第８章　動的解析システム

8.1 はじめに

本章では、動的解析システムにおける動的ソルバーの役割と動的解析システム内のグラフィック処理と動的ソルバーを管理するシステムについて解説する。ただし、紙面の都合により動的解析システムで使用している全ての技術・テクニックを紹介することはできない。特にグラフィックやウインドウ管理の詳細な説明は、紙面の都合で同様な処理を行っているプレゼンターのマニュアルに譲ることにする。ここでは、動的解析システムに関連し、中心的な役割を果たしている技術・テクニックについて解説する。

SPACE 内の動的解析システムは、解析結果をリアルタイムにアニメーションとして描画しつつ、また、計算を途中で停止、再開、中止などを行うことができる。このような処理を可能とするためには、数値計算以外に多くの処理が必要となる。ここでは、これらの処理が可能となるような仕組みとそのプログラムコードを解説する。この章では、visual C++言語を多少理解していることを前提に書かれている。特に、グラフィックシステムに興味をお持ちの読者は、visual C++言語のリファレンスマニュアルなどを傍に置き、新しい関数が出てくる毎に、その関数の意味を調べると良い。標準的なテクニックが直ぐに理解できるようになろう。

オブジェクト指向型言語である C++によるプログラムは、数値計算に用いるコードと大きく異なる部分があり、FORTRAN で育った人達には理解しにくいかもしれない。その原因は、プログラムがメッセージ駆動型であることとオブジェクト、クラス、マルチスレッドなどの新しい概念が出現することにある。一般的に、プログラム中の関数は、メッセージによって駆動され、関数が実行される。このように逐次型のプログラムと全く異なった動作で処理されるため、プログラムを読むことも書くことも、最初は途惑うことになろう。このメッセージ駆動型の構造を一旦理解すれば、動的ソルバー管理システムを理解することはそれほど難しくはない。

オブジェクト指向の本質を理解することは容易ではないし、C++の入門書を丁寧に読んでも、その本質を理解するまでには時間がかかる。ここでは、記述は多少あいまいでも、読者側に立った解説ができるよう心がけるつもりである。

SPACE における動的ソルバーはマルチスレッドの技術を利用して作成されている。この技術抜きでは、リアルタイムで結果をアニメーション

にしたり、ユーザーからのメッセージを受け取ったりすることは難しい。なぜなら、数値計算部分が cpu を占有するため、ユーザーからのメッセージを受け取ることができなくなり、画面がフリーズ状態となってしまうからである。ここでは、マルチスレッドに関する参考書を読んで、その概念と入門程度の技術を理解しておいてほしい。動的ソルバーに利用されているマルチスレッドの技術は決して程度の高いものではない。良く読んでいただければ、容易に、その構成と本質を理解できる。

8.2 SPACE の中の動的解析システム

SPACE システムは多くのサブシステムと多数のファイル群とで構成され、それらが有機的に結合することよって効果的な働きをする。ここでは、SPACE における動的解析システムの役割と、動的解析システムの起動時と終了時の処理内容について説明する。

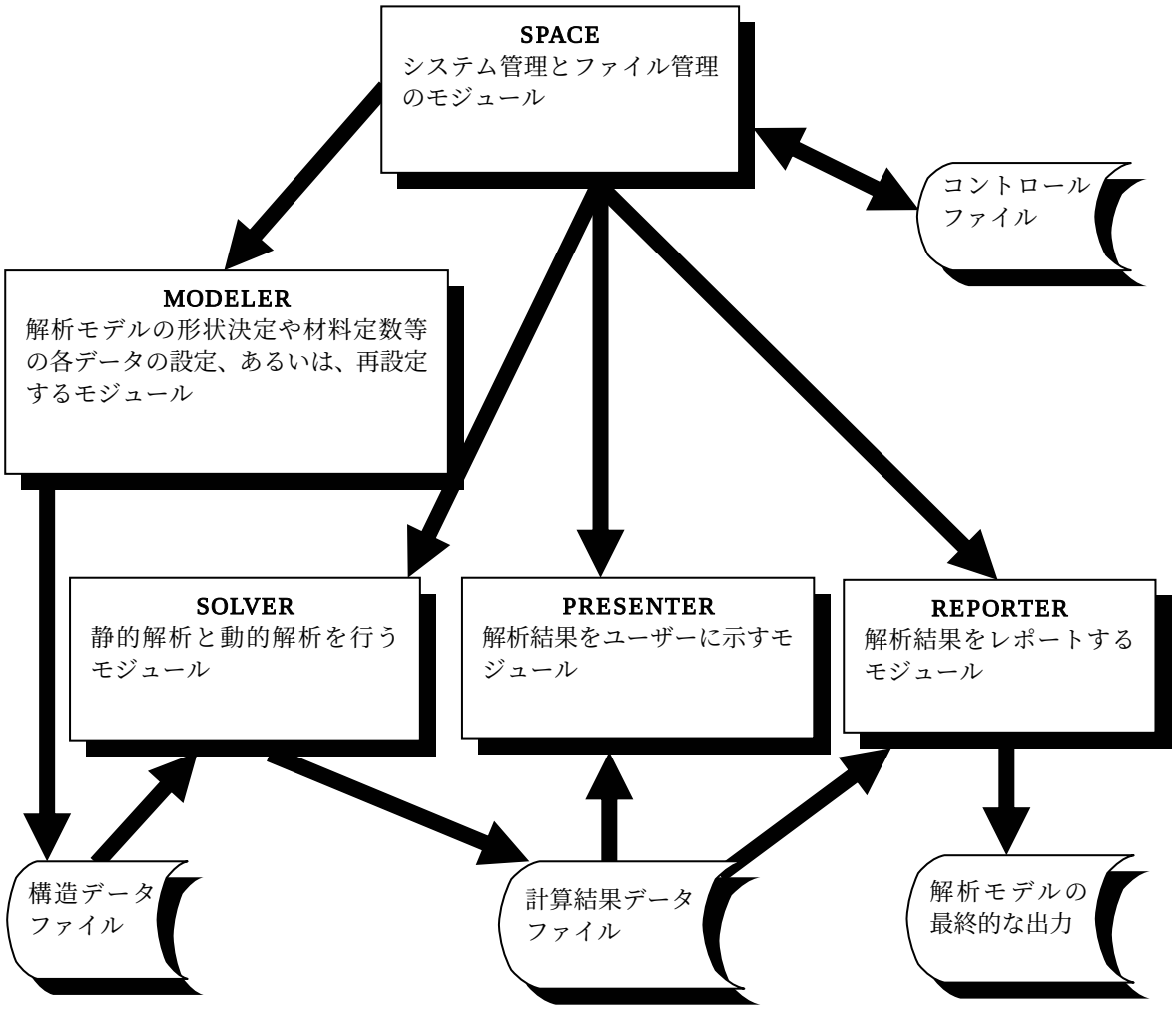

図 8-1　SPACE システムの構成

SPACE システムは図 8-1 に示すように多くのサブシステムによって構成されている。同図を参照しながら、最初に、動的解析システムが起動されるまでの働きを見てみよう。

SPACE を使用する場合、まず、SPACE を利用してコントロールファイルを作成し、他のファイル群の名前を設定する。次にモデラーを用いて、構造データや他のデータ群を作成し、該当する各々のファイルに出力する。最後に、動的解析に必要となる制御用パラメータを、SPACE を用いて設定し、ファイルに出力する。これで準備が終わり、後は動的解析システムを起動して、解析を実行することになる。

動的解析システムの起動は、SPACE によって行われ、情報の伝達は TEMP 領域に書き出される spacesys.xxx ファイルで行われる。このファイルの仕様は以下のようであり、SPACE から動的解析システムに解析種別とコントロールファイル名を受け渡す。ファイルの内容は 2 行で構成されており、一行目は解析種別を表し、2 行目はコントロールファイルの絶対パス名を表す。

```
10
D:¥space-frame¥DISK2¥動的解析編¥アーチの動的安定¥両端ピン支持アーチ¥Arch_弾性分岐動座屈.ctl
```

動的解析システムに起動がかかると、まず、一般的な手続きを行った後、Mainfrm.cpp 内の Mainframe クラスでシステム内の初期設定を行う。ここでの処理の重要な部分を以下に示し、その処理内容の説明を行う。プログラム言語は、C++で、オブジェクト指向型である。このプログラムは、Microsoft 社の Developer Studio を用いて、基本部分を作成したものである。

動的解析システムが起動して、メインウインドウが描画される前に、メインウインドウを管理する mainframe クラスが構築される。クラスが構築されるときと消滅するとき、必ず次に示す関数が呼ばれる。これをそのクラスのコンストラクタ及びデストラクタと呼ぶ。このクラスのコンストラクタ CMainFrame::CMainFrame()関数では、初期設定が行われる。その内容については、後で、プログラムを参照しながら説明する。メインウインドウが消滅するとき、つまり、解析が終了するとき、デストラクタ CMainFrame::~CmainFrame()が呼ばれる。この関数では、主にコンストラクタで動的領域確保した変数領域を解放している。

少し、MainFrame クラスの内容を見てみよう。関数定義の前に見られる BEGIN_MESSAGE_MAP と END_MESSAGE_MAP のキィワードが目を引く。ま

ず、このキィワードについて説明しよう。この言葉に挟まれて記述されている関数に特別な意味がある。例えば、この２つのキィワードの間にある ON_WM_CREATE()と ON_WM_CLOSE()の存在は、他からのメッセージによって該当する関数が呼び出されることを意味する。つまり、ウインドウが構築されるとき、関数 CMainFrame::OnCreate()が起動され、同じくウインドウが閉じられるとき、関数 CMainFrame::OnClose()が実行される。この２つのキィワードの中に含まれる関数は、他のクラスやユーザーのマウス操作からのメッセージによって駆動されるわけである。このような記述は他のクラスにも見られるので、出現するときは注意して見られたい。

