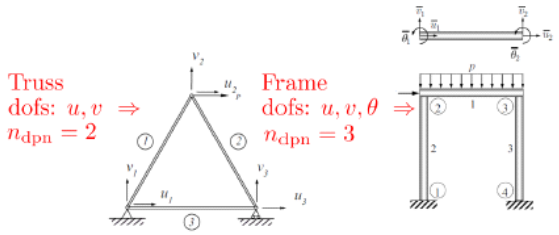
2017/11/07

Tuesday, November 07, 2017 7:52 AM

Steps 8 and 9 set element dofMap (8) element dofs edofs
(vector of unknowns, done in step 9)
Simplified pseudo-codes

Step 8: Element dof maps $\mathbf{M}_t^e$: Simplified limited case

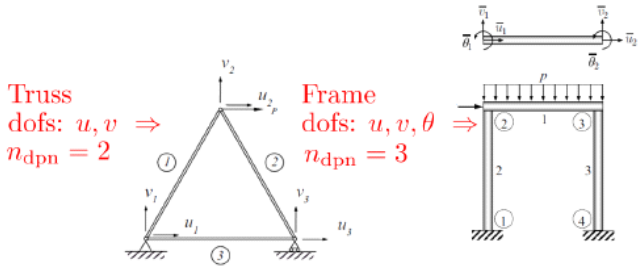


A simplified pseudo code looks like: ecdof = 1 dof counter for element
for en = 1: neNodes number of element nodes
  gn = LEM(en) global node number for element node en
  for endof = 1: ndofpn This number is fixed now, *e.g.*, 2 for 2D trusses
    dofMap(ecdof) = node(gn).dof(endof).pos
      gndof = endof, we bypass some steps here
    ecdof = ecdof + 1 increment counter
  end
end

- Two major simplifications are:
  - number of dof per node is fixed for all nodes.
  - gndof = endof: That is dof number endof of node en of element is the same dof in its corresponding global node. For example U_1 in trusses is always 1st dof for element nodes and 1st for global nodes as well.

420 / 456

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

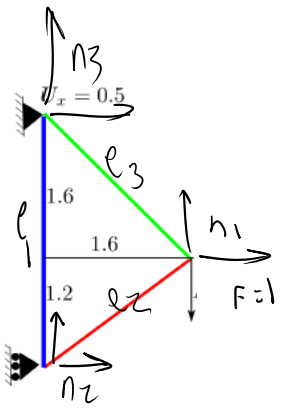


- Similar to steps 1, 2, and 8, step 9 can be greatly simplified if we assume all nodes share exactly the same set of dofs.
- Noting n_{dpn} (ndofpn) = Number of dof. per node, simplified merged steps 8 & 9 are:

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

422 / 456

Tuesday, November 07, 2017
7:52 AM



node	p	pos	V	f
1	false	1	?	0
	false	2	?	-1
2	true	1	0	
	false	3	?	0
3	true	2	.5	
	true	3	0	

$U_p = \begin{matrix} 2 \\ 3 \end{matrix}$ $\begin{bmatrix} \\ \\ \end{bmatrix}$

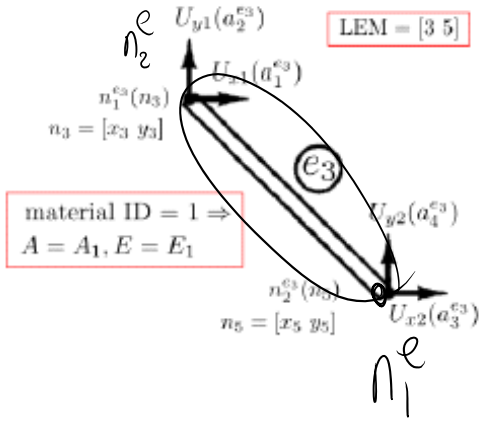
$f_p = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$ $\begin{bmatrix} \\ \\ \\ \end{bmatrix}$

e	LEM	dofMap	dofs	
1	[2, 3]	[1 2 3]	[0 ? .5]	
2	[2, 1]	[1 3 1 2]	[0]	
3	[3, 1]	[2 3 1 2]	[.5]	

Step 10: Computing element stiffness matrix

Three things needed:

1. Element type
2. Element connectivity (what nodes form the element)
3. Material and section properties



$$k^e = \begin{bmatrix} k^b_{2x2} & -k^b \\ -k^b & k^b \end{bmatrix}$$

$$k^b = \left(\frac{AE}{L}\right)^e \begin{bmatrix} c^2 & cs \\ cs & s^2 \end{bmatrix}$$

