3D Code for Static Linear-Elastic Problems (2/3) Matrix Assembling

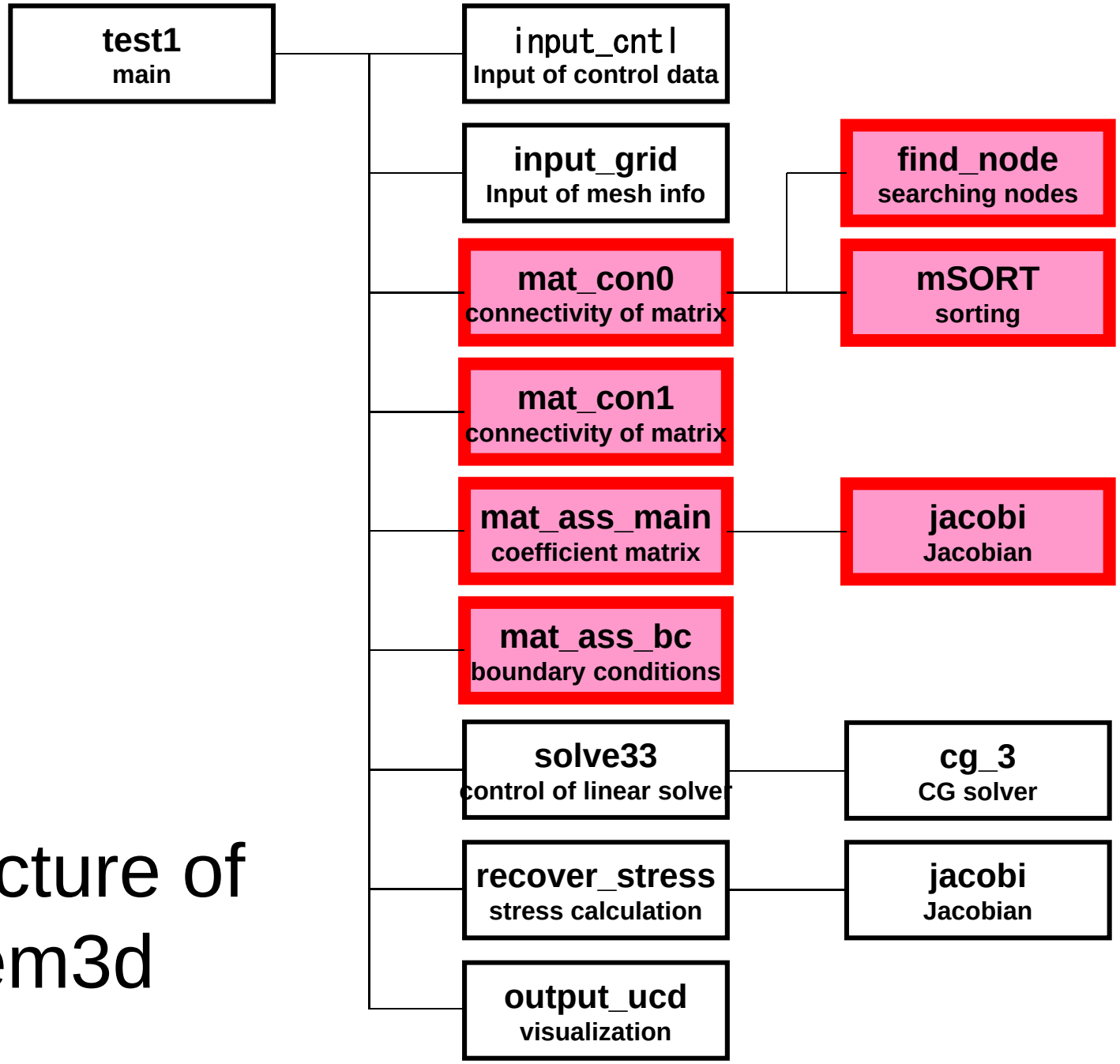
Kengo Nakajima
Information Technology Center

Technical & Scientific Computing I (4820-1027)
Seminar on Computer Science I (4810-1204)

FEM Procedures: Program

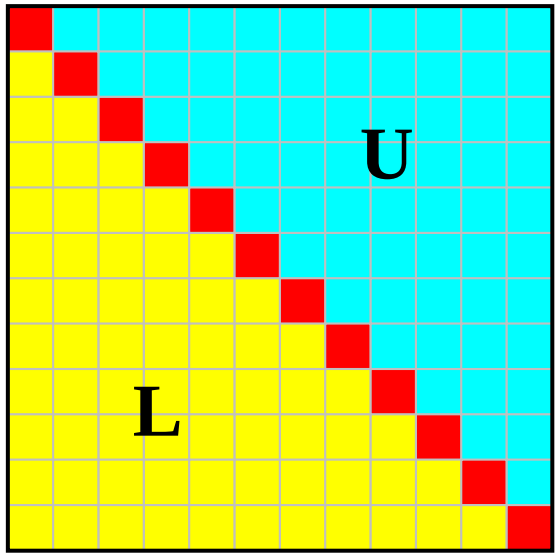
- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - Initialization of Arrays (Global/Element Matrices)
  - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
  - Element-by-Element Operations (do icel= 1, NE)
    - Element matrices
    - Accumulation to global matrix
  - Boundary Conditions
- Linear Solver
  - Conjugate Gradient Method
- Calculation of Stress

Structure of fem3d



Some Features of “fem3d”

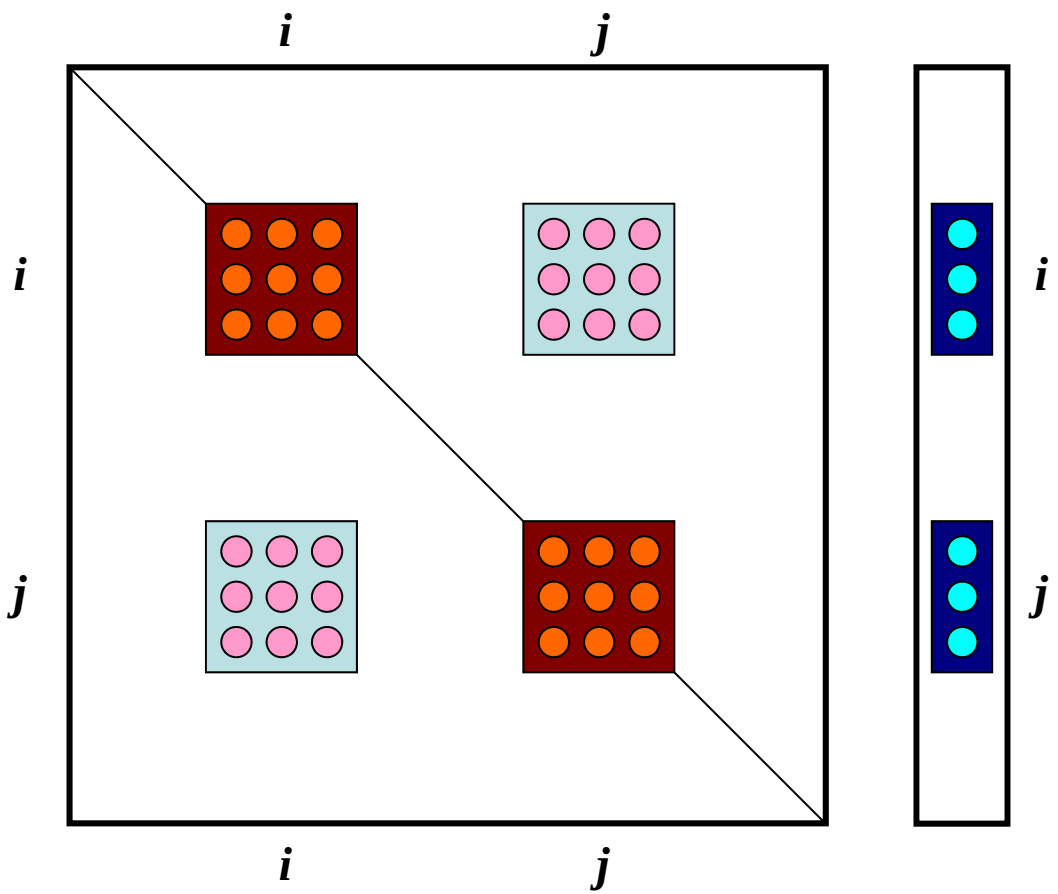
- Non-Zero Off-Diagonals
  - Upper/Lower triangular components are stored separately.



- Stored as Block
  - Vector: 3-components per node
  - Matrix: 9-components per block
  - Processed as “block” based on 3 variables on each node (not each component of matrix)

Storing 3x3 Block (1/3)

- Less memory requirement
  - Index, Item



Storing 3x3 Block (2/3)

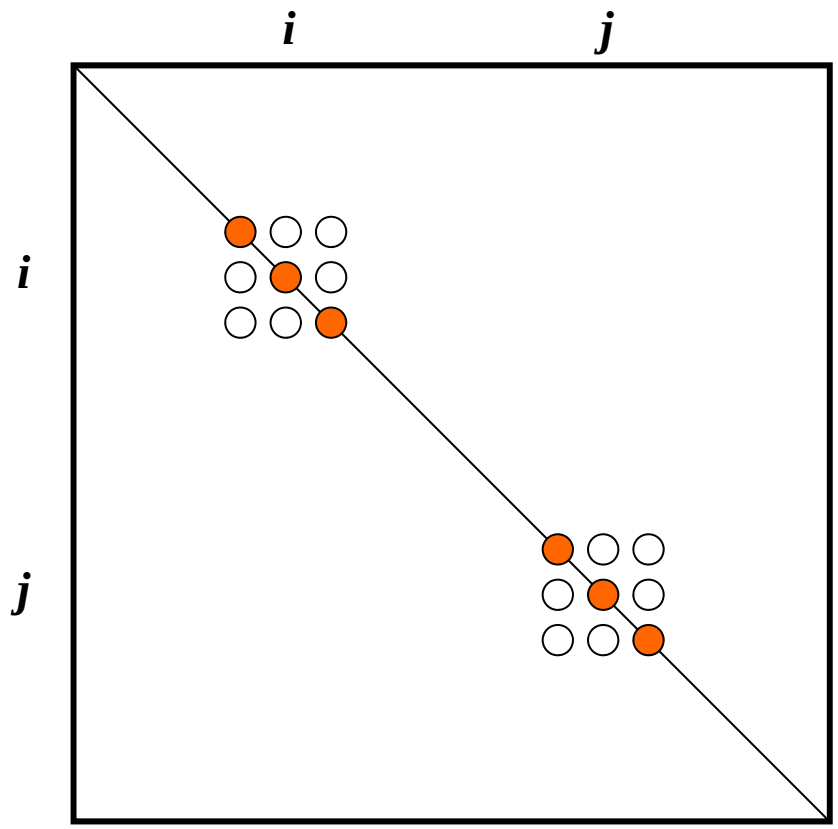
- Computational Efficiency
  - Ratio of (Comptation/Indirect Memory Access) is larger
  - >2x speed-up both for vector/scalar processors
    - Contiguous memory access, Cache Utilization, Larger Flop/Byte

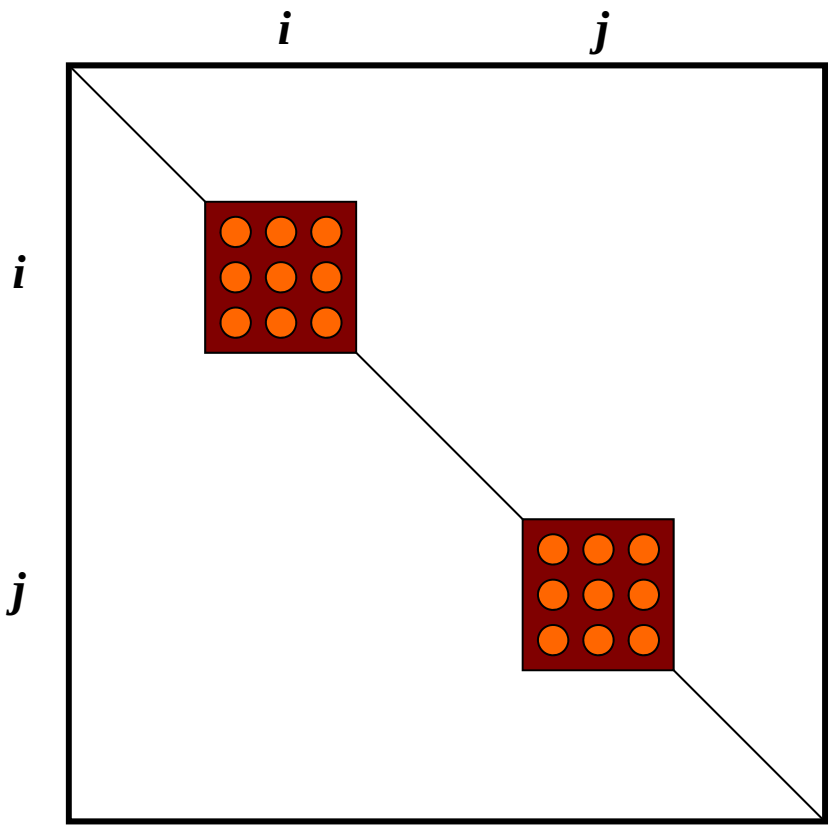
```
do i= 1, 3*N
  Y(i)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    kk= item(k)
    Y(i)= Y(i) + AMAT(k)*X(kk)
  enddo
enddo
```

```
do i= 1, N
  X1= X(3*i-2)
  X2= X(3*i-1)
  X3= X(3*i)
  Y(3*i-2)= D(9*i-8)*X1+D(9*i-7)*X2+D(9*i-6)*X3
  Y(3*i-1)= D(9*i-5)*X1+D(9*i-4)*X2+D(9*i-3)*X3
  Y(3*I  )= D(9*i-2)*X1+D(9*i-1)*X2+D(9*I  )*X3
  do k= index(i-1)+1, index(i)
    kk= item(k)
    X1= X(3*kk-2)
    X2= X(3*kk-1)
    X3= X(3*kk)
    Y(3*i-2)= Y(3*i-2)+AMAT(9*k-8)*X1+AMAT(9*k-7)*X2 &
                                     +AMAT(9*k-6)*X3
    Y(3*i-1)= Y(3*i-1)+AMAT(9*k-5)*X1+AMAT(9*k-4)*X2 &
                                     +AMAT(9*k-3)*X3
    Y(3*I  )= Y(3*I  )+AMAT(9*k-2)*X1+AMAT(9*k-1)*X2 &
                                     +AMAT(9*k  )*X3
  enddo
enddo
```

Storing 3x3 Block (3/3)

- Stabilization of Computation（計算の安定化）
  - Instead of division by diagonal components, full LU factorization of 3x3 Diagonal Block is applied.
  - Effective for ill-conditioned problems





Global Variables: pfem_util.h (1/3)

Name	Type	Size	I/O	Definition
fname	C	[80]	I	Name of mesh file
N,NP	I		I	# Node
ICELTOT	I		I	# Element
NODGRPtot	I		I	# Node Group
XYZ	R	[N][3]	I	Node Coordinates
ICELNOD	I	[ICELTOT][8]	I	Element Connectivity
NODGRP_INDEX	I	[NODGRPtot+1]	I	# Node in each Node Group
NODGRP_ITEM	I	[NODGRP_INDEX[NODGRPTOT+1]]	I	Node ID in each Node Group
NODGRP_NAME	C80	[NODGRP_INDEX[NODGRPTOT+1]]	I	Name of NodeGroup
NL, NU	I		O	# Upper/Lower Triangular Non-Zero Off-Diagonals at each node
NPL, NPU	I		O	# Upper/Lower Triangular Non-Zero Off-Diagonals
D	R	[9*N]	O	Diagonal Block of Global Matrix
B,X	R	[3*N]	O	RHS, Unknown Vector

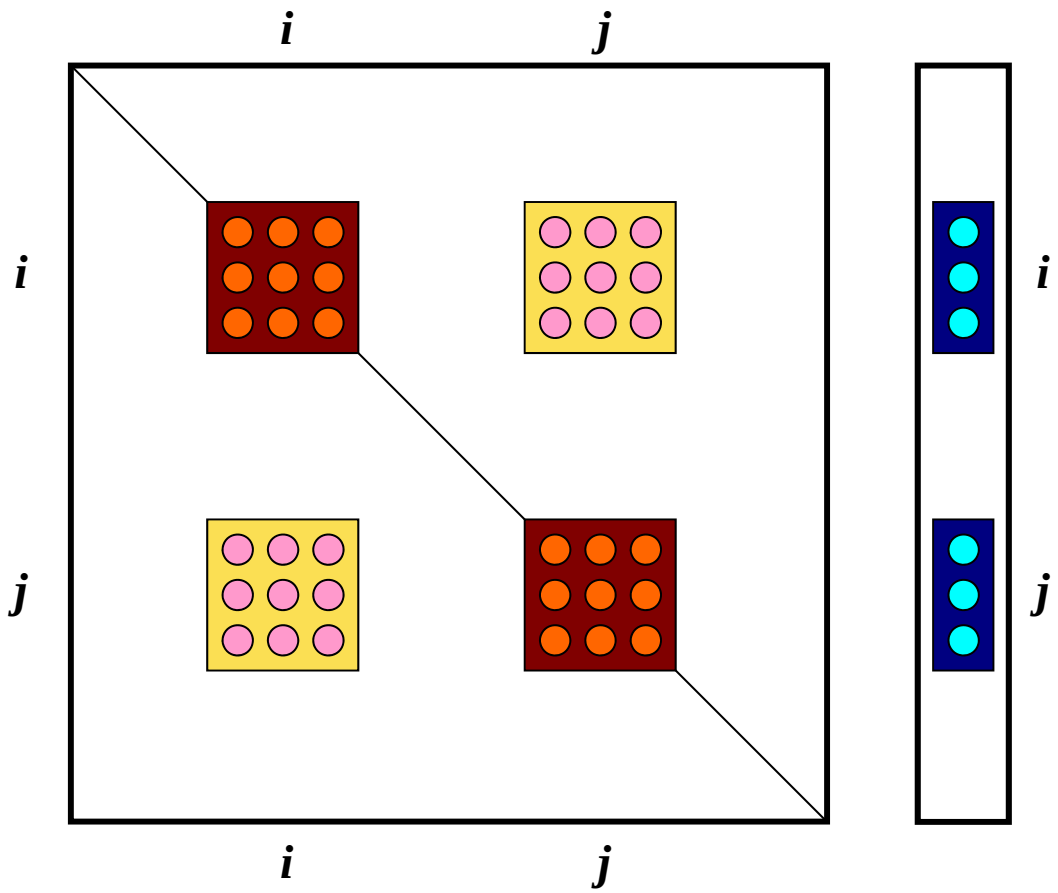
Global Variables: pfem_util.h (2/3)

Name	Type	Size	I/O	Definition
ALUG	R	[9*N]	O	Full LU factorization of Diagonal Blocks D
AL,AU	R	[9*NPL],[9*NPU]	O	Upper/Lower Triangular Components of Global Matrix
indexL, indexU	I	[N+1]	O	# Non-Zero Off-Diagonal Blocks
itemL, itemU	I	[NPL], [NPU]	O	Column ID of Non-Zero Off-Diagonal Blocks
INL, INU	I	[N]	O	Number of Off-Diagonal Blocks at Each Node
IAL, IAU	I	[N][NL],[N][NU]	O	Off-Diagonal Blocks at Each Node
IWKX	I	[N][2]	O	Work Arrays
METHOD	I		I	Iterative Method (fixed as 1)
PRECOND	I		I	Preconditioning Method (0: SSOR, 1: Block Diagonal Scaling)
ITER, ITERactual	I		I	Number of CG Iterations (MAX, Actual)
RESID	R		I	Convergence Criteria (fixed as 1.e-8)
SIGMA_DIAG	R		I	Coefficient for LU Factorization (fixed as 1.00)
pfemIarray	I	[100]	O	Integer Parameter Array
pfemRarray	R	[100]	O	Real Parameter Array

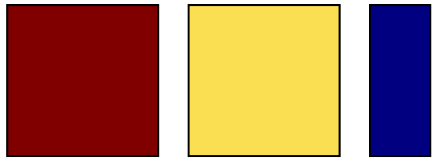
Global Variables: pfem_util.h (3/3)

Name	Type	Size	I/O	Definition
O8th	R		I	$= 0.125$
PNQ, PNE, PNT	R	[2][2][8]	O	$\frac{\partial N_i}{\partial \xi}, \frac{\partial N_i}{\partial \eta}, \frac{\partial N_i}{\partial \zeta} (i = 1 \sim 8)$ at each Gaussian Quad. Point
POS, WEI	R	[2]	O	Coordinates, Weighting Factor at each Gaussian Quad. Point
NCOL1, NCOL2	I	[100]	O	Work arrays for sorting
SHAPE	R	[2][2][2][8]	O	N_i (i=1~8) at each Gaussian Quad Point
PNX, PNY, PNZ	R	[2][2][2][8]	O	$\frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial z} (i = 1 \sim 8)$ at each Gaussian Quad. Point
DETJ	R	[2][2][2]	O	Determinant of Jacobian Matrix at each Gaussian Quad. Point
ELAST, POISSON	R		I	Young's Modulus, Poisson's Ratio
SIGMA_N, TAU_N	R	[N][3]	O	Normal/Shear Stress at each Node

Definitions of Terms



Block (Node):
ブロック（節点）:



Component (DOF):
成分（自由度）:

Towards Matrix Assembling

- In 1D, it was easy to obtain information related to <u>index</u> and <u>item</u>.
  - 2 non-zero off-diagonals for each node
  - ID of non-zero off-diagonal : i+1, i-1, where “i” is node ID
- In 3D, situation is more complicated:
  - Number of non-zero off-diagonal “blocks” is between 7 and 26 for the current target problem
  - More complicated for real problems.
  - Generally, there are no information related to number of non-zero off-diagonal “blocks” beforehand.

