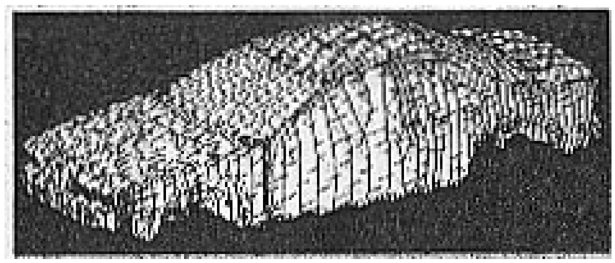
Automotive Application

Of Sparse Solver (ODU)



44,188 Nodes

48,894 Elements

263,574 Equations (263,096 active)

Solution took 83 sec for full static analysis

(on 1 Cray C-90 processor)

(42 sec reordering, 41 sec factor)

Fillin: 6,267,099 before, 36,744,123 after

+

Lanczos Eigen-Solver etc…

EFFICIENT SPARSE EQUATION SOLVER WITH UNROLLING STRATEGIES FOR COMPUTATIONAL MECHANICS

J.Qin[#], D.T.Nguyen[#], T.Y.P.Chang[*], and P. Tong[*]

[#]Old Dominion University, Virginia, USA

[*]Hong Kong University of Science and Technology, Kowloon, Hong Kong

$$[K] = [L]\,[D]\,[L]^T \tag{1}$$

$$\begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \begin{bmatrix} D_1 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & D_3 \end{bmatrix} \begin{bmatrix} 1 & L_{21} & L_{31} \\ 0 & 1 & L_{32} \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

SPARSE ALGORITHMS

Basic Choleski

$$[K]\{Z\} = \{f\} \tag{1}$$

$$[U]^T \underbrace{[U]\{Z\}}_{\vec{y}} = \{f\} \tag{3}$$

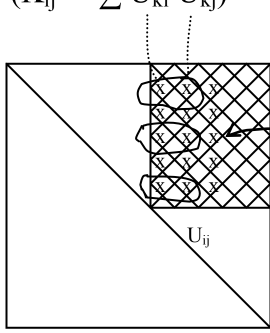
$$[U]\{Z\} = \{y\} \tag{4}$$

$$\begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} = \begin{bmatrix} u_{11} & 0 & 0 \\ u_{12} & u_{22} & 0 \\ u_{13} & u_{23} & u_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \tag{5}$$

Eq. (5) can be used to solve for [U], thus; generalize to:

$$U_{ii} = \sqrt{K_{ii} - \sum_{k=1}^{i-1} U_{ki}^{\,2}} \tag{6}$$

$$U_{ij} = 1/U_{ii} * \left(K_{ij} - \sum U_{ki} U_{kj}\right) \tag{7}$$



Information required to factorize current row #i

Current i^{th} row = 5^{th} row

3 nested Do loops

what if $u_{ii} \approx 0$??

$K_{ij} = 0$

However u_{ij} may be non-zero

→ Fills-in effects

$$u_{57} = 1/u_{55} * (K_{57} - u_{15}u_{17} - u_{25}u_{27} - u_{35}u_{37} - u_{45}u_{47})$$

Basic L D L^T Algorithm

$$[K] = [L]\,[D]\,[L]^T \tag{8}$$

$$\begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \begin{bmatrix} D_1 & & \\ & D_2 & \\ & & D_3 \end{bmatrix} \begin{bmatrix} 1 & L_{21} & L_{31} \\ 0 & 1 & L_{32} \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Eq. (9) can be solved for [L] & [D], formulas for general size n can be found in K.J. Bathe, or Pin Tong's books!

```
      c……  Assuming row 1 has been factorized earlier
           Dϕ  11    I = 2, N
           Dϕ  22   K = 1, I-1
      c……  Compute the multiplier (Note: U represent L^T)
           XMULT = U(K , I)/U(K , K)
           Dϕ    33  J =  I , N
           U(I , J) = U(I , J) – XMULT * U(K , J)
      33   continue
           U(K , I) = XMULT
      22   continue
      11   continue
```

Table 1 : Skelaton FORTRAN Code For L D L^T

(assuming the matrix U is completely full)

what if $u_{kk} = 0$?? (usually the case for "indefinite" matrix)

$1/u(k,k) \approx [u(k,k)]^{-1}$

$$[K] = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \quad (10)$$

will lead to the following factorized matrix

$$[U] = \begin{bmatrix} 2 & -1/2 & 0 \\ & 3/2 & -2/3 \\ & & 1/3 \end{bmatrix} \quad (11)$$

From Eq. (11), one can obtain

$$[D] \equiv \begin{bmatrix} \text{Diagonal} \\ \\ \text{of } U \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3/2 & 0 \\ 0 & 0 & 1/3 \end{bmatrix} \quad (12)$$

$$\text{and } [L]^T \equiv \begin{bmatrix} \text{Upper} \\ \text{Triangular} \\ \text{Portion of U} \end{bmatrix} = \begin{bmatrix} 1 & -1/2 & 0 \\ & 1 & -2/3 \\ & & 1 \end{bmatrix} \quad (13)$$

Sparse Storage Scheme

$$[K] = \begin{bmatrix} 11 & 0 & 0 & 41 & 0 & 52 \\ & 44 & 0 & 0 & 63 & 0 \\ & & 66 & 0 & 74 & 0 \\ & & & 88 & 85 & 0 \\ & & & & 110 & 97 \\ & & & & & 112 \end{bmatrix} \tag{14}$$

$$\begin{bmatrix} x & & & 1^{st} & & 2^{nd} \\ & x & & & 3^{rd} & \\ & & x & & 4^{th} & \\ & & & x & 5^{th} & \\ & & & & x & 6^{th} \\ & & & & & x \end{bmatrix}$$

$$[U] = \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} x & 0 & 0 & x & 0 & x \\ & x & 0 & 0 & x & 0 \\ & & x & 0 & x & 0 \\ & & & x & x & \boxed{F} \\ & & & & x & x \\ & & & & & x \end{bmatrix} \end{array} \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \tag{15}$$

$$\text{Istartrow}\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 = N+1 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 7 \end{pmatrix} \tag{16}$$

$$\text{Icolumn}\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 = \text{Ncoef} \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 5 \\ 5 \\ 5 \\ 6 \end{pmatrix} \tag{17}$$

$$\mathrm{Diag}\begin{pmatrix}1\\2\\3\\4\\5\\6=N\end{pmatrix}=\begin{pmatrix}11\\44\\66\\88\\110\\112\end{pmatrix}$$ (18)

$$\mathrm{AK}\begin{pmatrix}1\\2\\3\\4\\5\\6=\mathrm{Ncoef}\end{pmatrix}=\begin{pmatrix}41\\52\\63\\74\\85\\97\end{pmatrix}$$ off-diag. values (19)

