実数座標系グラフィックサブルーチン

FRAMES 使用説明書

(第 7.3.1 版)

摂南大学 理工学部 電気電子工学科

2013 年 2 月 18 日

はじめに

計算機シミュレーションとは「物理現象をコンピュータ上で実現する」ことです。遠い宇宙の果てで起こっている大爆発を再現することもできれば，人間の目には見えないようなミクロの現象を起こすこともできます。計算する方程式や用いる数値を人為的に変えれば，現実では起こり得ない現象を起こすことさえも可能なのですが，実際の実験と違って実体が無いので，シミュレーションプログラム開発段階では何を目的に研究しているのか理解できず，ただ言われた通りにプログラムを作っている学生さんが多いと思います。

そもそも，汎用のプログラミング言語，C 言語や Fortran，は標準では「文字」の出力しか扱えないため，結果を「数値」で見るしかありません。しかし，空間的・時間的に変化する流体やプラズマの計算機シミュレーションデータは膨大で，数値だけで現象の全体像をつかむのは至難のわざです。

そこで有用なのがコンピュータグラフィックスです。パソコンやワークステーションはディスプレイ画面に図形を描くハードウェアを持っており，ソフトウェアでこれを動かすことで様々な図形が描けます。筆者の研究室では Linux を入れたパソコンを使ってプログラム開発をしているので，グラフィック環境は X ウィンドウシステムを用いています。X ウィンドウシステムには標準でグラフィックライブラリ (Xlib) が付属していて，これらをプログラムから直接呼び出すことで画面上に様々な図形を描くことができます。

しかし Xlib は，画面座標の指定点間を直線で結ぶとか，指定点を中心とした円を描くとか，文字を描く，とかいった基礎的なものしかありません。このためシミュレーションデータを画面上に表示するには，まず物理データを画面座標に合わせて縮尺する必要があります。図形のサイズを明示するには座標軸を描く必要もあるでしょう。２次元データなら，等高線や鳥瞰図で表現したほうがわかりやすいし，３次元データなら隠線処理を含んだ立体図形表示が欲しくなります。その上，X ウィンドウの性格上，まず図形を描くためのウィンドウを開く処理も必要だし，これを別のウィンドウが隠した時にどういう動作をさせるかという処理も考えておかなければなりません。

ということで，通常は直接 Xlib を使って図形を描くことなどしません。シミュレーションで作成したデータを保存しておいて，これを使って別の図形描画ソフトで描きます。UNIX 系 OS では `gnuplot` というのが有名で，雑誌に解説記事が良く出ています。うちの研究室にはパソコンの Macintosh が数台ありますから，これにデータを転送して便利な市販のグラフィックソフトで描くこともできます。

この別のソフトに依存する方法も悪くはないのですが，その使い方は個別に覚える必要があります。また，そのソフトの「入力」に合わせてシミュレーションデータを「出力」させねばならず，やはりある程度計算機に慣れていることが必要ですし，データ作成→データ処理という２段階を必要としますから手間がかかり，プログラムの作りはじめで結果が合っているかどうかを確かめる段階から実行するにはちょっと面倒です。卒業研究を始めたころは，Fortran の使い方を覚え，シミュレーションコードを作るのがやっとですから，別のソフトの使い方を視野に入れてプログラムする余裕がある人は少ないでしょう。また，こういう市販のソフトは２次元の図形は描けるが３次元はだめとか，３次元表示はきれいだが２次元はもう一つといった「守備範囲」があり，それぞれを使い分けることも必要です。

そこで，Fortran でプログラムを書いている人が，その知識だけで手軽に計算結果を図形表示できるようにしたのが，サブルーチンパッケージ *FRAMES* です。*FRAMES* は Fortran のサブルーチン群であり，少ない手順で計算結果をグラフにすることができるようになっています。`gnuplot` や市販のソフトのように高機能ではありませんが，Fortran プログラムで直接描くことができるため，計算させながら結果をグラフ化することが可能です。筆者が今までシミュレーションデータをグラフ化するに当たって必要だと思われる図形表現は一通りそろっていますから，とりあえずの結果を見るだけならばこれで十分なはずです。*FRAMES* を活用し，計算結果のイメージを具現化しながらプログラム開発することで，計算機シミュレーションを楽しんで下さい。

目　次

第 1 章　*FRAMES* の概要と作成方法

1.1　*FRAMES* の特徴

FRAMES は Fortran プログラムからコールすることで図形描画を行うことができるサブルーチンライブラリであり，以下のような特徴を持っています。

- Fortran のサブルーチン形式なので計算を実行させながら結果を可視化できる。
- X ウィンドウライブラリ (Xlib) のみで動作するので移植性が高い。
- 倍精度実数データをそのままサブルーチンに与えて描画できる。
- 自動スケーリング機能でとりあえず描画することもできれば，座標軸を用途に合わせて描くこともできる。
- 画面切換による簡易アニメーションが可能。
- 描画デバイス切り替え機能を持つ。現在は描画デバイスとして，ディスプレイ (X ウィンドウ)，描画データファイル (*FRAMES* 独自形式)，GIF 形式ファイルがある。これらの描画デバイスは同時に使うこともできるし，途中で切り離したり再度つないだりすることもできる。これにより，ある図形はディスプレイに，別の図形は描画データファイルに，また別の図形は GIF ファイルにという出力制御が可能。
- 描画データファイル出力は独自形式 (plt 形式) であるが，機種に依存しない。つまり，あるコンピュータで作った描画ファイルを使って別のアーキテクチャのコンピュータで再描画することが可能。この出力データは，計算終了後に付属の再描画プログラム (`tplot`) を使って何度でも見直すことができる。また，その際に拡大・縮小・３次元視点変換などの加工をすることが可能。
- 再描画プログラム `tplot` により，描画ファイルの画像出力を PostScript コードに精度良く変換可能。これを使えばプリンタに精度良く出力可能。カラー PostScript にも対応。
- 再描画プログラム `tplot` により，描画ファイルの画像出力を GIF イメージに変換可能。図形が複数あるときには GIF アニメーションにすることも可能。
- 再描画プログラム `tplot` の自動フィルター機能により，描画イメージの連続作成が可能。これを，`pbmplus` などと併用すれば ppm や JPEG 形式での保存が可能。

1.2　*FRAMES* で描画できる図形

FRAMES で描画できる図形は以下の通りです。

- １次元配列を用いた２次元の線図，点図，矢印 (ベクトル)
- ２次元配列を用いた等高線・等高色分布図
- ３角形メッシュでの等高線・等高色分布図
- ２次元ベクトルデータを用いた２次元流線図
- １次元配列を用いた２次元の線図の隠線処理付き重ね描き (波形の時間変化表示など)
- ２次元データを用いた鳥瞰図 (隠線処理付き)
- １次元配列を用いた３次元の線図，点図，矢印 (ベクトル)
- ２次元配列を用いた３次元空間における曲面表示
- ３次元曲面上への別の２次元データを用いた等高色分布図の投影描画
- ３角形メッシュデータを用いた３次元空間における曲面表示
- ２次元または３次元スカラーデータを用いた断面等高線・等高色分布図
- ２次元または３次元ベクトルデータを用いた３次元流線図 (円筒対称座標のデータも可能)
- ３次元スカラーデータを用いた等値面
- 任意の位置への文字表示 (横書きのみ)

1.3 動作条件

FRAMES は，Fortran から呼び出す形式のサブルーチンパッケージで，Fortran77 と C 言語を用いて記述されており，UNIX 上で動作する X ウィンドウライブラリ (Xlib) を用いて描画します。UNIX，X ウィンドウ共に最近のフリー UNIX (Linux，FreeBSD) の隆盛のおかげでポピュラーになりましたが，ワークステーションに付属の商用版も各種あり，少しずつライブラリ内容が異なっています。しかし，幸いなことに X ウィンドウが標準化されているため，ソースレベルでは機種にほとんど依存しないプログラミングが可能です。このため，動作条件は以下の通りです。

1. 拡張された Fortran77 コンパイラが動作する (`g77` 含む)
   ここで,「拡張」とは，`implicit` 文，`do`〜`enddo` 文，`iand` 関数等を指し，最近の Fortran77 では大抵実装されているものです。
2. ANSI C コンパイラが動作する
   C についてはほとんど標準的な仕様しか使っていないので，まずコンパイルエラーが出ることはありません。ただ，非常に古いコンパイラを使っている場合には関数宣言ではねられる可能性があります。
3. X ライブラリがインストールされている
   単に Fortran プログラムから *FRAMES* ルーチンをコールして描画するだけならば Xlib (`libX11.a`) だけで OK です。再描画プログラム `tplot` (付録 C 参照) を作成するためには Athena ウィジェット Toolkit (`libXaw.a`，`libXmu.a`，`libXt.a`，`libXext.a`) が必要です。これは，X の開発ライブラリをインストールすれば大抵付属しているものですが，最近の商用 X ではデフォルトでインストールされていない場合があります。
4. 256 色以上描画可能なカラーのディスプレイ使用が望ましい
   基本 8 色のみで図形を描く場合には低発色数のディスプレイでも描画できます。コンパイルレベルでは描画色数は関係ありません。

FRAMES は Fortran77 で作られていますが，Fortran90 や Fortran95 で作成したプログラムからでも問題なく使うことができます。また，一部注意事項がありますが C 言語からでも使うことができます。C 言語での使い方は付録 E を参照してください。ただし，ライブラリ自体が Fortran で記述されているので Fortran コンパイラとそのライブラリは必ず必要です。

また，ディスクファイルや GIF ファイルを作ることのみを目的とする方のために，ディスプレイ出力を省略したライブラリを作成することもできます。この場合には 3 と 4 は必要ありません (1.4 参照)。

以上のように記述言語レベルはかなり標準的なものに限定していますから，コンパイルエラーが出ることはまずありません。しかし，Fortran と C 言語を混在させているため，機種やコンパイラによってコンパイルオプションやライブラリのリンク方法が異なります。筆者は回りにある全てのマシンで試行錯誤的に動作を確認しました。以下に確認されている機種を示します。

- Linux，mkLinux，LinuxPPC，Linux Alpha
- Mac OS X，FreeBSD
- Cygwin
- AIX3.2，AIX4.1
- HP-UX
- DEC (Compaq) Digital UNIX (Alpha machine)
- Solaris 1.1
- Iris-Crimson
- NEC EWS4800

これら以外の機種でもライブラリさえ適切に指定すれば動作するはずです。詳細は付録 B を参照して下さい。

1.4 基本的な *FRAMES* ライブラリの作成とインストール

確認済みの機種における *FRAMES* の作成とインストールの手順は以下の通りです。

1. パッケージ `frames7.3.1.tar.gz` を展開し，ディレクトリ `frames7.3.1` に移動する。

   ```
   cd frames7.3.1
   ```

2. `make` を実行する。実行の際に OS に応じた *TARGET* を指定する。

   ```
   make TARGET
   ```

   現在用意している *TARGET* は以下の通り。

`linux`	Linux/FreeBSD/Alpha Linux/LinuxPPC/Cygwin 用 (デフォルト)
	Intel Compiler での動作可能 (Version 7.1 以上)(*),
	Compaq Compiler for Linux Alpha も動作可能 (*)
`osx`	Mac OS X+X11 用
	Absoft Fortran，IBM xlf も動作可能 (*)
	Power PC Mac はオプションの変更が必要 (*)
`alpha`	Alpha マシンの Digital UNIX 用
`aix`	AIX 用
`hp`	HP-UX 用
`others`	上記以外で動作が確認されている機種用 (*)

   例えば Mac OS X 用ならば

   ```
   make osx
   ```

   と実行する。デフォルトは `linux` になっているので，Linux/FreeBSD なら *TARGET* は省略できる。

   なお，(*) で示した `g77` 以外のコンパイラを使いたい時や機種を指定したい時など，デフォルトの設定外で利用したい場合には，ディレクトリ `frames7.3.1` 内にあるメーク設定ファイル，`makefile.`*TARGET*，を編集し，OS のバージョン，コンパイラ等に合わせて，コメント (`#`) を付け替えて下さい。

3. `make` の結果，色々メッセージが出てくるが，エラーで止まらず，最後に次のメッセージが出れば成功である。

   ```
   Congratulation !! Frames is now for you.
   Compile sequence is as follows in this machine.

   g77 -O sample.f -o sample libfrX.a -L/usr/X11R6/lib -lX11 -lgdfr
   ∧                                  ∧               ∧
   ```

   ∧ の部分は機種やコンパイラによって異なる。この最後の行は *FRAMES* を使用するときのリンクの方法を示している。

4. `make` に成功すれば，ライブラリ `libfrX.a`，`libgdfr.a` と再描画アプリ `tplot` が `frames7.3.1/src` ディレクトリ中にできる。これらを適当なディレクトリに移動する。

   移動先が，それぞれ`/usr/local/lib` と`/usr/local/bin` で良ければ `root` 権限で

   ```
   make install
   ```

   を実行する。

1.5 X11 に依存しない *FRAMES* ライブラリの作成，その他

make 時に LIB 変数を指定することで，X ライブラリに依存しない *FRAMES* を作成したり，サンプルを作成することが可能です。例えば，X に依存しないライブラリを Mac OS X で作成したい時には

```
make LIB=nox osx
```

を実行します。現バージョンで指定できる LIB 変数は以下の通りです。

all	通常のライブラリと同時にサンプルを作成
nox	X に依存しないライブラリ作成
noxall	X に依存しないライブラリとサンプル作成
forc	C 言語で使用するためのライブラリ作成
noxforc	X に依存しない C 言語で使用するためのライブラリ作成

make のデフォルト機種である Linux/FreeBSD の場合には，LIB=は不要で，例えば，

```
make nox
```

だけで，X に依存しないライブラリができます。

なお，nox を指定した場合には libfrX.a の代わりに libfrNOX.a ができ，tplot は作成されません。libfrNOX.a は，遠隔のマシンで，plt ファイルや GIF ファイルを作成することのみが目的の場合に使用します。

1.6 *FRAMES* の使用方法

FRAMES を使った Fortran プログラムをコンパイル･リンクするときは libfrX.a，libgdfr.a と一緒に libX11.a をリンクします。例えば，gtest.f の時は

```
g77 -O gtest.f -o gtest libfrX.a -lX11 libgdfr.a
```

のように使います。このコマンドは 1.4 で説明したように，make が成功したときに出力されるので記録しておくとよいでしょう。X11 ライブラリのディレクトリ指定が必要な場合は

```
g77 -O gtest.f -o gtest libfrX.a -L/usr/X11R6/lib -lX11 libgdfr.a
```

となります。

X に依存しないライブラリ libfrNOX.a を使う場合には

```
g77 -O gtest.f -o gtest libfrNOX.a libgdfr.a
```

のように-lX11 は不要です。なお，-O オプションは最適化です。必要に応じてその他のオプションも付加してください。

C 言語から使用する時には，付録 E を参照してください。

1.7 *FRAMES* のサブルーチンに与える実数型について

FRAMES は倍精度計算用です。このため，実定数，実変数，実配列はすべて倍精度実数型を用いなければなりません。

第２章　*FRAMES*における図形描画の基本

本章では，*FRAMES* を使って図形描画をする際に共通する基本的概念と描画手順および描画色について述べます。

2.1　描画空間と物理空間

FRAMES はコンピュータで計算して得られたデータを可視化することが目的です。得られる数値は 10^{10}m のような宇宙的スケールから 10^{-9}m のようなナノ領域まで広範囲にわたっています。これに対し，描画装置であるディスプレイやプリンタには，画素数や解像度で決まる座標があり，その数値の範囲には限りがあります。通常この数値は整数で，多くても縦横それぞれ数千程度です。とても実際のデータをそのまま使って図形を描くことはできません。さらに，３次元図形を描く場合には２次元である表示装置に図形を投影する必要があります。また，ディスプレイは通常左上が原点で，右下が縦横とも最大値の左手系であるのに対し，物理空間は右手系が主流です。これらのことは，図形を描画装置に出力するには必ず「実際の座標 (物理座標)」→「描画座標」という座標変換が必要であることを意味します (図 2.1)。

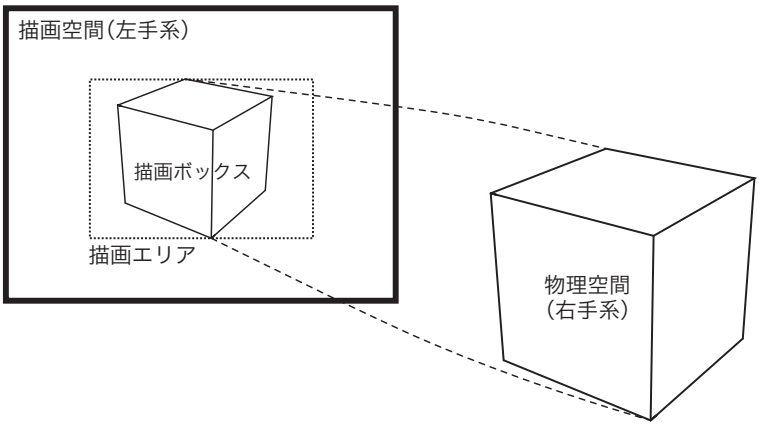


図 2.1　描画空間と物理空間

FRAMES の一番の特徴は，物理空間のデータである数値を直接使って図形を描くことができることです。つまり，上記の座標変換をユーザーがあまり意識しなくても図形を描けます。しかし，出力装置のどのあたりに図形を描くのか，例えばディスプレイいっぱいに描くのか，それとも左半分に描くのか，等の表示デザインはユーザが意識する必要があります。ところが描画空間は出力装置によって異なり，例えばディスプレイとプリンタは描画精度が違うので座標の範囲が違います。このため，出力装置に応じてデザインの部分を変更しなければなりません。さらに GIF や JPEG の様なイメージファイルを作成する場合を考慮すれば，状況に応じて出力装置を増やすことも考えなければなりません。

そこで，*FRAMES* では図形の描画空間を統一化し，各出力装置とは独立して描画する形式を取っています。この仮想的な描画空間を *CANVAS* と呼んでいます。

2.2　仮想描画空間 *CANVAS*

上記のように出力デバイスが多種ある場合には，プログラムからの図形描画と各デバイスへの出力とは切り離して考える方が便利です。*FRAMES* ではこれらの出力デバイスを *CANVAS* という概念で制御します。*CANVAS* とはユーザー定義可能な属性 (名前やサイズなど) を持つ仮想描画空間であり，*FRAMES* で作成した図形は全てこの *CANVAS* に描く形式をとっています。イメージとしては図 2.2 のようになります。さらに *CANVAS* は出力デバイスを接続することができるソケット (*OUTLET* と呼ぶ) を持ち，この *OUTLET* にディスプレイやディスクファイルなどの出力デバイスを接続して実際の描画が完了する仕組みになっています。

図 2.2 では *CANVAS* を 1 個しか描いていませんが，実際には複数個定義して切り替えて使用することができま

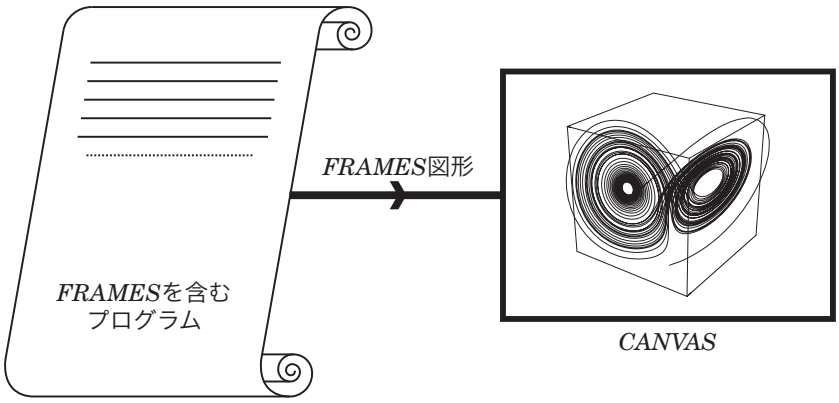


図 2.2　*FRAMES* から *CANVAS* への出力

す。この切り替え機能こそが *CANVAS* の最も重要なポイントで，これを使うことで一つのプログラムから複数の図形を描画する際に出力デバイスを自由に切り替えることができます。例えば，ある図形はディスプレイに表示して別の図形はディスクファイルに出力するとか，数個の異なる連続図形を別々の GIF アニメーションにすることなどが可能です。*CANVAS* の詳細は 3.2 を参照して下さい。

逆に，ディスプレイ描画のみでファイル出力を必要としない場合や，すべての図形を 1 個のディスクファイルに出力するだけ，というように切り替え機能を必要としない場合には，*CANVAS* を意識せずに直接デバイスに出力する形式のプログラミングも可能です。この時，内部的には「デフォルト *CANVAS*」を利用します。詳細は 3.3 を参照して下さい。デフォルト *CANVAS* は `fr_gfile` のような旧バージョンのルーチンとの互換性を保つためにも使用しています。

さて，*CANVAS* に図形表示をするにはどのようなことを考えなければならないでしょうか。少なくとも，図形を「描画空間のどのあたりに描くか」という指定は必要です。*FRAMES* では，これを「描画エリア」と呼び，ユーザーが描画空間座標 (*CANVAS* 座標) を使って指定します。

最も単純には，描画エリアと縮尺された物理空間を一致させればいいのですが，３次元空間では直方体でも視点によって形状が変化するので描画空間座標の指定が複雑です。また２次元空間でも，縦横比が物理座標の x 方向・y 方向のサイズで決まるときには，物理空間をぴったりおさめる描画エリアを指定するのは結構やっかいです。例えば，縦の長さが横の長さの２倍あったとすれば，当然画面上の描画エリアも縦横比を 2:1 にしなければなりません。つまり，描画エリアを決めるために元の物理空間の縦横比を常に意識する必要があります。整数倍の場合にはまだ楽ですが，無理数比であったり，データから決まるような場合にはさらにやっかいになります。

そこで，*FRAMES* では描画エリアと縮尺された物理空間領域とは必ずしも一致せず，まず描画エリアを決め，次にその中に縮尺された物理空間を埋め込む，という形式をとっています。この縮尺された物理領域を「描画ボックス」と呼んでいます。すなわち，*FRAMES* における描画空間は３種類あり，

(1) 数値データに応じた物理領域 (物理空間)
(2) 物理座標ボックスを描画空間に縮小した描画ボックス
(3) 描画ボックスを内包し，ユーザーが指定する描画エリア

となります。(2) の描画ボックスはユーザーが座標で指定する必要はありません。２次元図形の場合には「描画エリア」と「描画ボックス」が一致する場合もあります。１パラメータ図形群では常に一致しています。３次元図形の場合にはほとんど一致することはありません。詳細は各図形描画の節を見てください。また，描画エリアはデフォルトがあるので，指定しなければ，*CANVAS* 中央に適当に位置しています。

2.3 描画手順の概要

FRAMES を用いて図形を描くには次のような手順を用います。

(A) 2次元図形
- (1) 初期化する (fr_ginit)
- (2) *CANVAS* を定義する (fr_gwsize，fr_gfile [，fr_opencanvas]，デフォルトあり)
- (3) 描画空間での描画エリアを指定する (fr_view2d，デフォルトあり)
- (4) 必要ならば描画アスペクト比を指定する (fr_aspect2d，デフォルトあり)
- (5) 物理領域の座標を決め，描画ボックスを決定し，必要に応じて座標軸を描く (fr_frame2d)
- (6) 描画ルーチンを用いて図形を描く (fr_graph2d，fr_contour，等)
- (7) 終了化する (fr_gend)

(B) 1パラメータ図形群
- (1) 初期化する (fr_ginit)
- (2) *CANVAS* を定義する (fr_gwsize，fr_gfile [，fr_opencanvas]，デフォルトあり)
- (3) 描画空間での描画エリアを指定する (fr_pmview，デフォルトあり)
- (4) (不要)
- (5) 物理領域の座標を決め，描画ボックスを決定し，必要に応じて座標軸を描く (fr_pmframe)
- (6) 描画ルーチンを用いて図形を描く (fr_pmgraph，fr_profile，等)
- (7) 終了化する (fr_gend)

(C) 3次元図形
- (1) 初期化する (fr_ginit)
- (2) *CANVAS* を定義する (fr_gwsize，fr_gfile [，fr_opencanvas]，デフォルトあり)
- (3) 描画空間での描画エリアを指定する。(fr_view3d，デフォルトあり)
- (4) 視点の角度，描画アスペクト比を決定する (fr_angle3d，fr_aspect3d)
- (5) 物理領域の座標を決め，描画ボックスを決定し，必要に応じて座標軸を描く (fr_frame3d)
- (6) 描画ルーチンを用いて図形を描く (fr_graph3d，fr_surface，fr_flow3dline，等)
- (7) 終了化する (fr_gend)

ここで，(B) の1パラメータ図形群とは，複数の2次元図形を指定した方向にずらしながら重ね書きすることです。それだけでは表示が煩雑なため，簡単な隠線処理ができるようにしてあり，簡易3次元グラフィックスとも呼べるものです。詳細は 3.5 を参照して下さい。

これらの手順の意味を以下に示します。

(1) fr_ginit による初期化

FRAMES のサブルーチン群を使用するには，最初に必ず一回コールしなければなりません。

(2) *CANVAS* の定義

描画データをディスクに保存したり GIF イメージにする場合は，fr_opencanvas を用いて *CANVAS* を定義し，同時に出力デバイスを指定します。*CANVAS* は複数定義して，描画データに応じて出力先を切替えることが可能です。fr_gifimage の様に出力デバイスを限定した簡易型定義ルーチンも用意してあります。

切り替えが必要でない場合には fr_gfile などを用いて *CANVAS* を意識せずに出力デバイスを指定することも可能です。さらに，ディスプレイ描画のみでよい場合には *CANVAS* の定義は不要です。

なお，*CANVAS* における仮想描画空間のサイズは fr_gwsize を用いて変更することが可能ですが，このような *CANVAS* 属性に関するコールは *CANVAS* 定義の前に行います。

(3) 描画空間での描画エリアの指定

CANVAS の描画空間は2次元長方形で，デフォルトでは横 640・縦 480 のサイズであり，その左上隅が (0,0)，右下隅が (639,479) の左手系座標で表されています。そこで次に描画空間座標を使って図形を描画す

るための描画エリアを指定します。２次元図形および３次元図形描画では長方形領域の座標を指定し，１パラメータ図形群の場合は長方形領域に加えてパラメータにより変化する方向の座標を指定します。

(4) 描画ボックスのアスペクト比を指定する

必要に応じて描画ボックスのアスペクト比を指定します。ここで，アスペクト比とは実際に物理空間を埋め込む領域の各次元方向のサイズ比のことです。例えば，２次元空間なら X の区間と Y の区間の比，３次元空間なら X の区間，Y の区間，Z の区間の比です。

ただし，２次元図形の場合に縦と横の物理量が違ってサイズ比に余り意味がないときには指定する必要はありません。このときは描画ボックスと描画エリアは一致します。１パラメータ図形群の場合には描画ボックスと描画エリアが常に一致するのでアスペクト比の指定はありません。

さらに３次元図形の場合には視点の方向や視点までの距離を指定しておきます。

(5) 物理領域の座標を決め，描画ボックスを決定する

データの数値に応じた物理座標を与えることにより，(3) で指定した描画エリア内に収まるような描画ボックスを決定します。描画ボックスの決定はアスペクト比の指定により以下のようになります。

２次元図形でアスペクト比を指定しなかった場合と１パラメータ図形群の場合には，描画エリアと描画ボックスは一致します。描画エリアに直接目盛りを入れると思えばいいでしょう。この時指定があれば座標軸を描きます。

２次元図形でアスペクト比を指定した場合と３次元図形の場合は，描画エリアをファインダーと見て，それに合わせるように図形をズーミングします。この時指定があれば座標軸を描きます。

(6) 図形を描画する

図形描画のために様々なルーチンが用意されています。

(7) `fr_gend` による終了化

これは，図形描画が不要になったときに必ず一回コールしてください。つまり，*FRAMES* グラフィックルーチン群は `call fr_ginit` と `call fr_gend` ではさむことになります。なお，ディスプレイ出力をする時には，終了化のときにプログラムの実行を一時停止しますので，`stop` 文の直前に `fr_gend` をコールする方が良いと思います。(A.3 参照)

2.4 描画単位 (描画ページ) について

データを可視化する時，一つのプログラムで一つの図形だけを描いて終わるとは限りません。時系列データを元に時々刻々変化する図形を描いたり，密度の図を描いてから温度の図を描き，次に電場の図を描く，というように異なる図形を一つのプログラムで描くことが多いと思います。ディスプレイ出力でこのような異なる図形を描く場合には，まず一つの図形を描いて，見終わったらそれを消し，次の図形を描いて，それも終わったら消してまた描く，という動作をします。

FRAMES は，論理的には仮想描画空間 *CANVAS* に描く，という方式をとっていますが，元来ディスプレイ出力をするプログラムとして発展してきたため，この「画面消去」という動作を一つの描画のチェックポイントとしています。その上で「一枚の図形描画」というのを，白紙の状態から描き始めて描き終わるまでと定義し，これを「描画単位」または「描画ページ」と呼びます。白紙の状態というのは次の動作後を言います。

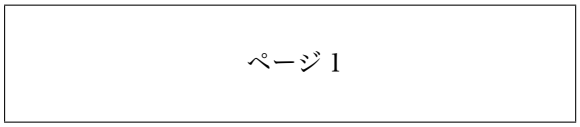
- `fr_ginit` をコールした時
- `fr_gclear` をコールした時
- `fr_canvas` をコールした時

これに対し，描画ページ終了は次の動作後を言います。

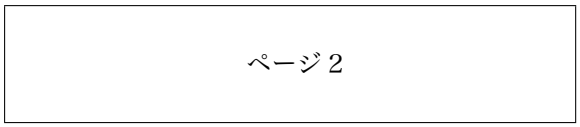
- `fr_gclear` をコールした時
- `fr_canvas` をコールした時
- `fr_gend` をコールした時。

例えば，プログラムの流れは，

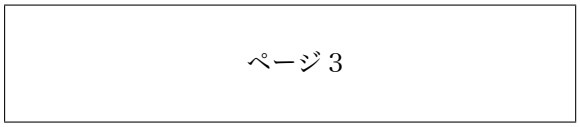
```
call fr_ginit
```



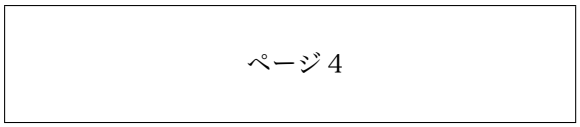

```
call fr_gclear
```




```
call fr_canvas(2)
```




```
call fr_canvas(1)
```




```
call fr_gend
```

というような区切りになり，ページ 1，2，3，4 は別の図形として認識されます。ディスプレイ出力であれば，名前通り fr_gclear は画面を消去します。fr_canvas も画面消去を行います。

描画ページ終了のタイミングは plt ファイル保存時に記録され，tplot による再描画の際に図形切り替えのタイミングとして使われるし，GIF イメージ作成時ならイメージの保存や，アニメーションのイメージ切り替えに使われます。

2.5 描画色の指定について

FRAMES での色指定は 4 種類あります。

(1) 基本 8 色指定

FRAMES では指定を簡単にするため，基本的な色は 8 種類です。これらは，描画ルーチンにおいて 0〜9 の数値で与えます。数値と描画色の対応は ic を描画色指定変数として以下の通りです。

ic=0	白色 (Background，背景色)	ic=1	青色 (Blue)
ic=2	赤色 (Red)	ic=3	紫色 (Magenta)
ic=4	緑色 (Green)	ic=5	空色 (Cyan)
ic=6	黄色 (Yellow)	ic=7	黒色 (White，背景色の補色)
ic=8	黒色 (Black，絶対色)	ic=9	白色 (White，絶対色)

デフォルトでは，ic=0 は白色，ic=7 は黒色ですが，正確には，ic=0 は「背景色」，ic=7 は「背景色の補色」です。ディスプレイ出力や GIF イメージ出力では背景を黒色に切り替えることができます。また，PostScript ファイル出力では白紙への出力を前提としているため，常に背景が白色です。この場合，ic=0 に設定した色は白色に，ic=7 で設定した色は黒色になります。しかし，曲面表示においてメッシュの色を指定したい時など，勝手に切り替えてもらっては困るときのために「絶対黒色」「絶対白色」として，ic=8 と ic=9 を用意してあるわけです。

また，描画色指定の無いルーチン (fr_frame2d の座標軸など) は黒色 (背景色の補色) で描画しますが，内部では上記の ic=7 によって指定した黒色とは区別されています。

(2) HLS **色指定**

等高線ルーチン (fr_contour，fr_contour3d，等) および基礎図形ルーチン (fr_fpline，fr_fpcircle，等) には細かな色の設定ができる HLS 色指定があります。HLS とは Hue (色相)，Lightness (明度)，Saturation (彩度) のことです。

Hue	(色相)	0〜360	角度が単位で，0 度が赤，60 度が黄，120 度が緑， 180 度が空色，240 度が青，300 度が紫である。 360 度で 0 度に戻る。
Lightness	(明度)	0〜1	0 の時に真黒，1 の時に真白である。 0.5 の時にもっとも色がはっきり出る。
Saturation	(彩度)	0〜1	1 の時に一番はっきりと色が出て， 0 に近づくにつれ色彩がなくなる。

デフォルトは，128 色で，虹色の短波長側から長波長側に，即ち，紫色→青色→空色→緑色→黄色→赤色の順で変化するようになっています。この HLS 色の並びは fr_hlsset で変更することができ，最大 256 色までの指定が可能です。設定方法は 3.1.9 の fr_hlsset の項を見てください。

