第２章　動的解析システムを理解するための基礎

2.1 はじめに

SPACE システムにおける動的解析では、数値計算は FORTRAN を、ユーザーとのインターフェイスは C++を用いて書かれており、この 2 つの言語を併用している。本章と次章で、SPACE の動的解析システムを理解するうえで、必要となる数値解析理論の基礎、並びにプログラムコードを理解するためのプログラミングに関する基礎的な知識について述べる。

最初に、プログラムをコーディングする際、注意しなければならない点に触れよう。**「プログラムは記号の集まりではなく、本のように読むことができなければならない」**、という基本的な考えを持つことが大切である。時々、数学の証明問題のような記号の塊で作られたプログラムを見かけることがある。開発当初は理解できていても、少し時間を置くとほとんど理解できなくなってしまう。ましてや、他人のプログラムはちんぷんかんぷんとなる。各自がちょっとした工夫をすることで読み易く、理解しやすいプログラムを作ることができる。例えば、変数や配列は、小文字も含めた長くて意味のある名前にすると良い。無論、FORTRAN では、大文字と小文字の区別はなく、全て大文字としてコンパイラーは理解するが、プログラムコードは小文字を利用しても良いことになっている。ただし、バイリンガルで C や C++とインターフェイスをとるとき、これらの言語は大文字と小文字を区別するため、気を付ける必要がある。C や C++の中で、FORTRAN のサブルーチンをコールするとき、その関数名は全て大文字を用いないといけない。同じ名前でも、小文字を含むと他の関数として見なされ、リンクできないことになる。また、小文字の使用と共に、多くの文字で意味のある変数名や配列名を設定することは、読みやすさの点で大いに効果がある。なお、変数等の名前の文字数は、FORTRAN77 では 6 文字に制限されているが、ここでは、FORTRAN90 以上を用いることにしているためこの制限はない。

次の注意点は、**「プログラムはシンプルでなければならない」**ことである。この動的解析システムでも十数万ステップを必要とし、システム全体が複雑となっている。そのため、個々のサブルーチンや関数は単純でないとシステムはより複雑となり、全体を理解することができなくなる。プログラムの構造がシンプルであれば、より広い範囲を一度に理解できることになる。システムは一旦作成すれば、それで終了と云うようなことはない。仕様の変更や拡張が必ず生じ、メンテナンスを行うことになる。個々のルーチンが複雑であったり、他のルーチンと複雑に絡み

合ったりすると、そこを理解するだけで手間が掛かってしまうだけでなく、バグが発生する確率が一段とたかくなる。

「読みやすいプログラムはコメントを多く含む」。特に、各サブルーチンの仕様をそのコード内に書いておくと良い。後から読むとき理解し易く、また変更も容易となる。

「数値計算用プログラムは安定性を第一義」とすべきである。安定性が良いとは、桁落ち、オーバーフロー、アンダーフローなどの誤差が極力発生しない、また、異常終了しないことを指す。特にゼロ割り算は、システムエラーを生じさせるため、ゼロ割り算の可能性がある場合は、プログラムでチェックしておかなければならない。これらを常に考慮してコーディングすることを心がけるべきである。

次節以降では、最初に FORTRAN77 では許されていない動的領域の確保と構造体について紹介する。これらは、FORTRAN90 以上で使用することができ、本章以降でも頻繁に用いる。この２つの新しい技術は FORTRAN を使用する数値解析では非常に有効である。動的解析システム SPACE Ver.3.00 では、開発言語として FORTRAN と C++を併用しており、ここでは、両言語間のインターフェイスやデータの交換方法について述べる。次に、数値解析で必須の基礎知識である座標系とスカイライン法について述べる。まず、部材座標系、全体座標系、釣合座標系について説明し、各座標間の変換について解説する。さらに、線形方程式を解法するための手法のひとつで、現在、最も多用されているスカイライン法について解説する。

説明に使用するソースコードは、動的解析システム SPACE のコードをそのまま用いており、その内容についても記憶に留めておいてもらいたい。また、以降の FORTRAN ソースコードは全て固定フォーマットを用いている。ソースコードの骨組みや概略を説明する際、ソースコードを省いて記述する。その際、記号・・は、ソースコードの省略を意味し、サブルーチンに引数がない場合は、引数が省略されている。

2.2 動的領域について

数値解析プログラムを作成するとき最も悩むのは、配列の大きさである。配列の大きさは、構造物の大きさ、つまり自由度の数や部材数、節点数に依存する。そのため、大きく設定すれば、内部メモリの小さいコンピュータでは、仮想メモリを使用することになり、極端に処理能力が低下することになる。逆に、小さく設定すると大きな構造物が解析できないことになる。

このような不合理を解消するために、FORTRAN90 以降では、実行時に領域を確保することが可能となった。これを動的領域確保（dynamic memory allocation）と呼ぶ。逆にコンパイル時にメモリ領域を確保する方法を静的領域確保（static memory allocation）と称し、FORTRAN77 での配列設定はこれに相当する。無論、構造物の規模によって、配列の大きさが変化しない場合は、この静的領域確保を用いることになる。

動的領域確保は、どのようにして行うのか。一般によく使用される方法を用いて説明しよう。動的領域を確保するためには、少し面倒な手続きが必要となる。以下にその手続きを、動的解析システム SPACE の数値解析部分の submain.f ファイル中の submain_dynanic_a()サブルーチンを用いて説明する。このサブルーチンは、動的ソルバーの主関数であり、今後頻繁に引用される。次節以降では、必要となる部分の一部をこのサブルーチンから抜き出して説明する。

2.2.1 動的領域の宣言

最初に、実行時に変数領域を動的確保するために、その変数あるいは配列を動的領域とするという宣言を行わなければならない。その方法が以下に示されている。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /submain_dynamic_a
C       ―――――――――――――――――――――――――――――――
C     ●   動的解析主プログラム（反復解法＋陰解法）Ver.3.00
C     ―――――――――――――――――――――――――――――――――
  c―――――――――――――――――――――――――――――――――――――――――
  c    ★         動的配列定義
  c―――――――――――――――――――――――――――――――――――――――――
  c     １） スカイライン行列
        real*8, ALLOCATABLE :: gskym(:),gskym_d(:),twork(:)
        integer,ALLOCATABLE :: nwork(:)
  c―――――――――――――――――――――――――――――――――――――――――
  c      real*8  gskym(n_skyline)       ! スカイライン行列
  c―――――――――――――――――――――――――――――――――――――――――
  c     ２）自由度関連
        real*8, ALLOCATABLE ::
       *   disp_point(:),vel_point(:),acc_point(:),
       *   est_disp_point(:),est_vel_point(:),est_ddisp_point(:)
        real*8, ALLOCATABLE ::
       *   ld_point(:),ld_point_repeat(:),fld_static(:,:),
       *   a_vector(:),b_vector(:)
```

宣言文は、ALLOCATABLE :: から始まり、その後に配列名などを記述する。ALLOCATABLE 文の前の real*8 などは、後に続く変数の型を表す

宣言文である。この変数が１次元配列の場合は gskym(:)で、二次元配列は fld_static(:,:)などと記述する。

次に、構造体の動的領域確保の方法について説明する。特に、動的領域として宣言したい変数が構造体でしかも配列の場合について解説する。構造体の中身とその宣言の仕方は後節で説明するが、SPACE で使用する全ての構造体は、ヘッダーファイル"submain.h"と"submainx.h"で定義されており、それらの構造体を引用したい場合は、次に示す方法でこれら２つのファイルをインクルードする。

```
c――――――――――――――――――――――――――――――――★構造体の定義ヘッダーファイル
      include "submain.h"
      include "submainx.h"
```

submain.h ヘッダーファイル内には point_s、element_s、member_s などの構造体に関する成分が定義されている。

話題を構造体の動的確保に戻すことにしよう。まず、変数が構造体であることを以下のように record 文で宣言する。ここでは、節点に関する構造体として Point が point_s に、要素に関する構造体として Element が elemen_s に、部材に関する構造体として Member が member_s に割り付けられている。実際のコードを見てみよう。

```
c―――――――――――――――――――――――――――――――――――――――――――――
c
c    ★          構造体定義
c
c―――――――――――――――――――――――――――――――――――――――――――――
      record /point_s        / Point
      record /element_s      / Element
      record /member_s       / Member
```

次に、この構造体を配列の動的領域として、ALLOCATBLE::文を使用して、以下のように宣言する。これで、構造体配列の動的領域宣言が行われたことになる。

```
c―――――――――――――――――――――――――――――――――――――――――――――
c
c    ★          構造体の動的領域宣言
c
c―――――――――――――――――――――――――――――――――――――――――――――
      ALLOCATABLE :: Point(:)
      ALLOCATABLE :: Element(:)
      ALLOCATABLE :: Member(:)
```

2.2.2 動的領域の確保

次に、動的領域変数として定義した変数の領域を、実際に確保する方法について説明する。動的領域は、実行時に適切な大きさの領域が確保

され、また、不必要になるとその領域を切り離して解放することもできる。動的領域を確保するためには、その配列がどの程度の大きさなのかを知る必要がある。例えば、前節で定義した Point、Element、Member は、プログラムが実際に実行され、解析モデルの大きさが分かってから、その配列の大きさが決定される。ここでは、その値として節点数、要素数、部材数が用いられ、それらは構造体 Parameter_C に収められている。無論、動的確保が実行される前に、各値は、構造体 Parameter_C の成分に、例えば節点数は n_member にセットされていなければならない。

動的領域の確保は、以下のように ALLOCATE 文を用いて行われる。このコードが実行されるとき、具体的に領域が確保されることになる。

