assume all nodes share exactly the same set of dofs.
- Noting n_{dpn} (ndofpn) = Number of dof. per node, simplified merged steps 8 & 9 are:

```
dofs = zeros(nedof) element dofs (edof) resized to number of element dofs and zeroed
ecdof = 1 dof counter for element
for en = 1: neNodes number of element nodes
   gn = LEM(en) global node number for element node en
   for endof = 1: ndofpn This number is fixed now, e.g., 2 for 2D trusses
      if (node(gn).dof(endof).p == true)    gndof = endof, we bypass some steps here
         dofs(ecdof) = node(gn).dof(endof).value; e dof val = corresponding global val
      end
      dofMap(ecdof) = node(gn).dof(endof).pos
      ecdof = ecdof + 1    increment counter
   end
end
```


Step 10: Compute element stiffness/force

Example: Truss element

```
class PhyElementTruss : public PhyElement
{
public:
    virtual void setInternalMaterialProperties(PhyMaterial*
    pMat);
    virtual void Calculate_ElementStiffness_Force();


void PhyElementTruss::Calculate_ElementStiffness_Force()
{
    // compute stiffness matrix:
    ke.resize(4, 4);
    double factor = A * E / L;
    for (int I = 0; I < 2; ++I)
        for (int J = 0; J < 2; ++J)
        {
            double f2 = factor;
            if ((I + J) % 2 != 0)
                f2 = -factor;
            ke(2 * I, 2 * J) = c * c * f2;
            ke(2 * I + 1, 2 * J) = ke(2 * I, 2 * J + 1) = c * s * f2;
            ke(2 * I + 1, 2 * J + 1) = s * s * f2;
        }
    cout << "ke\n" << ke << endl;
}
```

$\frac{AE}{L}$

$$\frac{AE}{L}\begin{bmatrix} c^2 & cs & -1\times & \\ cs & s^2 & & \\ -1\times & & -1 & \end{bmatrix}$$

We need to have a few things for this element

ⓐ A, E ⟶ Material properties

(b) L, c, s

(a) A, E :

1 2 10 10000
2 3 0.01 100 0.3

- In the sample data input, material refers to truss element material parameters $A = 10, E = 10000$ while material 2 is for 2D solid with t(thickness) $= 0.01, E = 100, \nu$(Poisson ratio) $= 0.3$.
- At element level a matID refers to global {mats}; *e.g.*,

$$\text{For } e_3, \text{matID} = 1, \quad \Rightarrow A = \{\text{mats}\}(1)(1) = 10, E = \{\text{mats}\}(1)(2) = 10000 \qquad (450)$$

eType $= 1$: bar element	eType $= 2$: beam element	(451)
eType $= 3$: truss element	eType $= 4$: frame element	(452)
eType $= 5$: 2D thermal element	eType $= 6$: 2D solid element	(453)

eType $= 1$	Bar	mat$\{en\} = [E\ A]$	(454a)
eType $= 2$	Beam	mat$\{en\} = [E\ I]$	(454b)
eType $= 3$	Truss	mat$\{en\} = [E\ A]$	(454c)
eType $= 4$	Frame	mat$\{en\} = [E\ A\ I]$	(454d)
eType $= 5$	2D thermal	mat$\{en\} = [\kappa\ t]$	(454e)
eType $= 6$	2D solid	mat$\{en\} = [E\ \nu\ t]$	(454f)

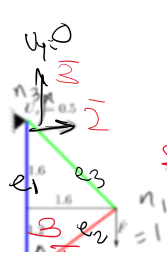
material #1

Materials
nMat 1
id numPara Paras
1 2 1.0 1.0

w/ E, A
2 properties

id elementType matID neNodes eNodes
1 3 1 2 2 3
2 3 1 2 2 1
3 3 1 2 3 1

(b) L, C, S How do we calculate these?