(3) 疑似ライティング指定

３次元曲面を表示するルーチン (fr_surface，fr_trisurface，fr_isosurface) には，指定した曲面に外部から光が当たっているように表現する「疑似ライティング」指定があります。疑似ライティングは 7 種類で，上記の基本 8 色指定の色番号 (1〜7) に対応した色の表面であるように表現します。疑似ライティングを使用する時には，使用するルーチンにおいて色変数を 1〜7 のいずれかに指定して，表示モードで疑似ライティングを選択します。表示モードの指定はそれぞれのルーチンの項を見てください。また，光源の方向は fr_setlight(3.8.13) を用いて，曲面の表面の反射係数などは fr_setsurface(3.8.14) を用いて変更可能です。

(4) 小球色指定

fr_graph3d を用いると３次元空間に多数存在する点を描くことができますが，単なるドットだと遠近が表現できません。そこで，遠近を考慮しながら小さな球を描いて描画する機能があります。この機能を用いる時も，疑似ライティング指定と同じく基本 8 色の色番号 (1〜7) で指定し，それぞれがその色に対応した表面であるかのような濃淡で表現します。但し，これはあくまでも見せかけだけなので，疑似ライティング指定のように光源の方向などを考慮したものではありません。また，球のサイズを変更することはできますが，あまり大きくすると色の変化の粗さが見えます。あくまでも小さな粒子の集合を描画する時だけ利用してください。

なお，小球色指定の効果を使って円を描きたい場合のために fr_graph2d 等でも指定することができます。ただし，遠近の効果は入りませんので，利用者の責任で小球の重なりを考慮する必要があります。

第 3 章　*FRAMES* サブルーチンの使用方法

3.1　描画動作を制御するルーチン

本節では，描画動作を制御するルーチンについて述べます。この中で，fr_ginit，fr_gend は必ずコールする必要があります。

3.1.1　*FRAMES* を初期化する (fr_ginit)

書式　　call fr_ginit
引数　　なし
動作　　*FRAMES* におけるグラフィックルーチン群の初期化を行なう。
注意　☆ 以下のサブルーチンをコールする前に必ず一回コールする必要がある。

3.1.2　*FRAMES* を終了化する (fr_gend)

書式　　call fr_gend
引数　　なし
動作　　*FRAMES* におけるグラフィックルーチン群の終了化を行なう。
ディスプレイ出力をしている時にはプログラムを一時停止する。一時停止を解除すると描画ウィンドウは破棄され，図形は消滅する。
ファイル出力をしている時にはファイルの書き出しを行う。
注意　☆ プログラム終了前に必ず一回コールする必要がある。
☆ ディスプレイ出力時の一時停止状態の解除方法などについては A.2 参照のこと。
☆ ディスプレイ出力時に本ルーチンをコールしないでプログラムが終了すると，描画ウィンドウは破棄され，図形は消滅する (A.3 参照)。

3.1.3　描画空間のサイズと座標の原点を変更する (fr_gwsize)

書式　　call fr_gwsize(isize,jsize,ioff,joff)
引数　　integer isize,jsize,ioff,joff
動作　　描画空間のサイズを isize×jsize にし，その左上隅から (ioff,joff) の座標点を原点とする。
注意　☆ デフォルトは isize=640，jsize=480，ioff=joff=0 である。
☆ 本ルーチンは *CANVAS* の定義前にコールする必要がある。*CANVAS* を複数定義する時には，各 *CANVAS* 定義の直前の指定が有効となる。
☆ *CANVAS* を定義しない時は，全ての描画ルーチンより前にコールする必要があり，本ルーチンを数度呼んだ場合には，図形描画前の最後の指定が有効となる。

3.1.4　描画ページにステップを付与する (fr_gwstep)

書式　　call fr_gwstep(istep)
引数　　integer istep
動作　　描画ステップ数を istep にする。
ここで，描画ステップ数とは描画する図形が istep ページで 1 グループになって，istep ページ毎に同じ種類の図形が来ることを明示するものである。この描画ステップ数は *FRAMES* で描画している間には意味を持たないが，tplot で再描画する時などには istep ページづつジャンプするようになっている。
注意　☆ 本ルーチンは *CANVAS* の定義前にコールする必要がある。*CANVAS* を複数定義する時には，各 *CANVAS* 定義の直前の指定が有効となる。
☆ *CANVAS* を定義しない時は，全ての描画ルーチンより前にコールする必要があり，本ルーチンを数度呼んだ場合には，図形描画前の最後の指定が有効となる。

3.1.5 描画ページを切り替えてディスプレイウィンドウを消去する (fr_gclear)

書式　call fr_gclear

引数　なし

動作　描画ページを切り替える。ディスプレイ出力をしている時には描画ウィンドウの図形を消去する。

注意　☆ 何も描画せずに本ルーチンをコールした場合には無視される。

☆ 本ルーチンを呼んでも fr_view2d，fr_margin2d，fr_pmview，fr_view3d，fr_margin3d の指定は残る。しかし，fr_frame2d，fr_pmframe，fr_frame3d (自動スケールを含む) の指定はクリアされるので，改めて指定する必要がある。

3.1.6 ディスプレイ出力中にプログラムの実行を一時停止する (fr_gpause)

書式　call fr_gpause

引数　なし

動作　ディスプレイ出力中に描画を一時停止する。

注意　☆ 本ルーチンをコールすると，プログラム実行が一時停止し，イベント待ちをする (ファンクションキーを押すとかマウスボタンを押すなど)。停止時のウィンドウ操作は付録 A.2 参照。

☆ ディスプレイ出力をしない時には本ルーチンは無視される。

3.1.7 ディスプレイ出力中にプログラムの実行を一時停止してイベント取得をする (fr_grequest)

書式　call fr_grequest(key,mode)

引数　integer key,mode

動作　ディスプレイ出力中に入力からのイベント取り込みを行なう。また，必要に応じてプログラム実行の一時停止を行う。

mode は以下の指定ができる。

mode=1	プログラムの実行を一時停止してイベント待ちをする イベント取得の結果を key に返す
mode=0	プログラムの実行を一時停止ぜずイベントチェックをする イベント取得の結果を key に返す
mode=-1	プログラムの実行を一時停止してイベント待ちをする イベント取得の結果は返さない

注意　☆ 現在取得できるイベントは以下の通り。

マウス	key=1	左ボタン押下
	key=2	中ボタン押下
	key=3	右ボタン押下
ファンクションキー	key=10+n	ファンクションキー fn 押下

☆ mode=-1 は fr_gpause と動作が同じである。このときの停止時のウィンドウ操作は A.2 参照。

3.1.8 ディスプレイ表示のタイミングを変更する (fr_delayupdate)

書式　call fr_delayupdate(mode)

引数　integer mode

動作　ディスプレイ表示のタイミングの取り方を変更する。

mode=0	表示中のウィンドウに直接描画する
mode=1	内部ウィンドウに描画して，描画ページが切り替わったときに表示する (デフォルト)

mode=0 の時には，描画動作がウィンドウ表示と一致しているので描画する様子が観察できる。

mode=1 の時には，描画しているときにはウィンドウ表示の変更はなく，fr_gclear や fr_gend など，描画ページを切り替えたり終了させるルーチンをコールした時点で瞬時に表示される。よって，時間的に変化する図形を描画ページを切り替えながら次々に描けば，簡易アニメーションになる。

注意　☆ mode=0 の場合でも，描画量が少なかったりグラフィック表示の高速なコンピュータを用いた場合には描画の様子を見るのは難しい。

☆ mode=0 にすると，描画速度は遅くなる。

3.1.9　カラーマップを変更する (fr_hlsset)

書式　`call fr_hlsset(sh,sl,ss,ndiv,mode)`

引数　`real*8 sh(mode+1),sl(mode+1),ss(mode+1)`

`integer ndiv(mode),mode`

動作　mode+1 個の HLS 色指定列 (sh(1),sl(1),ss(1))〜(sh(mode+1),sl(mode+1),ss(mode+1)) に対し，隣り合う mode 個の色区間をそれぞれ ndiv(1)〜ndiv(mode) 分割したカラーマップに変更する。ここで HLS の意味は下表の通りで，それぞれ表に示した範囲の実数値で与える。

sh ： Hue	(色相)	0〜360	角度が単位で，0 度が赤，60 度が黄，120 度が緑，180 度が空色，240 度が青，300 度が紫である。360 度で 0 度に戻る。
sl ： Lightness	(明度)	0〜1	0 の時に真黒，1 の時に真白である。0.5 の時にもっとも色がはっきり出る。
ss ： Saturation	(彩度)	0〜1	1 の時に一番はっきりと色が出て，0 に近づくにつれ色彩がなくなる。

ここで，ndiv(i) で指定する色区間分割数はその両端の指定色，(sh(i)，sl(i)，ss(i)) と (sh(i+1)，sl(i+1)，ss(i+1)) を含む。例えば，ndiv(i)=2 の時は，その区間は (sh(i)，sl(i)，ss(i)) と (sh(i+1)，sl(i+1)，ss(i+1)) の 2 色だけになる。

このことにより隣接する色区間の境界色が両区間で共通となるので，設定される色数の合計は

全設定色数＝ ndiv(1)+ndiv(2)+...+ndiv(mode)-(mode-1)

となる。もし 2 個の色区間を完全に分離したい時には，3 個の色区間 (色区間 1，色区間 2，色区間 3) を用意し，中央の色区間 2 の分割数を ndiv(2)=2 と設定する。

注意　☆ 本ルーチンは色分け等高分布図や基礎図形ルーチンで HLS 色指定するときの番号に対する色の並びを決定するものである。

☆ 全設定色数は 256 以下にしなければならない。

☆ ndiv(i)=0 の区間は ndiv(i)=2 と同じ効果を持つ。すなわち，不連続な色区間の接続に用いることができる。但し，上記の公式を用いて色数の合計を計算する時には ndiv(i)=2 として計算する必要がある。

☆ mode=0 を指定すると，デフォルトの色設定になる。デフォルトは sh(1)=300，sh(2)=0，sl(1)=sl(2)=0.5，ss(1)=ss(2)=1 である。この並びは虹色の短波長側から長波長側に，即ち，紫色→青色→空色→緑色→黄色→赤色の順で変化するようになっている。

☆ ndiv(1)=0 とすると 128 分割する。すなわち，本ルーチンでは 128 分割がデフォルトである。

☆ mode≤1 の場合，引数 ndiv は変数である必要はない。

☆ 本ルーチンで 1 回に指定できるカラーマップは 256 色以下であるが，ディスプレイのカラーモードが True Color の場合には，同じ画面でカラーマップを変更して合計が 256 を越えることになっても同時発色は可能である。

3.1.10 HLS 色設定を呼び出す (fr_hlsrecall)

書式　call fr_hlsrecall(imap)

引数　integer imap

動作　imap 番目の HLS 色設定を呼び出す。
HLS 色設定ルーチン fr_hlsset はその内部にコールした順のシリアル番号を持っている，即ち，最初のコールが 1，次のコールが 2... 等である。本ルーチンを使えば，1 回設定した HLS 色指定ならば fr_hlsset で設定し直さなくてもシリアル番号を指定するだけで色設定ができる。

注意　☆ デフォルトの HLS 色指定に戻すときは imap=0 を与える。
☆ 色指定の保存数には限りがある（1 回の保存量により変化する)。この保存数の上限を越えた場合には色設定ができなくなる。しかし，plt ファイルに保存しているときには保存データに記録が残るため，tplot での再描画時には問題なく表示される。
☆ シリアル番号を確認するために 3.1.11 のサブルーチン fr_hlsquery を用意している。

3.1.11 HLS 色設定シリアル番号を確認する (fr_hlsquery)

書式　call fr_hlsquery(imap)

引数　integer imap

動作　imap に現在の色設定のシリアル番号を返す。
HLS 色設定ルーチン fr_hlsset は内部にコールした順のシリアル番号を持っている，即ち，最初のコールが 1，次のコールが 2... 等である。本ルーチンにより，現在の HLS 色指定シリアル番号を確認することができる。

注意　☆ imap=0 が返ってきたときは，デフォルトの HLS 色指定である。

3.2 *CANVAS* 定義・操作ルーチン

FRAMES による具体的な図形描画は 3.4 以降に記述されている描画ルーチンで行いますが，2.2 で述べたように，描画ルーチンはまず *CANVAS* に描き，次に *CANVAS* が各デバイスに出力する，という構造になっています。*CANVAS* は図形を描く前に定義されている必要があります。本節では *CANVAS* の定義と操作に関するルーチンについて述べます。

CANVAS は描画の出力先をプログラム中で自由に変更できることが特徴ですが,「画面に出力するだけ」とか「一つのファイルに出力するだけ」とかの単純な出力の場合にはあまり詳細を知る必要がありません。そこで単純な出力の場合に *CANVAS* を意識せずとも使えるようなファイル出力ルーチンも用意してあります。これを「デフォルト *CANVAS*」と呼んでいます。とりあえず使ってみようと思われる方は，本節を飛ばして 3.3 のデフォルト *CANVAS* の項に進んで下さい。

CANVAS は各種出力デバイスと接続するための連結装置を持ち，これを *OUTLET* と呼びます。イメージとしては図 3.1 のようになります (実際にはプリンタ直接出力はサポートしていません)。なお,「*OUTLET*」という表現は個々の接続装置を指すだけではなく，*CANVAS* に接続した装置の状態を指すときにも使います。例えば,「デバイス 1 とデバイス 2 の *OUTLET* にする」と書いてあるときは「デバイス 1 とデバイス 2 が *OUTLET* に接続されている状態にする」という意味です。

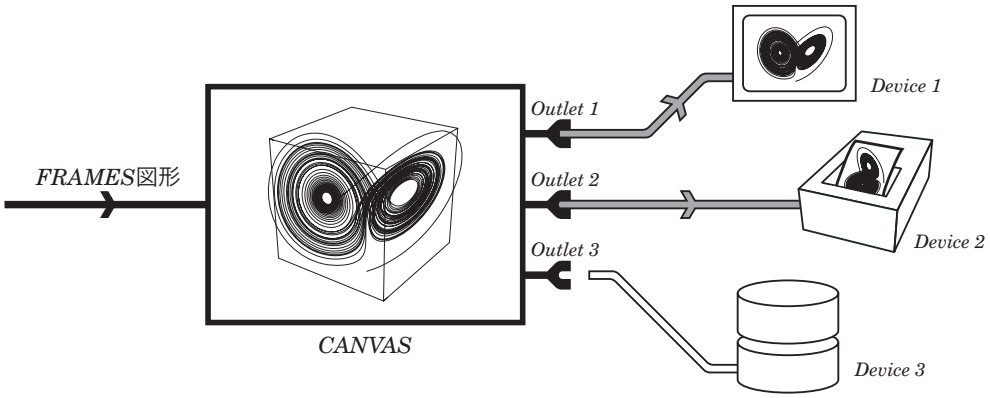


図 3.1 *CANVAS* からデバイスへの出力

OUTLET には図形描画を開始してからでも必要に応じてデバイスを接続したり，切り離したりすることができ，複数のデバイスへ同時に出力することも可能です。現在，接続可能な出力デバイスは以下の通りです。

(A) ディスプレイ (X ウィンドウ)
(B) plt ファイル（`tplot` による再描画用 *FRAMES* 独自形式データファイル）
(C) GIF イメージ
(D) GIF アニメーション

ここで，(B) の plt ファイルというのは *FRAMES* 独自の形式を持つベクトルデータファイルで，精度をあまり落とすことなく描画データ (点の位置など) を物理座標のままでコンパクトに保存します。plt ファイルで保存すれば，プログラム終了後に付属のアプリケーション `tplot` を用いて再描画することができます。再描画の時には描画図形の一部を拡大したり，３次元データなら回転して別の角度で見たり，PostScript プリンタへ出力したりすることができるようになっています。詳細は付録 C を参照して下さい。

これに対し，(C) や (D) の GIF ファイル出力はビットマップなので出力後の図形を拡大するとあらが見えてしまいます。しかしファイル容量が plt ファイルよりもかなり小さいので，大きなデータを用いて図形を描画するときには有効です。また作成した GIF アニメーションをパソコンのプレゼンテーションファイルに直接貼り付けることも可能です。筆者は遠隔の大型計算機を使うことが多いので，シミュレーションデータから GIF ファイルを作って，それをネットワーク経由で取り込んで使っています。

CANVAS は複数定義することができ，それらを切り替えて使用します。ただし一度に描画できる *CANVAS* は一つです。複数の *CANVAS* に同時に出力することはできません。*CANVAS* でユーザーが定義できるものとして,

主なものを以下に示します。

(A) *CANVAS* 番号
(B) 名称
(C) 描画空間のサイズ (縦，横)，描画座標の原点
(D) 描画ステップ数
(E) 出力デバイスの情報

CANVAS のサイズ，座標の原点，描画ステップ数は，その直前までに実行された `fr_gwsize` と `fr_gwstep` の指定で決まります。もしこれらの指定がなければデフォルト値 (サイズは 640×480，原点は (0,0)，ステップは 1) が適用されます。

ディスプレイ表示時には，異なる *CANVAS* に異なるウィンドウが割り当てられるので，いくつかの図形を同時に見ることが可能です。また，plt ファイルや GIF イメージなどは，*CANVAS* を切り替えることで別の名前のファイルに出力することができます。また，ある *CANVAS* では plt ファイルを *OUTLET* にし，別の *CANVAS* では GIF アニメーションを *OUTLET* にしておけば，ある図形は plt ファイルに出力し，別の図形は GIF アニメーションにする，ということも可能です。

CANVAS は汎用性を持たせるように作成してあるため，自由度は高いのですが指定が若干煩雑です。そこで，*OUTLET* をディスプレイとファイル形式一つ (plt 形式または GIF 形式) に固定した簡易型定義ルーチンも用意してあります。

3.2.1 *CANVAS* を定義する (`fr_opencanvas`)

書式　`call fr_opencanvas(ncanvas,name,ioutlet)`

引数
```
integer ncanvas,ioutlet
character name
```

動作　`name` で指定した名前を持つ *CANVAS* を定義する。その際，*CANVAS* に `ncanvas` で指定した番号を付ける。以下の *CANVAS* に関係するルーチンでは `ncanvas` で指定する。これを *CANVAS* 番号と呼ぶ。`ioutlet` により，定義する *CANVAS* の初期 *OUTLET* を指定する。

`ioutlet=  0`	*FRAMES* を使わない (全ての描画ルーチンを無視)
`ioutlet=  1` の位が 1	ディスプレイ出力する (サイズ毎の共有ウィンドウ)
`ioutlet=  1` の位が 2	ディスプレイ出力する (*CANVAS* 別ウィンドウ)
`ioutlet= 10` の位が 1	plt ファイルを作成する
`ioutlet=100` の位が 1	描画 1 ページごとに一つの GIF イメージを作成する
`ioutlet=100` の位が 2	GIF アニメーションを作成する (カラーマップは別)
`ioutlet=100` の位が 3	GIF アニメーションを作成する (カラーマップ共通)

例えば，`ioutlet=11` と指定すれば，ディスプレイに表示しながら plt ファイルが作成される。また，`ioutlet=210` と指定すれば，ディスプレイには出力せず，plt ファイルと GIF アニメーションが作成される。

`ioutlet` を負 (`ioutlet=-211` など) にすると，背景色を黒色にし，文字などの黒色 (色指定 =7) を白色にする白黒反転モードになる。

現在のバージョンでは，*CANVAS* 番号は 1〜100 である。これ以外を指定するとエラーになる。1 番の *CANVAS* はデフォルト *CANVAS* でもある。`fr_ginit` により初期化をした時点での描画先はこのデフォルト *CANVAS* になっている。`fr_gfile` などの *CANVAS* 番号を指定しないルーチンは，内部的にはデフォルト *CANVAS* を使用している (3.3 参照)。

以下の場合には，エラーとなり，エラーメッセージを出力する。

- 定義済みの *CANVAS* 番号を破棄せずに指定したとき。
- 指定した *CANVAS* 番号が 0 以下か 101 以上の時。

注意　☆ name で指定した文字列の拡張子は切り取られる。例えば，name='test.plt' と指定した場合は'test' が *CANVAS* 名として登録される。

☆ 本ルーチンをコールしても現在の出力先 *CANVAS* は変更されない。逆に ncanvas=1 のデフォルト *CANVAS* を本ルーチンで変更すると，その設定で以後描画する。

☆ デフォルト *CANVAS* は破棄しなくても一回に限り本ルーチンを用いて定義可能である。

☆ 一度定義した *CANVAS* の名前は変更できない。即ち，同じ ncanvas を指定した fr_opencanvas をコールすると，２回目はエラーとなる。逆に一度定義した *CANVAS* と同じ名前で別の ncanvas を指定してコールした場合には特にチェックしていないのでエラーにはならないが，動作の保証はない。少なくとも，plt ファイルに関してはうまくいかない。

☆ ioutlet の 1 の位が 1 の場合には，共有ウィンドウを使用してディスプレイ表示する。すなわち，縦横のサイズが同じ *CANVAS* は，共通のウィンドウを利用して表示する。これに対し，ioutlet の 1 の位を 2 にすると，全ての *CANVAS* の表示ウィンドウが別になる。

☆ 描画ページ毎の GIF イメージを作成したとき (ioutlet=100)，イメージファイルの名前はページ順に *CANVAS* 名＋通し番号＋.gif となる。例えば，*CANVAS* 名が'test' の時は，test0001.gif，test0002.gif，... が作成される。

3.2.2　*CANVAS* を未定義にする (fr_closecanvas)

書式　call fr_closecanvas(ncanvas)

引数　integer ncanvas

動作　ncanvas で指定した番号の *CANVAS* を破棄する。破棄した後，ncanvas の番号の *CANVAS* が再定義可能となる。

注意　☆ 基本的に fr_gend により定義された全ての *CANVAS* が破棄されるので，特に同じ *CANVAS* 番号を使いたいとき以外は使用する必要はない。

☆ 現在出力している *CANVAS* を破棄した場合には，*CANVAS* 番号は不定となり，画面描画もファイル出力もストップする。別の *CANVAS* にしたい時には，fr_canvas(3.2.3) を用いて *CANVAS* を指定する必要がある。

☆ 定義されていない *CANVAS* を指定したときには無視される。

3.2.3　描画 *CANVAS* を変更する (fr_canvas)

書式　call fr_canvas(ncanvas)

引数　integer ncanvas

動作　描画ページを切り替えた後，出力先を ncanvas で指定した *CANVAS* にする。その際，それまでに指定した *CANVAS* の属性が使われる。

注意　☆ ディスプレイ出力をしているときには描画ウィンドウの図形を消去する。

☆ 定義されていない canvas 番号を指定したときにはそれ以降の *FRAMES* の出力は無視される。即ち，次に定義された *CANVAS* の指定をするまで画面にもファイルにも出力されない。

☆ 現在描画中の *CANVAS* 番号を指定してコールしても，描画ページは切り替わる。即ち，fr_gclear として働く。

3.2.4　現在出力中の *CANVAS* の *OUTLET* を変更する (fr_outlet)

書式　call fr_outlet(ioutlet)

引数　integer ioutlet

動作　現在描画している *CANVAS* の *OUTLET* を ioutlet に変更する。ここで，ioutlet の指定は以下の通り。

ioutlet= 0	*FRAMES* を使わない (全ての描画ルーチンを無視)
ioutlet= 1 の位が 1	ディスプレイ出力する
ioutlet= 10 の位が 1	plt ファイルを作成する
ioutlet=100 の位が 1	描画 1 ページごとに一つの GIF イメージを作成する
ioutlet=100 の位が 2	GIF アニメーションを作成する (カラーマップは別)
ioutlet=100 の位が 3	GIF アニメーションを作成する (カラーマップ共通)

注意 ☆ 本ルーチンは現在出力中の *CANVAS* に対して作用するので，*CANVAS* 番号の指定はない。
☆ ioutlet の負数指定はできない。すなわち，*CANVAS* の背景色反転モードは *CANVAS* 定義時か GIF オプション指定でしかできない。現バージョンでは，描画途中での背景色の切り替えはサポートしていない。

3.2.5 指定 *CANVAS* の *OUTLET* を変更する (fr_setoutlet)

書式 call fr_setoutlet(ncanvas,ioutlet)

引数 integer ncanvas,ioutlet

動作 ncanvas で指定した番号の *CANVAS* の *OUTLET* を ioutlet に変更する。ここで，ioutlet の指定は以下の通り。

ioutlet= 0	*FRAMES* を使わない (全ての描画ルーチンを無視)
ioutlet= 1 の位が 1	ディスプレイ出力する
ioutlet= 10 の位が 1	plt ファイルを作成する
ioutlet=100 の位が 1	描画 1 ページごとに一つの GIF イメージを作成する
ioutlet=100 の位が 2	GIF アニメーションを作成する (カラーマップは別)
ioutlet=100 の位が 3	GIF アニメーションを作成する (カラーマップ共通)

注意 ☆ 本ルーチンと fr_outlet(3.2.4) とは，*CANVAS* 番号を指定するところが異なるだけである。
☆ ioutlet の負数指定はできない。すなわち，*CANVAS* の背景色反転モードは *CANVAS* 定義時か GIF オプション指定でしかできない。現バージョンでは，描画途中での背景色の切り替えはサポートしていない。

3.2.6 GIF イメージまたはアニメーションの設定を変更する (fr_gifoption)

書式 call fr_gifoption(ioption,idelay,loop)

引数 integer ioption,idelay,loop

動作 現在描画している *CANVAS* の GIF 出力に関する設定を変更する。ioption は次の指定ができる。

ioption= 1 の位が 1	イメージの白黒を反転する（背景色が黒になる）
ioption= 1 の位が 2	イメージをグレイスケールにする（背景色は白）
ioption= 1 の位が 3	イメージをグレイスケールにする（背景色は黒）
ioption= 10 の位が 1	バックグラウンド色を透過にする
ioption= 100 の位が 1	インターレースモードにする
ioption=1000 の位が 1	アニメーションのカラーマップを共通にする

idelay はアニメーションのページ切替のタイミングで $\frac{1}{100}$ 秒を単位とする。ただし，0 を指定したときには，idelay=10 (即ち $\frac{1}{10}$ 秒) のデフォルト値を取る。
loop はループカウントで，loop 回繰り返し動作をした後，ストップするアニメーションとなる。0 を指定したときには，無限に繰り返し動作を続ける。

注意 ☆ 本ルーチンは現在描画中の *CANVAS* に作用する。ただし，さかのぼっての設定はできないので，できる限り *CANVAS* 作成ルーチン (簡易ルーチン含む) の直後に指定する。特にアニメーションの設定変更に関しては 1 ページ出力してしまった後の変更はできない。イメージの透過設定などは 2 ページ目以降で設定しても効力を持つ。

3.2.7 画面描画をするだけの簡易型 *CANVAS* 定義ルーチン (fr_window)

書式　call fr_window(ncanvas,name,ioption)

引数　character name

integer ncanvas,ioption

動作　name で指定した名前を持つ *CANVAS* を定義する。その際，ncanvas で指定した *CANVAS* 番号を付ける。定義された *CANVAS* の *OUTLET* はディスプレイ描画のみである。ioption は次の指定ができる。

ioption=0	背景色が白色
ioption=1	背景色が黒色

注意　☆ 本ルーチンを用いて定義した *CANVAS* は全て個別のウィンドウを持つ。

3.2.8 plt ファイル出力と画面描画をする簡易型 *CANVAS* 定義ルーチン (fr_pltfile)

書式　call fr_pltfile(ncanvas,name,ioption)

引数　character name

integer ncanvas,ioption

動作　name で指定した名前を持つ *CANVAS* を定義する。その際，ncanvas で指定した *CANVAS* 番号を付ける。定義された *CANVAS* の *OUTLET* はディスプレイ描画と plt ファイル出力で，ディスプレイ描画に関しては，ncanvas の正負でオンオフできる。

ncanvas>0	ディスプレイ描画する
ncanvas<0	ディスプレイ描画しない

ioption は次の指定ができる。

ioption=0	背景色が白色
ioption=1	背景色が黒色

注意　☆ 本ルーチンを用いて定義した *CANVAS* でディスプレイに描画する時はサイズ毎の共有ウィンドウを使用する。

3.2.9 GIF イメージ出力と画面描画をする簡易型 *CANVAS* 定義ルーチン (fr_gifimage)

書式　call fr_gifimage(ncanvas,name,ioption)

引数　character name

integer ncanvas,ioption

動作　name で指定した名前を持つ *CANVAS* を定義する。その際，ncanvas で指定した *CANVAS* 番号を付ける。定義された *CANVAS* の *OUTLET* はディスプレイ描画と GIF イメージ出力で，ディスプレイ描画に関しては，ncanvas の正負でオンオフできる。

ncanvas>0	ディスプレイ描画する
ncanvas<0	ディスプレイ描画しない

ioption は次の指定ができる。

ioption= 1 の位が 1	イメージの白黒を反転する（背景色が黒になる）
ioption= 1 の位が 2	イメージをグレイスケールにする（背景色は白）
ioption= 1 の位が 3	イメージをグレイスケールにする（背景色は黒）
ioption= 10 の位が 1	バックグラウンド色を透過にする
ioption=100 の位が 1	インターレースモードにする

注意　☆ 本ルーチンを用いて定義した *CANVAS* でディスプレイに描画する時はサイズ毎の共有ウィンドウを使用する。

☆ イメージファイルの名前はページ順に *CANVAS* 名＋通し番号＋.gif となる。例えば，*CANVAS* 名が'test' の時は，test0001.gif，test0002.gif，... が作成される。

3.2.10 GIF アニメーション出力と画面描画をする簡易型 *CANVAS* 定義ルーチン (fr_gifanimation)

書式　call fr_gifanimation(ncanvas,name,ioption,idelay,loop)

引数　character name
integer ncanvas,ioption,idelay,loop

動作　name で指定した名前を持つ *CANVAS* を定義する。その際，ncanvas で指定した *CANVAS* 番号を付ける。定義された *CANVAS* の *OUTLET* はディスプレイ描画と GIF アニメーションで，ディスプレイ描画に関しては，ncanvas の正負でオンオフできる。

ncanvas>0　ディスプレイ描画する
ncanvas<0　ディスプレイ描画しない

ioption は次の指定ができる。

ioption= 1 の位が 1	イメージの白黒を反転する（背景色が黒になる）
ioption= 1 の位が 2	イメージをグレイスケールにする（背景色は白）
ioption= 1 の位が 3	イメージをグレイスケールにする（背景色は黒）
ioption= 10 の位が 1	バックグラウンド色を透過にする
ioption= 100 の位が 1	インターレースモードにする
ioption=1000 の位が 1	アニメーションのカラーマップを共通にする

idelay はアニメーションのページ切替のタイミングで $\frac{1}{100}$ 秒を単位とする。0 を指定したときには，idelay=10 のデフォルト値を取る。
loop はループカウントで，loop 回繰り返し動作をした後にストップするアニメーションとなる。0 を指定したときには，無限に繰り返し動作を続ける。

注意　☆ 本ルーチンを用いて定義した *CANVAS* でディスプレイに描画する時はサイズ毎の共有ウィンドウを使用する。

3.2.11 plt ファイルに文字列を埋め込み，tplot から参照可能にする (fr_gnotes)

書式　call fr_gnotes(ch)

引数　character ch

動作　文字列 ch を plt ファイルに埋め込む。埋め込んだ文字列は再描画アプリケーション tplot の機能で表示することが可能である。文字列の先頭が '#' の時はそれ以下の文字列を tplot ウィンドウの文字列表示領域に表示する。それ以外の文字列は，Notes ダイアログを用いて表示する。詳細は付録 C.1 参照のこと。

注意　☆ 文字列の長さは 256 までである。それ以上埋め込みたいときは 256 文字ずつに分割すること。
☆ 本ルーチンを用いて埋め込んだ文字列は描画図形には何の影響も与えない。
☆ '#' を用いて表示できるのは 1 行だけである。
☆ 本ルーチンの他にも描画図形に影響を与えることなく plt ファイルにデータを埋め込むルーチンをいくつか用意してある (D.2)。

3.2.12 plt ファイルの最大保存容量を設定する (fr_filemax)

書式　call fr_filemax(maxp)

引数　integer maxp

動作　plt ファイルの容量の最大値を maxp に変更する。単位は kB で与える。maxp=0 の時は，無制限となる。

注意　☆ デフォルトは無制限 (maxp=0) である。

3.3 デフォルト *CANVAS* を用いたファイル操作ルーチン

CANVAS を用いればデバイスへの出力を自由に切り替えることができますが，とりあえず描画図形を保存して `tplot` で見るとか，一連の図形を GIF アニメーションにするだけの場合にはちょっと煩雑です。*FRAMES* の Version 6 以前には *CANVAS* という概念は無く，一つのファイルを指定する `fr_gfile` だけでファイル操作を行っていました。この方法だと *CANVAS* というものを意識する必要がないため指定が簡単です。

そこで，*CANVAS* を導入した現バージョンでも `fr_gfile` を残して，従来のバージョンとの互換性を保つと同時に，*CANVAS* というものを意識しなくても plt ファイルを作成することができるようにしてあります。また，同じ要領で GIF ファイルを作るだけの `fr_gfilegif` も用意してあります。

この機能は，内部的には「デフォルト *CANVAS*」を経由して実現しています。具体的には *CANVAS* 番号 1 の *CANVAS* を *FRAMES* の初期化と同時に接続します。これがデフォルト *CANVAS* です。初期化直後のデフォルト *CANVAS* の *OUTLET* はディスプレイのみです。このため，もし，*CANVAS* の指定も `fr_gfile` によるファイル指定もせずに描画を開始すると，X ウインドウが開いてディスプレイに図形を描きます。これでよければ以下の記述を読むのも省略して，図形描画ルーチンの説明に進んでいただいて結構です。

デフォルト *CANVAS* は一般的な *CANVAS* と異なる性質をいくつか持っています。まず，デフォルト *CANVAS* は「名前なし」の状態で定義されます。通常，*CANVAS* は `fr_opencanvas` などで名前を指定しない限り定義されませんが，デフォルト *CANVAS* は名前がなくても動作します。ただし，名前の指定がない状態での *OUTLET* はディスプレイ描画だけです。次に，デフォルト *CANVAS* は既に内部で定義されているにもかかわらず，名前を付けて定義し直すことができます。このため，ユーザーから見れば，`fr_opencanvas` を用いて「新たに」*CANVAS* 番号 1 を定義することが可能です。