```
c―――――――――――――――――――――――――――――――――――――――
c
c    ★          構造体の大きさを動的確保する（その１）
c
c―――――――――――――――――――――――――――――――――――――――
      ALLOCATE (Member(Parameter_C.n_member))
      ALLOCATE (Element(Parameter_C.n_element))
      ALLOCATE (Point(Parameter_C.n_point))
```

2.2.3 動的領域の解放

一旦、動的領域を確保すると、プログラムの中でその領域を解放しなければならない。もし、解放し忘れると、プログラムが終了しても占有したメモリがリークした状態、つまり、メモリが使用できない状態となる。システムをシャットダウンしない限り、そのメモリ領域は使用できない。

ここでは、動的領域の解放方法を述べる。動的に確保した領域を解放するには、以下のように DEALLOCATE 文を用いる。

```
c――――――――――★          配列の動的領域を解放する（その１）
      if(M_alloc(1).eq.1)then
      DEALLOCATE (Point,Element,Member,Max_disp,Max_stress)        !ok
      DEALLOCATE (fll_static_point,am_point,fll_force_point)       !ok
      DEALLOCATE
      (am_member,rot_memb_t,rot_memb,ak_linear,ak_nonlinear)       !ok
      N=Parameter_C.n_local_coord
      if(N.ne.0) DEALLOCATE (rot_local)       !ok
      endif
```

ここで、注意すべきは、記憶領域を実際に確保せずに、この動的領域の解放を行わないことである。動的確保をせずして解放を行うと重大なシステムエラーとなる。そのため、プログラム内で如何なる配列や構造

体が動的に領域確保を行ったか、あるいは確保しなかったかを常に意識する必要がある。例えば、入力データに不備があり、解析途中で終了しなければならない場合、どの変数が動的領域確保されたかを知っておくことが大切となる。SPACE では、上の記述で示すように、配列 M_alloc で管理している。この管理技法を、具体的に後節で説明する。

2.2.4 動的領域の確保の復習

本節では、動的領域確保の復習を行う。使用するプログラムは、両端ファイバーの部材モデルに関する線形剛性の縮合計算を行うコードの一部である。ファイル名は、Cal_lin_stiff_M11.f であり、そのコードを以下に示す。このサブルーチンでは、動的領域変数の宣言、確保、解放を行い、その動的領域をワーク領域として使用している。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Cal_lin_stiff_M11
C       ―――――――――――――――――――――――――――――――――
C     ●   代表的な部材モデルの剛性(両端ファイバーモデル)
C     ―――――――――――――――――――――――――――――――――
      subroutine Cal_lin_stiff_M11(Model_type,Member,Element,ak,
     *      E_model11, E_model_fiber,
     *      M_model11, M_model_fiber,
     *      Bilinear_work,Trilinear_work,Concrete_work)
C     ―――――――――――――――――――――――――――――――――
      implicit real*8(A-H,O-Z)
      include "submain.h"
      include "submainx.h"
      record / member_s    / Member
      record / element_s   / Element
      record / n_model_s   / Model_type
      record / E_model11_s / E_model11
      record / E_model_fiber_s    / E_model_fiber
      record / M_model11_s        / M_model11
      record / M_model_fiber_s    / M_model_fiber
      record / Bilinear_work_s    / Bilinear_work
      record / Trilinear_work_s   / Trilinear_work
      record / Concrete_work_s    / Concrete_work
      dimension E_model_fiber(*),M_model_fiber(*)
      dimension E_model11(*),M_model11(*)
      dimension ak(12,12),akk(12,12)
      real*8, ALLOCATABLE :: c(:,:),ab(:,:),ba(:),alength(:)
      integer, ALLOCATABLE :: irest_Point(:,:),n_type(:)
C     ―――――――――――――――――――――――――――――――――
      iet = Member.n_model                    ! モデルタイプ番号
      n_div = Model_type.n_div_model(iet)     ! 部材分割数
      imm= Element.n_element                  ! 要素番号
      immm= Member.n_model_type               ! モデルタイプ別番号
      n_if = 6*(n_div-1)                      ! 内部自由度
c―――――――――――――――――――――――――――――――――★動的記憶領域の確保
```

```
      ALLOCATE (
     *   irest_Point(6,n_div+1),n_type(n_div),alength(n_div)
     *         )
     c――――――――――――――――――――――――――――――――★節点拘束表の作成
     c                                                  未知数等をセット
      call set_model11_dat(irest_Point,n_if,n_div,iubw,
     *         Element,Member,
     *         n_type,alength,
     *  Member.i_rigid_length,    ! i端剛域
     *  Member.j_rigid_length,    ! j端剛域
     *  Member.i_shear_G,         ! i 端せん断剛性
     *  Member.j_shear_G)         ! j 端せん断剛性
     c――――――――――――――――――――――――――――――――★動的記憶領域の確保
      ALLOCATE (
     *   c(0:iubw,n_if),ab(n_if,12),ba(n_if)
     *         )
     c――――――――――――――――――――――――――――――――★剛性行列のゼロクリア
      do i=1,12
      do j=1,12
      ak(j,i)=0.
      enddo
      enddo
      do i=1,n_if
      do j=0,iubw
      c(j,i)=0.
      enddo
      enddo
      do i=1,12
      do j=1,n_if
      ab(j,i)=0.
      enddo
      enddo
      do i=1,n_div
      Member.an_stress(i)=0.
      enddo
      do i=1,n_div
      Member.an_vv(i)=0.
      Member.an_ww(i)=0.
      enddo
     c――――――――――――――――――――――――――――――――★部材剛性行列の作成
      it = 0
      do i=1,n_div
      call Stiff_M11_I(i,it,n_type(i),akk,Member,alength,
     *       Model_type,Element,
     *       E_model11(imm), E_model_fiber,
     *       M_model11(immm), M_model_fiber,
     *       Bilinear_work,Trilinear_work,Concrete_work)
           call Bnd_FEM(i,akk,irest_Point,ak,c,ab,iubw,n_if)
      enddo
     c――――――――――――――――――――――――――――――――★部材剛性行列の縮合
      call Typical_member_model(c,ab,ak, n_if,n_if,iubw,iubw,ba,ier)
     c――――――――――――――――――――――――――――――――★両端の剛域処理
      call Deal_Rigid_element(ak,Member.i_rigid_length,
     *                        Member.j_rigid_length)
```

```
c―――――――――――――――――――――――――――――――★動的記憶領域の解放
      DEALLOCATE (
     *   c ,ab ,ba,irest_point,n_type,alength
     *         )
      return
      end
```

上記の Cal_lin_stiff_M11()というサブルーチンは、部材モデルの縮合剛性を得る一般的なコード記述となっている。動的領域のワークエリアとして、６つの配列が用いられており、各部材モデルごとに配列の大きさを変化させて領域を動的確保している。

このサブルーチンの最初で、構造体を定義したヘッダーファイルをインクルードしている。

```
      include "submain.h"
      include "submainx.h"
```

次に、record 文で構造体を実際の変数名に割り付けている。

```
      record / member_s    / Member
      record / element_s   / Element
      record / n_model_s   / Model_type
```

このサブルーチンでは、既に Member 等は領域を確保されているため、ここでは、動的領域として確保する必要がない。ワーク領域として、次の配列が動的領域として宣言される。

```
      real*8, ALLOCATABLE :: c(:,:),ab(:,:),ba(:),alength(:)
      integer, ALLOCATABLE :: irest_Point(:,:),n_type(:)
```

さらに、下に示すような２つの ALLOCATE 文で、動的領域を確保している。

```
c―――――――――――――――――――――――――――――――★動的記憶領域の確保
      ALLOCATE (
     *   irest_Point(6,n_div+1),n_type(n_div),alength(n_div)
     *         )
c―――――――――――――――――――――――――――――――★動的記憶領域の確保
      ALLOCATE (
     *   c(0:iubw,n_if),ab(n_if,12),ba(n_if)
     *         )
```

このワーク領域を計算で利用した後、サブルーチンを抜ける前に、この領域を以下のように解放する。

```
c―――――――――――――――――――――――――――――――★動的記憶領域の解放
      DEALLOCATE (
```

```
*   c ,ab ,ba,irest_point,n_type,alength
*           )
```

以上が動的領域の確保と解放に関する処理の全てである。この復習を通して、動的領域の確保と解放の手続きを確認されたい。

動的領域確保の例として、代表的な部材モデルの線形剛性(両端ファイバーモデル)を求めるサブルーチンを示した。折角なので、ここで、その内容について少しだけ触れておこう。このサブルーチンは、SPACEの重要な部材モデルである静的縮合型ファイバーモデルの線形剛性行列を作成する。その中で主要なサブルーチンは以下の５つである。静的縮合モデルについては、後章で詳細に説明するので、コメント行を参照して、概略を理解しておけばよい。

```
c―――――――――――――――――――――――――――――――★節点拘束表の作成
      call set_model11_dat(  )      ! 部材モデル作成、未知数等をセット
      do i=1,n_div                  ! 部材モデル内のエレメント数分繰り返す
      call Stiff_M11_I( )           ! 線形の各エレメント剛性作成
      call Bnd_FEM( )               ! エレメント剛性行列の組み込み
      enddo
c―――――――――――――――――――――――――――――――★部材剛性行列の静的縮合
      call Typical_member_model(  ) ! ak:部材剛性行列
c―――――――――――――――――――――――――――――――★両端の剛域処理
      call Deal_Rigid_element(  )   ! 剛域処理を行う
```

