SPACE では、３つの座標系を用いている。本節では、この３つの座標系がどのように使用されているか、また、どのように座標変換を行っているかについて述べる。まず、SPACE で使用している３つの座標系の名称を、

１．　部材座標系
２．　全体座標系
３．　釣合座標系

としよう。これら３つの座標系の関係が、図 2-1 に示されている。

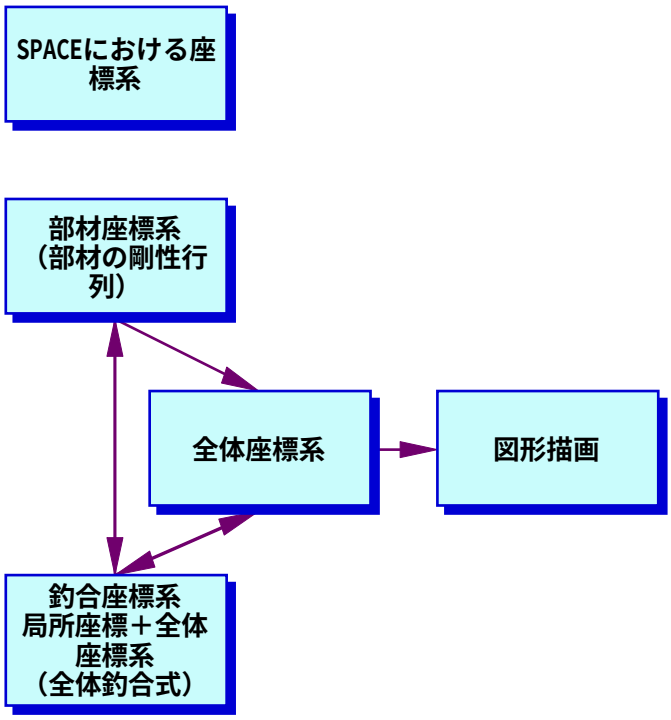


図 2-1 SPACE における座標系

最初の部材座標系とは、各部材モデルで定義した座標系であり、各モデルで異なる場合がある。特殊なモデル以外は、部材の i 端を原点にして、右手右ネジの法則に従う直行座標系が用いられている。具体的には、部材の軸方向がｘ方向、他の 2 つは部材断面の主軸方向にｙ、ｚ方向がとられている。また、節点の自由度は、6 であり、一般の部材モデルの自由度は 2 節点で 12 自由度となっている。

構造物全体で、自由度の方向を合わせるために、ひとつの座標系を用いる。これを全体座標系と呼ぶ。ここでも、右手右ネジの法則に従う直行座標系が用いられている。この座標系は、構造物の節点を定義する場合、自然に用いられている座標系である。節点での変形の適合と力の釣

合を取るために、部材座標系からこの全体座標系に、変位ベクトルや応力ベクトルを変換しなければならない。また、剛性行列も座標変換する必要がある。したがって、各部材について、その部材座標系から全体座標系に変換する座標変換行列を作成することになる。

変位や応力を図形表示する場合、部材座標系では情報を的確に伝達することが難しい。そこで、全ての物理量を全体座標に変換して図形表示を行う。また、プレゼンターのために全体座標に変換したデータをファイルに出力する。

解析モデルによっては、局所座標系を用いなければならない場合もある。例えば、境界条件が全体座標系では表せない場合などがある。そこで、SPACE では、この局所座標系を含んだ全体座標系を釣合座標系とし、この座標系を用いて構造系全体の釣合式を立てることになる。そのため、釣合座標系で求めた節点変位などは、図形描画するためには全体座標系に座標変換しなければならない。無論、局所座標系を用いていないモデルでは、全体座標系と釣合座標系は同一である。

ここでは、SPACE で使用しているサブルーチンを例にして座標変換行列の作成とその使用法について述べる。最初に、構造体 Point と Member を用いて、部材両端の自由度を部材座標系から全体座標系に変換するサブルーチン Get_rotete_all()、節点自由度を全体座標系から局所座標系に変換するサブルーチン Get_rot_local()、最後に両者の変換行列から部材座標系から釣合座標系に変換するサブルーチン Get_rotate()について説明する。まず、submain.f の中で次のように続けてサブルーチンコールが行われる。

```
c――――――――――――――――――――――――――――――★座標変換行列計算
      call Get_rotate_all(rot_memb_t,Parameter_C,Point,Member)
c
c――――――――――――――――――――――――――――――★局所座標変換行列計算
      call Get_rot_local(rot_local,Parameter_C,Point)
c
c――――――――――――――――――――――――――――――★釣合座標系標変換行列計算
      call Get_rotate(rot_memb,rot_memb_t,rot_local,
     *                Parameter_C,Member)
```