それでは、次に mainframe クラスの記述を検討しよう。

```
#include "stdafx.h"
#include "sf3st.h"
#include "sf3stdatex.h"
#include "MainFrm.h"
#include "fort.h"
//    ――――――――――――――――――――――――――――――――――――――――
//      ●    CMainFrame
//    ――――――――――――――――――――――――――――――――――――――――
IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
          //{{AFX_MSG_MAP(CMainFrame)
          ON_WM_CREATE()
          ON_WM_CLOSE()
          //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// CMainFrame クラスの構築/消滅
CMainFrame::CMainFrame()
{
         View_C_panel = 0;  // コントロールパネルの描画初期設定
         n_step = 10;       // 描画間隔初期セット
//    ――――――――――――――――――――――――――――――――――――――――
//      ●    制御用データの初期設定
//    ――――――――――――――――――――――――――――――――――――――――
          int ihan,ndim,nnode;
          ihan=0;
          ndim=3;
          nnode=1;
          int xxfig[3];
          F_time_ii =0;
          F_Speed = 4;
          F_read_disp=0;                    //変位配列 F_disp を動的領域確保を行ったか否かチェック
          F_read_ndbalanceF = 0;            //
          F_read_spring = 0;                //
          F_time_ii =0;
          F_Time=0;
          F_holt=0;
```

```
//  ------------------------------------------------------------
//    ●   描画用ペンの設定
//  ------------------------------------------------------------
      create_penx();
//  ------------------------------------------------------------
//    ●   システムデータの入力
//  ------------------------------------------------------------
      SYSNAM_C(&F_idfile[0],F_title,&F_calnum);
//  ------------------------------------------------------------
//    ●   透視図用データの入力
//  ------------------------------------------------------------
      PERSCT(&xxfig[0],&xxfig[1],&xxfig[2],&F_scrnps[0],
                  &F_viewps[0],&F_scalps,&F_scalep[0],&F_mag[0]);
      FF_scalps = F_scalps;
//  ------------------------------------------------------------
//    ●   動的解析用制御データその１入力
//  ------------------------------------------------------------
      DYCTL1_C(&ihan,&F_nindis,&F_gindis,&F_f1sec);
//  ------------------------------------------------------------
//    ●   動的解析用制御データその２入力
//  ------------------------------------------------------------
      DYCTL2_C(&ihan,&F_nstep,&F_delt,&F_igra[0],&F_ibata,
                    &F_beta,&F_gamma,&F_xgal[0],&F_ntime,&F_dlamst);
//  ------------------------------------------------------------
//    ●   動的解析用出力制御データ入力
//  ------------------------------------------------------------
      DOUTCL_C(&ihan,&F_iwstp,&F_soutsc,&F_dmaxck,&n_step);
//  ------------------------------------------------------------
//    ●   描画ステップのセット
//  ------------------------------------------------------------
      if( n_step == 0)  n_step = 10;  //      描画間隔の規定値セット
           F_mstep = F_f1sec/(F_delt*(float)F_iwstp);
           F_nstep=F_nstep/(float)F_iwstp+1;
           F_all_step = F_mstep+F_nstep;
           F_delt_cl = F_delt*(float)F_iwstp;
           F_all_time = (F_all_step +1)*F_delt_cl;
//  ------------------------------------------------------------
//    ●   構造用データの予備入力（ただし、画面描画用）
//  ------------------------------------------------------------
           int m_axis;
           INPTFX(&node,&nelem,&mnbsb,&nrzero,&locod,&m_axis);
//  ------------------------------------------------------------
//    ●   描画用データ領域の動的確保
//  ------------------------------------------------------------
           ihan=1;
           nnode=node;
      posit = new float[nnode*ndim];
           if(posit == 0){
             MessageBox("Sorry! Memory allocation failure. You should stop this application.");
                 }
      iconsb = new int[2*mnbsb];
           if(iconsb == 0){
             MessageBox("Sorry! Memory allocation failure. You should stop this application.");
                 }
```

```
F_al = new float[2*mnbsb];
     if(F_al == 0){
        MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
F_alqq = new int[nelem];
     if(F_alqq == 0){
        MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
imeme =  new int[mnbsb];
     if(imeme == 0){
        MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
imeme_line =  new int[mnbsb];
     if(imeme_line == 0){
        MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
  F_bz =   new float[nnode*2];
      if(F_bz == 0){
        MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
 F_bound =   new int[nnode];
      if( F_bound == 0){
          MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
 F_free =   new int[6*nnode];
      if( F_free == 0){
          MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
 mtype =   new int[nelem];
      if(mtype == 0){
          MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
 F_mytype =   new int[nelem];
      if(F_mytype == 0){
          MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
 Fnm_type =   new int[nelem];
      if(Fnm_type == 0){
          MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
 F_snp =   new float[5*nelem];
      if(F_snp == 0){
          MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
 F_smyp =   new float[5*nelem];
      if(F_smyp == 0){
          MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
 F_smzp =   new float[5*nelem];
      if(F_smzp == 0){
          MessageBox("Sorry! Memory allocation failure. You should stop this application.");
           }
 Rrot =   new float[9*mnbsb];
      if(Rrot == 0){
```

```
                MessageBox("Sorry! Memory allocation failure. You should stop this application.");
                 }
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●　描画用の構造データの入力
//    ―――――――――――――――――――――――――――――――――――――――――――――
        int nelemx = nelem;
        INPTFY(&node,&nelemx,&mnbsb,&nrzero,&locod,posit,&ndim,
          iconsb,imeme,imeme_line,&nm_line,mtype,F_snp,F_smyp,F_smzp,
          F_bound,&xxfig[0],Rrot,&F_nindis,&F_gindis,F_free,F_mytype,
          F_al,F_alqq ,&m_axis,Fnm_type);
        F_mtype[0]=2;
        F_mtype[1]=4;
        F_mtype[2]=3;
        F_mtype[3]=5;
        numb_method = 0;   // 解析手法：反復法
        nm_iterate=0;
}
CMainFrame::~CMainFrame()
{
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●　動的領域の解放
//    ―――――――――――――――――――――――――――――――――――――――――――――
 delete [] posit;
 delete [] iconsb;
 delete [] imeme;
 delete [] imeme_line;
 delete [] F_bz;
 delete [] mtype;
 delete [] F_mytype;
 delete [] F_snp;
 delete [] F_smyp;
 delete [] F_smzp;
 delete [] Rrot;
 delete [] F_bound;
 delete [] F_free;
 delete [] F_al;
 delete [] F_alqq;
 delete [] Fnm_type;
 if(F_read_disp == 1){
  delete [] F_disp;
 }
 if(F_read_spring == 1){
            delete [] F_fay;
            delete [] F_n_spring;
            delete [] F_my_spring;
            delete [] F_mz_spring;
            delete [] F_stat_spring;
 }
 if(F_read_ndbalanceF == 1){
            delete [] F_ndbalanceF;
}
 if(F_read_mass == 1){
            delete [] F_mass;
}
```

```
}
void CMainFrame::create_penx()
{
        int i1;
//  ―――――――――――――――――――――――――――――――――――――――――
//    ●　描画用ペンの設定（太さと色）
//  ―――――――――――――――――――――――――――――――――――――――――
 Brash_color = RGB(0,80,130);
 for(int i=0;i<50;i++){
  i1=5*(50-i);
  NewPenx_a[i].CreatePen(PS_SOLID,1,RGB(255,255-i1,255-i1));
  NewPenx_a[100-i].CreatePen(PS_SOLID,1,RGB(255-i1,255-i1,255));
 }
  NewPenx_a[50].CreatePen(PS_SOLID,1,RGB(255,255,255));
  PrPenx_a[50].CreatePen(PS_SOLID,F_edit_line_width_m,RGB(255,255,255));
  NewPenx_a[101].CreatePen(PS_SOLID,1,RGB(255,255,255));
  NewPenx_a[102].CreatePen(PS_SOLID,2,RGB(0,100,255));
  NewPenx_a[103].CreatePen(PS_SOLID,2,RGB(255,255,0));
  NewPenx_a[104].CreatePen(PS_SOLID,2,RGB(255,255,255));
  NewPenx_a[105].CreatePen(PS_SOLID,2,RGB(255,0,0));
  NewPenx_a[106].CreatePen(PS_SOLID,1,RGB(255,0,0));
  NewPenx_a[107].CreatePen(PS_SOLID,1,RGB(255,255,0));
  NewPenx_a[108].CreatePen(PS_SOLID,1,RGB(0,0,200));
  NewPenx_a[109].CreatePen(PS_SOLID,1,RGB(0,0,0));
  NewPenx_a[110].CreatePen(PS_SOLID,1,RGB(0,0,220));
  NewPenx_a[111].CreatePen(PS_SOLID,3,RGB(255,0,0));
  NewPenx_a[112].CreatePen(PS_SOLID,1,RGB(0,0,255));
  NewPenx_a[113].CreatePen(PS_SOLID,1,RGB(0,255,0));
  NewPenx_a[116].CreatePen(PS_SOLID,2,RGB(0,0,0));
  NewPenx_a[117].CreatePen(PS_SOLID,2,RGB(0,255,150));
  NewPenx_a[118].CreatePen(PS_SOLID,2,RGB(255,255,0));
  NewPenx_a[119].CreatePen(PS_SOLID,2,RGB(255,255,255));
  NewPenx_a[120].CreatePen(PS_SOLID,1,RGB(0,0,0));
  NewPenx_a[121].CreatePen(PS_SOLID,2,RGB(0,0,0));
  NewPenx_a[122].CreatePen(PS_SOLID,3,RGB(0,0,0));
  NewPenx_a[123].CreatePen(PS_SOLID,4,RGB(0,0,0));
  NewPenx_a[124].CreatePen(PS_SOLID,5,RGB(0,0,0));
  NewPenx_a[125].CreatePen(PS_SOLID,6,RGB(0,0,0));
  NewPenx_a[130].CreatePen(PS_SOLID,1,RGB(0,100,100));

  NewPen_pr[0].CreatePen(PS_SOLID,1,RGB(0,0,0));
  NewPen_pr[1].CreatePen(PS_SOLID,2,RGB(0,0,0));
  NewPen_pr[2].CreatePen(PS_SOLID,3,RGB(0,0,0));
  NewPen_pr[3].CreatePen(PS_SOLID,4,RGB(0,0,0));
  NewPen_pr[4].CreatePen(PS_DASH,1,RGB(0,0,0));
  NewPen_pr[5].CreatePen(PS_DASHDOT,1,RGB(0,0,0));
  NewPen_pr[6].CreatePen(PS_DASHDOT,1,RGB(0,0,0));
  NewPen_pr[7].CreatePen(PS_SOLID,1,RGB(0,0,0));
  NewPen_pr[8].CreatePen(PS_SOLID,1,RGB(0,0,0));
  NewPen_pr[9].CreatePen(PS_SOLID,1,RGB(0,0,0));

//  ―――――――――――――――――――――――――――――――――――――――――
//    ●　描画用フォントの設定
//  ―――――――――――――――――――――――――――――――――――――――――
```

```
        float xx= 4.;
        F_ipx=15*xx;
              F_ipy=15*xx;
              int point =12*xx;
              CClientDC dc(AfxGetMainWnd());
        prFont.CreateFont(point
                        ,0,0,0,FW_NORMAL,FALSE,FALSE,0,
                        SHIFTJIS_CHARSET,
                        OUT_DEFAULT_PRECIS,
                        CLIP_DEFAULT_PRECIS,
                        DEFAULT_QUALITY,
                        DEFAULT_PITCH | FF_MODERN,
                        "ＭＳ　Ｐゴシック");
             point =10*xx;
             prFontx.CreateFont(point,0,0,0,FW_NORMAL,FALSE,FALSE,0,
                        SHIFTJIS_CHARSET,
                        OUT_DEFAULT_PRECIS,
                        CLIP_DEFAULT_PRECIS,
                        DEFAULT_QUALITY,
                        DEFAULT_PITCH | FF_MODERN,
                        "ＭＳ　Ｐゴシック");
}
void CMainFrame::OnClose()
{
 if(thread_on != 0) {
 MessageBox("現在、計算実行中もしくは停止中です。計算を中止した後ウィンドウを閉じてください。");
 return;
 }
 CMDIFrameWnd::OnClose();
}
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
          if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
                    return -1;
          if (!m_wndToolBar.Create(this) ||
                    !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
          {
                    TRACE0("Failed to create toolbar¥n");
                    return -1;      // 作成に失敗
          }
          if (!m_wndStatusBar.Create(this) ||
                    !m_wndStatusBar.SetIndicators(indicators,
                      sizeof(indicators)/sizeof(UINT)))
          {
                    TRACE0("Failed to create status bar¥n");
                    return -1;      // 作成に失敗
          }
          m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
                    CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);
          m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
          EnableDocking(CBRS_ALIGN_ANY);
          DockControlBar(&m_wndToolBar);
          return 0;
}
```