これらのサブルーチンによって、各部材モデルに関する縮合された剛性行列が得られる。

2.2.5 動的領域の保護

ローカル変数あるいは動的領域は、一度定義したサブルーチンから抜けると、再度アクセスできなかったり、値を変更されたりする場合がある。重要な変数や配列で、変数領域の内容を変更されたくない場合は、save 文を用いて、その内容を保護する必要がある。

前節で示した構造体に関する save 文を示そう。

```
      save Point,Element,Member,Max_disp,Max_stress
```

SPACE システムでは、多くの変数、配列、構造体で動的に確保した領域は、この save 文で内容を保護している。詳細は、付録 submain.f の内容を参照されたい。

2.3 構造体

2.3.1 構造体の定義

本節では構造体の使用方法について説明する。詳しい仕様は、FORTRAN90 の文法書[1)]を参照すると良い。以下に、動的解析システム SPACE で使用している構造体の一部で、control_s、dynamic_load_s、parameter_s、n_model_s、newmark_s の５つが示されている。これらは、SPACE を制御するためのデータ群を各々の仕様に合わせてまとめたものである。どのような変数、パラメータがあるかよく読んで記憶しておいて欲しい。また、これら構造体は、ヘッダーファイル submain.h で定義されており、これらの構造体を必要とするとき、そのルーチン内でインクルードすることになる。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /submain.h
C     ―――――――――――――――――――――――――――――――――
C     ●   構造体の定義用ヘッダーファイル
C     ―――――――――――――――――――――――――――――――――
      parameter(n_div1 = 6 ,nm_div1 = 32, n_free=6, nx_divfree = 42)
C
C     ―――――――――――――――――――――――――――――――――
C     ●   control_s 構造体
C     ―――――――――――――――――――――――――――――――――
c
      structure  / control_s/
      integer    type_analysis        !  解析種別
      integer    analysis_3D          !  解析型 0:3D 1:2D(x-z) 2:2D(y-z)
      integer    init_disp            !  初期変位ありか
      integer    init_stress          !  初期応力ありか
      integer    init_imperfection    !  形状初期不整ありか
      real*8     amp_imperfection     !  初期不整の大きさ
      real*8     collapse_maxdisp     !  解析最大変位
      real*8     out_start_sec        !  データ出力の最初の時刻（現在使用不可）
      integer    interval_out         !  データ出力間隔
      integer    jikuzero             !  軸剛性ゼロチェック
      real*8     jikuzero_alph        !  軸剛性ゼロチェック用剛性係数
      end structure
c      record /control_s/ Control
c
C     ―――――――――――――――――――――――――――――――――
C     ●   dynamic_load_s 構造体
C     ―――――――――――――――――――――――――――――――――
c
      structure / dynamic_load_s/
      integer    load_point(3)        !  静的節点荷重ありか
      real*8     dt_load_point(3)     !  静的節点荷重の増分時間
      integer    n_load_point         !  静的節点荷重ファイルの最大個数
      integer    load_dynamic(3)      !  地震荷重ありか
      real*8     amp_load_dynamic(3)  !  地震荷重の大きさ
      real*8     dt_load_dynamic(3)   !  地震荷重の増分時間
      integer    n_load_dynamic       !  地震荷重ファイルの最大個数
      integer    load_mass            !  整合質量ありか
```

```
      end structure
c      record / dynamic_load_s / Dynamic_load
c
C     ―――――――――――――――――――――――――――――――――
C     ●   parameter_s 構造体
C     ―――――――――――――――――――――――――――――――――
c    解析パラメータ
      structure  / parameter_s/
      integer    n_unknown       !  全自由度
      integer    n_point         !  節点数
      integer    n_element       !  要素数
      integer    n_element_dll   !  DLL 用要素数
      integer    n_member        !  部材数
      integer    n_rot_axis      !  主軸回転部材数
      integer    n_local_coord   !  局所座標系を使用する節点数
      integer    n_boundary_p    !  境界節点数
      integer    nc_member       !  部材減衰機構を有する部材数
      integer    n_member_dll    !  DLL 用部材数
      integer    n_free          !  節点当たり解析自由度数
      integer    n_dim           !  解析次元数
      integer    n_skyline       !  スカイライン行列の領域数
      integer    n_sky_ave       !  平均バンド幅
      end structure
c      record /parameter_s/ Parameter_C
c
C     ―――――――――――――――――――――――――――――――――
C     ●   n_model_s 構造体
C     ―――――――――――――――――――――――――――――――――
c
c  モデルパラメータ
      structure  / n_model_s/
      integer    n_e_models          ! 要素モデルの最大数
      integer    no_e_model(100)     ! 要素モデルの番号
      integer    n_div_model(100)    ! 要素モデルの分割数
      integer    n_e_model(100)      ! 要素モデルの数
      integer    n_m_model(100)      ! 部材モデルの数
      integer    n_damp(100)         ! 部材減衰
      integer    n_m_damp            ! 全部材減衰数
      integer    nm_div_fmodel       ! ファイバー要素の最大数
      integer    nm_div_felement     ! ファイバー要素のエレメント最大数
      integer    n_m_bilinear        ! ファイバー要素バイリニア用の最大要素数
      integer    n_m_trilinear       ! ファイバー要素トリリニア用の最大要素数
      integer    n_m_Concrete        ! ファイバー要素コンクリート用の最大要素数
      integer    n_m_analogy         ! アナロジー要素の最大要素数
      integer    nm_div_msmodel      ! ms 要素の最大数
      integer    nm_div_mselement    ! ms 要素のエレメント最大数
      integer    n_m_ro_model        ! せん断型モデルで使用する RO モデルの総数
      integer    n_m_filter          ! Maxwell 用フィルターの設計あり、なし
      integer    n_spring            ! MSS モデルのばね要素の数
      real*8     cosin(2,16)         ! MSS モデルの角度係数を入れるワークエリア
      end structure
c      record / n_model_s / Model_type
c
C     ―――――――――――――――――――――――――――――――――
```

```
C     ●   newmark_s 構造体
C     ―――――――――――――――――――――――――――――――――
c
c    ニューマークβ法のパラメータ
      structure  / newmark_s/
      real*8     total_T     !  解析時間
      real*8     f1_T        !  第１段階解析時間
      real*8     f2_T        !  第２段階解析時間
      integer    nn_step     !  解析ステップ数
      integer    n2_step     !  第２段階開始ステップ数
      integer    n_damp_type ! 減衰タイプ　1:質量比例　2:剛性比例　3:レーリー型 4:その他
      integer    n_damp_1    !  減衰定数決定用第一モード番号
      integer    n_damp_2    !  減衰定数決定用第二モード番号
      real*8     alf1_1      !  第１段階レーリー減衰パラメータ１
      real*8     alf1_2      !  第１段階レーリー減衰パラメータ 2
      real*8     alf2_1      !  第２段階レーリー減衰パラメータ１
      real*8     alf2_2      !  第２段階レーリー減衰パラメータ 2
      real*8     dt          !  増分時間Δt
      real*8     dt2         !  増分時間Δt の 2 乗
      real*8     beta        !  ニューマークのβ
      real*8     delta       !  ニューマークのδ
      real*8     ddt         !  δ・Δt
      real*8     bdt         !  β・Δt2
      real*8     ddt_1       !  (1 -δ)Δt
      real*8     bdt_5       !  (0.5 -β)Δt2
      real*8     dt5         !  0.5・Δt2
      real*8     eps_v       !  収束判定閾値
      integer    max_repeat  !  収束計算のための最大反復回数
      real*8     gumma       !  収束計算のための収束予測係数
      end structure
c      record / newmark_s / Newmark_P
```

構造体の定義は、２つの予約語 structure と end structure によって行われる。structure の後の/で囲まれた control_s は、この構造体の名前を表す。構造体の成分は 2 つの予約語の間に、変数もしくは配列の名前とその型を用いて定義することになる。例えば以下の用である。