以上の３つのサブルーチンとそれに付随するサブルーチンを以下に示す。

```
C     ――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Get_rotate_all
C       ――――――――――――――――――――――――――――――――
C     ●   全体座標系標変換行列計算(ok)
C     ――――――――――――――――――――――――――――――――――
      subroutine Get_rotate_all(rot_memb_t,Parameter_C,Point,Member)
```

```
      implicit real*8(A-H,O-Z)
      include "submain.h"
      record /parameter_s / Parameter_C
      record / member_s   / Member
      record / point_s    / Point
      dimension Member(*),Point(*)
      dimension rot_memb_t(3,3,*)
C     ―――――――――――――――――――――――――――――――――――――
c           rot_memb_t     real*8   全体座標系への座標変換行列
c           Parameter_C          structure
c           Point                structure
c           Member               structure
C     ―――――――――――――――――――――――――――――――――――――
c      write(76,*) '部材回転行列:',Parameter_C.n_member
      n_member = Parameter_C.n_member
      do i=1,n_member                                                  ! 1
      i1 = Member(i).nm_point(1)                                       ! 2
      i2 = Member(i).nm_point(2)
      r1 = Point(i2).coord(1) - Point(i1).coord(1)                     ! 3
      r2 = Point(i2).coord(2) - Point(i1).coord(2)
      r3 = Point(i2).coord(3) - Point(i1).coord(3)
      call rot_member(r1,r2,r3,ty,tz)                                  ! 4
      tx = Member(i).rot_x                                             ! 5
      call rotsb(tx,ty,tz,rot_memb_t(1,1,i))                           ! 6
      end do
      return
      end
C     ―――――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /rotsb
C       ―――――――――――――――――――――――――――――――――――
C     ●   立体線材部材の回転行列作成(ok)
C     ―――――――――――――――――――――――――――――――――――――
      subroutine rotsb(tx,ty,tz,rot)
      implicit real*8(A-H,O-Z)
      data PAIX/0.0174532/
      dimension rot(3,3)
C     ―――――――――――――――――――――――――――――――――――――
c        TX     :real*8  各部材のＸ軸回りの回転角
c        TY     :real*8  各部材のＹ軸回りの回転角
c        TZ     :real*8  各部材のＺ軸回りの回転角
c        rot    :real*8  回転行列
c       PAIX    :        Π/360.
C     ―――――――――――――――――――――――――――――――――――――
      X =tx*PAIX                                                       ! 7
      Y =ty*PAIX
      Z =tz*PAIX
      STZ=DSIN(Z )                                                     ! 8
      CTZ=DCOS(Z )
      STY=DSIN(Y )
      CTY=DCOS(Y )
      STX=DSIN(X )
      CTX=DCOS(X )
      rot(1,1)=CTY*CTZ                                                 ! 9
      rot(1,2)=CTY*STZ
```

```
      rot(1,3)=-STY
      rot(2,1)=STX*STY*CTZ-CTX*STZ
      rot(2,2)=STX*STY*STZ+CTX*CTZ
      rot(2,3)=STX*CTY
      rot(3,1)=CTX*STY*CTZ+STX*STZ
      rot(3,2)=CTX*STY*STZ-STX*CTZ
      rot(3,3)=CTX*CTY
      return
      end
C     ――――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /rot_member
C       ――――――――――――――――――――――――――――――――――
C     ●   部材の回転角度計算(ok)
C     ――――――――――――――――――――――――――――――――――――
      subroutine rot_member(r1,r2,r3,TY,TZ)
      implicit real*8(A-H,O-Z)
      data  PAI/3.14159265/
      RL2=(r1**2+r2**2)                                          ! 10
      R12=DSQRT(RL2)
      if(r2.ne.0.0) then                                         ! 11
      if(r2.gt.0.0) then                                         ! 12
      TZ =(PAI/2.-(DATAN(r1/r2)))*180./PAI
      else
      TZ =(-PAI/2.0-(DATAN(r1/r2)))*180./PAI                     ! 13
      end if
      TY =-(DATAN(r3/R12))*180./PAI                              ! 14
      else
      TZ =0.0                                                    ! 15
      if(r1.lt.0.0) TZ =180.0
      if(r1.ne.0.0) then                                         ! 16
      TY =-(DATAN(r3/R12))*180./PAI
      else
      TY =90.0                                                   ! 17
      if(r3.gt.0.0) TY =-90.0
      end if
      end if
      return
      end
C     ――――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Get_rot_local
C       ――――――――――――――――――――――――――――――――――
C     ●   局所座標系 変換行列計算(ok)
C     ――――――――――――――――――――――――――――――――――――
      subroutine Get_rot_local(rot_local,Parameter_C,Point)
      implicit real*8(A-H,O-Z)
      include "submain.h"
      record /parameter_s / Parameter_C
      record / point_s    / Point
      dimension Point(*)
      dimension rot_local(3,3,*)
C     ――――――――――――――――――――――――――――――――――――
c           rot_local      :real*8   局所座標系への座標変換行列
c           Parameter_C    :structure
c           Point          :structure
```

```
C     ―――――――――――――――――――――――――――――――――――――
c      write(76,*) '部材回転行列:',Parameter_C.n_point
      if(Parameter_C.n_local_coord.eq.0) return                          ! 18
      n_point = Parameter_C.n_point
      do i=1,n_point                                                     ! 19
      i1 = Point(i).local_coord                                          ! 20
      if(i1.ne.0) then                                                   ! 21
      tx = Point(i).coord_local(1)
      ty = Point(i).coord_local(2)
      tz = Point(i).coord_local(3)
      call rotsb_local(tx,ty,tz,rot_local(1,1,i1))                       ! 22
      endif
      end do
      return
      end
C     ―――――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /rotsb_local
C       ―――――――――――――――――――――――――――――――――――
C     ●   立体線材部材の局所座標系回転行列作成(ok)
C     ―――――――――――――――――――――――――――――――――――――
      subroutine rotsb_local(tx,ty,tz,rot)
      implicit real*8(A-H,O-Z)
      data PAIX/0.0174532/
      dimension rot(3,3)
C     ―――――――――――――――――――――――――――――――――――――
c        TX     :real*8  各部材のＸ軸回りの回転角
c        TY     :real*8  各部材のＹ軸回りの回転角
c        TZ     :real*8  各部材のＺ軸回りの回転角
c        rot    :real*8  回転行列
c       PAIX    :        Π/360.
C     ―――――――――――――――――――――――――――――――――――――
      X =-tx*PAIX                                                        ! 23
      Y =-ty*PAIX
      Z =-tz*PAIX
      STZ=DSIN(Z )                                                       ! 24
      CTZ=DCOS(Z )
      STY=DSIN(Y )
      CTY=DCOS(Y )
      STX=DSIN(X )
      CTX=DCOS(X )
      rot(1,1)=CTY*CTZ                                                   ! 25
      rot(1,2)=CTY*STZ
      rot(1,3)=-STY
      rot(2,1)=STX*STY*CTZ-CTX*STZ
      rot(2,2)=STX*STY*STZ+CTX*CTZ
      rot(2,3)=STX*CTY
      rot(3,1)=CTX*STY*CTZ+STX*STZ
      rot(3,2)=CTX*STY*STZ-STX*CTZ
      rot(3,3)=CTX*CTY
      return
      end
C     ―――――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Get_rotate
C       ―――――――――――――――――――――――――――――――――――
```

```
C     ●   釣合座標系座標変換行列計算(ok)
C     ―――――――――――――――――――――――――――――――――――
      subroutine Get_rotate(rot_memb,rot_memb_t,rot_local,
     *               Parameter_C,Member)
      implicit real*8(A-H,O-Z)
      include "submain.h"
      record /parameter_s / Parameter_C
      record / member_s   / Member
      dimension Member(*)
      dimension rot_memb_t(3,3,*),rot_memb(3,3,2,*),rot_local(3,3,*)
C     ―――――――――――――――――――――――――――――――――――
c           rot_memb       :real*8   釣合系への座標変換行列
c           rot_memb_t     :real*8   全体座標系への座標変換行列
c           rot_local      :real*8   局所座標系への座標変換行列
c           Parameter_C    :structure
c           Member         :structure
C     ―――――――――――――――――――――――――――――――――――
c      write(76,*) '部材回転行列:',Parameter_C.n_member
      n_member = Parameter_C.n_member                                  ! 26
      do i=1,n_member
      do j=1,2
      ij=Member(i).nm_local_coord(j)                                   ! 27
      if(ij.eq.0) then                                                ! 28
      do ii=1,3
      do jj=1,3
      rot_memb(jj,ii,j,i)=rot_memb_t(jj,ii,i)
      end do
      end do
      else
      call mult_m(rot_memb_t(1,1,i),rot_local(1,1,ij),                 ! 29
     *            rot_memb(1,1,j,i))
      end if
      end do
      end do
      return
      end
C     ―――――――――――――――――――――――――――――――――――
C     ●   FUNCTION /mult_m
C       ―――――――――――――――――――――――――――――――――――
C     ●   行列の掛け算(ok)
C     ―――――――――――――――――――――――――――――――――――
      subroutine mult_m(r1,r2,rr)
      implicit real*8(A-H,O-Z)
      dimension r1(3,3),r2(3,3),rr(3,3)
      do i=1,3
      do j=1,3
      sum=0.0
      do k=1,3
      sum=sum+r1(i,k)*r2(k,j)
      end do
      rr(i,j)=sum
      end do
      end do
      return
```

```
end
```

上記のサブルーチンの説明は、プログラムコード右のコメント番号にしたがって行われる。

1. 全部材について以下の処理を行い、部材座標系から全体座標系に変換する変換行列を求める。
2. 部材両端の節点番号を取得する。
3. 部材３方向の長さを計算する。
4. サブルーチン rot_member()により、全体座標系における部材角度 ty、tz を求める。
5. ユーザー定義の部材 x 軸方向の回転 tx をセットする。
6. サブルーチン rotsb()により、部材の回転行列を求める。
7. サブルーチン rotsb()では、まず、３方向の回転角度（度）をラジアンに変換する。
8. ３方向の SIN 及び COS を求める。
9. 回転行列 rot を求める。
10. サブルーチン rot_member()では、まず、部材の y 方向と z 方向の部材長さ成分 R12 を求める。
11. 部材の長さで y 方向成分がある場合、以下の処理を行う。y 方向成分がない場合、処理 15 に制御を移す。
12. 部材の長さで y 方向成分が正の場合、角度 TZ（度）を求める。
13. 部材の長さで y 方向成分が負の場合、角度 TZ（度）を求める。
14. 角度 TY（度）を求める。
15. 部材の y 方向成分がない場合で、x 方向成分が負の場合は、TZ=180.とし、その他は、TZ=0.とする。
16. 部材の x 方向成分がある場合は、TY を求める。
17. 部材の x 方向成分がない場合は、z 方向成分が正の場合は、TY=-90.とし、その他は、TY=90.とする。

部材角度とは、解析モデルの中で、該当する部材が全体座標でどのような位置におかれているかを表す角度であり、全体座標系と部材座標系との間の角度でもある。なお、x 軸の角度は断面主軸が所定の方向に向いていない場合に、ユーザーが回転量を指定する。

このサブルーチン rot_member()は、全体座標における部材の角度を求めるプログラムであるが、結構複雑となっている。流れ図などを用いて理解されると良い。

18. サブルーチン Get_rot_local()は、節点が局所座標系で定義されている場合、全体座標系から局所座標系へ変換する行列を求めるプログラムである。ここでは、まず、局所座標系を有する節点の数を構

