第 6 章　マルチスレッドと動的解析

ポイント：マルチスレッドの技術を学ぶ

1．動的ソルバーとマルチスレッド処理
2．動的解析の制御

6.1 はじめに

本章では、動的ソルバーで利用されているマルチスレッド技術について学ぶ。マルチスレッドはウインドウズプログラミングでは必須の技術である。ここでは、マルチスレッド処理と共に動的ソルバーの管理についても説明する。

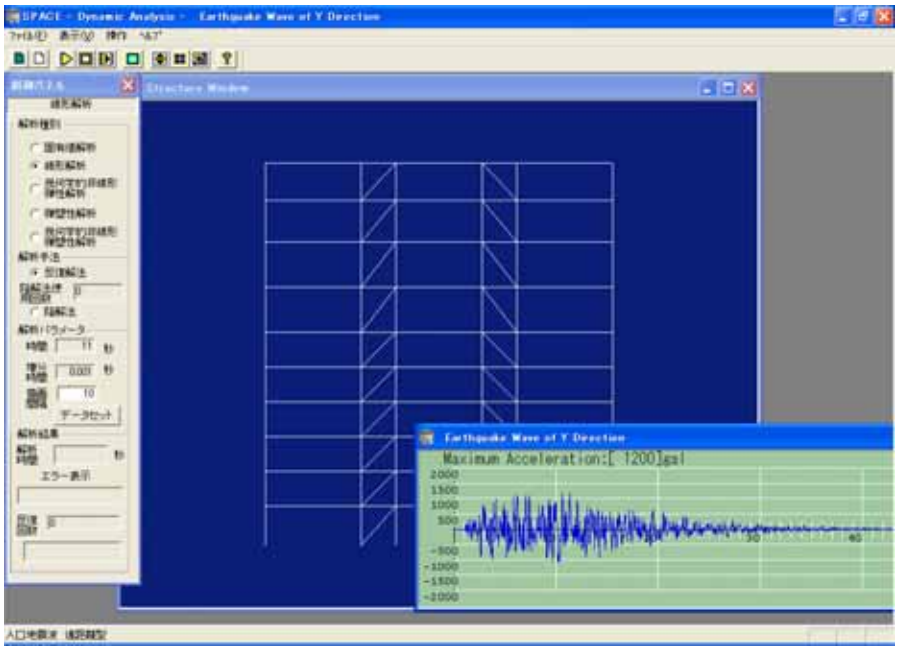

図 6-1 マルチスレッド技術による動的解析

6.2 マルチスレッド処理と動的ソルバー

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。まず、子ウインドウを管理する 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 つであり、これらの関数によってマルチスレッドを使用した動的ソルバーの仕組みが構成される。