```
structure  / control_s/
integer    type_analysis        !  解析種別
integer    analysis_3D          !  解析型 0:3D 1:2D(x-z) 2:2D(y-z)
  •
  •
end structure
```

ここで示した構造体は、SPACE の動的解析を行う上で大切なパラメータを集めたものであり、その内容はコメントで与えられている。構造体 control_s は解析モデルを表すパラメータの集合であり、また、構造体 dynamic_load_s は解析方法に関するパラメータを、parameter_s は解析モデルの詳細を、n_model_s は使用する部材モデルの個数等の内容を表

す。最後に、newmark_s は SPACE で動的解析手法として用いているニューマークβ法に関するパラメータを集めている。どのようなパラメータが設定されているか、記憶しておいて頂きたい。

2.3.2 構造体の使用法

前節で定義した構造体の使用法について述べる。構造体の使用は、まず、構造体を定義し、続いて、その構造体を実体のある構造体名に割り付けるという手続きが必要である。以下にその手続きを、動的解析システム SPACE の数値解析部分の submain.f を用いて説明する。以下に、ファイル submain.f の中の構造体に関連する部分を抜き出す。最初の部分はサブルーチン名とその引数を表す。このサブルーチンは C++とインターフェイスがとられており、引数はグラフィック用のデータ領域である。これらについては、第 8 章で詳しく説明する。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /submain_dynamic_a
C       ―――――――――――――――――――――――――――――――
C     ●   動的解析主プログラム（反復解法＋陰解法）Ver.2.10
C     ―――――――――――――――――――――――――――――――――c
      subroutine submain_dynamic_a(i_calnum,iend_code,icontrol,ierr_dat,
     *           T,dt,n_step,ns_step,d_max_v,id_max_v,
     *           i_read_disp,F_disp,i_read_ndbalanceF,F_ndbalanceF,
     *           i_read_spring,F_fay,F_n_spring,F_my_spring,
     *           F_mz_spring,i_stat_spring,n_iterate,
     *           nm_iterate,numb_method)
c―――――――――――――――――――――――――――――――――――――――――
      implicit  real*8(A-H,O-Z)
c――――――――――――――――――――――――――――――――★構造体の定義ヘッダーファイル
      include "submain.h"
      include "submainx.h"
c―――――――――――――――――――――――――――――――――――――――――
c
c    ★        構造体定義
c
c―――――――――――――――――――――――――――――――――――――――――
      record /control_s      / Control
      record /dynamic_load_s / Dynamic_load
      record /parameter_s    / Parameter_C
      record /n_model_s      / Model_type
      record /newmark_s      / Newmark_P
      record /out_section_s  / Out_section
      record /point_s        / Point
      record /element_s      / Element
      record /member_s       / Member
      record /max_disp_s     / Max_disp
      record /max_stress_s   / Max_stress
c―――――――――――――――――――――――――――――――――――――――――
c
```

```
c    ★        構造体の動的領域宣言
c
c-------------------------------------------------------------
      ALLOCATABLE :: Bilinear_work(:)
      ALLOCATABLE :: Trilinear_work(:)
      ALLOCATABLE :: Concrete_work(:)
      ALLOCATABLE :: Point(:)
      ALLOCATABLE :: Element(:)
      ALLOCATABLE :: Member(:)
      ALLOCATABLE :: Max_disp(:)
      ALLOCATABLE :: Max_stress(:)
c-------------------------------------------------------------
c
c    ★        制御情報を構造体にセット
c
c-------------------------------------------------------------
      call Set_control(Control,N_analysis,NINDIT,GINDIS,
     *                 IWSTP,SOUTSC,DMAXCK,in_disp,in_stres,
     *                 IT_ANALYS,JIKUZERO,G_JIKUZERO_ALPH)
      call Set_model_type(Model_type)
      call Set_newmark(Newmark_P,F1SEC,F2SEC,HH(1),HH(2),
     *                 QHH(1),QHH(2),DELT,BETA,EPSDSP,NNTIME,GUMMA,
     *                 ITYDP,NDMP,NDMP2)
      call Set_dynamic_load(Dynamic_load,IST,IGRA,
     *                      XGAL,load_memb_mass)

c-------------------------------------------------------------
c
c    ★        反復計算開始
c
c-------------------------------------------------------------
      n_roop=Newmark_P.max_repeat
      do iroop=1,n_roop
```

サブルーチン名の下の implicit 文で、SPACE の動的解析は、全て 2 倍精度の実数を用いることを宣言している。その下の include 文では、2 つのヘッダーファイルをインクルードしている。これら 2 つのヘッダーファイルでは、システムで使用している構造体の全てを定義している。

構造体定義では、定義した構造体と使用するサブルーチン内での実体名を割り付けている。割付法は、

```
record /newmark_s      / Newmark_P
```

であり、予約語は record である。ここで構造体名は newmark_s、実体名は Newmark_P であり、それらの構造体の内容は、ヘッダーファイル submain.h で定義されている。次に、この構造体が動的配列領域であることを宣言する。予約語は ALLOCATABLE 文であり、次のように動的配列領域であることを宣言する。

```
ALLOCATABLE :: Concrete_work(:)
```

次に、構造体中の成分にアクセスする方法を示す。以下に示すように、構造体の実体名と構造体内の変数名をピリオドで結合して、その成分にアクセスする。もちろん、データの書き込みも同様に行えばよい。

```
n_roop = Newmark_P.max_repeat
```

構造体が配列の場合は、その配列番号を構造体名の配列要素とすればよい。例えば、次のように用いる。

```
ij=Point(i).local_coord
```

また、構造体の成分が配列の場合は、以下のように使用する。

```
irest =Point(i).irest(j)
```

サブルーチンの引数として構造体を受け渡す場合について説明する。例えば、SPACE における制御情報を構造体にセットする部分が参考になる。その一部であるサブルーチンコールを以下に示す。

```
      call Set_newmark(Newmark_P,F1SEC,F2SEC,HH(1),HH(2),
     *                 QHH(1),QHH(2),DELT,BETA,EPSDSP,NNTIME,GUMMA,
     *                 ITYDP,NDMP,NDMP2)
```

上記のサブルーチンは、ファイルから取得した動的解析用パラメータを構造体 newmark_s にセットする処理を行う。構造体の受け渡しは、構造体の実体名、ここでは第一引数の Newmark_P のようにセットすれば良い。受け渡された構造体の利用法については、次節で述べる。

2.3.3 構造体のサブルーチン内使用法

前節で、例として使用したサブルーチン Set_newmark()を用いて、構造体の手続き、並びに使用法について解析する。以下にサブルーチン Set_newmark()の全てを示す。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Set_newmark
C       ―――――――――――――――――――――――――――――――――
C     ●   構造体 Newmark_P の値セット(ok)
C     ―――――――――――――――――――――――――――――――――
      subroutine Set_newmark(Newmark_P,t1,t2,bet1_1,bet1_2,
     *        bet2_1,bet2_2,dt,beta,eps_v,iroop,gumma,
     *        n_damp_type,n_damp_1,n_damp_2)
C     ―――――――――――――――――――――――――――――――――
      implicit real*8(A-H,O-Z)
```

```
      include "submain.h"
      record / newmark_s / Newmark_P
C     ―――――――――――――――――――――――――――――――――
c     ニューマークβ法のパラメータ
c      structure  / newmark_s/
c      real*8    total_T    !  解析時間
c      real*8    f1_T       !  第１段階解析時間
c      real*8    f2_T       !  第２段階解析時間
c      integer   nn_step    !  解析ステップ数
c      integer   n2_step    !  第２段階開始ステップ数
c      integer   n_damp_type! 減衰タイプ　1:質量比例　2:剛性比例　3:レーリー型 4:その他
c      integer   n_damp_1   !  減衰定数決定用第一モード番号
c      integer   n_damp_2   !  減衰定数決定用第二モード番号
c      real*8    alf1_1     !  第１段階レーリー減衰パラメータ１
c                              ここでは減衰定数をセットする（hh1）
c      real*8    alf1_2     !  第１段階レーリー減衰パラメータ 2
c                              ここでは減衰定数をセットする（hh2）
c      real*8    alf2_1     !  第２段階レーリー減衰パラメータ１
c                              ここでは減衰定数をセットする（qhh1）
c      real*8    alf2_2     !  第２段階レーリー減衰パラメータ 2
c                              ここでは減衰定数をセットする（qhh2）
c      real*8    dt         !  増分時間Δt
c      real*8    dt2        !  増分時間Δt の 2 乗
c      real*8    beta       !  ニューマークのβ
c      real*8    delta      !  ニューマークのδ
c      real*8    ddt        !  δ・Δt
c      real*8    bdt        !  β・Δt2
c      real*8    ddt_1      !  (1 -δ)Δt
c      real*8    bdt_5      !  (0.5 -β)Δt2
c      real*8    dt5        !  0.5・Δt2
c      real*8    eps_v      !  収束判定閾値
c      integer   max_repeat !  収束計算のための最大反復回数
c      real*8    gumma      !  収束計算のための収束予測係数
c      end structure
c      record / newmark_s / Newmark_P
c
c      t1                       第１段階解析時間
c      t2                       第２段階解析時間
c      bet1_1                   第１段階減衰定数（hh1）
c      bet1_2                   第１段階減衰定数（hh2）
c      bet2_1                   第２段階減衰定数（qhh1）
c      bet2_2                   第２段階減衰定数（qhh1）
c      dt                       増分時間Δt
c      beta                     ニューマークのβ
c      eps_v                    収束判定閾値
c      iroop                    収束計算のための最大反復回数
c      gumma                    収束計算のための収束予測係数
c      n_damp_type              減衰タイプ　1:質量比例　2:剛性比例　3:レーリー型 4:その他
c      n_damp_1                 減衰定数決定用第一モード番号
c      n_damp_2                 減衰定数決定用第二モード番号
C     ―――――――――――――――――――――――――――――――――
      delta             =  0.5
      if(beta.gt.0.3) delta = 0.6
      Newmark_P.f1_T        =  t1
```