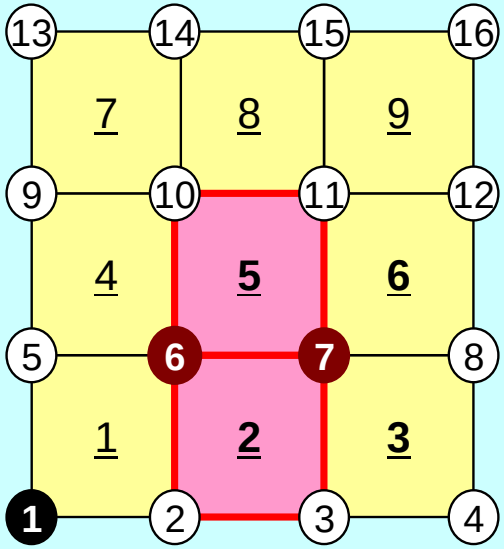
movie

Towards Matrix Assembling

- In 1D, it was easy to obtain information related to <u>index</u> and <u>item</u>.
  - 2 non-zero off-diagonals for each node
  - ID of non-zero off-diagonal : i+1, i-1, where “i” is node ID
- In 3D, situation is more complicated:
  - Number of non-zero off-diagonal “blocks” is between 7 and 26 for the current target problem
  - More complicated for real problems.
  - Generally, there are no information related to number of non-zero off-diagonal “blocks” beforehand.
- Count number of non-zero off-diagonals using arrays: INL[N], INU[N], IAL[N][NL], IAU[N][NU]

Main Part

```
#include <stdio.h>
#include <stdlib.h>
FILE* fp_log;
#define GLOBAL_VALUE_DEFINE
#include "pfem_util.h"

extern void INPUT_CNTL();
extern void INPUT_GRID();
extern void MAT_CON0();
extern void MAT_CON1();
extern void MAT_ASS_MAIN();
extern void MAT_ASS_BC();
extern void SOLVE33();
extern void RECOVER_STRESS();
extern void OUTPUT_UCD();
int main()
{

/** Logfile for debug **/
        if( (fp_log=fopen("log.log","w")) == NULL){
                fprintf(stdout,"input file cannot be opened!¥n");
                exit(1);
        }

        INPUT_CNTL();
        INPUT_GRID();


        MAT_CON0();
        MAT_CON1();

        MAT_ASS_MAIN();
        MAT_ASS_BC()   ;

        SOLVE33();

        RECOVER_STRESS();
        OUTPUT_UCD()    ;
}
```

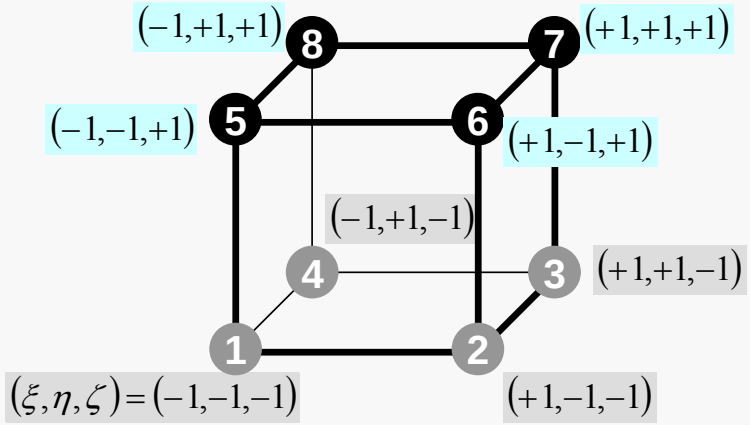


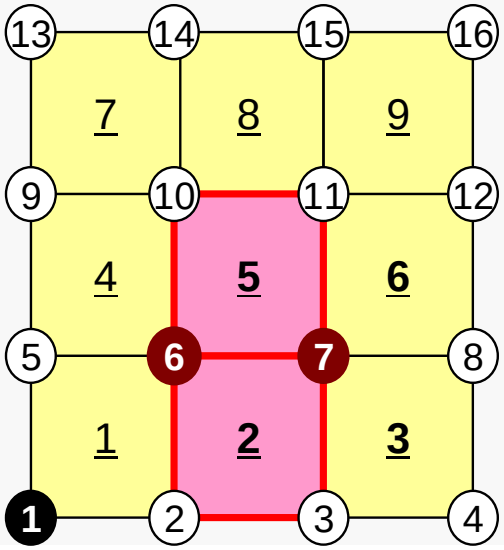
MAT_CON0: generates INL,INU, IAL, IAU
MAT_CON1: generates index, item

Node ID starting from “1”

MAT_CON0: Overview

```
do icel= 1, ICELTOT
  generate INL, INU, IAL, IAU
  according to 8 nodes of hexahedral element
  (FIND_NODE)
enddo
```





Generating Connectivity of Matrix MAT_CON0 (1/4)

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
extern void mSORT(int*, int*, int);
static void FIND_TS_NODE (int,int);

void MAT_CON0()
{
        int i,j,k,icel,in;
        int in1,in2,in3,in4,in5,in6,in7,in8;
        int NN;

        N2= 256: not in use
        NU= 26;
        NL= 26;

        INL=(KINT* )allocate_vector(sizeof(KINT),N);
        IAL=(KINT**)allocate_matrix(sizeof(KINT),N,NL);
        INU=(KINT* )allocate_vector(sizeof(KINT),N);
        IAU=(KINT**)allocate_matrix(sizeof(KINT),N,NU);

        for(i=0;i<N;i++) INL[i]=0;
        for(i=0;i<N;i++) for(j=0;j<NL;j++) IAL[i][j]=0;
        for(i=0;i<N;i++) INU[i]=0;
        for(i=0;i<N;i++) for(j=0;j<NU;j++) IAU[i][j]=0;
```

NU, NL:
Number of maximum number of connected nodes to each node (number of upper/lower non-zero off-diagonal blocks)

In the current problem, geometry is rather simple. Therefore we can specify NU and NL in this way.

If it's not clear -> implement that in Report #2.

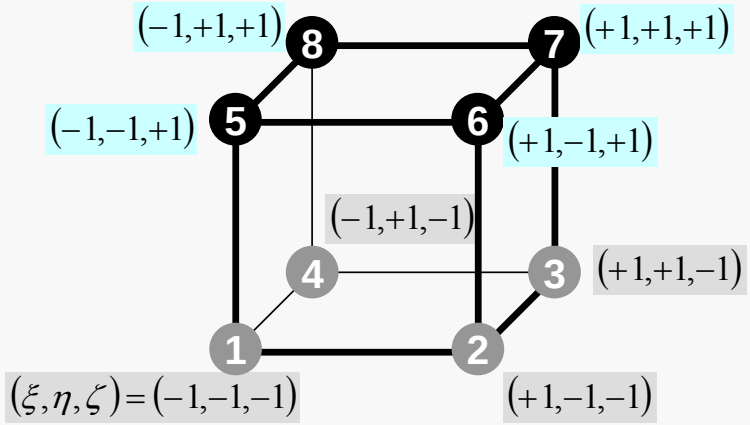
Generating Connectivity of Matrix MAT_CON0 (1/4)

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
extern void mSORT(int*, int*, int);
static void FIND_TS_NODE (int,int);

void MAT_CON0()
{
        int i,j,k,icel,in;
        int in1,in2,in3,in4,in5,in6,in7,in8;
        int NN;

        N2= 256: not in use
        NU= 26;
        NL= 26;

        INL=(KINT* )allocate_vector(sizeof(KINT),N);
        IAL=(KINT**)allocate_matrix(sizeof(KINT),N,NL);
        INU=(KINT* )allocate_vector(sizeof(KINT),N);
        IAU=(KINT**)allocate_matrix(sizeof(KINT),N,NU);

        for(i=0;i<N;i++) INL[i]=0;
        for(i=0;i<N;i++) for(j=0;j<NL;j++) IAL[i][j]=0;
        for(i=0;i<N;i++) INU[i]=0;
        for(i=0;i<N;i++) for(j=0;j<NU;j++) IAU[i][j]=0;
```

Array	Size	Description
INL, INU	[N]	Number of connected nodes to each node (lower/upper)
IAL, IAU	[N][NL], [N][NU]	Corresponding connected node ID (column ID)

Generating Connectivity of Matrix MAT_CON0 (2/4): Starting from 1

```
for( icel=0;icel< ICELTOT;icel++){
    in1=ICELNOD[icel][0];
    in2=ICELNOD[icel][1];
    in3=ICELNOD[icel][2];
    in4=ICELNOD[icel][3];
    in5=ICELNOD[icel][4];
    in6=ICELNOD[icel][5];
    in7=ICELNOD[icel][6];
    in8=ICELNOD[icel][7];

    FIND_TS_NODE (in1,in2);
    FIND_TS_NODE (in1,in3);
    FIND_TS_NODE (in1,in4);
    FIND_TS_NODE (in1,in5);
    FIND_TS_NODE (in1,in6);
    FIND_TS_NODE (in1,in7);
    FIND_TS_NODE (in1,in8);

    FIND_TS_NODE (in2,in1);
    FIND_TS_NODE (in2,in3);
    FIND_TS_NODE (in2,in4);
    FIND_TS_NODE (in2,in5);
    FIND_TS_NODE (in2,in6);
    FIND_TS_NODE (in2,in7);
    FIND_TS_NODE (in2,in8);

    FIND_TS_NODE (in3,in1);
    FIND_TS_NODE (in3,in2);
    FIND_TS_NODE (in3,in4);
    FIND_TS_NODE (in3,in5);
    FIND_TS_NODE (in3,in6);
    FIND_TS_NODE (in3,in7);
    FIND_TS_NODE (in3,in8);
```



FIND_TS_NODE: Search Connectivity

INL,INU,IAL,IAU: Automatic Search

```
static void FIND_TS_NODE (int ip1,int ip2)

{
        int kk,icou;
        if( ip1 > ip2 ){
                for(kk=1;kk<=INL[ip1-1];kk++){
                        if(ip2 == IAL[ip1-1][kk-1]) return;
                }
                icou=INL[ip1-1]+1;
                IAL[ip1-1][icou-1]=ip2;
                INL[ip1-1]=icou;
                return;
        }

        if( ip2 > ip1 ) {
                for(kk=1;kk<=INU[ip1-1];kk++){
                        if(ip2 == IAU[ip1-1][kk-1]) return;
                }
                icou=INU[ip1-1]+1;
                IAU[ip1-1][icou-1]=ip2;
                INU[ip1-1]=icou;
                return;
        }
}
```

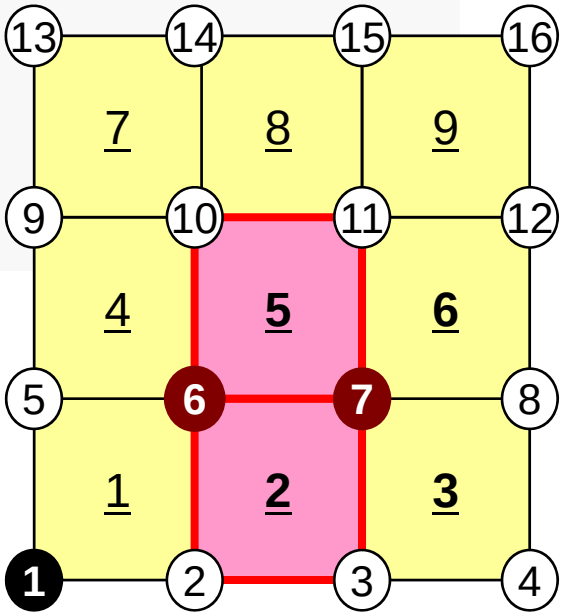
Array	Size	Description
INL, INU	[N]	Number of connected nodes to each node (lower/upper)
IAL, IAU	[N][NL], [N][NU]	Corresponding connected node ID (column ID)

FIND_TS_NODE: Search Connectivity

INL,INU,IAL,IAU: Automatic Search

```
static void FIND_TS_NODE (int ip1,int ip2)

{
        int kk,icou;
        if( ip1 > ip2 ){
                for(kk=1;kk<=INL[ip1-1];kk++){
                        if(ip2 == IAL[ip1-1][kk-1]) return;
                }
                icou=INL[ip1-1]+1;
                IAL[ip1-1][icou-1]=ip2;
                INL[ip1-1]=icou;
                return;
        }

        if( ip2 > ip1 ) {
                for(kk=1;kk<=INU[ip1-1];kk++){
                        if(ip2 == IAU[ip1-1][kk-1]) return;
                }
                icou=INU[ip1-1]+1;
                IAU[ip1-1][icou-1]=ip2;
                INU[ip1-1]=icou;
                return;
        }
}
```

If the target node is already included in IAL, IAU, proceed to next pair of nodes

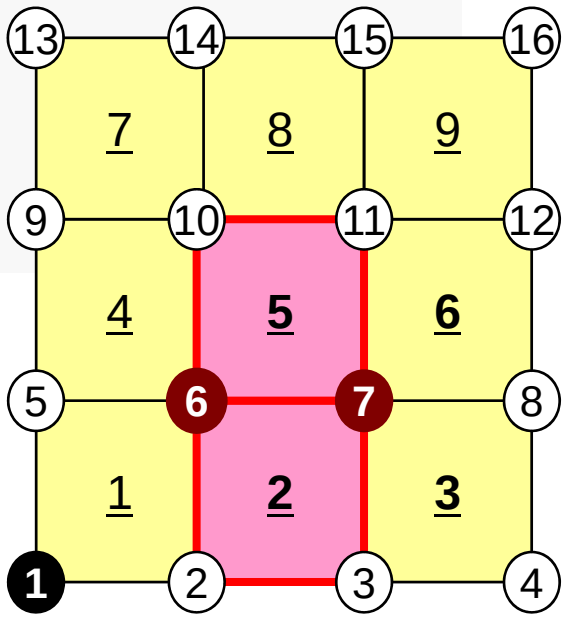


FIND_TS_NODE: Search Connectivity

INL,INU,IAL,IAU: Automatic Search

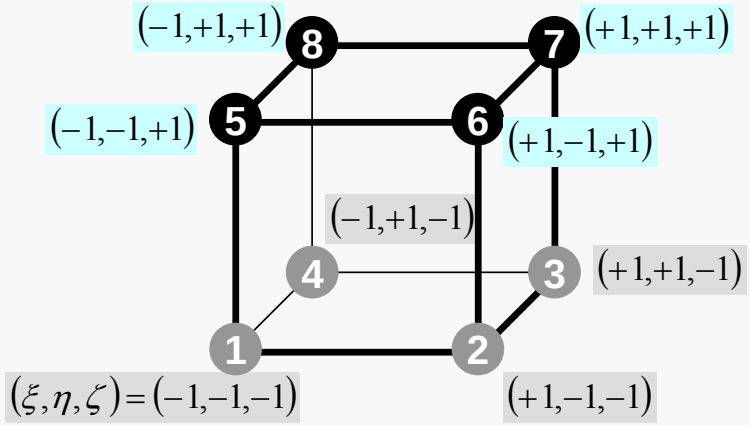
```
static void FIND_TS_NODE (int ip1,int ip2)

{
        int kk,icou;
        if( ip1 > ip2 ){
                for(kk=1;kk<=INL[ip1-1];kk++){
                        if(ip2 == IAL[ip1-1][kk-1]) return;
                }
                icou=INL[ip1-1]+1;
                IAL[ip1-1][icou-1]=ip2;
                INL[ip1-1]=icou;
                return;
        }

        if( ip2 > ip1 ) {
                for(kk=1;kk<=INU[ip1-1];kk++){
                        if(ip2 == IAU[ip1-1][kk-1]) return;
                }
                icou=INU[ip1-1]+1;
                IAU[ip1-1][icou-1]=ip2;
                INU[ip1-1]=icou;
                return;
        }
}
```

If the target node is NOT included in IAL, IAU, store the node in IAL/IAU, and add 1 to INL/INU.



Generating Connectivity of Matrix MAT_CON0 (3/4)

```
FIND_TS_NODE (in4,in1);
FIND_TS_NODE (in4,in2);
FIND_TS_NODE (in4,in3);
FIND_TS_NODE (in4,in5);
FIND_TS_NODE (in4,in6);
FIND_TS_NODE (in4,in7);
FIND_TS_NODE (in4,in8);

FIND_TS_NODE (in5,in1);
FIND_TS_NODE (in5,in2);
FIND_TS_NODE (in5,in3);
FIND_TS_NODE (in5,in4);
FIND_TS_NODE (in5,in6);
FIND_TS_NODE (in5,in7);
FIND_TS_NODE (in5,in8);

FIND_TS_NODE (in6,in1);
FIND_TS_NODE (in6,in2);
FIND_TS_NODE (in6,in3);
FIND_TS_NODE (in6,in4);
FIND_TS_NODE (in6,in5);
FIND_TS_NODE (in6,in7);
FIND_TS_NODE (in6,in8);

FIND_TS_NODE (in7,in1);
FIND_TS_NODE (in7,in2);
FIND_TS_NODE (in7,in3);
FIND_TS_NODE (in7,in4);
FIND_TS_NODE (in7,in5);
FIND_TS_NODE (in7,in6);
FIND_TS_NODE (in7,in8);
```



Generating Connectivity of Matrix MAT_CON0 (4/4)

```
            FIND_TS_NODE (in8,in1);
            FIND_TS_NODE (in8,in2);
            FIND_TS_NODE (in8,in3);
            FIND_TS_NODE (in8,in4);
            FIND_TS_NODE (in8,in5);
            FIND_TS_NODE (in8,in6);
            FIND_TS_NODE (in8,in7);
        }

        for(in=0;in<N;in++){
                NN=INL[in];
                for  (k=0;k<NN;k++){
                        NCOL1[k]=IAL[in][k];
                }
                mSORT(NCOL1,NCOL2,NN);
                for(k=NN;k>0;k--){
                        IAL[in][NN-k]= NCOL1[NCOL2[k-1]-1];
                }

                NN=INU[in];
                for  (k=0;k<NN;k++){
                        NCOL1[k]=IAU[in][k];
                }
                mSORT(NCOL1,NCOL2,NN);
                for(k=NN;k>0;k--){
                        IAU[in][NN-k]= NCOL1[NCOL2[k-1]-1];
                }
        }

}
```

Sort IAL[i][k], IAU[i][k] in ascending order by “bubble” sorting for less than 100 components.

MAT_CON1: CRS format

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1()
{       int i,k,kk;

        indexL=(KINT*)allocate_vector(sizeof(KINT),N+1);
        indexU=(KINT*)allocate_vector(sizeof(KINT),N+1);
        for(i=0;i<N+1;i++) indexL[i]=0;
        for(i=0;i<N+1;i++) indexU[i]=0;

        for(i=0;i<N;i++){
                indexL[i+1]=indexL[i]+INL[i];
                indexU[i+1]=indexU[i]+INU[i];
        }
        NPL=indexL[N];
        NPU=indexU[N];

        itemL=(KINT*)allocate_vector(sizeof(KINT),NPL);
        itemU=(KINT*)allocate_vector(sizeof(KINT),NPU);

        for(i=0;i<N;i++){
                for(k=0;k<INL[i];k++){
                        kk=k+indexL[i];
                        itemL[kk]=IAL[i][k]-1;
                }
                for(k=0;k<INU[i];k++){
                        kk=k+indexU[i];
                        itemU[kk]=IAU[i][k]-1;
                }
        }
        deallocate_vector(INL);
        deallocate_vector(INU);
        deallocate_vector(IAL);
        deallocate_vector(IAU);
}
```

C

$$\mathrm{indexL}[i+1]=\sum_{k=0}^{i}\mathrm{INL}[k]$$

$$\mathrm{indexU}[i+1]=\sum_{k=0}^{i}\mathrm{INU}[k]$$

$$\mathrm{indexL}[0]=\mathrm{indexU}[0]=0$$

FORTRAN

$$\mathrm{indexL}[i]=\sum_{k=1}^{i}\mathrm{INL}[k]$$

$$\mathrm{indexU}[i]=\sum_{k=1}^{i}\mathrm{INU}[k]$$

$$\mathrm{indexL}[0]=\mathrm{indexU}[0]=0$$

MAT_CON1: CRS format

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1()
{       int i,k,kk;

        indexL=(KINT*)allocate_vector(sizeof(KINT),N+1);
        indexU=(KINT*)allocate_vector(sizeof(KINT),N+1);
        for(i=0;i<N+1;i++) indexL[i]=0;
        for(i=0;i<N+1;i++) indexU[i]=0;

        for(i=0;i<N;i++){
                indexL[i+1]=indexL[i]+INL[i];
                indexU[i+1]=indexU[i]+INU[i];
        }
        NPL=indexL[N];
        NPU=indexU[N];

        itemL=(KINT*)allocate_vector(sizeof(KINT),NPL);
        itemU=(KINT*)allocate_vector(sizeof(KINT),NPU);

        for(i=0;i<N;i++){
                for(k=0;k<INL[i];k++){
                        kk=k+indexL[i];
                        itemL[kk]=IAL[i][k]-1;
                }
                for(k=0;k<INU[i];k++){
                        kk=k+indexU[i];
                        itemU[kk]=IAU[i][k]-1;
                }
        }
        deallocate_vector(INL);
        deallocate_vector(INU);
        deallocate_vector(IAL);
        deallocate_vector(IAU);
}
```

NPL=indexL[N]
Size of array: itemL
Total number of lower non-zero off-diagonal blocks

NPU=indexU[N]
Size of array: itemU
Total number of upper non-zero off-diagonal blocks

MAT_CON1: CRS format

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1()
{       int i,k,kk;

        indexL=(KINT*)allocate_vector(sizeof(KINT),N+1);
        indexU=(KINT*)allocate_vector(sizeof(KINT),N+1);
        for(i=0;i<N+1;i++) indexL[i]=0;
        for(i=0;i<N+1;i++) indexU[i]=0;

        for(i=0;i<N;i++){
                indexL[i+1]=indexL[i]+INL[i];
                indexU[i+1]=indexU[i]+INU[i];
        }
        NPL=indexL[N];
        NPU=indexU[N];

        itemL=(KINT*)allocate_vector(sizeof(KINT),NPL);
        itemU=(KINT*)allocate_vector(sizeof(KINT),NPU);

        for(i=0;i<N;i++){
                for(k=0;k<INL[i];k++){
                        kk=k+indexL[i];
                        itemL[kk]=IAL[i][k]-1;
                }
                for(k=0;k<INU[i];k++){
                        kk=k+indexU[i];
                        itemU[kk]=IAU[i][k]-1;
                }
        }
        deallocate_vector(INL);
        deallocate_vector(INU);
        deallocate_vector(IAL);
        deallocate_vector(IAU);
}
```

itemL, itemU
store node ID starting from 0

MAT_CON1: CRS format

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1()
{       int i,k,kk;

        indexL=(KINT*)allocate_vector(sizeof(KINT),N+1);
        indexU=(KINT*)allocate_vector(sizeof(KINT),N+1);
        for(i=0;i<N+1;i++) indexL[i]=0;
        for(i=0;i<N+1;i++) indexU[i]=0;

        for(i=0;i<N;i++){
                indexL[i+1]=indexL[i]+INL[i];
                indexU[i+1]=indexU[i]+INU[i];
        }
        NPL=indexL[N];
        NPU=indexU[N];

        itemL=(KINT*)allocate_vector(sizeof(KINT),NPL);
        itemU=(KINT*)allocate_vector(sizeof(KINT),NPU);

        for(i=0;i<N;i++){
                for(k=0;k<INL[i];k++){
                        kk=k+indexL[i];
                        itemL[kk]=IAL[i][k]-1;
                }
                for(k=0;k<INU[i];k++){
                        kk=k+indexU[i];
                        itemU[kk]=IAU[i][k]-1;
                }
        }
        deallocate_vector(INL);
        deallocate_vector(INU);
        deallocate_vector(IAL);
        deallocate_vector(IAU);
}
```

Not required any more

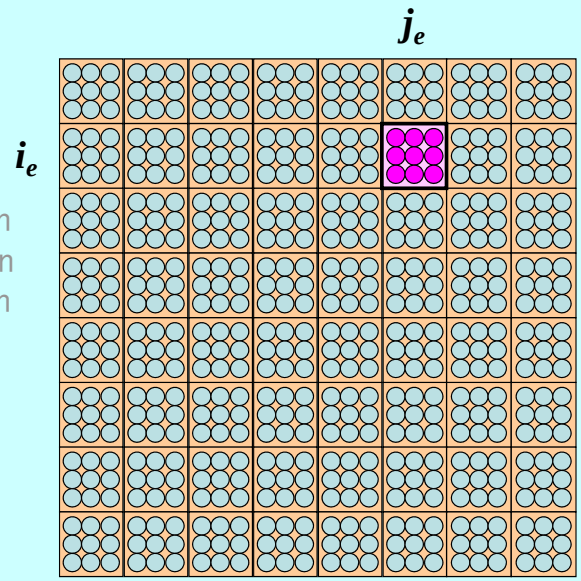
Main Part

```
#include <stdio.h>
#include <stdlib.h>
FILE* fp_log;
#define GLOBAL_VALUE_DEFINE
#include "pfem_util.h"

extern void INPUT_CNTL();
extern void INPUT_GRID();
extern void MAT_CON0();
extern void MAT_CON1();
extern void MAT_ASS_MAIN();
extern void MAT_ASS_BC();
extern void SOLVE33();
extern void RECOVER_STRESS();
extern void OUTPUT_UCD();
int main()
{

/** Logfile for debug **/
        if( (fp_log=fopen("log.log","w")) == NULL){
                fprintf(stdout,"input file cannot be opened!¥n");
                exit(1);
        }

        INPUT_CNTL();
        INPUT_GRID();


        MAT_CON0();
        MAT_CON1();

        MAT_ASS_MAIN();
        MAT_ASS_BC()  ;

        SOLVE33();

        RECOVER_STRESS();
        OUTPUT_UCD()    ;
}
```

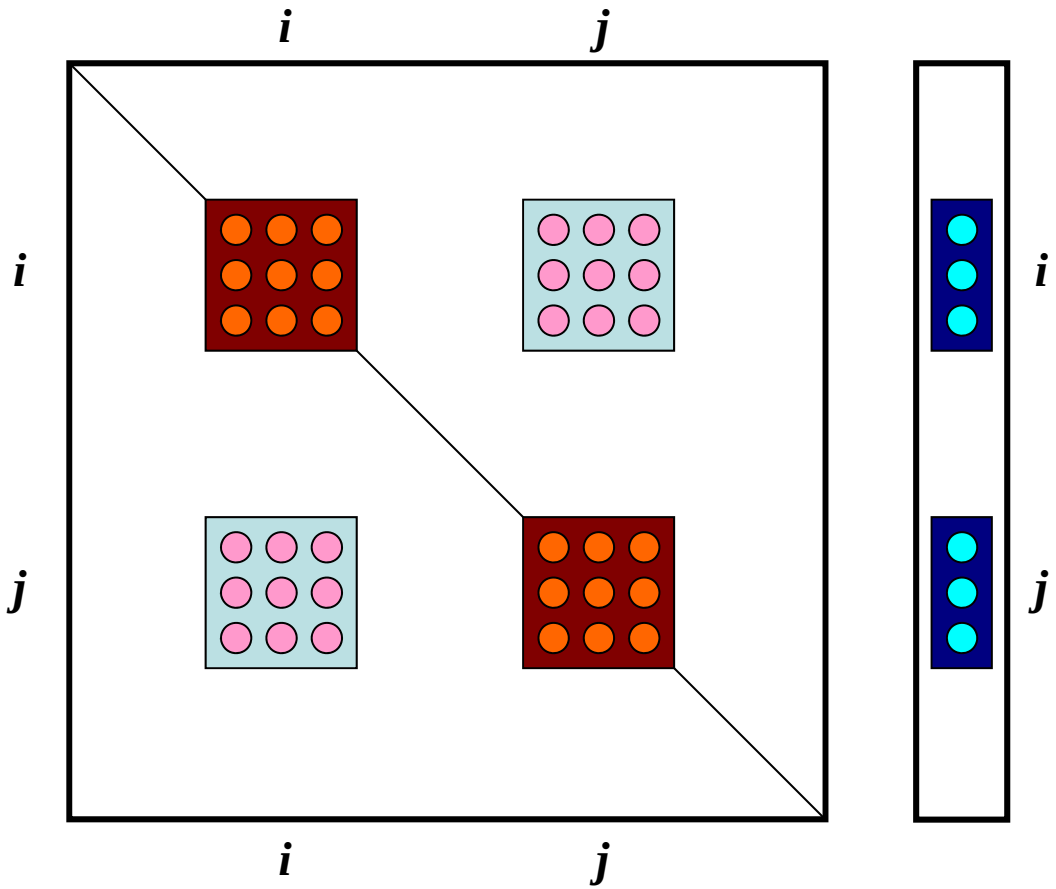
MAT_ASS_MAIN: Overview

```
do kpn= 1, 2          Gaussian Quad. points in ζ-direction
  do jpn= 1, 2        Gaussian Quad. points in η-direction
    do ipn= 1, 2      Gaussian Quad. Pointe in ξ-direction
      Define Shape Function at Gaussian Quad. Points (8-points)
      Its derivative on natural/local coordinate is also defined.
    enddo
  enddo
enddo
```

```
do icel= 1, ICELTOT    Loop for Element
  Jacobian and derivative on global coordinate of shape functions at
  Gaussian Quad. Points are defined according to coordinates of 8 nodes. (JACOBI)

  do ie= 1, 8          Local Node ID
    do je= 1, 8        Local Node ID
      Global Node ID: ip, jp
      Address of A_ip, jp in "itemL, itemU" : kk

      do kpn= 1, 2       Gaussian Quad. points in ζ-direction
        do jpn= 1, 2     Gaussian Quad. points in η-direction
          do ipn= 1, 2   Gaussian Quad. points in ξ-direction
            integration on each element
            coefficients of element matrices
            accumulation to global matrix
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```



Storing 3x3 Block



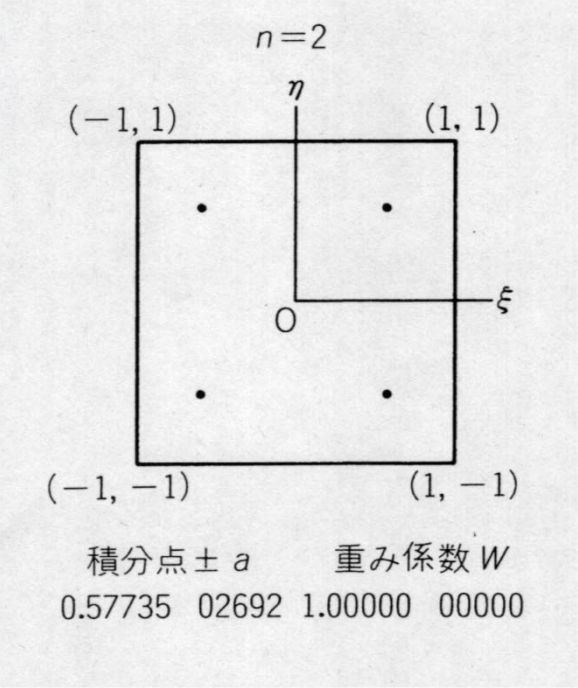
MAT_ASS_MAIN (1/7)