Geometry aspects

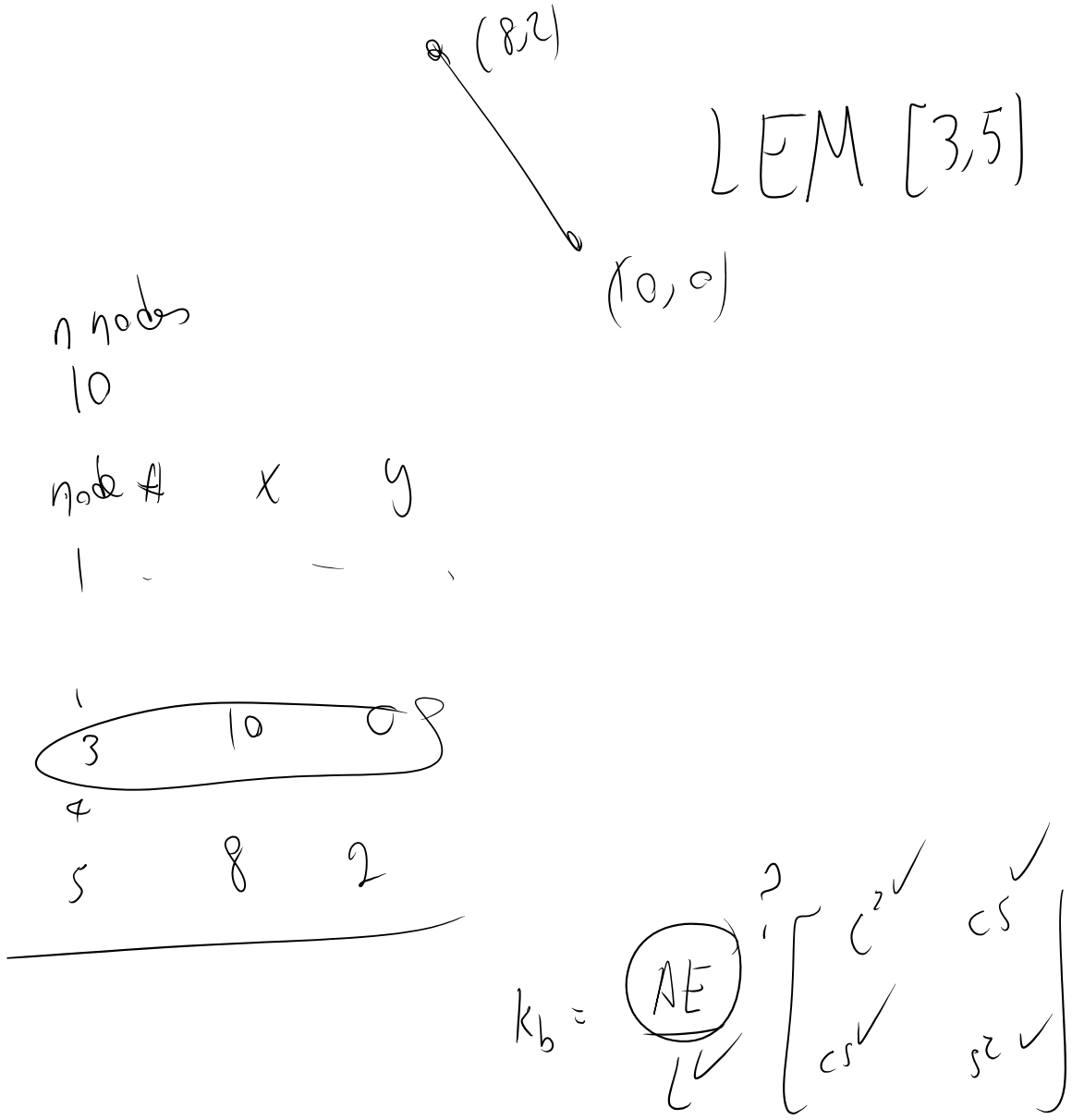
1. $L = \sqrt{\Delta x^2 + \Delta y^2}$
2. $c = \frac{\Delta x}{L}$
3. $s = \frac{\Delta y}{L}$

$\Delta x = x_{n_2^e} - x_{n_1^e}$

$\Delta y = y_{n_2^e} - y_{n_1^e}$

We don't store element node coordinates in it (either in element class or in the text file) because:

1. No need to store repeated geometry information of the same node in the global system.
2. What if we change the coordinate of a node (for example sensitivity analysis)

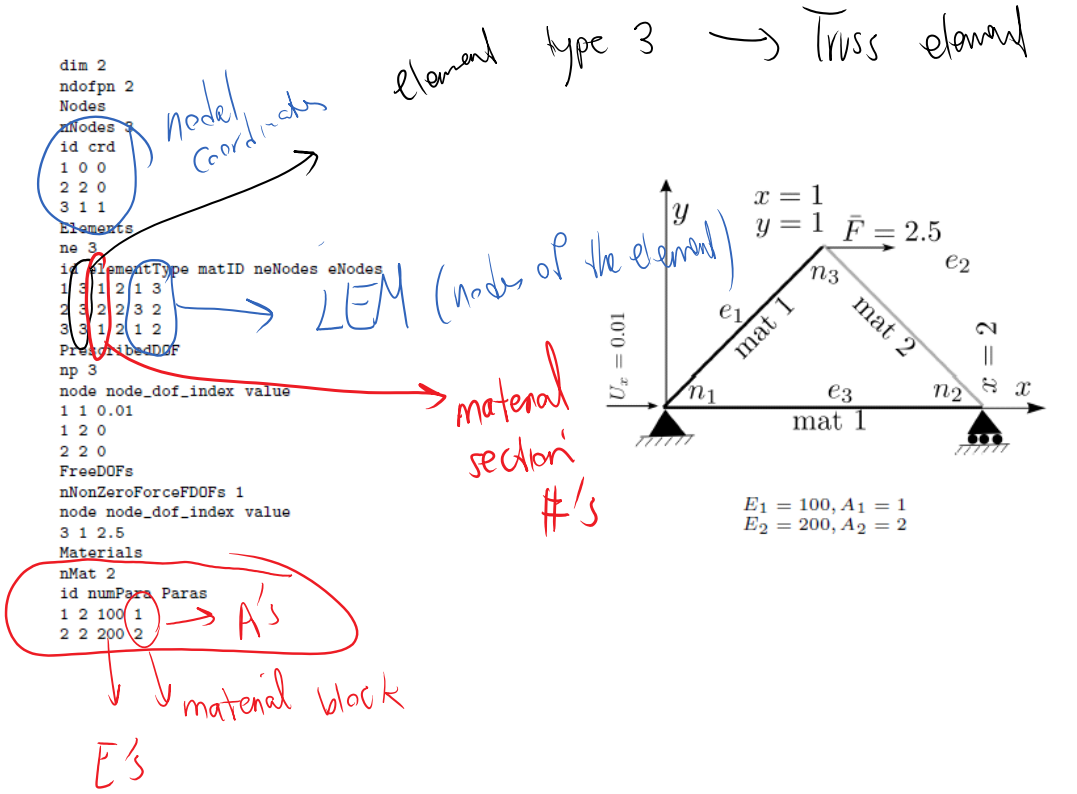


Again for material (E) and section (A) properties, we specify element material section number and read those later in the file.