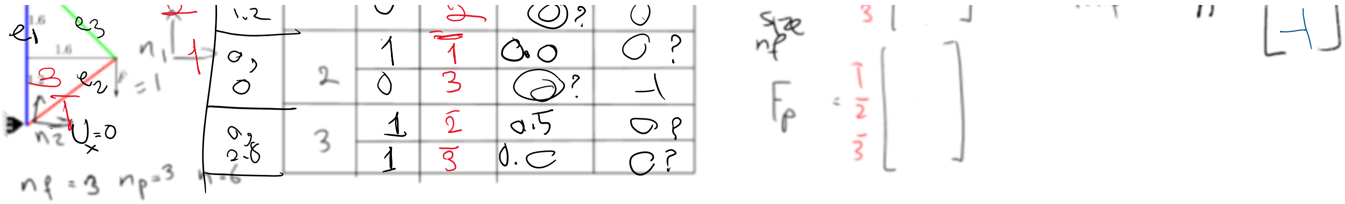


xy	node	p	pos	v	f
1.6, 1.2	1	0	1	0 ?	0
		0	2	0 ?	0
0, 0	2	1	1	0.0	0 ?
		0	3	0 ?	-1

U_p : 1, 2, 3 $\begin{bmatrix} \\ \\ \end{bmatrix}$ size n_f

$K_{n_f \times n_f}$

$F_n = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$

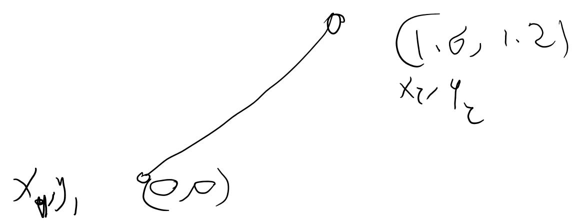


e	LEM	dof Map	dofs	fe
1	[2,3]	[$\bar{1}$ 3 $\bar{2}$ $\bar{3}$]	[0.0 0 .5 0]	
2	[2,1]	[$\bar{1}$ 3 1 2]	[0.0 0 0 0]	
3	[3,1]	[$\bar{2}$ $\bar{3}$ 1 2]	[.5 0 0 0]	

LEM

e 2 $node(1)_{e_2} = 2$ crd 0,0

$node(2)_{e_2} = 1$ crd 1.6, 1.2



$$L = \sqrt{(x_1-x_0)^2 + (y_1-y_0)^2}$$

$$c = \frac{x_1-x_0}{L} \qquad s = \frac{y_1-y_0}{L}$$

First, we add the extra data for each particular element type:

```
class PhyElementTruss : public PhyElement
{
public:
    virtual void setInternalMaterialProperties(PhyMaterial*
    pMat);
    virtual void Calculate_ElementStiffness_Force();

    virtual void SpecificOutput(ostream& out) const;

    double L;
    double A;
    double E;
    double c;
    double s;


void PhyElementTruss::setInternalMaterialProperties(PhyMaterial* pMat)
{
    A = pMat->paras(mpb_A);
    E = pMat->paras(mpb_E);
}

void PhyElementTruss::setGeometry()
{
    VECTOR *crd0, *crd1;
    crd1 = &eNodePtrs[1]->coordinate;
    crd0 = &eNodePtrs[0]->coordinate;

    int sz = crd1->size();
```

```
    if (sz != 2)
        THROW("implementation only for 2D truss");
    double delX, delY;
    delX = (*crd1)(0) - (*crd0)(0);
    delY = (*crd1)(1) - (*crd0)(1);
    L = sqrt(delX * delX + delY * delY);
    c = delX / L;
    s = delY / L;
}
```