造体から取得し、その値をチェックする。それがゼロの場合、直ちにこのサブルーチンから戻ることになる。
19. 全節点について以下の処理を行う。
20. 当該の節点における局所座標番号を取得する。
21. その番号がゼロ以外の場合、この節点は局所座標系を使用していることになり、３方向の全体座標からの角度 tx,ty,tz を取得する。
22. サブルーチン rotsb_local()をコールして、座標変換行列を求める。
23. サブルーチン rotsb_local()において、最初に、角度（度）をラジアンに変換する。
24. この値から、３方向の SIN と COS を求める。
25. 座標変換行列 rot を求める。
26. このサブルーチン Get_rotate()では、部材座標系から部材両端の釣合座標系への変換行列 rot_memb を求める。最初に、構造体より部材数を取得し、以下の処理をこの部材数及び両端について行う。
27. 部材 i で部材端 j の局所座標番号を取得する。
28. この番号がゼロの場合、この節点では局所座標系を使用していないので、全体座標系への変換行列をそのまま rot_memb にコピーする。
29. 節点が局所座標系を使用している場合は,サブルーチン mult_m()を用いて、全体座標系への変換行列と局所座標系への変換行列との行列積をとり、その結果を釣合座標系への変換行列にセットする。

以上が、部材及び節点の座標変換行列の作成処理である。

次に、この座標変換行列がどのように使用されるかについて述べる。まず、次に示すように、部材座標系で作成した剛性行列を釣合座標系に変換する処理について考えてみよう。線形の剛性行列は、サブルーチン Cal_stiff_linear()で作られ、サブルーチン Rotate_stiffness()で釣合座標系に変換される。変換式は以下のようである。

$$\left[\bar{k}\right]=\left[R^{T}\right]\left[k\right]\left[R\right] \qquad \cdots\cdots\cdots\cdots\cdots\cdots(2.1)$$

ここで、$\left[k\right]$は部材座標系での剛性行列であり、$\left[\bar{k}\right]$は釣合座標系での剛性行列である。また、$\left[R\right]$は変換行列である。ただし、変換行列は、一般の部材では、両端の自由度から次式で与えられる。ここで、$\left[R_1\right]$は、i 節点での座標変換行列であり、$\left[R_2\right]$は j 節点での座標変換行列である。

$$[R]=\begin{bmatrix}[R_1] & & & \\ & [R_1] & & \\ & & [R_2] & \\ & & & [R_2]\end{bmatrix} \qquad \cdots\cdots\cdots(2.2)$$

上式を用いると、釣合座標系における剛性行列は、以下のように表すことができる。

$$[\bar{k}]=\begin{bmatrix}[R^T_1][k_{11}][R_1] & [R^T_1][k_{12}][R_1] & [R^T_1][k_{13}][R_2] & [R^T_1][k_{14}][R_2] \\ & [R^T_1][k_{22}][R_1] & [R^T_1][k_{23}][R_2] & [R^T_1][k_{24}][R_2] \\ & \text{sym} & [R^T_2][k_{33}][R_2] & [R^T_2][k_{34}][R_2] \\ & & & [R^T_2][k_{44}][R_2]\end{bmatrix}$$

$\cdots\cdots\cdots\cdots\cdots\cdots(2.3)$

ただし、部材座標系の剛性行列は、

$$[k]=\begin{bmatrix}[k_{11}] & [k_{12}] & [k_{13}] & [k_{14}] \\ & [k_{22}] & [k_{23}] & [k_{24}] \\ & \text{sym} & [k_{33}] & [k_{34}] \\ & & & [k_{44}]\end{bmatrix} \qquad \cdots\cdots\cdots\cdots\cdots\cdots(2.4)$$

である。無論、変換前と変換後の剛性行列は共に実対称行列となる。実際に、上記の処理をプログラムコードで見てみよう。まず、線形の剛性行列を作成するサブルーチンと剛性行列を座標変換するサブルーチンの呼び出しコードを示す。

```
c―――――――――――――――――――――――――――――――★部材の線形剛性計算(ok)
      call Cal_stiff_linear(Model_type,Element,Member,Parameter_C,
     *      ak_linear,E_model11,E_model_fiber,M_model11,M_model_fiber,
     *      E_model12,M_model12,E_model13,M_model13,E_model15,M_model15,
     *      E_model21,M_model21,E_model22,M_model22,
     *      E_model31,M_model31,E_model32,M_model32,
     *      E_model33,M_model33,
     *      Bilinear_work,Trilinear_work,Concrete_work,
     *      work1_elememt,work2_elememt,work1_member,work2_member)
c         write(damp_out,*) ' Cal_stiff_linear Ok'
c―――――――――――――――――――――――――――――――★剛性の釣合座標系への変換(ok)
      call Rotate_stiffness(Parameter_C,ak_linear,rot_memb)
```

線形の剛性行列を作成するサブルーチン Cal_stiff_linear()については、後章で詳細に説明することになる。ここでは、剛性行列を座標変

換するサブルーチンについて説明しよう。以下に、この変換サブルーチン Rotate_stiffnes()とそれに付随するサブルーチンを示す。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Rotate_stiffness
C       ―――――――――――――――――――――――――――――――――
C     ●   部材行列の釣合系への座標変換(ok)
C     ―――――――――――――――――――――――――――――――――
      subroutine Rotate_stiffness(Parameter_C,ak_linear,rot_memb)
      implicit real*8(A-H,O-Z)
      include "submain.h"
      record /parameter_s  / Parameter_C
      dimension bk(12,12),rot_memb(3,3,2,*)
      dimension ak_linear(12,12,*)
C     ―――――――――――――――――――――――――――――――――
c     n_member                  : integer   部材数
c     ak_member(12,12,n_member)   : real*8    線形剛性行列
c     rot_memb(12,12,2,*)         : real*8    座標変換行列
C     ―――――――――――――――――――――――――――――――――
      n_member = Parameter_C.N_member
      do i=1,n_member                                                  ! 1
      call Rotate_K(ak_linear(1,1,i),rot_memb(1,1,1,i),                ! 2
     *              rot_memb(1,1,2,i),bK)
      do  j=1,12                                                       ! 3
      do  k=1,12
      ak_linear(j,k,i)=bk(j,k)
      end do
      end do
      end do
      return
      end
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Rotate_K
C       ―――――――――――――――――――――――――――――――――
C     ●   部材行列の釣合系への座標変換(ok)
C     ―――――――――――――――――――――――――――――――――
      subroutine Rotate_K(ak,ri,rj,bK)
      implicit real*8(A-H,O-Z)
      dimension ak(12,12),bk(12,12),ri(3,3),rj(3,3)
      dimension c(3,3)
C     ―――――――――――――――――――――――――――――――――
c          ak(12,12)        :real*8    部材座標系の行列
c          bk(12,12)        :real*8    釣合座標系の行列
c          ri(3,3)          :real*8    i 節点の座標変換行列
c          rj(3,3)          :real*8    j 節点の座標変換行列
C     ―――――――――――――――――――――――――――――――――
      do 10 ii=1,2                                                     ! 4
      i1=(ii-1)*3
      do 10 jj=ii,2
      j1=(jj-1)*3
      do 12 i=1,3                                                      ! 5
      do 12 j=1,3
      sum=0.0
```

```
      do 14 k=1,3
      sum=sum+ak(i+i1,k+j1)*ri(k,j)
   14 continue
      c(i,j)=sum                                                    ! 6
   12 continue
      do 16 i=1,3                                                   ! 7
      do 16 j=1,3
      sum=0.0
      do 18 k=1,3
      sum=sum+ri(k,i)*c(k,j)
   18 continue
      bk(i+i1,j+j1)=sum
   16 continue
   10 continue
c
      do 20 ii=1,2                                                  ! 8
      i1=(ii-1)*3
      do 20 jj=3,4
      j1=(jj-1)*3
      do 22 i=1,3
      do 22 j=1,3
      sum=0.0
      do 24 k=1,3
      sum=sum+ak(i+i1,k+j1)*rj(k,j)
   24 continue
      c(i,j)=sum
   22 continue
      do 26 i=1,3
      do 26 j=1,3
      sum=0.0
      do 28 k=1,3
      sum=sum+ri(k,i)*c(k,j)
   28 continue
      bk(i+i1,j+j1)=sum
   26 continue
   20 continue
c
      do 40 ii=3,4                                                  ! 9
      i1=(ii-1)*3
      do 40 jj=ii,4
      j1=(jj-1)*3
      do 42 i=1,3
      do 42 j=1,3
      sum=0.0
      do 44 k=1,3
      sum=sum+ak(i+i1,k+j1)*rj(k,j)
   44 continue
      c(i,j)=sum
   42 continue
      do 46 i=1,3
      do 46 j=1,3
      sum=0.0
      do 48 k=1,3
      sum=sum+rj(k,i)*c(k,j)
```