```
#include <stdio.h>
#include <math.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
extern void JACOBI();
void MAT_ASS_MAIN()
{
            int i,k,kk;
            int ip,jp,kp;
            int ipn,jpn,kpn;
            int icel;
            int ie,je;
            int iiS,iiE;
            int in1,in2,in3,in4,in5,in6,in7,in8;
            double valA,valB,valX;
            double QP1,QM1,EP1,EM1,TP1,TM1;
            double X1,X2,X3,X4,X5,X6,X7,X8;
            double Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8;
            double Z1,Z2,Z3,Z4,Z5,Z6,Z7,Z8;
            double PNXi,PNYi,PNZi,PNXj,PNYj,PNZj;
            double VOL;
            double coef;
            double a11,a12,a13,a21,a22,a23,a31,a32,a33;

            KINT nodLOCAL[8];

            AL=(KREAL*) allocate_vector(sizeof(KREAL),9*NPL);          Lower Triangle Blocks
            AU=(KREAL*) allocate_vector(sizeof(KREAL),9*NPU);          Upper
            B =(KREAL*) allocate_vector(sizeof(KREAL),3*N );           RHS
            D =(KREAL*) allocate_vector(sizeof(KREAL),9*N);            Diagonal Blocks
            X =(KREAL*) allocate_vector(sizeof(KREAL),3*N);            Unkonws

            for(i=0;i<9*NPL;i++) AL[i]=0.0;
            for(i=0;i<9*NPU;i++) AU[i]=0.0;
            for(i=0;i<3*N ;i++) B[i]=0.0;
            for(i=0;i<9*N ;i++) D[i]=0.0;
            for(i=0;i<3*N ;i++) X[i]=0.0;
```

MAT ASS MAIN (2/7)

```
            WEI[0]= 1.0000000000e0;
            WEI[1]= 1.0000000000e0;

            POS[0]= -0.5773502692e0;
            POS[1]=  0.5773502692e0;

/***
            INIT.
            PNQ   - 1st-order derivative of shape function by QSI
            PNE   - 1st-order derivative of shape function by ETA
            PNT   - 1st-order derivative of shape function by ZET
***/
            valA=            POISSON /      (1.e0-POISSON);
            valB= (1.e0-2.e0*POISSON)/(2.e0*(1.e0-POISSON));
            valX= ELAST*(1.e0-POISSON)/((1.e0+POISSON)*(1.e0-2.e0*POISSON));

            valA= valA * valX;
            valB= valB * valX;

            for(ip=0;ip<2;ip++){
                for(jp=0;jp<2;jp++){
                    for(kp=0;kp<2;kp++){
                        QP1= 1.e0 + POS[ip];
                        QM1= 1.e0 - POS[ip];
                        EP1= 1.e0 + POS[jp];
                        EM1= 1.e0 - POS[jp];
                        TP1= 1.e0 + POS[kp];
                        TM1= 1.e0 - POS[kp];

                        SHAPE[ip][jp][kp][0]= O8th * QM1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][1]= O8th * QP1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][2]= O8th * QP1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][3]= O8th * QM1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][4]= O8th * QM1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][5]= O8th * QP1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][6]= O8th * QP1 * EP1 * TP1;
                        SHAPE[ip][jp][kp][7]= O8th * QM1 * EP1 * TP1;
```

POS: Quad. Point
WEI: Weighting Factor



MAT ASS MAIN (2/7)

```
            WEI[0]= 1.0000000000e0;
            WEI[1]= 1.0000000000e0;

            POS[0]= -0.5773502692e0;
            POS[1]=  0.5773502692e0;

/***
            INIT.
            PNQ   - 1st-order derivative of shape function by QSI
            PNE   - 1st-order derivative of shape function by ETA
            PNT   - 1st-order derivative of shape function by ZET
***/
            valA=            POISSON /      (1.e0-POISSON);
            valB= (1.e0-2.e0*POISSON)/(2.e0*(1.e0-POISSON));
            valX= ELAST*(1.e0-POISSON)/((1.e0+POISSON)*(1.e0-2.e0*POISSON));

            valA= valA * valX;
            valB= valB * valX;

            for(ip=0;ip<2;ip++){
                for(jp=0;jp<2;jp++){
                    for(kp=0;kp<2;kp++){
                        QP1= 1.e0 + POS[ip];
                        QM1= 1.e0 - POS[ip];
                        EP1= 1.e0 + POS[jp];
                        EM1= 1.e0 - POS[jp];
                        TP1= 1.e0 + POS[kp];
                        TM1= 1.e0 - POS[kp];

                        SHAPE[ip][jp][kp][0]= O8th * QM1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][1]= O8th * QP1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][2]= O8th * QP1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][3]= O8th * QM1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][4]= O8th * QM1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][5]= O8th * QP1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][6]= O8th * QP1 * EP1 * TP1;
                        SHAPE[ip][jp][kp][7]= O8th * QM1 * EP1 * TP1;
```

Strain-Stress Relationship

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{Bmatrix} = \underbrace{\boxed{\frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1-2\nu) & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}(1-2\nu) & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}(1-2\nu) \end{bmatrix}}}_{[D]} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{Bmatrix}$$

$$\{\sigma\} = [D]\{\varepsilon\}$$

MAT ASS MAIN (2/7)

```
            WEI[0]= 1.0000000000e0;
            WEI[1]= 1.0000000000e0;

            POS[0]= -0.5773502692e0;
            POS[1]=  0.5773502692e0;

/***
            INIT.
            PNQ   - 1st-order derivative of shape function by QSI
            PNE   - 1st-order derivative of shape function by ETA
            PNT   - 1st-order derivative of shape function by ZET
***/
            valA=            POISSON /      (1.e0-POISSON);
            valB= (1.e0-2.e0*POISSON)/(2.e0*(1.e0-POISSON));
            valX= ELAST*(1.e0-POISSON)/((1.e0+POISSON)*(1.e0-2.e0*POISSON));

            valA= valA * valX;
            valB= valB * valX;

            for(ip=0;ip<2;ip++){
                for(jp=0;jp<2;jp++){
                    for(kp=0;kp<2;kp++){
                        QP1= 1.e0 + POS[ip];
                        QM1= 1.e0 - POS[ip];
                        EP1= 1.e0 + POS[jp];
                        EM1= 1.e0 - POS[jp];
                        TP1= 1.e0 + POS[kp];
                        TM1= 1.e0 - POS[kp];

                        SHAPE[ip][jp][kp][0]= O8th * QM1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][1]= O8th * QP1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][2]= O8th * QP1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][3]= O8th * QM1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][4]= O8th * QM1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][5]= O8th * QP1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][6]= O8th * QP1 * EP1 * TP1;
                        SHAPE[ip][jp][kp][7]= O8th * QM1 * EP1 * TP1;
```

$$\mathrm{valX} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}$$

$$\mathrm{valA} = \frac{\nu}{(1-\nu)}\frac{E(1-\nu)}{(1+\nu)(1-2\nu)}$$

$$\mathrm{valB} = \frac{(1-2\nu)}{2(1-\nu)}\frac{E(1-\nu)}{(1+\nu)(1-2\nu)}$$

MAT ASS MAIN (2/7)

```
            WEI[0]= 1.0000000000e0;
            WEI[1]= 1.0000000000e0;

            POS[0]= -0.5773502692e0;
            POS[1]=  0.5773502692e0;

/***
            INIT.
            PNQ   - 1st-order derivative of shape function by QSI
            PNE   - 1st-order derivative of shape function by ETA
            PNT   - 1st-order derivative of shape function by ZET
***/
            valA=            POISSON /      (1.e0-POISSON);
            valB= (1.e0-2.e0*POISSON)/(2.e0*(1.e0-POISSON));
            valX= ELAST*(1.e0-POISSON)/((1.e0+POISSON)*(1.e0-2.e0*POISSON));

            valA= valA * valX;
            valB= valB * valX;

            for(ip=0;ip<2;ip++){
                for(jp=0;jp<2;jp++){
                    for(kp=0;kp<2;kp++){
                        QP1= 1.e0 + POS[ip];
                        QM1= 1.e0 - POS[ip];
                        EP1= 1.e0 + POS[jp];
                        EM1= 1.e0 - POS[jp];
                        TP1= 1.e0 + POS[kp];
                        TM1= 1.e0 - POS[kp];

                        SHAPE[ip][jp][kp][0]= O8th * QM1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][1]= O8th * QP1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][2]= O8th * QP1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][3]= O8th * QM1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][4]= O8th * QM1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][5]= O8th * QP1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][6]= O8th * QP1 * EP1 * TP1;
                        SHAPE[ip][jp][kp][7]= O8th * QM1 * EP1 * TP1;
```

$$\mathrm{valX} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}$$

$$\mathrm{valA} = \frac{\nu}{\cancel{(1-\nu)}}\frac{E\cancel{(1-\nu)}}{(1+\nu)(1-2\nu)} = \frac{E\nu}{(1+\nu)(1-2\nu)}$$

$$\mathrm{valB} = \frac{\cancel{(1-2\nu)}}{2\cancel{(1-\nu)}}\frac{E\cancel{(1-\nu)}}{(1+\nu)\cancel{(1-2\nu)}} = \frac{E}{2(1+\nu)}$$

Strain-Stress Relationship

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{Bmatrix} = \begin{bmatrix} \text{valX} & \text{valA} & \text{valA} & 0 & 0 & 0 \\ \text{valA} & \text{valX} & \text{valA} & 0 & 0 & 0 \\ \text{valA} & \text{valA} & \text{valX} & 0 & 0 & 0 \\ 0 & 0 & 0 & \text{valB} & 0 & 0 \\ 0 & 0 & 0 & 0 & \text{valB} & 0 \\ 0 & 0 & 0 & 0 & 0 & \text{valB} \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{Bmatrix}$$

$$[D]$$

$$\{\sigma\} = [D]\{\varepsilon\}$$

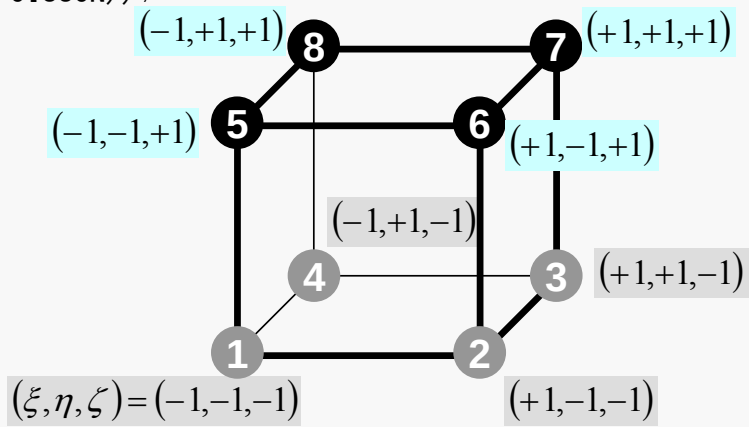
MAT ASS MAIN (2/7)

```
            WEI[0]= 1.0000000000e0;
            WEI[1]= 1.0000000000e0;

            POS[0]= -0.5773502692e0;
            POS[1]=  0.5773502692e0;

/***
            INIT.
            PNQ   - 1st-order derivative of shape function by QSI
            PNE   - 1st-order derivative of shape function by ETA
            PNT   - 1st-order derivative of shape function by ZET
***/
            valA=            POISSON /      (1.e0-POISSON);
            valB= (1.e0-2.e0*POISSON)/(2.e0*(1.e0-POISSON));
            valX= ELAST*(1.e0-POISSON)/((1.e0+POISSON)*(1.e0-2.e0*POISSON));

            valA= valA * valX;
            valB= valB * valX;

            for(ip=0;ip<2;ip++){
                for(jp=0;jp<2;jp++){
                    for(kp=0;kp<2;kp++){
                        QP1= 1.e0 + POS[ip];
                        QM1= 1.e0 - POS[ip];
                        EP1= 1.e0 + POS[jp];
                        EM1= 1.e0 - POS[jp];
                        TP1= 1.e0 + POS[kp];
                        TM1= 1.e0 - POS[kp];

                        SHAPE[ip][jp][kp][0]= O8th * QM1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][1]= O8th * QP1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][2]= O8th * QP1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][3]= O8th * QM1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][4]= O8th * QM1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][5]= O8th * QP1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][6]= O8th * QP1 * EP1 * TP1;
                        SHAPE[ip][jp][kp][7]= O8th * QM1 * EP1 * TP1;
```

$$\mathrm{QP1(i)} = (1+\xi_i), \quad \mathrm{QM1(i)} = (1-\xi_i)$$
$$\mathrm{EP1(j)} = (1+\eta_j), \quad \mathrm{EM1(j)} = (1-\eta_i)$$
$$\mathrm{TP1(k)} = (1+\zeta_k), \quad \mathrm{TM1(k)} = (1-\zeta_k)$$

MAT ASS MAIN (2/7)

```
            WEI[0]= 1.0000000000e0;
            WEI[1]= 1.0000000000e0;

            POS[0]= -0.5773502692e0;
            POS[1]=  0.5773502692e0;

/***
            INIT.
            PNQ   - 1st-order derivative of shape function by QSI
            PNE   - 1st-order derivative of shape function by ETA
            PNT   - 1st-order derivative of shape function by ZET
***/
            valA=            POISSON /      (1.e0-POISSON);
            valB= (1.e0-2.e0*POISSON)/(2.e0*(1.e0-POISSON));
            valX= ELAST*(1.e0-POISSON)/((1.e0+POISSON)*(1.e0-2.e0*POISSON));

            valA= valA * valX;
            valB= valB * valX;

            for(ip=0;ip<2;ip++){
                for(jp=0;jp<2;jp++){
                    for(kp=0;kp<2;kp++){
                        QP1= 1.e0 + POS[ip];
                        QM1= 1.e0 - POS[ip];
                        EP1= 1.e0 + POS[jp];
                        EM1= 1.e0 - POS[jp];
                        TP1= 1.e0 + POS[kp];
                        TM1= 1.e0 - POS[kp];

                        SHAPE[ip][jp][kp][0]= 08th * QM1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][1]= 08th * QP1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][2]= 08th * QP1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][3]= 08th * QM1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][4]= 08th * QM1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][5]= 08th * QP1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][6]= 08th * QP1 * EP1 * TP1;
                        SHAPE[ip][jp][kp][7]= 08th * QM1 * EP1 * TP1;
```



MAT ASS MAIN (2/7)

```
            WEI[0]= 1.0000000000e0;
            WEI[1]= 1.0000000000e0;

            POS[0]= -0.5773502692e0;
            POS[1]=  0.5773502692e0;

/***
            INIT.
            PNQ   - 1st-order derivative of shape function by QSI
            PNE   - 1st-order derivative of shape function by ETA
            PNT   - 1st-order derivative of shape function by ZET
***/
            valA=            POISSON /      (1.e0-POISSON);
            valB= (1.e0-2.e0*POISSON)/(2.e0*(1.e0-POISSON));
            valX= ELAST*(1.e0-POISSON)/((1.e0+POISSON)*(1.e0-2.e0*POISS

            valA= valA * valX;
            valB= valB * valX;

            for(ip=0;ip<2;ip++){
                for(jp=0;jp<2;jp++){
                    for(kp=0;kp<2;kp++){
                        QP1= 1.e0 + POS[ip];
                        QM1= 1.e0 - POS[ip];
                        EP1= 1.e0 + POS[jp];
                        EM1= 1.e0 - POS[jp];
                        TP1= 1.e0 + POS[kp];
                        TM1= 1.e0 - POS[kp];

                        SHAPE[ip][jp][kp][0]= O8th * QM1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][1]= O8th * QP1 * EM1 * TM1;
                        SHAPE[ip][jp][kp][2]= O8th * QP1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][3]= O8th * QM1 * EP1 * TM1;
                        SHAPE[ip][jp][kp][4]= O8th * QM1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][5]= O8th * QP1 * EM1 * TP1;
                        SHAPE[ip][jp][kp][6]= O8th * QP1 * EP1 * TP1;
                        SHAPE[ip][jp][kp][7]= O8th * QM1 * EP1 * TP1;
```

$$N_1(\xi,\eta,\zeta)=\frac{1}{8}(1-\xi)(1-\eta)(1-\zeta)$$

$$N_2(\xi,\eta,\zeta)=\frac{1}{8}(1+\xi)(1-\eta)(1-\zeta)$$

$$N_3(\xi,\eta,\zeta)=\frac{1}{8}(1+\xi)(1+\eta)(1-\zeta)$$

$$N_4(\xi,\eta,\zeta)=\frac{1}{8}(1-\xi)(1+\eta)(1-\zeta)$$

$$N_5(\xi,\eta,\zeta)=\frac{1}{8}(1-\xi)(1-\eta)(1+\zeta)$$

$$N_6(\xi,\eta,\zeta)=\frac{1}{8}(1+\xi)(1-\eta)(1+\zeta)$$

$$N_7(\xi,\eta,\zeta)=\frac{1}{8}(1+\xi)(1+\eta)(1+\zeta)$$

$$N_8(\xi,\eta,\zeta)=\frac{1}{8}(1-\xi)(1+\eta)(1+\zeta)$$

MAT ASS MAIN (3/7)

```
                PNQ[jp][kp][0]= - 08th * EM1 * TM1;
                PNQ[jp][kp][1]= + 08th * EM1 * TM1;
                PNQ[jp][kp][2]= + 08th * EP1 * TM1;
                PNQ[jp][kp][3]= - 08th * EP1 * TM1;
                PNQ[jp][kp][4]= - 08th * EM1 * TP1;
                PNQ[jp][kp][5]= + 08th * EM1 * TP1;
                PNQ[jp][kp][6]= + 08th * EP1 * TP1;
                PNQ[jp][kp][7]= - 08th * EP1 * TP1;
                PNE[ip][kp][0]= - 08th * QM1 * TM1;
                PNE[ip][kp][1]= - 08th * QP1 * TM1;
                PNE[ip][kp][2]= + 08th * QP1 * TM1;
                PNE[ip][kp][3]= + 08th * QM1 * TM1;
                PNE[ip][kp][4]= - 08th * QM1 * TP1;
                PNE[ip][kp][5]= - 08th * QP1 * TP1;
                PNE[ip][kp][6]= + 08th * QP1 * TP1;
                PNE[ip][kp][7]= + 08th * QM1 * TP1;
                PNT[ip][jp][0]= - 08th * QM1 * EM1;
                PNT[ip][jp][1]= - 08th * QP1 * EM1;
                PNT[ip][jp][2]= - 08th * QP1 * EP1;
                PNT[ip][jp][3]= - 08th * QM1 * EP1;
                PNT[ip][jp][4]= + 08th * QM1 * EM1;
                PNT[ip][jp][5]= + 08th * QP1 * EM1;
                PNT[ip][jp][6]= + 08th * QP1 * EP1;
                PNT[ip][jp][7]= + 08th * QM1 * EP1;
                            }
                }
}

for( icel=0;icel< ICELTOT;icel++){
                in1=ICELNOD[icel][0];
                in2=ICELNOD[icel][1];
                in3=ICELNOD[icel][2];
                in4=ICELNOD[icel][3];
                in5=ICELNOD[icel][4];
                in6=ICELNOD[icel][5];
                in7=ICELNOD[icel][6];
                in8=ICELNOD[icel][7];
```

$$PNQ(j,k) = \frac{\partial N_l}{\partial \xi}(\xi = \xi_i, \eta = \eta_j, \zeta = \zeta_k)$$

$$PNE(i,k) = \frac{\partial N_l}{\partial \eta}(\xi = \xi_i, \eta = \eta_j, \zeta = \zeta_k)$$

$$PNT(i,j) = \frac{\partial N_l}{\partial \zeta}(\xi = \xi_i, \eta = \eta_j, \zeta = \zeta_k)$$

$$\frac{\partial N_1}{\partial \xi}(\xi_i, \eta_j, \zeta_k) = -\frac{1}{8}(1-\eta_j)(1-\zeta_k)$$

$$\frac{\partial N_2}{\partial \xi}(\xi_i, \eta_j, \zeta_k) = +\frac{1}{8}(1-\eta_j)(1-\zeta_k)$$

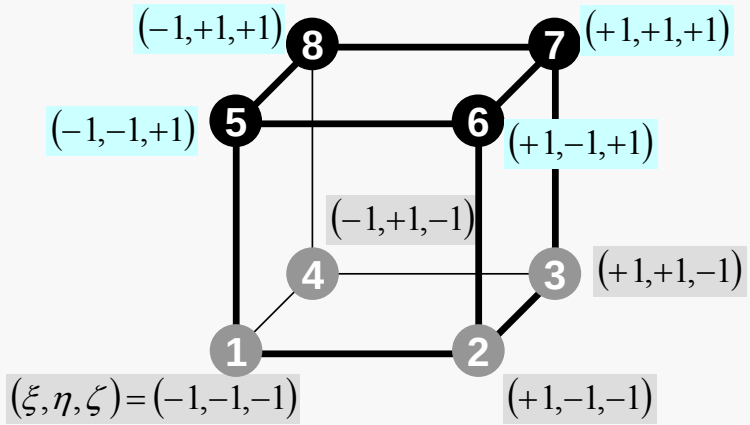
$$\frac{\partial N_3}{\partial \xi}(\xi_i, \eta_j, \zeta_k) = +\frac{1}{8}(1+\eta_j)(1-\zeta_k)$$

$$\frac{\partial N_3}{\partial \xi}(\xi_i, \eta_j, \zeta_k) = -\frac{1}{8}(1+\eta_j)(1-\zeta_k)$$

First Order Derivative of Shape Functions at (ξ_i, η_j, ζ_k)

MAT ASS MAIN (3/7)

```
                    PNQ[jp][kp][0]= - 08th * EM1 * TM1;
                    PNQ[jp][kp][1]= + 08th * EM1 * TM1;
                    PNQ[jp][kp][2]= + 08th * EP1 * TM1;
                    PNQ[jp][kp][3]= - 08th * EP1 * TM1;
                    PNQ[jp][kp][4]= - 08th * EM1 * TP1;
                    PNQ[jp][kp][5]= + 08th * EM1 * TP1;
                    PNQ[jp][kp][6]= + 08th * EP1 * TP1;
                    PNQ[jp][kp][7]= - 08th * EP1 * TP1;
                    PNE[ip][kp][0]= - 08th * QM1 * TM1;
                    PNE[ip][kp][1]= - 08th * QP1 * TM1;
                    PNE[ip][kp][2]= + 08th * QP1 * TM1;
                    PNE[ip][kp][3]= + 08th * QM1 * TM1;
                    PNE[ip][kp][4]= - 08th * QM1 * TP1;
                    PNE[ip][kp][5]= - 08th * QP1 * TP1;
                    PNE[ip][kp][6]= + 08th * QP1 * TP1;
                    PNE[ip][kp][7]= + 08th * QM1 * TP1;
                    PNT[ip][jp][0]= - 08th * QM1 * EM1;
                    PNT[ip][jp][1]= - 08th * QP1 * EM1;
                    PNT[ip][jp][2]= - 08th * QP1 * EP1;
                    PNT[ip][jp][3]= - 08th * QM1 * EP1;
                    PNT[ip][jp][4]= + 08th * QM1 * EM1;
                    PNT[ip][jp][5]= + 08th * QP1 * EM1;
                    PNT[ip][jp][6]= + 08th * QP1 * EP1;
                    PNT[ip][jp][7]= + 08th * QM1 * EP1;
                                }
                    }
          }

          for( icel=0;icel< ICELTOT;icel++){
                    in1=ICELNOD[icel][0];
                    in2=ICELNOD[icel][1];
                    in3=ICELNOD[icel][2];
                    in4=ICELNOD[icel][3];
                    in5=ICELNOD[icel][4];
                    in6=ICELNOD[icel][5];
                    in7=ICELNOD[icel][6];
                    in8=ICELNOD[icel][7];
```



MAT ASS MAIN (4/7)

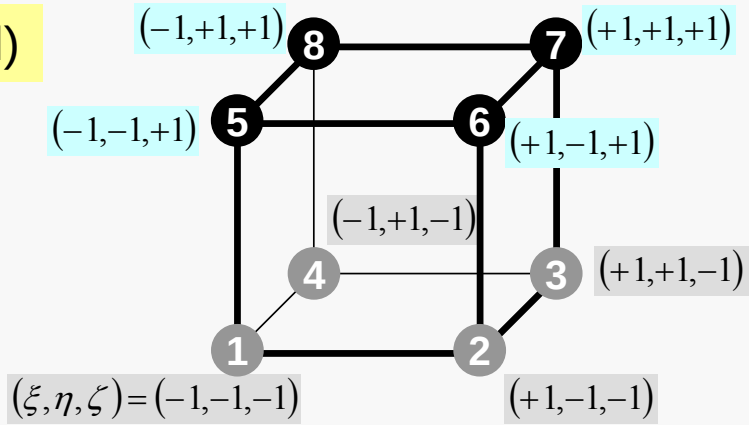
```
/**
 ** JACOBIAN & INVERSE JACOBIAN
**/
                        nodLOCAL[0]= in1;
                        nodLOCAL[1]= in2;
                        nodLOCAL[2]= in3;
                        nodLOCAL[3]= in4;
                        nodLOCAL[4]= in5;
                        nodLOCAL[5]= in6;
                        nodLOCAL[6]= in7;
                        nodLOCAL[7]= in8;