ファイル mainframe.cpp の最初で、多くのヘッダーファイルをインクルードしている。特に、fort.h は FORTRAN プログラムとのインターフェイス文が保存されており、また、sf3stdatex.h は、動的解析システムの制御を行うための重要なグローバル変数やグローバル配列を定義している。これらのグローバル変数やグローバル配列の定義は、extern 宣言を行っており、実際の宣言は sf3stdat.h に記述されている。これは CSf3stView.cpp ファイルにインクルードされており、ここでグローバル変数などが割り付けられることになる。また、これらのグローバル変数やグローバル配列は数値計算には直接関係せず、図形表示や動的ソルバーの制御に用いられる。各変数や配列の意味は定義文の後にコメントされているので参照されたい。また、これらの変数等は計算に関係しないため、実数は全て単精度で定義している。以下に、sf3stdatex.h を示す。

```
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　グローバル変数の定義
//    ―――――――――――――――――――――――――――――――――――――――――
   extern  int    View_C_panel;    // 制御パネル表示コード　０：非表示　１：表示
   extern  HWND    F_hwnd;         // ウインドウ用ハンドル
   extern  double T_analysis;      // 解析全体の時間（sec）
   extern  double dt_analysis;     // 解析増分時間（sec）
   extern  int    thread_on;       // 計算実行中かどうかを示すパラメータ ０：実行中でない １：実行中
   extern  int    icontrol;        // 計算コード      0:予備計算  1:動的解析 99:終了処理
   extern  int    iend_code;       // 計算終了コード 0=継続  1:終了
   extern  int    ierr_dat;        // エラーコード
   extern  int    n_step,ns_step;  // 今回の解析ステップ数及び解析開始ステップ
   extern  int    id_max_v;        // 時刻ｔにおける最大変位を示す節点番号
   extern  double d_max_v;         // 時刻ｔにおける最大変位
   extern  int    F_calnum;        // 解析種別
   extern  int    F_holt;          // 計算実行が停止中かどうかを示すパラメータ　０：停止中でない
   extern  char   F_title[60];     // コントロールファイルに書かれたタイトル
   extern  int    F_idfile[100];   // 書き込み可能チェック用  1:可能  0:不可
   extern  char   Title[100];      // 解析タイトル
   extern  float* posit;           // スペースフレーム描画用節点座標
   extern  int*   iconsb;          // 部材両端の節点番号
   extern  int*   imeme;           // 部材に付されたが要素番号
   extern  int*   imeme_line;      // 部材グループ番号
   extern  int    nm_line;         // 部材グループ番号の最大値
   extern  int*   mtype;           // 部材内で弾塑性状態を表示する個数
   extern  float* F_bz;            // 透視図形描画用座標
   extern  CPen   NewPenx_a[131]; // 画面描画用ペンの種類
   extern  CPen   PrPenx_a[131];  // プリント用ペンの種類
   extern  CPen   NewPen_pr[10];  // プリント用ペンの種類その２
   extern  int*   F_mytype;        // 部材モデル番号
   extern  int*   Fnm_type;        // 履歴モデルのタイプ
   extern  int*   F_free;          // 各節点の自由度
   extern  float* F_snp;           // 降伏軸力
   extern  float* F_smyp;          // ｙ軸塑性モーメント
   extern  float* F_smzp;          // ｚ軸塑性モーメント１
   extern  int*   F_bound;         // 境界出力用データ
```

```
   extern  float* F_load_d1;       // ファイル１の静的荷重
   extern  float* F_load_d2;       // ファイル２の静的荷重
   extern  float* F_load_d3;       // ファイル３の静的荷重
   extern  float* F_al;            // 応力位置決定出力用データ
   extern  int*   F_alqq;          // 応力設定用オプション（１：モーメント、２：せん断力）
//                                 ●　ウインドウ管理
   extern  int    F_time_ii;       // 解析過程のステップ数
   extern  int    F_read_mass;     // 節点質量動的確保オプション
   extern  int*   F_mass;          // 節点質量表示用データ
   extern  int    mem1;            // 画面出力ワーク領域
   extern  int    mem2;            // 画面出力ワーク領域
   extern  int    fno_xx;          // 画面コードワーク領域
   extern  int    fno_yy;          // 画面コードワーク領域
   extern  int    F_wind[100][5];  // ウインドウ管理配列
   extern  HWND   F_hwnd_timer;    // 解析を開始したウインドウのハンドル
   extern  HWND   F_hwind[100];    // ウインドウハンドル管理配列
//                                 ●　透視図描画用データ
   extern  float F_scrnps[3],F_viewps[3],F_scalps,F_scalep[5],F_mag[6],FF_scalps;
//  透視画面の原点位置、視点位置、透視図スケール、
//  加速度などのスケール、矢印などの大きさ設定、透視図スケールの保存用
   extern  int  mnbsb,nrzero,locod,node,nelem;
// 部材数、境界節点数、局所座標数、節点数、要素数
   extern  int    F_pos[2][2];     // 画面の左上と右下の座標
   extern  int    F_pos_pr[2][2];  // プリンタ用紙の左上と右下の座標
   extern  CPoint F_point;         // マウス位置の座標を入れる入れ物
   extern  float  a_Zoom;          // マウス操作の拡大・縮小の大きさ
   extern  float  F_amx,F_amy;     // 図形移動量
   extern  int    F_hantei;        // マウス操作判定領域
   extern  float  F_Time;          // 解析過程における時間
   extern  int    F_ontimer;       // 解析状態を表すパラメータ
//                                 ●　動的コントロールその１
   extern  int    F_nindis;        // 初期不整使用有無パラメータ
   extern  float  F_gindis,F_f1sec;// 初期不整大きさ、第一段階解析時間
//                                 ●　動的コントロールその２
   extern  int    F_nstep,F_igra[3],F_ibata,F_ntime;
//解析ステップ数、地震波解析使用パラメータ、βの値を決めるパラメータ、反復最大回数
   extern  float  F_delt,F_beta,F_gamma,F_xgal[3],F_dlamst,F_delt_cl,F_all_time;
//解析用増分時間、ニューマークβ、ニューマークγ、地震最大加速度
//結果を出力する時間、解析時間
//                                 ●　解析結果の出力コントロール
   extern  int    F_iwstp,F_mstep,F_all_step;
// 解析結果をファイルに出力する間隔、描画する間隔、解析全ステップ
   extern  float  F_soutsc,F_dmaxck;
// ＊＊結果を出力する最初の時刻、崩壊と見做す最大変位（現在使用されていない）
   extern  float  F_delugg;          // 地震波形増分時間
//                                 ●　荷重、反力と不釣合い力
   extern  int    F_read_disp;       // ３次元座標の設定オプション
   extern  float* F_disp;            // 解析モデルの３次元座標
   extern  int    F_read_unbalanceF; // 不釣合力ベクトルの設定オプション
   extern  float* F_unbalanceF;      // 不釣合力ベクトル
   extern  float  F_unbalanceF_max;  // 不釣合力ベクトルの最大値
   extern  int    F_read_ndbalanceF; // 釣合力ベクトルの設定オプション
   extern  float* F_ndbalanceF;      // 釣合力ベクトル
//                                 ●　節点速度と加速度
   extern  int    F_read_vel;        // 速度ベクトルの設定オプション
   extern  float* F_vel;             // 速度ベクトル
```

```
   extern  int    F_read_acc;        // 加速度ベクトルの設定オプション
   extern  float* F_acc;             // 加速度ベクトル
   extern  int    F_read_accab;      // 絶対加速度ベクトルの設定オプション
   extern  float* F_accab;           // 絶対加速度ベクトル
   extern  float* Rrot ;             // 座標変換行列
   extern  int    mm_opt[10];        // 部材描画オプションの入れ物
//                              ●   部材応力
   extern  int    F_read_spring;     // 部材応力ベクトルの設定オプション
   extern  float* F_fay;             // 降伏関数値ベクトル
   extern  float* F_n_spring;        // 軸力ベクトル
   extern  float* F_my_spring;       // ｙ軸曲げモーメントベクトル
   extern  float* F_mz_spring;       // ｚ軸曲げモーメントベクトル
   extern  int*   F_stat_spring;     // 部材の弾塑性状態
   extern  int    F_mtype[5];        // 塑性ヒンジ出力位置の個数を設定する入れ物
//                              ●   グローバル変数の定義
   extern  int    n_iterate;          // 反復回数
   extern  int    nm_iterate;         // 陰解法回数
   extern  int    numb_method;        // 解析法番号
```

この中で注目すべき項目は、動的領域の宣言である。例えば、先に示したsf3stdatex.hの

```
extern int F_read_disp;
extern float* F_disp;
```

では、変数F_dispに記号＊が付いている。これは、この変数が動的領域であることを意味する。そのためこの領域を使用するためにはプログラムの中で動的領域確保を行うコードが必要となる。

```
//    ―――――――――――――――――――――――――――――
//      ●  動的領域の確保
//    ―――――――――――――――――――――――――――――
  if(F_read_disp == 0){
     F_disp = new float[3*node];
      if(F_disp == 0){
        MessageBox("Sorry! Memory allocation failure.");
      return;
  }}
```

ただし、ユーザーがそのコードを実際に実行し、その変数の動的領域確保がなされたかどうかは、プログラムを記述した段階では不明である。そのため、ここでは動的領域確保を行った場合は、その上の変数F_read_dispを１に設定する。動的領域確保を行ったかどうかを知る必要が生じたとき、この変数を利用する。例えば、デストラクタでは、以下に示すように、このF_read_dispをチェックして動的領域を解放するか否かを決定している。

```
if(F_read_disp == 1){
delete [] F_disp;
}
```

記号＊が付いた他のグローバル変数も同様な使い方をしており、これらの変数の全てが実行時に大きさが決定する配列として用いられる。

次に、先に示したクラスコンストラクタ CMainFrame::CMainFrame()を見てみよう。ここでは３つの処理を行う。一つ目はグローバル変数の初期化であり、二つ目は図形用データや節点変位などの動的領域を確保すること、三つ目は動的解析用パラメータや動的構造データを入力することである。