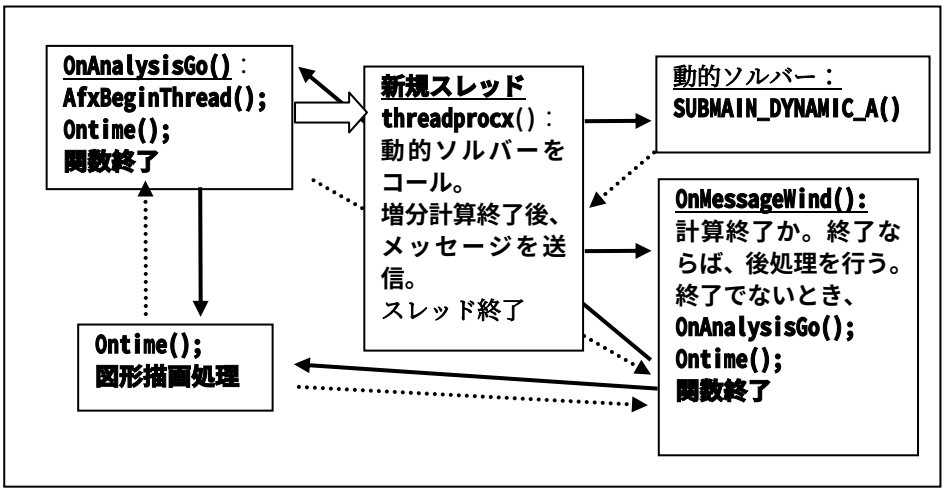

図 6-2 マルチスレッドを用いた動的解析システムのメカニズム

それでは、関連するメッセージマップと、この 5 つのメンバー関数と 2 つの図形処理用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;
}
//    ―――――――――――――――――――――――――――――――――――
//     ●   メッセージに応じて、各ウインドウにおける解析中の図形処理
//    ―――――――――――――――――――――――――――――――――――
LONG CSf3stView::OnMessageWind_G(UINT wParam, LONG lParam)
{
//    ―――――――――――――――――――――――
//     ●   ウインドウコードを取得
//    ―――――――――――――――――――――――
         LONG d_han = 1;
         int fno_yx = lParam;
         int ii = fno_yx-1;
         HWND hWnd = this->GetSafeHwnd();
         if(checkhwnd(hWnd,ii) == FALSE) return (d_han);  // ウインドウが空きとなっている
         if(::IsIconic(hWnd)) return (d_han);             // ウインドウがアイコンとなっている
         CWnd* pWnd = CWnd::FromHandle(hWnd);
         CClientDC dc(pWnd);
         int fno_xy = getwindx((long)fno_yx);
         int fno_xxx = getwindy((long)fno_yx);
         int ihan =0;
//    ―――――――――――――――――――――――
//     ●   ダイアログにデータを渡す
//    ―――――――――――――――――――――――
         if(m_pDlg->GetSafeHwnd() != 0 ){
           m_pDlg->Timeup_D();
         }
//    ―――――――――――――――――――――――
//     ●   ウインドウコードに従って描画関数を呼ぶ
//    ―――――――――――――――――――――――
  switch (fno_xy){
//    ―――――――――――――――――――――――
//     ●   構造物の変形と応力
//    ―――――――――――――――――――――――
  case STRUCT:
         if(F_time_ii <= 0   ){
         disp_mem_persp((CDC*)&dc);
         }else{
         if( fno_xxx == 0) pre_disp_mem_persp( hWnd,ii);
         if( fno_xxx == 1   ) pre3_disp_mem_persp( hWnd,ii);
```

この子ウインドウが制御パネルを管理しているかどうかチェックし、管理している場合は、関数 Timeup_D()をコールしてダイアログに解析時刻などを表示する。

この子ウインドウが表示している図形をチェックし、その図形を更新する。

```
            }
            break;
//    ―――――――――――――――――――――
//      ●　地震波及び節点の変位の履歴図
//    ―――――――――――――――――――――
  case WAVE:
            if(m_dat_struct == 0) sf31wave.set_wave_dat((CDC*)&dc, hWnd, F_Time,ihan);
//           sf31wave.set_wave_dat((CDC*)&dc, hWnd, F_Time,ihan);
            break;
            default:
            break;
}
            d_han = 0;
            return(d_han);
}
//    ―――――――――――――――――――――――――――――――――
//      ●　各ウインドウの図形処理を制御
//    ―――――――――――――――――――――――――――――――――
void CSf3stView::Ontime()
{
            if(fno_yy != 0){          //        fno_yy:現在描画されているウインドウの数
            for(int i = 0; i<fno_yy; i++)
            {
//    ―――――――――――――――――――――
//      ●　各ウインドウのコードを取得
//    ―――――――――――――――――――――
            HWND hWnd;
            int fno_yx = i+1;
            int fno_xy = getwindx((long)fno_yx);
            hWnd =getwindh((long)fno_yx);
            LONG lParam = fno_yx;
//    ―――――――――――――――――――――
//      ●　図形処理を行なうようにメッセージを出す
//    ―――――――――――――――――――――
            if(fno_xy > 0)::SendMessage(hWnd,IDS_MESSAGE_WIND,0,lParam);
            }
      }
}
```

現在、表示されている全ての子ウインドウを更新するために、各ウインドウにメッセージを送信する。

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いこう。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 VC++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。

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

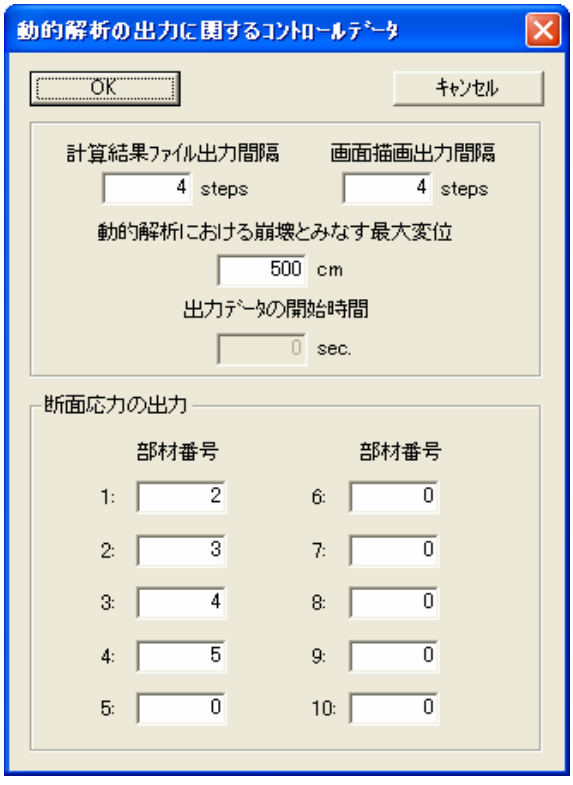

図 6-3 動的解析の出力に関するダイアログ

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、この関数 OnAnalysisInit()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、関数 OnAnalysisInit()の主な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。VC++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、そのスレッ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。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作していることになる。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。

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 6-3 参照)、ここでは１回分の解析と呼ぶ。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。このメッセージを発した後、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、動的ソルバーの一般的な処理内容である。こ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、計算途中で、ユーザー操作に対するシステム応答がフリーズしてしまうことになる。

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 VC++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で先に説明した主な動作以外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、関数実行を制限するためのチェック機構が設けられている。ここでは、スレッドコード thread_on が 1 及び 2 の場合、つまり計算中か停止中の場合、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、これについては先に述べた。

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、この動的領域が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、マニュアル動的解析編の第 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
```