                        X1=XYZ[in1-1][0];
                        X2=XYZ[in2-1][0];
                        X3=XYZ[in3-1][0];
                        X4=XYZ[in4-1][0];
                        X5=XYZ[in5-1][0];
                        X6=XYZ[in6-1][0];
                        X7=XYZ[in7-1][0];
                        X8=XYZ[in8-1][0];

                        Y1=XYZ[in1-1][1];
                        Y2=XYZ[in2-1][1];
                        Y3=XYZ[in3-1][1];
                        Y4=XYZ[in4-1][1];
                        Y5=XYZ[in5-1][1];
                        Y6=XYZ[in6-1][1];
                        Y7=XYZ[in7-1][1];
                        Y8=XYZ[in8-1][1];

                        Z1=XYZ[in1-1][2];
                        Z2=XYZ[in2-1][2];
                        Z3=XYZ[in3-1][2];
                        Z4=XYZ[in4-1][2];
                        Z5=XYZ[in5-1][2];
                        Z6=XYZ[in6-1][2];
                        Z7=XYZ[in7-1][2];
                        Z8=XYZ[in8-1][2];

                        JACOBI(DETJ, PNQ, PNE, PNT, PNX, PNY, PNZ, X1, X2, X3, X4, X5, X6, X7, X8,
                               Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8);
```

Node ID (Global)

X-Coordinates of 8 nodes

Y-Coordinates of 8 nodes

Z-Coordinates of 8 nodes



Coordinates: Node ID - 1

JACOBI (1/4)

```
#include <stdio.h>
#include <math.h>
#include "precision.h"
#include "allocate.h"
/***
 *** JACOBI
 ***/
void JACOBI(
            KREAL DETJ[2][2][2],
            KREAL PNQ[2][2][8],KREAL PNE[2][2][8],KREAL PNT[2][2][8],
            KREAL PNX[2][2][2][8],KREAL PNY[2][2][2][8],KREAL PNZ[2][2][2][8],
            KREAL X1,KREAL X2,KREAL X3,KREAL X4,KREAL X5,KREAL X6,KREAL X7,KREAL X8,
            KREAL Y1,KREAL Y2,KREAL Y3,KREAL Y4,KREAL Y5,KREAL Y6,KREAL Y7,KREAL Y8,
            KREAL Z1,KREAL Z2,KREAL Z3,KREAL Z4,KREAL Z5,KREAL Z6,KREAL Z7,KREAL Z8)
{
/**
            calculates JACOBIAN & INVERSE JACOBIAN
                         dNi/dx, dNi/dy & dNi/dz
**/
            int ip,jp,kp;
            double dXdQ,dYdQ,dZdQ,dXdE,dYdE,dZdE,dXdT,dYdT,dZdT;
            double coef;
            double a11,a12,a13,a21,a22,a23,a31,a32,a33;

            for(ip=0;ip<2;ip++){
                for(jp=0;jp<2;jp++){
                    for(kp=0;kp<2;kp++){
                        PNX[ip][jp][kp][0]=0.0;
                        PNX[ip][jp][kp][1]=0.0;
                        PNX[ip][jp][kp][2]=0.0;
                        PNX[ip][jp][kp][3]=0.0;
                        PNX[ip][jp][kp][4]=0.0;
                        PNX[ip][jp][kp][5]=0.0;
                        PNX[ip][jp][kp][6]=0.0;
                        PNX[ip][jp][kp][7]=0.0;
```

Input $\left[\frac{\partial N_l}{\partial \xi},\frac{\partial N_l}{\partial \eta},\frac{\partial N_l}{\partial \zeta}\right],(x_l,y_l,z_l)(l=1\sim 8)$

Output $\left[\frac{\partial N_l}{\partial x},\frac{\partial N_l}{\partial y},\frac{\partial N_l}{\partial z}\right],\det|J|$

Values at each Gaussian Quad. Points: [ip][jp][kp]

Partial Diff. on Natural Coord. (1/4)

- According to formulae:

$$\frac{\partial N_i(\xi,\eta,\zeta)}{\partial \xi} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \xi} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial \xi}$$

$$\frac{\partial N_i(\xi,\eta,\zeta)}{\partial \eta} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \eta} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial \eta}$$

$$\frac{\partial N_i(\xi,\eta,\zeta)}{\partial \zeta} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \zeta} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \zeta} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial \zeta}$$

$\left[\frac{\partial N_i}{\partial \xi}, \frac{\partial N_i}{\partial \eta}, \frac{\partial N_i}{\partial \zeta}\right]$ can be easily derived according to definitions.

$\left[\frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial z}\right]$ are required for computations.

Partial Diff. on Natural Coord. (2/4)

- In matrix form:

$$\begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \\ \dfrac{\partial N_i}{\partial \zeta} \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} & \dfrac{\partial z}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} & \dfrac{\partial z}{\partial \eta} \\ \dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta} & \dfrac{\partial z}{\partial \zeta} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \\ \dfrac{\partial N_i}{\partial z} \end{Bmatrix} = [J] \begin{Bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \\ \dfrac{\partial N_i}{\partial z} \end{Bmatrix}$$

$$[J] = \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix}$$

$[J]$: Jacobi matrix, Jacobian

Partial Diff. on Natural Coord. (3/4)

- Components of Jacobian:

$$J_{11} = \frac{\partial x}{\partial \xi} = \frac{\partial}{\partial \xi}\left(\sum_{i=1}^{8} N_i x_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \xi} x_i, \quad J_{12} = \frac{\partial y}{\partial \xi} = \frac{\partial}{\partial \xi}\left(\sum_{i=1}^{8} N_i y_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \xi} y_i,$$

$$J_{13} = \frac{\partial z}{\partial \xi} = \frac{\partial}{\partial \xi}\left(\sum_{i=1}^{8} N_i z_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \xi} z_i$$

$$J_{21} = \frac{\partial x}{\partial \eta} = \frac{\partial}{\partial \eta}\left(\sum_{i=1}^{8} N_i x_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \eta} x_i, \quad J_{22} = \frac{\partial y}{\partial \eta} = \frac{\partial}{\partial \eta}\left(\sum_{i=1}^{8} N_i y_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \eta} y_i,$$

$$J_{23} = \frac{\partial z}{\partial \eta} = \frac{\partial}{\partial \eta}\left(\sum_{i=1}^{8} N_i z_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \eta} z_i$$

$$J_{31} = \frac{\partial x}{\partial \zeta} = \frac{\partial}{\partial \zeta}\left(\sum_{i=1}^{8} N_i x_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \zeta} x_i, \quad J_{32} = \frac{\partial y}{\partial \zeta} = \frac{\partial}{\partial \zeta}\left(\sum_{i=1}^{8} N_i y_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \zeta} y_i,$$

$$J_{33} = \frac{\partial z}{\partial \zeta} = \frac{\partial}{\partial \zeta}\left(\sum_{i=1}^{8} N_i z_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \zeta} z_i$$

JACOBI (2/4)

```
                    PNY[ip][jp][kp][0]=0.0;
                    PNY[ip][jp][kp][1]=0.0;
                    PNY[ip][jp][kp][2]=0.0;
                    PNY[ip][jp][kp][3]=0.0;
                    PNY[ip][jp][kp][4]=0.0;
                    PNY[ip][jp][kp][5]=0.0;
                    PNY[ip][jp][kp][6]=0.0;
                    PNY[ip][jp][kp][7]=0.0;

                    PNZ[ip][jp][kp][0]=0.0;
                    PNZ[ip][jp][kp][1]=0.0;
                    PNZ[ip][jp][kp][2]=0.0;
                    PNZ[ip][jp][kp][3]=0.0;
                    PNZ[ip][jp][kp][4]=0.0;
                    PNZ[ip][jp][kp][5]=0.0;
                    PNZ[ip][jp][kp][6]=0.0;
                    PNZ[ip][jp][kp][7]=0.0;

/**
          DETERMINANT of the JACOBIAN
**/
                    dXdQ = PNQ[jp][kp][0]*X1 + PNQ[jp][kp][1]*X2
                         + PNQ[jp][kp][2]*X3 + PNQ[jp][kp][3]*X4
                         + PNQ[jp][kp][4]*X5 + PNQ[jp][kp][5]*X6
                         + PNQ[jp][kp][6]*X7 + PNQ[jp][kp][7]*X8;
                    dYdQ = PNQ[jp][kp][0]*Y1 + PNQ[jp][kp][1]*Y2
                         + PNQ[jp][kp][2]*Y3 + PNQ[jp][kp][3]*Y4
                         + PNQ[jp][kp][4]*Y5 + PNQ[jp][kp][5]*Y6
                         + PNQ[jp][kp][6]*Y7 + PNQ[jp][kp][7]*Y8;
                    dZdQ = PNQ[jp][kp][0]*Z1 + PNQ[jp][kp][1]*Z2
                         + PNQ[jp][kp][2]*Z3 + PNQ[jp][kp][3]*Z4
                         + PNQ[jp][kp][4]*Z5 + PNQ[jp][kp][5]*Z6
                         + PNQ[jp][kp][6]*Z7 + PNQ[jp][kp][7]*Z8;
                    dXdE = PNE[ip][kp][0]*X1 + PNE[ip][kp][1]*X2
                         + PNE[ip][kp][2]*X3 + PNE[ip][kp][3]*X4
                         + PNE[ip][kp][4]*X5 + PNE[ip][kp][5]*X6
                         + PNE[ip][kp][6]*X7 + PNE[ip][kp][7]*X8;
```

$$[J]=\begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix}$$

$$\mathrm{dXdQ}=\frac{\partial x}{\partial \xi}=J_{11}$$

$$\mathrm{dYdQ}=\frac{\partial y}{\partial \xi}=J_{12}$$

$$\mathrm{dZdQ}=\frac{\partial z}{\partial \xi}=J_{13}$$

JACOBI (3/4)

```
                    dYdE = PNE[ip][kp][0]*Y1 + PNE[ip][kp][1]*Y2
                                + PNE[ip][kp][2]*Y3 + PNE[ip][kp][3]*Y4
                                + PNE[ip][kp][4]*Y5 + PNE[ip][kp][5]*Y6
                                + PNE[ip][kp][6]*Y7 + PNE[ip][kp][7]*Y8;
                    dZdE = PNE[ip][kp][0]*Z1 + PNE[ip][kp][1]*Z2
                                + PNE[ip][kp][2]*Z3 + PNE[ip][kp][3]*Z4
                                + PNE[ip][kp][4]*Z5 + PNE[ip][kp][5]*Z6
                                + PNE[ip][kp][6]*Z7 + PNE[ip][kp][7]*Z8;
                    dXdT = PNT[ip][jp][0]*X1 + PNT[ip][jp][1]*X2
                                + PNT[ip][jp][2]*X3 + PNT[ip][jp][3]*X4
                                + PNT[ip][jp][4]*X5 + PNT[ip][jp][5]*X6
                                + PNT[ip][jp][6]*X7 + PNT[ip][jp][7]*X8;
                    dYdT = PNT[ip][jp][0]*Y1 + PNT[ip][jp][1]*Y2
                                + PNT[ip][jp][2]*Y3 + PNT[ip][jp][3]*Y4
                                + PNT[ip][jp][4]*Y5 + PNT[ip][jp][5]*Y6
                                + PNT[ip][jp][6]*Y7 + PNT[ip][jp][7]*Y8;
                    dZdT = PNT[ip][jp][0]*Z1 + PNT[ip][jp][1]*Z2
                                + PNT[ip][jp][2]*Z3 + PNT[ip][jp][3]*Z4
                                + PNT[ip][jp][4]*Z5 + PNT[ip][jp][5]*Z6
                                + PNT[ip][jp][6]*Z7 + PNT[ip][jp][7]*Z8;
                    DETJ[ip][jp][kp]= dXdQ*(dYdE*dZdT-dZdE*dYdT) +
                                      dYdQ*(dZdE*dXdT-dXdE*dZdT) +
                                      dZdQ*(dXdE*dYdT-dYdE*dXdT);
/**
    INVERSE JACOBIAN
**/
                    coef=1.0 / DETJ[ip][jp][kp];

                    a11= coef * ( dYdE*dZdT - dZdE*dYdT );
                    a12= coef * ( dZdQ*dYdT - dYdQ*dZdT );
                    a13= coef * ( dYdQ*dZdE - dZdQ*dYdE );

                    a21= coef * ( dZdE*dXdT - dXdE*dZdT );
                    a22= coef * ( dXdQ*dZdT - dZdQ*dXdT );
                    a23= coef * ( dZdQ*dXdE - dXdQ*dZdE );

                    a31= coef * ( dXdE*dYdT - dYdE*dXdT );
                    a32= coef * ( dYdQ*dXdT - dXdQ*dYdT );
                    a33= coef * ( dXdQ*dYdE - dYdQ*dXdE );

                    DETJ[ip][jp][kp]=fabs(DETJ[ip][jp][kp]);
```

$$[J]=\begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix}$$

Partial Diff. on Natural Coord. (4/4)

- Partial differentiation on global coordinate system is introduced as follows (with inverse of Jacobian matrix (3 × 3))

$$\begin{Bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \\ \dfrac{\partial N_i}{\partial z} \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} & \dfrac{\partial z}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} & \dfrac{\partial z}{\partial \eta} \\ \dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta} & \dfrac{\partial z}{\partial \zeta} \end{bmatrix}^{-1} \begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \\ \dfrac{\partial N_i}{\partial \zeta} \end{Bmatrix} = \left[J \right]^{-1} \begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \\ \dfrac{\partial N_i}{\partial \zeta} \end{Bmatrix}$$

JACOBI (3/4)

```
                    dYdE = PNE[ip][kp][0]*Y1 + PNE[ip][kp][1]*Y2
                                + PNE[ip][kp][2]*Y3 + PNE[ip][kp][3]*Y4
                                + PNE[ip][kp][4]*Y5 + PNE[ip][kp][5]*Y6
                                + PNE[ip][kp][6]*Y7 + PNE[ip][kp][7]*Y8;
                    dZdE = PNE[ip][kp][0]*Z1 + PNE[ip][kp][1]*Z2
                                + PNE[ip][kp][2]*Z3 + PNE[ip][kp][3]*Z4
                                + PNE[ip][kp][4]*Z5 + PNE[ip][kp][5]*Z6
                                + PNE[ip][kp][6]*Z7 + PNE[ip][kp][7]*Z8;
                    dXdT = PNT[ip][jp][0]*X1 + PNT[ip][jp][1]*X2
                                + PNT[ip][jp][2]*X3 + PNT[ip][jp][3]*X4
                                + PNT[ip][jp][4]*X5 + PNT[ip][jp][5]*X6
                                + PNT[ip][jp][6]*X7 + PNT[ip][jp][7]*X8;
                    dYdT = PNT[ip][jp][0]*Y1 + PNT[ip][jp][1]*Y2
                                + PNT[ip][jp][2]*Y3 + PNT[ip][jp][3]*Y4
                                + PNT[ip][jp][4]*Y5 + PNT[ip][jp][5]*Y6
                                + PNT[ip][jp][6]*Y7 + PNT[ip][jp][7]*Y8;
                    dZdT = PNT[ip][jp][0]*Z1 + PNT[ip][jp][1]*Z2
                                + PNT[ip][jp][2]*Z3 + PNT[ip][jp][3]*Z4
                                + PNT[ip][jp][4]*Z5 + PNT[ip][jp][5]*Z6
                                + PNT[ip][jp][6]*Z7 + PNT[ip][jp][7]*Z8;
                    DETJ[ip][jp][kp]= dXdQ*(dYdE*dZdT-dZdE*dYdT) +
                                      dYdQ*(dZdE*dXdT-dXdE*dZdT) +
                                      dZdQ*(dXdE*dYdT-dYdE*dXdT);
/**
    INVERSE JACOBIAN
**/
                    coef=1.0 / DETJ[ip][jp][kp];

                    a11= coef * ( dYdE*dZdT - dZdE*dYdT );
                    a12= coef * ( dZdQ*dYdT - dYdQ*dZdT );
                    a13= coef * ( dYdQ*dZdE - dZdQ*dYdE );

                    a21= coef * ( dZdE*dXdT - dXdE*dZdT );
                    a22= coef * ( dXdQ*dZdT - dZdQ*dXdT );
                    a23= coef * ( dZdQ*dXdE - dXdQ*dZdE );

                    a31= coef * ( dXdE*dYdT - dYdE*dXdT );
                    a32= coef * ( dYdQ*dXdT - dXdQ*dYdT );
                    a33= coef * ( dXdQ*dYdE - dYdQ*dXdE );

                    DETJ[ip][jp][kp]=fabs(DETJ[ip][jp][kp]);
```

$$[J]^{-1} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

JACOBI (4/4)

```
/**
    set the dNi/dX, dNi/dY & dNi/dZ components
**/
            PNX[ip][jp][kp][0]=a11*PNQ[jp][kp][0]+a12*PNE[ip][kp][0]+a13*PNT[ip][jp][0];
            PNX[ip][jp][kp][1]=a11*PNQ[jp][kp][1]+a12*PNE[ip][kp][1]+a13*PNT[ip][jp][1];
            PNX[ip][jp][kp][2]=a11*PNQ[jp][kp][2]+a12*PNE[ip][kp][2]+a13*PNT[ip][jp][2];
            PNX[ip][jp][kp][3]=a11*PNQ[jp][kp][3]+a12*PNE[ip][kp][3]+a13*PNT[ip][jp][3];
            PNX[ip][jp][kp][4]=a11*PNQ[jp][kp][4]+a12*PNE[ip][kp][4]+a13*PNT[ip][jp][4];
            PNX[ip][jp][kp][5]=a11*PNQ[jp][kp][5]+a12*PNE[ip][kp][5]+a13*PNT[ip][jp][5];
            PNX[ip][jp][kp][6]=a11*PNQ[jp][kp][6]+a12*PNE[ip][kp][6]+a13*PNT[ip][jp][6];
            PNX[ip][jp][kp][7]=a11*PNQ[jp][kp][7]+a12*PNE[ip][kp][7]+a13*PNT[ip][jp][7];
            PNY[ip][jp][kp][0]=a21*PNQ[jp][kp][0]+a22*PNE[ip][kp][0]+a23*PNT[ip][jp][0];
            PNY[ip][jp][kp][1]=a21*PNQ[jp][kp][1]+a22*PNE[ip][kp][1]+a23*PNT[ip][jp][1];
            PNY[ip][jp][kp][2]=a21*PNQ[jp][kp][2]+a22*PNE[ip][kp][2]+a23*PNT[ip][jp][2];
            PNY[ip][jp][kp][3]=a21*PNQ[jp][kp][3]+a22*PNE[ip][kp][3]+a23*PNT[ip][jp][3];
            PNY[ip][jp][kp][4]=a21*PNQ[jp][kp][4]+a22*PNE[ip][kp][4]+a23*PNT[ip][jp][4];
            PNY[ip][jp][kp][5]=a21*PNQ[jp][kp][5]+a22*PNE[ip][kp][5]+a23*PNT[ip][jp][5];
            PNY[ip][jp][kp][6]=a21*PNQ[jp][kp][6]+a22*PNE[ip][kp][6]+a23*PNT[ip][jp][6];
            PNY[ip][jp][kp][7]=a21*PNQ[jp][kp][7]+a22*PNE[ip][kp][7]+a23*PNT[ip][jp][7];
            PNZ[ip][jp][kp][0]=a31*PNQ[jp][kp][0]+a32*PNE[ip][kp][0]+a33*PNT[ip][jp][0];
            PNZ[ip][jp][kp][1]=a31*PNQ[jp][kp][1]+a32*PNE[ip][kp][1]+a33*PNT[ip][jp][1];
            PNZ[ip][jp][kp][2]=a31*PNQ[jp][kp][2]+a32*PNE[ip][kp][2]+a33*PNT[ip][jp][2];
            PNZ[ip][jp][kp][3]=a31*PNQ[jp][kp][3]+a32*PNE[ip][kp][3]+a33*PNT[ip][jp][3];
            PNZ[ip][jp][kp][4]=a31*PNQ[jp][kp][4]+a32*PNE[ip][kp][4]+a33*PNT[ip][jp][4];
            PNZ[ip][jp][kp][5]=a31*PNQ[jp][kp][5]+a32*PNE[ip][kp][5]+a33*PNT[ip][jp][5];
            PNZ[ip][jp][kp][6]=a31*PNQ[jp][kp][6]+a32*PNE[ip][kp][6]+a33*PNT[ip][jp][6];
            PNZ[ip][jp][kp][7]=a31*PNQ[jp][kp][7]+a32*PNE[ip][kp][7]+a33*PNT[ip][jp][7];
                                    }
                        }
            }
}
```

$$\begin{Bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \\ \dfrac{\partial N_i}{\partial z} \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} & \dfrac{\partial z}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} & \dfrac{\partial z}{\partial \eta} \\ \dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta} & \dfrac{\partial z}{\partial \zeta} \end{bmatrix}^{-1} \begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \\ \dfrac{\partial N_i}{\partial \zeta} \end{Bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \\ \dfrac{\partial N_i}{\partial \zeta} \end{Bmatrix}$$

MAT_ASS_MAIN (5/7)

```
/**
        CONSTRUCT the GLOBAL MATRIX
**/
                for(ie=0;ie<8;ie++){
                    ip=nodLOCAL[ie];

                    for(je=0;je<8;je++){
                        jp=nodLOCAL[je];

                        kk=0;
                        if( jp > ip ){
                            iiS=indexU[ip-1];
                            iiE=indexU[ip  ];
                            for( k=iiS;k<iiE;k++){
                                if( itemU[k] == jp-1 ){
                                    kk=k;
                                    break;
                                }
                            }
                        }

                        if( jp < ip ){
                            iiS=indexL[ip-1];
                            iiE=indexL[ip  ];
                            for( k=iiS;k<iiE;k++){
                                if( itemL[k] == jp-1 ){
                                    kk=k;
                                    break;
                                }
                            }
                        }

                        PNXi= 0.e0;
                        PNYi= 0.e0;
                        PNZi= 0.e0;
                        PNXj= 0.e0;
                        PNYj= 0.e0;
                        PNZj= 0.e0;