Example

Input file format

dim 2 element type 3 → Truss element



Function for calculating stiffness matrix

Approach 1: Matlab (non-object oriented), Fortran, …

Function:

ComputeStiffness(type, …)
{
If (type == 0) // bar
{
}
Else if (type == 1)
{
}
…
}

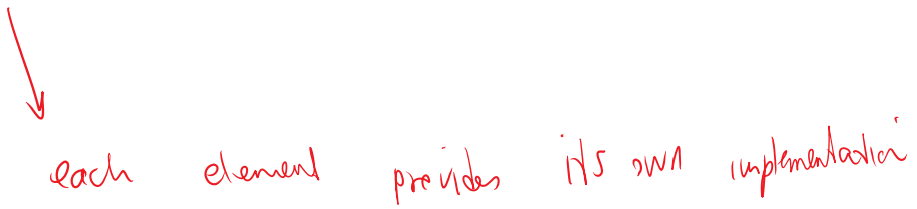
$k = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

Object-oriented design:

We have a general element class

```
class PhyElement
{

// Step 10: Compute element stiffness/force (ke, foe (fre: source term; fNe:
Neumann BC))
virtual void Calculate_ElementStiffness_Force() = 0;
};
```



Now for a bar element, we need to say a bar element is an element!

```
class PhyElementBar : public PhyElement
{

virtual void Calculate_ElementStiffness_Force();

}
```

it's a subclass

….
The function implementation is provided in a separate file for the bar element:

```
void PhyElementBar::Calculate_ElementStiffness_Force()
{
    // compute stiffness matrix:
    ke.resize(2, 2);
    double factor = A * E / L;
    ke(0, 0) = ke(1, 1) = factor;
    ke(1, 0) = ke(0, 1) = -factor;
}
```

If you want to learn how these elements are created, watch the C++ pre-recorded session and the notation of factory

FYI:

PhyElement* PhyElementFactory(ElementType eTypeIn)

Element forces:

$$f^e = f_f^e + f_N^e - f_D^e$$

f_f^e: force term, f_N^e: Neumann, f_D^e: k_a^{ee}

$f_f^e + f_N^e$: physics specific

f_0^e: sum of all the other forces

f_D^e: always the same formula, general

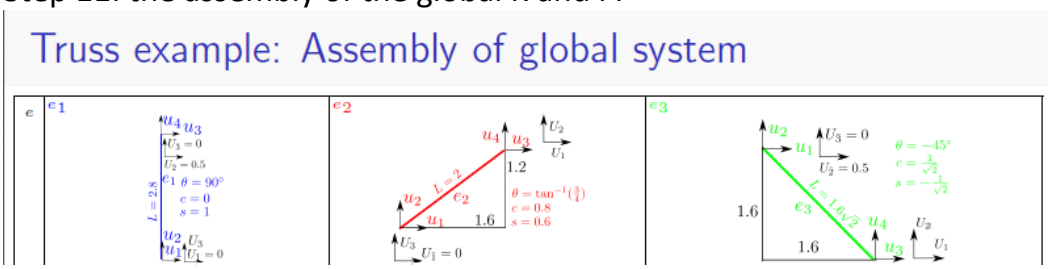
- element specific
- becomes virtual function in O.O. programming

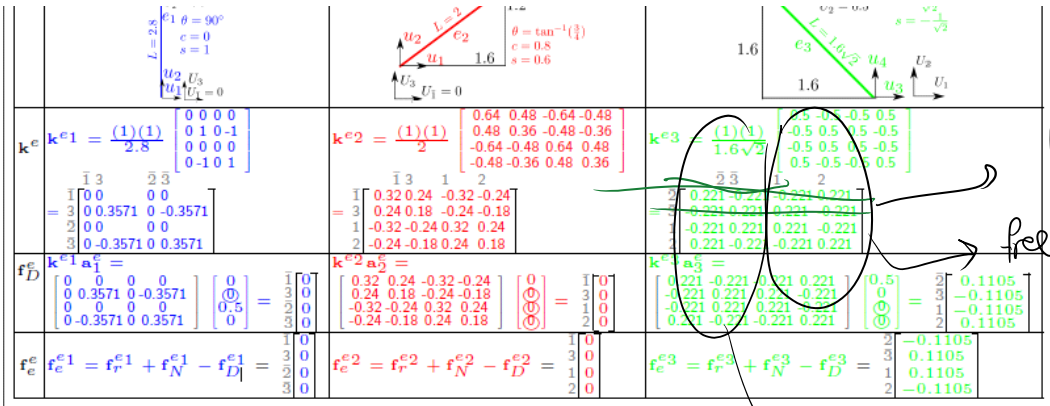
So the virtual function we discussed:

1. Calculates ke
2. Calculates feo

virtual void Calculate_ElementStiffness_Force();

Step 11: the assembly of the global K and F:





$$\mathbf{K} = \begin{bmatrix} 0.32+0.221 & 0.24-0.221 & -0.24 \\ 0.24-0.221 & 0.18+0.221 & -0.18 \\ -0.24 & -0.18 & 0.3571+0.18 \end{bmatrix} = \begin{bmatrix} 0.5410 & 0.019 & -0.24 \\ 0.019 & 0.401 & -0.18 \\ -0.24 & -0.18 & 0.5371 \end{bmatrix}$$

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

p. rows useless (assembly)

free columns K(i,j) A=k(i,j)

p. columns used in calculating $f_D^e = k a$

initiated by free of forces

for e_3 dofMap = [2 3 1 2]

for i = 1 : numbe Element dof (4)