Re-ordering Algorithms

- RCM & GS are good for skyline and/or variable bandwidth algorithms (Minimize bandwidth/column heights)
- ND & MMD are good for Sparse algorithms (Minimize "Fills-in")

Sparse Symbolic Factorization

$$\mathrm{Jstartrow}\begin{pmatrix}1\\2\\3\\4\\5\\6\\7=N+1\end{pmatrix}=\begin{pmatrix}1\\3\\4\\5\\7\\8\\8\end{pmatrix}$$ ---- after "fills-in" ----- (2C

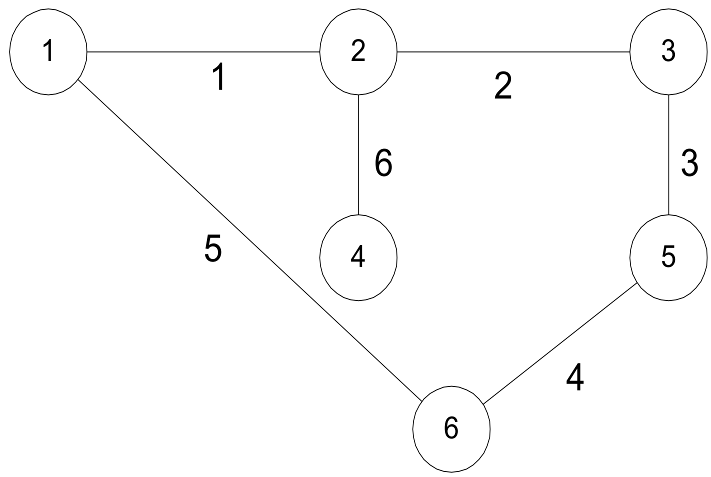
$$\mathrm{Jcolnum}\begin{pmatrix}1\\2\\3\\4\\5\\6\\7=\mathrm{Ncoef2}\end{pmatrix}=\begin{pmatrix}4\\6\\5\\5\\5\\6\\6\end{pmatrix}$$ ---- after "fills-in" ----- (21

➔ **Need to use "Chanined List" idea for better efficiency**

```
c……  Assuming row 1 has been factorized earlier
      Dϕ  11     I = 2, N
      Dϕ  22    K = Only those previous rows which have contribution to current
                row I
c……  Compute the multiplier (Note: U represent L^T)
      XMULT = U(K , I)/U(K , K)
      Dϕ    33   J =  appropriated column numbers of row K
      U(I , J) = U(I , J) – XMULT * U(K , J)
33    continue
      U(K , I) = XMULT
22    continue
11    continue
```

Table 2 : Pseudo FORTRAN Skelaton Code For Sparse LDL^T Factorization

Figure 3.2.1 Example of an adjacency structure



ADJNCY	2	6	1	3	4	2	5	2	3	6	1	5

XADJ	1	3	6	8	9	11	13
node number	1	2	3	4	5	6	

[K] =

	1	2	3	4	5	6
1	x	x				x
2	x	x	x	x		
3		x	x		x	
4		x		x		
5			x		x	x
6	x				x	x

Sparse Numerical Factorization

The "tricky" book-keeping process here is VERY SIMILAR to the chained list ideas used in symbolic phase!

Sparse Forward & Backward Phase

Relatively simple

"Master" (or "Super") Degree-of-Freedom

[K] =

1	2	3	4	5	6	7	8	9	10	11	12	13	14	
x	x	x			x			x	x	x		x	x	1
x	x	x			x			x	x	x		x	x	2
x	**x**	x			x			x	x	x		x	x	3
			x	x		x	x					x		4
				x		x	x					x		5
x	**x**	**x**			x	x	x	F	F	F		x	F	6
						x	x	x	x	F		x	x	7
							x	x	x	F		x	x	8
x	**x**	**x**						x	x	F		x	x	9
x	**x**	**x**	Sym						x	F		x	x	10
x	**x**	**x**								x	x	x	x	11
											x	x	x	12
x	**x**	**x**										x	x	13
x	**x**	**x**											x	14

Dot-Product

(42)

```
c......  Assuming row 1 has been factorized earlier
          Dϕ  11     I = 2, N
          Dϕ   22    K = Only those previous "MASTER" rows which have contribution to current
                         row I
c......  Compute the multiplier (Note: U represent L^T)
         NSLAVEDOF = MASTER(I) - 1
               XMULT = U(K , I)/U(K , K)
                    *
                    *
                    *
               XMUL_m = U(K+m , I)/U(K+m , K+m)
c......  m = 1,2,.......NSLAVEDOF
         Dϕ     33   J =  appropriated column numbers of "master" row# k
             U(I , J) = U(I , J)  -  XMULT * U(K , J)
                                  -  XMUL_m*U(K+m,J)
   33    continue
               U(K , I) = XMULT
               U(K+m , I) = XMUL_m
   22    continue
   11    continue
```

Table 2 : Pseudo FORTRAN Skelaton Code For Sparse LDL^T Factorization With Unrolling Strategies

$$\text{Master}\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 = N \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \\ 0 \\ 2 \\ 0 \\ 1 \\ 4 \\ 0 \\ 0 \\ 0 \\ 4 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

"Chained List" KEY Ideas → 1^{st} step: consider original non-zeroes
→ 2^{nd} step: consider "Fills-in"