                        VOL= 0.e0;
```

Non-Zero Off-Diagonal Block in Global Matix

$$A_{ip,jp}$$

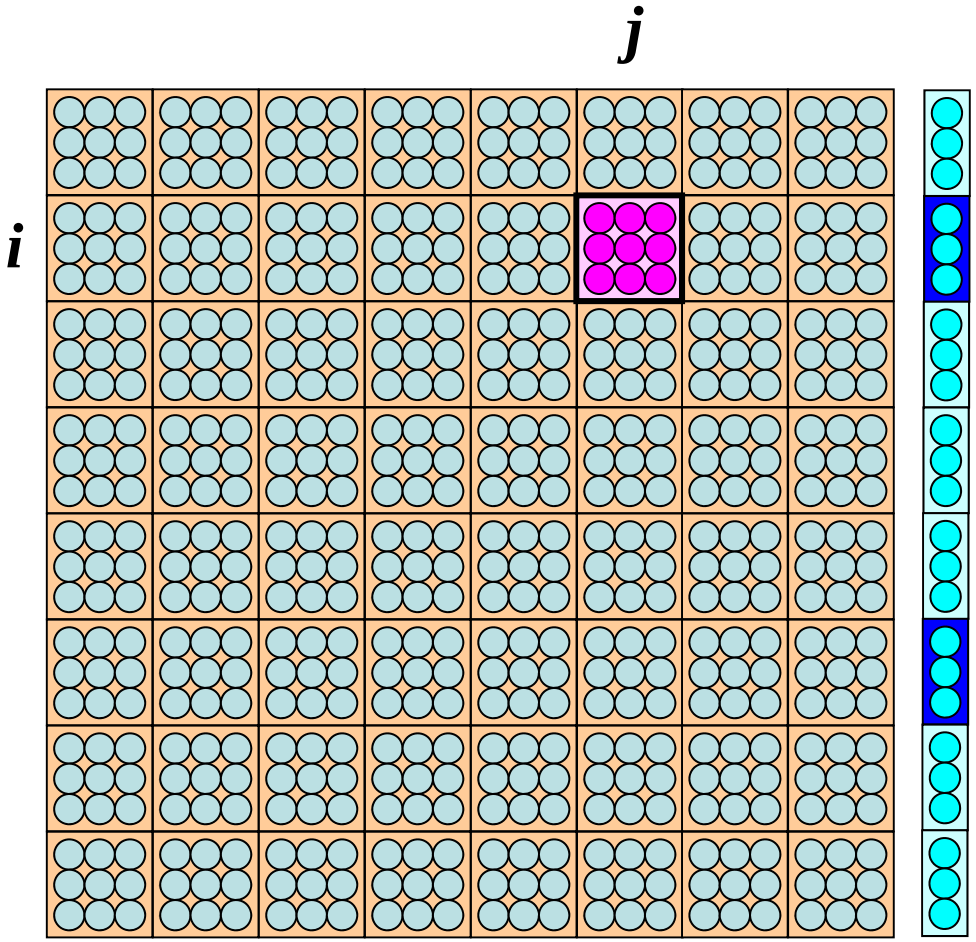
kk: address in “itemL”, “itemU”

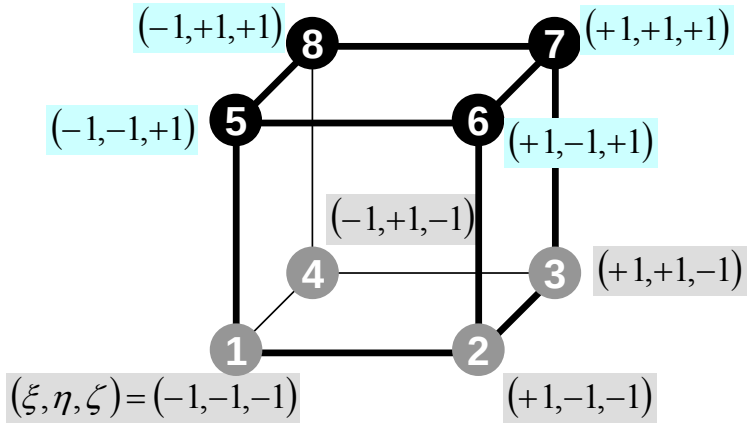
ip= nodLOCAL[ie]
jp= nodLOCAL[je]

Node ID (ip,jp) starting from 1

Element Matrix: 24×24

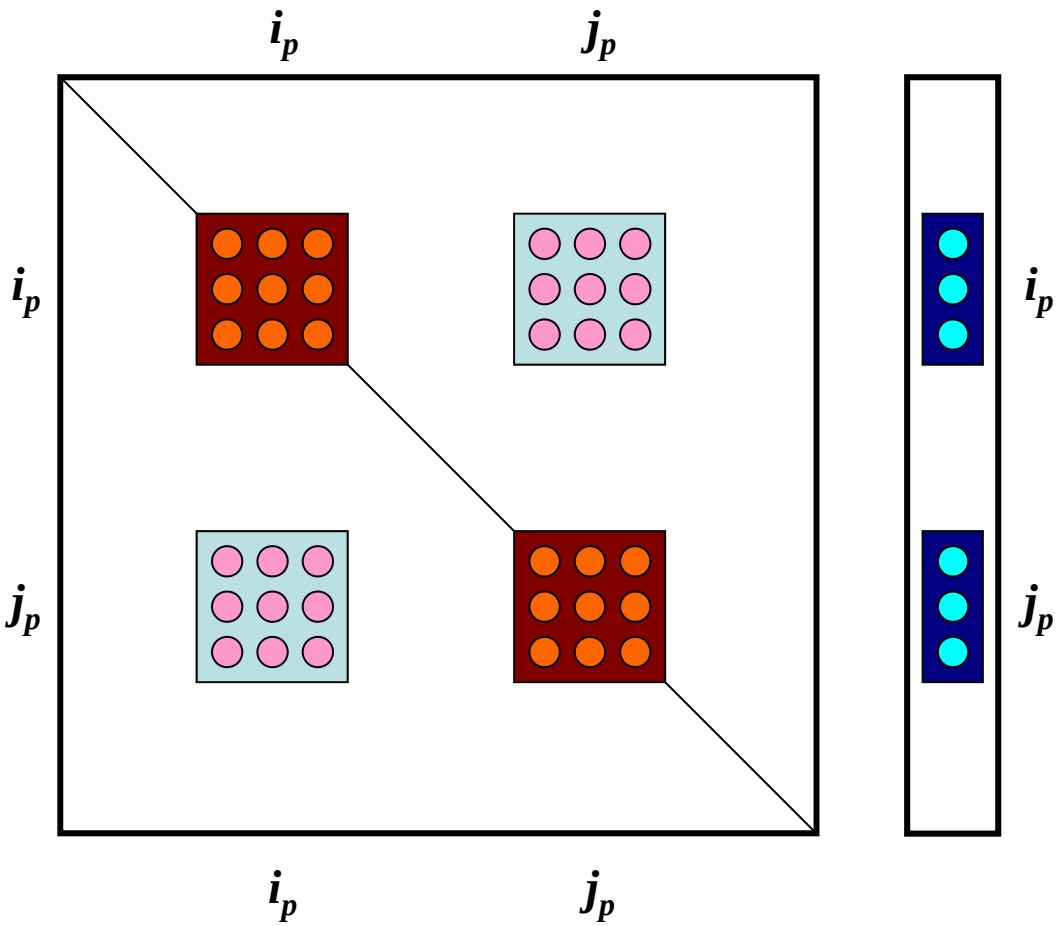
(u,v,w) components on each node are physically strongly-coupled: these three components are treated in block-wise manner: 8×8 matrix





$$\begin{bmatrix} a_{i_e j_e 11} & a_{i_e j_e 12} & a_{i_e j_e 13} \\ a_{i_e j_e 21} & a_{i_e j_e 22} & a_{i_e j_e 23} \\ a_{i_e j_e 31} & a_{i_e j_e 32} & a_{i_e j_e 33} \end{bmatrix} (i_e, j_e = 1 \ldots 8)$$

Global Matrix



MAT_ASS_MAIN (5/7)

```
/**
        CONSTRUCT the GLOBAL MATRIX
**/
                for(ie=0;ie<8;ie++){
                    ip=nodLOCAL[ie];

                    for(je=0;je<8;je++){
                        jp=nodLOCAL[je];

                        kk=0;
                        if( jp > ip ){
                            iiS=indexU[ip-1]+1;
                            iiE=indexU[ip  ];
                            for( k=iiS;k<=iiE;k++){
                                if( itemU[k-1] == jp ){
                                    kk=k;
                                    break;
                                }
                            }
                        }

                        if( jp < ip ){
                            iiS=indexL[ip-1]+1;
                            iiE=indexL[ip  ];
                            for( k=iiS;k<=iiE;k++){
                                if( itemL[k-1] == jp ){
                                    kk=k;
                                    break;
                                }
                            }
                        }

                        PNXi= 0.e0;
                        PNYi= 0.e0;
                        PNZi= 0.e0;
                        PNXj= 0.e0;
                        PNYj= 0.e0;
                        PNZj= 0.e0;

                        VOL= 0.e0;
```

Element Matrix (i_e~j_e): Local ID
Global Matrix (i_p~j_p): Global ID

kk : address in “itemU”, “itemL” starting from “0”

k: starting from “0”

ip,jp: starting from “1”

MAT_ASS_MAIN (6/7)

```
a11= 0.0e0; a12= 0.0e0; a13= 0.0e0;
a21= 0.0e0; a22= 0.0e0; a23= 0.0e0;
a31= 0.0e0; a32= 0.0e0; a33= 0.0e0;

for(ipn=0;ipn<2;ipn++){
    for(jpn=0;jpn<2;jpn++){
        for(kpn=0;kpn<2;kpn++){

            coef= -fabs(DETJ[ipn][jpn][kpn])*WEI[ipn]*WEI[jpn]*WEI[kpn];

            VOL+=coef;
            PNXi= PNX[ipn][jpn][kpn][ie];
            PNYi= PNY[ipn][jpn][kpn][ie];
            PNZi= PNZ[ipn][jpn][kpn][ie];

            PNXj= PNX[ipn][jpn][kpn][je];
            PNYj= PNY[ipn][jpn][kpn][je];
            PNZj= PNZ[ipn][jpn][kpn][je];

            a11+= (valX*PNXi*PNXj+valB*(PNYi*PNYj+PNZi*PNZj))*coef;
            a22+= (valX*PNYi*PNYj+valB*(PNZi*PNZj+PNXi*PNXj))*coef;
            a33+= (valX*PNZi*PNZj+valB*(PNXi*PNXj+PNYi*PNYj))*coef;

            a12+= (valA*PNXi*PNYj + valB*PNXj*PNYi)*coef;
            a13+= (valA*PNXi*PNZj + valB*PNXj*PNZi)*coef;
            a21+= (valA*PNYi*PNXj + valB*PNYj*PNXi)*coef;
            a23+= (valA*PNYi*PNZj + valB*PNYj*PNZi)*coef;
            a31+= (valA*PNZi*PNXj + valB*PNZj*PNXi)*coef;
            a32+= (valA*PNZi*PNYj + valB*PNZj*PNYi)*coef;
        }
    }
}
```

MAT_ASS_MAIN (6/7)

```
a11= 0.0e0; a12= 0.0e0; a13= 0.0e0;
a21= 0.0e0; a22= 0.0e0; a23= 0.0e0;
a31= 0.0e0; a32= 0.0e0; a33= 0.0e0;
```

$$-\int_V \left\{ D\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + b\left(\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} + \frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z}\right)\right\} dV =$$

$$-\iiint \left\{ D\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + b\left(\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} + \frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z}\right)\right\} dxdydz =$$

$$-\int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} \boxed{\left\{ D\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + b\left(\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} + \frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z}\right)\right\} \det|J|} \, d\xi d\eta d\zeta$$

$$\boxed{f(\xi,\eta,\zeta)}$$

$$I = \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} f(\xi,\eta,\zeta)\, d\xi d\eta d\zeta$$

$$= \sum_{i=1}^{L}\sum_{j=1}^{M}\sum_{k=1}^{N}\left[W_i \cdot W_j \cdot W_k \cdot f(\xi_i,\eta_j,\zeta_k)\right]$$

$$\mathrm{coef} = W_i \cdot W_j \cdot W_k \cdot \det\left|J\left(\xi_i,\eta_j,\zeta_k\right)\right|$$

```
pn][kpn])*WEI[ipn]*WEI[jpn]*WEI[kpn];

][ie];
][ie];
][ie];

][je];
][je];
][je];

a11+= (valX*PNXi*PNXj+valB*(PNYi*PNYj+PNZi*PNZj))*coef;
a22+= (valX*PNYi*PNYj+valB*(PNZi*PNZj+PNXi*PNXj))*coef;
a33+= (valX*PNZi*PNZj+valB*(PNXi*PNXj+PNYi*PNYj))*coef;

a12+= (valA*PNXi*PNYj + valB*PNXj*PNYi)*coef;
a13+= (valA*PNXi*PNZj + valB*PNXj*PNZi)*coef;
a21+= (valA*PNYi*PNXj + valB*PNYj*PNXi)*coef;
a23+= (valA*PNYi*PNZj + valB*PNYj*PNZi)*coef;
a31+= (valA*PNZi*PNXj + valB*PNZj*PNXi)*coef;
a32+= (valA*PNZi*PNYj + valB*PNZj*PNYi)*coef;
```

MAT_ASS_MAIN (6/7)

```
for(ipn=0;ipn<2;ipn++){
  for(jpn=0;jpn<2;jpn++){
    for(kpn=0;kpn<2;kpn++){

      coef= -fabs(DETJ[ipn][jpn][kpn])*WEI[ipn]*WEI[jpn]*WEI[kpn];
      PNXi= PNX[ipn][jpn][kpn][ie];
      PNYi= PNY[ipn][jpn][kpn][ie];
      PNZi= PNZ[ipn][jpn][kpn][ie];

      PNXj= PNX[ipn][jpn][kpn][je];
      PNYj= PNY[ipn][jpn][kpn][je];
      PNZj= PNZ[ipn][jpn][kpn][je];

      a11+= (valX*PNXi*PNXj+valB*(PNYi*PNYj+PNZi*PNZj))*coef;
      a22+= (valX*PNYi*PNYj+valB*(PNZi*PNZj+PNXi*PNXj))*coef;
      a33+= (valX*PNZi*PNZj+valB*(PNXi*PNXj+PNYi*PNYj))*coef;

      a12+= (valA*PNXi*PNYj + valB*PNXj*PNYi)*coef;
      a13+= (valA*PNXi*PNZj + valB*PNXj*PNZi)*coef;
      a21+= (valA*PNYi*PNXj + valB*PNYj*PNXi)*coef;
      a23+= (valA*PNYi*PNZj + valB*PNYj*PNZi)*coef;
      a31+= (valA*PNZi*PNXj + valB*PNZj*PNXi)*coef;
      a32+= (valA*PNZi*PNYj + valB*PNZj*PNYi)*coef;
    }
```

$$\begin{Bmatrix}\sigma_x\\ \sigma_y\\ \sigma_z\\ \tau_{xy}\\ \tau_{yz}\\ \tau_{zx}\end{Bmatrix}=\begin{bmatrix}\mathrm{valX} & \mathrm{valA} & \mathrm{valA} & 0 & 0 & 0\\ \mathrm{valA} & \mathrm{valX} & \mathrm{valA} & 0 & 0 & 0\\ \mathrm{valA} & \mathrm{valA} & \mathrm{valX} & 0 & 0 & 0\\ 0 & 0 & 0 & \mathrm{valB} & 0 & 0\\ 0 & 0 & 0 & 0 & \mathrm{valB} & 0\\ 0 & 0 & 0 & 0 & 0 & \mathrm{valB}\end{bmatrix}\begin{Bmatrix}\varepsilon_x\\ \varepsilon_y\\ \varepsilon_z\\ \gamma_{xy}\\ \gamma_{yz}\\ \gamma_{zx}\end{Bmatrix}$$

Equilibrium Eqn's in X-dir.

Define: $D = \dfrac{(1-\nu)E}{(1+\nu)(1-2\nu)}, \quad a = \dfrac{\nu E}{(1+\nu)(1-2\nu)}, \quad b = \dfrac{E}{2(1+\nu)}$

$$\sigma_x = D[N_{,x}]\{U\} + a[N_{,y}]\{V\} + a[N_{,z}]\{W\}$$
$$\tau_{xy} = b[N_{,y}]\{U\} + b[N_{,x}]\{V\}$$
$$\tau_{zx} = b[N_{,z}]\{U\} + b[N_{,x}]\{W\}$$

$$(*) =$$
$$-\int_V \left\{D[N_{,x}]^T[N_{,x}] + b\left([N_{,y}]^T[N_{,y}] + [N_{,z}]^T[N_{,z}]\right)\right\} dV \{U\}$$
$$-\int_V \left\{a[N_{,x}]^T[N_{,y}] + b\left([N_{,y}]^T[N_{,x}]\right)\right\} dV \{V\} - \int_V \left\{a[N_{,x}]^T[N_{,z}] + b[N_{,z}]^T[N_{,x}]\right\} dV \{W\}$$
$$+\int_V [N]^T \{X\} dV = 0$$

Equilibrium Eqn's in Y-dir.

$$-\int_V \left\{D\left[N_{,y}\right]^T\left[N_{,y}\right]+b\left(\left[N_{,z}\right]^T\left[N_{,z}\right]+\left[N_{,x}\right]^T\left[N_{,x}\right]\right)\right\}dV\left\{V\right\}$$

$$-\int_V \left\{a\left[N_{,y}\right]^T\left[N_{,x}\right]+b\left(\left[N_{,x}\right]^T\left[N_{,y}\right]\right)\right\}dV\left\{U\right\}-\int_V \left\{a\left[N_{,y}\right]^T\left[N_{,z}\right]+b\left[N_{,z}\right]^T\left[N_{,y}\right]\right\}dV\left\{W\right\}$$

$$+\int_V \left[N\right]^T\left\{Y\right\}dV=0$$

Equilibrium Eqn's in Z-dir.

$$-\int_V \left\{D\left[N_{,z}\right]^T\left[N_{,z}\right]+b\left(\left[N_{,x}\right]^T\left[N_{,x}\right]+\left[N_{,y}\right]^T\left[N_{,y}\right]\right)\right\}dV\left\{W\right\}$$

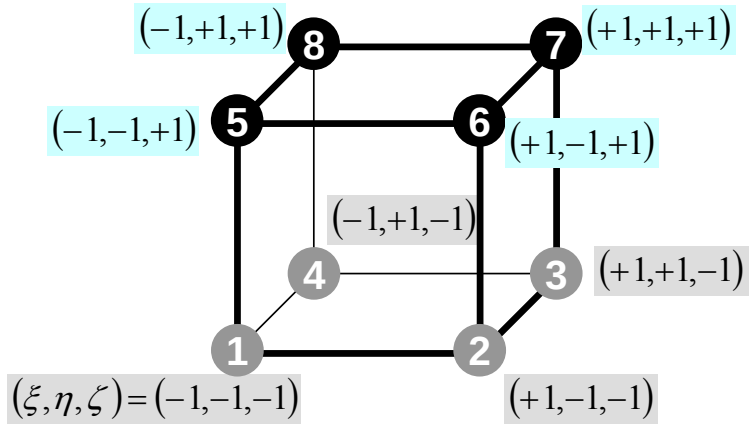
$$-\int_V \left\{a\left[N_{,z}\right]^T\left[N_{,x}\right]+b\left(\left[N_{,x}\right]^T\left[N_{,z}\right]\right)\right\}dV\left\{U\right\}-\int_V \left\{a\left[N_{,z}\right]^T\left[N_{,y}\right]+b\left[N_{,y}\right]^T\left[N_{,z}\right]\right\}dV\left\{V\right\}$$

$$+\int_V \left[N\right]^T\left\{Z\right\}dV=0$$

Elem. Matrix: *i-j* component, *X*-dir (1/3)

$$\begin{bmatrix} a_{ij_{11}} & a_{ij_{12}} & a_{ij_{13}} \\ a_{ij_{21}} & a_{ij_{22}} & a_{ij_{23}} \\ a_{ij_{31}} & a_{ij_{32}} & a_{ij_{33}} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (i, j = 1 \ldots 8)$$

valX=D; valA=a; valB=b



$$a_{ij_{11}} = -\int_V \left\{ \mathrm{valX} \cdot N_{i,x} \cdot N_{j,x} + \mathrm{valB} \cdot \left(N_{i,y} \cdot N_{j,y} + N_{i,z} \cdot N_{j,z} \right) \right\} dV$$

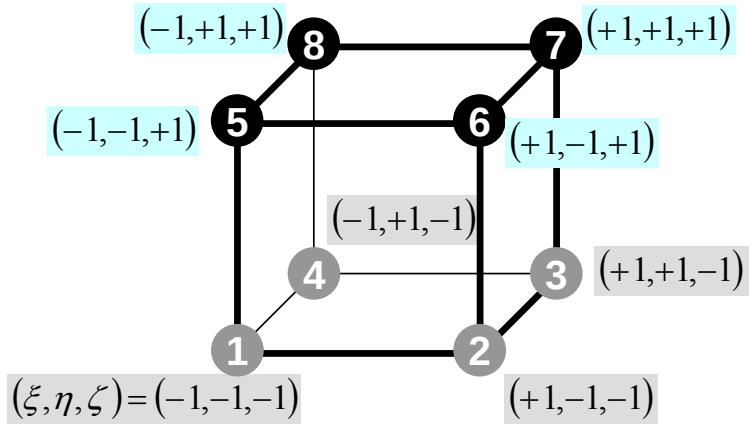
$$a_{ij_{12}} = -\int_V \left\{ \mathrm{valA} \cdot N_{i,x} \cdot N_{j,y} + \mathrm{valB} \cdot N_{i,y} \cdot N_{j,x} \right\} dV$$

$$a_{ij_{13}} = -\int_V \left\{ \mathrm{valA} \cdot N_{i,x} \cdot N_{j,z} + \mathrm{valB} \cdot N_{i,z} \cdot N_{j,x} \right\} dV$$

Elem. Matrix: *i*-*j* component, *X*-dir (1/3)

$$\begin{bmatrix} a_{ij11} & a_{ij12} & a_{ij13} \\ a_{ij21} & a_{ij22} & a_{ij23} \\ a_{ij31} & a_{ij32} & a_{ij33} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (i, j = 1 \ldots 8)$$

valX=D; valA=a; valB=b



$$-\int_V \left\{ D\left[N_{,x}\right]^T\left[N_{,x}\right] + b\left(\left[N_{,y}\right]^T\left[N_{,y}\right] + \left[N_{,z}\right]^T\left[N_{,z}\right]\right)\right\} dV \{U\}$$

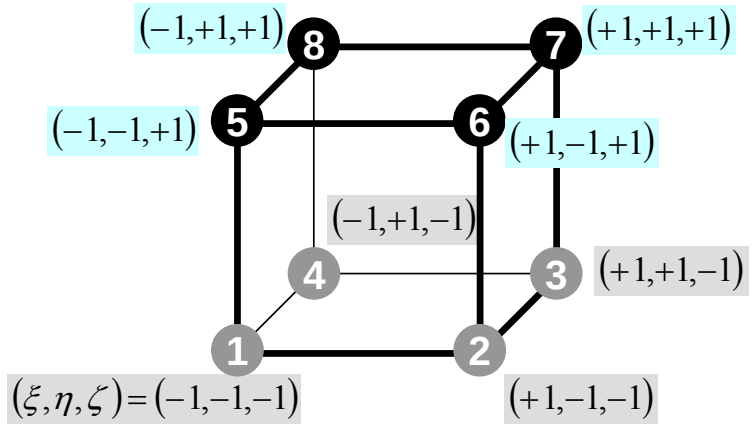
$$-\int_V \left\{ a\left[N_{,x}\right]^T\left[N_{,y}\right] + b\left(\left[N_{,y}\right]^T\left[N_{,x}\right]\right)\right\} dV \{V\}$$

$$-\int_V \left\{ a\left[N_{,x}\right]^T\left[N_{,z}\right] + b\left[N_{,z}\right]^T\left[N_{,x}\right]\right\} dV \{W\}$$

Elem. Matrix: *i-j* component, *Y*-dir (2/3)

$$\begin{bmatrix} a_{ij\,11} & a_{ij\,12} & a_{ij\,13} \\ a_{ij\,21} & a_{ij\,22} & a_{ij\,23} \\ a_{ij\,31} & a_{ij\,32} & a_{ij\,33} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (i, j = 1 \ldots 8)$$

valX=D; valA=a; valB=b



$$a_{ij\,21} = -\int_V \left\{ \mathrm{valA} \cdot N_{i,y} \cdot N_{j,x} + \mathrm{valB} \cdot N_{i,x} \cdot N_{j,y} \right\} dV$$

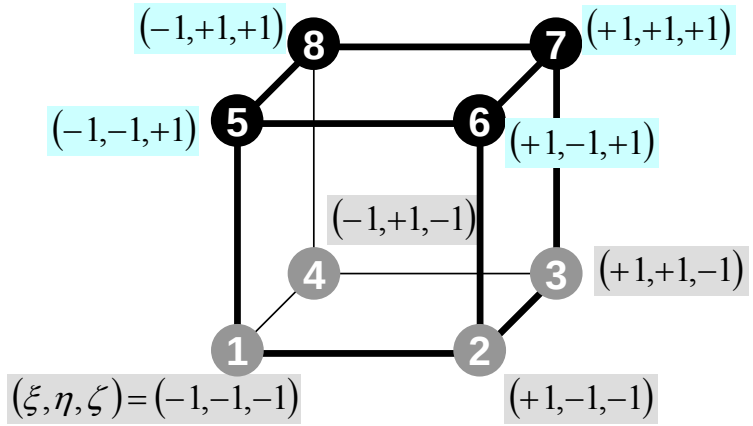
$$a_{ij\,22} = -\int_V \left\{ \mathrm{valX} \cdot N_{i,y} \cdot N_{j,y} + \mathrm{valB} \cdot \left(N_{i,z} \cdot N_{j,z} + N_{i,x} \cdot N_{j,x} \right) \right\} dV$$

$$a_{ij\,23} = -\int_V \left\{ \mathrm{valA} \cdot N_{i,y} \cdot N_{j,z} + \mathrm{valB} \cdot N_{i,z} \cdot N_{j,y} \right\} dV$$

Elem. Matrix: *i-j* component, *Y*-dir (2/3)

$$\begin{bmatrix} a_{ij11} & a_{ij12} & a_{ij13} \\ a_{ij21} & a_{ij22} & a_{ij23} \\ a_{ij31} & a_{ij32} & a_{ij33} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (i, j = 1 \ldots 8)$$

valX=D; valA=a; valB=b



$$-\int_V \left\{ a\left[N_{,y}\right]^T\left[N_{,x}\right] + b\left(\left[N_{,x}\right]^T\left[N_{,y}\right]\right)\right\} dV \{U\}$$

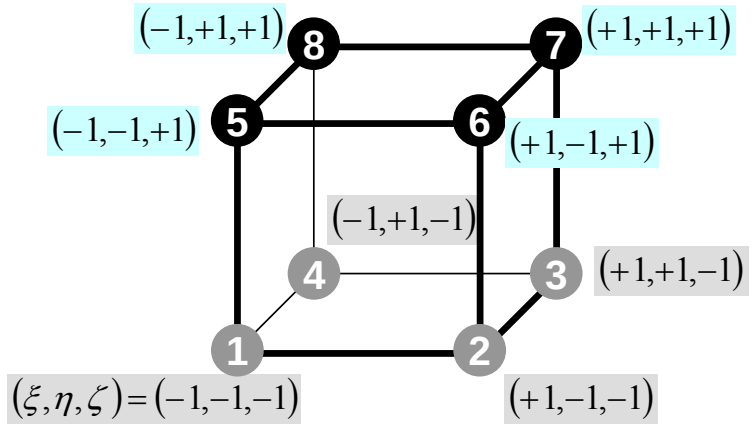
$$-\int_V \left\{ D\left[N_{,y}\right]^T\left[N_{,y}\right] + b\left(\left[N_{,z}\right]^T\left[N_{,z}\right] + \left[N_{,x}\right]^T\left[N_{,x}\right]\right)\right\} dV \{V\}$$

$$-\int_V \left\{ a\left[N_{,y}\right]^T\left[N_{,z}\right] + b\left[N_{,z}\right]^T\left[N_{,y}\right]\right\} dV \{W\}$$

Elem. Matrix: *i-j* component, *Z*-dir (3/3)

$$\begin{bmatrix} a_{ij11} & a_{ij12} & a_{ij13} \\ a_{ij21} & a_{ij22} & a_{ij23} \\ a_{ij31} & a_{ij32} & a_{ij33} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (i, j = 1 \ldots 8)$$

valX=D; valA=a; valB=b



$$a_{ij31} = -\int_V \left\{ \mathrm{valA} \cdot N_{i,z} \cdot N_{j,x} + \mathrm{valB} \cdot N_{i,x} \cdot N_{j,z} \right\} dV$$

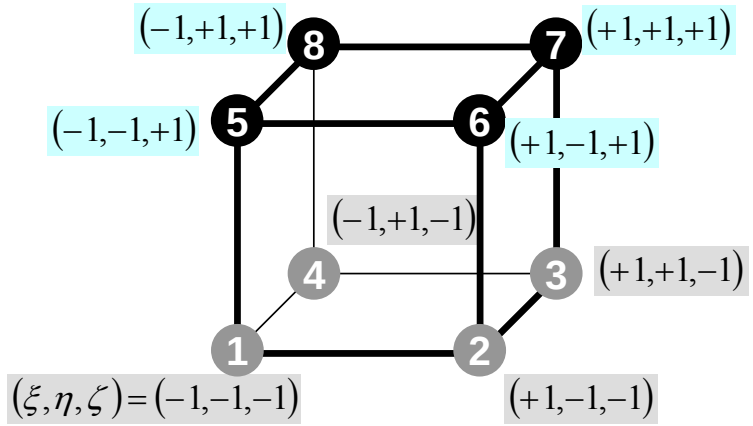
$$a_{ij32} = -\int_V \left\{ \mathrm{valA} \cdot N_{i,z} \cdot N_{j,y} + \mathrm{valB} \cdot N_{i,y} \cdot N_{j,z} \right\} dV$$

$$a_{ij33} = -\int_V \left\{ \mathrm{valD} \cdot N_{i,z} \cdot N_{j,z} + \mathrm{valB} \cdot \left(N_{i,x} \cdot N_{j,x} + N_{i,y} \cdot N_{j,y} \right) \right\} dV$$

Elem. Matrix: *i*-*j* component, *Z*-dir (3/3)

$$\begin{bmatrix} a_{ij11} & a_{ij12} & a_{ij13} \\ a_{ij21} & a_{ij22} & a_{ij23} \\ a_{ij31} & a_{ij32} & a_{ij33} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (i, j = 1 \ldots 8)$$

valX=D; valA=a; valB=b

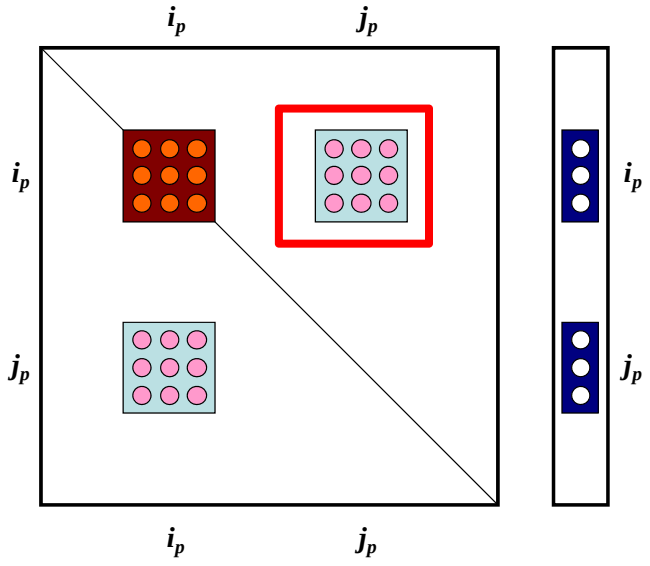


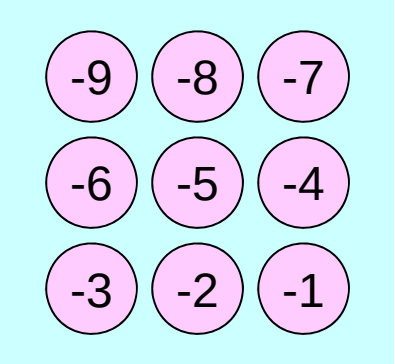
$$-\int_V \left\{ a\left[N_{,z}\right]^T\left[N_{,x}\right] + b\left(\left[N_{,x}\right]^T\left[N_{,z}\right]\right)\right\} dV \{U\}$$

$$-\int_V \left\{ a\left[N_{,z}\right]^T\left[N_{,y}\right] + b\left[N_{,y}\right]^T\left[N_{,z}\right]\right\} dV \{V\}$$

$$-\int_V \left\{ D\left[N_{,z}\right]^T\left[N_{,z}\right] + b\left(\left[N_{,x}\right]^T\left[N_{,x}\right] + \left[N_{,y}\right]^T\left[N_{,y}\right]\right)\right\} dV \{W\}$$

MAT_ASS_MAIN (7/7)





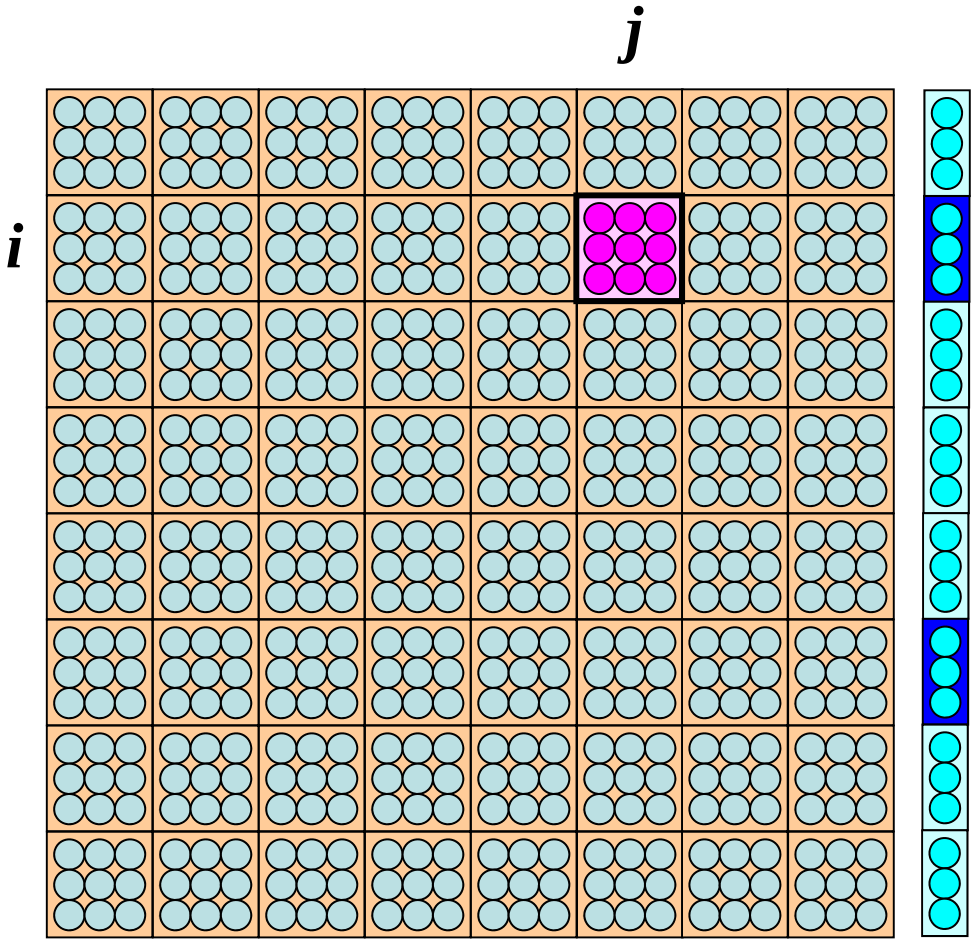