コンストラクタの最初の部分は、グローバル変数の初期値セットであり、図形処理用やウインドウ管理用など各種のグローバル変数が初期化されている。次に、描画ペンの設定を行う関数 create_penx( )をコールする。この関数では、図形用カラーペンの太さや色を設定している。この関数 create_penx( )については後で説明する。

次に、動的解析を制御するパラメータをファイルより入力する。これは、FORTRAN で記述したサブルーチン submain_dynamic_a()でも同じファイルからデータ入力を行っており、この制御パラメータや構造データは２度読み込まれることになる。最初に、関数 SYSNAM_C( )によって、SPACE からの伝達情報とコントロールファイルを読み込み、動的解析に必要となる各ファイル名をセットする。SPACE システムでは大文字で記述されている関数は SYSNAM_C( )のように FORTRAN のサブルーチンを表し、他のサブルーチンも全てこの仕様で書かれているので記憶されたい。

次は、透視図用ファイル、動的解析用制御ファイルその１、動的解析用制御ファイルその２、動的解析出力制御ファイルを読み込む。それらの関数は、

１．PERSCT( )　２．DYCTL1_C( )　３．DYCTL2_C( )　４．DOUTCL_C( )

であり、これらの関数は全て FORTRAN で記述されている。詳細は、付録を参照されたい。これらの関数から得られる情報から、描画と解析制御に必要なデータを取り出し、グローバル変数にセットする。

このコンストラクタの第 2 段階は、構造データを予備入力し、描画に必要な配列を動的確保することである。構造用ファイルの予備入力は、関数 INPTFX()で行う。この関数で、構造用ファイルの制御データを読み込み、その後、構造用制御パラメータを用いて描画用配列を動的に確保

する。例えば、配列 posit は節点の座標を表し、以下のように new コマンドで動的領域を確保する。次の if 文で posit に値がゼロの場合は、動的確保に失敗したことを意味し、エラーメッセージを表示する。

```
posit = new float[nnode*ndim];
if(posit == 0){
  MessageBox("Sorry! Memory allocation failure. You should stop this application.");
              }
```

最後の第 3 段階では、実際に構造データファイルを読み込む。ただし、描画に不必要なデータは読み飛ばすことになる。この構造データファイルを読むプログラムは、関数 INPTFY()であり、FORTRAN で記述されている。このコンストラクタの説明で分かるように、構造データファイルと上記のパラメータファイルは、数値計算に入る前にも、オープンされて読まれている。これらのファイルの仕様を変更する場合は、この部分のデータ入力プログラムも同様に変更しなければならない。

続いて、デストラクタについて説明する。デストラクタ~CMainFrame()における主な仕事は、グローバル動的領域を解放することである。特に、変数 F_read_spring 入力パラメータが１となっている場合、ユーザーが応力などの表示を行うよう指示したために動的領域を確保したことを意味し、デストラクタでは、これらの動的領域を解放する。同様に、F_read_ndbalanceF、F_read_mass は不釣合力と質量に関連する入力パラメータである。上記の動的領域を解放するコードを以下に示す。

```
if(F_read_spring == 1){
        delete [] F_fay;
        delete [] F_n_spring;
        delete [] F_my_spring;
        delete [] F_mz_spring;
        delete [] F_stat_spring;
 }
```

関数 create_penx()は、クラス CmainFrame のメンバー関数であり、コンストラクタの中で呼ばれている。ここでは、ペンの太さと色を設定し、また、フォントの設定も行っている。これらのフォントやペンのデータは、グローバル変数にセットされており、他のクラスの図形描画の中で使用されることになる。

クラス CmainFrame の中の他のメンバー関数として、OnClose()と OnCreate()関数がある。これらは、メインウインドウが閉じられるときと作られるときに呼ばれる関数である。特に、OnClose()関数では、計算実行中を表すグローバル変数 thread_on が 0 以外の場合は、ウインド

ウを閉じることを拒否するように設定されている。

8.3 動的解析システムの子ウインドウ管理

本節では、動的解析システム内の子ウインドウがどのように管理されているかについて解説する。動的解析システムに起動がかかると、動的解析システムは、まず、一般的な手続きを行った後、Mainfrm.cpp 内の Mainframe クラスで初期設定を行う。その後、ひとつの子ウインドウに関するオブジェクトを構築し、子ウインドウを描画する。

子ウインドウ描画管理は、CSf3stView クラスによって行われ、また、動的ソルバーも、この CSf3stView クラスの中で管理されている。そのため、動的解析システムが立ち上がったとき、全ての子ウインドウを消去すると CSf3stView オブジェクトが消滅してしまうため、動的解析が実際に実行できなくなる。具体的には、実行や停止などのメニューが使用不可となる。また、解析を実行したとき、アクティブであった子ウインドウが消去され、そのウインドウを管理していたオブジェクトが消去されると、動的解析の継続が不可能となる。つまり、動的ソルバーは、解析が開始したときアクティブであった子ウインドウの CSf3stView クラスオブジェクトによって管理されることになる。

まず、この CSf3stView クラスのコンストラクタとデストラクタ及びメッセージマップについて説明しよう。該当する部分の C++コードを以下に示す。

```
//    ―――――――――――――――――――――――――――――――――――――――
//      ●   CSf3stView クラス
//    ―――――――――――――――――――――――――――――――――――――――
//      ●   CSf3stView クラスの動作の定義を行います
//    ―――――――――――――――――――――――――――――――――――――――
//      ●   ヘッダーファイルの定義
//    ―――――――――――――――――――――――――――――――――――――――
#include "stdafx.h"         !
#include "sf3st.h"          !
#include "sf3stdat.h"       ! グローバル変数定義用
#include "fort.h"           ! FORTRAN 用インターフェイス
#include "sf3stDoc.h"       ! このクラスの Doc ヘッダーファイル
#include "sf3stView.h"      ! このクラスの View ヘッダーファイル
#include "DialogView.h"     ! このクラスの Dialog ヘッダーファイル
#include "MainFrm.h"        !
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
//    ―――――――――――――――――――――――――――――――――――――――
//      ●   メッセージによる関数呼び出し
```

```
//    ―――――――――――――――――――――――――――――――――――――――――――――――
IMPLEMENT_DYNCREATE(CSf3stView, CView)
BEGIN_MESSAGE_MAP(CSf3stView, CView)
          //{{AFX_MSG_MAP(CSf3stView)
          ON_COMMAND(ID_ANALYSIS_INIT, OnAnalysisInit)
          ON_WM_RBUTTONDOWN()
          ON_WM_LBUTTONDOWN()
          ON_WM_LBUTTONUP()
          ON_WM_MOUSEMOVE()
          ON_COMMAND(ID_SWND_ST, OnSwndSt)
          ON_COMMAND(ID_WND_PROP, OnWndProp)
          ON_COMMAND(ID_DYN_MAG, OnDynMag)
          ON_COMMAND(ID_SWND_WV_Y, OnSwndWvY)
          ON_COMMAND(ID_SWND_WV_Z, OnSwndWvZ)
          ON_COMMAND(ID_SWND_WV_DIS, OnSwndWvDis)
          ON_COMMAND(ID_SWND_WV, OnSwndWv)
          ON_COMMAND(ID_DYN_STOP, OnDynStop)
          ON_COMMAND(ID_DYN_RESTART, OnDynRestart)
          ON_COMMAND(ID_DYN_CLEAR, OnDynClear)
          ON_WM_DESTROY()
          ON_COMMAND(ID_DISPLAY_CC_PANEL, OnDisplayCcPanel)
          ON_WM_LBUTTONDBLCLK()
          //}}AFX_MSG_MAP
  // 標準印刷コマンド
          ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
          ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
          ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
          ON_MESSAGE(WM_THREADFINISHED, OnMessageWind)
          ON_MESSAGE(IDS_MESSAGE_WIND, OnMessageWind_G)
END_MESSAGE_MAP()
//    ―――――――――――――――――――――――――――――――――――――――――――――――
//      ●    CSf3stView クラスの構築
//    ―――――――――――――――――――――――――――――――――――――――――――――――
CSf3stView::CSf3stView()
{
//    ―――――――――――――――――――――――――――――――
//      ●    制御用データの初期設定
//    ―――――――――――――――――――――――――――――――
           m_base[0] = 0.;
          m_base[1] = 0.;
          m_base[2] = 0.;
          ButtonDown = FALSE;
          m_nstep=0;
          m_dat_struct = 0;
          m_dat_mode_number=1;
          m_pst_disp = 0;
          m_pst_hinge = 0;
           m_pst_color = 0;
          m_pst_options = 0;
           m_pst_graph = 0;
          m_pst_arrow= 0;
          m_pst_mass = 0;
           m_pst_dis = 0;
           m_pst_ef1 = 1.;
```

```
            m_pst_ef2 = 0.;
            On_mem_option =0;
            m_radio_mem = 0;
            m_radio_bound =0;
            m_radio_mass =0;
            m_arrow = F_mag[0];    //   矢印の大きさ
            m_bend  = F_mag[1];
            m_mag   = F_mag[2];
            m_shear = F_mag[3];
            m_stress = F_mag[4];
            m_strain = F_mag[5];
            m_radio_load = 0;
            m_radio_cood =0;
//  ―――――――――――――――――――――――――――――――
//    ●   モードレス  ダイアログの作成
//  ―――――――――――――――――――――――――――――――
      m_pDlg = new CDig_CC_panel(this);
}
//  ―――――――――――――――――――――――――――――――――――――――――――――
//    ●   CSf3stView クラスの消滅
//  ―――――――――――――――――――――――――――――――――――――――――――――
CSf3stView::~CSf3stView()
{
//  ―――――――――――――――――――――――――――――――
//    ●   モードレス  ダイアログの消去
//  ―――――――――――――――――――――――――――――――
        delete m_pDlg;
        View_C_panel = 0;
}
//  ―――――――――――――――――――――――――――――――――――――――――――――
//    ●   ウインドウ管理システム
//  ―――――――――――――――――――――――――――――――――――――――――――――
//    ●   ウインドウコードの取得
//  ―――――――――――――――――――――――――――――――――――――――――――――
 int CSf3stView::getwindx(long fno)
{
        int iix ;
        iix = (int) fno -1;
        if(iix >= 0 && iix < fno_yy) {
        if( F_wind[iix][0] == 1){
          iix = F_wind[iix][1];
          return iix;
        }}
        iix= -1;
        return iix;
}
//  ―――――――――――――――――――――――――――――――――――――――――――――
//    ●   ウインドウのハンドル取得
//  ―――――――――――――――――――――――――――――――――――――――――――――
 HWND CSf3stView::getwindh(long fno)
{
        int iix ;
        iix = (int) fno -1;
        if(iix >= 0 && iix < fno_yy) {
```

```
          if( F_wind[iix][0] == 1){
            return  (F_hwind[iix]);
          }}
         return (NULL);
}
//   ―――――――――――――――――――――――――――――――――――――――――――――――
//     ●   ウインドウコードのセット
//   ―――――――――――――――――――――――――――――――――――――――――――――――
void CSf3stView::setwindy(long fno,int ii)
{
          int iix = (int) fno;
          F_wind[iix-1][2]=ii;
}
//   ―――――――――――――――――――――――――――――――――――――――――――――――
//     ●   ウインドウコードを取得
//   ―――――――――――――――――――――――――――――――――――――――――――――――
int CSf3stView::getwindy(long fno)
{
          int iix = (int) fno;
          return F_wind[iix-1][2];
}
//   ―――――――――――――――――――――――――――――――――――――――――――――――
//     ●   ウインドウの初期セット及ぶコード、ハンドルセット
//   ―――――――――――――――――――――――――――――――――――――――――――――――
long CSf3stView::setwindx(long fno,int ii,HWND hWnd)
{
//   ―――――――――――――――――――――――――――――――
//     ●  ウインドウコード
//
//        fno_yy :: 現在開いているウインドウ数
//        F_wind[iix-1][0]::0:空き、1:現在使用
//        F_wind[iix-1][1]::ウインドウコード
//            STRUCT=1,WAVE=2,SLOAD=3,STRESS=4,MODEWD=5,DISMAX=6,DEMOD=7,SECTION=8
//        F_wind[iix-1][2]= :構造図出力用コード (0-5)
//          0:オプション図形;1:
//        F_wind[iix-1][3]=0:ダミー
//        F_wind[iix-1][4]=0:ダミー
//        F_hwind[iix-1] = hWnd : ウインドウのハンドル
//   ―――――――――――――――――――――――――――――――
          int i;
          int iix;
          iix = (int) fno;
          if(fno_yy != 0){
//   ―――――――――――――――――――――――――――――――
//     ● 指定した領域にデータをセット
//   ―――――――――――――――――――――――――――――――
          if(iix > 0 && iix <= fno_yy) {
             F_wind[iix-1][0]=1;
             F_wind[iix-1][1]=ii;
             F_wind[iix-1][2]=0;
             F_wind[iix-1][3]=0;
             F_wind[iix-1][4]=0;
             F_hwind[iix-1] = hWnd;
             return fno;
```

```
                }
//    ―――――――――――――――――――――――――――――――――
//      ●　空きとなっている制御データ領域を探索し、
//          あればそこにデータをセット
//    ―――――――――――――――――――――――――――――――――
     for(i = 0; i<fno_yy; i++){
       if(F_wind[i][0] == 0 ){
            F_wind[i][0]=1;
            F_wind[i][1]=ii;
            F_wind[i][2]=0;
            F_wind[i][3]=0;
            F_wind[i][4]=0;
            F_hwind[i] = hWnd;
            return i+1;
           }}
}
//    ―――――――――――――――――――――――――――――――――
//      ●　新規ウインドウ制御データ領域を確保
//    ―――――――――――――――――――――――――――――――――
            F_wind[fno_yy][0]=1;
            F_wind[fno_yy][1]=ii;
            F_wind[fno_yy][2]=0;
            F_wind[fno_yy][3]=0;
            F_wind[fno_yy][4]=0;
            F_hwind[fno_yy] = hWnd;
            fno_yy = fno_yy + 1;
            fno = fno_yy;
  return fno;
}
//    ―――――――――――――――――――――――――――――――――――――――――――
//      ●　ウインドウが生きているかどうかチェック
//    ―――――――――――――――――――――――――――――――――――――――――――
BOOL CSf3stView::checkhwnd(HWND hWnd,int jj)
{
         BOOL ihan = IsWindow(hWnd);
              if(ihan) return (ihan);
              F_wind[jj][0]=0;
              F_wind[jj][1]=0;
             F_wind[jj][2]=0;
             F_wind[jj][3]=0;
             F_wind[jj][4]=0;
             F_hwind[jj] = NULL;
         return (ihan);
}
//    ―――――――――――――――――――――――――――――――――――――――――――
//      ●　ウインドウを閉じる処理とウインドウ管理データの消去処理
//    ―――――――――――――――――――――――――――――――――――――――――――
void CSf3stView::OnDestroy()
{
  if(thread_on != 0) {
    MessageBox("現在、計算実行中もしくは停止中です。計算を中止した後ウィンドウを閉じてください。");
  return;
         }
         HWND hwnd = this->GetSafeHwnd();
```

```
        CView::OnDestroy();
        setwindcheck_x(hwnd );
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　閉じたウインドウの管理データ領域を解放する
//    ―――――――――――――――――――――――――――――――――――――――――
BOOL CSf3stView::setwindcheck_x(HWND hWnd)
{
        BOOL ihan = FALSE;
        if(fno_yy != 0){
        for(int i = 0; i<fno_yy; i++){
         if(F_wind[i][0] != 0 ){
          if(F_hwind[i] == hWnd){
           if(hWnd == F_hwnd_timer  ) F_ontimer = 0;
            F_wind[i][0]=0;
           F_wind[i][1]=0;
            F_wind[i][2]=0;
            F_wind[i][3]=0;
            F_wind[i][4]=0;
            F_hwind[i] = NULL;
            ihan = TRUE;
          }
         }}}
  return (ihan);
}
```