```
Newmark_P.f2_T        =  t2
Newmark_P.total_T     =  t1 + t2
Newmark_P.n2_step     =  t1/dt + 1
Newmark_P.nn_step     =  (t1 + t2)/dt + 1
Newmark_P.n_damp_type =  n_damp_type
Newmark_P.n_damp_1    =  n_damp_1
Newmark_P.n_damp_2    =  n_damp_2
Newmark_P.alf1_1      =  bet1_1
Newmark_P.alf1_2      =  bet1_2
Newmark_P.alf2_1      =  bet2_1
Newmark_P.alf2_2      =  bet2_2
Newmark_P.dt          =  dt
Newmark_P.beta        =  beta
Newmark_P.delta       =  delta
Newmark_P.eps_v       =  eps_v
Newmark_P.max_repeat  =  iroop
Newmark_P.gumma       =  gumma
Newmark_P.dt2         =  dt*dt
Newmark_P.ddt         =  delta*dt
Newmark_P.bdt         =  beta*dt*dt
Newmark_P.ddt_1       =  (1. - delta)*dt
Newmark_P.bdt_5       =  (0.5- beta )*dt*dt
Newmark_P.dt5         =  0.5*dt*dt
write(76,'(//a)') ' 構造体：Newmark_P '
write(76,*) Newmark_P.f1_T        ,'=  t1'
write(76,*) Newmark_P.f2_T        ,'=  t2'
write(76,*) Newmark_P.total_T     ,'=  t1 + t2'
write(76,*) Newmark_P.n2_step     ,'=  t1/dt + 1'
write(76,*) Newmark_P.nn_step     ,'=  (t1 + t2)/dt + 1'
write(76,*) Newmark_P.n_damp_type ,'=  n_damp_type'
write(76,*) Newmark_P.n_damp_1    ,'=  n_damp_1'
write(76,*) Newmark_P.n_damp_2    ,'=  n_damp_2'
write(76,*) Newmark_P.alf1_1      ,'=  bet1_1'
write(76,*) Newmark_P.alf1_2      ,'=  bet1_2'
write(76,*) Newmark_P.alf2_1      ,'=  bet2_1'
write(76,*) Newmark_P.alf2_2      ,'=  bet2_2'
write(76,*) Newmark_P.dt          ,'=  dt'
write(76,*) Newmark_P.beta        ,'=  beta'
write(76,*) Newmark_P.delta       ,'=  delta'
write(76,*) Newmark_P.eps_v       ,'=  eps_v'
write(76,*) Newmark_P.max_repeat  ,'=  iroop'
write(76,*) Newmark_P.gumma       ,'=  gumma'
write(76,*) Newmark_P.dt2         ,'=  dt*dt'
write(76,*) Newmark_P.ddt         ,'=  delta*dt'
write(76,*) Newmark_P.bdt         ,'=  beta*dt*dt'
write(76,*) Newmark_P.ddt_1       ,'=  (1. - delta)*dt'
write(76,*) Newmark_P.bdt_5       ,'=  (0.5- beta )*dt*dt'
write(76,*) Newmark_P.dt5         ,'=  0.5*dt*dt'
return
end
```

構造体を使用するサブルーチンは、次の手続きが必要となる。まず、使用する構造体を定義する。個々では、ヘッダーファイル submain.h に

構造体の定義が行われているので、以下のようにインクルードすればよい。

```
include "submain.h"
```

次に、構造体の実体名（オブジェクト名）と構造体の名前を割り付ける必要がある。実体はサブルーチンの引数として受け渡されており、ここでは、record 文を用いて、以下のように行う。

```
record / newmark_s / Newmark_P
```

これで、構造体と構造体の実体名が割り付けられたことになる。構造体内部の変数にアクセスする場合は、この実体名を用いて、以下のように行えば良い。

```
Newmark_P.f1_T      =  t1
Newmark_P.f2_T      =  t2
Newmark_P.total_T   =  t1 + t2
```

解析制御情報を構造体へセットするサブプログラム、Set_control、Set_model_type、Set_dynamic_load の内容を、付録に収録されているプログラムで確かめておいて頂きたい。

2.3.4 構造体配列の動的確保と使用法

構造体も配列として使用でき、さらに、その配列も動的領域として定義することが可能である。その例を以下に示そう。

```
c    ★ 1      構造体定義ファイルをインクルード
      include "submain.h"

c    ★ 2      構造体を実体名に割り付ける
      record /point_s         / Point
      record /element_s       / Element
      record /member_s        / Member

c    ★ 3      構造体の動的領域宣言
      ALLOCATABLE :: Point(:)
      ALLOCATABLE :: Element(:)
      ALLOCATABLE :: Member(:)

c    ★ 4      構造体の大きさを動的確保する
      ALLOCATE (Point(Parameter_C.n_point))
      ALLOCATE (Member(Parameter_C.n_member))
      ALLOCATE (Element(Parameter_C.n_element))

c    ★ 5      構造体である動的配列の解放
      DEALLOCATE (Point,Element,Member)
```

上記のように５つの手続きを行うことによって、構造体の配列を使用することができる。次に、動的に領域確保した構造体の使用法について述べる。例として用いるサブルーチンは Cal_member_length()であり、ここでは、部材の長さを計算し、構造体の成分 alength にセットする。その内容を以下に示す。

```
C     ――――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Cal_member_length
C       ――――――――――――――――――――――――――――――――――
C     ●   部材の長さ（節点間距離）を計算する(ok)
C     ――――――――――――――――――――――――――――――――――
      subroutine Cal_member_length(Member,Point,Parameter_C)
      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(*)
c――――――――――――――――――――――――――――――――――――――――――
c     Parameter_C     :structure
c     Member          :structure
c     Point           :sturcture
c――――――――――――――――――――――――――――――――――――――――――
      do i=1,Parameter_C.n_member
      al=0.
      i1 = Member(i).nm_point(1)
      j1 = Member(i).nm_point(2)
      do j=1,3
      al=al + (Point(i1).coord(j) - Point(j1).coord(j))**2
      end do
      Member(i).alength = dsqrt(al)
      end do
      return
      end
```

このサブルーチンの呼び出し法は、

```
call Cal_member_length(Member,Point,Parameter_C)
```

であり、引数としての構造体は、構造体の実体名を与えている。受け渡された構造体は、以下のように設定することになる。

```
include "submain.h"
record / member_s     / Member
record / point_s      / Point
```

もちろん、上記の構造体は、以下のように整合配列で定義される。

```
dimension Member(*),Point(*)
```

配列構造体の成分へのアクセスは、

```
i1 = Member(i).nm_point(1)
al = al + (Point(i1).coord(j) - Point(j1).coord(j))**2
```

で行われる。上の例では、構造体 Member の i 番目で、その配列成分 nm_point の１番目を表す。これは、i 部材目の１端の節点番号を表す。また、下の例では、構造体 Point の i1 番目で、その配列要素 coord の j 番目を表す。これは、i1 節点の j 方向の位置座標を表す。

構造体を引数としたサブルーチンの呼び出し方として次のような方法も使用される。

```
      call Cal_nonlin_stiff_M1(Member(i),Element(ie),
     *                         Member(i).stress(7),ak )
```

第１引数と第２引数は、配列構造体の第 i 番と第 ie 番目の構造体を受け渡している。また、第３番目の引数は、第 i 番目の構造体に関する配列 stress の第７番目を実数変数として受け渡している。呼ばれるサブルーチンを以下に示す。このサブルーチンでは、弾性立体フレーム部材の接線剛性行列を求める。サブルーチンの第 1、第 2 仮引数は構造体、第 3、第 4 引数は実変数となっている。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Cal_nonlin_stiff_M1
C       ―――――――――――――――――――――――――――――――――
C     ●   弾性立体フレーム部材の接線剛性行列の計算(ok)
C     ―――――――――――――――――――――――――――――――――
      subroutine Cal_nonlin_stiff_M1(Member ,Element ,an, ak )
      implicit real*8(A-H,O-Z)
      include "submain.h"
      record / member_s    / Member
      record / element_s   / Element
      dimension ak(12,12)
C     ―――――――――――――――――――――――――――――――――
c        an: 部材の軸力
c        ak: 部材の接線剛性行列
C     ―――――――――――――――――――――――――――――――――
      call Cal_lin_stiff_M1(Member,Element,ak)  ! 線形剛性を計算する
      call Cal_geomet_stiff(an,Member,ak)       ! 幾何学的非線形剛性を計算し、線形剛性に加える
      EA=Element.A*Element.E                    ! 部材の断面積＊ヤング係数
      call Create_Kn(ak,Member.an_vv(1),Member.an_ww(1),  ! 大変位剛性行列を計算し、線形剛性に加える
     *               EA,Member.alength )
      return
      end
```

構造体の使用法として、サブルーチンを呼ぶ方と呼ばれる方とで、構造体の内容が異なって使用することも可能である。ただし、構造体成分の数やその型等は同じでなくてはならない。その例を以下に示す。まず、呼び出し側サブルーチンの例として、Get_nonlinear_stiff()をあげる。このサブルーチンは、各種部材モデルの接線剛性行列を求めるもので、これについては、後章で詳細に述べる。ここでは、その内容の一部を示すに留める。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Get_nonlinear_stiff
C       ―――――――――――――――――――――――――――――――――
C     ●   接線剛性行列の計算(ok)
C     ―――――――――――――――――――――――――――――――――
      subroutine Get_nonlinear_stiff(N_analysis,
     *     ak_nonlinear,Member,n_member,
     *     Model_type,Element,past_disp_point,disp_point,rot_memb,
     *     E_model6_real,E_model7_real,E_model_fiber,M_model_fiber,
     *     E_model11, M_model11,
     *     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,
     *     MSS_work,
     *     work1_elememt,work2_elememt, work1_member, work2_member)
C     ―――――――――――――――――――――――――――――――――
      implicit real*8(A-H,O-Z)
      include "submain.h"
      include "submainx.h"
      record / member_s    / Member
      record / element_s   / Element
      record / n_model_s   / Model_type
c――――――――――――――――――――――――――――――――――★Model_No.1 通常の有限要素弾塑性モデル
c
c――――――――――――――――――――――――――――――――――★Model_No.2 3 次元せん断弾塑性モデル
c      record /element2_s / Element
c      record /member2_s  / Member
c――――――――――――――――――――――――――――――――――★Model_No.3 3 次元軸力弾塑性モデル
c      record / N_Buckling_s  /N_Buckling
c――――――――――――――――――――――――――――――――――★Model_No.4 3 次元ケーブル弾塑性モデル
c      record /element4_s / Element
c      record /member4_s  / Member
c――――――――――――――――――――――――――――――――――★Model_No.5 3 次元免振モデル
c      record /element5_s / Element
c      record /member5_s  / Member
       record / MSS_work_s   / MSS_work
c――――――――――――――――――――――――――――――――――★Model_No.6 3 次元制震 Maxwell モデル
c      record /element6_s         / Element
```