```
   48 continue
      bk(i+i1,j+j1)=sum
   46 continue
   40 continue
c
      do i=1,11                                                    ! 10
      do j=i+1,12
      bk(j,i)=bk(i,j)
      enddo
      enddo
      return
      end
C     ――――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Rotate_K6
C       ――――――――――――――――――――――――――――――――――
C     ●   部材行列の釣合系への座標変換(ok)
C     ――――――――――――――――――――――――――――――――――――
      subroutine Rotate_K6(ak,ri,rj,bK)
      implicit real*8(A-H,O-Z)
      dimension ak(12,12),bk(12,12),ri(6,6),rj(6,6)
      dimension c(6,6)
C     ――――――――――――――――――――――――――――――――――――
c         ak(12,12)        :real*8    部材座標系の行列
c         bk(12,12)        :real*8    釣合座標系の行列
c         ri(6,6)          :real*8    i 節点の剛域変換行列
c         rj(6,6)          :real*8    j 節点の剛域変換行列
C     ――――――――――――――――――――――――――――――――――――
      do 12 i=1,6                                                  ! 11
      do 12 j=1,6
      sum=0.0
      do 14 k=1,6
      sum=sum+ak(i,k)*ri(k,j)
   14 continue
      c(i,j)=sum
   12 continue
      do 16 i=1,6
      do 16 j=1,6
      sum=0.0
      do 18 k=1,6
      sum=sum+ri(k,i)*c(k,j)
   18 continue
      bk(i,j)=sum
   16 continue
   10 continue
c
      do 22 i=1,6
      do 22 j=1,6
      sum=0.0
      do 24 k=1,6
      sum=sum+ak(i,k+6)*rj(k,j)
   24 continue
      c(i,j)=sum
   22 continue
      do 26 i=1,6
```

```
      do 26 j=1,6
      sum=0.0
      do 28 k=1,6
      sum=sum+ri(k,i)*c(k,j)
   28 continue
      bk(i,j+6)=sum
      bk(j+6,i)=sum
   26 continue
   20 continue
c
      do 42 i=1,6
      do 42 j=1,6
      sum=0.0
      do 44 k=1,6
      sum=sum+ak(i+6,k+6)*rj(k,j)
   44 continue
      c(i,j)=sum
   42 continue
      do 46 i=1,6
      do 46 j=1,6
      sum=0.0
      do 48 k=1,6
      sum=sum+rj(k,i)*c(k,j)
   48 continue
      bk(i+6,j+6)=sum
   46 continue
   40 continue
      return
      end
```

剛性行列の座標変換は、このサブルーチン Rotate_stiffness()を用いて行う。変換行列は、直交行列であり上に示したように特異な形をしているため、変換後の剛性行列は、式(2.2)で示すような形となる。また、変換後の剛性行列は、対称行列であるため、変換処理も上半分について行い、後の下半分は上半分をコピーすることで得られる。プログラムコードの横に付した番号に従って説明する。

１．全部材について以下の処理を行う。
２．両端の変換行列を用いて、部材座標系で得た剛性行列 ak_linear を釣合座標系に変換する。
３．得られた釣合座標系での剛性 bk を元の剛性行列 ak_linear にコピーする。
４．このサブルーチン Rotate_K()では、３x３の変換行列を用いて、座標変換する。変換方法は、式(2.3)で示すように、10 個の行列三重積で求める。ここで、$[R_1]$は、配列 ri に、$[R_2]$は、配列 rj に相当する。最初に、ri にのみ関連する３つの行列三重積を行う。

５．ここから、部材座標系の剛性に、後から変換行列を掛ける。
６．ワーク領域の配列 c に結果を代入する。
７．行列 c の前から、行列 ri の転置を掛け、その結果を釣合座標系の剛性 bk に代入する。
８．左から ri を、右から rj を部材座標系の剛性に掛ける処理を、指標を換えて 4 回上記と同様に行う。
９．左から rj を、右から rj を部材座標系の剛性に掛ける処理を、指標を換えて 3 回上記と同様に行う。
10. 釣合座標系で得た上半分の剛性行列を、下半分にコピーする。
11. このサブルーチン Rotate_K6()は、上記の Rotate_K()と同様な処理を行うが、ここの変換行列は 6x6 の行列である。プログラムそのものは、ほとんど Rotate_K()と同様であり、理解することは容易である。なお、このサブルーチンは、両端に剛域を持つ部材モデルの変換に使用される。

次に、釣合座標系から全体座標系に変換する例を示そう。ここで示す例は、変位などを図形処理するときに必要となるデータを釣合座標系から全体座標系に変換するサブルーチンである。ここでは、解析で得られた変位を全体座標系に変換する。以下に、そのサブルーチンをコールする部分を取り出す。

```
c―――――――――――――――――――――――――――――――――――――――
c
c    ★          描画用データのセット
c
c―――――――――――――――――――――――――――――――――――――――
c――――――――――――――――――――――――★変位
      if(i_read_disp .ne. 0) then
      call Set_preset_disp(1,n_point,past_disp_point,F_disp,Point,
     *                           rot_local,Parameter_C)
      endif
```

このサブルーチンは、パラメータ i_read_disp が 0 でないとき、コールされる。これは、動的解析システムの方で、ユーザーが図形表示を選択した場合であり、描画用データはここでセットされることになる。以下に、Set_preset_disp()とそれに関連するサブルーチンを示す。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Set_preset_disp
C       ―――――――――――――――――――――――――――――――――
C     ●   節点の変位、速度、加速度を出力(ok)
C     ―――――――――――――――――――――――――――――――――
```

```
      subroutine Set_preset_disp(ihan,n_point,past_disp_point,
     *                           F_disp,Point,rot_local,Parameter_C)
C     ――――――――――――――――――――――――――――――――――――――
      implicit real*8(A-H,O-Z)
      include "submain.h"
      record / point_s     / Point
      record / parameter_s /Parameter_C
      dimension Point(*)
      dimension past_disp_point(*)
      dimension rot_local(3,3,*),v(6),vv(6)
      real*4    F_disp(3,*)
C     ――――――――――――――――――――――――――――――――――――――
      if(ihan.ne.0) goto 900                                          ! 1
      do i=1,n_point
      do j=1,3
      F_disp(j,i)=0.
      enddo
      enddo
      return
  900 continue
c――――――――――――――――――――――――――――――――★局所座標系なし
      if(Parameter_C.n_local_coord.eq.0) then                         ! 2
c――――――――――――――――――――――――――――――――★変位 3
      do i=1,n_point
      do j=1,3
      ires= Point(i).irest(j)
      F_disp(j,i)=0.
      if(ires.ne.0) F_disp(j,i) = past_disp_point(ires)
      end do
      end do
c――――――――――――――――――――――――――――――――★局所座標系あり
      else                                                            ! 3
c――――――――――――――――――――――――――――――――★
      do i=1,n_point
      ij=Point(i).local_coord
      if(ij.eq.0) then                                                ! 4
      do j=1,3
      ires= Point(i).irest(j)
      F_disp(j,i)=0.
      if(ires.ne.0) F_disp(j,i) = past_disp_point(ires)
      end do
c――――――――――――――――――――――――――――――――★
      else
      do j=1,3                                                        ! 5
      ires= Point(i).irest(j)
      v(j)=0.
      if(ires.ne.0) v(j ) = past_disp_point(ires)
      end do
      call trans_VT(v,vv,rot_local(1,1,ij))                           ! 6
      do j=1,3
      F_disp(j,i)=vv(j)                                               ! 7
      enddo
      endif
      end do
```

```
      endif
      return
      end
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /trans_VT
C       ―――――――――――――――――――――――――――――――――
C     ●   荷重ベクトル（釣合系）を全体系に座標変換する
C     ―――――――――――――――――――――――――――――――――
      subroutine trans_VT(p,q,rot)
      implicit real*8(A-H,O-Z)
      real*8 p(3),q(3)
      dimension rot(3,3)
      do i=1,3                                                    ! 8
      sum = 0.
      do j=1,3
      sum=sum + rot(i,j)*p(j)
      end do
      q(i)=sum
      end do
      return
      end
```