このクラスは、動的ソルバー管理と図形描画処理を主に行っており、そのため多数のメンバー関数が存在する。プログラムを見ると、まず、多くのヘッダーファイルがインクルードされていることが目に付く。各々のヘッダーファイルには、内容を記したコメントが付されており、各ファイルの内容は付録を参照されたい。これら多くのヘッダーファイルの中で、このクラスに付属する sf3stView.h の内容中、重要な部分を以下に示す。詳しい sf3stView.h の内容については、後節で説明する。

最初は、他のヘッダーファイルのインクルード文が続く。これは、sf3stView.h ファイルの中で、これらのクラスを参照しているからである。次に、define 文があるが、ここでは、各種のメッセージの番号を変数に割り付けている。例えば、WM_THREADFINISHED は、スレッドが終了したときに送るメッセージであり、番号としては、WM_USER+5 の値が設定されている。次の enum{}文は、FORTRAN の data 文と同様に、変数に値を割り付けている。例えば、STRUCT は１に、WAVE は２などである。

WM_USER は、C++システムが開発者に許しているメッセージ番号であり、プログラム内の私用メッセージ番号は、これ以降の値を用いることになる。

クラスの中でメンバー関数とメンバー変数が定義されている。この中で、public:と protected:文の下で定義されているメンバー関数やメンバー変数は、各々使用方法が異なる。public:以下で定義されている関数や変数は、他のクラスでも使用可能であるが、protedted:以下で定義されている関数や変数はこのクラスの中でしか使用できない。これらの

関数の説明は各処理の中で行う。

```
//     ―――――――――――――――――――――――――――――――――――――――――――――
//     ●   sf3stView.h
//     ―――――――――――――――――――――――――――――――――――――――――――――
//     ●   CSf3stView クラスの宣言およびインターフェイスの定義
//     ―――――――――――――――――――――――――――――――――――――――――――――
#include "Dig_pstruct.h"
#include "Dig_mag.h"
#include "Sf31wave.h"
#include "Nodedis.h"
#include "Dig_yesno.h"
#include "Dig_CC_panel.h"
#define WM_THREADFINISHED   WM_USER + 5
#define IDS_MESSAGE_WIND    WM_USER + 6
#define IDS_MESSAGE_DIALOG   WM_USER + 7
enum{STRUCT=1,WAVE=2,SLOAD=3,STRESS=4,MODEWD=5,DISMAX=6,DEMOD=7,SECTION=8 };
class CSf3stView : public CView
{
protected: // シリアライズ機能のみから作成します。
          CSf3stView();
          DECLARE_DYNCREATE(CSf3stView)
    public:
           CDig_mag         dig_mag;
          CSf31wave         sf31wave;
    public:
          virtual void OnDraw(CDC* pDC);  // このビューを描画する際にオーバーライドされます。
    protected:
           afx_msg LONG OnMessageWind(UINT wParam, LONG lParam);
           afx_msg LONG OnMessageWind_G(UINT wParam, LONG lParam);
    public:
          virtual ~CSf3stView();
    protected:
           void OnAnalysisGo();
          int getwindx(long);
           HWND getwindh(long);
           void setwindy(long,int);
           int getwindy(long);
           long setwindx(long ,int,HWND);
           BOOL checkhwnd(HWND,int);
           BOOL setwindcheck_x(HWND);
           BOOL setwindcheck();
           void pre_disp_mem_persp(HWND,int);
          void disp_mem_persp(CDC*);
// 生成されたメッセージ マップ関数
protected:
          afx_msg void OnAnalysisInit();
          afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
          afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
          afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
          afx_msg void OnMouseMove(UINT nFlags, CPoint point);
          afx_msg void OnSwndSt();
          afx_msg void OnWndProp();
```

```
afx_msg void OnDynMag();
afx_msg void OnSwndWvY();
afx_msg void OnSwndWvZ();
afx_msg void OnSwndWvDis();
afx_msg void OnSwndWv();
afx_msg void OnDynStop();
afx_msg void OnDynRestart();
afx_msg void OnDynClear();
afx_msg void OnDestroy();
afx_msg void OnDisplayCcPanel();
afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
};
```

次に、メッセージマップを見てみよう。このクラスのメッセージマップは、コードに見られるように多くの関数を含む。これらの関数は２つに分類される。ひとつは、ON_COMMAND()で始まるコードであり、第一引数は、メッセージ、第二引数は、そのメッセージがこのクラスに届いたとき実行される関数である。無論、この関数はクラスのメンバー関数であり、その関数では各種の処理が行われる。また、メッセージは他のクラスを含めてプログラムの中で発せられる。

次は、ON_WM_RBUTTONDOWN()のように ON_WM で始まる関数であり、これらの関数は、ウインドウからのメッセージによって駆動される。例えば、ON_WM_RBUTTONDOWN()がメッセージで駆動されると、メンバー関数 OnRButtonDown(UINT nFlags, CPoint point)が実行される。これらのメッセージは、主に、ユーザーのマウス操作などによって発せられ、アクティブとなっている子ウインドウのオブジェクトに届くようなメカニズムとなっている。

次に、各メンバー関数について説明しよう。まず、関数 CSf3stView のコンストラクタとデストラクタについてであるが、コンストラクタ CSf3stView::CSf3stView()では、このオブジェクトを制御するデータ群の初期設定を行う。次に、モードレスダイアログとして表示する操作パネルのオブジェクトを作成する。ただし、ここでは、パネルの表示を行わず、ユーザーが要求したときパネルの表示が行われる。次に、デストラクタ CSf3stView::~CSf3stView()では、操作パネルのオブジェクトを消去する。

デストラクタがオジェクの消去処理をするのと同時に、子ウインドウを消去する際の処理を以下の２つのメンバー関数が行う。

```
void CSf3stView::OnDestroy()
BOOL CSf3stView::setwindcheck_x(HWND hWnd)
```