さらに *CANVAS* 番号を意識しなくても，以下に記述されているルーチンを使えば，デフォルト *CANVAS* に名前を付けた上で *OUTLET* に plt ファイルを接続したり，GIF イメージ出力を接続したりすることができます。つまり「*CANVAS*」を意識する必要がありません。出力デバイスの切り替えが不要で，一種類のファイルに出力したいだけの場合には以下のルーチンのみを使えばいいでしょう。

3.3.1 plt ファイル出力と画面描画をする (`fr_gfile`)

書式　`call fr_gfile(name,mode)`

引数　`character name`
`integer mode`

動作　デフォルト *CANVAS* に `name` で指定した名前を持たせ，*OUTLET* をディスプレイ描画と plt ファイル出力にする。`mode` により以下の指定ができる。

`mode=0`	plt ファイルに出力し，ディスプレイ描画をする
`mode=1`	plt ファイルに出力し，ディスプレイ描画はしない
`mode=-1`	ディスプレイ描画はするが plt ファイルに出力しない

注意　☆ 本ルーチンは，従来の `fr_gfile` と互換性を持つ。

3.3.2 GIF 形式ファイル出力と画面描画をする (`fr_gfilegif`)

書式　`call fr_gfilegif(name,mode,ioption)`

引数　`character name`
`integer mode,ioption`

動作　デフォルト *CANVAS* に `name` で指定した名前を持たせ，*OUTLET* をディスプレイ描画と GIF 形式ファイル出力にする。`mode` により以下の指定ができる。

`mode=0`	GIF イメージを作成し，ディスプレイ描画をする
`mode=1`	GIF イメージを作成し，ディスプレイ描画はしない
`mode=2`	GIF アニメーションを作成し，ディスプレイ描画をする
`mode=3`	GIF アニメーションを作成し，ディスプレイ描画はしない
`mode=-1`	ディスプレイ描画はするが GIF ファイルに出力しない

ioption は次の指定ができる。

ioption= 1 の位が 1	イメージの白黒を反転する（背景色が黒になる）
ioption= 1 の位が 2	イメージをグレイスケールにする（背景色は白）
ioption= 1 の位が 3	イメージをグレイスケールにする（背景色は黒）
ioption= 10 の位が 1	バックグラウンド色を透過にする
ioption=100 の位が 1	インターレースモードにする

注意　☆ mode=2 または 3 の時の GIF アニメーションは，delay が 10 で loop が無限大である。細かい設定をしたいときには本ルーチンをコールした後に fr_gifoption(3.2.6) を使って変更する。

3.3.3 デフォルト *CANVAS* の *OUTLET* を変更する (fr_filesw)

書式　call fr_filesw(mode)

引数　integer mode

動作　デフォルト *CANVAS* の *OUTLET* を変更する。mode の設定により以下の変更が行われる。

mode=0	ディスプレイに描画しながらファイルに出力する
mode=1	ファイルに出力するがディスプレイに描画はしない
mode=-1	ディスプレイに描画するがファイルに出力はしない

ここで，‘ファイル’と称しているものは，デフォルト *CANVAS* 出力を fr_gfile で定義した場合には plt ファイル，fr_gfilegif で定義した場合には GIF 形式ファイルである。

注意　☆ 本ルーチンはデフォルト *CANVAS* の定義に fr_gfile を使用した場合には従来のバージョンの fr_filesw と同じ動作をする。fr_gfilegif を使った場合には GIF 形式ファイル作成に対して同様の動作をする。

3.4 ２次元図形描画ルーチン

本節では，２次元図形描画ルーチンについて述べます。２次元図形描画の際は，描画する前に fr_view2d または fr_margin2d を用いて描画 *CANVAS* 上の描画エリアを決め，fr_aspect2d，fr_frame2d を用いて２次元描画ボックスとその物理座標 (データ座標) を決定する必要があります (2.1 参照)。

fr_aspect2d を用いなければ，描画ボックスは描画エリアと一致します。すなわち，fr_frame2d による物理座標の指定は描画エリアに対して実行されます。図形が 1 次元データの折れ線グラフのように横軸と縦軸が独立している場合にはこの形式を使います。これに対し，横軸と縦軸のスケールが同じである２次元空間における粒子の位置を示したり，等高線を描く時には fr_aspect2d を用いて縦横の縮尺を指定します。この時には，描画ボックスは必ずしも描画エリアと一致せず，描画エリアにとれる最大の描画ボックスを生成し，これに物理座標を埋め込みます。

また手軽に描画できるように fr_graph2d などの複数の物理データを使って描画するルーチンを fr_frame2d を使わずにコールすると，与えられたデータの最大値・最小値で座標を指定するようになっています。

なお，以下の記述において，単に「座標」とあるときには，物理座標のことを示します。*CANVAS* 上の座標である「描画座標」と混同しないように注意して下さい。

3.4.1 *CANVAS* における描画エリアを描画座標で決定する (fr_view2d)

書式　call fr_view2d(ixmin,iymin,ixmax,iymax)

引数　integer ixmin,iymin,ixmax,iymax

動作　*CANVAS* における描画エリアを (ixmin,iymin)-(ixmax,iymax) に決定する。ここで (ixmin,iymin) は，それぞれエリアの左上，(ixmax,iymax) はエリアの右下の座標を表す (図 3.2 参照)。

注意　☆ 標準の描画座標は (0,0)-(639,479) なので (ixmin,iymin)，(ixmax,iymax) は，基本的にこの範囲になければならない。

☆ fr_margin2d で指定した場合には本ルーチンは不要である。

☆ 本ルーチンで指定しなければ，それぞれデフォルト値 (ixmin,iymin) = (80,50)，(ixmax,iymax) = (560,370) になる。

☆ 本ルーチンのコール後，それ以前の fr_frame2d の指定はクリアされる。

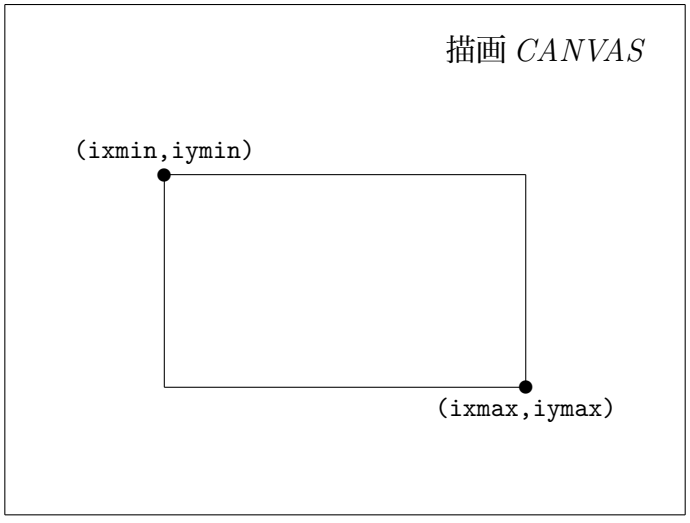


図 3.2　２次元図形描画エリアの指定点

3.4.2 *CANVAS* における描画エリアを上下左右のマージンで決定する (fr_margin2d)

書式　call fr_margin2d(ileft,itop,iright,ibottom)

引数　integer ileft,itop,iright,ibottom

動作　*CANVAS* における描画エリアを，*CANVAS* 全体領域の左マージン ileft，上マージン itop，右マージン iright，下マージン ibottom で決定する。もし，描画 *CANVAS* のサイズが xsize×ysize の場合には，描画座標で (ileft,itop)-(xsize-iright,ysize-ibottom) が描画エリアとなる (図 3.2 参照)。

注意　☆ fr_view2d で指定した場合には本ルーチンは不要である。

☆ 本ルーチンのコール後，それ以前の fr_frame2d の指定はクリアされる。

3.4.3 ２次元描画ボックスのアスペクト比を指定する (fr_aspect2d)

書式　call fr_aspect2d(rtx,rty,mode)

引数　real*8 rtx,rty

integer mode

動作　２次元描画ボックス (物理座標で指定した空間) の表示アスペクト比を rtx:rty に指定する。ここで，mode の指定によりアスペクト比は２通りの意味を持つ。

mode=-1	描画エリアと２次元描画ボックスを一致させる (デフォルト)
mode=0	座標の実スケール比
mode=1	２次元描画ボックスの横：縦の比

注意　☆ 本ルーチンをコールせずに fr_frame2d をコールしたときは mode=-1 の指定となる。すなわち，描画エリアと描画ボックスは一致する。

☆ mode=0 または 1 で本ルーチンを実行すると，fr_view2d または fr_margin2d で指定した描画エリアに含まれて指定した表示アスペクト比を持つ最大の長方形を２次元描画ボックスとする。

☆ mode=0 の時は fr_frame2d で指定した x 方向の物理座標区間を rtx 倍，y 方向の物理座標区間を rty 倍した長方形とする。このため，２次元描画ボックスのサイズは fr_frame2d 実行後に決定される。このモードは実際の２次元形状を表現するときに便利である。

☆ mode=1 の時は２次元描画ボックスの縦横の比が指定したアスペクト比に固定される。このモードは x 方向と y 方向のスケールや次元が異なるときに便利である。ただし３次元の場合と異なり，サイズや次元が違うときにはデフォルト (mode=-1) の描画エリアと描画ボックスを一致させるモードを使用すれば良いので，特に縦横比を指定したいとき以外は使う必要はない。

3.4.4 ２次元描画ボックスの物理座標を指定して座標軸を描く (fr_frame2d)

書式　call fr_frame2d(xmin,xmax,ymin,ymax,mode)

引数　real*8 xmin,xmax,ymin,ymax

integer mode

動作　２次元描画ボックスにおける x 座標の境界を xmin と xmax に，y 座標の境界を ymin と ymax に指定する。この値を用いて２次元描画ボックスを決定する。同時に指定に応じて座標軸を描く。座標軸の種類は mode の値により以下の指定ができる。

mode=0	座標軸を描く	
mode=1	スケールのみを行なう	(枠は描かない)
mode=2	長方形の枠を描く	
mode=3	座標軸を描く	(mode=0 と同じだが，0 の軸は描かない)

なお，mode の 10 の位を 1 にすると (10，11，12 のように)，描画ボックスでクリッピングを行う。すなわち，以下の描画ルーチン使用の際に描画ボックスの外にはみ出した部分は描かない。

注意　☆ fr_setaxis(3.8.8) を併用することで，対数軸を描くことができる。対数で軸を与えるときは，その方向の最小・最大指定は実際の値の対数値を与える (10^5 ならば 5.d0 を与える)。

☆ 対数軸指定の際に，最大値と最小値の差が 1 より小さいときは正常に表示されない。

☆ fr_view2d(3.4.1) の描画エリアの決定では (左上座標，右下座標) の指定であったが，本ルーチンでは (x の境界値，y の境界値) を指定するので順序を間違わないように。

☆ mode=0 の座標軸描画の際，もう一方の軸が 0 点を含むときはそこにも座標軸を描く。これを避けたいときは mode=3 を指定する。

☆ fr_setaxis(3.8.8) を併用することで，座標の数字を描画しないようにすることができる。この場合，目盛りは描画する。

☆ fr_graph2d 等，自動スケール機能を持つルーチンを用いる場合には，本ルーチンを省略できる。

3.4.5 座標軸にタイトルを付ける (fr_xyname)

書式　call fr_xyname(chx,chy)

引数　character chx,chy

動作　fr_frame2d で描いた座標軸にタイトルを付ける。ここで，chx は x 座標のタイトルを表す文字列，chy は y 座標のタイトルを表す文字列である。

注意　☆ 本ルーチンは 2 次元描画ボックスに対して文字の位置を決定するので，fr_frame2d の前でコールした場合はその時点では描かれず，設定だけしてその後の fr_frame2d 実行時にタイトルを描く。fr_graph2d など自動スケール機能を持つルーチンを使用した場合も同様である。

☆ ただし，その設定はページ切り替え時にクリアされる。すなわち，設定した後で図形を描かなければ本ルーチンは無視されることになる。

3.4.6 配列に格納されたデータを用いて２次元図形を描く (fr_graph2d)

書式　call fr_graph2d(x,y,n,ic,mode)

引数　real*8 x(n),y(n)
integer n,ic,mode

動作　n 個の物理座標点 (x(i),y(i)) を用いて画面上にグラフを描く。ic は描画の色である。グラフの種類は mode の値により以下の指定ができる。

mode=1	n 個の点を順に結んだ折れ線グラフを描く
mode=2	n 個の点を描く
mode=3	n 個の小円を描く
mode=-3	n 個の小球を描く
mode=-13	色の異なる n 個の小球を描く (注意参照)
mode=4	n 個の点を番号順に 2 点ずつまとめてその 2 点間全てに線分を描く
mode=5	n 個の点を番号順に 2 点ずつまとめてその 2 点間全てに矢印を描く
mode=6	n 個の点をスプライン補間により順に結んだ曲線を描く データ配列 x(i)，y(i) は，1 番目の点座標，1 番目の微係数，2 番目の点座標，2 番目の微係数... というように，座標とその点の微係数を交互に並べて与える。よって n は点の数の 2 倍になる。
mode=7	n 個のデータの棒グラフを描く デフォルトは y 座標のデータを用いた縦の棒グラフである。

mode の 100 の位 m3 を 1 または 2 にすることで (101，205 のように)，1 次元等間隔データを描くことができる。m3=1 の場合は y(n) に y 座標データ，m3=2 の場合は x(n) に x 座標データを与える。この指定の時には，もう片方の座標配列は要素数 2 で宣言し (y(n) なら x(2)，x(n) なら y(2))，1 番目と 2 番目の要素にデータ要素の最初と最後の座標値をそれぞれ与える。例えば，m3=1 の時には，x(1) に y(1) の x 座標を，x(2) に y(n) の x 座標を与える。等間隔であることがわかっているときにはこの指定を使うことで，メモリの節約ができる。

注意　☆ mode=3 の小円は fr_setcircle(3.8.4) で大きさを変更したり fr_setmark(3.8.5) で多角形に変更することが可能である。

☆ mode=-13 を使う時には色指定変数 ic を点の数と同じ要素数の配列 ic(n) にし，それぞれの小球の色 (1〜7) を格納しておく。なお，7 色全て使っても 256 色以内の収まるので同時発色は可能である。
☆ mode=-3 と mode=-13 の小球は fr_setcircle(3.8.4) で大きさを変更することが可能である。但し，色の分割数が少ないのであまり大きな球にすると粗さが目立つ。
☆ 小球描画は球の重なりを考慮していないので球の間隔が球の直径より短くなると正しい表現ではなくなる。
☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。ただし，小球モードでは機能しないので ic>0 にしなければならない。
☆ mode=4 は複数のばらばらな線分を一度に描くのに使用する。この場合と mode=5 の時，n は線分または矢印の数の 2 倍を指定することになる。
☆ mode=5 の矢印において，矢の先端のサイズは fr_setarrow(3.8.6) で変更することが可能である。
☆ mode=6 のスプライン補間における微係数は点番号 k に対しての座標点 x_k を関数 $x(k)$ と見なし，k で微分した数値を与えるとよい。
☆ mode=6 のスプライン補間では，実際には各点間を 16 分割して曲線を描く。ただし，再描画アプリケーション tplot を使って PostScript 出力をする際には Bezier 曲線を出力するのでプリンタの解像度に合った精度のよい曲線を描くことができる。
☆ mode=7 の棒グラフ指定の場合に fr_setbar(3.8.7) を用いて x 座標のデータを用いた横の棒グラフを描くこともできる。
☆ mode=7 の棒グラフ指定の場合に，ic の色は棒の内部の色を指定する。しかし，内部色だけだと境界がわかりにくいので，境界を同系統の色で縁取りするようになっている。この境界の色は ic の 100 の位 ic1 で変更可能である。例えば，ic=205 にすれば，内部は空色，境界は赤色になる。ただし，ic<0 の HLS 指定の場合には，ic の 1000 の位で指定する。
☆ 本ルーチンを使用する前に，fr_frame2d でスケール指定をしない場合は x(i)，y(i) それぞれの最小値・最大値で自動的にスケールし，座標軸を描く。

3.4.7 点または線分を描く (fr_draw2d)

書式　call fr_draw2d(x,y,ic,mode)

引数　real*8 x,y
integer ic,mode

動作　(x,y) で指定した座標に点または円を描く。または現在点から (x,y) で指定した座標へ線分を描く。ic は点または線分の色である。mode の値により以下の描画指定を行なう。

mode=0	現在点に点を移動する
mode=1	直前に fr_draw2d で指定した点から現在点まで線分を描く
mode=2	現在点に点を描く
mode=3	現在点に小円を描く
mode=-3	現在点に小球を描く

注意　☆ 本ルーチンをコールするとその点の座標は 1 回だけ記憶される。これは mode=1 の指定によりつぎつぎに点を線分でつなぐときに便利である。ただし，一番最初は mode=0，または mode=2 を用いて現在点を移動しておく必要がある。
☆ mode=3 の小円は fr_setcircle(3.8.4) で大きさを変更したり fr_setmark(3.8.5) で多角形に変更することが可能である。
☆ mode=-3 の小球は fr_setcircle(3.8.4) で大きさを変更することが可能である。
☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。ただし，小球モードでは機能しないので ic>0 にしなければならない。
☆ mode=-3 の小球を複数描くとき，小球が重なると正しく描画しない。
☆ 線分を描く時には fr_fpline(3.7.1) を用いることもできる。
☆ 本ルーチンを多用するときは注意が必要である。(付録 A.3 参照)

3.4.8 ２次元ベクトル (矢印) を描く (fr_vector2d)

書式　call fr_vector2d(x,y,dx,dy,ic,mode)

引数　real*8 x,y,dx,dy
integer ic,mode

動作　(x,y) で指定した座標を始点としたベクトルを描く。ic はベクトルの色である。ベクトルの終点 (矢印の先) は (dx,dy) を用いて指定するが，mode の値により以下のような指定方法がある。

mode=0　(dx,dy) を物理座標でのベクトルと見なして描く
即ち，(x+dx,y+dy) を終点とする

mode=1　(dx,dy) を終点とする

mode=2　(dx,dy) を描画座標でのベクトルと見なして描く
即ち，x 座標と y 座標のスケールは物理座標の指定に関係なく 1 : 1 である

mode=3　dx を描画座標の x 座標でスケールし，dy を物理座標での x 対 y の比に
等しくなるようにスケールして，(dx,dy) のベクトルを描く

注意　☆ mode=0 と mode=1 は，ベクトルを fr_frame2d で指定した物理座標値で描くときに使用する。このため，fr_frame2d での座標指定を変更した場合にはベクトルの長さも変わる。

☆ mode=2 は fr_frame2d の指定を変更しても，ベクトルは変わらない。

☆ mode=3 は fr_frame2d の指定を変更しても，ベクトルの x 方向成分は変わらない。y 方向成分は x と y のスケール比が変われば変化する。比が同じならば変化しない。

☆ 矢の先端のサイズは fr_setarrow(3.8.6) で変更することが可能である。

3.4.9 等高線図を描く (fr_contour)

書式　call fr_contour(f,imax,jmax,nd,ic)

引数　real*8 f(imax,jmax)
integer imax,jmax,nd,ic

動作　imax×jmax 個の２次元配列データ f(i,j) を，(i,j) 座標に対する高さを表すデータと見なした等高線図を描く。等高線図は２次元描画ボックスの x 方向，y 方向をそれぞれ imax-1，jmax-1 分割し，その格子点に f(i,j) のデータがあると仮定して描く。

nd は等高線の間隔を決める。即ち，等高線は f(i,j) の最小値と最大値の間を nd 等分した時の等分線になる。なお，fr_setcontour(3.8.11) を併用することにより，高さ指定の等高線なども描画可能である。

ic は等高線の色指定であり，次の選択が可能である。

ic が 0 以上 100 未満のとき
標準の色指定

ic が 100 以上のとき
下 2 桁を第 1 種色，ic/100 を第 2 種色とし，通常は第 1 種色で等高線を描き，5 本毎に第 2 種色を用いる。色指定は標準どおり。

ic=-1　下から n 番目の等高線に HLS モードの ncmax/nc×n 番目の色を付ける。
ここで，nc は等高線の本数，ncmax は HLS 色設定数である。

ic=-2　等高線を用いて描画領域を分割し，下から n 番目の分割領域を n 番目の HLS 色で塗る。

ic=-3　(i,j)-(i+1,j)-(i,j+1)-(i+1,j+1) の座標点で決まる長方形区間を，それぞれの配列要素の平均値を高さと見なしたときの HLS 色で塗る。

ic=-4　(i,j)-(i+1,j)-(i,j+1)-(i+1,j+1) の座標点で決まる長方形区間を f(i,j) を高さと見なしたときの HLS 色で塗る。

ic=-5　(i,j) の座標点を f(i,j) を高さと見なしたときの HLS 色で着色する。

ic<-1 の指定において，データの分割数は HLS 色設定数である。この時，nd の指定は無視される。

注意　☆ 本ルーチンをコールする前に fr_frame2d で与えた物理座標の境界値は描画ボックスの形状決定以外は描画図形に影響しない。しかし，データを plt ファイルに保存する際には fr_frame2d が配列領域の境界座標を指定していると仮定して保存される。このデータを基準にして，再描画の際には表示境界値を変更することで図形の拡大・縮小が可能である。よって，２次元配列の境界の物理座標を用いて fr_frame2d の指定をするのが望ましい。

☆ 描画ボックス全体ではなく，一部の場所に描きたいときには fr_set2dbox(3.4.10) を用いる。

☆ 本ルーチンを使用する前に，fr_frame2d で座標指定をしない場合は，x を (0,imax-1) で，y を (0,jmax-1) で座標を決め，描画ボックスに枠線を描く。

☆ ic=-2 の場合には HLS 設定色数が多いと描画に時間がかかる。これに対し，ic=-3 や ic=-4 の場合には ic=-2 の場合より高速なので大きな２次元配列を用いる場合には有効である。

☆ 本ルーチンでは，ic=-2 の場合に格子点間隔が狭い場合には自動的に ic=-3 または ic=-4 に切り替わるので，通常は ic=-2 を指定すれば良い。

3.4.10　等高線図領域に座標を入れる (fr_set2dbox)

書式　call fr_set2dbox(xmin,xmax,ymin,ymax,mode)

引数　real*8 xmin,xmax,ymin,ymax
integer mode

動作　等高線図を描く際の配列の境界値を (xmin,xmax)(x 方向)，(ymin,ymax)(y 方向) とする。
fr_contour(3.4.9) は，２次元描画ボックスと２次元配列領域が一致していると仮定して等高線図を描くため，図形は２次元ボックスで決まり，座標はそのボックスを決めた fr_frame2d が決める。もし，fr_frame2d と独立に配列境界に座標を入れて，２次元描画ボックスの自由な場所に等高線図を描きたい場合には本ルーチンを使用する。

mode の値により，以下の指定ができる。

mode=0	x，y 共に座標を入れる
mode=1	x は座標を入れない，y は座標を入れる
mode=2	x は座標を入れる，y は座標を入れない
mode=3	x，y 共に座標を入れない

ここで，座標を入れない時は暗黙の座標か fr_frame2d の指定に従う。

注意　☆ 本ルーチンは等高線図を描く直前にコールするのが望ましい。

☆ 座標を入れない指定をした方向の最大値・最小値の変数の値は使用しない。

☆ mode の指定は，座標を入れるか否かを 0・1 とし，yx の順で並べた２進数になっている。

3.4.11　４角形メッシュデータを用いた等高線を描く (fr_sqrcontour)

書式　call fr_sqrcontour(x,y,f,imax,jmax,nd,ic,mc)

引数　real*8 x(imax,jmax),y(imax,jmax),f(imax,jmax)
integer imax,jmax,nd,ic,mc

動作　imax×jmax 個の２次元配列データ f(i,j) を，４角形メッシュ座標 (x(i,j)，y(i,j)) での高さを表すデータと見なした等高線図を描く。

nd は等高線の間隔を決める。即ち，等高線は f(i,j) の最小値と最大値の間を nd 等分した時の等分線になる。

ic は等高線の色指定である。mc は４角形メッシュの色指定である。

注意　☆ 分割方法，等高線の色指定は 3.4.9 の fr_contour 参照のこと。

☆ mc=0 を指定すると，４角形メッシュは描かない。

☆ 本ルーチンを使用する前に，fr_frame2d で座標指定をしない場合は座標データの最小値・最大値で座標を決める。

3.4.12 ３角形メッシュデータを用いた等高線を描く (fr_tricontour)

書式　call fr_tricontour(x,y,f,np,node,ns,nd,ic,mc)

引数　real*8 x(np),y(np),f(np)
integer node(3,ns),np,ns,nd,ic,mc

動作　与えられた２次元領域 (連結していなくても良い) が ns 個の３角形で分割されているとする。各３角形の頂点は総計 np 個の座標点 (x(i),y(i)) から構成されていて，n (=1〜ns) 番目の３角形は頂点番号が node(1,n)，node(2,n)，node(3,n) の座標点 (node(k,n)=1〜np) からなるとする。本ルーチンは，np 個の配列データ f(i) が，座標 (x(i),y(i)) での高さを表すデータと見なした等高線図を描く。

nd は等高線の間隔を決める。即ち，等高線は f(i,j) の最小値と最大値の間を nd 等分した時の等分線になる。

ic は等高線の色指定である。mc は４角形メッシュの色指定である。

注意　☆ 分割方法，等高線の色指定は 3.4.9 の fr_contour 参照のこと。

☆ mc=0 を指定すると，３角形メッシュは描かない。

☆ 本ルーチンを使用する前に，fr_frame2d で座標指定をしない場合は座標データの最小値・最大値で座標を決める。

3.4.13 ２次元流線を描く (fr_flow2dline)

書式　call fr_flow2dline(x,y,nmax,fx,fy,imax,jmax,ic,fmin,nd)

引数　real*8 x(*),y(*),fmin
real*8 fx(imax,jmax),fy(imax,jmax)
integer nmax,imax,jmax,ic,nd

動作　２次元ベクトル場データ (fx(i,j),fy(i,j)) を用いて流線を描く。ベクトル場データは２次元描画ボックスの x 方向，y 方向をそれぞれ imax-1，jmax-1 分割し，その格子点にあると仮定する。

x(n)，y(n) は描画用の補助配列であり，|nmax|以上の要素数の配列を用意しておく必要がある。さらに本ルーチンをコールする前に x(1)，y(1) に流線の開始 x 座標，y 座標をそれぞれ代入しておかなければならない。

ic は流線の色である。fmin は描画領域においてベクトルの長さがこれより小さくなったときに描画を止めるための下限値である。即ち，流線は，

(1) 点データが|nmax|を越えたとき
(2) ベクトルの長さが fmin より小さくなったとき
(3) 流線が指定領域外に出たとき

のいずれかの場合に描画を停止する。

nd は２つの意味がある。一つは描画精度を決めるもので格子間の nd 分の 1 の間隔で点を計算する。即ち，nd の絶対値が大きいほど精度が上がる。今一つは流線の向きを指定する。即ち，nd が正の時にはベクトルの方向に流線を描き，負の時には逆方向に描く。

nmax<0 の指定をすると，|nmax|を最大値として (x(n),y(n)) に流線座標を代入するが，流線は描かずに終了する。また，nmax に代入した座標の数を代入する。このため，nmax<0 を指定するときは nmax は変数でなければならない。

注意　☆ アルゴリズムは線形補間を用いた 4 次の Runge-Kutta 法である。

☆ n≥2 に対する x(n)，y(n) にデータを設定する必要はないが，本ルーチンをコールした後，x(n)，y(n) は破壊される。

☆ nmax<0 の時に nmax に代入される数値は正の数であり，|nmax|以下である。

☆ (3) の「指定領域」は，デフォルトでは描画ボックス全体であるが，次の fr_flow2drgn により，物理座標で指定した部分領域に限定することも可能である。

☆ nmax>0 の時，流線の物理座標は本ルーチンの前にコールした fr_frame2d に依存する。もし，

fr_frame2d を指定していなければ，x=0〜imax-1，y=0〜jmax-1 の座標系となる。

☆ nmax<0 の時，流線の物理座標は本ルーチンの前にコールした fr_frame2d に依存し，fr_frame2d の指定がなければ fr_flow2drgn に依存して，描画ページ切り替え後，どちらか最初にコールされたルーチンで指定した境界値を２次元配列の境界と一致するように決める。もし，どちらも１回も指定していなければ，x=0〜imax-1，y=0〜jmax-1 の座標系となる。

☆ fr_set2drgn (3.8.2) を指定して fx，fy の使用配列領域を限定することもできる。この時，fr_frame2d の指定がなければ，配列領域の最大値・最小値からそれぞれ 1 を引いた値の座標系となる。

☆ 本ルーチンは，直接描画するのではなく，(x(1),y(1)) を初期値とする流線の座標を配列，x(i)，y(i) に代入した後，内部で fr_graph2d をコールして曲線を描画するものである。plt ファイルからデータを抽出するときは，fr_graph2d として抽出をしなければならない (付録 D 参照)。その場合，一般的に描画点の数が nmax より小さいことを考慮すること。

☆ nmax<0 の時は，plt ファイルにデータは保存されない。

3.4.14 ２次元流線を描く際の流線描画領域を限定する (fr_flow2drgn)

書式　call fr_flow2drgn(xrmin,xrmax,yrmin,yrmax)

引数　real*8 xrmin,xrmax,yrmin,yrmax

動作　流線描画を物理座標 (xrmin,yrmin)-(xrmax,yrmax) で指定される長方形領域に限定する。

注意　☆ 本ルーチンで指定した物理座標の領域外に流線が出たときは流線描画を中止する。

☆ 本ルーチンの指定は描画ページ切り替えにより解除される。

☆ fr_flow2dline で nmax<0 を指定するとき，その前で本ルーチンをコールすると配列境界の座標指定となる。もし２回以上コールしたときは，最初のコールは配列境界の座標の指定となり，２回目以降の指定が限定描画領域の指定となる。

3.4.15 ベクトル場の強度配列から等分布点を計算する (fr_flowspace)

書式　call fr_flowspace(f,nmax,x,imax,area,mode)

引数　real*8 f(nmax),x(imax),area

integer nmax,imax,mode

動作　パラメータ（距離，角度など）x で変化する強度値（流速，磁束密度など）$f(x)$ があり，等間隔に並んだパラメータ，$x_1 \ldots x_{\mathtt{nmax}}$ に対する強度値が f(1)，f(2)，...f(nmax) で与えられているとする。この nmax 個のデータ f(n) を元に，imax 分割したパラメータ x の等分布点を計算して配列 x(i) に代入する。ここで，等分布点とは，$\int_{x^{(i)}}^{x^{(i+1)}} |f(x)|dx$ が等しくなるように並んだ点 $x^{(i)}$ のことである。

ただし，両端の位置，x(1)=x_1 と x(imax)=$x_{\mathtt{nmax}}$ は本ルーチンコール前に代入しておく必要がある。mode により３種類のオフセット指定ができる。

mode=0　x_1〜$x_{\mathtt{nmax}}$ を imax-1 区間に分割し，その格子点 $x^{(i)}$ を生成する
x(1)= $x^{(1)} = x_1$，x(imax)= $x^{(\mathtt{imax})} = x_{\mathtt{nmax}}$ なので，x(1) と x(imax) は初期値のままで，x(2)〜x(imax-1) が生成される

mode=1　x_1〜$x_{\mathtt{nmax}}$ を imax-$\frac{1}{2}$ 区間に分割し，その格子中点 $x^{(i+\frac{1}{2})}$ を生成する
x(1)= $x^{(\frac{3}{2})} \geq x_1$，x(imax)= $x^{(\mathtt{imax}+\frac{1}{2})} = x_{\mathtt{nmax}}$ なので，x(imax) は初期値のままで，x(1)〜x(imax-1) が生成される

mode=2　x_1〜$x_{\mathtt{nmax}}$ を imax-$\frac{1}{2}$ 区間に分割し，その格子点 $x^{(i)}$ を生成する
x(1)= $x^{(1)} = x_1$，x(imax)= $x^{(\mathtt{imax})} \leq x_{\mathtt{nmax}}$ なので，x(1) は初期値のままで，x(2)〜x(imax) が生成される

mode=3　x_1〜$x_{\mathtt{nmax}}$ を imax 区間に分割し，その格子中点 $x^{(i+\frac{1}{2})}$ を生成する
x(1)= $x^{(\frac{3}{2})} \geq x_1$，x(imax)= $x^{(\mathtt{imax}+\frac{1}{2})} \leq x_{\mathtt{nmax}}$ なので，x(1)〜x(imax) 全てが生成される

本ルーチンのコール後，area には，$\int_{x_1}^{x_{\mathtt{nmax}}} |f(x)|dx$ が代入される。このため，area は変数でなけ

ればならない。

注意　☆ fr_flow2dline はある開始点から出発する 1 本の流線を描くだけである。多数本の流線を描くときには流線の間隔が流れの強さを表すように決める必要があるが，本ルーチンはそれを補助するために用いる。

☆ 積分は台形公式を使用している。

☆ 本ルーチン終了後，分布の増分値，$\int_{x_i}^{x_{i+1}} |f(x)|dx$ は保持される。imax<0 を指定すると，保持された増分値を用いて分布点 x(i) を計算する。ただし，i≤|imax|である。この時，代入した x(i) の個数を imax に代入する。このため，imax<0 を指定するときは，imax は変数でなければならない。このとき mode によるオフセット指定は上記に準じる。

☆ 増分値が大きいと，i<|imax|の点で x(|imax|) を越えてしまう可能性があるので，imax<0 の指定をしたとき，imax に代入される戻り値 i_{max} は，$i_{max} \leq$|imax|である。また，$i_{max} =$|imax|であっても x(|imax|)$= x_{\mathrm{nmax}}$ とは限らない。即ち，この場合には，x(|imax|) も生成される。