```
for(icel=0;icel<ICELTOT;icel++){

    for(ie=0;ie<8;ie++){
        ip=nodLOCAL[ie];

            for(je=0;je<8;je++){
                jp=nodLOCAL[je];
...
                    if (jp > ip) {  AU[9*kk-9]+=a11;
                                    AU[9*kk-8]+=a12;
                                    AU[9*kk-7]+=a13;
                                    AU[9*kk-6]+=a21;
                                    AU[9*kk-5]+=a22;
                                    AU[9*kk-4]+=a23;
                                    AU[9*kk-3]+=a31;
                                    AU[9*kk-2]+=a32;
                                    AU[9*kk-1]+=a33;}

                    if (jp < ip) {  AL[9*kk-9]+=a11;
                                    AL[9*kk-8]+=a12;
                                    AL[9*kk-7]+=a13;
                                    AL[9*kk-6]+=a21;
                                    AL[9*kk-5]+=a22;
                                    AL[9*kk-4]+=a23;
                                    AL[9*kk-3]+=a31;
                                    AL[9*kk-2]+=a32;
                                    AL[9*kk-1]+=a33;}

...
            }
    }
}
```

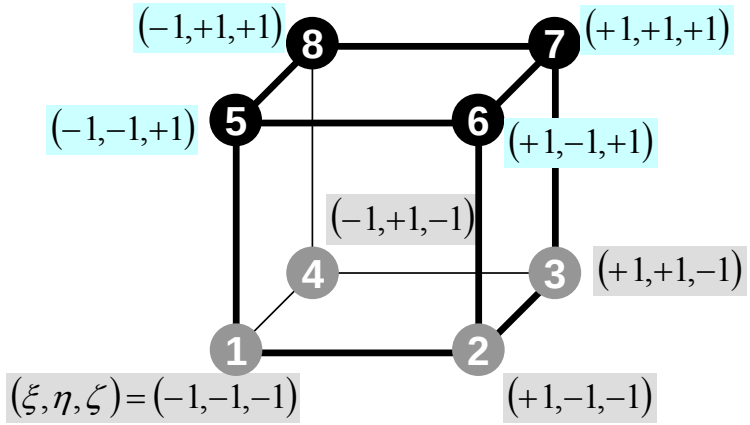
Element Matrix (i_e~j_e): Local ID
Global Matrix (i_p~j_p): Global ID

kk: address in “itemU”, “itemL” starting from “0”

Element Matrix: 24×24

(u,v,w) components on each node are physically strongly-coupled: these three components are treated in block-wise manner: 8×8 matrix





$$\begin{bmatrix} a_{i_e j_e 11} & a_{i_e j_e 12} & a_{i_e j_e 13} \\ a_{i_e j_e 21} & a_{i_e j_e 22} & a_{i_e j_e 23} \\ a_{i_e j_e 31} & a_{i_e j_e 32} & a_{i_e j_e 33} \end{bmatrix} (i_e, j_e = 1 \ldots 8)$$

MAT_ASS_MAIN (7/7)

```
for(icel=0;icel<ICELTOT;icel++){

    for(ie=0;ie<8;ie++){
          ip=nodLOCAL[ie];

             for(je=0;je<8;je++){
                jp=nodLOCAL[je];
...
                        if (jp > ip) {  AU[9*kk  ]+=a11;
                                        AU[9*kk+1]+=a12;
                                        AU[9*kk+2]+=a13;
                                        AU[9*kk+3]+=a21;
                                        AU[9*kk+4]+=a22;
                                        AU[9*kk+5]+=a23;
                                        AU[9*kk+6]+=a31;
                                        AU[9*kk+7]+=a32;
                                        AU[9*kk+8]+=a33;}

                        if (jp < ip) {  AL[9*kk  ]+=a11;
                                        AL[9*kk+1]+=a12;
                                        AL[9*kk+2]=a13;
                                        AL[9*kk+3]+=a21;
                                        AL[9*kk+4]+=a22;
                                        AL[9*kk+5]+=a23;
                                        AL[9*kk+6]+=a31;
                                        AL[9*kk+7]+=a32;
                                        AL[9*kk+8]+=a33;}

                        if (jp == ip) { D[9*ip  ]+=a11;
                                        D[9*ip+1]+=a12;
                                        D[9*ip+2]+=a13;
                                        D[9*ip+3]+=a21;
                                        D[9*ip+4]+=a22;
                                        D[9*ip+5]+=a23;
                                        D[9*ip+6]+=a31;
                                        D[9*ip+7]+=a32;
                                        D[9*ip+8]+=a33; }
...
             }
    }
}
```

MAT_ASS_BC: Overview

```
do i= 1, N     Loop for Nodes
   "Mark"  nodes where Dirichlet B.C. are applied (IWKX)
enddo
```

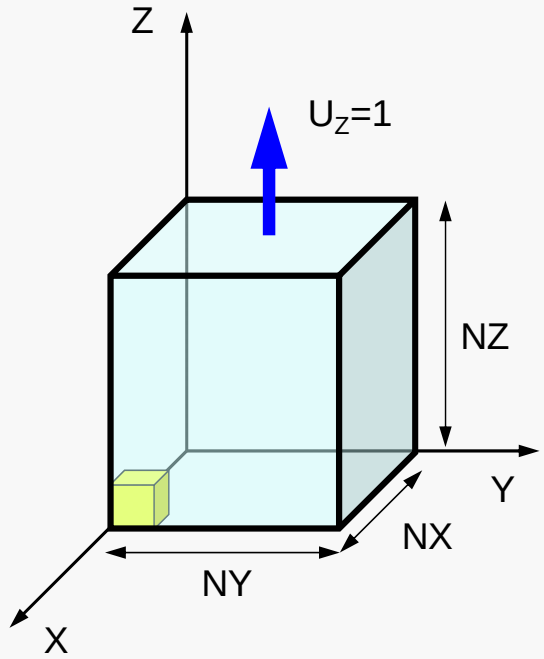
```
do i= 1, N  Loop for Nodes
  if (IWKX(i,1).eq.1) then   if "marked" nodes
    corresponding components of RHS (B), Diagonal Block (D) are corrected (row, col.)
    do k= indexL(i-1)+1, indexL(i)  Lower Triangular Block
      corresponding comp. of non-zero off-diagonal (lower-triangular) blocks (AL) are corrected
    enddo
    do k= indexU(i-1)+1, indexU(i)   Upper Triangular Block
       corresponding comp. of non-zero off-diagonal (upper-triangular) blocks (AU) are corrected
    enddo
  endif
enddo

do i= 1, N  節点ループ
  do k= indexL(i-1)+1, indexL(i)      Lower Triangular Block
    if (IWKX(itemL(k),1).eq.1) then   if corresponding node (off-diagonal block) is "marked"
      corresponding components of RHS and AL are corrected (col.)
    endif
  enddo
  do k= indexU(i-1)+1, indexU(i)      Upper Triangular Block
    if (IWKX(itemU(k),1).eq.1) then   if corresponding node (off-diagonal block) is "marked"
      corresponding components of RHS and AU are corrected (col.)
    endif
  enddo
enddo
```

MAT_ASS_BC (1/9)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
void MAT_ASS_BC()
{
            int i,j,k,in,ib,ib0,icel;
            int in1,in2,in3,in4,in5,in6,in7,in8;
            int iq1,iq2,iq3,iq4,iq5,iq6,iq7,iq8;
            int iS,iE;
            double STRESS,VAL;

            IWKX=(KINT**) allocate_matrix(sizeof(KINT),N,2);
            for(i=0;i<N;i++) for(j=0;j<2;j++) IWKX[i][j]=0;
```

MAT_ASS_BC (2/9)

```
/**
        Z=Zmax
**/
        for(in=0;in<N;in++) IWKX[in][0]=0;

        ib0=-1;

        for( ib0=0;ib0<NODGRPtot;ib0++){
                if( strcmp(NODGRP_NAME[ib0].name,"Zmax") == 0 ) break;
        }

        for( ib=NODGRP_INDEX[ib0];ib<NODGRP_INDEX[ib0+1];ib++){
                in=NODGRP_ITEM[ib];
                IWKX[in-1][0]=1;
        }

        for(in=0;in<N;in++){
                if( IWKX[in][0] == 1 ){
                        B[3*in  ]= B[3*in  ] - D[9*in+2]*1.e0;
                        B[3*in+1]= B[3*in+1] - D[9*in+5]*1.e0;
                        D[9*in+2]= 0.e0;
                        D[9*in+5]= 0.e0;
                        D[9*in+6]= 0.e0;
                        D[9*in+7]= 0.e0;
                        D[9*in+8]= 1.e0;
                        B[3*in+2]= 1.e0;
                        iS= indexL[in];
                        iE= indexL[in+1];
                        for(k=iS;k<iE;k++){
                                AL[9*k+6]= 0.e0;
                                AL[9*k+7]= 0.e0;
                                AL[9*k+8]= 0.e0;
                        }
                        iS= indexU[in];
                        iE= indexU[in+1];
                        for(k=iS;k<iE;k++){
                                AU[9*k+6]= 0.e0;
                                AU[9*k+7]= 0.e0;
                                AU[9*k+8]= 0.e0;
                        }
                }  }
```



MAT_ASS_BC (2/9)

```
/**
        Z=Zmax
**/
        for(in=0;in<N;in++) IWKX[in][0]=0;

        ib0=-1;

        for( ib0=0;ib0<NODGRPtot;ib0++){
                if( strcmp(NODGRP_NAME[ib0].name,"Zmax") == 0 ) break;
        }

        for( ib=NODGRP_INDEX[ib0];ib<NODGRP_INDEX[ib0+1];ib++){
                in=NODGRP_ITEM[ib];
                IWKX[in-1][0]=1;
        }

        for(in=0;in<N;in++){
                if( IWKX[in][0] == 1 ){
                        B[3*in  ]= B[3*in  ] - D[9*in+2]*
                        B[3*in+1]= B[3*in+1] - D[9*in+5]*
                        D[9*in+2]= 0.e0;
                        D[9*in+5]= 0.e0;
                        D[9*in+6]= 0.e0;
                        D[9*in+7]= 0.e0;
                        D[9*in+8]= 1.e0;
                        B[3*in+2]= 1.e0;
                        iS= indexL[in];
                        iE= indexL[in+1];
                        for(k=iS;k<iE;k++){
                                AL[9*k+6]= 0.e0;
                                AL[9*k+7]= 0.e0;
                                AL[9*k+8]= 0.e0;
                        }
                        iS= indexU[in];
                        iE= indexU[in+1];
                        for(k=iS;k<iE;k++){
                                AU[9*k+6]= 0.e0;
                                AU[9*k+7]= 0.e0;
                                AU[9*k+8]= 0.e0;
                        }
                }  }
```

If the node “in” is included in the node group “Zmax”

IWKX[in-1][0]= 1

MAT_ASS_BC (2/9)

```
/**
        Z=Zmax
**/
        for(in=0;in<N;in++) IWKX[in][0]=0;

        ib0=-1;

        for( ib0=0;ib0<NODGRPtot;ib0++){
                if( strcmp(NODGRP_NAME[ib0].name,"Zmax") == 0 ) break;
        }

        for( ib=NODGRP_INDEX[ib0];ib<NODGRP_INDEX[ib0+1];ib++){
                in=NODGRP_ITEM[ib];
                IWKX[in-1][0]=1;
        }

        for(in=0;in<N;in++){
                if( IWKX[in][0] == 1 ){
                        B[3*in  ]= B[3*in  ] - D[9*in+2]*1.e0;
                        B[3*in+1]= B[3*in+1] - D[9*in+5]*1.e0;
                        D[9*in+2]= 0.e0;
                        D[9*in+5]= 0.e0;
                        D[9*in+6]= 0.e0;
                        D[9*in+7]= 0.e0;
                        D[9*in+8]= 1.e0;
                        B[3*in+2]= 1.e0;
                        iS= indexL[in];
                        iE= indexL[in+1];
                        for(k=iS;k<iE;k++){
                                AL[9*k+6]= 0.e0;
                                AL[9*k+7]= 0.e0;
                                AL[9*k+8]= 0.e0;
                        }
                        iS= indexU[in];
                        iE= indexU[in+1];
                        for(k=iS;k<iE;k++){
                                AU[9*k+6]= 0.e0;
                                AU[9*k+7]= 0.e0;
                                AU[9*k+8]= 0.e0;
                        }
                }  }
```

If the node “in” is included in “Zmax”, displacement in Z-direction is equal to 1. i.e.

B[3*in+2]= 1.0 (in= 0~N-1)

In FORTRAN:

B[3*in-2]= 1.0 (in= 1, N)

MAT_ASS_BC (3/9)

```
        for(in=0;in<N;in++){
                        iS= indexL[in];
                        iE= indexL[in+1];
                        for(k=iS;k<=iE;k++){
                                    if (IWKX[itemL[k]][0] == 1 ) {
                                                B[3*in  ]= B[3*in  ] - AL[9*k+2]*1.e0;
                                                B[3*in+1]= B[3*in+1] - AL[9*k+5]*1.e0;
                                                B[3*in+2]= B[3*in+2] - AL[9*k+8]*1.e0;
                                                AL[9*k+2]= 0.e0;
                                                AL[9*k+5]= 0.e0;
                                                AL[9*k+8]= 0.e0;
                                    }
                        }
                        iS= indexU[in];
                        iE= indexU[in+1];

                        for(k=iS;k<=iE;k++){
                                    if (IWKX[itemU[k]][0] == 1 ) {
                                                B[3*in  ]= B[3*in  ] - AU[9*k+2]*1.e0;
                                                B[3*in+1]= B[3*in+1] - AU[9*k+5]*1.e0;
                                                B[3*in+2]= B[3*in+2] - AU[9*k+8]*1.e0;
                                                AU[9*k+2]= 0.e0;
                                                AU[9*k+5]= 0.e0;
                                                AU[9*k+8]= 0.e0;
                                    }
                        }
            }
```

1D

(Linear) Equation at x=0

$u_1 = 0$ (or $u_0 = 0$)

- Only deforms in x-direction (displacement: u)
  - Uniform: Sectional Area A, Young's Modulus E
  - Boundary Conditions (B.C.)
    - x=0 : u=0 (fixed)
    - x=x_{max} : F (axial force)
- Truss: NO bending deformation by G-force

1D

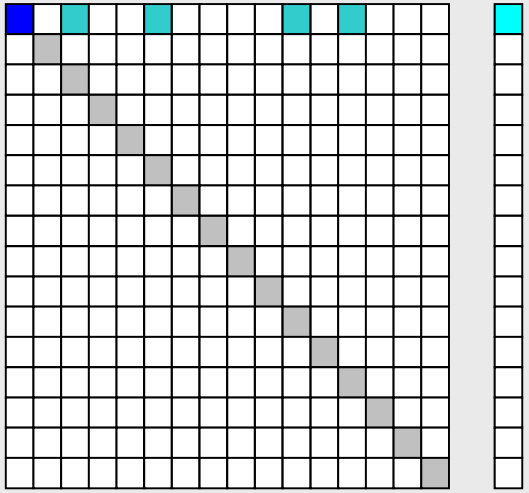
Program: 1d.c (6/7)

Boundary Conditions

```
/*
// +----------------------+
// | BOUNDARY conditions |
// +----------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= 0.0;

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}

/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

Diagonal Component=1
RHS=0
Off-Diagonal Components= 0.



1D

Program: 1d.c (6/7)

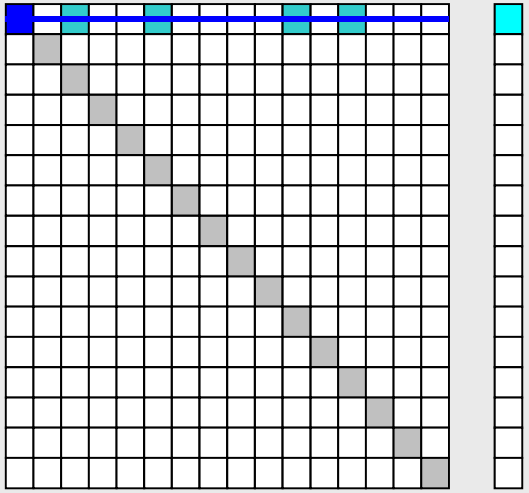
Boundary Conditions

```
/*
// +----------------------+
// | BOUNDARY conditions |
// +----------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= 0.0;

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}

/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

Diagonal Component=1
RHS=0
Off-Diagonal Components= 0.

Erase !



1D

Program: 1d.c (6/7)

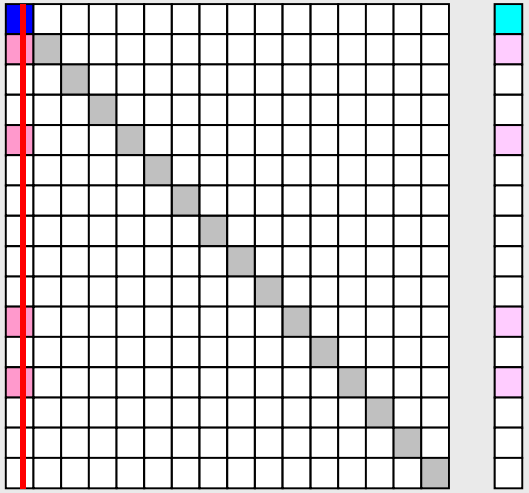
Boundary Conditions

```
/*
// +----------------------+
// | BOUNDARY conditions |
// +----------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= 0.0;

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}
/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

Diagonal Component=1
RHS=0
Off-Diagonal Components= 0.

Elimination and Erase



Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix (in this case just erase off-diagonal components)

1D

Program: 1d.c (6/7)

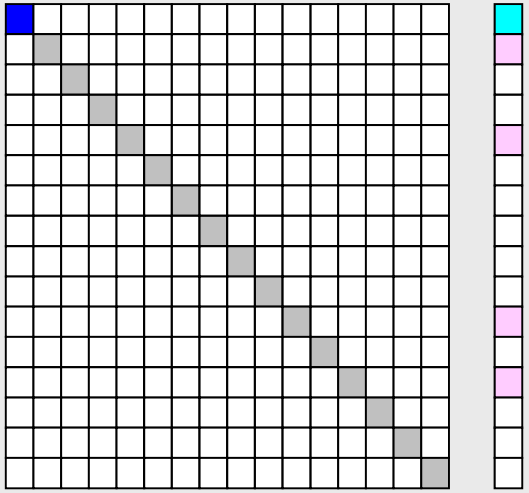
Boundary Conditions

```
/*
// +----------------------+
// | BOUNDARY conditions |
// +----------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= 0.0;

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}
/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

Diagonal Component=1
RHS=0
Off-Diagonal Components= 0.