Recalled [U] =

1	2	3	4	5	6	
x	0	0	x	0	x	1
	x	0	0	x	0	2
		x	0	x	0	3
			x	x	F	4
				x	x	5
					x	6

Step 0 : Initialize $IchainL\begin{pmatrix}1\\2\\3\\4\\5\\6\end{pmatrix} = \{0\} = \text{Loc update}\begin{pmatrix}1\\2\\3\\4\\5\\6\end{pmatrix}$

Step 1 : Consider "original" row i = 1

Since IchainL (1) = 0 → no contributions from previous rows

IchainL (4) = 1 → "Future" row# 4 will need row# 1

IchainL (1) = 1

LocUpdate (1) = 1

Step 2 : Consider "original" row I = 2

Since IchainL (2) = 0 → no contributions from previous rows

IchainL (5) = 2 → row 2 will have contribution to "future" row# 5

IchainL (2) = 2

LocUpdate (i = 2) = 3

Step 3 : Consider "original" row i = 3

Since IchainL (i = 3) = 0 → get no contributions from previous rows

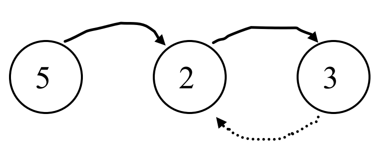
"Current" Row 3 will also have contribution to "future" row# 5

Hence, row# 3 has to be added into the "chained list" of

"future" row# 5 → IchainL (3) = 2

IchainL (2) = 3 → overwrite old value!

Thus: LocUpdate (i = 3) = 4



Step 4 : Consider "original" row i = 4

Since $\left\{\begin{array}{l} \text{IchainL } (i = 4) = 1 \\ \text{\& IchainL } (1) = 1 \end{array}\right\}$ Thus, "current" row 4 needs contribution from previous row# 1, only.

Now, LocUpdate (1) = $\underbrace{\text{LocUpdate (1)}}_{\equiv 1}$ + 1 = 2[nd] location

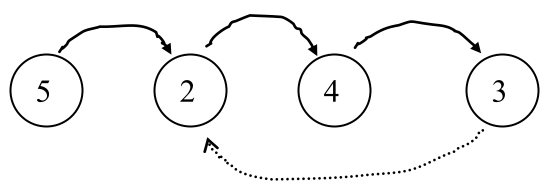
We need to have "Fills-in", occurs in col.# 6 of "current" row# 4	←	Nonzero occurs in col.# 6 of previous row# 1

Also, since the first nonzero of the "original" row# 4 occurs in col.# 5 → "current" row# 4 will <u>ALSO</u> have contributions to "future" row# 5. Thus:

$\left\{\begin{array}{l} \text{IchainL (4) = 3} \\ \text{IchainL (2) = 4} \end{array}\right.$ ⋯⋯> overwrite old value!

LocUpdate (i = 4) = 5

Hence



Step 5 : Consider "original" row i = 5

Now $\left.\begin{array}{l} \text{IchainL (I = 5) = 2} \\ \text{IchainL (2) = 4} \\ \text{IchainL (4) = 3} \end{array}\right\}$ Thus, these "previous" rows will have contributions to "current" row# 5

IchainL (3) = 2

Will not be true for numerical factorization

However, since the "original" $K_{5,6}$ = non-zero

Hence row#5 is already full → previous rows #2, #4, #3 will <u>not</u> have any "fills-in" effects on current row#5

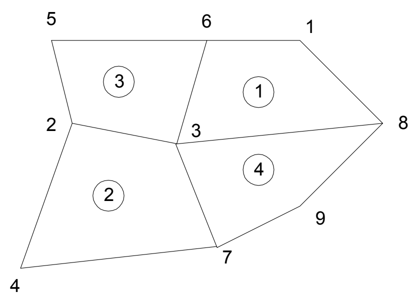
Step 6 : Stop, no need to consider <u>LAST row (no fills-in)</u> !

```
       Subroutine metisreord(NEQNS,XADJ,ADJNCY,PERM,INVP)

       integer XADJ(1), ADJNCY(1), PERM(1), INVP(1)
       integer itemp(8)

c .....................................................................
c Purpose : driver to call METIS_NODEND routine from the Metis Library
c          METIS_NodeND (int *n, idxtype *xadj, idxtype adjncy, int *num-flag
c                       int *options, idxtype *perm, idxtype *iperm)
c INPUT   : NEQNS
c          XADJ
c          ADJNCY
c output  : PERM
c          INVP
c .....................................................................

c
c      options flags
c      itemp(0)=0
c      itemp(1)=3
c      itemp(2)=1
c      itemp(3)=2
c      itemp(4)=0
c      itemp(5)=1
c      itemp(6)=0
c      itemp(7)=1

       do i = 0,7
        itemp(i)=0
       enddo

       write(23,*) 'METIS =', (itemp(i),i=0,7)
       ncount=1               ! 0 for C, 1 for Fortran
c      METIS_NodeND (int *n, idxtype *xadj, idxtype adjncy, int *num-flag
c                  int *options, idxtype *perm, idxtype *iperm)
       call METIS_NodeND(NEQNS,XADJ,ADJNCY,ncount,itemp,PERM,INVP) ! fast
c...or..call METIS_NodeWD(NEQNS,XADJ,ADJNCY,ncount,itemp,PERM,INVP) ! slow
       return
       end
```

PERM(i)=j
Perm(1)=8
Perm(old#)=new#
INVP(i)=j
INVP(new#)=old#

XADJ(n+1)
ADJNCY(2*ncoff)
Itemp(8)
PERM(n)
INVP(n)

EFFICIENT SPARSE ASSEMBLY



To simplify the discussions, <u>assuming</u> <u>each node</u> has <u>1 dof</u>. However, the same algorithms can also be applied to <u>multi-dof per node</u>

Figure 1 : A Simple F.E. Mesh

$$[E]= \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

= Element-Node Connectivity Information

To save memory, the above matrix can be described by 2 integer arrays :

$\overrightarrow{IE}$ = 1^{st}, 5^{th}, 9^{th}, 13^{th}, 17^{th} = Locations (or pointers) of the first non-zero of each row