Non-object oriented calculation of stiffness matrix

if (type==1) % bar

$$k = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

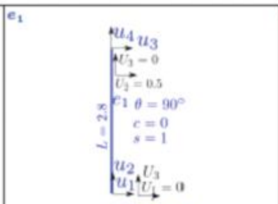
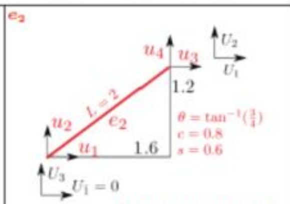
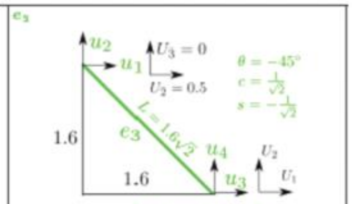
elseif (type==2) % beam

$$k = \begin{bmatrix} & \\ & \end{bmatrix}_{4\times4}$$

or cases

Step 11: Assembly from local to global system

Truss example: Assembly of global system

e	e_1	e_2	e_3
			
$\mathbf{k}^e$	$\mathbf{k}^{e_1}=\frac{(1)(1)}{2.8}\begin{bmatrix}0&0&0&0\\0&1&0&-1\\0&0&0&0\\0&-1&0&1\end{bmatrix}$ $= \begin{array}{c\|cccc} & \bar{1} & 3 & 2 & \bar{3}\\ \hline \bar{1} & 0&0&0&0\\ 3 & 0&0.3571&0&-0.3571\\ 2 & 0&0&0&0\\ \bar{3} & 0&-0.3571&0&0.3571\end{array}$	$\mathbf{k}^{e_2}=\frac{(1)(1)}{2}\begin{bmatrix}0.64&0.48&-0.64&-0.48\\0.48&0.36&-0.48&-0.36\\-0.64&-0.48&0.64&0.48\\-0.48&-0.36&0.48&0.36\end{bmatrix}$ p f $= \begin{array}{c\|cccc} & \bar{1} & 3 & 1 & 2\\ \hline \bar{1} & 0.32&0.24&-0.32&-0.24\\ 3 & 0.24&0.18&-0.24&-0.18\\ 1 & -0.32&-0.24&0.32&0.24\\ 2 & -0.24&-0.18&0.24&0.18\end{array}$ p f	$\mathbf{k}^{e_3}=\frac{(1)(1)}{1.6\sqrt{2}}\begin{bmatrix}0.5&-0.5&-0.5&0.5\\-0.5&0.5&0.5&-0.5\\-0.5&0.5&0.5&-0.5\\0.5&-0.5&-0.5&0.5\end{bmatrix}$ $= \begin{array}{c\|cccc} & \bar{2} & \bar{3} & 1 & 2\\ \hline \bar{2} & 0.221&-0.221&-0.221&0.221\\ \bar{3} & -0.221&0.221&0.221&-0.221\\ 1 & -0.221&0.221&0.221&-0.221\\ 2 & 0.221&-0.221&-0.221&0.221\end{array}$ only p matters
$\mathbf{f}_D^e$	$\mathbf{k}^{e_1}\mathbf{a}_1^e = \begin{bmatrix}0&0&0&0\\0&0.3571&0&-0.3571\\0&0&0&0\\0&-0.3571&0&0.3571\end{bmatrix}\begin{bmatrix}0\\0\\0.5\\0\end{bmatrix}=\begin{array}{c}\bar{1}\\3\\2\\\bar{3}\end{array}\begin{bmatrix}0\\0\\0\\0\end{bmatrix}$ f_D^e	$\mathbf{k}^{e_2}\mathbf{a}_2^e = \begin{bmatrix}0.32&0.24&-0.32&-0.24\\0.24&0.18&-0.24&-0.18\\-0.32&-0.24&0.32&0.24\\-0.24&-0.18&0.24&0.18\end{bmatrix}\begin{bmatrix}0\\0\\0\\0\end{bmatrix}=\begin{array}{c}\bar{1}\\3\\1\\2\end{array}\begin{bmatrix}0\\0\\0\\0\end{bmatrix}$ f_D	p p f f $\begin{bmatrix}\text{[illegible]}&\text{[illegible]}&-0.221&0.221\\-0.221&0.221&0.221&-0.221\\-0.221&0.221&0.221&-0.221\\0.221&-0.221&-0.221&0.221\end{bmatrix}\begin{bmatrix}0.5\\0\\0\\0\end{bmatrix}=\begin{array}{c}\bar{2}\\\bar{3}\\1\\2\end{array}\begin{bmatrix}0.1105\\-0.1105\\-0.1105\\0.1105\end{bmatrix}$ f_D f_p
$\mathbf{f}_e^e$	$\mathbf{f}_e^{e_1}=\mathbf{f}_r^{e_1}+\mathbf{f}_N^{e_1}-\mathbf{f}_D^{e_1}=\begin{array}{c}\bar{1}\\3\\2\\\bar{3}\end{array}\begin{bmatrix}0\\0\\0\\0\end{bmatrix}$	$\mathbf{f}_e^{e_2}=\mathbf{f}_r^{e_2}+\mathbf{f}_N^{e_2}-\mathbf{f}_D^{e_2}=\begin{array}{c}\bar{1}\\3\\1\\2\end{array}\begin{bmatrix}0\\0\\0\\0\end{bmatrix}$	$\mathbf{f}_e^{e_3}=\mathbf{f}_r^{e_3}+\mathbf{f}_N^{e_3}-\mathbf{f}_D^{e_3}=\begin{array}{c}p\ \bar{2}\\p\ \bar{3}\\f\ 1\\f\ 2\end{array}\begin{bmatrix}-0.1105\\0.1105\\\boxed{0.1105}\\\boxed{-0.1105}\end{bmatrix}$