☆ 本ルーチンは *FRAMES* の他のルーチンには依存していないため，単独で使用できる。

3.5 1パラメータ図形群描画ルーチン

本節では，1パラメータ図形群の描画ルーチンについて述べます。1パラメータ図形群とは，ある1変数 (時間とか空間座標とか) に依存して変化する複数の2次元図形のことで，これらをある方向にずらしながら重ねて描画します。重ね描きするにつれて表示が煩雑になるため，簡単な隠線処理ができるようにしてあります。主に2次元データの時間的または空間的変化を描画する場合に使用します。

隠線処理ができることから，簡易3次元グラフィックスとも呼べるものですが，2次元の軸 (x, y) とずらせる方向の軸 (z) で3次元図形を描画すると実際に見える図形と異なります。例えば球は丸く見えず，楕円体に見えます。3次元図形を描きたいときには 3.6 の3次元図形描画ルーチンを利用してください。

3.5.1 *CANVAS* での描画エリアを決定する (fr_pmview)

書式　`call fr_pmview(ixmin,iymin,ixmax,iymax,izx,izy)`

引数　`integer ixmin,iymin,ixmax,iymax,izx,izy`

動作　*CANVAS* における1パラメータ図形群の描画領域 (描画エリア) を決定する。ここで，パラメーターを z としたとき，z が最小値の時の描画エリアを2次元図形と同じく (ixmin,iymin)-(ixmax,iymax) で決定し，(ixmax,iymax) の点の z が最大値の時の移動先座標を (izx,izy) とする (図 3.3 参照)。1パラメータ図形群の描画ボックスは常に描画エリアと一致するので，本ルーチンの指定がそのまま描画ボックスを決定する。

注意　☆ 標準の *CANVAS* 座標は (0,0)-(639,479) なので (ixmin,iymin)，(ixmax,iymax)，(izx,izy) は基本的にこの範囲になければならない。ただし，図形の一部のみ表示すればよいのであれば，範囲外の指定をしても差しつかえない。

☆ 本ルーチンで指定しなければ，それぞれデフォルト値 (ixmin,iymin) = (70,150)，(ixmax,iymax) = (440,370)，(izx,izy) = (560,250) になる。

☆ 本ルーチンのコール後，それ以前の fr_pmframe の指定はクリアされる。

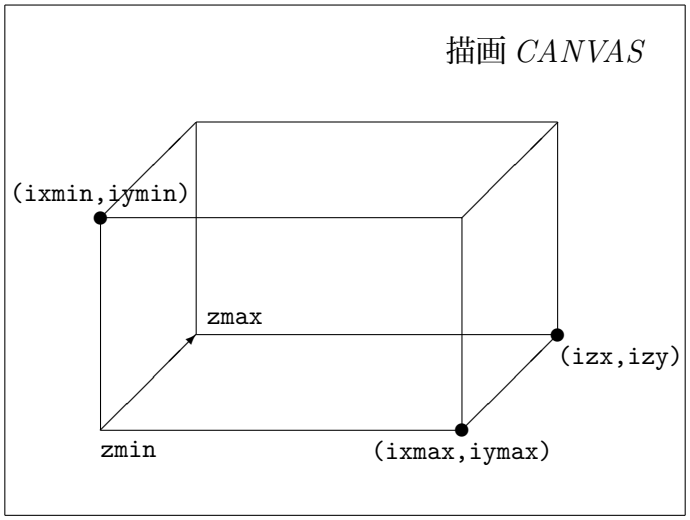


図 3.3　1パラメータ図形群の描画エリア (描画ボックス) の指定点

3.5.2 描画ボックスの物理座標を指定し座標軸を描く (fr_pmframe)

書式　call fr_pmframe(xmin,xmax,ymin,ymax,zmin,zmax,mode)

引数　real*8 xmin,xmax,ymin,ymax,zmin,zmax
integer mode

動作　描画ボックスに対し x 座標の境界を xmin と xmax に，y 座標の境界を ymin と ymax に指定する。また，パラメータ z 座標の境界を zmin と zmax にする。ここで，描画ボックスの手前の面が zmin であり，向こう側の面が zmax であるとする (図 3.3 参照)。
指定に応じて座標軸を描く。枠の種類は mode の値により以下の指定ができる。

mode=0	座標軸のみを描く
mode=1	スケールのみを行なう (枠は描かない)
mode=2	直方体の枠を描く
mode=3	座標軸と直方体の枠を描く

注意　☆ fr_setaxis(3.8.8) を併用することで，x 方向と y 方向は対数軸を描くことができる。対数で軸を与えるときは，その方向の最小・最大指定は実際の値の対数値を与える (10^5 ならば 5.d0 を与える)。
☆ z 方向は対数軸にはできない。
☆ 対数軸指定の際に，最大値と最小値の差が 1 より小さいときは正常に表示されない。
☆ fr_setaxis(3.8.8) を併用することで，座標の数字を描画しないようにすることができる。この場合，目盛りは描画する。
☆ fr_profile(3.5.8) には自動スケール機能があるので，本ルーチンは省略できる。
☆ クリッピングの指定はない。

3.5.3 座標軸にタイトルを付ける (fr_xyzname)

書式　call fr_xyzname(chx,chy,chz)

引数　character chx,chy,chz

動作　fr_pmframe で描いた座標軸にタイトルを付ける。ここで，chx は x 座標のタイトルを表す文字列，chy は y 座標のタイトルを表す文字列，chz はパラメータ z 座標のタイトルを表す文字列である。

注意　☆ 本ルーチンは描画ボックスに対して文字の位置を決定するので，fr_pmframe の前でコールした場合はその時点では描かれず，設定だけしてその後の fr_pmframe 実行時にタイトルを描く。fr_profile など自動スケール機能を持つルーチンを使用した場合も同様である。
☆ ただし，その設定はページ切り替え時にクリアされる。すなわち，設定した後で図形を描かなければ本ルーチンは無視されることになる。
☆ chx と chy は fr_xyname(3.4.5) と同じ位置に描く。chz はパラメータ座標軸に沿って描く。

3.5.4 隠線消去指定を行なう (fr_hclear)

書式　call fr_hclear(mode)

引数　integer mode

動作　隠線消去モードの指定を行なう。mode の値により以下の指定ができる。

mode=0	隠線消去しない
mode=1	図の下方への飛び出しを許す隠線消去 (通常の方法)
mode=2	図の下方への飛び出しを許さない隠線消去

注意　☆ fr_pmgraph(3.5.5) 等において隠線処理指定をしたときは，最大最小法を用いた隠線処理を行っている。これは描画を手前から順に行なって，手前の図形で遮られた場所を計算し，隠線かそうでないかの判断を行なうものである。下方への飛び出しは，手前の図形を線図と考えるか，それともその線で上方を切り取られた面と考えるかにより異なる。前者は飛び出しを許す方法であり，後者は許さない方法である。
☆ デフォルトでは mode=0 になっている。即ち隠線消去はしない。

3.5.5 配列に格納されたデータを用いて図形を描く (fr_pmgraph)

書式　call fr_pmgraph(x,y,z,n,ic,mode)

引数　real*8 x(n),y(n),z　　mode の 10 の位が 0 の場合
real*8 x(n),y(n),z(n)　　mode の 10 の位が 9 の場合
integer n,ic,mode

動作　mode の 10 の位が 0 の場合はパラメータが z のときの n 個のデータ点 (x(i),y(i)) を用いて画面上にグラフを描く。
mode の 10 の位が 9 の場合は x,y と同時にパラメータ z も変化すると考える。即ち，n 個のデータ点 (x(i),y(i),z(i)) を用いて画面上にグラフを描く。ic は描画の色である。グラフの種類は mode の 1 の位の値 m1 により以下の指定ができる。

- m1=1　n 個の点を順に結んだ折れ線グラフを描く
- m1=2　n 個の点を描く
- m1=3　n 個の小円を描く
- m1=4　n 個の点を番号順に 2 点ずつまとめてその 2 点間全てに線分を描く
- m1=5　n 個の点を番号順に 2 点ずつまとめてその 2 点間全てに矢印を描く
- m1=6　n 個の点をスプライン補間により順に結んだ曲線を描く
  データ配列 x(i)，y(i) は，1 番目の点座標，1 番目の微係数，2 番目の点座標，2 番目の微係数... というように，座標とその点の微係数を交互に並べて与える。よって n は点の数の 2 倍になる。
- m1=7　n 個のデータの棒グラフを描く
  デフォルトは y 座標のデータを用いた縦の棒グラフである。
- mode<0，m1=1 の場合
  n 個の点を隠線処理をしながら順に結んだ折れ線グラフを描く

mode の 100 の位 m3 を 1 または 2 にすることで (112，205 のように)，x 方向か y 方向の 1 次元等間隔データを描くことができる。m3=1 の場合は y(n) に y 座標データ，m3=2 の場合は x(n) に x 座標データを与える。この指定の時には，もう片方の座標配列は要素数 2 で宣言し (y(n) なら x(2)，x(n) なら y(2))，1 番目と 2 番目の要素にデータ要素の最初と最後の座標値をそれぞれ与える。例えば，m3=1 の時には，x(1) に y(1) の x 座標を，x(2) に y(n) の x 座標を与える。等間隔であることがわかっているときにはこの指定を使うことで，メモリの節約ができる。

注意　☆ mode<0 で隠線処理をする場合には以下の 2 点が必要である。
1. あらかじめ fr_hclear を用いて隠線消去指定を行なう。
2. 図を手前から順に描く。即ち，パラメータ z が zmin → zmax となるような順序で描く。

☆ m1=3 の小円は fr_setcircle(3.8.4) で大きさを変更したり fr_setmark(3.8.5) で多角形に変更することが可能である。

☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。

☆ m1=4 は複数のばらばらな線分を一度に描くのに使用する。この場合と m1=5 の時，n は線分または矢印の数の 2 倍を指定することになる。

☆ m1=5 の矢印において，矢の先端のサイズは fr_setarrow(3.8.6) で変更することが可能である。

☆ m1=6 の場合，z 方向はスプライン係数は計算しないので微係数は不要。よって，mode の 10 の位が 9 の場合には，配列 z の要素数が x，y の要素数の半分でなければならない。

☆ スプライン補間における微係数は点番号 k に対しての座標点 x_k を関数 $x(k)$ と見なし，k で微分したオーダーの数値を与えるとよい。

☆ m1=6 のスプライン補間では，各点間を 16 分割して曲線を描く。ただし，再描画アプリケーション tplot を使って postscript 出力をする際には bezier 曲線を出力するのでプリンタの解像度に合った精度のよい曲線を描くことができる。

☆ m1=7 の棒グラフ指定の場合に fr_setbar(3.8.7) を用いて x 座標のデータを用いた横の棒グラフを

描くこともできる。

☆ m1=7 の棒グラフ指定の場合に，ic の色は棒の内部の色を指定する。しかし，内部色だけだと境界がわかりにくいので，境界を同系統の色で縁取りするようになっている。この境界の色は ic の 100 の位 ic1 で変更可能である。例えば，ic=205 にすれば，内部は空色，境界は赤色になる。ただし，ic<0 の HLS 指定の場合には，ic の 1000 の位で指定する。

☆ m1=7 の棒グラフ指定の場合には隠線処理の機能がないが，パラメータ z が zmax → zmin となるように遠い方から描けば，隠線処理されているように描画される。

☆ 本ルーチンでは最大最小法を用いて隠線処理を行なうため，1 価関数でないものはうまく表示しない。

☆ 本ルーチンは fr_graph2d と異なり自動スケール機能はない。

3.5.6 パラメータを設定する (fr_pmset)

書式　call fr_pmset(z)

引数　real*8 z

動作　パラメータを z にする。

注意　☆ 本ルーチンは fr_pmdraw(3.5.7) で図形を描く際にパラメータ z を設定をするために用いる。

3.5.7 隠線処理をしながら点または線分を描く (fr_pmdraw)

書式　call fr_pmdraw(x,y,ic,mode)

引数　real*8 x,y
integer ic,mode

動作　(x,y) で指定した座標に点を描く。または現在点から (x,y) で指定した座標へ線分を描く。この時，指定により陰点または隠線は消去する。ic は点または線分の色である。mode の値により以下の描画指定を行なう。

mode=0	現在点に点を移動する
mode=1	直前に fr_pmdraw で指定した点から現在点まで線分を描く 隠線処理は行なわない
mode=-1	直前に fr_pmdraw で指定した点から現在点まで線分を描く 隠線処理を行なう
mode=2	現在点に点を描く，陰点は消去しない
mode=-2	現在点に点を描く，陰点は消去する
mode=3	現在点に小円を描く

注意　☆ 本ルーチンをコールする前に fr_pmset でパラメータ z を決定しておく必要がある。

☆ 本ルーチンをコールするとその点の座標は 1 回だけ記憶される。これは mode=1 または mode=-1 の指定によりつぎつぎに点を線分でつなぐときに便利である。ただし，一番最初は mode=0 または mode=2 または mode=-2 を用いて現在点を移動しておく必要がある。

☆ 隠線処理をする時は fr_pmgraph の注意と同様に fr_hclear(3.5.4) による指定を行ない，手前から描いていくことが必要である。なお，原理的に陰点・隠線を消去するか否かは図形の局所的データだけでは判断できないので，本ルーチンによる隠線消去は必ずしもうまくいかない。

☆ mode=3 の小円は fr_setcircle(3.8.4) で大きさを変更したり fr_setmark(3.8.5) で多角形に変更することが可能である。

☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。

☆ 本ルーチンを多用するときは注意が必要である。(付録 A.3 参照)

3.5.8 ２次元配列データを用いて鳥瞰図を描く (fr_profile)

書式　call fr_profile(f,imax,jmax,ic,mode)

引数　real*8 f(imax,jmax)

integer imax,jmax,ic,mode

動作　imax×jmax の２次元配列データ f(i,j) を，(i,j) 座標に対する高さを表すデータと見なした鳥瞰図を描く。ここで，i は x 座標に対応し，j はパラメータ z 座標に対応し，高さ方向は y 座標に対応する。鳥瞰図は描画ボックスの x 方向，z 方向をそれぞれ imax-1，jmax-1 分割し，その格子点に f(i,j) の高さデータがあるとして描く。

ic は図形の色である。mode の値により以下の隠線処理指定を行なう。

mode=0	隠線消去しない
mode=1	図の下方への飛び出しを許す隠線消去 (通常の方法)
mode=2	図の下方への飛び出しを許さない隠線消去

注意　☆ 本ルーチンを使用するためにはあらかじめ fr_pmview で描画ボックスを指定し，fr_pmframe により描画空間座標系を指定する必要がある。

☆ 本ルーチンをコールする前に fr_pmframe で与えた物理座標の境界値は図形に影響しない。しかし，データを plt ファイルに保存する際には fr_pmframe の指定が配列領域の境界座標を指定していると仮定して保存される。このデータを基準にして，再描画の際には表示境界値を変更することで図形の拡大・縮小が可能である。よって，２次元配列の境界の物理座標を用いて fr_pmframe の指定をするのが望ましい。

☆ 隠線処理指定は fr_hclear と同じであり，本ルーチンのみ使用するときは fr_hclear の指定は不要である。

☆ 本ルーチンを使用する前に，fr_pmframe でスケール指定をしない場合は x を (0,imax-1) で，y を最大値・最小値で，z を (0,jmax-1) でそれぞれ自動的にスケールし，枠線を描く。

3.6 ３次元図形描画ルーチン

本節では，３次元図形描画ルーチンについて述べます。これは３次元座標 (x, y, z) で指定されるデータを用いて描画するもので，前節の１パラメータ図形群とは異なり，リアルな３次元図形を表示することができます。*FRAMES* における３次元図形の描画は (1)*CANVAS* 上での描画エリアの決定，(2) 視点の指定，(3) ３次元描画ボックスのアスペクト比指定，(4) ３次元描画ボックスの座標指定，(5) 図形の描画，という手順が必要です。

３次元図形描画ルーチン `fr_graph3d`，`fr_draw3d` を使って空間曲線を描けば，１パラメータ図形群と同じアルゴリズムで隠線消去をすることができますが，少し注意が必要です。そもそも空間曲線が何を隠せるのでしょうか？１パラメータ図形群が隠線消去できたのは，パラメータの大小によって「前方の図形」,「後方の図形」の区別ができたからです。３次元曲線ではそれらに優劣はありません。よって，`fr_graph3d`，`fr_draw3d` で隠線消去するときは，描画している図形が前方か後方かをプログラマが意識し，かつ前方の図形が後方の図形を隠すべきか否かを判断する必要があります。一般的に３次元図形で隠線消去するためには「面」を定義する必要があります。曲面描画用の `fr_surface` や `fr_trisurface` ではデータから「面」を描いているため隠線が消去できます。

3.6.1 ３次元図形描画の手順

２次元図形を描くときには，単に縦横の縮尺を考えるだけで良かったが，３次元図形を２次元平面上に表現するには，その図形をどこから見ているかの指定が必要である。これには,「観測者の視点方向」と「図形と視点との距離」の二つを指定しなければならない。*FRAMES* では，指定変数を少なくするために，距離の指定は任意で，指定しなければ視点距離無限大の正射影表示になって，視点方向の角度 (緯度，経度) だけが指定変数となる。更に，図形の縮尺を考えなくてもよいように，描画エリアを指定すれば，その領域に収まるような描画ボックスを自動的に生成する。以下に手順の概要を述べる。

(1) *CANVAS* 上での描画エリアを決定する (`fr_view3d` または `fr_margin3d`)

まず `fr_view3d` または `fr_margin3d` を用いて描画空間上での描画エリアを決定する。この描画エリアは１パラメータ図形群と異なり，３次元描画ボックスを縮尺して収める長方形領域である (図 3.4 参照)。３次元描画ボックスは，x 座標，y 座標，z 座標の最小・最大値で決定される直方体であり，`fr_aspect3d` や `fr_frame3d` で決定される。このため，図 3.4 のように決定した描画エリアと３次元描画ボックスは必ずしも一致しない。(縦か横かどちらか１方向は一致する)

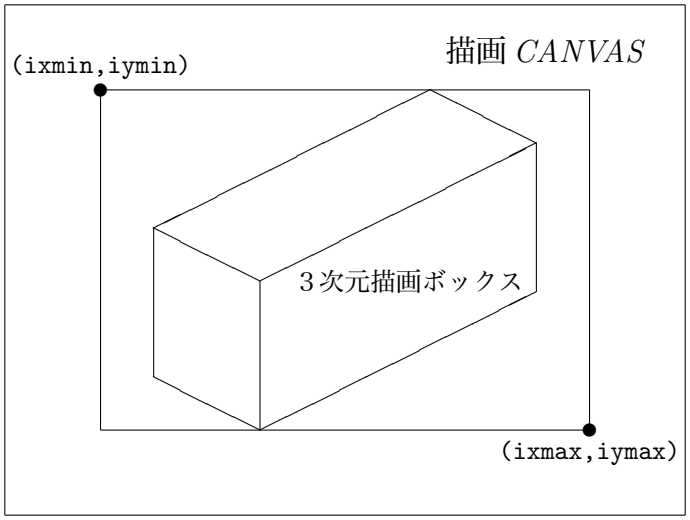


図 3.4　３次元描画エリアの指定点

(2) 視点の方向と距離を決定する (`fr_angle3d`，`fr_rotate`，`fr_project`)

次に `fr_angle3d` を用いて３次元描画ボックスに対しての視点の方向を緯度・経度で指定する (図 3.5 参照)。z 軸は常に画面の縦座標軸に投影されるようになっている (変更可能)。透視法を用いない場合は，視点方向に垂直な面上に正射影した図形を描くので，視点方向の指定だけで表示図形が決定される。

FRAMES で用いている３次元描画ボックスの大きさ決定法は，指定の描画エリアに自動的に図形を収めることができるという利点がある半面，同じ描画エリアで同じ３次元描画ボックスを指定しても視点の角度を変えると図形の縮尺が異なるという欠点を持つ。そこで，一旦決定された縮尺を保ちつつ視点方向を変化させる `fr_rotate` を用意してある。

透視図表示をする場合は，さらに `fr_project` を用いて，視点と物体との距離，および透視図モードの設定を行なう。

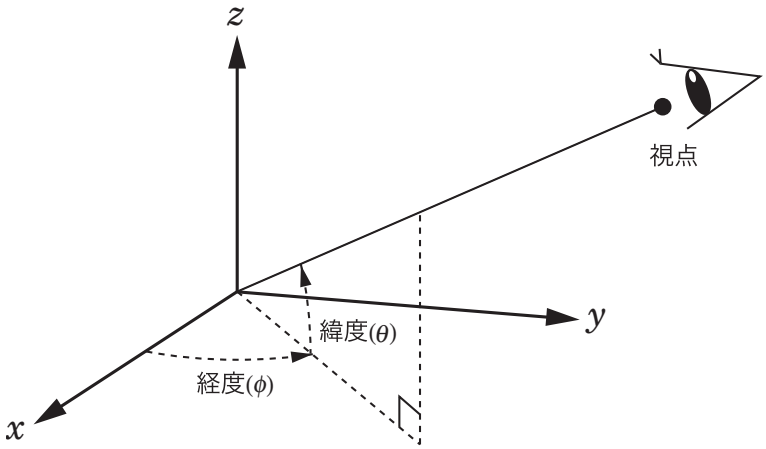


図 3.5　３次元図形の視点の設定方法

(3) ３次元描画ボックスのアスペクト比を決定する (`fr_aspect3d`)

次に，`fr_aspect3d` を用いて３次元描画ボックスの x 方向，y 方向，z 方向の表示のアスペクト比 $r_x : r_y : r_z$ を決定する。３次元図形は視点方向によって形が変化するので，x，y，z 座標それぞれの縮尺をあらかじめ決定する必要がある。このアスペクト比の指定は３通りある。

第１はそれぞれの方向の物理座標で測った距離の比を与える方法である。例えば $r_x : r_y : r_z = 1 : 2 : 3$ であれば，x 方向を１倍，y 方向を２倍，z 方向を３倍した図形を描く。この時は物理座標の境界値で縮尺された図形を描くわけであるから，３次元描画ボックスは次の `fr_frame3d` により物理座標の境界値を指定してから決定される。

第２は３次元描画ボックスの表示サイズの比を直接与える方法である。例えば $r_x : r_y : r_z = 1 : 2 : 3$ であれば，x 方向：y 方向：z 方向が $1 : 2 : 3$ の３次元描画ボックスになる。後で `fr_frame3d` で指定する物理座標の境界値は描画ボックスの形状に影響を与えない。

第３の方法は，$r_x : r_y$ は x，y 方向の物理座標の距離比，$r_x : r_z$ は x，z 方向の表示サイズ比を与える方法である。

第１の方法は実際の３次元形状を表現するのに最適であり，第２の方法は x，y，z 方向全てのスケールや次元が異なる場合に便利である。また第３の方法は `fr_profile` を用いて実験データの鳥瞰図表示をする時のように z 方向のみスケールや次元が異なる場合に用いる。

(4) ３次元描画ボックスの境界の物理座標を指定する (`fr_frame3d`)

次に，`fr_frame3d` を用いて３次元描画ボックスの x 座標，y 座標，z 座標の境界値を指定する。これにより描画ボックスが決定される。この時同時に座標軸を描くことができる。

(5) 図形を描く (`fr_graph3d`，`fr_draw3d`，`fr_vector3d`，等)

最後に，描画ルーチンを用いて３次元図形を描く。以下に説明する `fr_graph3d` や `fr_vector3d` などの他に，１パラメータ図形群の項目にも含まれいてる `fr_xyzname`，`fr_profile` もこの３次元図形描画ルーチンとして使用可能である。これらは，デフォルトでは３次元図形描画ルーチンとして働き，`fr_pmview`，`fr_pmframe` のいずれかをコールした後には１パラメータ図形群のルーチンとして働く。

3.6.2 *CANVAS* における描画エリアを描画座標で決定する (fr_view3d)

書式　call fr_view3d(ixmin,iymin,ixmax,iymax)

引数　integer ixmin,iymin,ixmax,iymax

動作　*CANVAS* における 3 次元描画エリアを (ixmin,iymin)-(ixmax,iymax) に決定する。ここで (ixmin, iymin) は，それぞれエリアの左上，(ixmax,iymax) はエリアの右下の座標を表す (図 3.4 参照)。

注意　☆ 標準の *CANVAS* 領域は (0,0)-(639,479) なので (ixmin,iymin)，(ixmax,iymax) は，基本的にこの範囲になければならない。ただし，図形の一部のみ表示すればよいのであれば，範囲外の指定をしても差しつかえない。

☆ fr_margin3d で指定する場合には本ルーチンは不要である。

☆ 本ルーチンで指定しなければ，それぞれデフォルト値 (ixmin,iymin)=(30,40)，(ixmax,iymax)=(610,420) になる。

☆ *FRAMES* における 3 次元ルーチンのサイズ指定方法では，同じ fr_view3d 指定で同じ 3 次元描画ボックスを指定しても視点の角度を変えると図形の縮尺が異なる。

☆ 本ルーチンのコール後 fr_frame3d の指定はクリアされる。

3.6.3 *CANVAS* における描画エリアを上下左右のマージンで決定する (fr_margin3d)

書式　call fr_margin3d(ileft,itop,iright,ibottom)

引数　integer ileft,itop,iright,ibottom

動作　*CANVAS* における 3 次元描画エリアを，*CANVAS* 全体領域の左マージン ileft，上マージン itop，右マージン iright，下マージン ibottom で決定する。もし，描画 *CANVAS* のサイズが xsize×ysize の場合には，描画座標で (ileft,itop)-(xsize-iright,ysize-ibottom) が描画エリアとなる (図 3.4 参照)。

注意　☆ fr_view3d で指定する場合には本ルーチンは不要である。

☆ 本ルーチンのコール後 fr_frame3d の指定はクリアされる。

3.6.4 視点方向を決定する (fr_angle3d)

書式　call fr_angle3d(theta,phi)

引数　real*8 theta,phi

動作　視点方向の緯度を theta に，経度を phi に決定する。単位は「度」である。ここで緯度は x–y 平面を赤道面として測り，経度は x 軸方向を 0 度として測るものとする (図 3.5 参照)。

注意　☆ 基本的に-90≤theta≤90，-180<phi≤180 の範囲で与える。これ以外の値を与えた場合は theta が 180，phi が 360 を周期として換算した角度となる

☆ デフォルトは theta=45，phi=-45 である

3.6.5 視点距離を決定する (fr_project)

書式　call fr_project(d,mode)

引数　real*8 d
integer mode

動作　透視図法を用いる場合の図形の中心と視点までの距離を d に設定する。
mode の 1 の位 m1 と 10 の位 m2 とで以下の射影方法が指定できる。

m1=0	正射影 (デフォルト)，d には依存しない
m1=1	透視図
m2=0	z 軸を *CANVAS* の上下方向とする (デフォルト)
m2=1	x 軸を *CANVAS* の上下方向とする
m2=2	y 軸を *CANVAS* の上下方向とする

注意　☆ d は 3 次元描画ボックスの対角線の半分を単位としてボックスの中心から測った長さを与える。このため実際の図形の大きさには無関係である。5 程度が適当である。

☆ m2=1 の場合と m2=2 の場合，緯度と経度の基準面はそれぞれ，y–z 平面，z–x 平面となる。
☆ 本ルーチンの設定は描画ページを切り替えてもリセットされず，次に本ルーチンを使って設定変更するまで有効となる。

3.6.6 ３次元描画ボックスのアスペクト比を指定する (fr_aspect3d)

書式　call fr_aspect3d(rtx,rty,rtz,mode)

引数　real*8 rtx,rty,rtz
integer mode

動作　３次元描画ボックス (物理座標で指定した空間) の表示アスペクト比を rtx:rty:rtz に指定する。ここで，mode の指定によりアスペクト比は３通りの意味を持つ。

mode=0	物理座標のスケール比 (デフォルト)
mode=1	３次元描画ボックスの表示サイズの比
mode=2	rtx:rty は物理座標のスケール比 rtx:rtz は３次元描画ボックスの表示サイズの比
mode=3	rtx:rty は物理座標のスケール比 rty:rtz は３次元描画ボックスの表示サイズの比

注意　☆ mode=0 の時は fr_frame3d で指定した x 方向の物理座標区間を rtx 倍，y 方向の物理座標区間を rty 倍，z 方向の物理座標区間を rtz 倍した直方体とする。このため，３次元描画ボックスのサイズは fr_frame3d 実行後に決定される。このモードは実際の３次元形状を表現するときに便利である。
☆ mode=1 の時は３次元描画ボックスの縦横高さの比が指定したアスペクト比に固定される。このモードは全ての方向のスケールや次元が異なるデータのときに便利である。
☆ mode=2 と mode=3 の時は fr_frame3d が何であっても z 方向の３次元描画ボックスの表示サイズ比は固定される。このモードは fr_profile を用いて２次元空間データの鳥瞰図表示をするときのように z 方向だけ次元が異なるときに用いる。
☆ デフォルトは 1 : 1 : 1 で，mode=0 である。

3.6.7 ３次元描画ボックスの物理座標を指定して座標軸を描く (fr_frame3d)

書式　call fr_frame3d(xmin,xmax,ymin,ymax,zmin,zmax,mode)

引数　real*8 xmin,xmax,ymin,ymax,zmin,zmax
integer mode

動作　３次元描画ボックスにおける x 座標の境界を xmin と xmax に，y 座標の境界を ymin と ymax に，z 座標の境界を zmin と zmax に指定する。この値を用いて３次元描画ボックスを決定する。同時に指定に応じて座標軸を描く。座標軸の種類は mode の値により以下の指定ができる。

mode=0	座標軸のみを描く
mode=1	スケールのみを行なう (枠は描かない)
mode=2	３次元描画ボックスの枠を描く
mode=3	座標軸と直方体の枠を描く
mode=4	座標軸と直方体の枠を描く，さらに視点と反対側の境界線を描く

なお，mode の 10 の位を 1 にすると (10，12 のように)，３次元描画ボックスでクリッピングを行う。すなわち，以下の描画ルーチン使用の際に描画ボックスの外にはみ出した部分は描かない。

注意　☆ fr_graph3d，fr_profile，fr_surface，fr_trisurface には自動スケール機能がある。
☆ fr_setaxis(3.8.8) を併用することで，座標の数字を描画しないようにすることができる。この場合，目盛りは描画する。

3.6.8 視点方向を変更する (fr_rotate)

書式　call fr_rotate(theta,phi,mode)

引数　real*8 theta,phi
integer mode

動作　一度決定した３次元描画ボックスに対し，視点方向の緯度を theta に，経度を phi に変更する。単位は「度」である。mode の指定により座標軸を描く。mode の指定は fr_frame3d に準じる。

注意　☆ 角度の指定は fr_angle3d と同じである。
☆ *FRAMES* における３次元描画ボックスの指定方法では，同じ描画エリア指定，同じ３次元ボックスの指定でも，視点方向が異なれば図形の拡大・縮小率が異なる。本ルーチンはこの欠点を解消するために用意したものである。即ち，fr_aspect3d，fr_frame3d などを用いて一度３次元描画ボックスを決定した後，本ルーチンを用いて視点を変更すると，拡大・縮小率を保ったまま視点のみ変更した３次元描画ボックスとなる。
☆ 視点方向によっては３次元描画ボックスが描画エリアからはみ出す可能性がある。

3.6.9 ３次元クリッピング領域を決定する (fr_clip3d)

書式　call fr_clip3d(xmin,xmax,ymin,ymax,zmin,zmax,mode)

引数　real*8 xmin,xmax,ymin,ymax,zmin,zmax
integer mode

動作　(xmin,ymin,zmin)-(xmax,ymax,zmax) で決定される３次元物理座標直方体領域をクリッピング領域とする。ここで，クリッピング領域とはその内部の領域は描画し，外部領域は必要に応じて描画したりしなかったりできる領域のことである。

mode=0　外部領域を描く (クリッピングしない)
mode=1　外部領域を描かない (クリッピングする)

注意　☆ fr_frame3d で３次元描画ボックスにクリッピング指定をした後に本ルーチンでクリッピング指定をした場合には本ルーチンの指定が優先する。

3.6.10 座標軸にタイトルを付ける (fr_xyzname)

書式　call fr_xyzname(chx,chy,chz)

引数　character chx,chy,chz

動作　fr_frame3d で描いた座標軸にタイトルを付ける。ここで，chx は x 座標のタイトルを表す文字列，chy は y 座標のタイトルを表す文字列，chz は z 座標のタイトルを表す文字列である。

注意　☆ 本ルーチンは３次元描画ボックスに対して文字の位置を決定するので，fr_frame3d の前でコールした場合はその時点では描かれず，設定だけしてその後の fr_frame3d 実行時にタイトルを描く。fr_graph3d など自動スケール機能を持つルーチンを使用した場合も同様である。
☆ ただし，その設定はページ切り替え時にクリアされる。すなわち，設定した後で図形を描かなければ本ルーチンは無視されることになる。

3.6.11 隠線消去指定を行なう (fr_hclear)

書式　call fr_hclear(mode)

引数　integer mode

動作　隠線処理モードの指定を行なう。mode の値により以下の指定ができる。

mode=0　隠線消去しない
mode=1　図の下方への飛び出しを許す隠線消去 (通常の方法)
mode=2　図の下方への飛び出しを許さない隠線消去

注意　☆ fr_graph3d (3.6.12) 等において隠線処理指定をしたときは，最大最小法を用いた隠線処理を行っている。これは描画を手前から順に行なって，手前の図形で遮られた場所を計算し，隠線かそうでないかの判断を行なうものである。下方への飛び出しは，手前の図形を線図と考えるか，それともそ

の線で上方を切り取られた面と考えるかにより異なる。前者は飛び出しを許す方法であり，後者は許さない方法である。

☆ デフォルトでは mode=0 になっている。即ち隠線消去はしない。

3.6.12 配列に格納されたデータを用いて３次元図形を描く (fr_graph3d)