Dimension IE(NEL + 1)

#Elements

$\overrightarrow{JE}$ = $\boxed{3, 8, 1, 6}$; $\boxed{7, 3, 2, 4}$; $\boxed{5, 2, 3, 6}$; $\boxed{7, 9, 8, 3}$

Global dof for each element

$\overrightarrow{JE}$ = Element Node column index numbers (<u>UNORDER</u>) for each row

$\overrightarrow{JE}$ = Global dof (or node) numbers associate with each row (= element)

Dimension JE (NEL * NDOFPE)

#dof per element

Transposing the "Element-Node Connectivity" matrix will give:

$$[E^T] = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

= Node-Element Connectivity Information

To save memory, the above matrix can be described be the following 2 integer arrays:

$\overrightarrow{\text{IET}}$ = $1^{st}, 2^{nd}, 4^{th}, 8^{th}, 9^{th}, 10^{th}, 12^{th}, 14^{th}, 16^{th}, 17^{th}$

= Locations (or pointers) of the first nonzero of each row

Dimension IET(NDOF + 1) #Degree-of-freedom

$\overrightarrow{\text{JET}}$ = 1, 2, 3, 1, 2, 3, 4, 2, 3, 1, 3, 2, 4, 1, 4, 4

= Node-Element Column Index numbers (<u>ORDER</u>) for each row

$\overrightarrow{\text{JET}}$ = Element numbers associated with each dof

Dimension JET (NEL * NDOFPE)

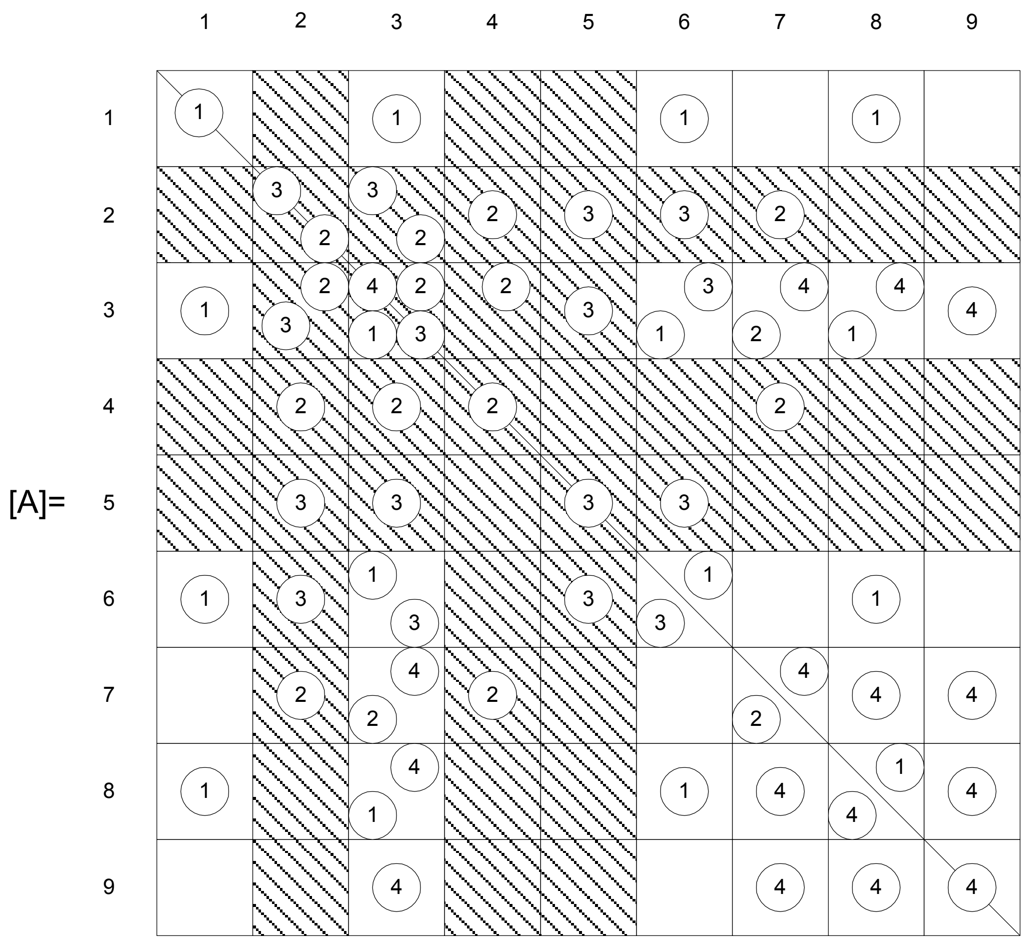


Figure 2 : Total (stiffness) Matrix

IA = 1, 4, 9, 15, 16, 17, 18, 20, 21, 21

JA = 3, 8, 6, 7, 3, 4, 5, 6, 8, 6, 7, 4, 5, 9, 7, 6, 8, 9, 8, 9

Could be unordered

How to impose (Dirichlet) boundary conditions

Assuming $[A]\vec{x} = \vec{b}$, ndof = 4 and with Dirichlet boundary conditions

$x_2 = k_2$ and $x_3 = k_3$

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ X_2 = k_2 \\ X_3 = k_3 \\ X_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} \Longleftrightarrow \begin{pmatrix} A_{11} & 0 & 0 & A_{14} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ A_{41} & 0 & 0 & A_{44} \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix} = \begin{pmatrix} b_1 - A_{12} \cdot k_2 - A_{13} \cdot k_3 \\ k_2 \\ k_3 \\ b_4 - A_{42} \cdot k_2 - A_{43} \cdot k_3 \end{pmatrix}$$

Note: After all x_i are found, reactions

$$R_2 = \sum_{j=1}^{4} A_{2j} \cdot X_j \qquad \text{and} \qquad R_3 = \sum_{j=1}^{4} A_{3j} \cdot X_j$$

$$
[A]=\begin{array}{cc}
 & \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\begin{array}{c} 1\\2\\3\\4\\5\\6\\7\\8\\9 \end{array} &
\begin{pmatrix}
X & 0 & X & 0 & 0 & X & 0 & X & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
X & 0 & X & 0 & 0 & X & X & X & X \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
X & 0 & X & 0 & 0 & X & 0 & X & 0 \\
0 & 0 & X & 0 & 0 & 0 & X & X & X \\
X & 0 & X & 0 & 0 & X & X & X & X \\
0 & 0 & X & 0 & 0 & 0 & X & X & X
\end{pmatrix}
\end{array}
$$

→ B.C.