Elimination and Erase



Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix (in this case just erase off-diagonal components)

1D

if $u_1 \neq 0$

```
/*
// +----------------------+
// | BOUNDARY conditions |
// +----------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= Umin;

          for(j=1;i<N;i++){
            for (k=Index[j];k<Index[j+1];k++){
              if(Item[k]==0){
                Rhs [j]= Rhs[j] - Amat[k]*Umin;
                AMat[k]= 0.0;
            }}
```

Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix.

$$Diag_j u_j + \sum_{k=Index[j]}^{Index[j-1]} Amat_k \, u_{Item[k]} = Rhs_j$$

1D

if $u_1 \neq 0$

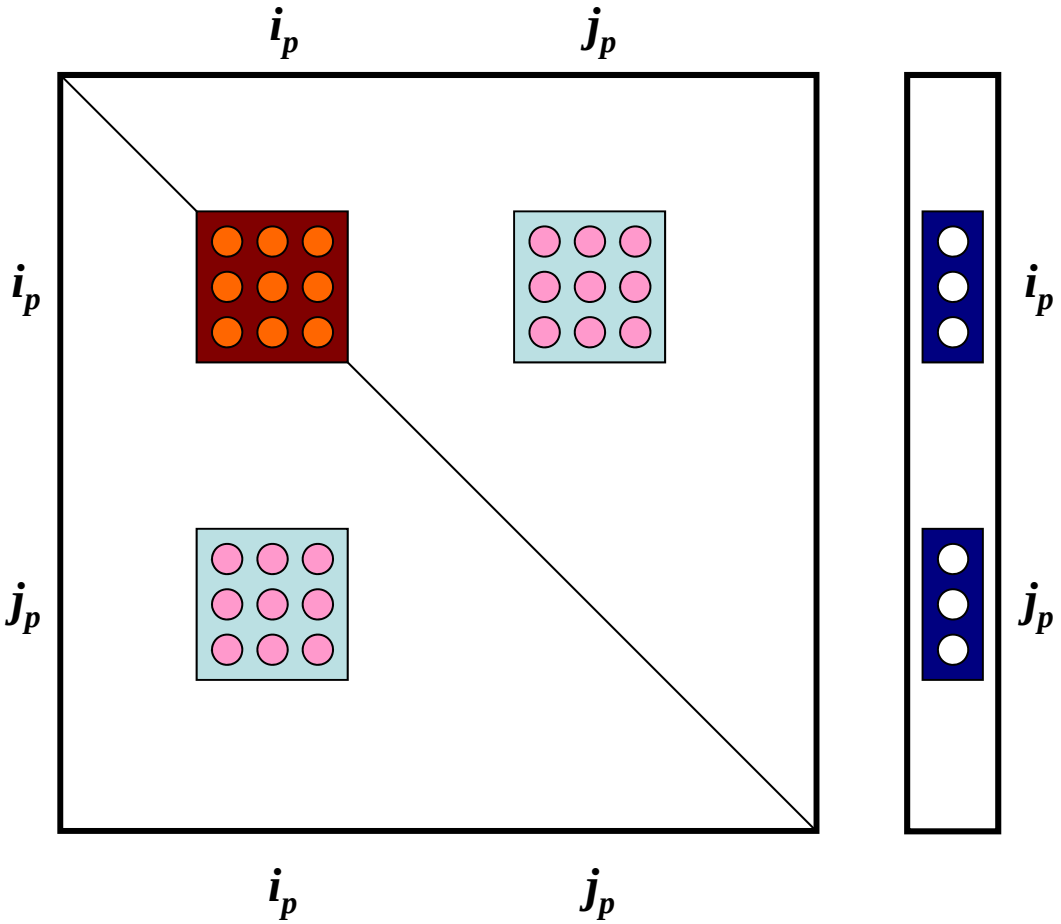
```
/*
// +----------------------+
// | BOUNDARY conditions |
// +----------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= Umin;

          for(j=1;i<N;i++){
            for (k=Index[j];k<Index[j+1];k++){
              if(Item[k]==0){
                Rhs [j]= Rhs[j] - Amat[k]*Umin;
                AMat[k]= 0.0;
            }}
```

Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix.

$$Diag_j u_j + \sum_{k=Index[j],k\neq k_s}^{Index[j-1]} Amat_k \, u_{Item[k]}$$

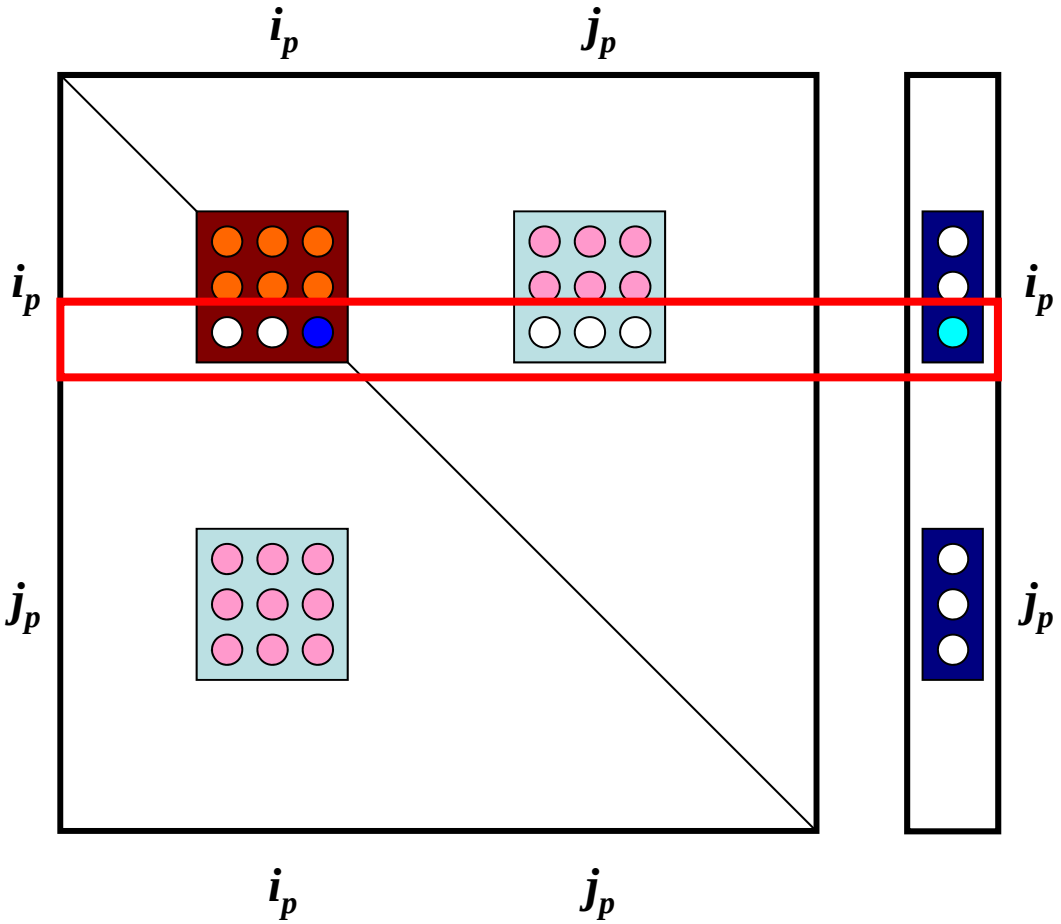
$$= Rhs_j - Amat_{k_s} u_{Item[k_s]} = Rhs_j - Amat_{k_s} u_{\min} \quad where\ Item[k_s] = 0$$

Global Matrix



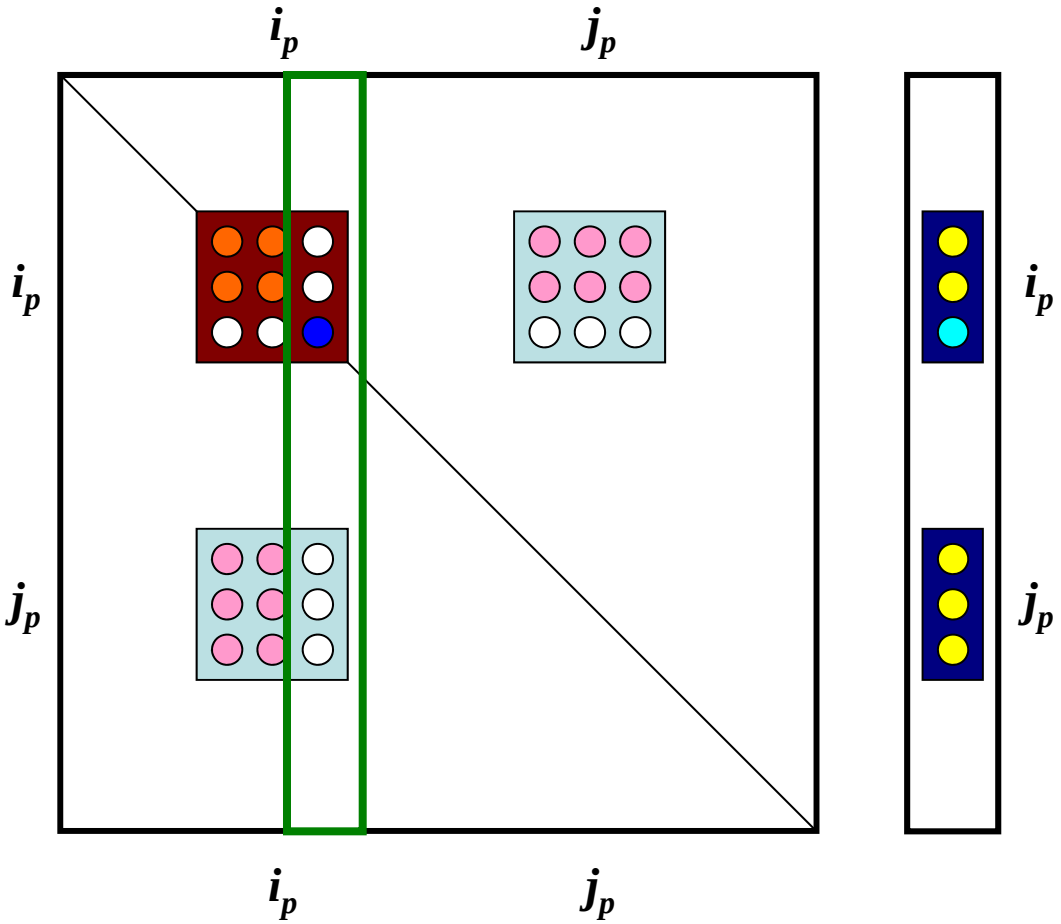
Same Procedure

Corresponding Row:
Diag. Component=1, Other Comp.=0



Same Procedure

Corresponding Column:
Move to RHS, Off-Diag. Component=0



MAT_ASS_BC (2/9)

```
/**
        Z=Zmax
**/
        for(in=0;in<N;in++) IWKX[in][0]=0;

        ib0=-1;

        for( ib0=0;ib0<NODGRPtot;ib0++){
                if( strcmp(NODGRP_NAME[ib0].name,"Zmax") == 0 ) break;
        }

        for( ib=NODGRP_INDEX[ib0];ib<NODGRP_INDEX[ib0+1];ib++){
                in=NODGRP_ITEM[ib];
                IWKX[in-1][0]=1;
        }

        for(in=0;in<N;in++){
                if( IWKX[in][0] == 1 ){
                        B[3*in  ]= B[3*in  ] - D[9*in+2]*
                        B[3*in+1]= B[3*in+1] - D[9*in+5]*
                        D[9*in+2]= 0.e0;
                        D[9*in+5]= 0.e0;
                        D[9*in+6]= 0.e0;
                        D[9*in+7]= 0.e0;
                        D[9*in+8]= 1.e0;
                        B[3*in+2]= 1.e0;
                        iS= indexL[in];
                        iE= indexL[in+1];
                        for(k=iS;k<iE;k++){
                                AL[9*k+6]= 0.e0;
                                AL[9*k+7]= 0.e0;
                                AL[9*k+8]= 0.e0;
                        }
                        iS= indexU[in];
                        iE= indexU[in+1];
                        for(k=iS;k<iE;k++){
                                AU[9*k+6]= 0.e0;
                                AU[9*k+7]= 0.e0;
                                AU[9*k+8]= 0.e0;
                        }
                }  }
```

If the node “in” is included in the node group “Zmax”

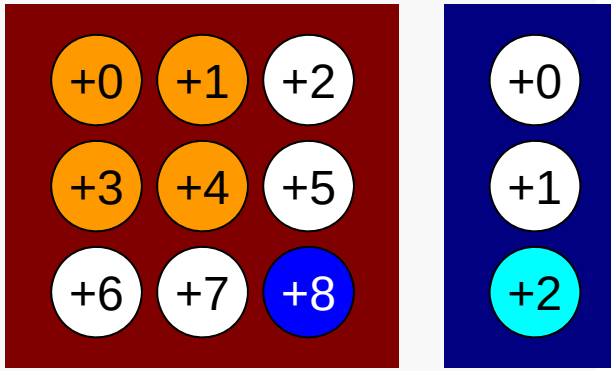
IWKX[in-1][0]= 1

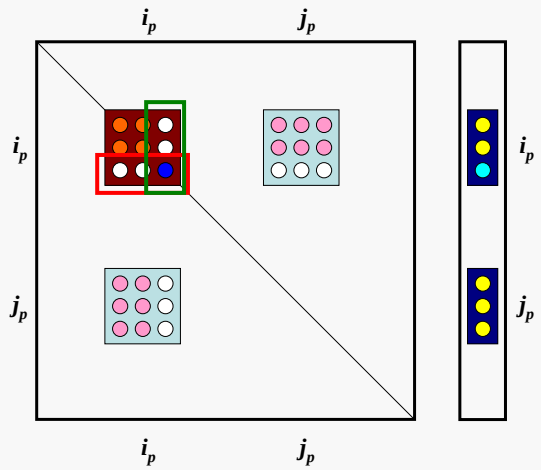
MAT_ASS_BC (2/9)

```
/**
        Z=Zmax
**/
        for(in=0;in<N;in++) IWKX[in][0]=0;

        ib0=-1;

        for( ib0=0;ib0<NODGRPtot;ib0++){
                if( strcmp(NODGRP_NAME[ib0].name,"Zmax") == 0 ) break;
        }

        for( ib=NODGRP_INDEX[ib0];ib<NODGRP_INDEX[ib0+1];ib++){
                in=NODGRP_ITEM[ib];
                IWKX[in-1][0]=1;
        }

        for(in=0;in<N;in++){
                if( IWKX[in][0] == 1 ){
                        B[3*in  ]= B[3*in  ] - D[9*in+2]*1.e0;
                        B[3*in+1]= B[3*in+1] - D[9*in+5]*1.e0;
                        D[9*in+2]= 0.e0;
                        D[9*in+5]= 0.e0;
                        D[9*in+6]= 0.e0;
                        D[9*in+7]= 0.e0;
                        D[9*in+8]= 1.e0;
                        B[3*in+2]= 1.e0;
                        iS= indexL[in];
                        iE= indexL[in+1];
                        for(k=iS;k<iE;k++){
                                AL[9*k+6]= 0.e0;
                                AL[9*k+7]= 0.e0;
                                AL[9*k+8]= 0.e0;
                        }
                        iS= indexU[in];
                        iE= indexU[in+1];
                        for(k=iS;k<iE;k++){
                                AU[9*k+6]= 0.e0;
                                AU[9*k+7]= 0.e0;
                                AU[9*k+8]= 0.e0;
                        }
                }  }
```

Modify:
B[3*in], B[3*in+1]

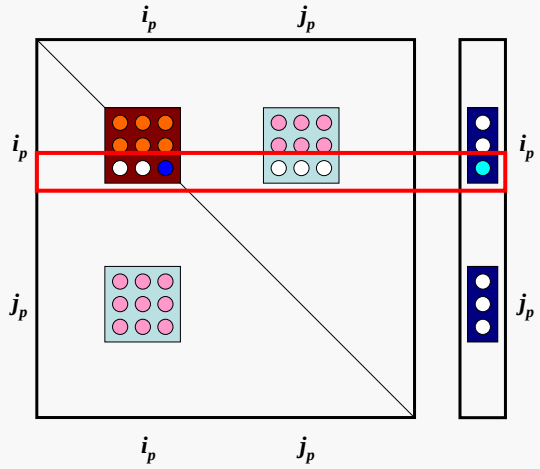
Then, clear:
D[9*in+6], D[9*in+7]

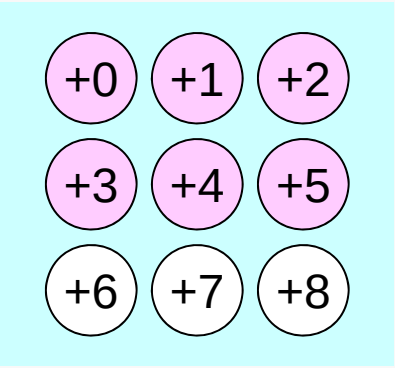




MAT_ASS_BC (2/9)

```
/**
        Z=Zmax
**/
        for(in=0;in<N;in++) IWKX[in][0]=0;

        ib0=-1;

        for( ib0=0;ib0<NODGRPtot;ib0++){
                if( strcmp(NODGRP_NAME[ib0].name,"Zmax") == 0 ) break;
        }

        for( ib=NODGRP_INDEX[ib0];ib<NODGRP_INDEX[ib0+1];ib++){
                in=NODGRP_ITEM[ib];
                IWKX[in-1][0]=1;
        }

        for(in=0;in<N;in++){
                if( IWKX[in][0] == 1 ){
                        B[3*in  ]= B[3*in  ] - D[9*in+2]*1.e0;
                        B[3*in+1]= B[3*in+1] - D[9*in+5]*1.e0;
                        D[9*in+2]= 0.e0;
                        D[9*in+5]= 0.e0;
                        D[9*in+6]= 0.e0;
                        D[9*in+7]= 0.e0;
                        D[9*in+8]= 1.e0;
                        B[3*in+2]= 1.e0;
                        iS= indexL[in];
                        iE= indexL[in+1];
                        for(k=iS;k<iE;k++){
                                AL[9*k+6]= 0.e0;
                                AL[9*k+7]= 0.e0;
                                AL[9*k+8]= 0.e0;
                        }
                        iS= indexU[in];
                        iE= indexU[in+1];
                        for(k=iS;k<iE;k++){
                                AU[9*k+6]= 0.e0;
                                AU[9*k+7]= 0.e0;
                                AU[9*k+8]= 0.e0;
                        }
                }  }
```

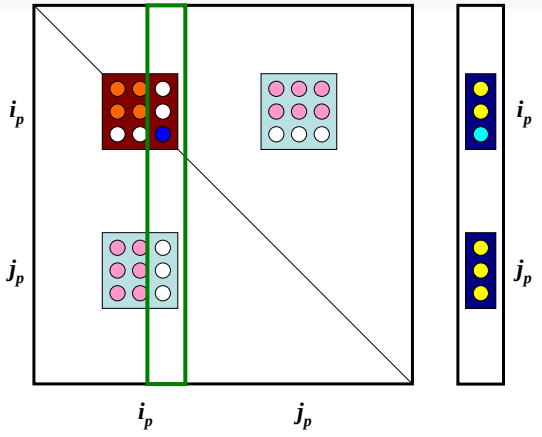


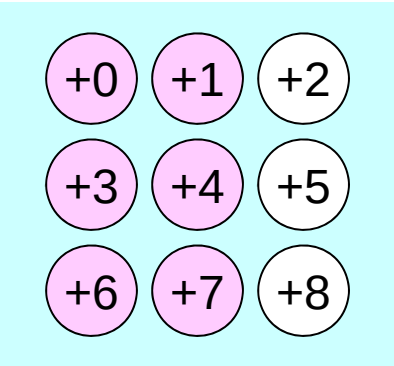


MAT_ASS_BC (3/9)

```
for(in=0;in<N;in++){
        iS= indexL[in];
        iE= indexL[in+1];
        for(k=iS;k<iE;k++){
                if (IWKX[itemL[k]][0] == 1 ) {
                        B[3*in  ]= B[3*in  ] - AL[9*k+2]*1.e0;
                        B[3*in+1]= B[3*in+1] - AL[9*k+5]*1.e0;
                        B[3*in+2]= B[3*in+2] - AL[9*k+8]*1.e0;
                        AL[9*k+2]= 0.e0;
                        AL[9*k+5]= 0.e0;
                        AL[9*k+8]= 0.e0;
                }
        }
        iS= indexU[in];
        iE= indexU[in+1];

        for(k=iS;k<iE;k++){
                if (IWKX[itemU[k]][0] == 1 ) {
                        B[3*in  ]= B[3*in  ] - AU[9*k+2]*1.e0;
                        B[3*in+1]= B[3*in+1] - AU[9*k+5]*1.e0;
                        B[3*in+2]= B[3*in+2] - AU[9*k+8]*1.e0;
                        AU[9*k+2]= 0.e0;
                        AU[9*k+5]= 0.e0;
                        AU[9*k+8]= 0.e0;
                }
        }
}
```

Modify RHS,
then clear AL and AU

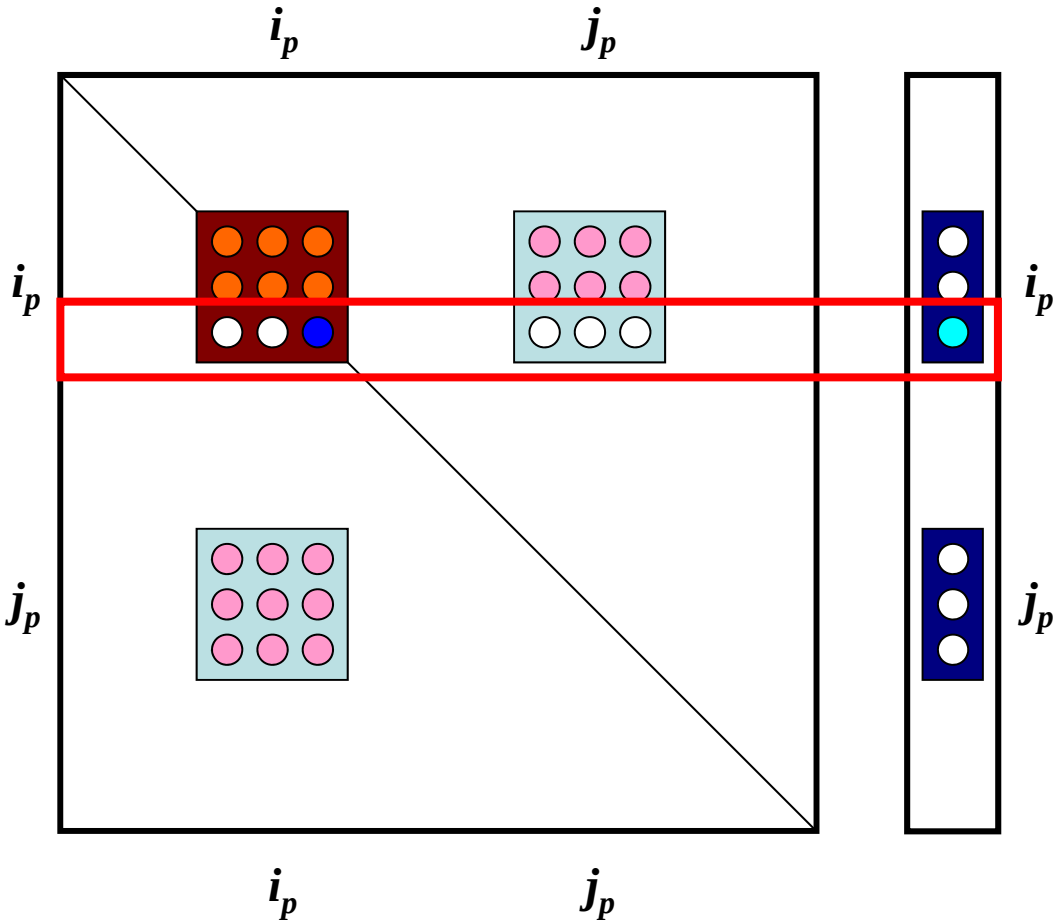




w=0@Z=Zmin

Corresponding Row:

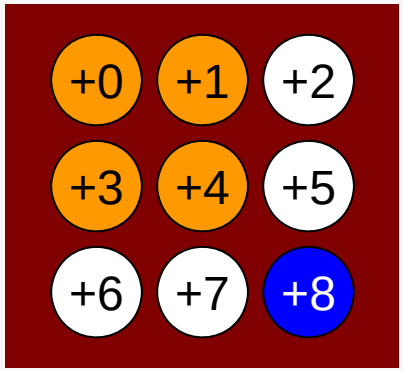
Diag. Component=1, Other Comp.=0

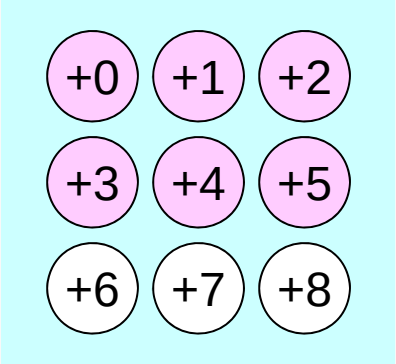


MAT_ASS_BC (4/9)

w=0@Zmin

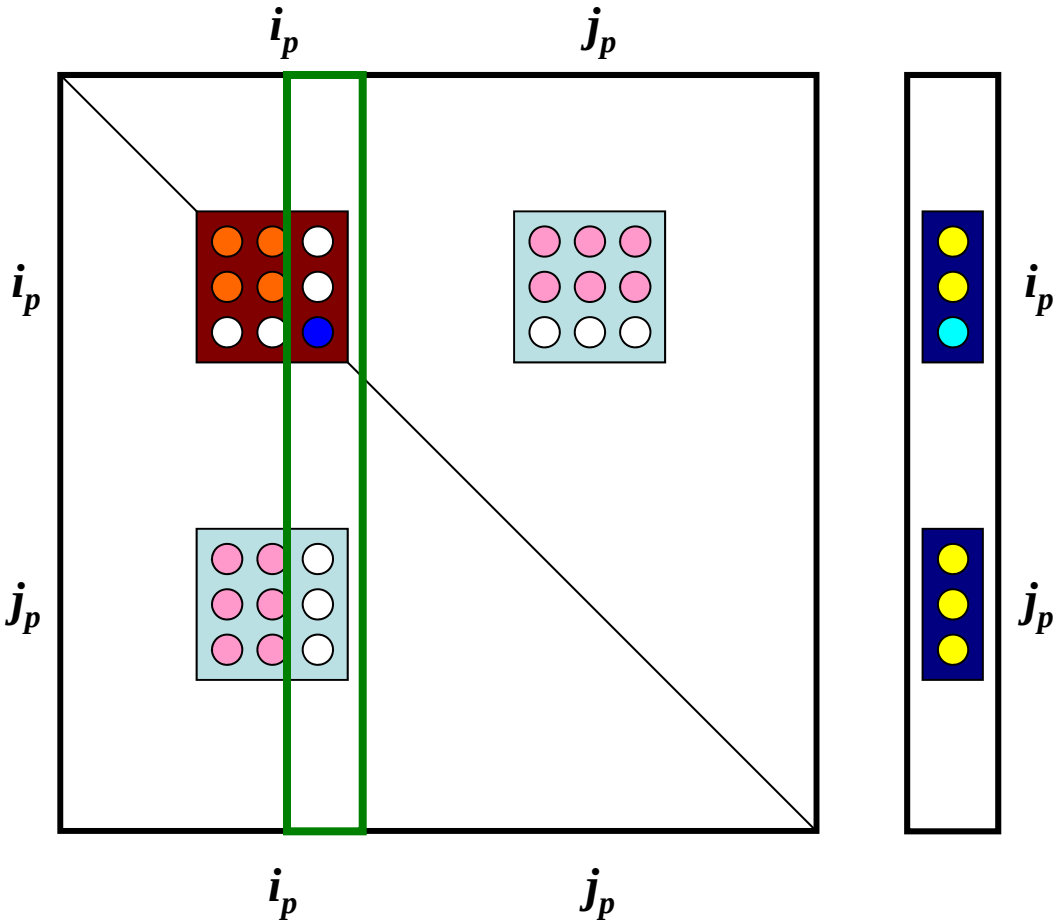
```
/**
        Z=Zmin
**/
        for(in=0;in<N;in++) IWKX[in][0]=0;

        ib0=-1;

        for( ib0=0;ib0<NODGRPtot;ib0++){
                if( strcmp(NODGRP_NAME[ib0].name,"Zmin") == 0 ) break;
        }

        for( ib=NODGRP_INDEX[ib0];ib<NODGRP_INDEX[ib0+1];ib++){
                in=NODGRP_ITEM[ib];
                IWKX[in-1][0]=1;
        }

        for(in=0;in<N;in++){
                if( IWKX[in][0] == 1 ){
                        D[9*in+2]= 0.e0;
                        D[9*in+5]= 0.e0;
                        D[9*in+6]= 0.e0;
                        D[9*in+7]= 0.e0;
                        D[9*in+8]= 1.e0;
                        B[3*in+2]= 0.e0;
                        iS= indexL[in];
                        iE= indexL[in+1];
                        for(k=iS;k<iE;k++){
                                AL[9*k+6]= 0.e0;
                                AL[9*k+7]= 0.e0;
                                AL[9*k+8]= 0.e0;
                        }
                        iS= indexU[in];
                        iE= indexU[in+1];
                        for(k=iS;k<iE;k++){
                                AU[9*k+6]= 0.e0;
                                AU[9*k+7]= 0.e0;
                                AU[9*k+8]= 0.e0;
                        }
                }
        }
```





w=0@Z=Zmin

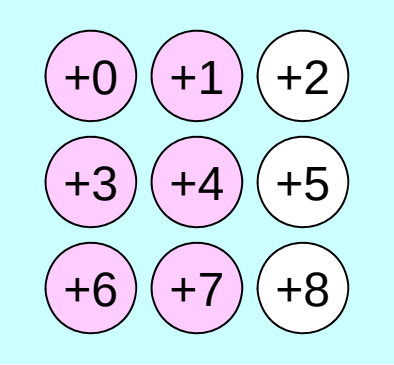
Corresponding Column:
Move to RHS, Off-Diag. Component=0



MAT_ASS_BC (5/9)

```
for(in=0;in<N;in++){
        iS= indexL[in];
        iE= indexL[in+1];
        for(k=iS;k<iE;k++){
                if (IWKX[itemL[k]][0] == 1 ) {
                        AL[9*k+2]= 0.e0;
                        AL[9*k+5]= 0.e0;
                        AL[9*k+8]= 0.e0;
                }
        }
        iS= indexU[in];
        iE= indexU[in+1];

        for(k=iS;k<iE;k++){
                if (IWKX[itemU[k]][0] == 1 ) {
                        AU[9*k+2]= 0.e0;
                        AU[9*k+5]= 0.e0;
                        AU[9*k+8]= 0.e0;
                }
        }
}
```

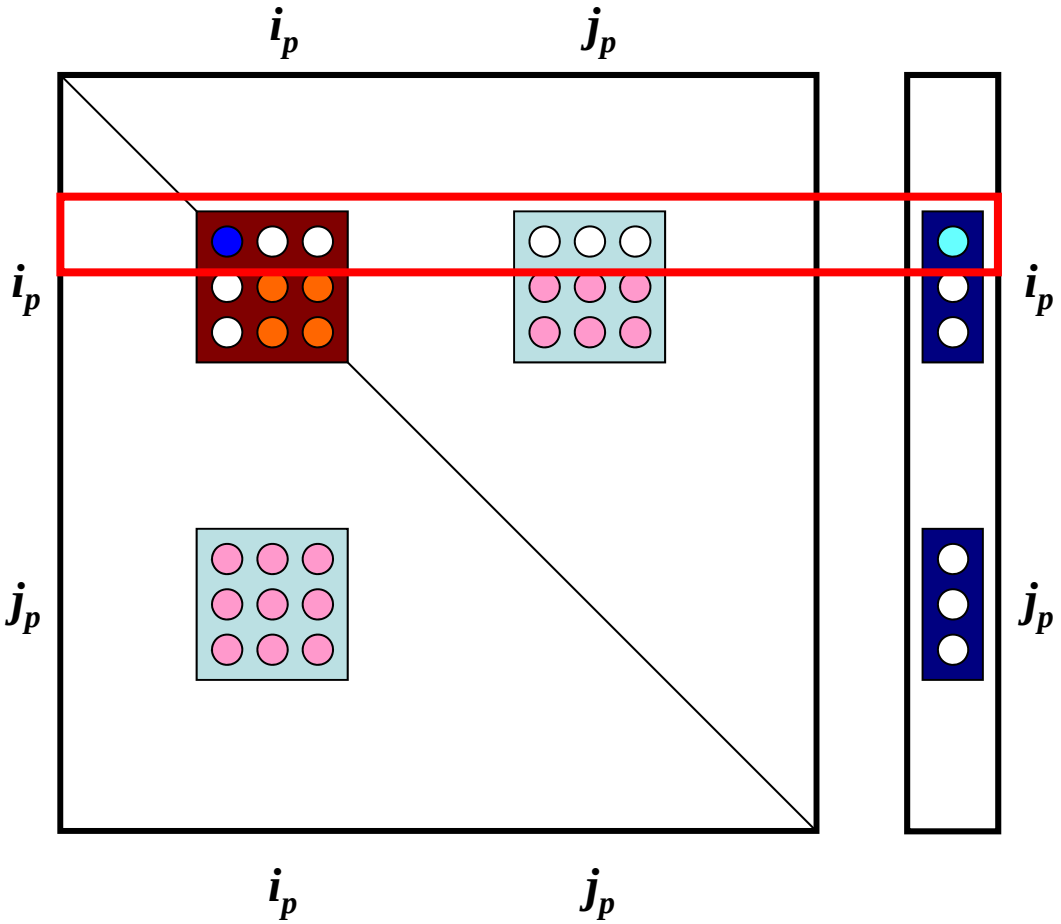
w=0@Zmin



u=0@X=Xmin

Corresponding Row:

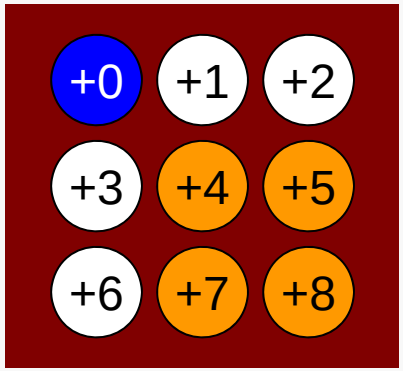
Diag. Component=1, Other Comp.=0

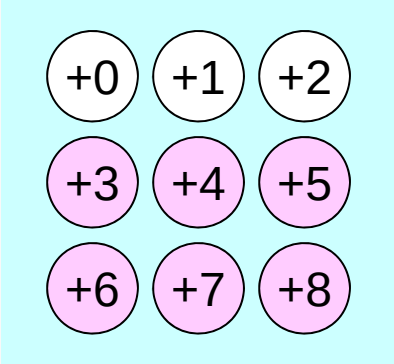


MAT_ASS_BC (6/9)

u=0@Xmin

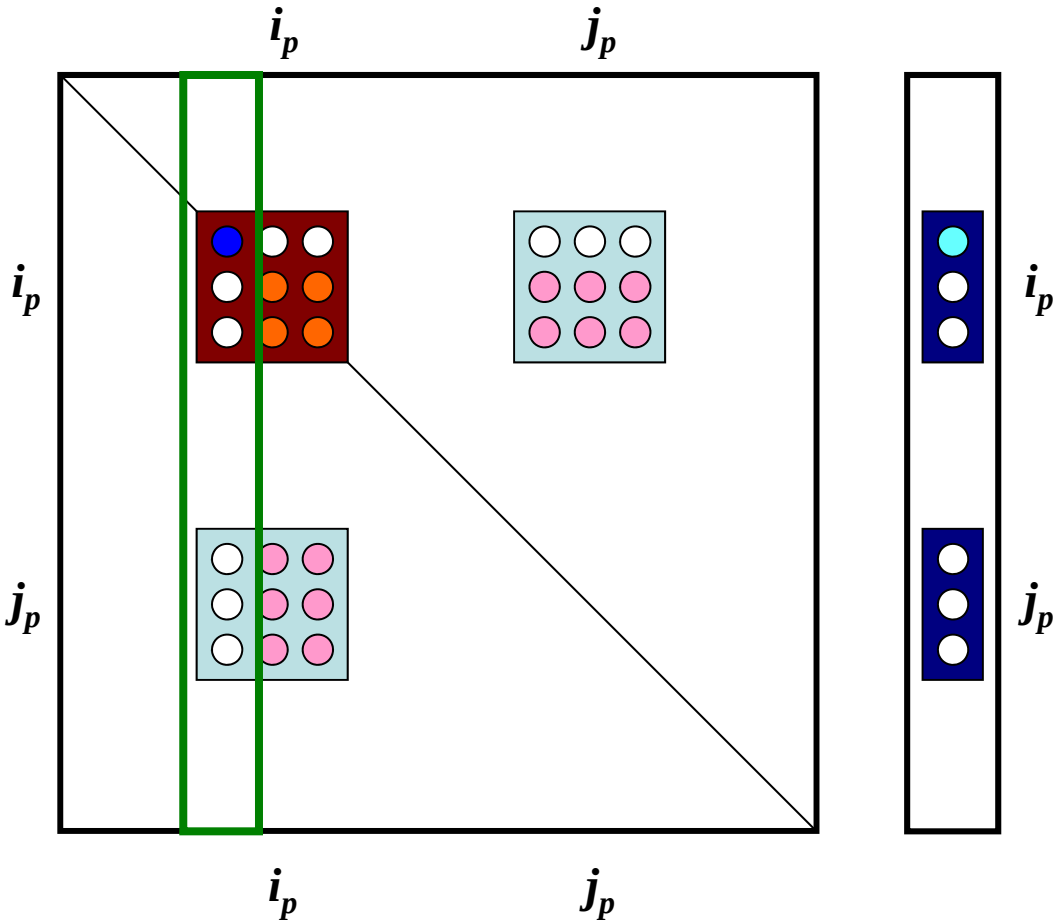
```
/**
        X=Xmin
**/
        for(in=0;in<N;in++) IWKX[in][0]=0;

        ib0=-1;

        for( ib0=0;ib0<NODGRPtot;ib0++){
                if( strcmp(NODGRP_NAME[ib0].name,"Xmin") == 0 ) break;
        }

        for( ib=NODGRP_INDEX[ib0];ib<NODGRP_INDEX[ib0+1];ib++){
                in=NODGRP_ITEM[ib];
                IWKX[in-1][0]=1;
        }

        for(in=0;in<N;in++){
                if( IWKX[in][0] == 1 ){
                        D[9*in  ]= 1.e0;
                        D[9*in+1]= 0.e0;
                        D[9*in+2]= 0.e0;
                        D[9*in+3]= 0.e0;
                        D[9*in+6]= 0.e0;
                        B[3*in  ]= 0.e0;

                        iS= indexL[in];
                        iE= indexL[in+1];
                        for(k=iS;k<iE;k++){
                                AL[9*k  ]= 0.e0;
                                AL[9*k+1]= 0.e0;
                                AL[9*k+2]= 0.e0;
                        }

                        iS= indexU[in];
                        iE= indexU[in+1];
                        for(k=iS;k<iE;k++){
                                AU[9*k  ]= 0.e0;
                                AU[9*k+1]= 0.e0;
                                AU[9*k+2]= 0.e0;
                        }
                }  }
```





u=0@X=Xmin

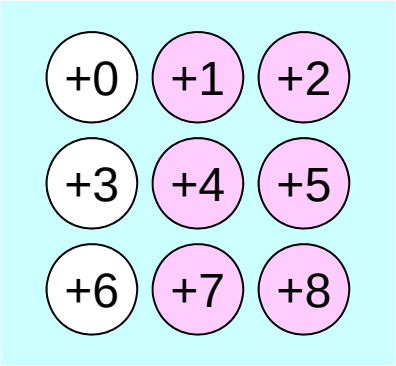
Corresponding Column:
Move to RHS, Off-Diag. Component=0



MAT_ASS_BC (7/9)

```
for(in=0;in<N;in++){
        iS= indexL[in];
        iE= indexL[in+1];
        for(k=iS;k<iE;k++){
                if (IWKX[itemL[k]][0] == 1 ) {
                        AL[9*k  ]= 0.e0;
                        AL[9*k+3]= 0.e0;
                        AL[9*k+6]= 0.e0;
                }
        }

        iS= indexU[in];
        iE= indexU[in+1];
        for(k=iS;k<iE;k++){
                if (IWKX[itemU[k]][0] == 1 ) {
                        AU[9*k  ]= 0.e0;
                        AU[9*k+3]= 0.e0;
                        AU[9*k+6]= 0.e0;
                }
        }
}
```

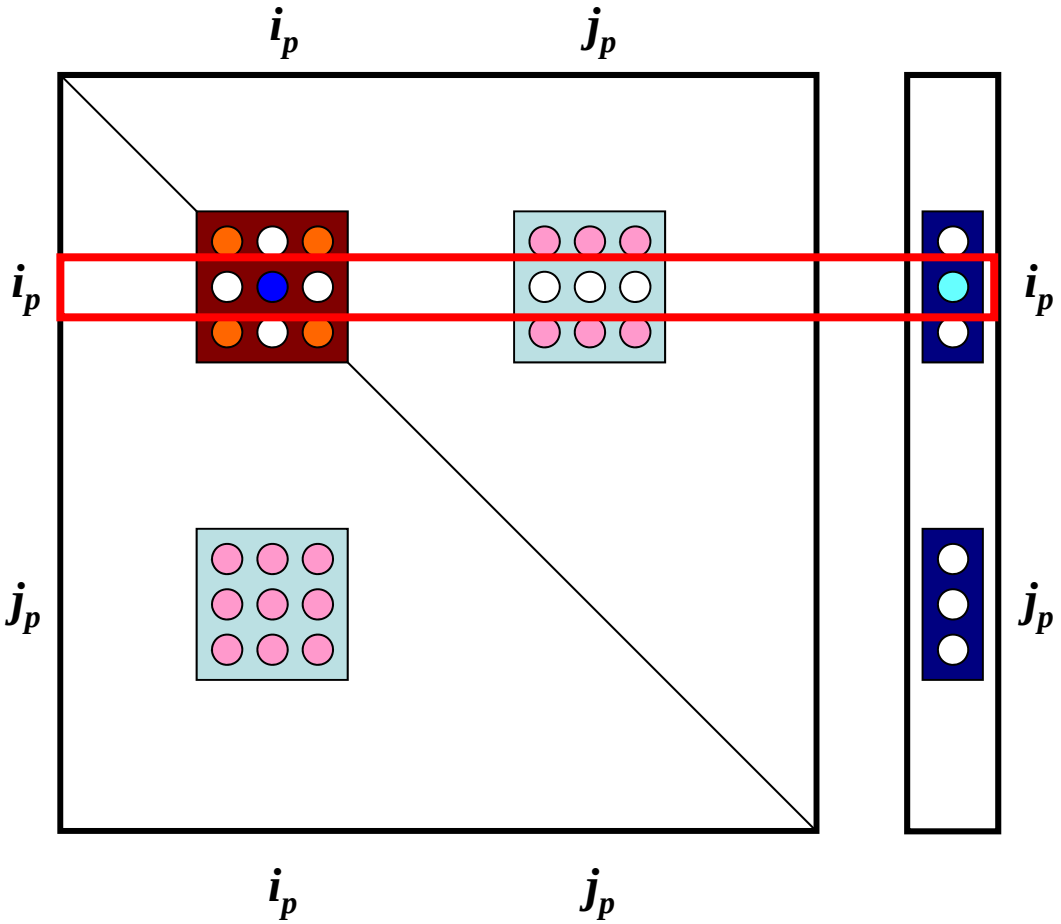
u=0@Xmin



v=0@Y=Ymin

Corresponding Row:

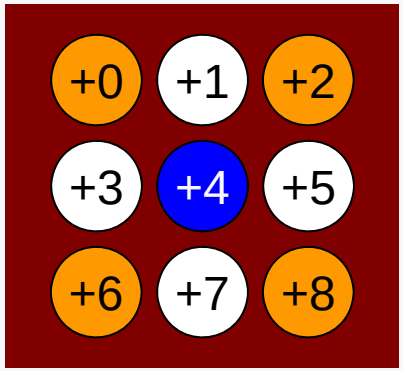
Diag. Component=1, Other Comp.=0

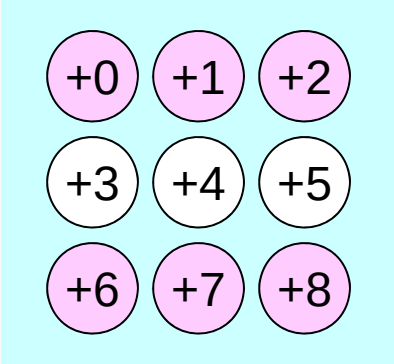


MAT_ASS_BC (8/9)

v=0@Ymin

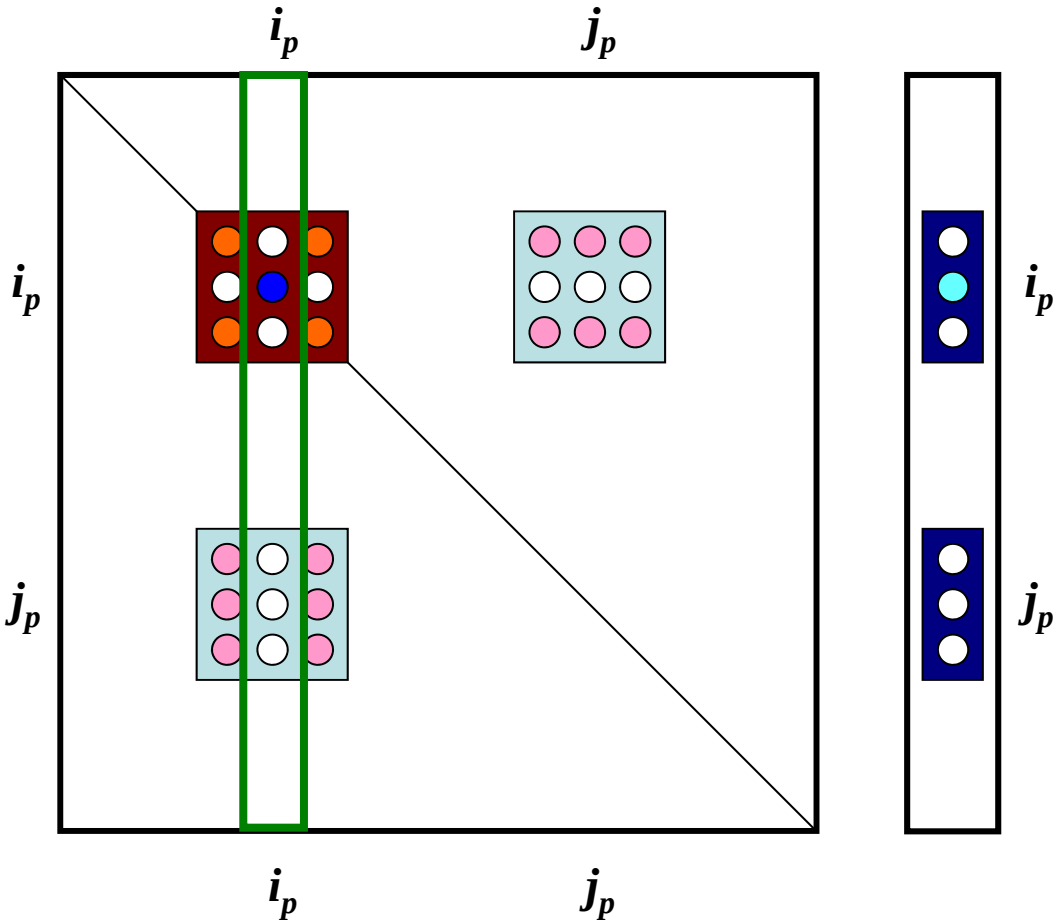
```
/**
        Y=Ymin
**/
        for(in=0;in<N;in++) IWKX[in][0]=0;

        ib0=-1;

        for( ib0=0;ib0<NODGRPtot;ib0++){
                if( strcmp(NODGRP_NAME[ib0].name,"Ymin") == 0 ) break;
        }


        for( ib=NODGRP_INDEX[ib0];ib<NODGRP_INDEX[ib0+1];ib++){
                in=NODGRP_ITEM[ib];
                IWKX[in-1][0]=1;
        }

        for(in=0;in<N;in++){
                if( IWKX[in][0] == 1 ){
                        D[9*in+1]= 0.e0;
                        D[9*in+4]= 1.e0;
                        D[9*in+7]= 0.e0;
                        D[9*in+3]= 0.e0;
                        D[9*in+5]= 0.e0;
                        B[3*in+1]= 0.e0;

                        iS= indexL[in];
                        iE= indexL[in+1];
                        for(k=iS;k<iE;k++){
                                AL[9*k+3]= 0.e0;
                                AL[9*k+4]= 0.e0;
                                AL[9*k+5]= 0.e0;
                        }

                        iS= indexU[in];
                        iE= indexU[in+1];
                        for(k=iS;k<iE;k++){
                                AU[9*k+3]= 0.e0;
                                AU[9*k+4]= 0.e0;
                                AU[9*k+5]= 0.e0;
                        } } }
```





v=0@Y=Ymin

Corresponding Column:
Move to RHS, Off-Diag. Component=0



MAT_ASS_BC (9/9)

```
        for(in=0;in<N;in++){
                iS= indexL[in];
                iE= indexL[in+1];
                for(k=iS;k<iE;k++){
                        if (IWKX[itemL[k]][0] == 1 ) {
                                AL[9*k+1]= 0.e0;
                                AL[9*k+4]= 0.e0;
                                AL[9*k+7]= 0.e0;
                        }
                }

                iS= indexU[in];
                iE= indexU[in+1];
                for(k=iS;k<iE;k++){
                        if (IWKX[itemU[k]][0] == 1 ) {
                                AU[9*k+1]= 0.e0;
                                AU[9*k+4]= 0.e0;
                                AU[9*k+7]= 0.e0;
                        }
                }
        }

}
```

v=0@Ymin