6.3 制御パネルと動的解析制御

SPACE の動的解析システムでは、解析方法の再選択や解析途中での計算停止、再計算あるいは計算終了などの指示を与えることができる。これを可能とする技術は、前節で説明したマルチスレッド処理にある。ここでは、その仕掛けと命令を取り込むためのユーザーインターフェイスについて解説する。動的ソルバーのユーザーインターフェイスは、メニューと操作パネルがある。まず、操作パネルの表示に関連するコードを見てみよう。この操作パネルも CSf3stView クラスの中で、モードレスダイアログとして定義されている。従って、任意の子ウインドウで操作パネルを表示させた後、その子ウインドウを消去すると操作パネルも当然消去される。また、操作パネルは一旦表示するとパネル上から消去することはできない仕様となっている。また、他の子ウインドウから新たな操作パネルを表示することもできない。以下に、操作パネルの表示に関連するコードを示す。ここでは、まず、子ウインドウの描画を管理するクラス sf3stView のヘッダーファイルと操作パネルを制御するクラス CDig_CC_panel.cpp について以下に示す。

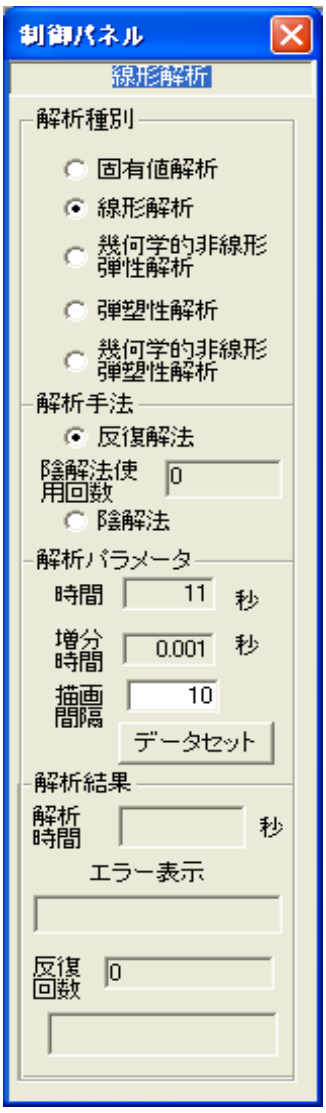

図 6-4 動的ソルバーの制御パネル

```
//――――――――――――――――――――――――――――――――――――
//     ●   sf3stView.h
//     ――――――――――――――――――――――――――――――――――――
//     ●   CSf3stView クラスの宣言およびインターフェイスの定義
//     ――――――――――――――――――――――――――――――――――――
#if !defined(AFX_SF3STVIEW_H__4F12991F_1BF6_11D3_8638_006008D28218__INCLUDED_)
#define AFX_SF3STVIEW_H__4F12991F_1BF6_11D3_8638_006008D28218__INCLUDED_
#include "Dig_pstruct.h"
#include "Dig_mag.h"
#include "Sf31wave.h"
#include "Nodedis.h"
#include "Dig_yesno.h"
#include "Dig_CC_panel.h"
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
#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)
// アトリビュート
public:
          CSf3stDoc*       GetDocument();
          CDig_pstruct    Dlg_pstruct;
          CDig_mag        dig_mag;
          CSf31wave       sf31wave;
          CNodedis        nodedis;
          CDig_CC_panel*  m_pDlg;        // 操作パネルのクラスオブジェクトの定義
// オペレーション
public:
// オーバーライド
         // ClassWizard は仮想関数のオーバーライドを生成します。
         //{{AFX_VIRTUAL(CSf3stView)
         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。
         virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
         protected:
         virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
         virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
         virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
         virtual void OnActivateView(BOOL bActivate, CView* pActivateView, CView* pDeactiveView);
         //}}AFX_VIRTUAL
         afx_msg LONG OnMessageWind(UINT wParam, LONG lParam);
         afx_msg LONG OnMessageWind_G(UINT wParam, LONG lParam);
// インプリメンテーション
          BOOL ButtonDown;              // マウスボタンの押下状態パラメータ
          float m_rr[4][4];             // 透視変換行列
          float m_scrnpsx[3];           // 視点位置（制御用
```

```
            float m_viewpsx[3];          // 視点方向中心位置
            float m_scalepx[3];          // グラフスケール
            float m_scalpsx;             // 透視スケール
            int m_nstep;                 // 解析ステップ数
            float m_base[3];             // 図形中心位置の座標
            int m_fno_xy;                // 画面コード
            int m_dat_struct;            // 構造図オプション
            int m_pst_disp;              // 変位出力オプション
            int m_pst_hinge;             // 塑性ヒンジ表示オプション
            int m_pst_color;             // 部材応力のカラー表示オプション
            int m_pst_graph;             // グラフの表示オプション
            int m_pst_arrow;             // 矢印の表示オプション
            int m_pst_dis;               // 変位、速度などの表示オプション
            float m_pst_ef1;             // 上位表示応力レベル
            float m_pst_ef2;             // 下位表示応力レベル
            int m_pst_options;           // グループ表示オプション
            int m_pst_line[1000];        // グループ番号
            float m_mlimit;              // 表示応力の制限値