→ A lot of elements contribute to this dof (i=3)

Figure 3 : Total (Stiffness) Matrix With Dirichlet Boundary Conditions @ dof = 2, 4 and 5

The above symmetrical matrix [A] can be described by the following 2 integer arrays :

{IA} (NDOF + 1) = 1^{st}, 4^{th}, 4^{th}, 8^{th}, 8^{th}, 8^{th}, 9^{th}, 11^{th}, 12^{th}, 12^{th}

= starting location of the first non-zero, off-diagonal terms for each row

{JA} (NCOEF1) = $\underline{3, 6, 8}$, 6, 7, 8, 9, $\underline{8}$, 8, 9, $\underline{9}$

= Column numbers associated with each non-zero off-diagonal terms for each row (could be unordered)

#of non-zero, off-diagonal terms of the original (before factorization) matrix

```
      subroutine spass1(ndof,nelpdof,nel,ndofpe,ieldof,ielconect
     $,nelpdofare)
      implicit real*8(a-h,o-z)
      dimension nelpdof(ndof),ieldof(ndofpe),ielconect(nel,ndofpe)
     $,nelpdofare(ndof,6)
c......Purposes:  To find HOW MANY, and WHICH elements are connected to each dof
c......           This code is stored under file name *symb*.f, in sub-directory
c......           ~/*odu*cl*/
c*******************************************************************************
c......Key Ideas:
c......Input-----> ndof,nel,ieldof(-)=global dof# associated with each element
c......Output ---> Get # elements associated with each dof, and stored in nelpdof(-)
c......       ---> Get the element numbers associated with each dof = nelpdofare(-)
c*******************************************************************************
c......Initialization
      do 11 i=1,ndof
 11   nelpdof(i)=0   ! # elements coonected to the i-th dof
c......
      do 12 i=1,nel
c......Get ndofpe (=4, for 2D truss) associated with each i-th element
c......For example:  ieldof(1)=3
c......              ieldof(2)=4
c......              ieldof(3)=7
c......              ieldof(4)=8
       do 14 j=1,ndofpe
 14    ieldof(j)=ielconect(i,j)
c
      do 13 j=1,ndofpe
      iglobdof=ieldof(j)
      nelpdof(iglobdof)=nelpdof(iglobdof)+1
      icount=nelpdof(iglobdof)
       if (icount .gt. 6) then
       write(6,*) 'ERROR: increase dimension nelpdofare(iglobdof,6)'
       write(6,*) ' assumed max # elements connect to a dof is only 6'
       stop
       else
       nelpdofare(iglobdof,icount)=i
       endif
 13   continue
 12   continue
c
      write(6,*) 'nelpdof= ',(nelpdof(i),i=1,ndof)
      do 21 i=1,ndof
      j=nelpdof(i)
      write(6,*) 'nelpdofare= ',i, (nelpdofare(i,k),k=1,j)
 21   continue
c
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine spass2(ncoff,ndof,tempok,nelpdof,nelpdofare,ndofpe
     $,ieldof,ielconect
     $,elk,offdiag,kptrs,k11indx,force,nel,diagk)
      implicit real*8(a-h,o-z)
      dimension tempok(ndof),nelpdof(ndof),nelpdofare(ndof,6)
     $,ielconect(nel,ndofpe),elk(ndofpe,ndofpe),offdiag(ncoff)
     $,k11indx(ncoff),force(ndof),ieldof(ndofpe),kptrs(ndof)
      dimension diagk(ndof)
```