子ウインドウを閉じるようユーザーからの指示があったとき、この関数OnDestroy()が呼ばれ、閉じてよいかどうかチェックする。グローバル変数 thread_on は、現在動的解析を実行中かもしくは停止中かを判断するパラメータであり、値が０の場合は計算停止中でそのウインドウを閉じる処理を行うことができるが、それ以外の値の場合は、エラーメッセージを表示してウインドウを閉じる操作を拒否する。

ウインドウを閉じて良い場合は、そのウインドウのハンドルを取得し、まず、Cview クラスのメンバー関数 OnDestroy()をコールする。その後、メンバー関数 BOOL CSf3stView::setwindcheck_x()を用いて、先に取得したウインドウハンドルを頼りに子ウインドウを管理する配列からこのウインドウを抹消する。

前段の説明が長くなってしまったが、これからは、SPACE における各種の管理技法について述べることにする。これらはクラス CSf3stView に実装されている。

8.4 マルチスレッド処理と動的ソルバー

本節では、マルチスレッド処理と動的ソルバーの管理について解説する。マルチスレッド処理を利用した動的ソルバーは、CSf3stView クラスで実現している。前節に続いて、CSf3stView クラスのメンバー関数について見ていこう。ここでは、このクラスで動的ソルバーに関連するメンバー関数とメッセージマップを取り上げる。ここで説明するメンバー関数は、

```
void CSf3stView::OnAnalysisInit()
void CSf3stView::OnAnalysisGo()
UINT threadprocx(LPVOID pParam)
UINT threadprocy(LPVOID pParam)
LONG CSf3stView::OnMessageWind(UINT wParam, LONG lParam)
```

の５つである。それでは、関連するメッセージマップと、この５つのメンバー関数の内容を具体的に示すことにしよう。

```
BEGIN_MESSAGE_MAP(CSf3stView, CView)
        ON_COMMAND(ID_ANALYSIS_INIT, OnAnalysisInit)
        ON_MESSAGE(WM_THREADFINISHED, OnMessageWind)
END_MESSAGE_MAP
//    ―――――――――――――――――――――――――――――――――――――――
//      ●　解析開始：解析開始用ボタンによって起動される
//    ―――――――――――――――――――――――――――――――――――――――
void CSf3stView::OnAnalysisInit()
{
```

```
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●　解析中におけるこのコマンドの実行排除
//    ―――――――――――――――――――――――――――――――――――――――――――――
          if(thread_on == 1 ) {
    MessageBox("現在計算中です。このコマンドは無視します。");
          return;
          }
          if(thread_on == 2 ) {
    MessageBox("現在計算停止中です。計算を中止した後、実行して下さい。");
          return;
          }
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●　固有値解析開始：マルチスレッド処理開始
//    ―――――――――――――――――――――――――――――――――――――――――――――
          if(F_calnum == 6){
          MessageBox("固有値解析開始");
          thread_on = 1;
          CWinThread* pThread =
          AfxBeginThread(threadprocy, GetSafeHwnd(),THREAD_PRIORITY_NORMAL);
          return;
          }
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●　時刻歴解析開始
//    ―――――――――――――――――――――――――――――――――――――――――――――
//    ―――――――――――――――――――――――――――――――
//      ●　予備計算：シングルスレッド
//    ―――――――――――――――――――――――――――――――
           icontrol = 0;
          thread_on = 1;
          T_analysis =0;
          nm_iterate =0;
          int nx_step;
          MessageBox("予備計算開始");
    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);
          F_time_ii =0;
          m_nstep = nx_step;
          F_Time=0;
          if(ierr_dat != 0) {
//    ―――――――――――――――――――――――――――――――
//      ●　予備計算でエラーが発生した
//    ―――――――――――――――――――――――――――――――
          int nx_step;
          icontrol = 99;
//    ―――――――――――――――――――――――――――――――
//      ●　後処理：動的領域の解放を行う
//    ―――――――――――――――――――――――――――――――
    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);
//    ―――――――――――――――――――――――――――――――
//      ●　エラーの原因を表示
//    ―――――――――――――――――――――――――――――――
          err_out(ierr_dat);
          if(ierr_dat == 299 ){
          MessageBox("モデルは解析制限を越えていました。処理を中止します。");
          }else{
          MessageBox("エラーがありました。処理を中止します。エラーの詳細は、表示：動的解析の途中結果
の表示を見てください。");
          }
          thread_on = 0;
    return  ;
          }
//    ―――――――――――――――――――――――――――――――
//      ●　予備計算正常終了
//    ―――――――――――――――――――――――――――――――
//    ―――――――――――――――――――――――――――――――
//      ●　図形処理
//    ―――――――――――――――――――――――――――――――
          Ontime();
          MessageBox("予備計算終了:計算開始");
//    ―――――――――――――――――――――――――――――――
//      ●　解析開始：マルチスレッド処理開始
//    ―――――――――――――――――――――――――――――――
          OnAnalysisGo() ;
}
//    ―――――――――――――――――――――――――――――――――――――――――――
//      ●　　動的解析用スレッド「threadprocx」を発生させる。
//    ―――――――――――――――――――――――――――――――――――――――――――
void CSf3stView::OnAnalysisGo()
{
          icontrol=1;                        // 1:解析実行中パラメータ
          ierr_dat=0;
          CWinThread* pThread =
          AfxBeginThread(threadprocx, GetSafeHwnd(),THREAD_PRIORITY_NORMAL);
}
//    ―――――――――――――――――――――――――――――――――――――――――――
//      ●　　動的解析スレッド：
//    ―――――――――――――――――――――――――――――――――――――――――――
UINT threadprocx(LPVOID pParam)
{
    ierr_dat=0;
//    ―――――――――――――――――――――――――――――――
//      ●　　動的解析用 FORTRAN サブルーチンを呼ぶ
//    ―――――――――――――――――――――――――――――――
    SUBMAIN_DYNAMIC_A(&F_calnum,&iend_code,&icontrol,&ierr_dat,&T_analysis,&dt_analysis,
           &n_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);
    F_time_ii = ns_step-1;
    F_Time=T_analysis;
//    ―――――――――――――――――――――――――――――――
```

```
//     ●    １回分の解析終了：終了メッセージを送る
//    ―――――――――――――――――――――――――――――――
    ::PostMessage((HWND) pParam, WM_THREADFINISHED, 0,0);
    return 0;
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//     ●    固有値解析スレッド処理
//    ―――――――――――――――――――――――――――――――――――――――――――――
UINT threadprocy(LPVOID pParam)
{
        ierr_dat=0;
//    ―――――――――――――――――――――――――――――――
//     ●  固有値解析用 FORTRAN サブルーチンを呼ぶ
//    ―――――――――――――――――――――――――――――――
        icontrol =2;
        SUBMAIN_DYNAMIC_B(&F_calnum,&iend_code,&icontrol,&ierr_dat);
//    ―――――――――――――――――――――――――――――――
//     ●  解析終了コードのメッセージを送る
//    ―――――――――――――――――――――――――――――――
    ::PostMessage((HWND) pParam, WM_THREADFINISHED, 0,0);
    return 0;
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//     ●    解析終了処理：解析終了メッセージによって起動される
//    ―――――――――――――――――――――――――――――――――――――――――――――
LONG CSf3stView::OnMessageWind(UINT wParam, LONG lParam)
{
//    ―――――――――――――――――――――――――――――――
//     ●  固有値解析の後処理
//    ―――――――――――――――――――――――――――――――
        if(icontrol == 2){
//    ―――――――――――――――――――――――――――――――
//     ●  固有値解析エラーあり
//    ―――――――――――――――――――――――――――――――
            if(ierr_dat != 0) {
                    if(ierr_dat == 299){
            MessageBox("モデルは解析制限を越えていました。処理を中止します。");
                    }else{
            MessageBox("エラーがありました。処理を中止します。");
                    }
            thread_on = 0;
       return 1;
            }
//    ―――――――――――――――――――――――――――――――
//     ●  固有値解析正常終了
//    ―――――――――――――――――――――――――――――――
            MessageBox("固有値解析は正常終了しました。");
            thread_on = 0;
             return 1;
          }else{
//    ―――――――――――――――――――――――――――――――
//     ●  時刻歴解析エラーあり
//    ―――――――――――――――――――――――――――――――
            if(ierr_dat != 0) {
```

```
            MessageBox("エラーがありました。処理を中止します。");
            thread_on = 0;
             return 1;
          }
//    ―――――――――――――――――――――――――――――――
//      ●  時刻歴解析の後処理
//    ―――――――――――――――――――――――――――――――
        if(iend_code == 0){
//    ―――――――――――――――――――――――――――――――
//      ●  時刻歴解析一時停止処理
//    ―――――――――――――――――――――――――――――――
         if(F_holt== 1){
         Ontime() ;            //  図形処理
        MessageBox("処理を中断します。");
                }else{
//    ―――――――――――――――――――――――――――――――
//      ●  次ステップの時刻歴解析を実行
//    ―――――――――――――――――――――――――――――――
       OnAnalysisGo() ;       //  解析実行
       Ontime() ;             //  図形処理
                }
       }else{
//    ―――――――――――――――――――――――――――――――
//      ●  時刻歴解析正常終了
//    ―――――――――――――――――――――――――――――――
        Ontime() ;             //  図形処理
        MessageBox("動的解析は正常終了しました。");
        thread_on = 0;
        }
}
return 1;
}
```

ここでは、CSf3stView クラスの中でも、特に動的ソルバーがどのように管理されているかについて解説する。このクラスは、子ウインドウの描画に関連するメンバー変数やメンバー関数を多く有するものであるが、動的ソルバーも管理している。したがって、子ウインドウが全て閉じてしまうと動的ソルバーが動作しなくなることに注意しなければならない。

最初に、動的ソルバーがマルチスレッドで実行される仕組みを見てみよう。ここでは、このマルチスレッドに関連する部分を抽出して説明する。解析種別は、通常の非線形振動解析である。動的ソルバーを実行させる方法は各種あるが、SPACE では最も単純な方法を用いている。まず、ツールバーの解析開始チップを押すか、メニューの動的解析開始を選択すると、メッセージ ID_ANALYSIS_INIT が出される。これを受けて、クラス CSf3stView のメッセージマップ中の ON_COMMAND で処理され、関数 OnAnalysisInit()が実行されることになる。

動的解析を実行するために、クラス CSf3stView のメンバー関数 OnAnalysisInit()の時刻歴解析開始処理へ飛び、動的解析を制御するグローバル変数を初期化した後、予備計算開始を画面に表示する。その後、図形描画用の配列と共に次に示す制御パラメータを引数として、動的ソルバーのサブシステム SUBMAIN_DYNAMIC_A()をコールする。下の表で示した変数名はサブルーチン内の仮引数であり、また、括弧内の変数名はC++で書かれている関数の実引数名である。

i_calnum(F_calnum)	:解析種別
iend_code	:計算終了コード 0=継続　1:終了
icontrol	:計算コード　　0:予備計算　1:動的解析 99:終了処理
ierr_dat	:エラーコード
T(T_analysis)	:計算時刻
dt(dt_analysis)	:解析増分時間
n_step(nx_step)	:今回の解析ステップ数
ns_step(ns_step)	:解析開始ステップ（最初はゼロセットが必要）
n_iterate	:反復回数