I = dofMap(i) (i=1, I=2)

if I < 0 prescribed

continue

end

F(I) += $f^e(i)$; assembly of force

for j = 1 : element # of dof (4)

J = dofMap(j)

if J < 0

continue

end

K(I, J) += $k^e(i,j)$

end

$K(I,J) = K(I,J) + k^e(i,j);$

end

end

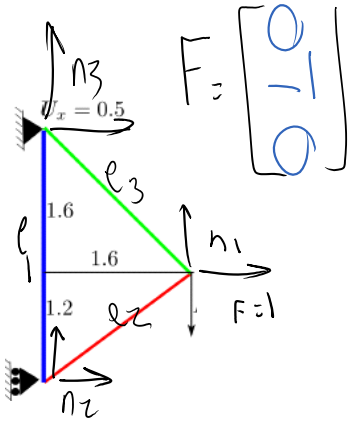
```
for e = 1:ne loop over elements
  fee = feo element total force = element all forces except essential force
  for i = 1:nedof loop over rows of ke; nedof = element # dof
    I = dofMap(i) local to global dof map M_t^e
    if (I > 0) I corresponds to a free dof, we skip prescribed dofs
      for j = 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 f_D^e = k^e 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 12: Solve global (free) dof $\mathbf{a}$ 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 $\Rightarrow$ 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

At this stage for the truss example we have:

Tuesday, November 07, 2017
7:52 AM



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

node	p	pos	v	f
1	false	1		0
	false	2		-1
2	true	$\bar{1}$	0	
	false	3		0
3	true	$\bar{2}$	.5	
	true	$\bar{3}$	0	

$$u_p = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} -.2123 \\ -3.29 \\ -1.20 \end{bmatrix}$$

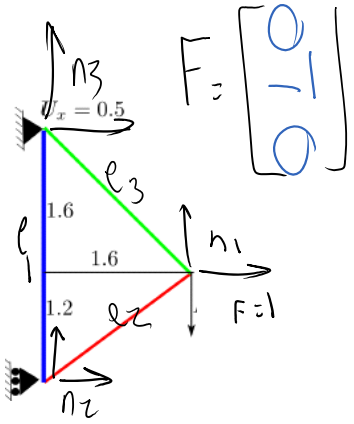
$$f_p = \begin{matrix} \bar{1} \\ \bar{2} \\ \bar{3} \end{matrix} \begin{bmatrix} \\ \\ \end{bmatrix}$$

e	LEM	dofMap	dofs
1	[2, 3]	[$\bar{3}$ 2 $\bar{3}$]	[.5]
2	[2, 1]	[$\bar{1}$ 3 1 2]	[0]
3	[3, 1]	[$\bar{2}$ $\bar{3}$ 1 2]	[.5]

Solution is done:

FEM postprocessing:

Tuesday, November 07, 2017



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

node unknowns: vs of free dofs & fs at pres. dofs

(511)

node	p	pos	v	f
1	false	1	?	0
	false	2		-1
2	true	$\bar{1}$	0	?
	false	3		0
3	true	$\bar{2}$	.5	
	true	$\bar{3}$	0	

$$u_p = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} -.2123 \\ -3.29 \\ -1.20 \end{bmatrix}$$

$$f_p = \begin{matrix} \bar{1} \\ \bar{2} \\ \bar{3} \end{matrix} \begin{bmatrix} ? \\ \\ \end{bmatrix}$$

e	LEM	dofMap	dofs	fe
1	[2, 3]	[$\bar{3}$ 2 $\bar{3}$]	[0 ? .5 0]	?
2	[2, 1]	[$\bar{1}$ 3 1 2]	[0]	
3	[3, 1]	[$\bar{2}$ $\bar{3}$ 1 2]	[.5]	

We fine:

Nodes:

- Free dofs -> value
- Prescribed dofs -> forces

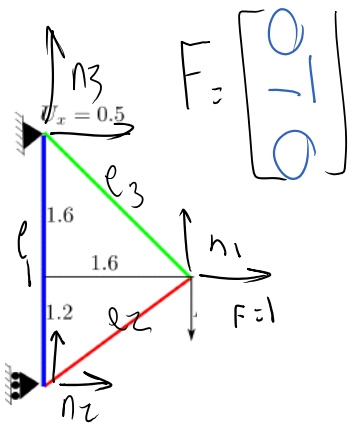
Elements:
- Dofs (element values): only the free dofs are unknowns at this stage.
- Element forces

Global Level:
- Support forces Fp

Step 13: Assign a to nodes and elements

```
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)
      end
   end
end
```
435 / 456

Step 13a filled in the tables below:



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

node unknowns: vs of free dofs & fs of pres. dofs

node	p	pos	v	f
1	false	1	-.2123	0
	false	2	-3.29	-1
2	true	1	0	?
	false	3	-1.2	0
3	true	2	.5	
	true	3	0	

step 13a

$U_p = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} -.2123 \\ -3.29 \\ -1.20 \end{bmatrix}$

$f_p = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix}$

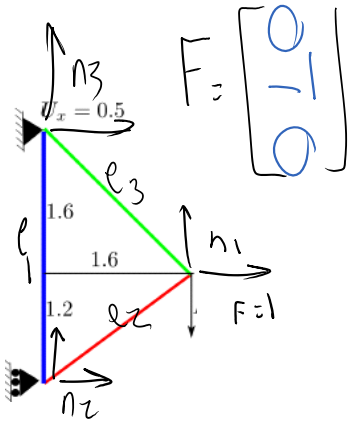
e	LEM	dofMap	dofs	fe
1	[2, 3]	[1 2 3]	[0 ? .5 0]	?
2	[2, 1]	[1 3 1 2]	[0]	
3	[3, 1]	[2 3 1 2]	[.5]	

Please **write** these in your notes and leave several lines blank below it

Step 13b: filling the element dof unknowns