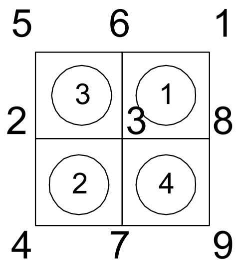
```
c......Purposes:  To do sparse stiffness assembly (according to NASA K* format), by doing
c......           1 row at a time
c......           This code is stored under file name *symb*.f, in sub-directory
c......           ~/*odu*cl*/
c*****************************************************************************************
c......Key Ideas:
c......             do 1 i 1,ndof
c......              do 2 j=1,nelpdof(i)
c......              j-th el = nelpdofare(i,j)
c......              get ieldof(ndofpe) = element global dof associated with j-th el
c......              get element stiffness = elk(ndofpe,ndofpe)
c......              contributions of element stiffness to "ONLY row #i" of total stiffness
c......               and store this information in a temporary array tempok(ndof)
c......               consider diagonal & upper triangular portion only)
c......          2   continue
c......
c......              do 5 j=i,ndof
c......              if (tempok(j) .ne. 0.) then
c......               get diagk(-), offdiag(-), kptrs(-), k11indx(-) in a row-by-row fashion
c......          5   continue
c......          1  continue
c++++++Note:  The user is required to properly initialize kptrs(-) array (including DIRICHLET
c++++++       boundary conditions) before calling these assembling routines
c*****************************************************************************************
      nzoffd=0
      do 1 i=1,ndof  ! (say, i=5-th row)
      nzpr=0
       do 6 j=1,ndof
 6     tempok(j)=0.
      do 2 j=1,nelpdof(i)
      jel=nelpdofare(i,j)
c......Get ndofpe (=4, for 2D truss) associated with each i-th element
       do 14 m=1,ndofpe
 14    ieldof(m)=ielconect(jel,m)
c......Get element stiffness matrix
      call elstif(jel,ndofpe,elk)
       do 3 k=1,ndofpe
       irowdof=ieldof(k)
       if(irowdof .ne. i) go to 3
        do 4 l=1,ndofpe
        icoldof=ieldof(l)
        if(icoldof .lt. i) go to 4    !  skip the lower half of stiffness matrix
        tempok(icoldof)=tempok(icoldof)+elk(k,l)
 4      continue
 3     continue
 2    continue
c
      diagk(i)=tempok(i)
      force(i)=diagk(i)
      do 5 j=i+1,ndof    !  Thus, this loop will be skipped for the last row i=ndof
      if( tempok(j) .eq. 0. ) go to 5   !  skip recording if Kij=0.
      nzoffd=nzoffd+1
      nzpr=nzpr+1
      offdiag(nzoffd)=tempok(j)
      kptrs(i)=nzpr
      k11indx(nzoffd)=j
 5    continue
 1    continue
      ncoef=nzoffd
c
      write(6,*) 'ncoef= ',ncoef
      write(6,*) 'kptrs= ',(kptrs(i),i=1,ndof)
      write(6,*) 'k11indx= ',(k11indx(i),i=1,ncoef)
      write(6,*) 'diagk= ',(diagk(i),i=1,ndof)
      write(6,*) 'offdiag= ',(offdiag(i),i=1,ncoef)
      write(6,*) 'force= ',(force(i),i=1,ndof)
c
      return
      end
```

```
      subroutine symbass(ie,je,iet,jet,n,ia,ja)
      implicit real*8(a-h,o-z)
      dimension ie(*),je(*),iet(*),jet(*),ia(*),ja(*)
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c......Purposes: symmetrical, sparse symbolic assembly
c......           This code is stored under file name *symb*.f, in sub-directory
c......           ~/*odu*cl*/
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c......NOTE:  This is a beta early version, please "do NOT" distribute
c......       this source code to general publics, as we have agreed !!
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c......Input: ie(nel+1)=locations (or pointers) of the first non-zero
c......       of each row (of element-dof connectivity info.)
c......        je(nel*ndofpe)=global dof column number for non-zero
c......        terms of each row (of element-dof connectivity info.)
c......       iet(ndof+1)=locations (pointers) of the first non-zero
c......       of each row (of dof-element connectivity info.)
c......        jet(nel*ndofpe)=locations (pointers) of the first non-zero
c......        of each row (of dof-element connectivity info.)
c......       ia(ndof)= ndof in the positions corespond to Dirichlet b.c.
c......                 0  elsewhere
c......Output:ia(ndof+1)=starting locations of the first non-zero
c......       off-diagonal terms for each row of structural stiffness
c......       matrix
c......        ja(ncoeff)=column numbers (unordered) corespond to
c......        each nonzero, off-diagonal term of each row of structural
c......        stiffness matrix
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      jp=1                            !001
      nm1=n-1                         !002
      do 30 i=1,nm1                   !003 last row (= eq) will be skipped
      jpi=jp                          !004 delayed counter for ia(-) array
      if ( ia(i) .eq. n ) go to 30    !005 skip row which corespond to Dirichlet b.c.
      ieta=iet(i)                     !006 begin index (to find how many elements)
      ietb=iet(i+1)-1                 !007 end index (to find how many elements)
        do 20 ip=ieta,ietb            !008 loop covering ALL elements attached to row i
        j=jet(ip)                     !009 actual "element number" attached to row i
        iea=ie(j)                     !010 begin index (to find how many nodes attached to
                                      !    element j)
        ieb=ie(j+1)-1                 !011 end index (to find how many nodes attached to
                                      !    element j)
          do 10 kp=iea,ieb            !012 loop covering ALL nodes attached to element j
          k=je(kp)                    !013 actual "node, or column number" attached to element
                                      !    j
          if (k .le. i) go to 10      !014 skip, if it involves with LOWER triangular portion
          if ( ia(k) .ge. i ) go to 10  !015 skip, if same node already been accounted by
                                      !    earlier elements
          ja(jp)=k                    !016 record "column number" associated with non-zero
                                      !    off-diag. term
          jp=jp+1                     !017 increase "counter" for column number array ja(-)
          ia(k)=i                     !018 record node (or column number) k already
                                      !    contributed to row i
 10       continue                    !019
 20     continue                      !020
 30   ia(i)=jpi                       !021 record "starting location" of non-zero off-diag.
                                      !    terms associated with row i
      ia(n)=jp                        !022 record "starting location" of non-zero term of LAST
                                      !    ROW
      ia(n+1)=jp                      !023 record "starting location" of non-zero term of LAST
                                      !   ROW + 1
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine numass(ia,ja,idir,ae,be,lm,ndofpe,an,ad,b,ip)
      implicit real*8(a-h,o-z)
      dimension ia(*),ja(*),idir(*),ae(*),be(*),lm(*),an(*)
      dimension ad(*),b(*),ip(*)
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c......Purposes: symmetrical, sparse numerical assembly
c......           This code is stored under file name *symb*.f, in sub-directory