最初に呼ぶサブシステム SUBMAIN_DYNAMIC_A()では、icontrol=0 として、予備計算を実行する。ここでは、新たなスレッドを発生せずにプログラムの制御は、サブシステム SUBMAIN_DYNAMIC_A()に受け渡される。そのため、予備計算では新たなスレッドを発生させずに実行される。

予備計算が終了して、制御が SUBMAIN_DYNAMIC_A()から戻ると、エラーコード ierr_dat の値をチェックして、予備計算の段階でエラーがあったかどうかチェックする。ここで、もしエラーがあった場合は、計算コード icontrol を 99 にセットした後、サブルーチン内で確保した動的領域を解放するために、再度、サブシステム SUBMAIN_DYNAMIC_A()を呼び、後処理を実行する。後処理が終了した後、関数 err_out( )でエラー情報をファイルに出力し、また、画面にエラーが存在したことを表示する。ユーザーが確認した後、return コードによって、関数を抜けることになる。

予備計算でエラーがない場合は、いよいよマルチスレッド技術を用いて動的解析を行うことになる。まず、図形処理を行うため、メンバー関数 Ontime()をコールする。ここで、図形処理の初期設定を行うことになる。「予備計算終了:計算開始」というメッセージを表示した後、マルチスレッド処理を行うメンバー関数 OnAnalysisGo()をコールし、この関数から制御が戻った後、この OnAnalysisInit()関数を抜け出る。これで、動的ソルバーの予備計算とエラー処理、マルチスレッド処理の開始と、主な OnAnalysisInit()関数の処理内容を説明した。いよいよ、マルチス

レッド処理を用いた動的ソルバーの数値計算処理の仕組みについて説明しよう。

メンバー関数 OnAnalysisGo()は、スレッドを発生させる非常に単純なルーチンである。計算コードを動的解析を示す icontrol=1 にセットし、また、エラーコード ierr_dat をゼロセットした後、スレッドを発生させる関数をコールする。C++の内部関数である AfxBeginThread()によってスレッドを発生させる。この関数の引数は３つあり、最初はスレッドを発生した後、そのスッレドで最初に実行される関数名、次は現在のオブジェクトのハンドル名、最後はこのスレッドの実行プライオリティを表す。詳細は、visual C++に関する参考書を参照されたい[6]。この関数は、スレッドを発生した後、直ちに元のコールした関数に戻る。これで、プログラム内には2つの処理、つまりスレッドが動いていることになる。ひとつは、新たなスレッドで、動的解析を実行する関数 threadprocx()が動作し、また、他の一つは、関数から抜け出てユーザーからのメッセージ、具体的には、図形の拡大や縮小、あるいは回転などの操作に関するメッセージを受け取り、その動作のための処理を行う。ここでは、前者を解析スレッド、後者を管理スレッドと呼ぶ。

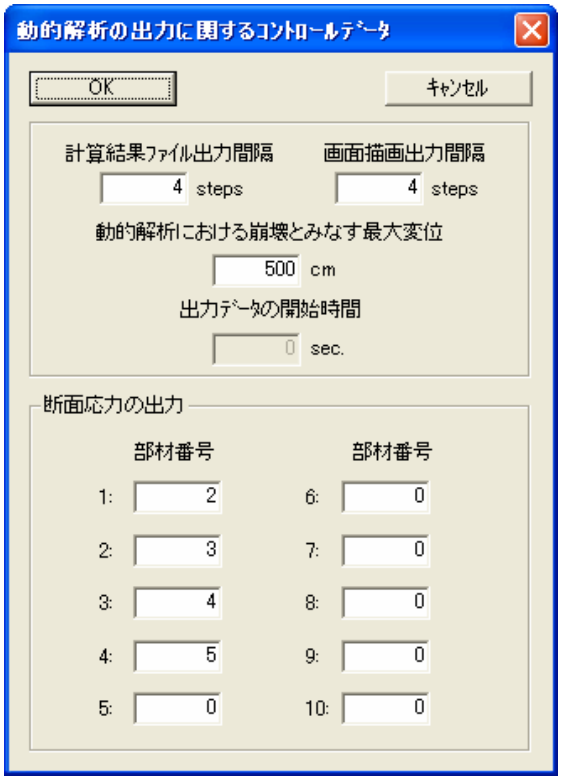

図 8-2 動的解析の出力に関するダイアログ

動的解析スレッドにおける関数は threadprocx()であり、この関数が

終了した時点でこのスレッドは消滅する。この関数では、FORTRAN で書かれた動的ソルバーSUBMAIN_DYNAMIC_A()をコールする。増分計算を数ステップ進めた後、動的ソルバーから制御が戻る。制御がこの関数に戻ると、解析の進行ステップ数と時間をセットする。この計算用ステップ数 ns_step とは、SPACE のダイアログ中の画面描画出力間隔で指定したものであり（図 8-2 参照)、ここでは１回分の解析と呼ぼう。最後に、この一回分の解析が終了し、解析スレッドが消滅することを知らせるために、このスレッドを発生させたオブジェクトにメッセージを送る。メッセージを送る関数は PostMessage()であり、WM_THREADFINISHED を引数とする。管理スレッドはこのメッセージを受け取ることで、動的解析の終了を知ることになる。このメッセージを出した後、この関数から抜け出し、解析スレッドが消滅する。

動的ソルバー終了のメッセージは、同じクラス CSf3stView の中のメッセージマップで受け取ることになる。

```
ON_MESSAGE(WM_THREADFINISHED, OnMessageWind)
```

この ON_MESSAGE()では、受け取ったメッセージが WM_THREADFINISHED であると、関数 OnMessageWind()が実行される。この関数は、解析終了処理を行うもので、管理スレッドで処理される。詳細な説明は、後にして、解析の流れを追うことにする。解析途中の場合は、時刻歴の後処理コードへと進み、そこで、グローバル変数 iend_code と F_holt をチェックした後、解析後の処理を決定する。まず、終了コード iend_code がゼロであるかどうかで、解析が終了したかどうかをチェックする。解析終了でない場合は、一時停止かどうか、解析停止コード F_holt でチェックする。

解析停止の場合は、図形処理を行う関数 Ontime()を実行した後、中断するという表示を行い、関数を抜ける。その後ユーザーからの反応を待つことになる。停止でない場合、これが通常の解析の処理となるわけであるが､新たなスレッドを発生させる関数 OnAnalysisGo()を再びコールし、解析を進める。新たな解析スレッドが発生し、動的ソルバーが計算を進めると同時に、管理スレッドは、直ちに元の関数に戻ってくる。その後、図形処理 Ontime()をコールした後、関数を抜ける。最後に、解析終了コードが１となっている場合は、正常終了の表示を行った後、スレッドコード thread_on をゼロとして関数を抜ける。

以上が、動的ソルバーの管理システムの一般的な処理内容である。そ

の仕組みを理解できただろうか。このような面倒な方法を取らなければならない理由は、数値計算と同時にリアルタイムでアニメーションを実行し、また、ユーザーからのメッセージを受け取る必要があるからである。もし、マルチスレッドで処理しないと数値計算の途中では、ユーザーの操作に対するシステムの応答がフリーズしてしまう。

マルチスレッドによる数値計算は、SPACE で用いた方法とは異なる手法でも実現可能である。例えば、動的ソルバーの SUBMAIN_DYNAMIC_A()の中で、図形処理メッセージを出せばよく、この場合、上記のように頻繁にスレッドを消滅させて戻る必要はなくなる。つまり、一度新スレッドを発生させ、そこで動的解析を最後まで実行し、適当な時間に図形処理を行うためのメッセージを出せば良いことになる。この例の方が、SPACE で実現したマルチスレッド処理より単純ではあるが、図形用の多くのデータをグローバル化しなければならないし、また C++と FORTRAN で書かれたプログラムのインターフェイスをなるべく単純にしたいという理由で、SPACE では現在の手法を選択した。

さらに、メンバー関数を詳細に観察し、動的ソルバー管理システムの動きを見てみよう。再度、メンバー関数 OnAnalysisInit()を見ると、最初に、この関数が呼ばれると、実行する場合の制限についての記述がある。つまり、スレッドコード thread_on が 1 及び 2 の場合、計算中か停止中、この関数は実行できないようになっている。最初に一度だけ、あるいは、計算が終了して再計算を行いたい場合のみ実行可能となっている。

次に、解析種別コードをチェックし、F_calnum が 6 の場合、固有値解析を行う。固有値解析開始のメッセージの後、新たなスレッドを発生させ、関数 threadprocy()を実行する。この関数では、FORTRAN で書かれた固有値解析用のソルバーSUBMAIN_DYNAMIC_B()を起動し、固有値解析が終了するまで待つ。終了すると、解析終了コード WM_THREADFINISHED をメッセージとして送り、その後、関数を閉じ、スレッドを消滅させた後、元の関数に戻る。

次に、解析終了処理を行う関数 OnMessageWind()を詳細に見てみよう。まず、計算コード icontrol が 2 の場合、固有値解析の終了処理となる。エラーコードをチェックし、固有値解析でエラーが発生した場合、そのエラーコードの値によって出力を変えて表示する。正常終了の場合は、正常終了と表示して関数から抜ける。

計算コードが 2 以外の場合、これは通常の動的解析にあたるが、エラーコードをチェックし、エラーが発生している場合は、処理を中止する

旨の表示を出した後、関数から抜ける。エラーが発生していない場合は、通常の応答解析の後処理であり、これについては上記した。

以上が、動的ソルバーのマルチスレッド化であり、動的ソルバーの管理法である。理解できただろうか。上記の解説をよく読み、プログラムコードとつき合わすことで、より理解が深まることでしょう。

動的ソルバーSUBMAIN_DYNAMIC_A()が出てきたついでに、このサブルーチンの引数、特に、図形処理用のデータについて説明しよう。このサブルーチンコールを再びここに示す。

```
//    ―――――――――――――――――――――――――――――
//      ●　動的解析用 FORTRAN サブルーチンを呼ぶ
//    ―――――――――――――――――――――――――――――
    SUBMAIN_DYNAMIC_A(&F_calnum,&iend_code,&icontrol,&ierr_dat,&T_analysis,&dt_analysis,
          &n_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);
```