1．初期設定であるかどうか判定する。パラメータ ihan が 0 の場合は、初期設定であり、以下のように節点変位 F_disp をゼロクリアする。
2．構造体成分 Parameter_C.n_local_coord で、このモデルが局所座標を使用しているかどうかチェックする。使用していない場合は、以下のように、解析して得た変位ベクトル past_disp_point を節点変位 F_disp にコピーする。
3．局所座標系を使用する場合の処理を行う。全節点について以下の処理を行う。
4．その節点が局所座標を使用しているかどうかチェックする。使用していない場合は、以下のコードで示すようにコピーする。
5．局所座標を使用している場合は、まず、節点変位をワーク領域の変位 v にコピーする。
6．サブルーチン trans_VT()を用いて、この変位を釣合座標系から全体座標系に変換する。変換行列は rot_local である。その結果はワーク領域 v に得られる。
7．ワーク領域 v から節点変位を F_disp にコピーする。
8．サブルーチン trans_VT()では、変換行列と変位ベクトルを掛け算して、全体座標系における節点変位を得る。

以上で、座標変換に関する説明は終了する。理解できただろうか。ここで説明した座標変換は、後章で再度出現する。そのとき、この節をもう一度参照されると良い。

2.6 境界条件と拘束表

解析モデルには、必ず境界が設定される。しかも、その境界は全体座標系で表すことができないこともあり、局所座標系を考慮しなければならない。また、解析によっては、他の節点の変位と同じとなって欲しい場合もある。ここでは、これらの境界条件を与える仕様にしたがって、拘束表を作成する方法を説明する。境界を与える仕様は、リファレンスマニュアルの第 4.1 節における節点の拘束指標を参照されたい。

節点拘束表を作成するサブルーチンは、以下のようにコールされる。

節点の拘束指標
０：自由
－１：固定
－(10×K+L)：
他節点の変位と同一視
K:節点番号
L:自由度番号

節点の自由度番号
1： X 方向変位
2： Y 方向変位
3： Z 方向変位
4： X 軸回りの回転、
5： Y 軸回りの回転、
6： Z 軸回りの回転

```
c―――――――――――――――――――――――――――――――★節点拘束表の作成
      call Set_restraint_point(Parameter_C,Point,Control)
```

このサブルーチン Set_restraint_point()を以下に示す。

```
C     ―――――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Set_restraint_point
C       ―――――――――――――――――――――――――――――――――――
C     ●   節点拘束表の作成（未知数等をセット)(ok)
C     ―――――――――――――――――――――――――――――――――――――
      subroutine Set_restraint_point(Parameter_C,Point,Control)
      implicit real*8(A-H,O-Z)
      include "submain.h"
      record /control_s     / Control
      record / parameter_s  / Parameter_C
      record / point_s      / Point
      dimension Point(*)
C     ―――――――――――――――――――――――――――――――――――――
c
c     ★        節点拘束表の作成（未知数等をセット)(ok)
c         -1:拘束 0:自由
c         負:他の自由度と変位を一緒にする
c            節点*10 + 自由度番号
c         注：このオプションを利用する場合は、必ずその節点における自由度番号は
c             設定されているものとする。そうでない場合は、その自由度は拘束され、
c             ゼロが入る。
c     ★       解析モデルが平面応力の場合、節点拘束表を変更する。
c       Control.analysis_3D          !  解析型 0:3D 1:2D(x-z) 2:2D(y-z)
c
c     Parameter_C    :structure
c     Point          :structure
C     ―――――――――――――――――――――――――――――――――――――
c
c―――――――――――――――――――――――――――――――★平面応力の場合のセット
      if(Control.analysis_3D.eq.1) then                                  ! 1
      do i=1,Parameter_C.n_point
      Point(i).irest(2)=-1
      Point(i).irest(4)=-1
      Point(i).irest(6)=-1
      enddo
```

```
      elseif(Control.analysis_3D.eq.2) then                          ! 2
      do i=1,Parameter_C.n_point
      Point(i).irest(1)=-1
      Point(i).irest(5)=-1
      Point(i).irest(6)=-1
      enddo
      endif
c―――――――――――――――――――――――――――――――――――――――――――――★拘束表の作成
      ire = 0
      do i=1,Parameter_C.n_point                                     ! 3
      do j=1,6
      if(Point(i).irest(j).ge.0) then                                ! 4
      ire = ire + 1
      Point(i).irest(j) = ire
      elseif(Point(i).irest(j).eq.-1) then                           ! 5
      Point(i).irest(j) = 0
      endif
      end do
      end do
      Parameter_C.n_unknown = ire     !未知数のセット
      do i=1,Parameter_C.n_point                                     ! 6
      do j=1,6
      if(Point(i).irest(j).lt.0) then                                ! 7
      ip = iabs(Point(i).irest(j))
      Point(i).irest(j)=0
      ipp = ip/10
      ifre = ip - ipp*10                                             ! 8
      if(ipp.le.Parameter_C.n_point.and.                             ! 9
     *   ipp.ge.0) then
      jpp = Point(ipp).irest(ifre)                                   ! 10
      if(jpp .gt. 0 ) Point(i).irest(j) = jpp                        ! 11
      endif
      end if
      end do
      end do
c
      write(76,*) ' n_unknown:',Parameter_C.n_unknown
      do i=1,Parameter_C.n_point
      write(76,'(i8,6i8)') i,(Point(i).irest(j),j=1,6)
      enddo
      return
      end
```

１．ユーザーの指定により、平面応力解析であるかどうかチェックする。構造体 Control.analysis_3D が 1 の場合、平面(x,z)の解析となり、全節点について境界条件を付け加える。

２．構造体成分 Control.analysis_3D が 2 の場合、平面(y,z)の解析となり、全節点について境界条件を付け加える。

３．全節点について拘束条件をチェックするため、以下の処理を行う。その前に、未知番号 ire をゼロクリアする。

４．その節点の自由度 point(i).irest(j)が 0 以上の場合は、拘束されていないので、未知番号 ire に１を足し、point(i).irest(j)にセットする。
５．節点の自由度 point(i).irest(j)が-1 に等しいとき、拘束を表すので、point(i).irest(j)に 0 をセットする。これで、一回目の処理を終わり、節点自由度が自由の場合と、拘束の場合について処理が完了した。
６.全未知数 ire を構造体成分 Parameter.n_unknown に設定する。次に、２回目のチェックを行う。この調査は未知自由度の同一視であり、全節点について行う。
７．節点の自由度 point(i).irest(j)が負の場合、以下の処理を行う。まず、point(i).irest(j)に 0 をセットして拘束状態とする。
８．次に、同一視する節点番号 ipp と自由度番号 ifre を計算する。
９．この節点番号が適切かどうかチェックする。適切である場合は、以下の処理を行う。適切でない場合は、先の拘束状態が生きることとなる。
10. 指定した節点と自由度から未知数番号を取得する。
11. この未知数番号が正である場合は、point(i).irest(j)にその未知番号をセットする。

2.7 スカイライン行列とその操作

第３章で示すように、動的解析は、ベクトルや行列で記述されている。当然プログラムコードも、行列やベクトルをそのまま使用して記述されている。特に、振動方程式では、剛性などが行列で記述されているため、この処理を誤ると極端に処理時間がかかる。変位法系の剛性行列は、実対称で、しかも、バンド性を有していることが知られており、方程式の解法として、直接法のひとつであるバンドコレスキー法が使用されてきた。しかし、現在では、より効率的で解析時間の短いスカイライン法が多用されており、SPACE でもこの手法が用いられている。ここでは、このスカイライン法の考え方とその手法を使ったサブルーチンを具体的に解説しよう。