            float m_bend;                // 倍率：曲げモーメント
            float m_shear;               // 倍率：せん断力
            float m_mag;                 // 倍率：グラフの円
            float m_stress;              // 倍率：ファイバーの応力
            float m_strain;              // 倍率：ファイバーのひずみ
            float m_arrow;               // 倍率：矢印
            CString m_header;            // ヘッダーストリング
            CString m_footer;            // フッターストリング public:
            virtual ~CSf3stView();       // デストラク関数
#ifdef _DEBUG
           virtual void AssertValid() const;
           virtual void Dump(CDumpContext& dc) const;
#endif
protected:                                                    //
            void OnAnalysisGo();                             // 動的ソルバー（スレッド発生用関数）
            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*);
            void disp_frame_w(CDC*);
            void disp_persp(CDC* );
            void disp_graph(CDC*) ;
            void pre3_disp_mem_persp(HWND ,int);
            void pre4_disp_mem_persp(HWND ,int);
            void pre5_disp_mem_persp(HWND ,int);
            void pre6_disp_mem_persp(HWND ,int);
            void disp2_mem_persp(CDC* );
            void disp3_mem_persp(CDC* );
            void disp5_mem_persp(CDC* );
            void disp6_mem_persp(CDC* );
```

```
            void disp_graph_1(CDC*);
           void disp_graph_5(CDC*);
           void disp_graph_4(CDC*);
           void disp_graph_7(CDC*);
           void disp_load(CDC*  ,int ,int , int );
           void disp_cood(CDC* ,int);
           void disp_graph_8(CDC*);
           void disp_graph_9(CDC*,int);
           void disp_unload(CDC* ,int ,int );
           void disp_ndload(CDC* ,int ,int ) ;
           int lin_options();
           void Ontime();                                  // 図形処理起動用関数
           void wave_data_set(int) ;
           CString getheader(int);                         //ウインドウのタイトル表示処理
           CString getfooter(int,int,int,int,
                             float,float,float,CString); //ステイタスバーへの情報出力処理
           void err_out(int);                              // エラー表示処理関数
// 生成されたメッセージ マップ関数
protected:
           //{{AFX_MSG(CSf3stView)
           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);
           //}}AFX_MSG
           DECLARE_MESSAGE_MAP()
};
//    ――――――――――――――――――――――――――――――――
//      ●   CSf3stView クラスの構築
//    ――――――――――――――――――――――――――――――――
CSf3stView::CSf3stView()
{
//    ――――――――――――――――――――――
//      ●   制御用データの初期設定
//    ――――――――――――――――――――――

//    ――――――――――――――――――――――
//      ●   モードレス  ダイアログの生成
//    ――――――――――――――――――――――
       m_pDlg = new CDig_CC_panel(this);          // 操作パネルの生成
```

```
}
//    ――――――――――――――――――――――――――――――――――
//      ●　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;                  // 操作パネルの消滅
        View_C_panel = 0;               // パネル表示コードを 0 にセット
}
//    ――――――――――――――――――――――――――――――――――
//      ●　解析制御パネル表示処理
//    ――――――――――――――――――――――――――――――――――
void CSf3stView::OnDisplayCcPanel()
{
          if(View_C_panel == 0){
          if(m_pDlg->GetSafeHwnd() == 0 )
          {
                m_pDlg->Create();               // 操作パネルの表示
                View_C_panel = 1;               // パネル表示コードを 1 にセット
          }}
}
```