上のサブルーチンの引数中で、節点の変位を表すF_disp[]と、この動的領域か確保されたか否かを示すパラメータF_read_dispに注目する。ユーザーが構造図を描くことを指示した場合、動的領域を確保し、このパラメータF_read_dispを１に設定する。このような状態になったとき、サブルーチン内でどのような処理を行い、図形用の節点変位を持ち帰るかを説明する。サブルーチンSUBMAIN_DYNAMIC_A()の中で、変位データと応力データのコピーが行われており、その部分のコードとそのサブルーチンを以下に示す。プログラムの内容は、第7.4.3節で示した節点変位の出力サブルーチンOut_disp_vell_acc()と、第7.4.4節で示した部材の応力出力サブルーチンOut_stress()を参照すれば、理解できよう。後に示すように構造透視図処理では、ここでセットした変位や応力が用いられることになる。

```
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
c―――――――――――――――――――――――――――――――――――――★応力
      if(i_read_spring .ne. 0) then
      call Set_preset_spring(Member,Element,E_model6_real,
     *               M_model11,M_model12,M_model13,
     *               M_model15,M_model21,M_model22,
     *               n_member,F_fay,F_n_spring,F_my_spring,
     *               F_mz_spring,i_stat_spring)
```

```
      endif

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
      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
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
c-------------------------------------------------------★
      do i=1,n_point
      ij=Point(i).local_coord
      if(ij.eq.0) then
      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
      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))
      do j=1,3
       F_disp(j,i)=vv(j)
      enddo
      endif
      end do
      endif
      return
      end
      endif
C     ------------------------------------------------------------------
C     ●   SUBROUTINE /Set_preset_spring
C       ----------------------------------------------------------------
C     ●   描画用データのセット（部材応力）
C     ------------------------------------------------------------------
      subroutine Set_preset_spring(Member,Element,E_model6_real,
     *               M_model11,M_model12,M_model13,
     *               M_model15,M_model21,M_model22,
     *               n_member,
     *               F_fay,F_n_spring,F_my_spring,
     *               F_mz_spring,i_stat_spring)
C     ------------------------------------------------------------------
      implicit real*8(A-H,O-Z)
      include "submain.h"
      include "submainx.h"
      record / Member_s         / Member
      record / Element_s        / Element
      record / E_model6_real_s  / E_model6_real
      record / M_model11_s      / M_model11
      record / M_model12_s      / M_model12
      record / M_model13_s      / M_model13
      record / M_model15_s      / M_model15
      record / M_model21_s      / M_model21
      record / M_model22_s      / M_model22
      dimension Member(*),Element(*),E_model6_real(*)
      dimension M_model11(*),M_model12(*),M_model15(*)
      dimension M_model21(*),M_model22(*)
      dimension mxtype(100),myytype(4)
                  data mxtype/1,1,1,1,1,1,1,1,1,1,  1,3,1,3,1,1,3,3,3,1,
     3            1,1,1,1,1,1,1,1,1,1,  1,1,1,1,1,1,1,1,1,1,
     5            1,1,1,1,1,1,1,1,1,1,  1,1,1,1,1,1,1,1,1,1,
     7            1,1,1,1,1,1,1,1,1,1,  1,1,1,1,1,1,1,1,1,1,
     9            1,1,1,1,1,1,1,1,1,1,  1,1,1,1,1,1,1,1,1,1/
      data myytype/2,4,3,5/
      real*4    v(6),vv(100)
      real*4 F_fay(5,*),F_n_spring(5,*),F_my_spring(5,*),
     *               F_mz_spring(5,*)
      integer i_stat_spring(5,*)
c-------------------------------------------------★応力 5
      do i=1,n_member
      if(Member(i).element_type.eq.6) then
c-------------------------------------------★Maxwell モデル
c     Maxwell モデルの出力は、次のような特殊な仕様である。
```

```
c     使用する場合は、気をつけること
c     ダンパー変位とダンパー速度を 0.001 倍して送り出す。
c
      istat = 0                                          ! 現在ダミー　塑性状態
      ien= Member(i).n_model_type
      F_n_spring(1,i)=Member(i).stress(1)                ! 軸力
      F_my_spring(1,i)=Member(i).stress(3)               ! ｙ軸せん断力
      F_mz_spring(1,i)=Member(i).stress(2)               ! ｚ軸せん断力
      F_fay(1,i)=E_model6_real(ien).fn*0.001             ! ダンパー軸力
      i_stat_spring(1,i)=istat
      F_n_spring(2,i)=Member(i).stress(1)                !  軸力
      F_my_spring(2,i)=Member(i).stress(3)               ! ｙ軸せん断力
      F_mz_spring(2,i)=Member(i).stress(2)               ! ｚ軸せん断力
      F_fay(2,i)=E_model6_real(ien).u1d*0.001            ! ダンパー速度
      i_stat_spring(2,i)=istat
c-------------------------------------------★3 次元せん断弾塑性モデル
      elseif(Member(i).element_type.ge.2.and.
     *   Member(i).element_type.le.10) then
      istat = Member(i).d_stat(3)                        ! 現在ダミー　塑性状態
      F_n_spring(1,i) =Member(i).stress(1)               ! 軸力
      F_my_spring(1,i)=Member(i).stress(3)               ! ｙ軸せん断力
      F_mz_spring(1,i)=Member(i).stress(2)               ! ｚ軸せん断力
      F_fay(1,i)=0.
      i_stat_spring(1,i)=istat
      F_n_spring(2,i) =Member(i).stress(1)          ! 軸力
      F_my_spring(2,i)=Member(i).stress(3)          ! ｙ軸せん断力
      F_mz_spring(2,i)=Member(i).stress(2)          ! ｚ軸せん断力
      F_fay(2,i)=0.
      i_stat_spring(2,i)=istat
c-------------------------------------------★トラスモデル
      elseif(Member(i).element_type.eq.18.or.
     *       Member(i).element_type.eq.19) then
      i_t=Member(i).element_type
      im=mxtype(i_t)
      ns=myytype(im)
      ie = Member(i).nm_element
      immm= Member(i).n_model_type                  ! モデルタイプ別番号
      do j=1,ns
      istat = 0
      jj=6*(j-1)
      if(j.ge.3) then
      if(j.eq.3) then
      F_n_spring(ns,i) = F_n_spring(2,i)            ! 軸力
      F_my_spring(ns,i)= F_my_spring(2,i)           ! ｙ軸モーメント
      F_mz_spring(ns,i)= F_mz_spring(2,i)           ! ｚ軸モーメント
      i_stat_spring(ns,i)=istat
      F_fay(ns,i)=F_fay(2,i)                        ! 現在ダミー　塑性関数値
      endif
c-------------------------------
      F_n_spring(j-1,i) = Member(i).stress(jj+1)          ! 軸力
      F_my_spring(j-1,i)= Member(i).stress(jj+5)          ! ｙ軸モーメント
      F_mz_spring(j-1,i)= -Member(i).stress(jj+6)         ! ｚ軸モーメント
      i_stat_spring(j-1,i)=istat
      rrxx = 0.                                 ! 現在ダミー　塑性関数値
```

```
      if(Element(ie).ANP.ne.0.)
     *     rrxx =(Member(i).stress(jj+1)/Element(ie).ANP)**2
      rrx=0.
      if(Element(ie).AMPY.ne.0.)
     *      rrx=(Member(i).stress(jj+5)/Element(ie).AMPY)**2
      if(Element(ie).AMPZ.ne.0.)
     *      rrx=rrx+(Member(i).stress(jj+6)/Element(ie).AMPZ)**2
      F_fay(j-1,i)=rrxx+Dsqrt(rrx)
c-------------------------------
      else
      F_n_spring(j,i) = Member(i).stress(jj+1)          ! 軸力
      F_my_spring(j,i)= Member(i).stress(jj+5)          ! ｙ軸モーメント
      F_mz_spring(j,i)= -Member(i).stress(jj+6)         ! ｚ軸モーメント
      i_stat_spring(j,i)=istat
      rrxx=0.                                           ! 現在ダミー　塑性関数値
      if(Element(ie).ANP.ne.0.)
     *      rrxx=(Member(i).stress(jj+1)/Element(ie).ANP)**2
      rrx=0.
      if(Element(ie).AMPY.ne.0.)
     *      rrx=(Member(i).stress(jj+5)/Element(ie).AMPY)**2
      if(Element(ie).AMPZ.ne.0.)
     *      rrx=rrx+(Member(i).stress(jj+6)/Element(ie).AMPZ)**2
      F_fay(j,i)=rrxx+Dsqrt(rrx)
      endif
      enddo
      i_stat_spring(2,i)=Member(i).d_stat(1)
c-----------------------------------------★有限要素モデル
      else
      i_t=Member(i).element_type
      im=mxtype(i_t)
      ns=myytype(im)
      ie = Member(i).nm_element
      immm= Member(i).n_model_type                  ! モデルタイプ別番号
      do j=1,ns
      istat = Member(i).d_stat(j)
      jj=6*(j-1)
      if(j.ge.3) then
      if(j.eq.3)then
      F_n_spring(ns,i) = F_n_spring(2,i)            ! 軸力
      F_my_spring(ns,i)= F_my_spring(2,i)           ! ｙ軸モーメント
      F_mz_spring(ns,i)= F_mz_spring(2,i)           ! ｚ軸モーメント
      i_stat_spring(ns,i)=i_stat_spring(2,i)
      F_fay(ns,i)=F_fay(2,i)                        ! 現在ダミー　塑性関数値
      endif
c-------------------------------
      F_n_spring(j-1,i) = Member(i).stress(jj+1)          ! 軸力
      F_my_spring(j-1,i)= Member(i).stress(jj+5)          ! ｙ軸モーメント
      F_mz_spring(j-1,i)= -Member(i).stress(jj+6)         ! ｚ軸モーメント
      i_stat_spring(j-1,i)=istat
      rrxx = 0.                                           ! 現在ダミー　塑性関数値
      if(Element(ie).ANP.ne.0.)
     *     rrxx =(Member(i).stress(jj+1)/Element(ie).ANP)**2
      rrx=0.
      if(Element(ie).AMPY.ne.0.)
```

```
*       rrx=(Member(i).stress(jj+5)/Element(ie).AMPY)**2
 if(Element(ie).AMPZ.ne.0.)
*       rrx=rrx+(Member(i).stress(jj+6)/Element(ie).AMPZ)**2
 F_fay(j-1,i)=rrxx+Dsqrt(rrx)
c------------------------------
 else
 F_n_spring(j,i) = Member(i).stress(jj+1)          ! 軸力
 F_my_spring(j,i)= Member(i).stress(jj+5)          ! ｙ軸モーメント
 F_mz_spring(j,i)= -Member(i).stress(jj+6)         ! ｚ軸モーメント
 i_stat_spring(j,i)=istat
 rrxx=0.                                           ! 現在ダミー　塑性関数値
 if(Element(ie).ANP.ne.0.)
*       rrxx=(Member(i).stress(jj+1)/Element(ie).ANP)**2
 rrx=0.
 if(Element(ie).AMPY.ne.0.)
*       rrx=(Member(i).stress(jj+5)/Element(ie).AMPY)**2
 if(Element(ie).AMPZ.ne.0.)
*       rrx=rrx+(Member(i).stress(jj+6)/Element(ie).AMPZ)**2
 F_fay(j,i)=rrxx+Dsqrt(rrx)
 endif
 enddo
c-------------------------------------------★
 endif
 enddo
 return
 end
```