```

```
c......          ~/*odu*cl*/
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c......NOTE:  This is a beta early version, please "do NOT" distribute
c......       this source code to general publics, as we have agreed !!
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c......Input: ia(ndof+1)=starting locations of the first non-zero
c......       off-diagonal terms for each row of structural stiffness
c......       matrix
c......        ja(ncoeff)=column numbers (unordered) corespond to
c......        each nonzero, off-diagonal term of each row of structural
c......        stiffness matrix
c......       idir(ndof)= 1 in the positions corespond to Dirichlet b.c.
c......                   0 elsewhere
c......         ae(ndofpe**2),be(ndofpe)= element (stiffness) matrix,
c......         and element (load) vector
c......       lm(ndofpe)= global dof associated with a finite element
c......         ndofpe= number of dof per element
c......       b(ndof)= before using this routine, values of b(-) should
c......       be initialized to:
c......       Ci, values of prescribed Dirichlet bc at proper locations
c......       or  values of applied nodal loads
c
c......Output:  an(ncoeff1)= values of nonzero, off-diagonal terms of
c......         structural stiffness matrix
c......       ad(ndof)= values off diagonal terms of structural stiffness
c......       matrix
c......         b(ndof)= right-hand-side (load) vector of system of linear
c......         equations
c......Temporary Arrays:
c......       ip(ndof)= intialized to 0
c......                then IP(-) is used and reset to 0
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      do 40 L=1,ndofpe                   !001 local "row" dof
      i=lm(L)                            !002 global"row" dof
      if ( idir(i) .ne. 0 ) go to 40     !003 skip, if DIRICHLET b.c.
      k=L-ndofpe                         !004 to find location of element k-diag
      ad(i)=ad(i)+ae(k+L*ndofpe)         !005 assemble K-diag
      b(i)=b(i)+be(L)                    !006 assemble element rhs load vector
      kk=0                               !007 flag, to skip contribution of entire
                                         !          global row i if all global col #
                                         !          j < i, or if entire row i belongs
                                         !          to LOWER triangle
        do 20 LL=1,ndofpe                !008 local "column" dof
        k=k+ndofpe                       !009 find location of element stiffness k
        if (LL .eq. L) go to 20          !010 skip, diag. term already taken care
        j=lm(LL)                         !011 global column dof
        if ( idir(j) .ne. 0 ) go to 10   !012 skip, if DIRICHLET b.c.
        if (j .lt. i) go to 20           !013 skip, if LOWER portion
        ip(j)=k                          !014 record global column # j (associated with
                                         !    global row # i) corespond to k-th term
                                         !    of element stiffness k
        kk=1                             !015 FLAG, indicate row L of [k] do have
                                         !    contribution to global row I of [K]
        go to 20                         !016
 10     b(i)=b(i)-b(j)*ae(k)             !017 modify rhs load vector due to DIRICHLET b.c.
 20     continue                         !018
      if (kk .eq. 0) go to 40            !019 skip indicator (see line 007)
      iaa=ia(i)                          !020 start index
      iab=ia(i+1)-1                      !021 end index
        do 30 j=iaa,iab                  !022 loop covering all col numbers associated
                                         !    with global row i
        k=ip( ja(j) )                    !023 ip ( col # ) already defined on line 014
                                         !                or initialized to ZERO initially
        if (k .eq. 0) go to 30           !024 skip
        an(j)=an(j)+ae(k)                !025 assemble [K] from [k]
        ip( ja(j) )=0                    !026 reset to ZERO for col # j before considering
                                         !    the next row L
 30     continue                         !027
 40   continue                           !028
c......print debugging results
      ndof=9
```

```
      ncoeff1=20

c      write(6,*) 'ia(-) array = ',(ia(i),i=1,ndof+1)
c      write(6,*) 'ja(-) array = ',(ja(i),i=1,ncoeff1)
c      write(6,*) 'ad(-) array = ',(ad(i),i=1,ndof)
c      write(6,*) 'an(i) array = ',(an(i),i=1,ncoeff1)
      return
      end
```

```
assembling methods: 1=slow; 2=fast
method= 1
 1  3  8  1  6
 2  7  3  2  4
 3  5  2  3  6
 4  7  9  8  3
nelpdof=  1  2  4  1  1  2  2  2  1
nelpdofare=  1  1
nelpdofare=  2  2  3
nelpdofare=  3  1  2  3  4
nelpdofare=  4  2
nelpdofare=  5  3
nelpdofare=  6  1  3
nelpdofare=  7  2  4
nelpdofare=  8  1  4
nelpdofare=  9  4
ncoef=  20
kptrs=  3  5  6  1  1  1  2  1  0
k11indx=  3  6  8  3  4  5  6  7  4  5  6  7  8  9  7  6  8  8  9  9
diagk= 6.0000000000000 24.0000000000000  60.0000000000000  16.0000000000000
    6.0000000000000  32.0000000000000  12.0000000000000  28.0000000000000
    16.0000000000000
offdiag=  4.0000000000000  7.0000000000000  5.0000000000000
    25.0000000000000  14.0000000000000  9.0000000000000  18.0000000000000
    8.0000000000000  12.0000000000000  12.0000000000000  26.0000000000000
    26.0000000000000  31.0000000000000  24.0000000000000  10.0000000000000
    15.0000000000000  6.0000000000000  16.0000000000000  12.0000000000000
    20.0000000000000
force=  6.0000000000000  24.0000000000000  60.0000000000000  16.0000000000000
    6.0000000000000  32.0000000000000  12.0000000000000  28.0000000000000
    16.0000000000000
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```



$$[k^{(e)}] = k_{i,j} = (i + j)*e$$

$$(1) \quad \rightarrow \quad \boxed{3, 8, 1, 6} \quad \rightarrow \quad [k^{(1)}] = \begin{pmatrix} 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \\ 4 & 5 & 6 & 7 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

$$(2) \quad \rightarrow \quad \boxed{7, 3, 2, 4} \quad \rightarrow \quad [k^{(2)}] = \begin{pmatrix} 4 & 6 & 8 & 10 \\ 6 & 8 & 10 & 12 \\ 8 & 10 & 12 & 14 \\ 10 & 12 & 14 & 16 \end{pmatrix}$$

$$(3) \quad \rightarrow \quad \boxed{5, 2, 3, 6} \quad \rightarrow \quad [k^{(3)}] = \begin{pmatrix} 6 & 9 & 12 & 15 \\ 9 & 12 & 15 & 18 \\ 12 & 15 & 18 & 21 \\ 15 & 18 & 21 & 24 \end{pmatrix}$$

$$(4) \quad \rightarrow \quad \boxed{7, 9, 8, 3} \quad \rightarrow \quad [k^{(4)}] = \begin{pmatrix} 8 & 12 & 16 & 20 \\ 12 & 16 & 20 & 24 \\ 16 & 20 & 24 & 28 \\ 20 & 24 & 28 & 32 \end{pmatrix}$$

K=	1	2	3	4	5	6	7	8	9
1	6		4			7		5	
2		12+12	25	14	9	18	8		
3			2+8 +18+32	12	12	5+21	26	31	24
4				16			10		
5					6	15			
6						8+24		6	
7							4+8	16	12
8								4+24	20
9									16

Hence :

$\overrightarrow{AD}$ = (6, 24, 60, 16, 6, 32, 12, 28, 16)

$\overrightarrow{AN}$ = $\boxed{4, 7, 5}$

$\boxed{25, 14, 9, 18, 8}$

$\boxed{12, 12, 26, 26, 31, 24}$

$\boxed{10}$

$\boxed{15}$

$\boxed{6}$

$\boxed{16, 12}$

$\boxed{20}$