記述は、クラス sf3stView のヘッダーファイルの全てとそのコンストラクタの一部、デストラクタの一部及びメンバー関数である。この中で、太文字で書かれている以下の文によって、モードレスダイアログの生成と消滅が行われる。

```
CDig_CC_panel*  m_pDlg;             // 操作パネルのクラスオブジェクトの定義
m_pDlg = new CDig_CC_panel(this);   // 操作パネルの生成
delete m_pDlg;                      // 操作パネルの消滅
m_pDlg->Create();                   // 操作パネルの表示
View_C_panel = 1;                   // パネル表示コードを 1 にセット
```

まず、ヘッダーファイルの中で、クラスオブジェクトを定義し、次にコンストラクタの中で、モードレスダイアログとしてパネルが生成される。さらに、メニューかツールチップから制御パネル表示を選択すると、メンバー関数 OnDisplayCcPanel()がコールされ、関数 m_pDlg->Create()によってモードレスダイアログが表示される。さらに、グローバル変数のパネル表示コード View_C_panel を 1 にする。これは、ひとつ以上のパネルが表示されることがないようにするためで、この関数の最上部の if 文でチェックを行っている。最後に、このダイアログはクラス CSf3stView のデストラクタで消去される。

制御パネルに関連する処理を詳しく説明するために、クラス

CDig_CC_panel を以下に示す。このクラスを構成するメンバー関数から、ここでは、以下の 6 つのメンバー関数とメッセージマップについて説明する。

```
CDig_CC_panel::CDig_CC_panel()
void CDig_CC_panel::DoDataExchange()
BOOL CDig_CC_panel::Create()
void CDig_CC_panel::Timeup_D()
void CDig_CC_panel::OnButtonDtset()
void CDig_CC_panel::Set_title()
```