実対称行列をスカイライン行列として表現する方法は、下三角を対象とすると４通りある。

１．下三角を対象として、値の入っている最左端より対角項まで

２．下三角を対象として、対角項から値の入っている最左端まで
３．下三角を対象として、値の入っている最下端より対角項まで
４．下三角を対象として、対角項から値の入っている最下端まで

もちろん、スカイライン行列の上三角を対象にすれば、対称行列であるため、上記の４つの範疇に入る。

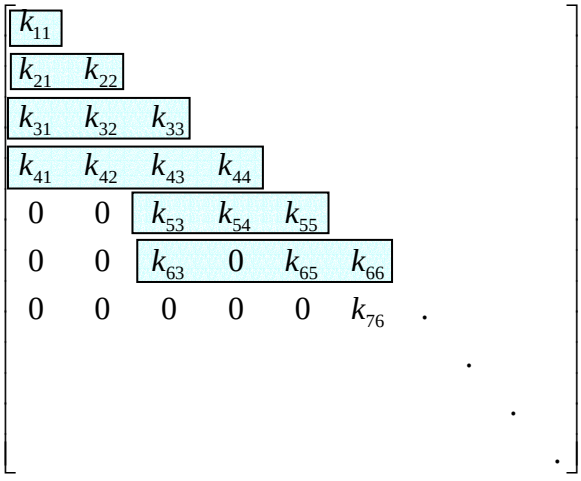


図 2-2 対称行列と水平型スカイライン行列

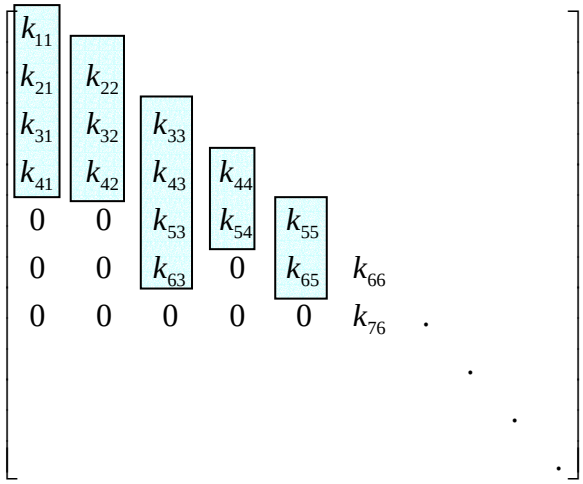


図 2-3 対称行列と垂直型スカイライン行列

どのタイプのスカイライン行列を使用するかを良く頭に入れておく必要がある。これによって、行列の分解や行列とベクトルの掛け算などの方法が異なるからである。SPACE では、数値計算のロジックなどを考慮して、１番目のデータ保存方法を用いている。

それでは、具体的にどのようにこのスカイライン行列を蓄え、操作するかについて説明しよう。ここでは、データの保存法とアクセス法、行列の作成法について説明する。ただし、このスカイライン行列の LDU 分解など、解析手法のロジックについては参考文献を参照されたい。ここでは、SPACE で使用されているプログラムを記載するに留める。

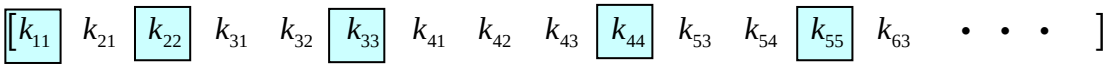
スカイライン行列は、１次元配列でデータを保存する。そのため、剛性などの行列をこのスカイライン行列に変換するテーブルが必要となる。このテーブルは、行列の各行の左より最初のゼロでない値を持つ要素から、対角要素までの数を並べたものである。あるいは、行列の各列の上から、最初のゼロでない値を持つ要素から、対角要素までの数を並べたものである。具体的に、図 2-2 の水平型の行列について、テーブルを作成してみよう。まず、上で示したテーブルを下のように作成する。ただし、最初の 0 はダミーであり、計算の都合上入れる。

[0　1　2　3　4　3　4　・　・　・　　]

次に、このテーブルを下から和を取っていき、次のように変更する。

[0　1　3　6　10　13　17　・　・　・　　]

上記がスカイライン行列に変換するためのテーブルであり、このテーブルを用いて、LDU 分解などの操作を行う。スカイライン変換表の各値は、一つ目のダミーを除いて、下に示すスカイライン行列の対角項を示す。従って、テーブルの最初の要素番号は、0 から始まるものとする。



このスカイライン変換表を作成するためのサブルーチンコールを以下に示す。

```
c―――――――――――――――――――――――――――――――――★部材両端の拘束表作成(ok)
      call Set_restraint_member(Parameter_C,Member,Point)
c―――――――――――――――――――――――――――――――――★スカイライン変換表作成(ok)
c―――――――――――――――――――――――――――――――――★スカイライン行列の領域数等をセット
      call Cal_table_sky(max_h_sky,Parameter_C,Member)
```

スカイライン変換表を作成するサブルーチンを以下に示す。

```
C     ―――――――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Set_restraint_member
C       ―――――――――――――――――――――――――――――――――――――
C     ●   部材両端の拘束表作成(ok)
C     ―――――――――――――――――――――――――――――――――――――――
      subroutine Set_restraint_member(Parameter_C,Member,Point)
      implicit real*8(A-H,O-Z)
      include "submain.h"
      record / parameter_s  / Parameter_C
      record / point_s      / Point
      record / member_s     / Member
      dimension Point(*),Member(*)
C     ―――――――――――――――――――――――――――――――――――――――
c     Parameter_C    :structure
c     Member         :structure
c     Point          :structure
C     ―――――――――――――――――――――――――――――――――――――――
      write(76,*) ' 部材拘束表',Parameter_C.n_member
      do i=1,Parameter_C.n_member                                    ! 1
      i1=Member(i).nm_point(1)                                       ! 2
      do j=1,6
      Member(i).irest(j) = Point(i1).irest(j)
```

```
      end do
      i1=Member(i).nm_point(2)                                          ! 3
      do j=7,12
      Member(i).irest(j) = Point(i1).irest(j-6)
      end do
      write(76,'(i4,6i8,4x,6i8)') i,(Member(i).irest(j),j=1,12)
      end do
      return
      end
C     ――――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Cal_table_sky
C       ――――――――――――――――――――――――――――――――――
C     ●   スカイライン作成用の表を作る(ok)
C     ――――――――――――――――――――――――――――――――――――
      subroutine Cal_table_sky(max_h_sky,Parameter_C,Member)
      implicit real*8(A-H,O-Z)
      include "submain.h"
      record / parameter_s  / Parameter_C
      record / member_s     / Member
      dimension Member(*),max_h_sky(0:*)                                ! 4
C     ――――――――――――――――――――――――――――――――――――
c     max_h_sky      :integer  スカイライン行列の各列の高さを表す表
c     Parameter_C    :structure
c     Member         :structure
c     実対称行列をスカイライン行列として表現する方法は、４通りある。
c        １．下三角行列で、値の入っている最左端より、対角項まで
c        ２．下三角行列で、対角項から値の入っている最左端まで
c        ３．下三角行列で、値の入っている最下端より、対角項まで
c        ４．下三角行列で、対角項から値の入っている最下端まで
c     もちろん、スカイライン行列を上三角行列と思えば、対称行列であるため、
c     上記の４つの範疇に入る。
c     ここでは、１のスカイライン行列を作成するための表を作る
C     ――――――――――――――――――――――――――――――――――――
      n_unknown=Parameter_C.n_unknown                                   ! 5
      n_member=Parameter_C.n_member
      max_h_sky(0)=0                                                    ! 6
      do i=1,n_unknown                                                  ! 7
      max_h_sky(i)=1
      end do
      do  i=1,n_member                                                  ! 8
      do  j=1,12
      if(Member(i).irest(j).ne.0) then                                  ! 9
      ISSI=iabs(Member(i).irest(j))
      do k=1,12
      if(Member(i).irest(k).ne.0) then                                  ! 10
      ISSS=iabs(Member(i).irest(k))
      if(ISSI.gt.ISSS)  then                                            ! 11
      IIBW=iabs(ISSS-ISSI)+1                                            ! 12
      max_h_sky(ISSI)=max0(max_h_sky(ISSI),IIBW)                        ! 13
      end if
      end if
      end do
      end if
      end do
```

```
end do

do i=1,n_unknown                                                    ! 14
max_h_sky(i)=max_h_sky(i-1)+max_h_sky(i)
end do
Parameter_C.n_skyline=max_h_sky(n_unknown)                          ! 15
Parameter_C.n_sky_ave=max_h_sky(n_unknown)/n_unknown
write(76,*) 'n_skyline',Parameter_C.n_skyline
write(76,*) 'n_sky_ave',Parameter_C.n_sky_ave
return
end
```