書式　call fr_graph3d(x,y,z,n,ic,mode)

引数　real*8 x(n),y(n),z(n)
integer n,ic,mode

動作　n 個の物理座標点 (x(i),y(i),z(i)) を用いて画面上にグラフを描く。ic は描画の色である。グラフの種類は mode の値により以下の指定ができる。

mode=-1	簡易隠線処理をしながら n 個の点を順に結んだ折れ線グラフを描く
mode=-3	遠近を考慮して n 個の小球を描く
mode=-13	遠近を考慮して色の異なる n 個の小球を描く (注意参照)
mode=1	n 個の点を順に結んだ折れ線グラフを描く
mode=2	n 個の点を描く
mode=3	n 個の小円を描く
mode=4	n 個の点を番号順に２点ずつまとめてその２点間全てに線分を描く
mode=5	n 個の点を番号順に２点ずつまとめてその２点間全てに矢印を描く
mode=6	n 個の点をスプライン補間により順に結んだ曲線を描く データ配列 x(i)，y(i)，z(i) は，１番目の点座標，１番目の微係数，２番目の点座標，２番目の微係数... というように，座標とその点の微係数を交互に並べて与える。よって n は点の数の 2 倍になる。

mode の 100 の位 m3 を指定することで (102，405 のように)，１次元または２次元の等間隔データを描くことができる。m3 は次の６個の指定ができる。

m3=1	y(n)・z(n) の２次元等間隔データ
m3=2	x(n)・z(n) の２次元等間隔データ
m3=3	z(n) の１次元等間隔データ
m3=4	x(n)・y(n) の２次元等間隔データ
m3=5	y(n) の１次元等間隔データ
m3=6	x(n) の１次元等間隔データ

この指定の時には，残りの座標配列は要素数２で宣言し (x(n) なら y(2)・z(2)，x(n)・z(n) なら y(2) など)，その１番目と２番目の要素にデータ要素の最初と最後の座標値をそれぞれ与える。例えば，m3=1 の時には，x(1) に y(1)・z(1) の x 座標を，x(2) に y(n)・z(n) の x 座標を与え，m3=5 の時には，x(1)・z(1) に y(1) の x 座標・z 座標を，x(2)・z(2) に y(n) の x 座標・z 座標を与える。等間隔であることがわかっているときにはこの指定を使うことで，メモリの節約ができる。

注意　☆ mode=-1 で隠線処理をする場合には以下の２点の注意が必要である。

1. あらかじめ fr_hclear を用いて隠線消去指定を行なう。
2. 図を手前から順に描く。即ち，視点から遠ざかるような順序で描く。

☆ 本ルーチンでは最大最小法を用いて隠線消去を行なうため，1 価関数でないものはうまく表示しない。

☆ mode=-13 を使う時には色指定変数 ic を点の数と同じ要素数の配列 ic(n) にし，それぞれの小球の色 (1〜7) を格納しておく。なお，７色全て使っても 256 色以内の収まるので同時発色は可能である。

☆ mode=-3 と mode=-13 の小球は fr_setcircle(3.8.4) で大きさを変更することが可能である。但し，色の分割数が少ないのであまり大きな球にすると粗さが目立つ。

☆ 小球描画は球の重なりを考慮していないので球の間隔が球の直径より短くなると正しい表現ではなくなる。

☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。ただし，小球モードでは機能しないので

ic>0 にしなければならない。

☆ mode=3 の小円は fr_setcircle(3.8.4) で大きさを変更したり fr_setmark(3.8.5) で多角形に変更することが可能である。

☆ mode=4 は複数のばらばらな線分を一度に描くのに使用する。この場合と mode=5 の時，n は線分または矢印の数の 2 倍を指定することになる。

☆ mode=5 の矢印において，矢の先端のサイズは fr_setarrow(3.8.6) で変更することが可能である。

☆ mode=6 の場合，データ配列 x(i)，y(i)，z(i) は，1 番目の点座標，1 番目の微係数，2 番目の点座標，2 番目の微係数... というように，座標とその点の微係数を交互に並べて与える。よってこのときも，n は点の数の 2 倍を指定することになる。

☆ スプライン補間における微係数は点番号 k に対しての座標点 x_k を関数 $x(k)$ と見なし，k で微分したオーダーの数値を与えるとよい。

☆ mode=6 のスプライン補間では，各点間を 16 分割して曲線を描く。ただし，再描画アプリケーション tplot を使って PostScript 出力をする際には Bezier 曲線を出力するのでプリンタの解像度に合った精度のよい曲線を描くことができる。

☆ m3 の指定は，x・y・z を要素数 2 で宣言するか否かを 1・0 とし，zyx の順で並べた 2 進数になっている。

☆ 本ルーチンを使用する前に fr_frame3d をコールしなかったときは，x(i)，y(i)，z(i) の最小値，最大値でスケールした 3 次元描画ボックス (座標軸あり) の中に描画する。

3.6.13 点または線分を描く (fr_draw3d)

書式　call fr_draw3d(x,y,z,ic,mode)

引数　real*8 x,y,z
integer ic,mode

動作　(x,y,z) で指定した座標に点を描く。または現在点から (x,y,z) で指定した座標へ線分を描く。この時，指定により陰点または隠線は消去する。ic は点または線分の色である。mode の値により以下の描画指定を行なう。

mode=0	現在点に点を移動する
mode=1	直前に fr_draw3d で指定した点から現在点まで線分を描く 隠線処理は行なわない
mode=-1	直前に fr_draw3d で指定した点から現在点まで線分を描く 隠線処理を行なう
mode=2	現在点に点を描く
mode=3	現在点に小円を描く
mode=-3	現在点に小球を描く

注意　☆ 本ルーチンをコールするとその点の座標は 1 回だけ記憶される。これは mode=1 または mode=-1 の指定によりつぎつぎに点を線分でつなぐときに便利である。ただし，一番最初は mode=0 または mode=2 を用いて現在点を移動しておく必要がある。

☆ 隠線処理は fr_graph3d の注意と同様に fr_hclear による指定を行ない，手前から書いていくことが必要である。

☆ 原理的に隠線消去をするか否かは図形の局所的データからは判断できない。本ルーチンは 2 次元ルーチンにおける fr_draw2d の対応形として作成したが，データ点が接近すると隠線消去が不完全になるので注意されたい。隠線消去する場合は，できる限り fr_profile や fr_surface を使用すること。

☆ mode=3 の小円は fr_setcircle(3.8.4) で大きさを変更したり fr_setmark(3.8.5) で多角形に変更することが可能である。

☆ mode=-3 の小球は fr_setcircle(3.8.4) で大きさを変更することが可能である。

☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。ただし，小球モードでは機能しないので ic>0 にしなければならない。

☆ 本ルーチンは遠近を考慮しないため，mode=-3 の小球を複数描くときには注意が必要である。
☆ 本ルーチンを多用するときは注意が必要である。(付録 A.3 参照)

3.6.14 3次元ベクトル (矢印) を描く (fr_vector3d)

書式　call fr_vector3d(x,y,z,dx,dy,dz,ic,mode)

引数　real*8 x,y,z,dx,dy,dz
integer ic,mode

動作　(x,y,z) で指定した座標を始点としたベクトルを描く。ic はベクトルの色である。ベクトルの終点 (矢印の先) は (dx,dy,dz) を用いて指定するが，mode の値により指定方法が以下のように異なる。

mode=0	(dx,dy,dz) を物理座標でのベクトルと見なして描く 即ち，(x+dx,y+dy,z+dz) を終点とする
mode=1	(dx,dy,dz) を終点とする

注意　☆ 2次元の fr_vector2d に対応する mode=2 と mode=3 は用意していない。
☆ 矢の先端のサイズは fr_setarrow(3.8.6) で変更することが可能である。

3.6.15 2次元配列データを用いて鳥瞰図を描く (fr_profile)

書式　call fr_profile(f,imax,jmax,ic,mode)

引数　real*8 f(imax,jmax)
integer imax,jmax,ic,mode

動作　imax×jmax の2次元配列データ f(i,j) を，(i,j) 座標に対する高さを表すデータと見なした鳥瞰図を描く。ここで，i は x 座標に対応し，j は y 座標に対応し，高さ方向は z 座標に対応する。鳥瞰図は描画ボックスの x 方向，y 方向をそれぞれ imax-1，jmax-1 分割し，その格子点に f(i,j) の高さデータがあるとして描く。

ic は図形の色である。ic<0 の時には，色分け等高線描画のように，面をその高さに応じた色で着色する。

ic=-2	各4辺形の4頂点の高さに応じた色で連続的に着色する
ic=-3	各4辺形の4頂点の平均の高さに応じた色で着色する
ic=-13	各4辺形の4辺を両端の平均高さに応じた色で着色する (内部は背景色)
ic=-23	各4辺形の4辺を両端の平均高さに応じた色で着色する (内部は着色しない)
ic=-4	各4辺形の (i,j) 点の高さに応じた色で着色する
ic=-14	各4辺形の4辺を両端の頂点の色で半分づつ着色する (内部は背景色)
ic=-24	各4辺形の4辺を両端の頂点の色で半分づつ着色する (内部は着色しない)
ic=-5	各4辺形の4頂点をその高さに応じた色で着色する

mode の値により以下の隠線処理指定を行なう。

mode=0	隠線消去しない
mode=1	図の下方への飛び出しを許す隠線消去 (通常の方法)
mode=2	図の下方への飛び出しを許さない隠線消去
mode=5	ic を疑似ライティング色とした表面表示をする (フラット)
mode=6	ic を疑似ライティング色とした表面表示をする (強度補間)

なお，ic<0 の時には，mode≠0 でなければならない。

注意　☆ 本ルーチンをコールする前に fr_frame3d で与えた x 方向と y 方向の物理座標の境界値は描画ボックスの形状決定を除いては描画図形に影響しない。しかし，データを plt ファイルに保存する際には fr_frame3d が配列領域の境界座標を指定していると仮定して保存される。このデータを基準にして，再描画の際には表示境界値を変更することで図形の拡大・縮小が可能である。よって，2次元配列の境界の物理座標を用いて fr_frame3d の x 方向と y 方向の指定をするのが望ましい。

☆ mode=1 と 2 の隠線処理指定は fr_hclear と同じであり，本ルーチンのみ使用するときは fr_hclear の指定は不要である。

☆ 3.6.17 の fr_surface ルーチンで使用できる mode=3 または 4 の指定は現バージョンでは用意していないので欠番である。同様の効果を持たせる描画をするために ic=-4 を用意した。

☆ 本ルーチンを使用する前に，fr_frame3d で座標範囲指定をしない場合は，x 座標を (0,imax-1) で，y 座標を (0,jmax-1)，z 座標を f(i,j) の最小値・最大値でそれぞれ自動的にスケールし，枠を描く。

3.6.16 ２次元配列データを用いて描いた鳥瞰図にデータに応じた等高色分布図を描く (fr_mapprofile)

書式　call fr_mapprofile(f,g,imax,jmax,ic,mode)

引数　real*8 f(imax,jmax),g(imax,jmax)
integer imax,jmax,ic,mode

動作　imax×jmax の２次元配列データ f(i,j) を，(i,j) 座標に対する高さを表すデータと見なした鳥瞰図を描き，その曲面上にデータ g(i,j) に応じた等高色分布図を描く。ここで，i は x 座標に対応し，j は y 座標に対応し，高さ方向は z 座標に対応する。鳥瞰図は描画ボックスの x 方向，y 方向をそれぞれ imax-1，jmax-1 分割し，その格子点に f(i,j) の高さデータがあるとして描く。

ic は図形の色である。ic<0 の時には，色分け等高線描画のように，面をその高さに応じた色で着色する。

ic=-2	各４辺形の４頂点のデータ値に応じた色で連続的に着色する
ic=-3	各４辺形の４頂点の平均のデータ値に応じた色で着色する
ic=-13	各４辺形の４辺を両端の平均データ値に応じた色で着色する (内部は背景色)
ic=-23	各４辺形の４辺を両端の平均データ値に応じた色で着色する (内部は着色しない)
ic=-4	各４辺形の (i,j) 点のデータ値に応じた色で着色する
ic=-14	各４辺形の４辺を両端の頂点の色で半分づつ着色する (内部は背景色)
ic=-24	各４辺形の４辺を両端の頂点の色で半分づつ着色する (内部は着色しない)
ic=-5	各４辺形の４頂点をそのデータ値に応じた色で着色する

1≤ic≤7 の時には，疑似ライティング色で着色する。ただし濃淡を表すだけなので，光源の位置などには依存しない。

また，ic の 100 の位を指定することで (103 とか-402 など)，第２種色を指定することができる。ここで第２種色とは４辺形の辺の色である。

注意　☆ 本ルーチンをコールする前に fr_frame3d で与えた x 方向と y 方向の物理座標の境界値は描画ボックスの形状決定を除いては描画図形に影響しない。しかし，データを plt ファイルに保存する際には fr_frame3d が配列領域の境界座標を指定していると仮定して保存される。このデータを基準にして，再描画の際には表示境界値を変更することで図形の拡大・縮小が可能である。よって，２次元配列の境界の物理座標を用いて fr_frame3d の x 方向と y 方向の指定をするのが望ましい。

☆ 本ルーチンを使用する前に，fr_frame3d で座標範囲指定をしない場合は，x 座標を (0,imax-1) で，y 座標を (0,jmax-1)，z 座標を f(i,j) の最小値・最大値でそれぞれ自動的にスケールし，枠を描く。

☆ 現バージョンでは mode の値に意味はない。

3.6.17 ３次元曲面データを用いて曲面を描く (fr_surface)

書式　call fr_surface(x,y,z,imax,jmax,ic,mode)

引数　real*8 x(imax,jmax),y(imax,jmax),z(imax,jmax)
integer imax,jmax,ic,mode

動作　imax×jmax の２次元配列 x(i,j)，y(i,j)，z(i,j) を，(i,j) をパラメータとする３次元空間における曲面のパラメータ表示データと見なして，その曲面を描く。ここで，x(i,j) は x 座標データ，y(i,j) は y 座標データ，z(i,j) は z 座標データに対応する。

ic は図形の色である。ic<0 の時には，色分け等高線描画のように，面をその高さに応じた色で着色する。

ic=-2	各４辺形の４頂点の高さに応じた色で連続的に着色する
ic=-3	各４辺形の４頂点の平均の高さに応じた色で着色する
ic=-13	各４辺形の４辺を両端の平均高さに応じた色で着色する (内部は背景色)
ic=-23	各４辺形の４辺を両端の平均高さに応じた色で着色する (内部は着色しない)
ic=-4	各４辺形の (i,j) 点の高さに応じた色で着色する
ic=-14	各４辺形の４辺を両端の頂点の色で半分づつ着色する (内部は背景色)
ic=-24	各４辺形の４辺を両端の頂点の色で半分づつ着色する (内部は着色しない)
ic=-5	各４辺形の４頂点をその高さに応じた色で着色する

mode の値により以下の隠線処理指定を行なう。

mode=0	隠線消去しない
mode=1	図の下方への飛び出しを許す隠線消去 (通常の方法)
mode=2	図の下方への飛び出しを許さない隠線消去
mode=3	高度な隠線処理
mode=4	高度で mode=3 より高速な隠線処理
mode=5	ic を疑似ライティング色とした表面表示をする (フラット)
mode=6	ic を疑似ライティング色とした表面表示をする (強度補間)
mode=-1	曲面のデータを保存する (注意参照)

なお，ic<0 の時には，mode≠0 でなければならない。また，ic の 100 の位を指定することで (103 とか 405 など)，第２種色を指定することができる。ここで第２種色とは，

ic<0 または mode>4 のとき	メッシュの色
ic>0 で mode=3 または 4 のとき	曲面の色 (単色)

である。

mode=3 または 4 の時，mode の 100 の位 m3 を指定することで１次元または２次元の等間隔データを描くことができる (注意参照)。

注意　☆ 本ルーチンでは隠れた部分の処理として,「隠線処理」と「隠面処理」に大別される。

☆ 隠線処理方法は fr_profile などで用いている最大最小法 (mode=1，2) と，曲面全体から陰点の判定を行なう方法 (mode=3，4) の２通りの選択ができる。

☆ 最大最小法 (mode=1，2) は，処理は早いが z 座標に対して 1 価関数でない図形においては正常な隠線処理を行なわない。

☆ 曲面全体から陰点の判定を行なう方法 (mode=3，4) では，かなり複雑な曲面でも表示可能である。ただし，処理に少し時間がかかる。

☆ 上記の欠点を補うために，データを plt ファイルに出力した場合，mode が 1 か 2 の場合でも tplot による再描画の際に mode=4 に変更することができるようになっている (付録 C.2(3)-(a) 参照)。

☆ mode=3 または 4 の時，mode の 100 の位 m3 を指定することで (103，404 のように)，１次元または２次元の等間隔データを描くことができる。m3 は次の 6 個の指定ができる。

m3=1　　y(imax,jmax)・z(imax,jmax) の２次元等間隔データ
m3=2　　x(imax,jmax)・z(imax,jmax) の２次元等間隔データ
m3=3　　z(imax,jmax) の１次元等間隔データ
m3=4　　x(imax,jmax)・y(imax,jmax) の２次元等間隔データ
m3=5　　y(imax,jmax) の１次元等間隔データ
m3=6　　x(imax,jmax) の１次元等間隔データ

この指定の時には，残りの座標配列は要素数３で宣言し (x(imax,jmax) なら y(3)・z(3)，x(imax,jmax)・z(imax,jmax) なら y(3) など)，その１番目の要素に (1,1) の値，２番目の要素に (imax,1) の値，３番目の要素に (1,jmax) の値をそれぞれ与える。例えば，m3=1 の時には，x(1) に y(1,1)・z(1,1) の x 座標を，x(2) に y(imax,1)・z(imax,1) の x 座標を，x(3) に y(1,jmax)・z(1,jmax) の x 座標を与える。等間隔であることがわかっているときにはこの指定を使うことで，メモリの節約ができる。

☆ fr_profile(3.6.15) は曲面全体から陰点の判定を行なう方法 (mode=3，4) ができないので，これを本ルーチンの m3=3 の指定で補うことができる。即ち，x(1)=x(3)=x_{min}，x(2)=x_{max}，y(1)=y(2)=y_{min}，y(3)=y_{max} を与え，m3=3 にすれば，(x_{min}, y_{min})〜(x_{max}, y_{max}) の区間を imax×jmax に区切った領域に z(i,j) の鳥瞰図を描く。

☆ 上記の m3 の指定は，x・y・z を要素数３で宣言するか否かを１・０とし，zyx の順で並べた２進数になっている。

☆ mode=-1 を指定すると，曲面データを内部データ領域に保存するだけで描画はしない。この状態で，再び mode=-1 で本ルーチンをコールすると，その曲面データは以前に保存されたデータに追加される。これを繰り返して，最後に mode=3 または mode=4 で曲面を描画すると，その曲面に加えてそれまでに保存されていた曲面全てを一度に描画する。この機能を用いれば，複数の曲面から構成されている３次元図形を正しく隠線処理して表示することができる。

☆ 隠面処理は，曲面を構成する点の高さに応じた色で塗る等高色分布図 (ic=-2，-3) と疑似ライティングを用いたシェーディング (mode=5，6) の２通りの選択ができる。

☆ ic≦-2 を用いるときの色分布指定は fr_contour(3.4.9) に準じている。また，fr_setcontour (3.8.11) や fr_colorbar(3.8.12) なども併用可能である。

☆ ic=-2 は各頂点の色に応じて４辺形内部を連続的に色分けるため時間がかかる。グリッド点が多いときには ic=-3 または ic=-4 を用いると良い。

☆ mode=5 と mode=6 の場合には疑似ライティングとなる。色は，ic=1〜7 の７種類である。

☆ mode=6 を用いると，各頂点の光強度に応じて４辺形内部を補間して着色するので，なめらかな表面を表現できる。これに対し，mode=5 の場合には，各４辺形ごとに不連続な色を付けるため表面の粗さが見えるが，mode=6 より高速に描画する。

☆ 光源設定は fr_setlight(3.8.13) を用いて，表面の反射係数などは fr_setsurface(3.8.14) を用いて変更可能である。

☆ 本ルーチンを使用する前に，fr_frame3d でスケール指定をしない場合は x，y，z をそれぞれ x(i,j)，y(i,j)，z(i,j) の最小値・最大値で自動的にスケールし，座標軸を描く。

☆ 本ルーチンは *FRAMES* 7.2.0 以前のバージョンに含まれていた fr_profile3d を改良し，ワーク領域を自動的に割り付けるようにしたものである。fr_profile3d ルーチンは旧バージョンとの互換性のために残してあるので，旧バージョンで作成したプログラムも問題なく動作する。

3.6.18 ３次元曲面上にデータに応じた等高色分布図を描く (fr_mapsurface)

書式　call fr_mapsurface(x,y,z,f,imax,jmax,ic,mode)

引数　real*8 x(imax,jmax),y(imax,jmax),z(imax,jmax),f(imax,jmax)
integer imax,jmax,ic,mode

動作　imax×jmax の２次元配列 x(i,j)，y(i,j)，z(i,j) を，(i,j) をパラメータとする３次元空間における曲面のパラメータ表示データと見なして曲面を描き，その曲面上にデータ f(i,j) に応じた等高色分布図を描く。ここで，x(i,j) は x 座標データ，y(i,j) は y 座標データ，z(i,j) は z 座標データに対応する。

ic は図形の色である。ic<0 の時には，色分け等高線描画のように，面を f(i,j) に応じた色で着色する。

ic=-2	各４辺形の４頂点のデータ値に応じた色で連続的に着色する
ic=-3	各４辺形の４頂点の平均のデータ値に応じた色で着色する
ic=-13	各４辺形の４辺を両端の平均データ値に応じた色で着色する (内部は背景色)
ic=-23	各４辺形の４辺を両端の平均データ値に応じた色で着色する (内部は着色しない)
ic=-4	各４辺形の (i,j) 点のデータ値に応じた色で着色する
ic=-14	各４辺形の４辺を両端の頂点の色で半分づつ着色する (内部は背景色)
ic=-24	各４辺形の４辺を両端の頂点の色で半分づつ着色する (内部は着色しない)
ic=-5	各４辺形の４頂点をそのデータ値に応じた色で着色する

1≤ic≤7 の時には，疑似ライティング色で着色する。ただし濃淡を表すだけなので，光源の位置などには依存しない。

また，ic の 100 の位を指定することで (103 とか-402 など)，第２種色を指定することができる。ここで第２種色とは４辺形の辺の色である。

mode=3 または 4 の時，mode の 100 の位 m3 を指定することで１次元または２次元の等間隔データを描くことができる (注意参照)。

注意　☆ mode の 100 の位 m3 を指定することで (100，401 のように)，１次元または２次元の等間隔データを描くことができる。m3 は次の６個の指定ができる。

m3=1	y(imax,jmax)・z(imax,jmax) の２次元等間隔データ
m3=2	x(imax,jmax)・z(imax,jmax) の２次元等間隔データ
m3=3	z(imax,jmax) の１次元等間隔データ
m3=4	x(imax,jmax)・y(imax,jmax) の２次元等間隔データ
m3=5	y(imax,jmax) の１次元等間隔データ
m3=6	x(imax,jmax) の１次元等間隔データ

この指定の時には，残りの座標配列は要素数３で宣言し (x(imax,jmax) なら y(3)・z(3)，x(imax,jmax)・z(imax,jmax) なら y(3) など)，その１番目の要素に (1,1) の値，２番目の要素に (imax,1) の値，３番目の要素に (1,jmax) の値をそれぞれ与える。例えば，m3=1 の時には，x(1) に y(1,1)・z(1,1) の x 座標を，x(2) に y(imax,1)・z(imax,1) の x 座標を，x(3) に y(1,jmax)・z(1,jmax) の x 座標を与える。等間隔であることがわかっているときにはこの指定を使うことで，メモリの節約ができる。

☆ 上記の m3 の指定は，x・y・z を要素数３で宣言するか否かを１・０とし，zyx の順で並べた２進数になっている。

☆ ic≤-2 を用いるときの色分布指定は fr_contour(3.4.9) に準じている。また，fr_setcontour (3.8.11) や fr_colorbar(3.8.12) なども併用可能である。

☆ ic=-2 は各頂点の色に応じて４辺形内部を連続的に色分けるため時間がかかる。グリッド点が多いときには ic=-3 または ic=-4 を用いると良い。

☆ 本ルーチンを使用する前に，fr_frame3d でスケール指定をしない場合は x，y，z をそれぞれ x(i,j)，y(i,j)，z(i,j) の最小値・最大値で自動的にスケールし，座標軸を描く。

3.6.19 ３次元の３角形メッシュデータを用いて曲面を描く (fr_trisurface)

書式　call fr_trisurface(x,y,z,np,node,ns,ic,mode)

引数　real*8 x(np),y(np),z(np)
integer node(3,ns),np,ns,ic,mode

動作　与えられた３次元曲面 (連結していなくても良い) が ns 個の３角形で分割されているとする。各３角形の頂点は総計 np 個の座標点 (x(i),y(i),z(i)) から構成されていて，n (=1〜ns) 番目の３角形は頂点番号が node(1,n)，node(2,n)，node(3,n) の座標点 (node(k,n)=1〜np) からなるとする。本ルーチンはこれらの３次元３角形メッシュデータを用いて３次元曲面を描く。

ic は図形の色である。ic<0 の時には，色分け等高線描画のように，面をその高さに応じた色で着色する。

ic=-2	各３角形の３頂点の高さに応じた色で連続的に着色する
ic=-3	各３角形の３頂点の平均の高さに応じた色で着色する
ic=-13	各３角形の３辺を両端の平均高さに応じた色で着色する (内部は背景色)
ic=-23	各３角形の３辺を両端の平均高さに応じた色で着色する (内部は着色しない)
ic=-4	各３角形の node(1,*) 点の高さに応じた色で着色する
ic=-14	各３角形の３辺を両端の頂点の色で半分づつ着色する (内部は背景色)
ic=-24	各３角形の３辺を両端の頂点の色で半分づつ着色する (内部は着色しない)
ic=-5	各３角形の３頂点をその高さに応じた色で着色する

mode の値により以下の隠線処理指定を行なう。

mode=0	隠線消去しない
mode=3	高度な隠線処理
mode=4	高度で mode=3 より高速な隠線処理
mode=5	ic を疑似ライティング色とした表面表示をする (フラット)
mode=6	ic を疑似ライティング色とした表面表示をする (強度補間)

なお，ic<0 の時には，mode≠0 でなければならない。また，ic の 100 の位を指定することで (103 とか 405 など)，第２種色を指定することができる。ここで第２種色とは，

ic<0 または mode>4 のとき	メッシュの色
ic>0 で mode=3 または 4 のとき	曲面の色 (単色)

である。

注意　☆ 本ルーチンでは fr_surface における最大最小法指定 (mode=1，2) はない。

☆ 本ルーチンでは隠れた部分の処理として,「隠線処理」と「隠面処理」に大別される。隠線処理は fr_surface における高度な隠線処理と同じアルゴリズムを用いている。

☆ 隠面処理は，曲面を構成する点の高さに応じた色で塗る等高色分布図 (ic=-2，-3) と疑似ライティングを用いたシェーディング (mode=5，6) の２通りの選択ができる。

☆ ic≤-2 を用いるときの色分布指定は fr_contour(3.4.9) に準じている。また，fr_setcontour (3.8.11) や fr_colorbar(3.8.12) なども併用可能である。

☆ ic=-2 は各頂点の色に応じて３角形内部を連続的に色分けるため時間がかかる。グリッド点が多いときには ic=-3 または ic=-4 を用いると良い。

☆ mode=5 と mode=6 の場合には疑似ライティングとなる。色は，ic=1〜7 の７種類である。

☆ mode=6 を用いると，各頂点の光強度に応じて３角形内部を補間して着色するので，なめらかな表面を表現できる。これに対し，mode=5 の場合には，各３角形ごとに不連続な色を付けるため表面の粗さが見えるが，mode=6 より高速に描画する。

☆ 光源設定は fr_setlight(3.8.13) を用いて，表面の反射係数などは fr_setsurface(3.8.14) を用いて変更可能である。

☆ 本ルーチンを使用する前に，fr_frame3d でスケール指定をしない場合は x，y，z をそれぞれ x(i)，y(i)，z(i) の最小値・最大値で自動的にスケールし，座標軸を描く。

3.6.20 ３次元の３角形メッシュ上にデータに応じた等高色分布図を描く (fr_maptrisurface)

書式　call fr_maptrisurface(x,y,z,f,np,node,ns,ic,mode)

引数　real*8 x(np),y(np),z(np),f(np)
integer node(3,ns),np,ns,ic,mode

動作　与えられた３次元曲面 (連結していなくても良い) が ns 個の３角形で分割されているとする。各３角形の頂点は総計 np 個の座標点 (x(i),y(i),z(i)) から構成されていて，n (=1〜ns) 番目の３角形は頂点番号が node(1,n)，node(2,n)，node(3,n) の座標点 (node(k,n)=1〜np) からなるとする。本ルーチンはこれらの３次元３角形メッシュデータを用いて３次元曲面を描き，その曲面上にデータ f(i) に応じた等高色分布図を描く。

ic は図形の色である。ic<0 の時には，色分け等高線描画のように，面をその高さに応じた色で着色する。

ic=-2	各３角形の３頂点のデータ値に応じた色で連続的に着色する
ic=-3	各３角形の３頂点の平均のデータ値に応じた色で着色する
ic=-13	各３角形の３辺を両端の平均データ値に応じた色で着色する (内部は背景色)
ic=-23	各３角形の３辺を両端の平均データ値に応じた色で着色する (内部は着色しない)
ic=-4	各３角形の node(1,*) 点のデータ値に応じた色で着色する
ic=-14	各３角形の３辺を両端の頂点の色で半分づつ着色する (内部は背景色)
ic=-24	各３角形の３辺を両端の頂点の色で半分づつ着色する (内部は着色しない)
ic=-5	各３角形の３頂点をそのデータ値に応じた色で着色する

1≤ic≤7 の時には，疑似ライティング色で着色する。ただし濃淡を表すだけなので，光源の位置などには依存しない。

また，ic の 100 の位を指定することで (103 とか-402 など)，第２種色を指定することができる。ここで第２種色とは３角形の辺の色である。

注意　☆ ic≤-2 を用いるときの色分布指定は fr_contour(3.4.9) に準じている。また，fr_setcontour (3.8.11) や fr_colorbar(3.8.12) なども併用可能である。

☆ ic=-2 は各頂点の色に応じて３角形内部を連続的に色分けるため時間がかかる。グリッド点が多いときには ic=-3 または ic=-4 を用いると良い。

☆ 本ルーチンを使用する前に，fr_frame3d でスケール指定をしない場合は x，y，z をそれぞれ x(i)，y(i)，z(i) の最小値・最大値で自動的にスケールし，座標軸を描く。

3.6.21 ３次元データの等値面を描く (fr_isosurface)

書式　call fr_isosurface(f,imax,jmax,kmax,f0,ic,mode)

引数　real*8 f(imax,jmax,kmax),f0
integer imax,jmax,kmax,ic,mode

動作　imax×jmax×kmax の３次元配列 f(i,j,k) を (i,j,k) 座標に対するスカラーデータと見なし，閾値 f0 での等値面 f(i,j,k)=f0 を描く。等値面は３次元描画ボックスの x 方向，y 方向，z 方向をそれぞれ imax-1，jmax-1，kmax-1 分割し，その格子点に f(i,j,k) のデータがあると仮定して描く。

mode の値により以下の隠線処理指定を行なう。

mode=0	隠線消去しない
mode=1	高度な隠線処理をする
mode=5	ic を疑似ライティング色とした表面表示をする (フラット)
mode=6	ic を疑似ライティング色とした表面表示をする (強度補間)
mode=7	閾値未満の領域を ic の色を持つ立方体で埋める
mode=8	閾値以上の領域を ic の色を持つ立方体で埋める

注意　☆ 本ルーチンをコールする前に fr_frame3d で与えた物理座標の境界値は描画ボックスの形状決定以外は描画図形に影響しない。しかし，データを plt ファイルに保存する際には fr_frame3d が配列領

域の境界座標を指定していると仮定して保存される。このデータを基準にして，再描画の際には表示境界値を変更することで図形の拡大・縮小が可能である。よって，３次元配列の境界の物理座標を用いて fr_frame3d の指定をするのが望ましい。

☆ 本ルーチンを使用する前に，fr_frame3d で物理座標の境界値指定をしない場合は，x を (0,imax-1) で，y を (0,jmax-1) で，z を (0,kmax-1) で座標を決め，描画ボックスに座標軸を描く。

☆ mode=5 と mode=6 の場合には疑似ライティングとなる。色は，ic=1〜7 の７種類である。

☆ mode=6 を用いると，各点の光強度に応じて各区画内を補間して着色するので，なめらかな表面を表現できる。これに対し，mode=5 の場合には，各区画ごとに不連続な色を付けるため表面の粗さが見えるが，mode=6 より高速に描画する。

☆ mode=7 または mode=8 を用いると，区画 (i,j,k) の値 f(i,j,k) と閾値 f0 との大小を比較して，条件に合致すればその区画を立方体で埋める。立方体の各面は，ic で決まる疑似ライティング色で描画される。

☆ mode=7 または mode=8 において第２種色を指定すると，その色で立方体の各辺を描画する。

☆ 光源設定は fr_setlight(3.8.13) を用いて，表面の反射係数などは fr_setsurface(3.8.14) を用いて変更可能である。

3.6.22 ３次元データの断面等高線図を描く (fr_slices)

書式　call fr_slices(f,imax,jmax,kmax,td,nu,nd,ic,mode)