以下に、上記の関数プログラムの内容を示す。

```
//     ――――――――――――――――――――――――――――――――
//     ●   CDig_CC_panel.cpp
//     ――――――――――――――――――――――――――――――――
//     ●   CDig_CC_panel ダイアログクラス
//     ――――――――――――――――――――――――――――――――
#include "stdafx.h"
#include "sf3st.h"
#include "Dig_CC_panel.h"
#include "sf3stdatex.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
CDig_CC_panel::CDig_CC_panel(CView* pView)
{
          m_pView = pView;
          m_edit_analysis = _T("");
          m_edit_err = _T("");
          m_edit_state = _T("");
          m_edit_t = 0.0f;
          m_edit_time = _T("");
          m_edit_zukei = 0;
          m_radio_analysis = 0;
          m_edit_dt = 0.0f;
          m_edit_dt = F_delt;
          m_edit_t = F_all_time;
          m_radio_analysis = F_calnum - 6;
          m_edit_zukei = n_step;
          m_edit_number = 0;
          m_edit_hanpuku = 0;
          m_radio_method = 0;
          Set_title();
}
void CDig_CC_panel::DoDataExchange(CDataExchange* pDX)
{
         CDialog::DoDataExchange(pDX);
```

```
        //{{AFX_DATA_MAP(CDig_CC_panel)
        DDX_Text(pDX, IDC_EDIT_ANALYSIS, m_edit_analysis);
        DDX_Text(pDX, IDC_EDIT_ERR, m_edit_err);
        DDX_Text(pDX, IDC_EDIT_STATE2, m_edit_state);
        DDX_Text(pDX, IDC_EDIT_T, m_edit_t);
        DDX_Radio(pDX, IDC_RADIO_ANALYSIS, m_radio_analysis);
        DDX_Text(pDX, IDC_EDIT_DT2, m_edit_dt);
        DDX_Text(pDX, IDC_EDIT_ZUKEI, m_edit_zukei);
        DDX_Text(pDX, IDC_EDIT_TIME, m_edit_time);
        DDX_Text(pDX, IDC_EDIT_hanpuku, m_edit_hanpuku);
        DDX_Text(pDX, IDC_EDIT_NUMBER, m_edit_number);
        DDX_Radio(pDX, IDC_RADIO_ex, m_radio_method);
        //}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CDig_CC_panel, CDialog)
        //{{AFX_MSG_MAP(CDig_CC_panel)
        ON_BN_CLICKED(IDC_BUTTON_DTSET, OnButtonDtset)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
BOOL CDig_CC_panel::Create()
{
        return CDialog::Create(CDig_CC_panel::IDD);
}
void CDig_CC_panel::Timeup_D()
{
         UpdateData(TRUE);
         char buff[10];
         sprintf(buff,"%8.3f", F_Time);
         m_edit_state = _T("");
         m_edit_time=_T(buff);
         m_edit_hanpuku = n_iterate;
         m_edit_number=nm_iterate;
         UpdateData(FALSE);
}
void CDig_CC_panel::OnButtonDtset()
{
         UpdateData(TRUE);
         n_step=m_edit_zukei;
         F_calnum = 6 + m_radio_analysis;
         m_edit_state = _T("data set ok");
         Set_title();
         numb_method = m_radio_method;
         UpdateData(FALSE);
}
void CDig_CC_panel::Set_title()
{
      char buff[60];
     if(m_radio_analysis == 0) sprintf(buff,"固有値解析");
     if(m_radio_analysis == 1) sprintf(buff,"線形解析");
     if(m_radio_analysis == 2) sprintf(buff,"幾何学的非線形弾性解析");
     if(m_radio_analysis == 3) sprintf(buff,"弾塑性解析");
     if(m_radio_analysis == 4) sprintf(buff,"幾何学的非線形弾塑性解析");
     m_edit_analysis = _T(buff);
}
```

最初の関数はコンストラクタであり、先に示したモードレスダイアログの生成でコールされる。この関数では、メンバー変数の初期化、グローバル変数からメンバー変数へのデータセットを行っている。その中で、関数 Set_title()を実行し、ダイアログの中に解析手法を表示する。関数 Set_title()では、メンバー変数 m_radio_analysis の内容によって、解析種別を変数 buff に書き出し、その内容をダイアログに割り付けた変数 m_edit_analysis にコピーする。これでダイアログ内のテキストエリアに表示されることになる。

次の関数 DoDataExchenge()は、ダイアログには必ず存在する関数で、メンバー変数とダイアログのデータ領域とのデータ交換が行われる。他からの交換要求がある時、システムが自動的にこの関数をコールし、データ交換が行われる。次は、メッセージマップであり、ここでは、２種類のメッセージを受け取る。一つは、解析種別の変更用ラジオボタンであり、他の一つはデータを入力した後のデータセットボタンである。この２つの関数でメッセージを受け取ると、関数 OnButtonDtset()へ制御が移ることになる。

メンバー関数 Create()は、先に示したようにクラス CSf3stView から操作パネルの表示を行うために呼ばれる。関数の内容は、そのまま、ダイアログの関数である Cdialog::Create()を呼んでおり、呼ばれた関数が実際のダイアログを作成し、表示する。

メンバー関数 Timeup_D()は、解析の途中経過を表示する関数である。この関数を呼び出すタイミングは、前節のアニメーションで説明した関数 OnMessageWind_G()で行う。表示する情報は、解析時刻、解析回数、反復回数である。この関数の最後に記述されている関数 UpdateData(FALSE)は、データの内容が更新されるためのトリガーとなっており、メンバー関数 DoDateExchange()をコールして、実際のパネル表示が行われる。

次のメンバー関数 OnButonDtset()では、操作パネル上で各種の解析パラメータを変更した場合、このボタンを押すことで、データがグローバル変数に渡され、解析を制御することができることになる。