$$\mathbf{K}=\begin{array}{c} \\ 1\\2\\3\end{array}\begin{array}{ccc}1&2&3\\ \hline 0.32+0.221 & 0.24-0.221 & -0.24\\ 0.24-0.221 & 0.18+0.221 & -0.18\\ -0.24 & -0.18 & 0.3571+0.18\end{array}=\begin{bmatrix}0.5410&0.019&-0.24\\0.019&0.401&-0.18\\-0.24&-0.18&0.5371\end{bmatrix}$$

$$\mathbf{F}=\mathbf{F}_N+\mathbf{F}_e=\begin{bmatrix}0\\-1\\0\end{bmatrix}+\begin{array}{c}1\\2\\3\end{array}\begin{bmatrix}0.1105\\-0.1105\\0\end{bmatrix}=\begin{bmatrix}0.1105\\-1.1105\\0\end{bmatrix}\Rightarrow \boxed{\mathbf{U}=\mathbf{K}^{-1}\mathbf{F}=\begin{bmatrix}U_1\\U_2\\U_3\end{bmatrix}=\begin{bmatrix}-0.2123\\-3.2980\\-1.200\end{bmatrix}}$$

- Numbers encircled in the computation of essential BC force are displacements corresponding to free dofs. As mentioned before, in reality we do not consider them in computation of this force, but in hand calculation we just put zero for those values.

row column

only f f components of k^e go to K (global)

which one contribute to f_D^e

ff

fp

pf

pp

ff — K^e

fp — F_D^e

```
Function
(rows)     for i = 1 : nedof        # edof
               I = dofMap(i)
               if I < 0          % row is prescribed
                   continue;
               end
                                    free rows
(columns)      for j = 1 : nedof
                   J = dofMap(j)
                   if J > 0          free columns
                       K(I,J) = K(I,J) + ke(i,j)   ff
                   else             ( J < 0 )
                       feD(i) = feD(i) + ke(i,j) * ae(j);   fp case
                   end
```