引数　real*8 f(imax,jmax,kmax),td(3)

integer imax,jmax,kmax,nu(3),nd,ic,mode

動作　imax×jmax×kmax 個の３次元配列データ f(i,j,k) を (i,j,k) 座標に対する高さを表すデータと見なした断面等高線図を描く。等高線図は３次元描画ボックスの x 方向，y 方向，z 方向をそれぞれ imax-1，jmax-1，kmax-1 分割し，その格子点に f(i,j,k) のデータがあると仮定して描く。

nd は等高線の間隔を決める。即ち，等高線は f(i,j) の最小値と最大値の間を nd 等分した時の等分線になる。なお，fr_setcontour(3.8.11) を併用することにより，高さ指定の等高線なども描画可能である。ic は等高線の色指定である。

等高線図を描く断面の位置と枚数は td と nu により指定する。すなわち，x 方向，y 方向，z 方向それぞれに対し，

x 方向	x＝td(1) の位置を含む y−z 面を等間隔に nu(1) 枚描く
y 方向	y＝td(2) の位置を含む z−x 面を等間隔に nu(2) 枚描く
z 方向	z＝td(3) の位置を含む x−y 面を等間隔に nu(3) 枚描く

なお，枚数 nu には 0 を指定することができる。その場合にはその方向の面は描かない。断面は fr_frame3d で指定した３次元描画ボックスのそれぞれの方向の最小値 tmin と最大値 tmax に対し，td の位置から等分値 Δt =(tmax-tmin)/nu ずつずれた面となる。

また，imax，jmax，kmax のどれか l つは 1 に指定することができる。このとき，その方向にはデータが一様にあると見なす。即ち，どこで断面をとっても同じ図形になる。

変数 mode は面の位置を決める変数 td の指定方法を決める。

mode=0	td を物理座標指定のデータと見なす
mode=1	３次元描画ボックスの最小値 tmin の位置を 0，最大値 tmax の位置を 1 とした相対ボックス座標指定

注意　☆ 本ルーチンを使用する前に，fr_frame3d で３次元描画ボックスの物理座標を指定する必要がある

☆ 分割方法，等高線の色指定は 3.4.9 の fr_contour 参照のこと。

☆ 本ルーチンをコールする前に fr_frame3d で与えた物理座標の境界値は描画ボックスの形状決定と t の位置指定を除けば描画図形に影響しない。しかし，データを plt ファイルに保存する際には fr_frame3d が配列領域の境界座標を指定していると仮定して保存される。このデータを基準にして，再描画の際には表示境界値を変更することで図形の拡大・縮小が可能である。よって，３次元配列の

境界の物理座標を用いて fr_frame3d の指定をするのが望ましい。

☆ 描画ボックス全体ではなく，一部の場所に描きたいときには fr_set3dbox(3.6.24) を用いる。

3.6.23 ３次元データの断面等高線図を描く（簡易版）(fr_contour3d)

書式　call fr_contour3d(f,imax,jmax,kmax,t,nd,ic,mode)

引数　real*8 f(imax,jmax,kmax),t

integer imax,jmax,kmax,nd,ic,mode

動作　imax×jmax×kmax 個の３次元配列データ f(i,j,k) を (i,j,k) 座標に対する高さを表すデータと見なした断面等高線図を描く。等高線図は３次元描画ボックスの x 方向，y 方向，z 方向をそれぞれ imax-1，jmax-1，kmax-1 分割し，その格子点に f(i,j,k) のデータがあると仮定して描く。

nd は等高線の間隔を決める。即ち，等高線は f(i,j) の最小値と最大値の間を nd 等分した時の等分線になる。なお，fr_setcontour(3.8.11) を併用することにより，高さ指定の等高線なども描画可能である。ic は等高線の色指定である。

等高線図を描く断面の位置は t と mode により指定する。変数 mode は，1 の位，10 の位，100 以上の位それぞれに意味がある。

mode の 1 の位 m1 は描く面の向きを指定する。

- m1=0　imax，jmax，kmax の中で最小の方向の t の位置に描く
- m1=1　x=t の位置の y−z 面に描く
- m1=2　y=t の位置の z−x 面に描く
- m1=3　z=t の位置の x−y 面に描く

mode の 10 の位 m2 は変数 t の指定方法を決める。

- m2=0　t を物理座標指定のデータと見なす
- m2=1　３次元描画ボックスの最小値 tmin の位置を 0，最大値 tmax の位置を 1 とした相対ボックス座標指定

mode の 100 以上の位 (m3=mode/100) は等高線を描く面の枚数を決定する。本ルーチンでは fr_frame3d で指定した t の方向の最小値 tmin と最大値 tmax に対し，t の位置から等分値 Δt =(tmax-tmin)/m3 ずつずれた面に等高線を描く。ただし，m3=0 の場合には t の位置に 1 枚描く。

また，imax，jmax，kmax のどれか 1 つは 1 に指定することができる。このとき，その方向にはデータが一様にあると見なす。即ち，どこで断面をとっても同じ図形になる。ただし，このときには m1=m3=0 でなければならない。即ち 1 の次元の方向に垂直な断面 1 枚しか描けない。

注意　☆ 本ルーチンは，fr_slices ルーチン (3.6.22) の簡易版で，1 方向のみの断面等高線を描くのに特化したものである。旧バージョンとの互換性を保つことも兼ねている。特に面倒でなければ fr_slices ルーチンを使用するのが望ましい。

☆ 注意に関しては，fr_slices ルーチン (3.6.22) を参照のこと。

3.6.24 等高線図領域に座標を入れる (fr_set3dbox)

書式　call fr_set3dbox(xmin,xmax,ymin,ymax,zmin,zmax,mode)

引数　real*8 xmin,xmax,ymin,ymax,zmin,zmax

integer mode

動作　断面等高線図を描く際の配列の境界値を (xmin,xmax)(x 方向)，(ymin,ymax)(y 方向)，(zmin,zmax)(z 方向) とする。

fr_contour3d(3.6.23) は，３次元描画ボックスと３次元配列領域が一致していると仮定して断面等高線図を描くため，図形は３次元ボックスで決まり，座標はそのボックスを決めた fr_frame3d が決める。もし，fr_frame3d と独立に配列領域に座標を入れて，３次元描画ボックスの自由な場所に３次元等高線図を描きたい場合には本ルーチンを使用する。

mode の値により，以下の指定ができる。

mode=0　　x，y，z 全て座標を入れる
mode=1　　x は座標を入れない，y，z は座標を入れる
mode=2　　x，z は座標を入れる，y は座標を入れない
mode=3　　x，y 座標を入れない，z は座標を入れない
mode=4　　x，y は座標を入れる，z は座標を入れない
mode=5　　x，z は座標を入れない，y は座標を入れる
mode=6　　x は座標を入れる，y，z は座標を入れない
mode=7　　x，y，z 全て座標を入れない

ここで，座標を入れない時は fr_frame3d の指定に従う。

注意　☆ 本ルーチンは 3 次元等高線図を描く直前にコールするのが望ましい。
☆ mode の指定は，座標を入れるか否かを 0・1 とし，zyx の順で並べた 2 進数になっている。

3.6.25　3次元流線を描く (fr_flow3dline)

書式　call fr_flow3dline(x,y,z,nmax,fx,fy,fz,imax,jmax,kmax,ic,fmin,nd)

引数　real*8 x(*),y(*),z(*),fmin
real*8 fx(imax,jmax,kmax),fy(imax,jmax,kmax),fz(imax,jmax,kmax)
integer nmax,imax,jmax,kmax,ic,nd

動作　3次元ベクトル場データ (fx(i,j,k),fy(i,j,k),fz(i,j,k)) を用いて流線を描く。ベクトル場データは 3 次元描画ボックスの x 方向，y 方向，z 方向をそれぞれ imax-1，jmax-1，kmax-1 分割し，その格子点にあると仮定する。

x(n)，y(n)，z(n) は描画用の補助配列であり，|nmax|以上の要素数の配列を用意しておく必要がある。さらに本ルーチンをコールする前に x(1)，y(1)，z(1) に流線の開始 x 座標，y 座標，z 座標をそれぞれ代入しておかなければならない。

ic は流線の色である。fmin は描画領域においてベクトルの長さがこれより小さくなったときに描画を止めるための下限値である。即ち，流線は，

(1) 点データが|nmax|を越えたとき
(2) ベクトルの長さが fmin より小さくなったとき
(3) 流線が指定領域外に出たとき

のいずれかの場合に描画を停止する。

nd は 2 つの意味がある。一つは描画精度を決めるもので格子間の nd 分の 1 の間隔で点を計算する。即ち，nd の絶対値が大きいほど精度が上がる。今一つは流線の向きを指定する。即ち，nd が正の時にはベクトルの方向に流線を描き，負の時には逆方向に描く。

imax，jmax，kmax のどれか 1 つは 1 に指定することができる。このとき，その方向には一様にベクトル場があると見なす。

nmax<0 の指定をすると，|nmax|を最大値として (x(n),y(n),z(n)) に流線座標を代入するが，流線は描かずに終了する。また，nmax に代入した座標の数を代入する。このため，nmax<0 を指定するときは nmax は変数でなければならない。

注意　☆ アルゴリズムは線形補間を用いた 4 次の Runge-Kutta 法である。
☆ n≥2 に対する x(n)，y(n)，z(n) にデータを設定する必要はないが，本ルーチンをコールした後，x(n)，y(n)，z(n) は破壊される。
☆ nmax<0 の時に nmax に代入される数値は正の数であり，|nmax|以下である。
☆ (3) の「指定領域」は，デフォルトでは 3 次元描画ボックス全体であるが，次の fr_flow3drgn により，物理座標で指定した部分領域に限定することも可能である。
☆ nmax>0 の時，流線の物理座標は本ルーチンの前にコールした fr_frame3d に依存する。もし，fr_frame3d を指定していなければ，x=0〜imax-1，y=0〜jmax-1，z=0〜kmax-1 の座標系となる。ただし，imax，jmax，kmax のどれか 1 つを 1 に指定した場合には fr_frame3d の指定が必要である。

☆ jmax=1 の時には，fr_setgeometry(3.6.27) を用いて円筒対称座標系における流線を描くことも可能である。

☆ nmax<0 の時，流線の物理座標は本ルーチンの前にコールした fr_frame3d に依存し，fr_frame3d の指定がなければ fr_flow3drgn に依存して，描画ページ切り替え後，どちらか最初にコールされたルーチンで指定した境界値を 3 次元配列の境界と一致するように決める。もし，どちらも 1 回も指定していなければ，x=0〜imax-1，y=0〜jmax-1，z=0〜kmax-1 の座標系となる。ただし，imax，jmax，kmax のどれか 1 つを 1 に指定した場合には fr_frame3d か fr_flow3drgn の指定が必要である。

☆ fr_set3drgn (3.8.3) を指定して fx，fy，fz の使用配列領域を限定することもできる。この時，fr_frame3d の指定がなければ，配列領域の最大値・最小値からそれぞれ 1 を引いた値の座標系となる。

☆ 本ルーチンは，直接描画するのではなく，(x(1),y(1),z(1)) を初期値とする流線の座標を配列，x(i)，y(i)，z(i) に代入した後，内部で fr_graph3d をコールして曲線を描画するものである。plt ファイルからデータを抽出するときは，fr_graph3d として抽出をしなければならない (付録 D 参照)。その場合，一般的に描画点の数が nmax より小さいことを考慮すること。

☆ nmax<0 の時は，plt ファイルにデータは保存されない。

3.6.26 3次元流線を描く際の流線描画領域を限定する (fr_flow3drgn)

書式　call fr_flow3drgn(xrmin,xrmax,yrmin,yrmax,zrmin,zrmax)

引数　real*8 xrmin,xrmax,yrmin,yrmax,zrmin,zrmax

動作　流線描画を物理座標 (xrmin,yrmin,zrmin)-(xrmax,yrmax,zrmax) で指定される直方体領域に限定する。

注意　☆ 本ルーチンで指定した物理座標の領域外に流線が出たときは流線描画を中止する。

☆ 本ルーチンの指定は描画ページ切り替えにより解除される。

☆ fr_flow3dline で nmax<0 を指定したとき，その前で本ルーチンをコールすると配列境界の座標指定となる。もし 2 回以上コールしたときは，最初のコールは配列境界の座標の指定となり，2 回目以降の指定が限定描画領域の指定となる。

3.6.27 2次元データを用いて3次元流線を描く際に座標系を指定する (fr_setgeometry)

書式　call fr_setgeometry(ngeom)

引数　integer ngeom

動作　2 次元配列データを用いて fr_flow3dline で 3 次元流線を描く時に 2 次元データの座標系を平板座標系か円筒座標系かの選択を行う。

ngeom は次の指定ができる。

ngeom=1	平板座標系 (デカルト座標系，デフォルト)
ngeom=2	円筒座標系 (x，z 座標データを与える)

ngeom=2 の時には，fr_flow3dline をコールする時に jmax=1 にし，fx，fy，fz を以下のような対応で与える。

fx(imax,1,kmax)	$\Longrightarrow$	fr(imax,kmax)	(径方向成分 f_r)
fy(imax,1,kmax)	$\Longrightarrow$	ft(imax,kmax)	(方位角成分 f_θ)
fz(imax,1,kmax)	$\Longrightarrow$	fz(imax,kmax)	(軸方向成分 f_z)

ここで，fr，ft，fz における imax は r 方向のグリッド数，kmax は z 方向のグリッド数である。

注意　☆ 作成される x(i)，y(i)，z(i) は直交座標系である。

3.7 基礎図形ルーチン

FRAMES は基本的に数値計算データを表示するためのものです。このため，データは全て物理座標で指定します。しかし，場合によっては fr_frame2d で描いた座標枠の上側に線を描きたいとか，右横に注釈文字を書きたいことがあります。ここで説明する基礎図形ルーチンは，描画座標を指定して，線分，円，多角形，文字を描くためのものです。なお，物理座標を用いて基礎図形を描きたい場合のために，オプションで物理座標も指定できるようになっています。さらに，３次元物理座標を用いて３次元空間に多角形を描くルーチンも用意しています。

fr_view2d などでの描画座標指定は整数で行いますが，以下のルーチンでは倍精度実数で指定します。*FRAMES* の描画空間 *CANVAS* はウィンドウのドット数を基礎にしているので描画エリアは整数で指定しますが，空間自体は連続であると仮定しているので，小数点以下の指定も意味を持ちます。特に高解像度の出力装置を用いる時には重要になります。

各基礎図形ルーチンは描画座標で位置を指定する絶対位置指定と，現在の「基準点」からのずれを指定する相対位置指定ができ，これは各ルーチン共通で含まれている引数 mode の 10 の位 m2 で指定します。

m2=0	x，y 共に絶対描画座標
m2=1	x は相対描画座標，y は絶対描画座標
m2=2	x は絶対描画座標，y は相対描画座標
m2=3	x，y 共に相対描画空間座標

物理座標で指定したい時には，mode の 100 の位 m3 で指定します。

m3=0	x，y は描画座標指定
m3=1	x，y は物理座標指定

ただしルーチンの性質上物理座標指定ができないものもありますので，各ルーチンの注意で確認して下さい。

物理座標を用いて図形を描くにはもう一つ方法があります。物理座標で指定した基準点移動ルーチンを用いて基礎図形共通の基準点を移動し，その点からの相対描画座標で指定する方法です。基準点移動には２次元座標の指定だけではなく３次元座標の指定もできます。

さらにこの方法では２通りの指定ができます。一つは fr_frame2d，fr_frame3d で決定される物理座標による指定と，３次元描画ボックスの (xyz)min の位置を 0，(xyz)max の位置を 1 とするボックス座標指定です。ボックス座標指定を用いると再描画の際に座標スケールを変換しても位置が変わらないという利点があります。

基礎図形のもう一つの特徴は，HLS 色指定ができることです。各ルーチン共通の色指定変数 ic に負数を与えると，現在のカラーマップに対し，|ic|番目の HLS 色指定になります。

3.7.1 線分を描く (fr_fpline)

書式　call fr_fpline(x1,y1,x2,y2,ic,mode)

引数
```
real*8 x1,y1,x2,y2
integer ic,mode
```

動作　(x1,y1)-(x2,y2) 間に線分，長方形，矢印を描く。ic は描画の色である。mode の 1 の位の値により以下の図形指定を行なう。

mode=0	２点間に線分を描く
mode=1	２点間を対角線とする長方形を描く
mode=2	２点間を対角線とする塗り潰した長方形を描く
mode=3	２点間に矢印を描く

注意　☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。
☆ 本ルーチンの実行後，基準点は (x2,y2) に移動する。
☆ mode の 10 と 100 の位の指定は上記の通り。
☆ mode=3 の矢印において，矢の先端のサイズは fr_setarrow(3.8.6) で変更することが可能である。

3.7.2 楕円を描く (fr_fpcircle)

書式　call fr_fpcircle(x,y,dx,dy,ic,mode)

引数　real*8 x,y,dx,dy
integer ic,mode

動作　(x,y) を中心とした楕円を描く。dx,dy は楕円の x 方向直径，y 方向直径である。ic は描画の色である。mode の 1 の位の値により内部塗り潰しの指定ができる。

mode=1　塗り潰さない
mode=2　塗り潰す

注意　☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。
☆ 本ルーチンの実行後，基準点は (x,y) に移動する。
☆ mode の 10 と 100 の位の指定は上記の通り。

3.7.3 正多角形（マーク）を描く (fr_fpmark)

書式　call fr_fpmark(x,y,r1,r2,n,ic,mode)

引数　real*8 x,y,r1,r2
integer n,ic,mode

動作　(x,y) を中心とした正多角形を描く。ここで，正多角形の中心からの半径は頂点が 1 個おきに等しく，2 個指定することが可能で，それぞれ r1，r2 で与える。n は正多角形の頂点数を与える。よって r1=r2 にすれば，正 n 角形となり，r1≠r2 にすれば，n/2 個の稜を持つ星形になる。
ic は描画の色である。mode の 1 の位の値により内部塗り潰しの指定ができる。

mode=1　塗り潰さない
mode=2　塗り潰す

注意　☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。
☆ n=0 にすれば，x 方向半径 r1，y 方向半径 r2 の楕円を描く。
☆ 本ルーチンの実行後，基準点は (x,y) に移動する。
☆ mode の 10 と 100 の位の指定は上記の通り。

3.7.4 多角形を描く (fr_fppolygon)

書式　call fr_fppolygon(x,y,n,ic,mode)

引数　real*8 x(n),y(n)
integer n,ic,mode

動作　n 個の点 (x(i),y(i)) を用いて画面上に多角形を描く。ic は描画の色である。mode の 1 の位の値により多角形の終端処理と塗り潰しを指定する。

mode=0　終端は閉じない
mode=1　最後の点と最初の点を結んで多角形を閉じる
mode=2　最後の点と最初の点を結んだ多角形を描き，内部を塗り潰す

注意　☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。
☆ 本ルーチンで描けるのは 500 角形までである。
☆ 本ルーチンの実行後，基準点は (x(1),y(1)) に移動する。
☆ mode の 10 と 100 の位の指定は上記の通り。

3.7.5 多数の点を描く (fr_fpdots)

書式　call fr_fpdots(x,y,ic,n,mode)

引数　real*8 x(n),y(n)
integer ic(*),n,mode

動作　n 個の点 (x(i),y(i)) に点を描く。ic(*) は描画の色である。mode の 1 の位の値により ic(*) の色の指定が異なる。

mode=0　　全ての点を ic(1) で指定された色にする
mode=1　　i 番目の点を ic(i) で指定された色にする

注意　☆ ic(*)<0 の時，その絶対値を HLS 指定色の指定番号とする。
☆ mode=0 の時は ic(2)〜ic(n) は使用しないので，ic は定数指定でもよい。
☆ 本ルーチンの実行後，基準点は (x(n),y(n)) で指定した点に移動する。
☆ mode の 10 と 100 の位の指定は上記の通り。

3.7.6　文字を描く (fr_fpchars)

書式　call fr_fpchars(x,y,ch,ic,ibx,iby,idx,mode)

引数　real*8 x,y
character ch
integer ic,ibx,iby,idx,mode

動作　(x,y) を基準点とした文字列 ch を描く。ic は描画の色である。基準点は文字列の中心にあり，そこからの文字列のずれを ibx，iby，idx で指定する。

ibx　　文字列全体の長さの半分を単位に右にずらす量 (負も可)
iby　　文字列の高さの半分を単位に下にずらす量 (負も可)
idx　　1 文字幅の半分を単位に右にずらす量 (負も可)

本ルーチンにおける「基準点」は他の基礎図形ルーチンに共通の基準点とは別の「文字基準点」というべきものである。文字基準点は，mode の 1 の位が 0 のときには共通の基準点と一致している。

これに対し，mode の 1 の位が 1 の時には共通の基準点は移動せず，文字基準点のみ移動する。文字基準点の移動は，直前の fr_fpchars に依存し，直前の文字基準点を原点とし，直前に描いた文字列の長さを単位として右に x，文字の高さを単位として下に y 移動した点を新たな文字基準点として描画する。これを用いて直前に実行した文字列に接続して描くことができる。例えば，x=1，y=0 として mode=1 とすれば，直前の文字列の右に続けて描き，x=0，y=1 として mode=1 とすれば，直前の文字列の下に続けて描く。

注意　☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。
☆ 文字列のずれ指定は細かく調節できるように作成したので少しわかりにくいかもしれない。良く使うものとして次の 3 つを覚えればよいであろう。y 方向は文字列の下側に基準がある。

(ibx,iby)=(0,-1)　　基準点に対してセンタリング
(ibx,iby)=(1,-1)　　基準点から右に書く (左寄せ)
(ibx,iby)=(-1,-1)　　基準点から左に書く (右寄せ)

☆ 通常は idx=0 で良い。
☆ mode の 1 の位が 0 の場合，本ルーチンの実行後，基準点は (x,y) に移動する。
☆ mode の 1 の位が 0 の場合，mode の 10 と 100 の位の指定は上記の通り。
☆ mode の 1 の位が 1 の場合，mode の 10 と 100 の位の指定は無視される。基準点は移動しない。
☆ 文字列の長さに，末尾のスペースは含まれない。もしスペースも入れて文字列の長さを測りたいときはその後に '|' (縦棒) を挿入する。このとき '|' 自体は描画されない。もし末尾に '|' を書きたいときは '||' と 2 本挿入する。
☆ もし，文字を書かないで位置だけ保存させたいときは文字列に半角スペースを 1 個与える。
☆ 本ルーチンの実行結果は描画文字の高さ・幅に依存する。このため，フォントセットが違うと実行結果は異なる。
☆ 日本語描画はサポートしていない。

3.7.7 ２次元物理座標で指定した点に基準点を移動する (fr_2dmove)

書式　call fr_2dmove(x,y,mode)

引数　real*8 x,y
integer mode

動作　２次元図形ルーチン fr_frame2d でスケールした物理座標 (x,y) に対応する描画座標に基準点を移動する。ここで mode は，0 を物理座標指定，1 をボックス座標指定 (最小値が 0，最大値が 1) として，(y 方向)(x 方向) のビットで表す。例えば，x・y 共に物理座標で指定するときは mode=0，x 方向は物理座標で y 方向はボックス座標としたければ，２進で (10)，即ち mode=2 を与える。

注意　☆ 本ルーチンは画面消去や画面切り換えをしてから 1 度も fr_frame2d による座標指定 (自動スケール含む) をしていなければ無視される。

3.7.8 ３次元物理座標で指定した点に基準点を移動する (fr_3dmove)

書式　call fr_3dmove(x,y,z,mode)

引数　real*8 x,y,z
integer mode

動作　１パラメータ図形群ルーチン fr_pmframe または３次元図形描画ルーチン fr_frame3d でスケールした物理座標 (x,y,z) に対応する描画座標に基準点を移動する。ここで mode は，0 を物理座標指定，1 をボックス座標指定 (最小値が 0，最大値が 1) として，(z 方向)(y 方向)(x 方向) のビットで表す。例えば，x・y・z 全て実座標で指定するときは mode=0，x 方向は実座標で y 方向はボックス座標で z 方向はボックス座標としたければ，２進で (110)，即ち mode=6 を与える。

注意　☆ 本ルーチンは，直前にスケールしたルーチンが fr_pmframe か fr_frame3d かで 1 パラメータ図形群の座標か３次元図形の座標かを判定する。
☆ 本ルーチンは，画面消去や画面切り換えをしてから 1 度も fr_pmframe または fr_frame3d による座標指定 (自動スケール含む) をしていなければ無視される。

3.7.9 ３次元空間線分を描く (fr_3dline)

書式　call fr_3dline(x1,y1,z1,x2,y2,z2,ic,mode)

引数　real*8 x1,y1,z1,x2,y2,z2
integer ic,mode

動作　１パラメータ図形群ルーチン fr_pmframe または３次元図形描画ルーチン fr_frame3d でスケールした３次元座標点 (x1,y1,z1) と (x2,y2,z2) を結ぶ線分を描く。ic は描画の色である。

注意　☆ 本ルーチンは，直前にスケールしたルーチンが fr_pmframe か fr_frame3d かで 1 パラメータ図形群の座標か３次元図形の座標かを判定する。
☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。
☆ 本ルーチンの実行後，基準点は (x2,y2,z2) で指定した点に移動する。
☆ mode の 10 の位 m2 の指定は，上記の説明において絶対描画座標を３次元物理座標に置き換えたものになる。
☆ mode の 100 の位による指定はない。また，現バージョンでは mode の 1 の位の値は意味がない。

3.7.10 ３次元多角形を描く (fr_3dpolygon)

書式　call fr_3dpolygon(x,y,z,n,ic,mode)

引数　real*8 x(n),y(n),z(n)
integer n,ic,mode

動作　１パラメータ図形群ルーチン fr_pmframe または３次元図形描画ルーチン fr_frame3d でスケールした n 個の３次元座標点 (x(i),y(i),z(i)) を用いて画面上に多角形を描く。ic は描画の色である。mode の 1 の位の値により多角形の終端処理と塗り潰しを指定する。

mode=0　　終端は閉じない
mode=1　　最後の点と最初の点を結んで多角形を閉じる
mode=2　　最後の点と最初の点を結んだ多角形を描き，内部を塗り潰す

注意　☆ 本ルーチンは，直前にスケールしたルーチンが fr_pmframe か fr_frame3d かで 1 パラメータ図形群の座標か 3 次元図形の座標かを判定する。
☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。
☆ 本ルーチンで描けるのは 500 角形までである。
☆ 本ルーチンの実行後，基準点は (x(1),y(1),z(1)) で指定した点に移動する。
☆ mode の 10 の位 m2 の指定は，上記の説明において絶対描画座標を 3 次元物理座標に置き換えたものになる。
☆ mode の 100 の位による指定はない。

3.7.11　3 次元座標指定で多数の点を描く (fr_3ddots)

書式　call fr_3ddots(x,y,z,ic,n,mode)
引数　real*8 x(n),y(n),z(n)
integer ic(*),n,mode
動作　1 パラメータ図形群ルーチン fr_pmframe または 3 次元図形描画ルーチン fr_frame3d でスケールした n 個の 3 次元座標点 (x(i),y(i),z(i)) に点を描く。ic(*) は描画の色である。mode の 1 の位の値により ic(*) の色の指定が異なる。

mode=0　　全ての点を ic(1) で指定された色にする
mode=1　　i 番目の点を ic(i) で指定された色にする

注意　☆ 本ルーチンは，直前にスケールしたルーチンが fr_pmframe か fr_frame3d かで 1 パラメータ図形群の座標か 3 次元図形の座標かを判定する。
☆ ic(*)<0 の時，その絶対値を HLS 指定色の指定番号とする。
☆ 本ルーチンで描けるのは 500 角形までである。
☆ 本ルーチンの実行後，基準点は (x(n),y(n),z(n)) で指定した点に移動する。
☆ mode の 10 の位 m2 の指定は，上記の説明において絶対描画座標を 3 次元物理座標に置き換えたものになる。
☆ mode の 100 の位による指定はない。

3.7.12　3 次元座標指定で円周を描く (fr_3dcircle)

書式　call fr_3dcircle(x,y,z,rx,ry,rz,rx1,ry1,rz1,ic,mode)
引数　real*8 x,y,z,rx,ry,rz,rx1,ry1,rz1
integer ic,mode
動作　1 パラメータ図形群ルーチン fr_pmframe または 3 次元図形描画ルーチン fr_frame3d でスケールした 3 次元座標点 (x,y,z) を中心とした円周を描く。mode の 1 の位の値により円の半径と向きを指定する。

mode=0　　3 次元座標点 (x+rx,y+ry,z+rz) を通り，ベクトル (rx,ry,rz) と (rx1,ry1,rz1) で指定した面内に円を描く
mode=1　　mode=0 と同じであるが，同じ半径の球の裏側の部分は描かない
mode=2　　円の半径を rx で指定し，(rx1,ry1,rz1) を法線ベクトルとする円を描く
mode=3　　mode=2 と同じであるが，同じ半径の球の裏側の部分は描かない

注意　☆ 本ルーチンは，直前にスケールしたルーチンが fr_pmframe か fr_frame3d かで 1 パラメータ図形群の座標か 3 次元図形の座標かを判定する。
☆ ic<0 の時，その絶対値を HLS 指定色の指定番号とする。
☆ 本ルーチンの実行後，基準点は (x+rx,y+ry,z+rz) で指定した点に移動する。

☆ 現バージョンでは正円しか描けないので，(rx1,ry1,rz1) の大きさは図形に無関係である。ただし，(0,0,0) や (rx,ry,rz) と同じ方向のベクトルを与えると面の指定ができないので，何も描かない。

☆ mode の 10 の位 m2 の指定は，上記の説明において絶対描画座標を 3 次元物理座標に置き換えたものになる。

☆ mode の 100 の位による指定はない。また，現バージョンでは mode の 1 の位の値は意味がない。

3.8 デフォルト値変更ルーチン，その他

FRAMES は手軽にグラフィック描画できることを目的としているため，サブルーチンの引数を少なくして，その他はデフォルト指定にしてあります。しかし，使い慣れてくると自分好みの指定をしたくなります。そのうちいくつかは以下に説明するルーチンにより変更可能です。

3.8.1 1次元配列を用いるルーチンに対するデータ間隔指定 (fr_setstep)

書式　call fr_setstep(ns)

引数　integer ns

動作　fr_graph2d，fr_pmgraph，fr_graph3d においては，通常全データを用いて描画するが，これを最初のデータから ns ステップ毎のデータを用いて描画するように変更する。

注意　☆ ns=0 を指定すると ns=1，即ちデフォルトと同様，全データを用いる。

☆ 本ルーチンの指定は fr_gclear 等による描画ページ切り替えにより解除される。

3.8.2 2次元配列を用いるルーチンの領域指定 (fr_set2drgn)

書式　call fr_set2drgn(imin,imax,jmin,jmax)

引数　integer imin,imax,jmin,jmax

動作　fr_contour，fr_sqrcontour，fr_profile，fr_flow2dline，fr_surface においては，通常2次元配列 f(i,j) の全データを用いて描画するが，これを imin≤i≤imax，jmin≤j≤jmax の範囲のデータを用いて描画するように変更する。

注意　☆ imin=0 を指定すると i 方向のデータは全て使用する。

☆ jmin=0 を指定すると j 方向のデータは全て使用する。

☆ 本ルーチンを用いても fr_contour，fr_sqrcontour，fr_profile，fr_flow2dline，fr_surface での要素数は配列の全要素数を指定しなければならない。

☆ fr_contour，fr_profile，fr_flow2dline の様に，2次元描画ボックスと配列の境界が一致していると仮定して図形を描画するルーチンにおいては2次元描画ボックスと指定された配列領域の境界が一致しているように描く。

☆ 本ルーチンの指定は fr_gclear 等による描画ページ切り替えにより解除される。

3.8.3 3次元配列を用いるルーチンの領域指定 (fr_set3drgn)

書式　call fr_set3drgn(imin,imax,jmin,jmax,kmin,kmax)

引数　integer imin,imax,jmin,jmax,kmin,kmax

動作　fr_contour3d，fr_flow3dline，fr_isosurface においては，通常3次元配列 f(i,j,k) の全データを用いて描画するが，これを imin≤i≤imax，jmin≤j≤jmax，kmin≤k≤kmax の範囲のデータを用いて描画するように変更する。

注意　☆ imin=0 を指定すると i 方向のデータは全て使用する。

☆ jmin=0 を指定すると j 方向のデータは全て使用する。

☆ kmin=0 を指定すると k 方向のデータは全て使用する。

☆ 本ルーチンを用いても fr_contour3d，fr_flow3dline，fr_isosurface での要素数は配列の全要素数を指定しなければならない。

☆ fr_contour3d，fr_flow3dline，fr_isosurface の様に，3次元描画ボックスと配列の境界が一致していると仮定して図形を描画するルーチンにおいては3次元描画ボックスと指定された配列領域の境界が一致しているように描く。

☆ 本ルーチンの指定は fr_gclear 等による描画ページ切り替えにより解除される。

3.8.4 小円の直径を変更する (fr_setcircle)

書式　call fr_setcircle(d)

引数　real*8 d

動作　fr_graph2d，fr_draw2d などで描画可能な小円の直径を d に変更する。

注意　☆ 指定する直径は描画座標での大きさである。

☆ d<0 を指定するとデフォルトサイズになる。デフォルトは d=6 である。

3.8.5 正多角形 (マーク) のサイズと頂点数を変更する (fr_setmark)

書式　call fr_setmark(r1,r2,n)

引数　real*8 r1,r2

integer n

動作　fr_graph2d，fr_draw2d などで小円を描くモードを使用したときの小円を正多角形に変更する。ここで，正多角形の中心からの半径は頂点が 1 個おきに等しく，2 個指定することが可能で，それぞれ r1，r2 で与える。n は正多角形の頂点数を与える。よって r1=r2 にすれば，正 n 角形となり，r1≠r2 にすれば，n/2 個の稜を持つ星形になる。