```
      record / E_model6_real_s   / E_model6_real
c-----------------------------------★Model_No.7 3 次元バネモデル
c     record /element7_s / Element
c     record /member7_s  / Member
c     record / E_model7_real_s    / E_model7_real
c-----------------------------------★Model_No.11 両端ファイバーモデル
      record / E_model11_s    / E_model11
      record / M_model11_s    / M_model11
c-----------------------------------★Model_No.12 両端、中央ファイバーモデル
      record / E_model12_s    / E_model12
      record / M_model12_s    / M_model12
c-----------------------------------★Model_No.13 両端ピン、中央ファイバーモデル
      record / E_model13_s    / E_model13
      record / M_model13_s    / M_model13
c-----------------------------------★Model_No.15 両端ファイバーFEM モデル
      record / E_model15_s    / E_model15
      record / M_model15_s    / M_model15
c-----------------------------------★Model_No.21 両端 MS モデル
      record / E_model21_s    / E_model21
      record / M_model21_s    / M_model21
c-----------------------------------★Model_No.22 両端、中央 MS モデル
      record / E_model22_s    / E_model22
      record / M_model22_s    / M_model22
c-----------------------------------★Model_No.31 両端アナロジーモデル
      record / E_model31_s    / E_model31
      record / M_model31_s    / M_model31
c-----------------------------------★Model_No.32 両端、中央アナロジーモデル
      record / E_model32_s    / E_model32
      record / M_model32_s    / M_model32
c-----------------------------------★Model_No.33 両端、中央アナロジーモデル
      record / E_model33_s    / E_model33
      record / M_model33_s    / M_model33
c-----------------------------------★Model_No.51 3 次元プレテンション動作モデル
c     record / E_model51_s    / E_model51
c     record / M_model51_s    / M_model51
c-----------------------------------★ファイバーモデル用エリア
      record / E_model_fiber_s   / E_model_fiber
      record / M_model_fiber_s   / M_model_fiber
c-------------------------------------------------------------
      dimension Member(*),Element(*),MSS_work(*)
      dimension ak_nonlinear(12,12,*),rot_memb(3,3,2,*)
      dimension work1_elememt(*),work2_elememt(*),
     *          work1_member(*), work2_member(*)
      dimension past_disp_point(*),disp_point(*),v(12),ak(12,12)
      dimension vv(12)
C     ---------------------------------------------
      if(N_analysis.eq.7) return     !   線形解析;7
      do  9999 i=1,n_member
      if(Member(i).nm_analysis .eq. -1) goto 9999 ! 各部材の解析種別のチェック(-1:線形解析)
c-----------------------------------★部材両端の変位取得
      do j=1,12
      ires=Member(i).irest(j)
      if(ires.gt.0) then
      v(j)=past_disp_point(ires)
      else
      v(j)=0.
```

```
      endif
      enddo
c―――――――――――――――――――――――――――――――――★変位を釣合系から部材座標系に変換
      call RotateL_v(1,v,rot_memb(1,1,1,i),rot_memb(1,1,2,i),vv)
c―――――――――――――――――――――――――――――――――★要素及びモデルのセット
      mem = i
      iet  = Member(i).element_type
      iett=(iet-1)/10
      ie = Member(i).nm_element
      im = Element(ie).n_element
      imm = Member(i).n_element_type
      ien= Member(i).n_model_type
      if(Member(i).nm_dll_element .ne. 0)  goto 9998   ! DLL 要素
      if(iett.eq.0)then
      goto(11,12,13,14,15,16,17,18,19,20),iet
   11 continue
c―――――――――――――――――――――――――――――――――★Model_No.1 通常の有限要素弾塑性モデル
      if(N_analysis.eq.9) goto 9999   ! 弾塑性解析はなし
      Member(i).an_vv(1)=vv(8)-vv(2)  ! 非線形剛性計算のための変位セット
      Member(i).an_ww(1)=vv(9)-vv(3)
      call Cal_nonlin_stiff_M1(Member(i),Element(ie),
     *     Member(i).stress(7),ak )
      if(Member(i).i_rigid_length.ne.0..or.
     *   Member(i).j_rigid_length.ne.0.)
     *    call Deal_Rigid_element(ak Member(i).i_rigid_length,Member(i).j_rigid_length)
      goto 100
   12 continue
c―――――――――――――――――――――――――――――――――★Model_No.2 3 次元せん断弾塑性モデル
      if(N_analysis.eq.8) goto 9999   ! 幾何学的非線型解析はなし
      call Cal_nonlin_stiff_M2(Member(i),Element(ie), ak )
      goto 100
   13 continue
      ・
      ・
      ・
```

次に、呼び出される側のサブルーチンとして Cal_nonlin_stiff_M2() を取り上げる。呼び出す方のサブルーチンコールはすぐ上で示され、太文字で書かれている。以下に、このサブルーチンの必要部分を示す。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   SUBROUTINE /Cal_nonlin_stiff_M2
C       ―――――――――――――――――――――――――――――――――
C     ●   Model_No.2 3 次元せん断弾塑性モデル
C     ―――――――――――――――――――――――――――――――――
      subroutine Cal_nonlin_stiff_M2(Member,Element,ak)
      implicit real*8(A-H,O-Z)
      include "submain.h"
      record / member_s2    / Member
      record / element_s2   / Element
      dimension ak(12,12)
```

サブルーチンを呼ぶ側と呼ばれる側の両者の構造体を比較すると、構造体 member と Element の定義が異なっていることに気付く。ヘッダーファイル submain.h の中の２つの構造体の内容を以下に示す。

```
C     ―――――――――――――――――――――――――――――――――
C     ●   element_s 構造体
C     ―――――――――――――――――――――――――――――――――
c
c     要素
      structure / element_s/
      integer  element_type   !  要素タイプ
      integer  n_element      !  非線形要素番号
      real*8   E              !  ヤング係数
      real*8   G              !  せん断係数
      real*8   A              !  断面積
      real*8   RIx            !  ねじり剛性
      real*8   RIy            !  ｙ軸断面二次モーメント
      real*8   RIz            !  ｚ軸断面二次モーメント
      real*8   ASy            !  各部材のＹ軸回りのせん断変形用等価断面積
      real*8   ASz            !  各部材のＺ軸回りのせん断変形用等価断面積
      real*8   AM(2)          !  単位長さ当たりの質量（1:第一 2:第二ステップ用）
      integer  nm_damp        !  部材減衰の有無
      integer  nm_type        !  （maxwell モデルでは、１：x 方向２：ｙ方向３：ｚ方向）
      integer  n_section(5)   !  断面番号
      integer  nm_section(5)  !  ファイバー数
      real*8   ANP            !  軸方向耐力
      real*8   AMPY           !  ｙ軸塑性モーメント
      real*8   AMPZ           !  ｚ軸塑性モーメント
      real*8   dmm(3)         !  ダミー
      real*8   i_rigid_length      !  i 端剛域長さ
      real*8   j_rigid_length      !  j 端剛域長さ
      real*8   i_shear_G           !  i 端せん断剛性
      real*8   j_shear_G           !  j 端せん断剛性
      end structure
c     record /element_s/ Element
c     ALLOCATABLE ::Element(:)
c     ALLOCATE (Element(n_element))
c
C     ―――――――――――――――――――――――――――――――――
C     ●   member_s 構造体
C     ―――――――――――――――――――――――――――――――――
c
c     部材
      structure / member_s/
      integer  nm_element        !  要素番号（入力した要素番号を示す）
      integer  element_type      !  要素タイプ番号
      integer  n_model           !  モデルの入れ物番号
      integer  n_model_type      !  モデル別の通し番号
      integer  n_element_type    !  要素タイプ別通し番号
      integer  analysis_3D       !  解析型 0:3D 1:2D(x-z) 2:2D(y-z)
      integer  nm_so             !  部材の層番号
      integer  nm_dll_element    !  DLL を用いた要素か（０；システム内要素、１：DLL 要素）
      integer  nm_point(2)       !  節点番号
      integer  irest(12)         !  部材両端の自由度番号表
```

```
      integer  ijp(2)             !  両端節点への結合状況（0:剛結合 1:ピン結合）
      integer  nm_analysis        !  部材解析種別（-1:弾性解析、その他：通常解析）
      integer  nm_group           !  部材グループ
      integer  nm_local_coord(2)  !  局所座標系の有無とその回転行列の番号
      integer  nm_damp            !  部材減衰の有無とその減衰行列の番号
      real*8   alength            !  長さ
      real*8   i_rigid_length     !  i 端剛域長さ
      real*8   j_rigid_length     !  j 端剛域長さ
      real*8   i_shear_G          !  i 端せん断剛性
      real*8   j_shear_G          !  j 端せん断剛性
      real*8   rot_x              !  部材主軸の回転角度（度）
      real*8   force(12)          !  部材両端の部材端力（釣合座標系）
      real*8   stress(18)         !  部材両端,中央の応力（部材座標系）
      real*8   an_stress(10)      !  部材軸力（部材座標系）
      integer   d_stat(3)         !  断面の弾塑性状態　(1) i 端 (2) j 端　(3) 中央
      real*8   an_vv(10)          !  部材軸力（計算用内部変位ｖ）
      real*8   an_ww(10)          !  部材軸力（計算用内部変位ｗ）
      end structure
c     record / member_s / Member
c     ALLOCATABLE :: Member (:)
c     ALLOCATE (Member (n_member))
C     ―――――――――――――――――――――――――――――――――――
C     ●　3 次元せん断弾塑性モデル（モデル No.2）
C     ―――――――――――――――――――――――――――――――――――
c    要素数（モデル No.2　3 次元せん断弾塑性モデル：トリリニア型）
c   element_s 構造体と同一
c
      structure / element_s2/
      integer  element_type   !  要素タイプ(6)
      integer  n_element      !  非線形要素番号
      real*8   AK_1           !  第一剛性
      real*8   AK_2           !  第二剛性
      real*8   AK_3           !  第三剛性
      real*8   Q_1            !  第一折れ点のせん断力
      real*8   Q_2            !  第二折れ点のせん断力
      real*8   AKu            !  軸方向バネ
      real*8   arf            !  武田モデルα(通常は 0.4)
      real*8   U_1            !  第一折れ点の変位
      real*8   U_2            !  第一折れ点の変位
      real*8   dm4            !  ダミー
      integer  nm_damp        !  部材減衰の有無(1)
      integer  nm_type        !  履歴モデルのタイプ
      integer  n_section(5)   !  断面番号
      integer  nm_section(5)  !  ファイバー数
      real*8   ANP            !  軸方向耐力
      real*8   AQPv           !  y 方向耐力
      real*8   AQPw           !  z 方向耐力
      real*8   dmm(3)         !  ダミー
      real*8   dmm2(4)        !  ダミー
      end structure
c     record /element_s2/ Element
c
C     ―――――――――――――――――――――――――――――――――――
C     ●　3 次元せん断弾塑性モデル用（モデル No.2）member_s 構造体
C     ―――――――――――――――――――――――――――――――――――
```

```
c
c     部材
      structure / member_s2/
      integer  nm_element          !  要素番号
      integer  element_type        !  要素タイプ
      integer  n_model             !  モデルの入れ物番号
      integer  n_model_type        !  モデル別の通し番号
      integer  n_element_type      !  要素タイプ別番号
      integer  analysis_3D         !  解析型 0:3D 1:2D(x-z) 2:2D(y-z)
      integer  nm_so               !  部材の層番号
      integer  nm_dll_element      !  DLL を用いた要素か（０；システム内要素、１：DLL 要素）
      integer  nm_point(2)         !  節点番号
      integer  irest(12)           !  部材両端の自由度番号表
      integer  istat_v             !  y 方向:履歴特性の状態(*)
      integer  istat_w             !  z 方向:履歴特性の状態(*)
      integer  nm_analysis         !  部材解析種別
      integer  nm_group            !  部材グループ
      integer  nm_local_coord(2)   !  局所座標系の有無とその回転行列の番号
      integer  nm_damp             !  部材減衰の有無とその減衰行列の番号
      real*8   alength             !  長さ
      real*8   i_rigid_length       !  i 端剛域長さ
      real*8   j_rigid_length       !  j 端剛域長さ
      real*8   i_shear_G            !  i 端せん断剛性
      real*8   j_shear_G            !  j 端せん断剛性
      real*8   rot_x                !  部材主軸の回転角度（度）
      real*8   force(12)            !  部材両端の部材端力（釣合座標系）
      real*8   stress(6)            !  部材中央の応力（部材座標系）
      real*8   AKv_tan              !  接線剛性(*)
      real*8   AKw_tan              !  接線剛性(*)
      real*8   dmv(10)              !  y 方向耐力:履歴特性で使用(*)
      real*8   dmw(10)              !  z 方向耐力:履歴特性で使用(*)
      integer  d_stat(3)            !  断面の弾塑性状態　(1) i 端 (2) j 端　(3) 中央
      real*8   an_vv(10)            !  部材軸力（計算用内部変位ｖ）
      real*8   an_ww(10)            !  部材軸力（計算用内部変位w）
      end structure
```