１．このサブルーチンでは、節点拘束表から部材拘束表を作成する。全部材について次の処理を行う。
２．部材端ｉの節点番号を取得し、節点の拘束表（既に、未知番号表となっている）をコピーする。
３．部材端ｊの節点番号を取得し、節点の拘束表をコピーする。
４．このサブルーチンでスカイライン行列の変換表を作成するが、変換表 max_h_sky は、0 から始まるよう設定する。
５．モデルの全未知数を構造体から取得する。
６．変換表 max_h_sky(0)に 0 を設定する。
７．全未知数に対し、変換表 max_h_sky を 1 に設定する。
８．全部材について次の処理を行い、変換表を作成する。
９．部材両端の 12 自由度に対し、拘束のない場合、次の処理を行う。まず、j 番目の未知数番号 ISSI を取得し、ゼロ以外であることをチェックする。
10．ｋ番目の未知数番号 ISSS を取得し、ゼロ以外であることをチェックする。
11．未知番号 ISSI が ISSS より大きい値の時、次の処理を行う。
12．j 番目とｋ番目の間の未知番号 IIBW を計算する。これがこの行の最大値を計算するための資料となる。
13．現在までのこの行の最大値と IIBW を比較し、大きい方を変換表 max_h_sky にセットする。これで、先に説明した変換表の第一段階の処理が終了する。
14．次に、第二段階の処理を行う。全未知番号数分、若い要素番号から２つの要素の和を取っていく。
15．変換表 max_h_sky(n_unknown)には、スカイライン行列の全要素数が計算されており、これを構造体にセットする。

このスカイライン変換表に関連する例を示そう。ここでは、スカイライン行列のゼロクリアと線形剛性のスカイライン行列への足しこみ、及び、このスカイライン行列の LDU 分解を行うサブルーチンをコールするコードを示す。

```
c――――――――――――――――――――――――――――★スカイライン行列のゼロセット(ok)
      n_skyline=Parameter_C.n_skyline
      call Set_sky_zero(gskym,n_skyline)
c――――――――――――――――――――――――――――★線形剛性の足し込み(ok)
c――――――――――――――――――――――――――――★レーリー減衰を含む
      call Build_sky_k(n_istep,gskym,n_skyline, Member,n_member,
     *                       Ak_linear, Newmark_P, max_h_sky)
c       write(damp_out,*) ' Build_sky_k ok'
c――――――――――――――――――――――――――――――――――★行列の LDU 分解(ok)
      n_skyline=Parameter_C.n_skyline
      call decomp_sky(n_skyline,n_unknown,n_unknown,max_h_sky,
     *                gskym,gskym_d,nwork,twork,iexit)
```

以上の 3 つのサブルーチンと関連するサブルーチンを具体的に示す。ただし、LDU 分解を行うサブルーチン decomp_sky()の説明は、参考文献を参照されたい[3)]。

```
C     ――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Set_sky_zero
C       ――――――――――――――――――――――――――――――――――
C     ●   スカイライン行列のゼロセット(ok)
C     ――――――――――――――――――――――――――――――――――
      subroutine Set_sky_zero(gskym,n_skyline)
      implicit real*8(A-H,O-Z)
      dimension gskym(*)
C     ――――――――――――――――――――――――――――――――――
c      gskym(n_skyline)          :real*8   スカイライン行列
c      n_skyline                 :integer  スカイライン行列の大きさ
C     ――――――――――――――――――――――――――――――――――
      do i=1,n_skyline                                              ! 1
      gskym(i)=0.0
      end do
      return
      end
C     ――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Build_sky_kk
C       ――――――――――――――――――――――――――――――――――
C     ●    部材剛性行列のスカイライン行列への組み込み(ok)
C     ――――――――――――――――――――――――――――――――――
      subroutine Build_sky_kk(n_istep,gskym,n_skyline, Member,n_member,
     *            ak_linear ,ak_nonlinear,Newmark_P, max_h_sky)
      implicit real*8(A-H,O-Z)
      include "submain.h"
      dimension ak_linear(12,12,*),gskym(*),ak_nonlinear(12,12,*)
      dimension max_h_sky(0:*)
      record / newmark_s  / Newmark_P
```

```
      record / member_s   / Member
      dimension Member(*)
c-------------------------------------------------------------
c     n_istep                   :integer   第一ステップか第二ステップか
c     n_skyline                 :integer   スカイライン行列の大きさ
c     Member                    :structure
c     n_member                  :integer   部材数
c     Ak_linear(12,12,n_member)  :real*8    線形剛性行列（釣合系）
c     Ak_nonlinear(12,12,n_member)  :real*8 非線形剛性行列（釣合系）
c     Newmark_P     : structure
c     max_h_sky(n_unknown+1)    : integer   スカイライン行列の各列の高さ
c-------------------------------------------------------------
c-------------------------------★線形減衰
      if(n_istep.eq.1) then                                          ! 2
      par=Newmark_P.ddt*Newmark_P.alf1_2
      else
      par=Newmark_P.ddt*Newmark_P.alf2_2
      endif
      do i=1,n_member                                                ! 3
      call Build_ak(gskym,
     *     Member(i).irest(1), max_h_sky,
     *     par,ak_linear(1,1,i) )
      end do
c-------------------------------★接線剛性
      if(n_istep.eq.1) then                                          ! 4
      par=Newmark_P.bdt
      else
      par=Newmark_P.bdt
      endif
      do i=1,n_member
      call Build_ak(gskym,                                           ! 5
     *     Member(i).irest(1), max_h_sky,
     *     par,ak_nonlinear(1,1,i) )
      end do
      return
      end
C     ---------------------------------------------
C     ●   SUBROUTINE /Build_ak
C       -------------------------------------------
C     ●   部材剛性行列のスカイライン行列への組み込み(ok)
C     ---------------------------------------------
      subroutine Build_ak(gskym,irest,max_h_sky,par,ak)
      implicit real*8(A-H,O-Z)
      dimension gskym(*)
      dimension max_h_sky(0:*),ak(12,12),irest(12)
c-------------------------------------------------------------
c     gskym(n_skyline)          :real*8   スカイライン行列
c     irest(12)                 :integer   部材拘束表
c     max_h_sky(n_unknown+1)    :integer   スカイライン行列の各列の高さ
c     AK(12,12)                 :real*8    部材行列
c-------------------------------------------------------------
      do  j=1, 12                                                    ! 6
      i1=irest(j )
      IF(i1.gt.0) then
```

```
      do  k=1, 12
      i2=irest(k )
      if(i2.gt.0.and.i2.le.i1) then                                    ! 7
      i3=max_h_sky(i1)-(i1-i2)                                         ! 8
      gskym(i3)=gskym(i3)+ak(j,k)*par                                  ! 9
      end if
      end do
      end if
      end do
      return
      end
c-----------------------------------------------------------------------
c     ●   SUBROUTINE /decomp_sky
c       ―――――――――――――――
c     ●   スカイライン法（ＬＤＵ分解）(ok)
c-----------------------------------------------------------------------
      subroutine decomp_sky(ntdil,ntdis,n_unknown,nsum_d,gskym,
     1                 gskym_d,nwork,twork,iexit)
      implicit real*8(A-H,O-Z)
      integer*4 fst,fsti,nsum_d(0:ntdis)
      dimension gskym(ntdil),gskym_d(ntdis)
      dimension nwork(ntdis),twork(ntdis)
c-----------------------------------------------------------------------
C     -   スカイライン法によるＬＤＵ分解。
C     -
C     -   n_unknown    : 構造体に使われている未知変位の総数。
C     -   nsum_d       : スカイライン法による未知数格納用指標。
C     -   gskym        : 全体剛性マトリックス。
C     -   gskym_d      : 全体剛性マトリックスの対角項
C     -   nwork        : ワークエリア
C     -   twork        : ワークエリア
C     -   iexit        : 計算コード
c-----------------------------------------------------------------------
      iexit=0
      i=1
      if(gskym(1).eq.0.0) goto 99
      gskym_d(1)=1.0D0/gskym(1)
      do i=1,n_unknown
      nwork(i)=i-(nsum_d(i)-nsum_d(i-1))+1
      enddo
      do 20 i=2,n_unknown
      fst=nwork(i)
      if(i.eq.2) goto 23
      ii=2
      if(fst.gt.2) ii=fst
      do j=ii,i-1
      fsti=nwork(j)
      lgth=MAX(fst,fsti)
      if(lgth.gt.j) goto 21
      ii=nsum_d(j)-j
      jj=nsum_d(i)-i
      if(fsti.eq.j) goto 21
      if(lgth.ge.j) goto 21
      s=0.0D0
```

```
      do  k=lgth,j-1
      s=s+gskym(ii+k)*gskym(jj+k)
      enddo
      gskym(jj+j)=gskym(jj+j)-s
   21 continue
      enddo
   23 continue
      s=0.0D0
      ii=nsum_d(i-1)-fst+1
      if(fst.eq.i) goto 26
      do j=fst,i-1
      twork(j)=gskym(ii+j)
      gskym(ii+j)=gskym_d(j)*twork(j)
      enddo
      do j=fst,i-1
      s=s+gskym(ii+j)*twork(j)
      enddo
      ss=gskym(nsum_d(i))-s
      if(ss.eq.0) goto 99
      gskym_d(i)=1.0D0/ss
      gskym(nsum_d(i))=1.0D0/gskym_d(i)
      goto 20
   26 continue
      if(gskym(nsum_d(i)).eq.0.0) goto 99
      gskym_d(i)=1.0D0/gskym(nsum_d(i))
   20 continue
      goto 999
   99 continue
      iexit=1
      write(76,'(a,i4,a,e16.5)') ' Near singular:',i
      return
  999 continue
      end
```