注意　☆ 指定する半径は描画座標での大きさである。

☆ n=0 にすれば，x 方向半径 r1，y 方向半径 r2 の楕円になる。

☆ デフォルトサイズはない。元の小円表示に戻したい場合には fr_setcircle を用いて小円の直径を設定し直す。

3.8.6 矢印のサイズを変更する (fr_setarrow)

書式　call fr_setarrow(al,ah)

引数　real*8 al,ah

動作　fr_vector2d，fr_vector3d などで描画可能な矢印の矢のサイズを変更する。

al>0 ならば，矢の先端の長さを al にする。

ah>0 ならば，矢の先端の幅を ah にする。

al<0 ならば，描画する矢の長さを|al|倍に拡大する。

ah<0 ならば，矢の先端を描く矢の長さの最小値を|ah|にする。

注意　☆ 指定する長さは描画座標での大きさである。

☆ al=0，ah<0 を指定するとデフォルトになる。デフォルトは al=8，ah=4 である。また，矢の拡大率は 1，先端を描く矢の最小長は 1 である。

3.8.7 棒グラフの基準値と方向を変更する (fr_setbar)

書式　call fr_setbar(base,mode)

引数　real*8 base

integer mode

動作　fr_graph2d，fr_pmgraph で描画可能な棒グラフの基準値を base にする。さらに，mode により，棒グラフの方向を指定する。

mode=0	x 方向に幅を持ち，y 方向に高さを持つ
mode=1	y 方向に幅を持ち，x 方向に高さを持つ

注意　☆ 指定する基準値は物理座標値である。

☆ デフォルトは base=0，mode=0 である。

3.8.8 座標軸を加工する (fr_setaxis)

書式　call fr_setaxis(mode)

引数　integer mode

動作　fr_frame2d または fr_pmframe の直前でコールすることにより，fr_frame2d または fr_pmframe で描く座標軸を対数軸にする。また，座標軸の数字を消去することができる。座標軸の種類は mode の 1 の位の値により以下の指定ができる。

mode の 1 の位 =0	x：線形	y：線形
mode の 1 の位 =1	x：対数	y：線形
mode の 1 の位 =2	x：線形	y：対数
mode の 1 の位 =3	x：対数	y：対数

また，mode の 10 の位の値により fr_frame2d，fr_pmframe，fr_frame3d ルーチンで座標軸を描く際に座標軸の数字を消去することができる。

mode の 10 の位 =0	x 軸：描画	y 軸：描画	z 軸：描画
mode の 10 の位 =1	x 軸：消去	y 軸：描画	z 軸：描画
mode の 10 の位 =2	x 軸：描画	y 軸：消去	z 軸：描画
mode の 10 の位 =3	x 軸：消去	y 軸：消去	z 軸：描画
mode の 10 の位 =4	x 軸：描画	y 軸：描画	z 軸：消去
mode の 10 の位 =5	x 軸：消去	y 軸：描画	z 軸：消去
mode の 10 の位 =6	x 軸：描画	y 軸：消去	z 軸：消去
mode の 10 の位 =7	x 軸：消去	y 軸：消去	z 軸：消去

ただし，10 の位の指定で 4 以上は 1 パラメータ図形群と 3 次元ルーチンの場合のみ有効である。

注意　☆ 本ルーチンによる座標軸指定は fr_frame2d，fr_pmframe，fr_frame3d の実行でクリアされるので，常に直前でコールする必要がある。

☆ 本ルーチンはあくまでも座標軸の加工指定だけである。fr_graph2d などの描画ルーチンに与える対数軸方向のデータは実際のデータの対数値を与える必要がある。

☆ mode の 1 の位の指定は，線形か対数かを 0・1 とし，yx の順で並べた 2 進数になっている。

☆ mode の 10 の位の指定は，描画か消去かを 0・1 とし，zyx の順で並べた 2 進数になっている。

3.8.9 デフォルトの２次元座標軸描画指定を変更する (fr_setframe2d)

書式　call fr_setframe2d(mode)

引数　integer mode

動作　fr_graph2d や fr_contour のように fr_frame2d(3.4.4) を使用しなくても自動スケール機能がある２次元図形描画ルーチンに対し，その座標軸描画モードを指定する。

引数 mode に fr_frame2d の座標軸指定における mode と同じ数値を与える。

注意　☆ mode=-1 を指定するとデフォルト座標軸描画指定に戻る。

3.8.10 デフォルトの３次元座標軸描画指定を変更する (fr_setframe3d)

書式　call fr_setframe3d(mode)

引数　integer mode

動作　fr_graph3d や fr_profile のように fr_frame3d(3.6.7) を使用しなくても自動スケール機能がある３次元図形描画ルーチンに対し，その座標軸描画モードを変更する。

引数 mode に fr_frame3d の座標軸指定における mode と同じ数値を与える。

注意　☆ mode=-1 を指定するとデフォルト座標軸描画指定に戻る。

3.8.11 等高線図の設定を変更する (fr_setcontour)

書式　call fr_setcontour(foff,df,mode)

引数　real*8 foff,df
integer mode

動作　等高線ルーチン (fr_contour など) の描画設定を変更する。mode の値により以下のような指定ができる。

mode=0	通常設定に戻す
mode=1	foff を高さのオフセット，df を高さの幅とした等高線を描く
mode=2	foff を最小値，df を最大値とした等高線を描く
mode=3	foff 以上を第 1 種色，foff 以下を第 2 種色とした等高線を描く
mode=4	等高線 (色指定 >0) ならば，foff に等しい高さの等高線のみ描く 等高色分布図 (色指定 ≤ -2) ならば，foff と foff+df で指定した範囲内にある区画は描画しない

注意　☆ fr_contour 等の等高線ルーチンでは，通常高さデータの最小値・最大値間を指定等分した等高線を描く。これに対し，本ルーチンを等高線ルーチンの前でコールすることにより，高さを指定した等高線を描くことができる

☆ mode=1 の時，等高線は n を整数としたとき df$\times n$+foff の高さを表示する

☆ mode=1 の時，df=0 を指定すると，等高線ルーチンの分割数指定に無関係に foff の位置に 1 本のみの等高線を描く

☆ mode=2 の時，分割数は等高線ルーチンで指定する。また，データが指定した最大値を超えたときには等高線を描かない

☆ 等高色分布図描画の際に，mode=4 を使用すると，ある高さの範囲にある領域を抜くことができる。これを用いれば，同じ描画ボックスに 2 個の等高色分布図を重ね合わせることができる。

☆ mode=4 で領域を抜く方法は，等高線ルーチンが指定する各小区画において，その区画の頂点の高さが全て指定した高さの範囲内にあれば抜き，そうでなければその区画を全て塗りつぶす。よって，抜いた領域の形状は，実際の高さの範囲で決まる領域よりも狭い。

3.8.12 等高線図用のカラーバーを描画する (fr_colorbar)

書式　call fr_colorbar(ix,iy,idx,idy,mode)

引数　integer ix,iy,idx,idy,mode

動作　HLS 色分け分布図を描画した際にその色分けに対応する数値を示すカラーバーを描画する。ここで (ix,iy) を基準点とし，x 方向，y 方向の幅をそれぞれ idx，idy とする。描画座標の指定は基礎図形ルーチン (3.7 参照) における基準点からの相対描画座標指定も可能である。

mode の値により以下のような指定ができる。

mode の 1 の位
- 0：数値は描かない
- 1：カラーバーの下 (x) または左 (y) に数値を描く
- 2：カラーバーの上 (x) または右 (y) に数値を描く

mode の 10 の位
- 0：ix，iy 共に絶対描画座標
- 1：ix は相対描画座標，iy は絶対描画座標
- 2：ix は絶対描画座標，iy は相対描画座標
- 3：ix，iy 共に相対描画座標

mode の 100 の位

0：基準点をカラーバーの左上隅点にする

1：基準点をカラーバーの左上隅点よりカラーバーの方向に半分ずらした中央点にする (カラーバーのセンタリング)

注意 ☆ idx≥idy なら x 方向のカラーバーとなり，idx<idy なら y 方向のカラーバーとなる。

☆ 本ルーチンは等高線表示するときに HLS 色指定にしたときのみ有効となる。それ以外ではコールしても無視される。

☆ 本ルーチンを使用するときには等高線モードが確定していなければならない。よって，fr_contour 等の等高線ルーチンの後でコールする必要がある。

☆ mode の 10 の位による座標指定が相対描画の時，y 方向の指定は実数座標の方向となる，すなわち，正の数を指定すると，画面の上方へ移動する。

3.8.13 疑似ライティングの光源を設定する (fr_setlight)

書式 `call fr_setlight(theta,phi,mode)`

引数 `real*8 theta,phi`

`integer mode`

動作 fr_surface など，疑似ライティングによる曲面表示ができるルーチンにおいて，光源の位置を変更する。

theta は光源の緯度，phi は光源の経度である。単位は「度」である。mode の値により方位角方向指定の制 0 ができる。

mode=0	phi は方位角の絶対値
mode=1	phi は視点の方位角を 0 とする相対値

注意 ☆ デフォルトでは，theta=90，phi=0，mode=0 である。すなわち光源は真上にある。

3.8.14 疑似ライティングの表面状態を設定する (fr_setsurface)

書式 `call fr_setsurface(amb,diff,nspec)`

引数 `real*8 amb,diff`

`integer nspec`

動作 fr_surface など，疑似ライティングによる曲面表示ができるルーチンにおいて，曲面の表面状態を変更する。

amb，diff，nspec で表面の光強度を設定する。

amb	:	K_A	0〜1	環境光強度率
diff	:	K_D	0〜1	拡散光強度率
	:	K_S	0〜1	反射光強度率 ($1-K_A-K_D$ に自動的に設定される)
nspec	:	n	≥0	表面反射係数

これらの係数により，以下の公式を用いて表面強度を決定する。

$$L = K_A + K_D(\mathbf{L}\cdot\mathbf{n}) + K_S(\mathbf{R}\cdot\mathbf{S})^n$$

ここで，**L** は光源への方向ベクトル，**n** は曲面の法線ベクトル，**R** は光源の全反射ベクトル，**S** は視点方向ベクトルである。

注意 ☆ amb+diff は 1 以下でなければならない。

☆ 環境光強度率 amb は，大きいほど曲面全体が一様な明るさになる。

☆ 拡散光強度率 diff は，小さいとつるつるになり，大きいほどざらざらの面になる。

☆ 表面反射係数 nspec が大きいとハイライトがきつくなって小さくなる。

☆ デフォルトでは，amb=0.1，diff=0.6，nspec=20 である。

3.8.15 1 次元配列データの最小値・最大値を計算する (fr_maxmin)

書式　call fr_maxmin(a,n,amin,amax)

引数　real*8 a(*),amin,amax
integer n

動作　n 要素の 1 次元配列 a(i) の最小値と最大値をそれぞれ，amin,amax に代入する。

注意　☆ fr_graph2d や fr_graph3d 等の自動スケール機能は x，y(，z) 全部同時にかかってしまう。逆に，fr_frame2d などは x，y (，z) 全部の範囲を指定しなければならない。本ルーチンは，どれか特定の方向のみ自動スケールをしたいユーザーのために用意したものである。

☆ n<0 の時は|n|個の 1 次元配列データ a(i) に amin，amax を加えたデータの最小値と最大値を amin，amax に代入する。

第４章　プログラム使用例

具体的プログラム使用例を以下に示します。ここで，コンパイラの自動倍精度化機能を利用することを仮定していますが，もししない場合には実数の宣言や定数の指定に注意してください。

(1)　fr_graph2d による２次元図形描画用ルーチンの作図例

```
program gsample01
   implicit none
   real x(101),y(101)                          !倍精度配列宣言
   integer i
   do i = 1, 101
      x(i) = 0.1*(i-1)                         !データの作成
      y(i) = sin(x(i))
   enddo
   call fr_ginit                               !初期化
   call fr_gfile('graph',0)                    !描画ファイル作成の宣言
   call fr_gclear                              !画面消去
   call fr_frame2d(0.0,10.0,-1.0,1.0,0)        ! x を 0 から 10 に y を-1 から 1 にスケール
                                               !して座標軸を描く
   call fr_xyname('x','sin(x)')                !座標に名前を書く
   call fr_graph2d(x,y,101,3,1)                !紫色の曲線で描画する
   call fr_gend                                !終了化
end program gsample01
```

fr_graph2d は，x 方向と y 方向のデータ配列を用意してコールするだけです。この例では fr_view2d 指定がありませんから，デフォルトの描画エリアが使われます。なお，fr_gclear を入れてありますが，初期状態では何も描かれていませんからこの場合には無視されます。fr_frame2d を使うと座標軸を描きます。

また，plt ファイルを作成するときは，fr_ginit の直後に fr_gfile を使用します。作成されるファイル名は拡張子.plt を付与して graph.plt となります。

(2)　fr_draw2d による２次元図形描画用ルーチンの作図例

```
program gsample02
   implicit none
   real  x                                     !倍精度宣言
   integer i
   call fr_ginit                               !初期化
   call fr_gfile('graph',0)                    !描画ファイル作成の宣言
   call fr_gclear                              !画面消去
   call fr_frame2d(0.0,10.0,-1.0,1.0,0)        ! x を 0 から 10 に y を-1 から 1 にスケール
                                               !して座標軸を描く
   call fr_xyname('x','cos(x)')                !座標に名前を書く
   call fr_draw2d(0.0,1.0,2,0)                 !描画開始点に点を移動する
   do i = 1, 100
      x = 0.1*i
      call fr_draw2d(x,cos(x),2,1)             !データ点を線分で結んでいく
   enddo
   call fr_gend                                !終了化
end program gsample02
```

fr_draw2d はメモリが少なく計算の遅いマシンで多くのデータを表示するときや，計算時間がかかるときにデータのふるまいを常に眺めながら計算するときに用います。ただし，データが多く，fr_draw2d を使う量が増えると処理速度が非常に遅くなります。できるだけ fr_graph2d を使ってください。

(3) 簡易アニメーション表示例

```
program gsample03
   implicit none
   integer, parameter :: ntmax=3000
   real  tt(0:ntmax),yy(0:ntmax)
   real  t,y(2),tmin,tmax,dt
   integer nt
   tmin = 0
   tmax = 20

   dt   = tmax/ntmax
   call fr_ginit
   t    = 0
   y(1) = 1
   y(2) = 0
   tt(0) = t
   yy(0) = y(1)
   call fr_ginit
   do nt = 1, ntmax
      call rkh4(2,t,dt,y)
      call fr_gclear                              !描画ページを切り替える
      call fr_frame2d(tmin,tmax,-1.0,1.0,0)       !以下は描画
      call fr_xyname('Time','Position')
      tt(nt) = t
      yy(nt) = y(1)
      call fr_graph2d(tt,yy,nt+1,3,1)             !曲線描画
      call fr_draw2d(t,y(1),3,-3)                 !小球描画
   enddo
   call fr_gend
end program gsample03

subroutine equat(t,y,dy)
   implicit none
   real  y(*),dy(*),t
   dy(1) = y(2)
   dy(2) = -0.2*y(2) - y(1)
end subroutine equat

subroutine rkh4(n,x,h,y)
   implicit none
   real, parameter :: c16=1.0/6.0
   real  x,h,y(n),z(n),k1(n),k2(n),k3(n),k4(n)
   integer n,i
   call equat(x,y,k1)
   do i = 1, n
      z(i) = y(i) + 0.5*h*k1(i)
   enddo
   call equat(x+0.5*h,z,k2)
   do i = 1, n
      z(i) = y(i) + 0.5*h*k2(i)
   enddo
   call equat(x+0.5*h,z,k3)
   do i = 1, n
      z(i) = y(i) + h*k3(i)
   enddo
   x = x + h
   call equat(x,z,k4)
   do i = 1, n
      y(i) = y(i) + c16*h*(k1(i) + 2.0*(k2(i) + k3(i)) + k4(i))
   enddo
end subroutine rkh4
```

デフォルトのディスプレイ表示では，図形描画を裏で行い，fr_gclear などのページ切り替えのタイミングで瞬時にウィンドウに表示するので，連続的に変化する図形を次々に描画すると簡易アニメーションになります。

(4) fr_setaxis を用いた対数軸指定

```
program gsample04
   implicit none
   integer, parameter :: nmax=1000
   real*8 x(0:nmax),yy(2),y1(0:nmax),y2(0:nmax),xmax,dx
   integer i

   xmax = 15
   dx   = xmax/nmax

   call fr_ginit
   call fr_gfile('log',0)
   call fr_setaxis(2)                                  !縦軸対数指定
   call fr_frame2d(0.0,xmax,-7.0,2.0,10)               ! y が 10^-7〜10^2 の対数軸描画
   call fr_xyname('x','y-log')

   x(0) = 0
   yy(1) = 1e-7
   yy(2) = 1e-7
   y1(0) = log10(yy(1))
   y2(0) = log10(yy(2))
   do i = 1, nmax
      x(i) = x(i-1)
      call rkh4(2,x(i),dx,yy)
      y1(i) = log10(yy(1))                             !対数データ作成
      y2(i) = log10(yy(2))
   enddo
   call fr_graph2d(x,y1,nmax+1,6,1)
   call fr_2dmove(x(nmax),y1(nmax),0)
   call fr_fpchars(0.0,0.0,' yellow',6,1,0,0,30)
   call fr_graph2d(x,y2,nmax+1,1,1)
   call fr_2dmove(x(nmax),y2(nmax),0)
   call fr_fpchars(0.0,0.0,' blue',1,1,0,0,30)
   call fr_gend
end program gsample04

subroutine equat(t,y,dy)
   implicit none
   real  y(*),dy(*),t
   dy(1) = 2.0*y(1)*(1.0 - 0.1*y(1))
   dy(2) = 3.0*y(2)*(1.0 - 10.0*y(2))
end subroutine equat
```

対数のグラフを描くには，まず fr_setaxis で対数軸の種類を指定しておいて，fr_frame2d で軸を描きます。このとき対数方向の範囲指定はその指数の最小値・最大値を与えます。

この後のグラフ描画では，対数方向のデータは元のデータの常用対数値を与える必要があります。また，この例では，線を２本描くので，線の横に文字を描いて区別しています。なお，微分方程式の解を計算するルンゲ・クッタ法のサブルーチン (rkh4) は (3) と同じです。

(5) fr_contour による等高線図例

```
program gsample05
   implicit none
   integer, parameter :: imax=100, jmax=100
   real  t(0:imax,0:jmax)
   real  x,y,xmin,xmax,ymin,ymax,dx,dy,func
   integer i,j
   xmin = -10
   xmax =  10
   ymin = -1.5
   ymax =  1.5
   dx   = (xmax-xmin)/imax
   dy   = (ymax-ymin)/jmax

   call fr_ginit
   call fr_gfile('cont',0)
   call fr_gclear
   call fr_frame2d(xmin,xmax,ymin,ymax,0)
   call fr_xyname('Real','Imaginary')
   do i = 0, imax
      x = dx*i + xmin
      do j = 0, jmax
         y = dy*j + ymin
         t(i,j) = func(x,y)                    !２次元データ作成
      enddo
   enddo
   call fr_contour(t,imax+1,jmax+1,20,2)       !等高線描画
   call fr_gend
end program gsample05

function func(x,y)                             !関数宣言の例
   implicit none
   complex c
   real  func,x,y
   c = sin(cmplx(x,y))
   func = abs(c)**2
end function func
```

等高線は２次元配列データを作成して fr_contour を使って描きます。デフォルトでは配列データの最小値・最大値を等分割した等高線を描きますが，局所的に急激な変化をするデータの場合には fr_setcontour で最小値・最大値を指定したり，データの対数値を配列に与えるなどの工夫が必要です。

(6) fr_aspect2d の使用例

```
program gsample06
   implicit none
   integer, parameter :: imax=100, jmax=100
   real  f(0:imax,0:jmax)
   real  xmin,xmax,ymin,ymax,dx,dy,x,y
   integer i,j

   xmin = -2
   xmax =  2
   ymin = -2
   ymax =  2
   dx   = (xmax - xmin)/imax
   dy   = (ymax - ymin)/jmax
   do j = 0, jmax
      do i = 0, imax
         x = dx*i+xmin
         y = dy*j+ymin
         f(i,j) = exp(-0.5*((x-1.0)**2+(y-0.5)**2)) + exp(-1.2*((x+1.5)**2+(y+0.5)**2))
      enddo
   enddo

   call fr_ginit
   call fr_gfile('aspect',0)
   call fr_aspect2d(1.0,1.0,0)                    !描画スケール比を 1 : 1 にする
   call fr_frame2d(xmin,xmax,ymin,ymax,0)
   call fr_xyname('Re','Im')
   call fr_contour(f,imax+1,jmax+1,20,-2)
   call fr_gend
end program gsample06
```

このように X : Y が 1 : 1 ということがわかっているときは fr_view2d で調節して描画エリアを指定するよりも fr_aspect2d を用いたほうが簡単です。なお，この例のように fr_contour の色指定を-2 にすると，等高色分布図になります。

(7) fr_tricontour による 3 角形メッシュデータを用いた等高線図例

```
program gsample07
   implicit none
   real  x(5),y(5),f(5)
   integer ndx(3,4)

   x(1) = 0
   y(1) = 10
   f(1) = 0
   x(2) = -10
   y(2) = 0
   f(2) = 2
   x(3) = 0
   y(3) = 2                                    !データ点の座標，データ入力
   f(3) = 5
   x(4) = 10
   y(4) = 0
   f(4) = 3
   x(5) = 0
   y(5) = -10
   f(5) = 1

   ndx(1,1) = 1
   ndx(2,1) = 2
   ndx(3,1) = 3
   ndx(1,2) = 1
   ndx(2,2) = 3
   ndx(3,2) = 4                                !3 角形構成データ入力
   ndx(1,3) = 2
   ndx(2,3) = 3
   ndx(3,3) = 5
   ndx(1,4) = 3
   ndx(2,4) = 4
   ndx(3,4) = 5

   call fr_ginit
   call fr_gfile('triangles',0)
   call fr_gclear
   call fr_view2d(50,50,590,350)
   call fr_aspect2d(1.0,1.0,0)
   call fr_tricontour(x,y,f,5,ndx,4,20,2,7)    !3 角形メッシュ等高線描画
   call fr_gend
end program gsample07
```

3 角形メッシュにおける等高線描画は，データ点の座標とその高さデータの入力，および点番号による 3 角形の構成データが必要です。この例では等高線は 2 (青色) で，3 角形メッシュは 7 (黒色) で描きます。

(8) fr_vector2d の使用例

```
program gsample08
   implicit none
   real  y(2),f(2),r
   integer i,j

   call fr_ginit
   call fr_gfile('vector',0)
   call fr_view2d(100,70,550,390)
   call fr_aspect2d(1.0,1.,0)
   call fr_frame2d(0.0,5.0,0.0,5.0,10)
   call fr_xyname('x','y')

   r = 0.05
   do i = 0, 20
      do j = 0, 20
         y(1) = 0.25*i
         y(2) = 0.25*j
         call react(y,f)
         call fr_vector2d(y(1),y(2),r*f(1),r*f(2),2,0)
      enddo
   enddo

   call fr_gend

end program gsample08

subroutine react(f,df)
   implicit none
   real  f(2),df(2)
   df(1) = 0.5 + f(1)*(f(1)*f(2) - 3.0)
   df(2) = f(1)*(2.0 - f(1)*f(2))
end subroutine react
```

fr_vector2d を使用するときは，ベクトルの長さを調節する必要があります。また，速度場のように多くのベクトルを一度に描く場合には，次の (9) のように fr_graph2d を使って一度に描くことをおすすめします。

(9) fr_graph2d によるベクトル場作図例

```
program gsample09
   implicit none
   real  x(2,441),z(2,441),y(2),f(2),r
   integer i,j,n

   call fr_ginit
   call fr_gfile('vec-fd',0)
   call fr_view2d(100,70,550,390)
   call fr_aspect2d(1.0,1.,0)
   call fr_frame2d(0.0,5.0,0.0,5.0,10)
   call fr_xyname('x','y')

   n = 0
   r = 0.05
   do i = 0, 20
      do j = 0, 20
         y(1) = 0.25*i
         y(2) = 0.25*j
         call react(y,f)
         n = n + 1
         x(1,n) = y(1)
         x(2,n) = y(1) + r*f(1)
         z(1,n) = y(2)
         z(2,n) = y(2) + r*f(2)
      enddo
   enddo

   call fr_graph2d(x,z,n*2,4,5)                !ベクトルの数の２倍を指定
   call fr_gend

end program gsample09

subroutine react(f,df)
   implicit none
   real  f(2),df(2)
   df(1) = 0.5 + f(1)*(f(1)*f(2) - 3.0)
   df(2) = f(1)*(2.0 - f(1)*f(2))
end subroutine react
```

fr_graph2d でベクトルを描く場合には，ベクトル１の始点，ベクトル１の終点，ベクトル２の始点，ベクトル２の終点，... という順で配列データを用意する必要があります。これには上記のプログラムのように x(2, ベクトルの数) のような２次元配列を用意すればわかりやすいと思います。ただし，プログラム中で注意しているようにデータ数の指定はベクトルの数の２倍 (指定点の数) ですから間違えないようにして下さい。

(10) fr_graph2d によるスプライン曲線作図例

```
program gsample10
   implicit none
   integer, parameter :: nmax=12
   real, parameter :: pi = 3.141592653589793, pi2 = pi*2
   real  x(0:nmax),y(0:nmax),r,th,dth
   real  xs(2,0:nmax),ys(2,0:nmax),xmax,ymax
   integer i,nx,ny

   xmax = 1.0
   ymax = 1.0
   nx   = 2
   ny   = 3
   r    = 1.03                                    !微係数の増倍係数

   call fr_ginit
   call fr_gfile('spline',0)
   call fr_aspect2d(1.0,1.0,0)
   call fr_frame2d(-xmax,xmax,-ymax,ymax,10)
   call fr_xyname('x-axis','y-datas')

   dth   = pi2/nmax
   do i = 0, nmax
      th = dth*i
      x(i) = sin(nx*th)
      y(i) = sin(ny*th)
   enddo

   call fr_graph2d(x,y,nmax+1,2,1)               !これは通常の折れ線描画

   do i = 0, nmax
      th = dth*i
      xs(1,i) = sin(nx*th)                       ! x 座標を指定
      xs(2,i) = nx*cos(nx*th)*dth*r              ! x 座標の微係数を指定
      ys(1,i) = sin(ny*th)                       ! y 座標を指定
      ys(2,i) = ny*cos(ny*th)*dth*r              ! y 座標の微係数を指定
   enddo

   call fr_graph2d(xs,ys,2*(nmax+1),4,6)         !スプライン曲線描画
   call fr_gend

end program gsample10
```

fr_graph2d でスプライン曲線を描く場合には，点 1 の座標，点 1 の微係数，点 2 の座標，点 2 の微係数，... という順で配列データを用意する必要があります。これには上記のプログラムのように x(2, 点の数) のような 2 次元配列を用意すればわかりやすいと思います。ただし，ベクトル場描画同様，データ数の指定は点の数の 2 倍 (指定点の数) ですから間違えないようにして下さい。

微係数は点番号 i で微分した値程度が適当ですが，この例のように正弦関数に近い曲線にするには，微係数をちょっと大きめにすると良いようです。スプライン曲線は関数形がわかっている曲線を少ない座標点で描くときに使うと効果的です。

(11) fr_flow2dline による流線図例

```
program gsample11
   implicit none
   real  y(2),xx(500),yy(500),fx(0:20,0:20),fy(0:20,0:20)
   integer i,j,n

   call fr_ginit
   call fr_gfile('flow',0)
   call fr_view2d(100,70,550,390)
   call fr_aspect2d(1.0,1.0,0)
   call fr_frame2d(0.0,5.0,0.0,5.0,10)
   call fr_xyname('x','y')

   do i = 0, 20
      do j = 0, 20
         y(1) = 0.25*i
         y(2) = 0.25*j
         call react(y,fx(i,j),fy(i,j))
      enddo
   enddo

   do n = 1, 10
      xx(1) = 0.4*n
      yy(1) = 0.2*n
      call fr_flow2dline(xx,yy,500,fx,fy,21,21,4,1.d-5,5)     !流線の描画
   enddo

   call fr_gend

end program gsample11

subroutine react(f,df1,df2)
   implicit none
   real  f(2),df1,df2
   df1 = 0.5 + f(1)*(f(1)*f(2) - 3.0)
   df2 = f(1)*(2.0 - f(1)*f(2))
end subroutine react
```

流線は，２次元ベクトルデータ配列を用いて fr_flow2dline で描きます。その際，描画用の 1 次元配列を用意し，その第 1 要素に流線の開始点座標を入れておきます。この例では，(fx(i,j),fy(i,j)) がベクトルデータ，(xx(k),yy(k)) が描画用 1 次元配列です。xx(1)，yy(1) に値を入力してから fr_flow2dline をコールしていることに注意して下さい。

(12) fr_flow2drgn と fr_flowspace 使用による流線図例

```
program gsample12
  implicit none
   real, parameter :: pi = 3.141592653589793
   integer, parameter :: ifmax=31, itmax=40, imax=30, jmax=40
   real  xx(500),yy(500),x,y,r,a0,th1
   real  bx(-imax:imax,0:jmax),by(-imax:imax,0:jmax)
   real  f(ifmax),th(itmax)
   integer i,j

   call fr_ginit
   call fr_aspect2d(1.0,1.0,0)
   call fr_frame2d(-0.25*imax,0.25*imax,0.0,0.25*jmax,0)
   call fr_xyname('x','y')
   do j = 0, jmax
      do i = -imax, imax
         x  = 0.25*i
         y  = 0.25*j
         call dipole(x,y,bx(i,j),by(i,j))
      enddo
   enddo
   r = 1
   do i = 1, ifmax
      th1 = pi*(i-1)/dble(ifmax-1)
      call dipole(r*cos(th1),r*sin(th1),x,y)
      f(i) = x*cos(th1)+y*sin(th1)
   enddo
   th(1)     = 0
   th(itmax) = pi
   call fr_flowspace(f,ifmax,th,itmax,a0,0)                    !等分布点の計算
   call fr_flow2drgn(0.0,0.25*imax,0.0,0.25*jmax)              !流線の描画領域の限定
   do i = 1, itmax/2
      xx(1) = r*cos(th(i))
      yy(1) = r*sin(th(i))
      call fr_flow2dline(xx,yy,500,bx,by,2*imax+1,jmax+1,4,1e-7,5)
   enddo
   call fr_flow2drgn(-0.25*imax,0.0,0.0,0.25*jmax)             !流線の描画領域の限定
   do i = itmax/2+1, itmax
      xx(1) = r*cos(th(i))
      yy(1) = r*sin(th(i))
      call fr_flow2dline(xx,yy,500,bx,by,2*imax+1,jmax+1,6,1e-7,-5)
   enddo
   call fr_gend

end program gsample12

subroutine dipole(x,y,bx,by)
   implicit none
   real  x,y,r2,r5,bx,by
   r2 = x*x+y*y
   if (r2 == 0.0) then
      bx = 0
      by = 0
    else
      r5 = r2**2.5
      bx = (3.0*x*x-r2)/r5
      by = 3.0*x*y/r5
   endif
end subroutine dipole
```

これは磁気双極子からの磁力線を描いたものですが，少し凝った描画をしています。磁力線は磁場の強さに反比例して間隔が狭くなる必要があります。このため，まず磁力線の初期位置の角度を fr_flowspace を使って計算しています。また，上例では磁力線が y 軸に対して対称になることがわかっているので，fr_flow2drgn を使って半分ずつ描いています。これは fr_flow2dline が領域外に出るか，ベクトルの長さが小さくならない限り描画が止まらないので，対称点で切るのが難しいからです。

(13) fr_pmgraph による 1 パラメータ図形群の作図例

```
program gsample13
   implicit none
   integer, parameter :: imax=200, jmax=50
   real  x(0:imax),y(0:imax),func
   real  xmin,xmax,ymin,ymax,zmin,zmax,dx,dz,z,t
   integer i,j

   xmin = -15
   xmax =  15
   ymin = -10
   ymax =  10
   zmin = -15
   zmax =  15
   dx   = (xmax-xmin)/imax
   dz   = (zmax-zmin)/jmax

   call fr_ginit
   call fr_gfile('param1',0)
   call fr_gclear
   call fr_pmframe(xmin,xmax,ymin,ymax,zmin,zmax,3)   !x,y,z をスケールして座標軸と枠を描く
   call fr_xyzname('x','y','t')                       !座標に名前を書く
   call fr_hclear(2)                                  !隠線消去指定

   do j = 0, jmax
      do i = 0, imax
         x(i) = dx*i + xmin
         y(i) = func(x(i),z,t)                        ! 1 次元データの作成
      enddo
      call fr_pmgraph(x,y,dz*j + zmin,imax+1,6,-1)    !隠線処理をしながら 1 次元データを描く
   enddo

   call fr_gend

end program gsample13

function func(x,z)                                    !関数宣言の例
   implicit none
   real  func,xx,x,z
   xx   = x*x + z*z
   func = 15.0*cos(sqrt(xx))*exp(-0.01*xx)
end function func
```

fr_pmgraph は 2 次元の fr_graph2d と使い方はほぼ同じです。ただし，fr_pmframe でパラメータの方向指定をすることと，パラメータ z を指定することが異なります。また，隠線消去するには，予め fr_hclear で隠線消去のモード指定をしておく必要があります。