```
method=   2
  1  3  8  1  6
  2  7  3  2  4
  3  5  2  3  6
  4  7  9  8  3
ia(-) array =   1  4  9  15  16  17  18  20  21  21
ja(-) array =   3  8  6  7  3  4  5  6  8  6  7  4  5  9  7  6  8  9  8  9
ad(-) array =     6.0000000000000    24.000000000000    60.000000000000
    16.000000000000    6.0000000000000    32.000000000000    12.000000000000
    28.000000000000    16.000000000000
an(i) array =     4.0000000000000    5.0000000000000    7.0000000000000
    8.0000000000000    25.000000000000    14.000000000000    9.0000000000000
    18.000000000000    31.000000000000    26.000000000000    26.000000000000
    12.000000000000    12.000000000000    24.000000000000   10.0000000000000
    15.000000000000    6.0000000000000    12.000000000000    16.000000000000
    20.000000000000
```

Unordered !!

Sparse Symbolic Matrix Assembly Algorithms

<u>Input</u> : $\overrightarrow{IE}$, $\overrightarrow{JE}$ Element-Node Connectivity Information

$\overrightarrow{IET}$, $\overrightarrow{JET}$ Node-Element Connectivity Information

N Number of Nodes (or dof) in a FE model

$$\overrightarrow{IA}(N) = \begin{cases} N, & \text{correspond to Dirichlet b.c. location} \\ O, & \text{elsewhere} \end{cases}$$

<u>Output</u> : $\overrightarrow{IA}$ (N + 1), NCOEF1

$\overrightarrow{JA}$ (NCOEF1) <u>is unorder</u>!

} Descriptions of Non-Zero location of matrix [A]

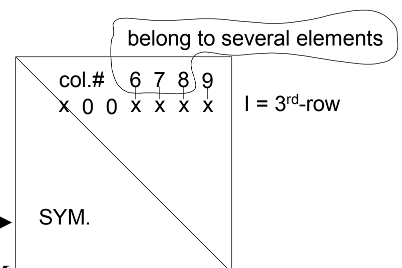
<u>Key Ideas</u>

Do 30 I = 1 → NM1 (= N – 1) for each I^{th} row of [A]
If (I^{th} row correspond to b.c.) Go to 30

Do 20 IP = consider all elements attached to I^{th} row (or dof)

Do 10 KP = consider all nodes (or dof) which belong to a particular element (in loop 20)

$K = f_1(KP) = \text{dof\#}$



*<u>Skip</u> assembling <u>Lower Triangle</u> of [A], due to SYM.

*Do <u>NOT</u> want to include the dof which already accounted for (from other elements) → Let IA (K) = I

*Record the column #K (on the I^{th} row) which has non-zero term → JA(JP) = K

*Increase the counter JP by 1 (for computing NCOEF1)

10 Continue

20 Continue

30 IA(I) = JPI

IA(N) = JP

IA(N + 1) = JP

Detailed Explanation of Sparse Symbolic Assembly routine

Line 01 : Initial pointer (or counter) for the column index array JA (JP) where JP = 1, 2, ….., NCOEF1. Later on, the value of JP will be kept increasing by 1 (see line 17)

Line 02 : We only need to consider the first (N – 1) rows, since in the last row (= row N), there is no non-zero off- diagonal term

*

*

*

Assuming $\overrightarrow{IA}$ and $\overrightarrow{JA}$ for the first row of matrix [A] has already been constructed. Now, we want to construct row #2 for $\overrightarrow{IA}$ and $\overrightarrow{JA}$

So far, we have

$$IA\begin{pmatrix} 1 \\ 3 \\ 8 \\ 6 \\ * \\ * \\ * \end{pmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{Bmatrix} \begin{array}{l} \\ \text{After Processing Row 1} \\ \\ \\ \\ \text{Initialized values} \\ \\ \end{array} \quad \text{and} \quad JA\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{Bmatrix} 3 \\ 8 \\ 6 \end{Bmatrix}$$

Now,

Do 30 I = 2
JPI = 4 (=JP)
IETA = 2
IETB = 3

Do 20 IP = 2, 3
J = 2 ➔ 3
IEA = 5 ➔ 9
IEB = 8 ➔ 12

Do 10 KP = 5, 8 ➔ 9, 12
K = 7→3→2→4 ➔ 5, 2, 3, 6

important

If (K•LE•I) Go to 10 → No→No→Yes→No➔No→Yes(skip)→No→No
➔If [IA(K)•GE•I] Go to 10 →No→No→No→No➔Yes, skip because col.# 3 of row 2 has already acct. by el.#2, hence col.#3 will <u>not</u> be counted again by el.#3

JA(4) = 7 , JA(5) = 3 , JA(6) = 4 → $JA\begin{pmatrix} 7 \\ 8 \end{pmatrix} = \begin{Bmatrix} 5 \\ 6 \end{Bmatrix}$

important

JP = 5, 6, 7→8, 9
➔IA(7) = 2 , IA(3) = 2 , IA(4) = 2 → $IA\begin{pmatrix} 5 \\ 6 \end{pmatrix} = \begin{Bmatrix} 2 \\ 2 \end{Bmatrix}$

10 Continue

20 Continue

30 IA(2) = 4

Hence, after finishing row I =2, we have

$$IA\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 = N+1 \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{array}{l} \\ \\ \\ \text{After processing row 2} \\ \\ \\ \\ \\ \text{Initialized values} \\ \\ \end{array} \qquad JA\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{pmatrix} = \begin{pmatrix} 3 \\ 8 \\ 6 \\ 7 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix} \begin{array}{l} \\ \text{for row 1} \\ \\ \\ \\ \text{for row 2} \\ \\ \\ \end{array}$$

Unorder col #