両者の構造体を比較すると、対応する位置の成分名が異なることが理解できる。前者の構造体 element_s や member_s は一般的な部材に対する標準的構造体として定義されたものであり、後者の構造体である element_s2 と member_s2 は、3 次元せん断型弾塑性モデル用に定義したものである。このような使い方は、一般的でない部材モデルに対し、そのモデルに合わせて構造体の内容を定義したい場合に多く用いられる。

配列型でしかも動的領域として定義された構造体を、その内容を異なって使用したい場合、上記のような使い方をすることになる。無論、記憶領域に配置された構造体配列は、element_s や member_s の内容で繰り返されている。しかし、その内容は、種々の部材モデルに合わせて異なった使い方がされていることを理解されたい。

2.4 FORTRAN と C++

動的解析システム SPACE では、開発言語として FORTRAN と C++を併用している。両言語間のインターフェイスやデータの交換について、簡単に復習しておこう。各々の言語仕様については、各自勉強されたい。

両言語間のインターフェイスは、特別なインターフェイス文が必要となる。SPACE でよく使用する C++で書かれたコードから FORTRAN で書かれたサブルーチンの呼び出し方法を示す。SPACE では、C++のインクルードファイルとして fort.h に全て書き込むようになっている。以下に、fort.h を示す。この中で代表的な関数の役割については、後章で述べることになる。

```
//     ――――――――――――――――――――――――――――――――
//     ●   include /fort.h
//     ――――――――――――――――――――――――――――――――
//     ●    FORTRAN 用の C++の外部宣言(ok)
//     ――――――――――――――――――――――――――――――――
extern "C"{
void __stdcall SYSNAM_C(int*,char*,int*);
void __stdcall SUBMAIN_DYNAMIC_A(int*,int*,int*,int*,double*,double*,int*,int*,double*,int*,
            int*,float*,int*,float*,int*,float*,float*,float*,float*,int*,int*,int*,int*);
void __stdcall SUBMAIN_DYNAMIC_B(int*,int*,int*,int*);
void __stdcall INPTFX(int*,int*,int*,int*,int*,int*);
void __stdcall INPTFY(int*,int*,int*,int*,int*,float*,
                      int*,int*,int*,int*,int*,int*,float*,float*,
                      float*,int*,int*,float*,int*,float*,int*,int*,float*,int*,int*,int*);
void __stdcall PERSCT(int*,int*,int*,float*,float*,float*,float*,float*,float*);
void __stdcall DYCTL1_C(int*,int*,float*,float*);
void __stdcall DYCTL2_C(int*,int*,float*,int*,int*,float*,
                        float*,float*,int*,float*);
void __stdcall DOUTCL_C(int*,int*,float*,float*,int*);
void __stdcall ROTSET(float*,float*,float*,float*,float*);
void __stdcall ROTSTX(float*,float*,float*);
void __stdcall TOSHIZ(float*,int*,float*,float*,int*,float*,float*,int*,int*,int*,float*);
void __stdcall RRSETE(int*, float*,float*,float*);
void __stdcall RRSET(int*, float*,float*,float*);
void __stdcall TOSHIP(int*,float*,int*,float*,int*,float*,float*,int*,
                      float*,float*,int*,int*,int*,float*,float*,int*);
void __stdcall TOSHIM(float*,float*,int*,float*,float*,int*,
                            float*,float*,int*,int*,int*,float*,float*,int*);
void __stdcall STCLSP(int*,int*,int*,float*,int*,int*,int*,float*);
void __stdcall SETSP1(int*,int*,float*,float*,int*,int*,
                      int*,int*,int*,float*,float*,float*,float*,float*,float*,float*,float*);
void __stdcall SETSP2(int*,int*,float*,float*,float*,int*,int*,
                      int*,int*,int*,float*,float*,float*,float*,float*,float*,float*,float*);
void __stdcall SETUNF(float*,int*,float*,float*,int*,int*,int*);
void __stdcall SETSX2(int*,int*,float*,float*,float*,float*,float*,int*,int*,int*,int*,int*,
                            int*,float*,float*,float*,float*,float*,float*);
void __stdcall SETSX1(int*,int*,float*,float*,float*,float*,int*,int*,int*,int*,int*,
                            int*,float*,float*,float*,float*,float*,float*,float*);
void __stdcall TOSHMC(float*,int*,float*,float*,int*,float*,float*,float*);
void __stdcall SETSP4(int*,int*,float*,float*,int*,float*,float*);
```

```
void __stdcall TOSHLD(float*,float*,float*,int*,float*,float*,float*);
void __stdcall SETSP3(int*,int*,int*,float*,float*,float*,int*,int*,int*,int*,float*,float*,float*,
                      int*,float*,float*,float*,float*,float*,int*,int*,float*,int*);
void __stdcall SETSP5(int*,int*,int*,float*,float*,float*,int*,int*,int*,int*,float*,float*,float*,
                    int*,float*,float*,float*,float*,float*,int*,int*,float*,int*,float*,int*,int*);
void __stdcall SETSP5X(int*,int*,int*,float*,float*,float*,int*,int*,int*,int*,float*,float*,float*,
                    int*,float*,float*,float*,float*,float*,int*,int*,float*,int*,float*,int*,int*);
void __stdcall TOSHMM(float*,float*,int*,float*,float*,int*,
                              float*,float*,int*,int*,int*,float*,float*);
void __stdcall SETSPG(int*,int*,float*,float*,float*,
                         float*,float*,float*,float*,float*);
void __stdcall AVERAG(int*,float*,int*,float*);
void __stdcall INUG(int*,int*,float*,int*,int*,float*,int*);
void __stdcall WAVEDIS(float*,float*,int*,int*,int*,int*,float*);
void __stdcall WAVESPG(float*,int*,int*,int*,int*,int*,
                     float*,float*,float*,float*,float*,float*,float*,float*,int*,float*);
void __stdcall WAVERES(float*,int*,int*,int*,int*,int*,
                       float*,float*,float*,float*,float*,float*,float*,float*,int*,
                                 int*,float*,int*,float*,float*,int*);
void __stdcall DYCTLX(char*,char*,char*,char*,char*,char*,char*,int*,float*);
void __stdcall DYCTLXP(char*,char*,int*);
void __stdcall SAIDAI(float*,float*,float*);
void __stdcall INUGG(int*,int*,float*,int*,int*,char*);
void __stdcall SETMAX_V(float*,float*,float*,int*);
void __stdcall INPMAS(int*,int*,int*);
}
```