１．サブルーチン Set_sky_zero()は、スカイライン行列のゼロクリアを行う。

２．サブルーチン Build_sky_kk()は、線形剛性が関連する減衰項と剛性項をスカイライン行列に組み込む。SPACE では、第一段階と第二段階の動的解析の始めに、このサブルーチンを使用する。パラメータ n_istep=1 は第一段階であり、減衰項に関連するパラメータを計算する。

３．全部材について、サブルーチン Build_ak()を用いて、減衰項をスカイライン行列に組み込む。

４．剛性項に関連するパラメータを計算する。

５．全部材について、サブルーチン Build_ak()を用いて、剛性項をスカイライン行列に組み込む。

６．列ｊについて拘束されていない自由度について、次の処理を行う。

７．また、行ｋについて拘束されていない自由度について、及び下三角に値が組み込まれるのかをチェックし、条件を満たすと次の処理を行う。
８．スカイライン変換表を用いて、スカイライン位置i3を取得する。
　i3　＝　(行 i1 のスカイライン行列対角項の位置)　－
　　(行 i1 の未知番号数-列 i2 の未知番号数)
９．スカイライン行列に部材の係数行列を足し込む。

次に、線形の方程式を解くサブルーチン solve_sky()について示す。

```
c――――――――――――――――――――――――――――――――★線形方程式を解く(ok)
      n_skyline=Parameter_C.n_skyline
      call solve_sky(n_skyline,n_unknown,n_unknown,
     *               max_h_sky,gskym,gskym_d,
     *               nwork,twork,ld_point_repeat,result_acc_point)
```

このサブルーチンを具体的に以下に示す。ただし、このサブルーチンの説明は、参考書を参照されたい。

```
c――――――――――――――――――――――――――――――――――――――――――――
c     ●   SUBROUTINE /solve_sky(ok)
c      ――――――――――――――――
c     ●   スカイライン法代入計算
c――――――――――――――――――――――――――――――――――――――――――――
      subroutine solve_sky(ntdil,ntdis,n_unknown,nsum_d,gskym,
     *                     gskym_d,nwork,twork,pload,disp)
C     ――――――――――――――――――――――――――――――――――
      implicit real*8(A-H,O-Z)
      integer*4 fst,nsum_d(0:ntdis)
      dimension gskym(ntdil),gskym_d(ntdis),
     1          pload(ntdis),disp(ntdis)
      dimension nwork(ntdis),twork(ntdis)
c――――――――――――――――――――――――――――――――――――――――――――
C     -   スカイライン法による代入計算。
C     -
C     -   n_unknown : 構造物に使われている未知変位の総数
C     -   nsum_d    : スカイライン法による未知数格納用指標
C     -   gskym     : 全体剛性マトリックス
C     -   gskym_d   : 全体剛性マトリックスの対角項
C     -   nwork     : ワークエリア
C     -   twork     : ワークエリア
C     -   pload     : 全体荷重マトリックス
C     -   disp      : 節点変位の値
c――――――――――――――――――――――――――――――――――――――――――――
c     ● 前進代入法
c――――――――――――――――――――――――――――――――――――――――――――
      TWORK(1)=pload(1)
      do i=2,n_unknown
      fst=nwork(i)
```

```
      ii=nsum_d(i-1)-fst+1
      s=pload(i)
      do j=fst,i-1
      s=s-gskym(ii+j)*twork(j)
      enddo
      pload(i)=S
      twork(i)=S
      enddo
c-----------------------------------------------------------
c     ● 後退代入法
c-----------------------------------------------------------
      do i=1,n_unknown
      disp(i)=twork(i)*gskym_d(i)
      enddo
      do i=n_unknown,2,-1
      fst=nwork(i)
      ii=nsum_d(i-1)-fst+1
      S=disp(i)
      if(fst.eq.i) goto 21
      do j=fst,i-1
      disp(j)=disp(j)-gskym(ii+j)*S
      enddo
   21 continue
      enddo
      return
      end
```

2.8 非線形数値解析の難しいところ

非線形解析では、同じ解析モデルを用いても、解析プログラムが異なると必ずしも同じ結果が得られるとは限らない。むしろ、程度の差こそあれ、差異が生じるのが普通であるといえる。特に、解析モデルの中に、座屈や塑性崩壊などの不安定性が存在する場合、この差異によって、大きく最終結果が異なる場合もある。この差異は、一体何が原因になって生じるのか。特に、他の解析ソフトと比較したり、結果を評価したりする場合は、これらを十分に理解したうえでなければならない。

ここでは、その主たる原因として、非線形性のモデル化と解析手法の選択に着目しよう。非線形性を考慮する場合、ある仮定に基づいて行われる。この仮定は妥当であるのか、また、その仮定に基づくモデル化の精度はどの程度なのか。これらを的確に知っておく必要がある。SPACEでは、幾何学的非線形性と材料非線形性を同時に考慮して解析を行っている。しかし、両非線形性とも、計算機の処理能力、特に計算コストを考慮して、各種のモデル化を行っている。そこには簡素化が含まれており、解析モデルによっては、それから影響を受ける場合もある。これらの仮定やモデル化については、マニュアル理論編を参照し、良く理解し

ておいて頂きたい。

結果に誤差が生じる原因のひとつは解析手法の選択にある。非線形性には、幾何学的非線形と材料非線形とがある。幾何学的非線形解析は、基本的には反復解法を必要とし、これを如何に実現しているかを理解し、これをどのように数値解析しているのかを知っておく必要がある。仮定と解析手法の選択を良く理解し、誤差の生じる可能性を学んでおくことが大切である。