dofMap = [$\bar{2}$ $\bar{3}$ 1 2] $\underline{e_3}$ nedof = 4

end
fee(i) = fee(i) - feD(i)
F(I) = F(I) + fee(i)
end

Step 11: Assembly from local to global system

- $\mathbf{K}$ and $\mathbf{F}$ (global stiffness and force) are already sized and set to zero.
- Element level (local) stiffness and force is calculated (Step 10).
- Element local to global dof M_t^e is already set (Step 8).
- Using dof map, we assemble local values to global values.
- Clearly, only free dofs are added to stiffness matrix and force vector.
- Element dof values (dofs: $\mathbf{a}^e$ is also set (Step 9).
- $\mathbf{f}_D^e = \mathbf{k}^e\mathbf{a}^e$ may not need to be formed and can be directly added to $\mathbf{f}_e^e$.

for e = 1:ne loop over elements
  fee = feo element total force = element all forces except essential force
  for i = 1:nedof loop over rows of ke; nedof = element # dof
    I = dofMap(i) local to global dof map M_t^e
    if (I > 0) I corresponds to a free dof, we skip prescribed dofs
      for j = 1:nedof loop over columns of ke
        J = dofMap(j) global dof corresponding to j
        if (J > 0) now both I and J are free and can add ke(i,j) to global K
          K(I, J) = K(I, J) + ke(i, j)
        else J < 0, prescribed dof j; add contributions of $\mathrm{f}_D^e = \mathrm{k}^e\mathrm{a}^e$ to f_e^e
          fee(i) = fee(i) - ke(i, j) * edofs(j) edofs: element dofs = a^e
        end
      end
      F(I) = F(I) + fee(i)element's total force fee component i'th is computed→added to F(I)
    end
  end
end

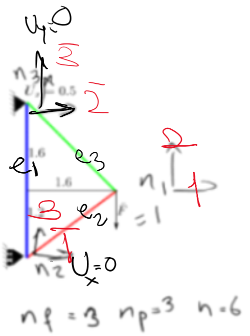
Step 11: Assembly from local to global system

- **Rows**: In loop over edof (element dofs) we skip prescribed dofs i.
- **Columns**: free columns j: Stiffness is assembled $K(I,J) = K(I,J) + ke(i,j)$.
- **Columns**: prescribed columns j: $\mathbf{f}_D^e$ is added to total element force $\mathbf{f}_e^e$: fee(i) = fee(i) - ke(i, j) * edofs(j).
- **Force**: Once fee is updated with fde (essential BD force), it is assembled to global F: F(I) = F(I) + fee(i).
- **Both K and k are symmetric**:
  1. K can be stored into a symmetric matrix → almost half the storage.
  2. ke symmetric: can be stored in symmetric matrix → almost half the storage.
  3. ke symmetric: half the loop computation: instead of looping (i: 1 → nedof; j: 1 → nedof) we can loop **(i: 1 → nedof; j: i → nedof)**. How?

Step 12: Solve global (free) dof a from $\mathbf{Ka} = \mathbf{F}$

- Two major computational costs during FEM solve are:
  1. Assembly: Refers to: all node, element, and dof set up; computation of local ke and fee; assembly of those to global system. This step scales linearly versus n_e (ne)
  2. Linear algebra solution: $\mathbf{Ka} = \mathbf{F}$: We solve for unknown $\mathbf{a}$. Although conceptually simple, this step is a major source of computational cost. It scales higher than linear versus n_e ⇒ As the problem size increases this term becomes more dominant.
- Solution of $\mathbf{Ka} = \mathbf{F}$:
  - WE DO NOT OBTAIN $\mathbf{a}$ from $\mathbf{a} = \mathbf{K}^{-1}\mathbf{F}$: We do not invert $\mathbf{K}$.
  - We only solve the problem for the specific RHS of $\mathbf{F}$.
  - In Comparison $\mathbf{K}^{-1}$ corresponds to the solution of $\mathbf{Ka} = \mathbf{F}$ for n_f RHS of $\mathbf{F} = e_i$, $i = 1, \ldots, n_f$ where n_f is the number of rows (and columns) of $\mathbf{K}$.
  - We employ methods such as LU factorization that computationally only solve the problem for the given RHS $\mathbf{F}$.
  - We take advantage of the structure of stiffness matrix: symmetry, bandedness, sparsity in choosing the right solution technique.
  - order of free dofs affects band of the matrix → various algorithms reorder free dofs such that the matrix band get smaller and the solution cost is optimized.
  - In your term projects you can simply employ simply compute