インクルード用ヘッダーファイル fort.h で、インターフェイスは、外部関数であることを示す extern “C”{ }で囲まれており、その中に関数名を書くことになる。標準的な書き方は、以下のようである。

```
void __stdcall SYSNAM_C(int*,char*,int*);
```

関数名、つまり、FORTRAN のサブルーチン名あるいは関数名は大文字でなくてはならない。これは、言語仕様からくるもので、たとえプログラムの表記に小文字を含んでいても変数、関数、サブルーチン名等は、FORTRAN では全て大文字とみなすからである。従って、大文字と小文字を区別する C++では、FORTRAN のサブルーチンを呼ぶ場合は、大文字でなくてはならない。

例として動的ソルバーの主サブルーチンとなる SUBMAIN_DYNAMIC_A() を取り上げ、FORTRAN と C++のコードを示す。まず、インターフェイス文は、

```
void __stdcall SUBMAIN_DYNAMIC_A(int*,int*,int*,int*,double*,double*,int*,int*,double*,int*,
             int*,float*,int*,float*,int*,float*,float*,float*,float*,int*,int*,int*,int*);
```

であり、この記述は fort.h に含まれている。次に、サブルーチンを呼

ぶ方の C++コードと、呼ばれる方の FORTRAN コードを以下に示す。

C++で書かれたコード

```
//    ――――――――――――――――――――――――――――――――
//      ●　時刻歴解析開始
//    ――――――――――――――――――――――――――――――――
//    ――――――――――――――――――――
//      ●　予備計算：シングルスレッド
//    ――――――――――――――――――――
      SUBMAIN_DYNAMIC_A(&F_calnum,&iend_code,&icontrol,&ierr_dat,&T_analysis,&dt_analysis,
                        &nx_step,&ns_step,&d_max_v,&id_max_v,
                        &F_read_disp,F_disp,&F_read_ndbalanceF,F_ndbalanceF,
                         &F_read_spring,F_fay,F_n_spring,F_my_spring,
                        F_mz_spring,F_stat_spring,&n_iterate,
                        &nm_iterate,&numb_method);
```

FORTRAN で書かれたサブルーチン

```
c――――――――――――――――――――――――――――――――――――――――
      subroutine submain_dynamic_a(i_calnum,iend_code,icontrol,ierr_dat,
     *           T,dt,n_step,ns_step,d_max_v,id_max_v,
     *           i_read_disp,F_disp,i_read_ndbalanceF,F_ndbalanceF,
     *           i_read_spring,F_fay,F_n_spring,F_my_spring,
     *           F_mz_spring,i_stat_spring,n_iterate,
     *           nm_iterate,numb_method)
```

上記のサブルーチンの括弧内は引数を表しており、当然、呼ぶ方と呼ばれる方、共に引数の型と数が一致しなければならない。また、C++側における引数の前の記号 & は、アドレス渡しを意味しており、C++と FORTRAN 間では、全てこのアドレス渡しとなっている。特に、C++のコードは、注意が必要であり、変数のアドレス渡しは、変数名の前に、& 記号が必要となる。ただし、配列は、C++でもアドレス渡しであるため、& 記号を付けてはならない。例えば、F_disp、F_ndbalanceF などがそれに該当する。詳細は文法書[2)]を見られたい。

関数間のデータ渡しは、アドレス渡しとデータ渡しがある。一般に FORTRAN はアドレス渡しであり、また C++では、単独の変数はデータ渡しで、配列はアドレス渡しとなっている。アドレス渡しは、受け渡す関数に引数となっている記憶番地を渡すため、受け渡される関数内でデータの変更を行うと、受け渡す方の引数である変数の値も変更されてしまう。一方、データ渡しは、受け渡す方の引数に入っている値をコピーして、受け渡される関数の中の対応する変数にこの値をセットする。そのため、呼ばれる方でデータを変更したとしても、呼び出し側では、値は変更されていないことになる。このように、両者は記述の仕方も動作も異なっているので注意して使用する必要がある。

次に、両言語間で配列の記述を考えてみよう。両者では、配列の記述

方法と記憶番地の順序が異なっている。例を用いて説明しよう。まず、２次元配列を以下に書いてみる。

```
FORTRAN：  A(10,10)
C++    ：  A[10][10]
```

何も指定しないと、配列の記憶順は、FORTRAN では A(1,1)から始まって、A(2,1)、A(3,1)となり、最後が、A(10,9)、A(10,10)の順番となる。一方、C++では、A[0][0]、A[0][1]から始まって、最後が A[9][8]、A[9][9]となる。このように、文法の相違として、一般的には FORTRAN は１から始まるが、C++は 0 から始まり、また、FORTRAN は最も左から記憶番地が始まるが、C++は外側から順次内側に変化する。例えば、A(5,6)は A[5][4]と同じ番地を表すことになる。両言語間でデータ交換を行う場合、これらのことを良く考慮して、データの受け渡しを行う必要がある。

以上で、FORTRAN と C++を併用する場合の最も基本的な注意点を述べた。後は、Visual C++と Visual FORTRAN の文法書や(株)マイクロソフトの Developer Studio に関する文書を参照されたい。

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。

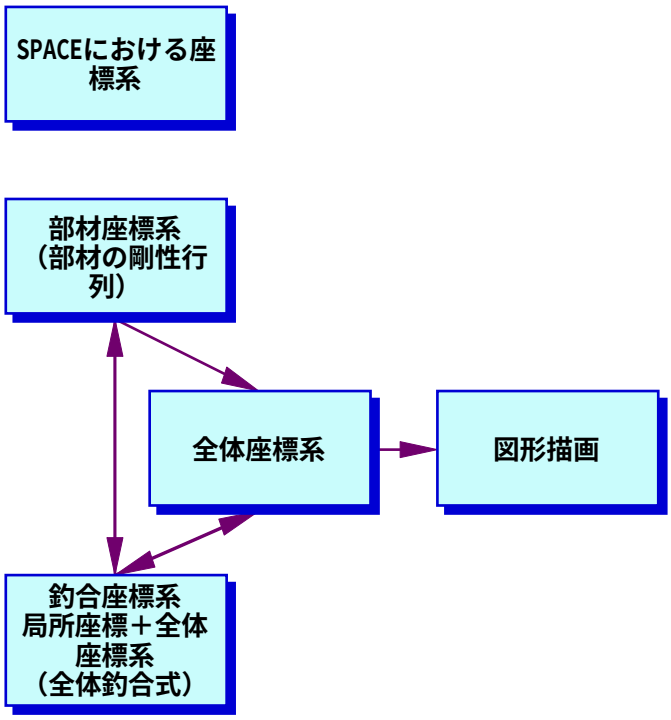


図 2-1 SPACE における座標系

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。

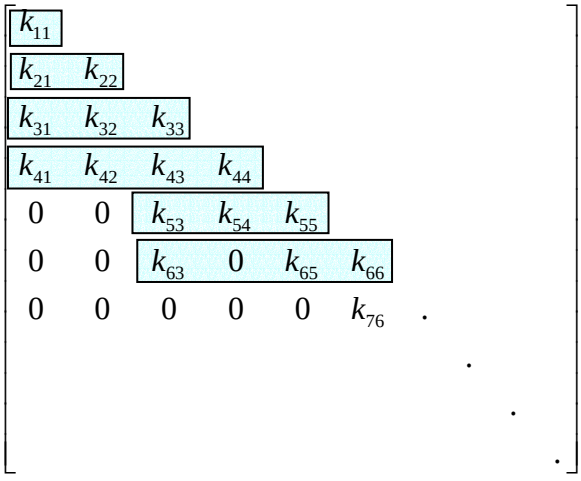


図 2-2 対称行列と水平型スカイライン行列

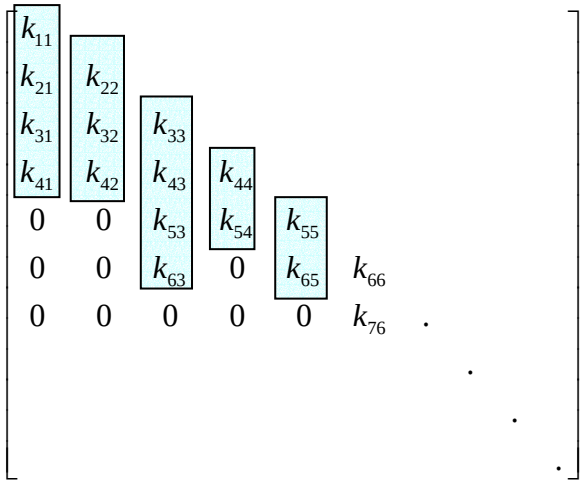


図 2-3 対称行列と垂直型スカイライン行列

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。

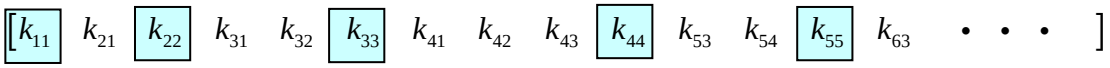
スカイライン行列は、１次元配列でデータを保存する。そのため、剛性などの行列をこのスカイライン行列に変換するテーブルが必要となる。このテーブルは、行列の各行の左より最初のゼロでない値を持つ要素から、対角要素までの数を並べたものである。あるいは、行列の各列の上から、最初のゼロでない値を持つ要素から、対角要素までの数を並べたものである。具体的に、図 2-2 の水平型の行列について、テーブルを作成してみよう。まず、上で示したテーブルを下のように作成する。ただし、最初の 0 はダミーであり、計算の都合上入れる。

[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。