```
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
```


node unknowns: #s of free dofs & #'s of pres. dofs



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

node	p	pos	v	f
1	false	1	-.2123	0
	false	2	-3.29	-1
2	true	$\bar{1}$	0	?
	false	3	-1.2	0
3	true	$\bar{2}$	.5	
	true	$\bar{3}$	0	

step 13b

step 13a

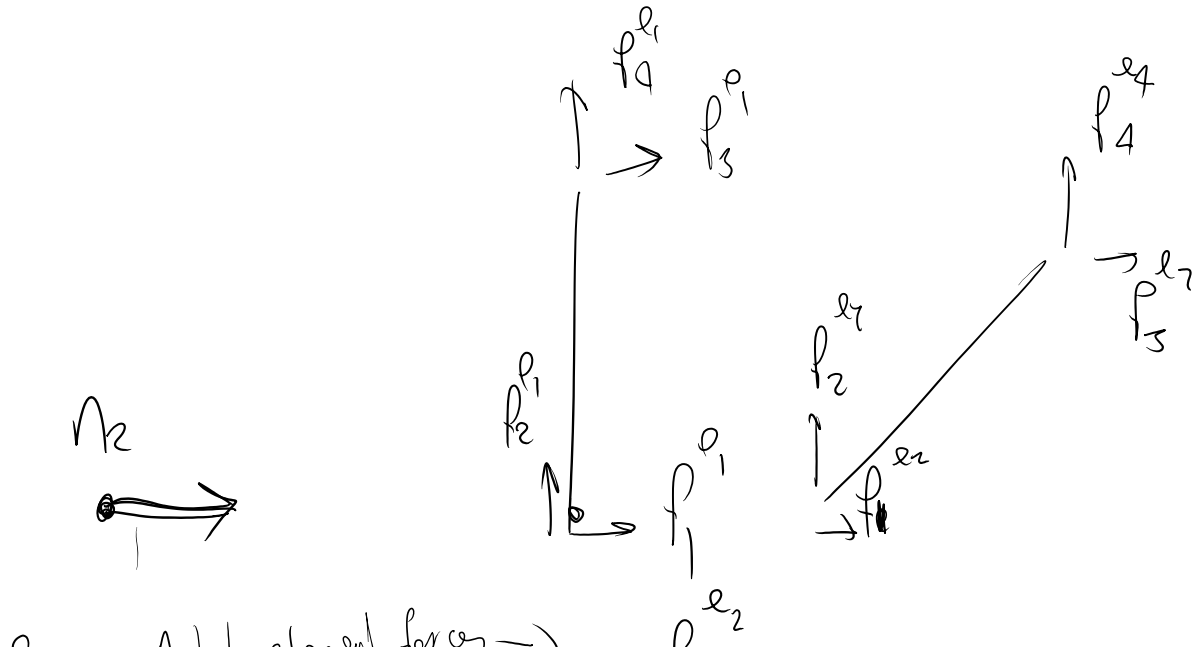
$$u_p = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} -.2123 \\ -3.29 \\ -1.20 \end{bmatrix} \quad (11)$$

$$f_p = \begin{matrix} \bar{1} \\ \bar{2} \\ \bar{3} \end{matrix} \begin{bmatrix} ? \\ . \\ \end{bmatrix}$$

e	LEM	dofMap	dofs	fe
1	[2, 3]	[$\bar{1}$ 3 $\bar{2}$ $\bar{3}$]	[0 0 -1.2 .05 0]	?
2	[2, 1]	[$\bar{1}$ 3 1 $\bar{2}$]	[0]	
3	[3, 1]	[$\bar{2}$ $\bar{3}$ 1 2]	[.5 0]	

For support forces we need to do in two steps:

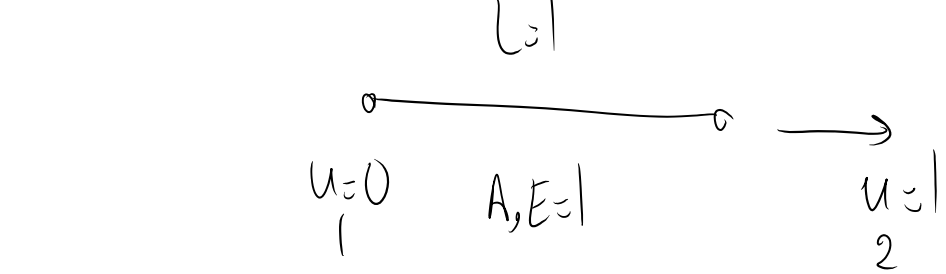
Step a: Compute element forces



Step 2: Add element forces ⟹ support forces

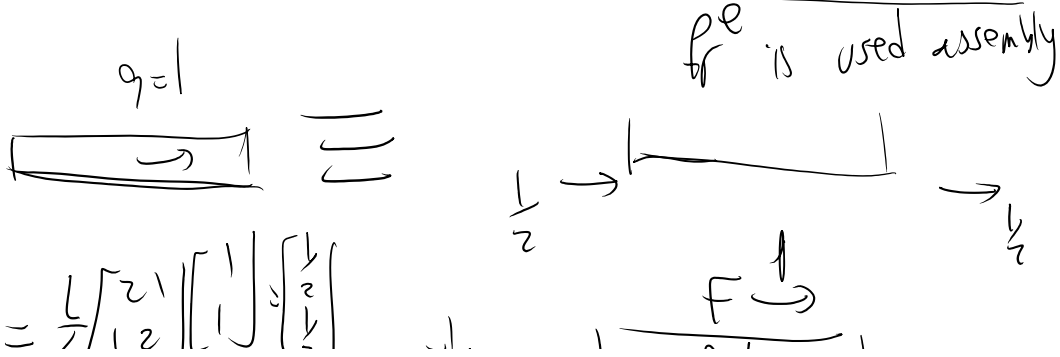
$$\left(\text{Force}\right)_{n_2,\text{dof}\,1} = f_1^{e_2} + f_1^{e_4}$$

Step a:
How to calculate element forces



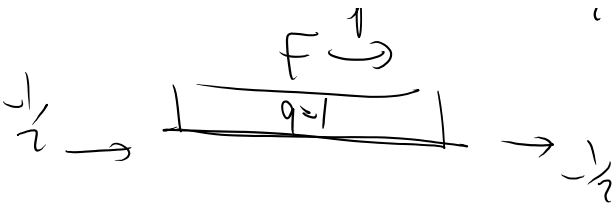
$$\begin{bmatrix} f_1^e \\ f_2^e \end{bmatrix} = k^e a^e = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

nodal force $= f_D^e = k^e a^e$



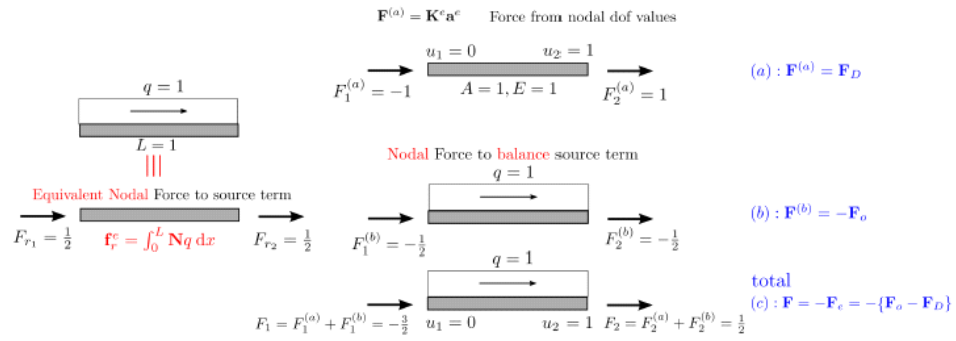
$$f_r^e = \frac{L}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$$

$f_r^e = \frac{L}{6}\begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}\begin{bmatrix}1\\ 1\end{bmatrix}q\begin{bmatrix}1/2\\ 1/2\end{bmatrix}$



$f_{nodal\ forces}^e = f_D^e - f_r^e = -(f_r^e - f_D^e)$ — $-f_r^e$ is applied on element nodes

Step 14: Compute prescribed dof forces: a) elements



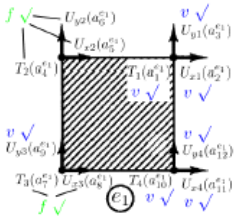
Nodal forces of the element

$= f^e = [redacted] + \ldots =$

$= -\left(f^e_{nodal} - f^e_{\lambda}\right)$

How to actually compute element forces:

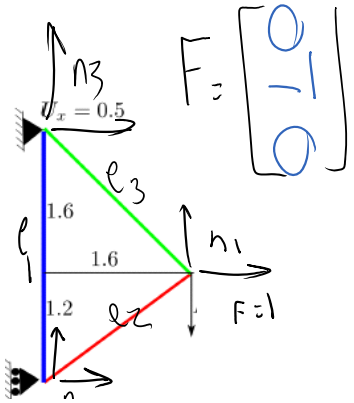
Step 14: Compute prescribed dof forces: a) elements



- Steps for calculation of $\mathbf{F}_p$ from elements: 1. calculation of element nodal forces; 2. Assembling those to correct position in $\mathbf{F}_p$. Corresponding pseudocode:

```
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 prescribed dof, we skip free dofs
      for j = 1:nedof loop over columns of ke. ALL columns (dofs) of p and f used
        fee(i) = fee(i) - ke(i, j) * edofs(j) edofs: element dofs = a^e
      end
      Fp(-I) = Fp(-I) - fee(i)
        1. element's total force fee component i'th is computed→added to Fp(-I)
        2. -I used because I < 0: prescribed dof
        3. fee is subtracted
    end
  end
end
```

$$f^e = -\left(f^e_{other} - f^e_D\right)$$



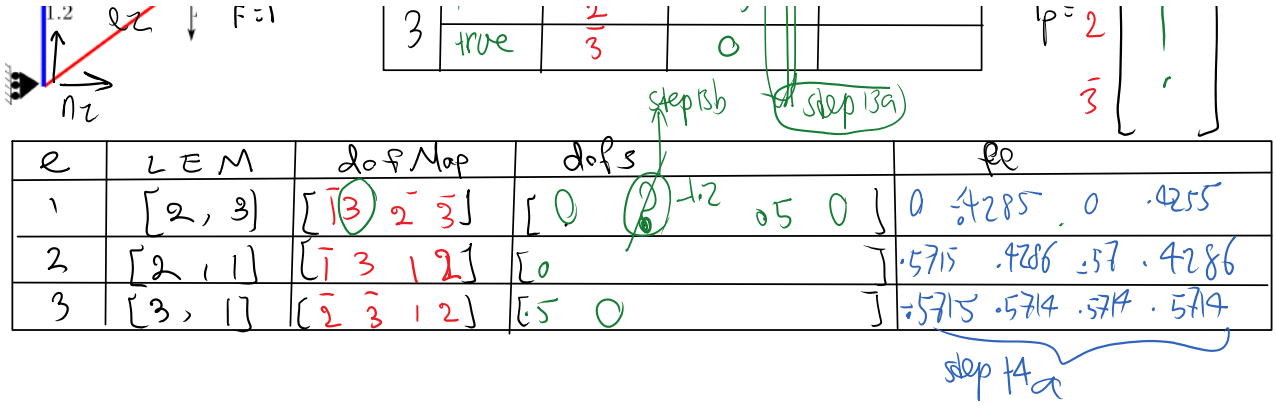
$$F = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

node unknowns: #'s of free dofs & #'s of pres. dofs

node	p	pos	v	f
1	false	1	-2123	0
	false	2	-3.29	-1
2	true	$\bar{1}$	0	?
	false	3	-1.2	0
3	true	$\bar{2}$	.5	
	true	$\bar{3}$	0	

$$v_p = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}\begin{bmatrix} -2123 \\ -3.29 \\ -1.20 \end{bmatrix} \quad (SI)$$

$$f_p = \begin{matrix} \bar{1} \\ \bar{2} \\ \bar{3} \end{matrix}\begin{bmatrix} ? \\ \\ \end{bmatrix}$$

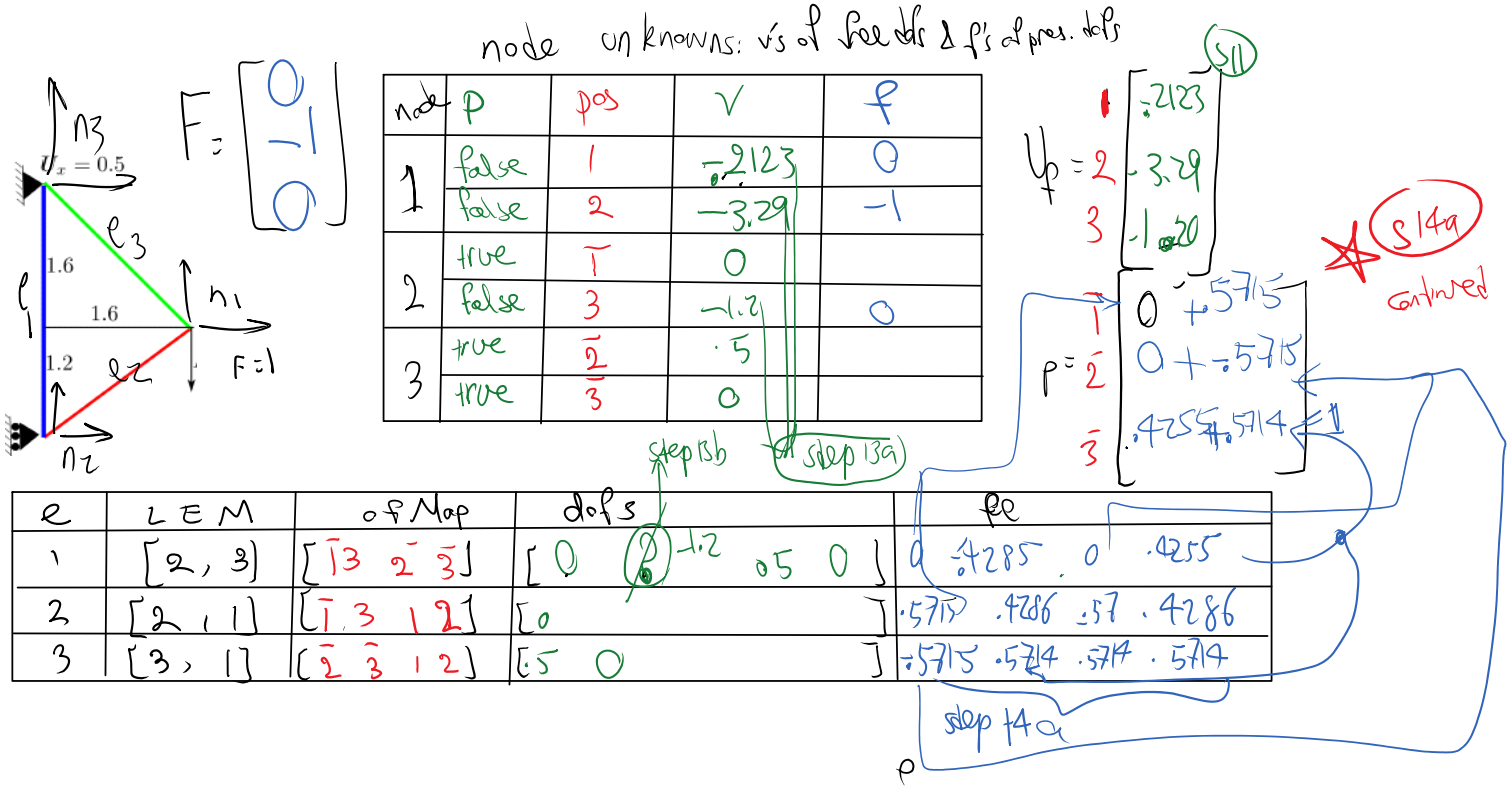


Step 14a … continued -> computing forces in Fp

We loop over the elements

Loop over their dofs

For prescribed dofs add the element force to global Prescribed forces Fp



Step 14: Compute prescribed dof forces: b) nodes

Step 14b:

Add Fp values to prescribed node forces

To do so:

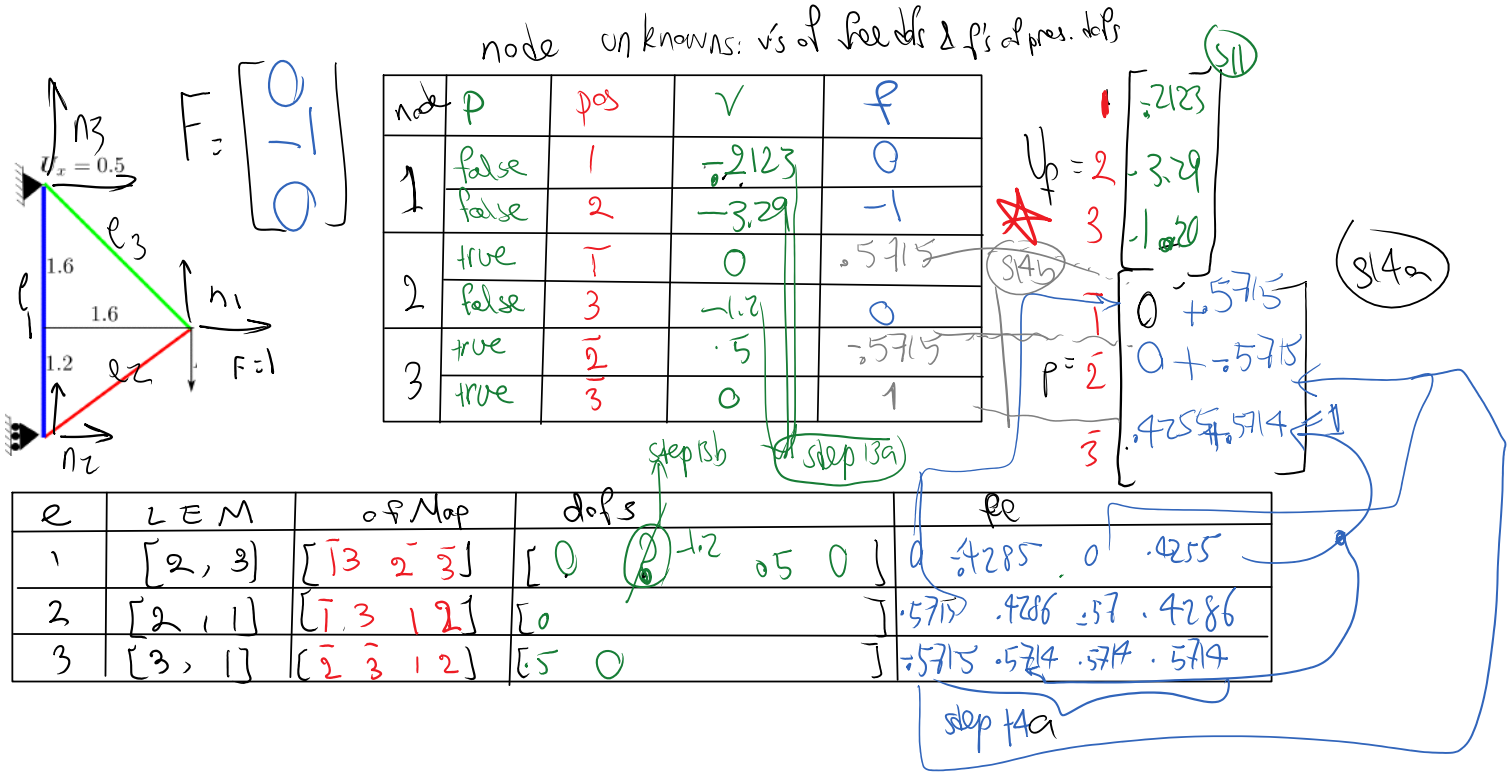
Loop over nodes

Loop over dofs of nodes

For those that are prescribed use their position to get the right force from global reaction forces (Fp)

Pseudo-code

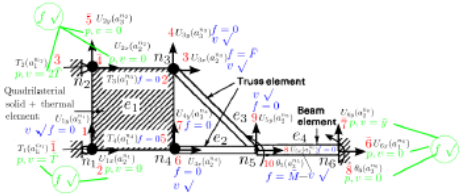
```
for n = 1:nNodes
  for dofi = 1: node(n).nndof num dof for node (n)
    if node(n).ndof(dofi).p == true prescribed dof
      posn = node(n).ndof(dofi).pos position of dof in global prescribed force F_p
      node(n).ndof(dofi).f = Fp(-posn)
        1. set prescribed dof force to corresponding force in global Fp (F_p)
        2. posn < 0; prescribed dof
    end
  end
```



At this point all node, element, and global values are computed and we can output things in the last step (step 15)
The only point to mention is that for elements, again each element type has its own virtual (in object oriented framework) output function. For example, for bars and trusses one axial force may suffice, whereas for beams and frames we may compute y, \theta, M, V at m

number of points along their length. The following slides describe this process

Step 15: Compute/output nodes & elements: a) nodes

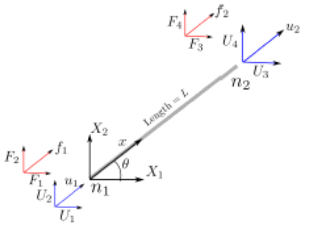


- After free dof values and prescribed dof forces are calculated, we can output the results.
- Node output is general, while element output is specific to its type. Sample pseudocode:

```
for n = 1:nNodes
  print: node (n) from (nNodes) nodes
  coord: (node(n).coordinate)
  for dofi = 1: node(n).nndof num dof for node (n)
    print dof:
      dof (dofi) from (node(n).nndof) dofs
        Field: (node(n).ndof(dofi).Field)
        Index: (node(n).ndof(dofi).index)
        Value: (node(n).ndof(dofi).value)
        Force: (node(n).ndof(dofi).force)
        prescribed: (node(n).ndof(dofi).p) may only be output for debugging purposes
        position: (node(n).ndof(dofi).position) may only be output for debugging purposes
  end
end
```

- Refer to slides 398 and 397 for other data members in node and dof objects.

Step 15: Compute/output nodes & elements: b) elements



- The computation and output of an element is again a black box function.
- Each element has its own specific compute/output function.
- For example, for truss elements we only need to compute axial force (figure above, also *cf.* 315 and (387)):

$$T = AE\,\{c(U_3 - U_1) + s(U_4 - U_2)\}$$

- For a beam element, moment M and shear force V change along the beam and they need a completely different process for computation (*cf.* page 374).
- These functions will be virtual in an object-oriented language. Sample pseudocode:
- For example, the virtual function's implementation for a truss element computes and outputs T from equation above/

```
for e = 1:ne loop over elements
  elements(e).Compute_Output_Element() virtual function on page 394
end
```