$a = K^{-1} F$ or $K \backslash F$



node	p	pos	v	f
1	0	$\bar{1}$	0?	0
	0	$\bar{2}$	0?	0
2	1	1	0.0	0 ?
	0	$\bar{3}$	0?	-1
3	1	2	0.5	0 p
	1	3	0.0	0 ?

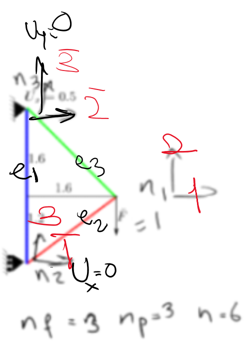
$$U_f = \begin{array}{c} 1 \\ 2 \\ 3 \end{array}\begin{bmatrix} -.2133 \\ -3.29 \\ -1.2 \end{bmatrix} \quad K_{nf \times nf} \quad F_n = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

size nf

$$F_p = \begin{array}{c} \bar{1} \\ \bar{2} \\ \bar{3} \end{array}\begin{bmatrix} \\ \\ \end{bmatrix}$$

e	LEM	dof Map	dofs	fe
1	[2,3]	[$\bar{1}$ 3 $\bar{2}$ $\bar{3}$]	[0.0 0 .5 0]	
2	[2,1]	[$\bar{1}$ 3 1 2]	[0.0 0 0 0]	
3	[3,1]	[$\bar{2}$ $\bar{3}$ 1 2]	[.5 0 0 0]	

Step 13: Assign a to nodes and elements



node	p	pos	v	f
1	0	$\bar{1}$	-.2123	0
	0	$\bar{2}$	-3.29	0
2	1	1	0.0	0 ?
	0	3	-1.2	-1
3	1	$\bar{2}$	0.5	0 p
	1	$\bar{3}$	0.0	0 ?

$$U_f = \begin{array}{c} 1 \\ 2 \\ 3 \end{array}\begin{bmatrix} -.2133 \\ -3.29 \\ -1.2 \end{bmatrix} \quad K_{nf \times nf} \quad F_n = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

size nf

$$F_p = \begin{array}{c} \bar{1} \\ \bar{2} \\ \bar{3} \end{array}\begin{bmatrix} \\ \\ \end{bmatrix}$$

e	LEM	dof Map	dofs	fe
1	[2 3]	[$\bar{1}$ 3 $\bar{2}$ $\bar{3}$]	[0.0 -1.2 .5 0]	
2	[2,1]	[$\bar{1}$ 3 1 2]	[0.0 -1.2 -.2123 -3.29]	
3	[3,1]	[$\bar{2}$ $\bar{3}$ 1 2]	[.5 0 -.2123 -3.29]	

for n = 1:nNodes
  for dofi = 1: node(n).nndof num dof for node (n)
    if node(n).ndof(dofi).p == false free dof
      posn = node(n).ndof(dofi).pos position of dof in global free F
      node(n).ndof(dofi).v = dofs(posn) set free dof val to corresponding val in global dofs (a) node d's
    end
  end
end

```
for e = 1:ne loop over elements
  for i = element(e).nedof loop over element dofs; nedof = # dof (n^e_dof)
    posn = element(e).dofMap(i) corresponding global position using dofMat (M^e_t)
    if (posn > 0) free dof
      element(e).edofs(i) = dofs(posn)
        set free element dof a^e to corresponding val in global dofs (a)
    end
  end
end
```

element ~