(14) fr_pmdraw による 1 パラメータ図形群の作図例

```
program gsample14
   implicit none
   integer, parameter :: imax=200, jmax=50
   real  func,xmin,xmax,ymin,ymax,zmin,zmax,dx,dz,x,y,z
   integer i,j

   xmin = -15
   xmax =  15
   ymin = -10
   ymax =  10
   zmin = -15
   zmax =  15
   dx   = (xmax-xmin)/imax
   dz   = (zmax-zmin)/jmax

   call fr_ginit
   call fr_gfile('param2',0)
   call fr_gclear
   call fr_pmframe(xmin,xmax,ymin,ymax,zmin,zmax,3)   !x,y,z をスケールして座標軸と枠を描く
   call fr_xyzname('x','y','t')                       !座標に名前を書く
   call fr_hclear(1)                                  !隠線消去指定

   do j = 0, jmax
      z = dz*j + zmin
      call fr_pmset(z)                                !パラメータの設定
      y = func(xmin,z)
      call fr_pmdraw(xmin,y,6,0)                      !現在点の移動
      do i = 1, imax
         x = dx*i + xmin
         y = func(x,z)
         call fr_pmdraw(x,y,6,-1)                     !隠線消去しながら曲線を描く
      enddo
   enddo

   call fr_gend

end program gsample14

function func(x,z)
   implicit none
   real  func,xx,x,z
   xx   = x*x + z*z
   func = 15.0*cos(sqrt(xx))*exp(-0.01*xx)
end function func
```

fr_pmdraw はデータの数が多く，計算時間がかかるときに，データのふるまいを常に眺めながら計算するときに用います。どちらの場合でも z が小さい方から大きい方へ描いていることに注意してください。fr_pmdraw を頻繁に使うと処理速度が遅くなります。できるだけ fr_pmgraph を使ってください。

(15) fr_profile による鳥瞰図の作図 (1 パラメータ図形群ルーチン使用)

```
program gsample15
   implicit none
   integer, parameter :: imax=100, jmax=100
   real  t(0:imax,0:jmax),func
   real  xmin,xmax,ymin,ymax,tmin,tmax,dx,dy,x,y
   integer i,j

   xmin = -15
   xmax =  15
   ymin = -10
   ymax =  10
   tmin = -1.5
   tmax =  1.5
   dx   = (xmax-xmin)/imax
   dy   = (ymax-ymin)/jmax

   call fr_ginit
   call fr_gfile('prof',0)
   call fr_pmview(50,180,400,390,500,280)              !描画ボックスの決定
   call fr_pmframe(xmin,xmax,tmin,tmax,ymin,ymax,3)    !x,t,y をスケールして座標軸と枠を描く
   call fr_xyzname('x','z','y')                        !座標に名前を書く
   do j = 0, jmax
      y = dy*j + ymin
      do i = 0, imax
         x = dx*i + xmin
         t(i,j) = func(x,y)                            ! 2次元データ作成
      enddo
   enddo

   call fr_profile(t,imax+1,jmax+1,5,1)                !鳥瞰図の作図
   call fr_gend

end program gsample15

function func(x,y)
   implicit none
   real  func,xx,x,y
   xx   = x*x + y*y
   func = 2*cos(sqrt(xx))*exp(-0.01*xx)
end function func
```

1 パラメータ図形群ルーチンを用いて鳥瞰図を描画するには fr_pmview と fr_pmframe の指定をし，2次元配列データを作成して fr_profile をコールします。その際この例のように，データは y 方向に与える必要があります。

なお，1 パラメータ図形群ルーチンを利用すると，3次元図形が歪んで見えます。鳥瞰図を 3 次元的に正しく表現するには，次の (16) のように 3 次元図形描画ルーチンを使用して下さい。

(16) fr_profile による鳥瞰図の作図 (3 次元図形描画ルーチン使用)

```
program gsample16
   implicit none
   integer, parameter :: imax=100, jmax=100
   real  t(0:imax,0:jmax),func
   real  xmin,xmax,ymin,ymax,tmin,tmax,dx,dy,x,y
   integer i,j

   xmin = -15
   xmax =  15
   ymin = -10
   ymax =  10
   tmin = -1.5
   tmax =  1.5
   dx   = (xmax-xmin)/imax
   dy   = (ymax-ymin)/jmax

   call fr_ginit
   call fr_gfile('prof3',0)
   call fr_gclear
   call fr_angle3d(30.0,40.0)
   call fr_aspect3d(1.0,1.0,1.0,2)
   call fr_frame3d(xmin,xmax,ymin,ymax,tmin,tmax,3)
   call fr_xyzname('x','y','z')

   do j = 0, jmax
      y = dy*j + ymin
      do i = 0, imax
         x = dx*i + xmin
         t(i,j) = func(x,y)                    !２次元データ作成
      enddo
   enddo

   call fr_profile(t,imax+1,jmax+1,5,1)        !鳥瞰図の作図
   call fr_gend

end program gsample16

function func(x,y)
   implicit none
   real  func,xx,x,y
   xx   = x*x + y*y
   func = 2*cos(sqrt(xx))*exp(-0.01*xx)
end function func
```

3次元図形描画ルーチンを用いて鳥瞰図を描画するには fr_angle3d, fr_aspect3d, fr_frame3d の指定をし, 2次元配列データを作成して fr_profile をコールします。1パラメータ図形群の場合と異なり, データは z 方向に与えます。fr_aspect3d の mode を 2 にしておけば, 表示する 2 次元空間は座標の比, データは 3 次元描画ボックス比で描画できます。

(17) fr_graph3d による３次元曲線作図例

```
module parm
   real  sig,r,b
end module parm

program gsample17
   use parm
   implicit none
   integer, parameter :: nmax=5000
   real  xs(nmax),ys(nmax),zs(nmax)
   real  f(3),xmin,xmax,ymin,ymax,zmin,zmax,dt,t
   integer i

   xmin  = -20
   xmax  = 20
   ymin  = -20
   ymax  = 20
   zmin  = 0
   zmax  = 40
   dt    = 0.01
   sig   = 10
   r     = 28
   b     = 8.0/3.0
   t     = 0
   xs(1) = 0.1
   ys(1) = 0.1
   zs(1) = 0.1
   f(1)  = xs(1)
   f(2)  = ys(1)
   f(3)  = zs(1)
   do i = 2, nmax
      call rkh4(3,t,dt,f)
      xs(i) = f(1)
      ys(i) = f(2)
      zs(i) = f(3)
   enddo

   call fr_ginit
   call fr_gfile('lorentz',0)
   call fr_gclear
   call fr_angle3d(30.0,40.0)
   call fr_aspect3d(1.0,1.0,1.0,0)
   call fr_frame3d(xmin,xmax,ymin,ymax,zmin,zmax,3)
   call fr_xyzname('x','y','z')
   call fr_graph3d(xs,ys,zs,nmax,5,1)
   call fr_gend

end program gsample17

subroutine equat(x,f,df)
   use parm
   implicit none
   real  f(*),df(*),x
   df(1) = sig*(f(2) - f(1))
   df(2) = (r - f(3))*f(1) - f(2)
   df(3) = f(1)*f(2) - b*f(3)
end subroutine equat
```

これは有名なローレンツアトラクターを描くプログラムです。ルンゲ・クッタ法のサブルーチン (rkh4) は (3) と同じです。このプログラムは fr_draw3d を使っても描けますが，処理は fr_graph3d を使う方が高速です。

(18) fr_rotate と fr_gpause の使用例

```
program gsample18
   use parm
   implicit none
   integer, parameter :: nmax=5000
   real  xs(nmax),ys(nmax),zs(nmax)
   real  f(3),xmin,xmax,ymin,ymax,zmin,zmax,phi,dt,t
   integer i

   xmin = -20
   xmax = 20
   ymin = -20
   ymax = 20
   zmin = 0
   zmax = 40
   dt  = 0.01
   sig = 10
   r   = 28
   b   = 8.0/3.0
   t     = 0
   xs(1) = 0.1
   ys(1) = 0.1
   zs(1) = 0.1
   f(1)  = xs(1)
   f(2)  = ys(1)
   f(3)  = zs(1)
   do i = 2, nmax
      call rkh4(3,t,dt,f)
      xs(i) = f(1)
      ys(i) = f(2)
      zs(i) = f(3)
   enddo

   call fr_ginit
   call fr_gfile('lorentz2',0)
   call fr_gclear
   call fr_angle3d(30.0,60.0)
   call fr_aspect3d(1.0,1.0,1.0,0)
   call fr_frame3d(xmin,xmax,ymin,ymax,zmin,zmax,3)
   call fr_xyzname('x','y','z')
   call fr_graph3d(xs,ys,zs,nmax,5,1)            !最初の視点での図形描画

   do i = 3, 14
     call fr_gpause                              !一時停止
     phi = i*30.0
     call fr_gclear                              !画面を消去
     call fr_rotate(30.0,phi,3)                  !縮尺を変えずに視点を変更する
     call fr_xyzname('x','y','z')
     call fr_graph3d(xs,ys,zs,nmax,5,1)          !その視点で図形描画
   enddo

   call fr_gend

end program gsample18
```

これは (17) のローレンツアトラクターのメインプログラムを変更したものです。このように fr_rotate を用いて描き直せば，縮尺率が同じですから同じ図形を別の角度から見るときに便利です。ここでは fr_gpause を使って描画 1 回ごとに停止していますが，停止せずに実行を継続すれば物体が回転する様子をリアルタイムで見ることもできます。

(19) fr_surface による曲面の作図例

```
program gsample19
   implicit none
   real, parameter :: pi = 3.141592653589793
   integer, parameter :: imax=61, jmax=21
   real  u(imax,jmax),v(imax,jmax),w(imax,jmax)
   real  xmin,xmax,ymin,ymax,zmin,zmax,ds,dt,t,s
   integer i,j

   xmin = -1.2
   xmax =  1.2
   ymin = -1.2
   ymax =  1.2
   zmin = -0.5
   zmax =  0.5
   ds   = 2*pi/(imax-1)
   dt   = 2*pi/(jmax-1)

   do j = 1, jmax
      t = dt*j
      do i = 1, imax
         s      = ds*i
         u(i,j) = (1+0.25*cos(t))*cos(s)
         v(i,j) = (1+0.25*cos(t))*sin(s)
         w(i,j) = 0.25*sin(t)
      enddo
   enddo

   call fr_ginit
   call fr_gfile('torus',0)
   call fr_gclear
   call fr_angle3d(30.0,-60.0)
   call fr_aspect3d(1.0,1.0,1.0,0)                    !アスペクト比の決定
   call fr_frame3d(xmin,xmax,ymin,ymax,zmin,zmax,3)   !x,y,z をスケールして座標軸と枠を描く
   call fr_xyzname('x','y','z')                       !軸に名称を付ける
   call fr_surface(u,v,w,imax,jmax,5,4)               !パラメータ曲面の描画
   call fr_gend

end program gsample19
```

曲面描画は fr_view3d, fr_angle3d, fr_aspect3d, fr_frame3d で描画ボックスと視点を指定し，3 個の 2 次元配列データを作成して fr_surface をコールします。

(20) fr_surface による複数の曲面で構成されている図形の作図例

```
program gsample20
   implicit none
   real, parameter :: pi = 3.141592653589793, pi2 = pi*2
   integer, parameter :: imax=51, jmax=51
   real  x(imax,jmax),y(imax,jmax),z(imax,jmax)
   real  u,v,du,dv
   integer i,j

   du = pi2/(imax-1)
   dv = pi2/(jmax-1)

   do j = 1, jmax
      v = dv*j
      do i = 1, imax
         u      = du*(i-1)
         x(i,j) = (1+0.25*cos(v))*cos(u)
         y(i,j) = (1+0.25*cos(v))*sin(u)
         z(i,j) = 0.25*sin(v)
      enddo
   enddo

   call fr_ginit
   call fr_gfile('torus2',0)
   call fr_project(5.0,1)
   call fr_margin3d(50,20,20,80)                    !マージン指定による描画エリア指定
   call fr_angle3d(30.0,-60.0)
   call fr_aspect3d(1.0,1.0,1.0,0)
   call fr_xyzname('x-ax','y-ax','z-ax')
   call fr_frame3d(-1.3,2.3,-1.3,1.3,-0.5,0.5,3)
!   call fr_set2drgn(1,26,0,35)                     ! ... (1)
   call fr_surface(x,y,z,imax,jmax,3,10)            ! 1 枚目は mode=10 を指定する

   do j = 1, jmax
      v = dv*j
      do i = 1, imax
         u      = du*(i-1)
         z(i,j) = (1+0.25*cos(v))*cos(u)
         x(i,j) = 1+(1+0.25*cos(v))*sin(u)
         y(i,j) = 0.25*sin(v)
      enddo
   enddo
   call fr_surface(x,y,z,imax,jmax,4,3)             ! 2 枚目は通常の指定
   call fr_gend

end program gsample20
```

複数の曲面から構成されている図形を隠線消去して描画する時は fr_surface を曲面の数だけコールします。ポイントは，最後の fr_surface 以外は mode=10 でコールすることです。曲面の描画色は，それぞれの fr_surface の指定で決まるので，この例では 2 枚の色違いのリングがからんでいる図になります。また，3 次元表示の描画エリア指定は，fr_view3d ルーチンより fr_margin3d ルーチンで行う方が便利です。

なお，(1) のコメントをはずして実行するとどうなるのかも一度試してみてください。

(21) fr_trisurface による３角形メッシュデータを用いた曲面の作図例

```
program gsample21
   implicit none
   integer, parameter :: imax=20, jmax=20, npmax=(imax+1)*(jmax+1)
   real  x(npmax),y(npmax),z(npmax)
   integer node(3,npmax*2)
   integer i,j,np,ns,np0

   np = 0
   do j = 0, jmax
      do i = 0, imax
         np = np + 1
         x(np) = (i-imax/2)*0.9
         z(np) = (j-jmax/2)*0.47
         y(np) = 0.2*(z(np)**3-1.5*x(np)*z(np))*imax/(4*imax-3*i)
      enddo
   enddo

   ns  = 0
   np0 = 0
   do j = 0, jmax-1
      do i = 0, imax-1
         ns  = ns + 1
         np0 = np0 + 1
         node(1,ns) = np0
         node(2,ns) = np0 + imax + 2
         node(3,ns) = np0 + imax + 1
         ns  = ns + 1
         node(1,ns) = np0
         node(2,ns) = np0 + 1
         node(3,ns) = np0 + imax + 2
      enddo
      np0 = np0 + 1
   enddo

   call fr_ginit
   call fr_gfile('catastroph',0)
   call fr_gclear
   call fr_project(5.0,1)
   call fr_margin3d(10,20,10,80)
   call fr_angle3d(30.0,10.0)
   call fr_aspect3d(1.0,1.0,1.0,0)
   call fr_xyzname('x-ax','y-ax','z-ax')
   call fr_trisurface(x,y,z,np,node,ns,801,6)
   call fr_gend

end program gsample21
```

fr_trisurface を用いれば，任意の３角形メッシュデータを用いた曲面を描くことができます。このとき，mode=6 を用いれば，メッシュ内を強度補間した疑似ライティング表現となります。この例 (801) のように，色指定の 100 の位で第２種色を指定すれば，メッシュを同時に描くこともできます。

(22) fr_set2drgn の使用例

```
program gsample22
   implicit none
   integer, parameter :: imax=101, jmax=101
   real  t(imax,jmax),func
   real  xmin,xmax,ymin,ymax,tmin,tmax,dx,dy,x,y
   integer i,j

   xmin = -15
   xmax =  15
   ymin = -10
   ymax =  10
   tmin = -2
   tmax =  2
   dx   = (xmax-xmin)/imax
   dy   = (ymax-ymin)/jmax

   call fr_ginit
   call fr_gfile('region',0)
   call fr_angle3d(30.0,40.0)
   call fr_aspect3d(1.0,1.0,1.,1)
   call fr_frame3d(xmin,xmax,ymin,ymax,tmin,tmax,3)
   call fr_xyzname('x','y','t')

   do j = 1, jmax
      y = dy*(j-1) + ymin
      do i = 1, imax
         x = dx*(i-1) + xmin
         t(i,j) = func(x,y)
      enddo
   enddo

   call fr_set2drgn(3,30,50,100)
   call fr_profile(t,imax,jmax,5,1)          !imax, jmax は宣言と同じ
   call fr_gend

end program gsample22

function func(x,y)
   implicit none
   real  func,xx,x,y
   xx   = x*x + y*y
   func = 2*cos(sqrt(xx))*exp(-0.01*xx)
end function func
```

2次元データの一部を fr_set2drgn で指定して fr_profile で描いています。ここで注意して欲しいのは，fr_profile での配列要素数の指定は宣言文と同じにしてあることです。さもなくば全く違った図形が現われます。

(23) fr_contour3d による 3 次元データの断面等高線作図例

```
program gsample23
   implicit none
   integer, parameter :: imax=20, jmax=15, kmax=10
   real  f(0:imax,0:jmax,0:kmax)
   real  xmin,xmax,ymin,ymax,zmin,zmax,dx,dy,dz,x,y,z
   integer i,j,k

   xmin = -2
   xmax =  2
   ymin = -2
   ymax =  2
   zmin = -2
   zmax =  2
   dx   = (xmax - xmin)/imax
   dy   = (ymax - ymin)/jmax
   dz   = (zmax - zmin)/kmax
   do k = 0, kmax
      z=dz*k+zmin
      do j = 0, jmax
         y=dy*j+ymin
         do i = 0, imax
            x=dx*i+xmin
            f(i,j,k)=exp(-0.5*((x-0.5)**2+(y-0.5)**2+(z-0.5)**2)) &
                    +exp(-1.2*((x+1.5)**2+(y+1.5)**2)-0.1*(z+1.5)**2)
         enddo
      enddo
   enddo

   call fr_ginit
   call fr_gfile('cont31',0)
   call fr_project(5.0,1)
   call fr_margin3d(10,20,10,80)
   call fr_angle3d(20.0,130.0)
   call fr_aspect3d(1.0,1.0,1.0,1)
   call fr_frame3d(xmin,xmax,ymin,ymax,zmin,zmax,3)
   call fr_xyzname('x-axis','y-axis','z-axis')
   call fr_contour3d(f,imax+1,jmax+1,kmax+1,-0.5,20,-2,403)
   call fr_colorbar(100,230,15,100,1)
   call fr_gend

end program gsample23
```

fr_contour3d は 3 次元データを作成して，どの方向のどの位置に何枚描くかを指定します。この例では z 方向に 4 枚描くことになります。

(24) fr_contour3d による２次元等高線の３次元表示

```
program gsample24
   implicit none
   integer, parameter :: imax=100, jmax=100
   real  f(0:imax,0:jmax)
   real  xmin,xmax,ymin,ymax,zmin,zmax,dx,dy,x,y
   integer i,j

   xmin = -2
   xmax =  2
   ymin = -2
   ymax =  2
   zmin = -2
   zmax =  2
   dx   = (xmax - xmin)/imax
   dy   = (ymax - ymin)/jmax
   do j = 0, jmax
      do i = 0, imax
         x = dx*i+xmin
         y = dy*j+ymin
         f(i,j) = exp(-0.5*((x-1.0)**2+(y-0.5)**2)) &
                + exp(-1.2*((x+1.5)**2+(y+0.5)**2))
      enddo
   enddo

   call fr_ginit
   call fr_gfile('cont32',0)
   call fr_margin3d(10,40,10,60)
   call fr_angle3d(30.0,40.0)
   call fr_aspect3d(1.0,1.0,1.0,1)
   call fr_frame3d(xmin,xmax,ymin,ymax,zmin,zmax,3)
   call fr_xyzname('x-axis','y-axis','z-axis')
   call fr_contour3d(f,1,imax+1,jmax+1,0.0,20,5,0)      !最後の引数を 0 にする
   call fr_gend

end program gsample24
```

fr_contour3d はこの例のようにある次元を 1 にすれば，その方向に一様にデータがあるとして断面の等高線を描きます。即ち，２次元データの等高線を３次元空間に表示する働きをします。なお，このとき最後の引数は 0 にして下さい。

(25) fr_slices による３次元データの複数の断面等高線作図例

```
program gsample25
   implicit none
   integer, parameter :: imax=30, jmax=20, kmax=20
   real  f(0:imax,0:jmax,0:kmax),td(3)
   real  xmin,xmax,ymin,ymax,zmin,zmax
   real  x,y,z,dx,dy,dz
   integer nu(3),i,j,k

   td(1) = 0
   td(2) = -0.5
   td(3) = -1
   nu(1) = 3
   nu(2) = 1
   nu(3) = 2

   xmin = -3
   xmax =  3
   ymin = -2
   ymax =  2
   zmin = -2
   zmax =  2
   dx   = (xmax - xmin)/imax
   dy   = (ymax - ymin)/jmax
   dz   = (zmax - zmin)/kmax
   do k = 0, kmax
      z = dz*k+zmin
      do j = 0, jmax
         y = dy*j+ymin
         do i = 0, imax
            x = dx*i+xmin
            f(i,j,k)=exp(-0.5*((x-0.5)**2+(y-0.5)**2+(z-0.5)**2)) &
                    +exp(-1.2*((x+1.5)**2+(y+1.5)**2)-0.1*(z+1.5)**2)
         enddo
      enddo
   enddo

   call fr_ginit
   call fr_gfile('slices',0)
   call fr_project(5.0,1)
   call fr_margin3d(10,20,10,80)
   call fr_angle3d(20.0,70.0)
   call fr_aspect3d(1.0,1.0,1.0,0)
   call fr_frame3d(xmin,xmax,ymin,ymax,zmin,zmax,3)
   call fr_xyzname('x-axis','y-axis','z-axis')
   call fr_slices(f,imax+1,jmax+1,kmax+1,td,nu,20,-2,0)
   call fr_colorbar(60,180,15,100,1)
   call fr_gend

end program gsample25
```

fr_slices を使うと，複数の交差した断面図を描くことができます。この例では x 方向に３枚，y 方向に１枚，z 方向に２枚描きます。

(26) fr_isosurface による３次元スカラーデータの等値面表示

```
program gsample26
   implicit none
   real, parameter :: pi2 = 2*3.141592653589793
   integer, parameter :: imax=10, jmax=10, kmax=50
   real  f(-imax:imax,-jmax:jmax,0:kmax)
   real  r0,f0,rc,rs
   integer i,j,k

   r0 = 5.5
   f0 = 0.75
   do k = 0, kmax
      rc = r0*cos(pi2*k/kmax)
      rs = r0*sin(pi2*k/kmax)
      do j = -jmax, jmax
         do i = -imax, imax
            f(i,j,k) =  exp(-((i-rc)*(i-rc)+(j-rs)*(j-rs))/20.0) &
                       +exp(-((i+rc)*(i+rc)+(j+rs)*(j+rs))/40.0)
         enddo
      enddo
   enddo

   call fr_ginit
   call fr_gfile('iso',0)
   call fr_project(5.0,21)
   call fr_angle3d(30.0,-30.0)
   call fr_aspect3d(1.0,1.0,1.0,0)
   call fr_frame3d(-dble(imax),dble(imax),-dble(jmax),dble(jmax),0.0,dble(kmax),4)
   call fr_xyzname('x-ax','y-ax','z-ax')
   call fr_isosurface(f,2*imax+1,2*jmax+1,kmax+1,f0,3,6)
   call fr_gend

end program gsample26
```

fr_isosurface を使えば，３次元スカラーデータから任意の閾値 f0 を持つ等値面を描くことができます。この例のように mode=6 にすると強度補間を用いた擬似ライティング表示になります。等値面表示は補間点によっては３角形が詰まって区別しにくくなるので擬似ライティングにした方が良いでしょう。mode=6 を用いれば，mode=5 よりもなめらかな等値面が表現できます。

なお，本例のように，fr_project の第２引数の 10 の位を 2 にすると，y 軸が描画空間の上下方向になった図になります。

(27) fr_flow3dline による円筒対称座標系での流線描画

```
program gsample27
   implicit none
   integer, parameter :: itmax=10, imax=20, kmax=50, nmax0=5000
   real  xx(nmax0),yy(nmax0),zz(nmax0)
   real  br(0:imax,0:kmax),bt(0:imax,0:kmax),bz(0:imax,0:kmax)
   real  f(0:imax,0:kmax)
   real  xmin,xmax,ymin,ymax,zmin,zmax,dr,dz,r,z,om
   integer i,k,nmax

   xmax = 0.25*imax
   xmin = -xmax
   ymax = xmax
   ymin = -ymax
   zmin = 0
   zmax = 10
   om   = -5

   call fr_ginit
   call fr_gfile('flow3d',0)
   call fr_project(5.0,1)
   call fr_angle3d(30.0,-60.0)
   call fr_aspect3d(1.0,1.0,1.0,0)
   call fr_frame3d(xmin,xmax,ymin,ymax,zmin,zmax,3)
   call fr_xyzname('x-ax','y-ax','z-ax')

   dr   = xmax/imax
   dz   = (zmax-zmin)/kmax
   do k = 0, kmax
      z = dz*k+zmin
      do i = 0, imax
         r = dr*i
         f(i,k)  = exp(-0.3*((r-2)**2+(z-zmax/2)**2))
         br(i,k) = 0
         bt(i,k) = r*om*f(i,k)
         bz(i,k) = 1
      enddo
   enddo

   call fr_setgeometry(2)
   dr = xmax/(itmax+1)
   do i = 1, itmax
      r     = dr*i
      xx(1) = r
      yy(1) = 0
      zz(1) = zmin+0.1
      nmax  = nmax0
      call fr_flow3dline(xx,yy,zz,nmax,br,bt,bz,imax+1,1,kmax+1,5,1e-7,3)
   enddo

   call fr_gend

end program gsample27
```

fr_flow3dline を使えば，３次元ベクトル場データを用いて流線を描くことができます。さらに，ベクトル場が $(f_r(r,z), f_\theta(r,z), f_z(r,z))$ の様な円筒対称形データの場合でもこの例のように fr_setgeometry を用いれば円筒対称形を仮定した流線を描くことができます。ただし，配列 xx，yy，zz に代入されるデータは直交座標系です。

(28) fr_gfilegif による GIF アニメーションの作成

```
program gsample28
   implicit none
   real  y(2),f(2),r
   integer i,j,nt

   call fr_ginit
   call fr_gwsize(500,400,0,0)                    !描画サイズを 横 500 縦 400 にする
   call fr_giffile('gifanime',3,1)

   do nt = 1, 20
      call fr_gclear
      call fr_margin2d(50,30,50,70)               !描画エリアをマージンで指定する
      call fr_aspect2d(1.0,1.0,0)
      call fr_frame2d(0.0,5.0,0.0,5.0,10)
      call fr_xyname('x','y')

      r = 0.05
      do i = 0, 20
         do j = 0, 20
            y(1) = 0.25*i
            y(2) = 0.25*j
            call react(y,f,real(nt)/1.5)
            call fr_vector2d(y(1),y(2),r*f(1),r*f(2),2,0)
         enddo
      enddo
   enddo

   call fr_gend

end program gsample28

subroutine react(f,df,p)
   implicit none
   real  f(2),df(2),p
   df(1) = 0.5 + f(1)*(f(1)*f(2) - p)
   df(2) = f(1)*(2.0 - f(1)*f(2))
end subroutine react
```

fr_gfilegif を使えば，簡単に GIF アニメーションを作ることができます。本例では，背景が黒色の GIF アニメーションを作成します。また fr_gwsize を使ってイメージのサイズを指定していますが，サイズを試行錯誤で決める時には描画エリアの指定を fr_margin2d ルーチンで行うと便利です。

(29)　*CANVAS* を用いた複数の GIF イメージの作成

```
program gsample29
   implicit none
   integer, parameter :: imax=20, jmax=20
   real  f(0:imax,0:jmax),xmin,xmax,ymin,ymax,dx,dy,x,y
   integer i,j

   xmin = -3
   xmax =  3
   ymin = -3
   ymax =  3
   dx   = (xmax - xmin)/imax
   dy   = (ymax - ymin)/jmax

   call fr_ginit
   call fr_gwsize(500,500,0,0)
   call fr_gifimage(1,'canvas1',1)
   call fr_gwsize(400,400,0,0)
   call fr_gifimage(2,'canvas2',2)
   call fr_gwsize(350,350,0,0)
   call fr_gifimage(3,'canvas3',3)

   do j = 0, jmax
      do i = 0, imax
         x = dx*i+xmin
         y = dy*j+ymin
         f(i,j) = exp(-0.5*((x-1)**2+y**2))-exp(-1.2*((x-1.5)**2+(y-0.5)**2))
      enddo
   enddo

   call fr_canvas(1)
   call fr_margin2d(50,30,50,70)
   call fr_aspect2d(1.0,1.0,0)
   call fr_frame2d(xmin,xmax,ymin,ymax,0)
   call fr_xyname('re','im')
   call fr_contour(f,imax+1,jmax+1,20,-2)

   call fr_canvas(2)
   call fr_margin2d(50,30,50,70)
   call fr_aspect2d(1.0,1.0,0)
   call fr_frame2d(xmin,xmax,ymin,ymax,0)
   call fr_xyname('re','im')
   call fr_contour(f,imax+1,jmax+1,20,-2)

   call fr_canvas(3)
   call fr_margin2d(50,30,50,70)
   call fr_aspect2d(1.0,1.0,0)
   call fr_frame2d(xmin,xmax,ymin,ymax,0)
   call fr_xyname('re','im')
   call fr_contour(f,imax+1,jmax+1,20,-2)

   call fr_gend

end program gsample29
```

CANVAS を使えば，複数の性質の異なる GIF イメージを同時に作ることができます。本例では，サイズやカラーモードの異なる 3 つの GIF イメージ，`canvas10001.gif`，`canvas20001.gif`，`canvas30001.gif` ができます。なお，サイズが同じで良ければ複数の *CANVAS* を用いる必要はなく，単一の *CANVAS* で描画ページを切り替えて描いていけば `OK` です。また，本例のようにディスプレイ表示も行った場合にはそれぞれ別のウィンドウに表示されますので，同時に 3 枚の図形を見ることができます。

(30) *CANVAS* を用いた複数の GIF アニメーションの作成

```
program gsample30
   implicit none
   integer, parameter :: imax=20, jmax=20
   real  f(0:imax,0:jmax),xmin,xmax,ymin,ymax,dx,dy,x,y
   integer i,j,nt

   xmin = -3
   xmax =  3
   ymin = -3
   ymax =  3
   dx   = (xmax - xmin)/imax
   dy   = (ymax - ymin)/jmax

   call fr_ginit
   call fr_gwsize(500,500,0,0)
   call fr_gifanimation(-1,'canvas1',1,0,0)
   call fr_gwsize(400,400,0,0)
   call fr_gifanimation(-2,'canvas2',2,0,0)
   call fr_gwsize(350,350,0,0)
   call fr_gifanimation(-3,'canvas3',3,0,0)

   do nt = -10, 10
      do j = 0, jmax
         do i = 0, imax
            x=dx*i+xmin
            y=dy*j+ymin
            f(i,j)=exp(-0.5*((x-0.25*nt)**2+y**2))-exp(-1.2*((x-1.5)**2+(y-0.5)**2))
         enddo
      enddo

      call fr_canvas(1)
      call fr_margin2d(50,30,50,70)
      call fr_aspect2d(1.0,1.0,0)
      call fr_frame2d(xmin,xmax,ymin,ymax,0)
      call fr_xyname('re','im')
      call fr_contour(f,imax+1,jmax+1,20,-2)

      call fr_canvas(2)
      call fr_margin2d(50,30,50,70)
      call fr_aspect2d(1.0,1.0,0)
      call fr_frame2d(xmin,xmax,ymin,ymax,0)
      call fr_xyname('re','im')
      call fr_contour(f,imax+1,jmax+1,20,-2)

      call fr_canvas(3)
      call fr_margin2d(50,30,50,70)
      call fr_aspect2d(1.0,1.0,0)
      call fr_frame2d(xmin,xmax,ymin,ymax,0)
      call fr_xyname('re','im')
      call fr_contour(f,imax+1,jmax+1,20,-2)
   enddo
   call fr_gend

end program gsample30
```

CANVAS を使えば，複数の GIF アニメーションを同時に作ることができます。本例では，サイズやカラーモードの異なる 3 つの GIF アニメーション，`canvas1.gif`，`canvas2.gif`，`canvas3.gif` ができます。

付録 A　ディスプレイウィンドウに描画する時の操作方法と注意点

FRAMES はディスプレイ描画をするために X ウィンドウライブラリを用いています。ディスプレイ描画は「プログラム実行」と「図形描画」のタイミングが一致しているリアルタイム表示なので，表示の際に要求に応じて入力デバイス (マウスやキーボード) から操作ができるようになっています。この節ではディスプレイ描画をする際の操作の方法と注意点について述べます。

A.1　X ウインドウシステムの起動

X ウインドウライブラリを用いてディスプレイ描画をしますので，当然のことながら X ウィンドウシステムを起動した上でプログラムを実行しなければ図形は描けません。しかし，X ウィンドウシステムを起動せずに実行してもエラーにはなりません。次のメッセージを出力後，画面に図形を描かずに実行を続けます。

```
Graphic Workstation not opened -- Skip Graphic Routines
```

この時ファイル出力ルーチンが入っていればファイルは作成されます。文字端末から起動したときや遠隔ホストから描画した時に DISPLAY 変数が適切に指定されていなくてサーバーが開けない場合も同様です。X の描画ができない遠隔ホストで最初からファイルを作ることのみが目的で使う場合には，X に依存しないライブラリ libfrNOX.a を作成しておけばいいでしょう (1.4 参照)。

A.2　一時停止時における描画ウィンドウの操作

FRAMES は Fortran プログラムの実行に沿って描画しますが，ディスプレイウィンドウに描画している時には fr_gend や fr_gpause により一時停止した状態になり，この間に図形を観察したり，xwd を起動して画面ダンプを取ったりすることができます。この一時停止の状態では以下のようなマウスとキーボードの操作が可能です。

なお，かっこ内の CNTL+□はキーボードショートカットです。コントロールキーを押しながら，□キーを押すことで同じ動作をさせることができます。

右ボタン　fr_gend や fr_gpause により一時停止した状態を解除します。fr_gend の場合には解除と同時にウィンドウがクローズします (CNTL+F)。

中ボタン　カラーモードを変更します。背景色 (白色) 以外の図形が全て黒色になる黒白モード，背景色を黒色にするモード，背景色を黒色にして，それ以外を全て白色にする白黒モード，およびデフォルトモードが順に切り替わります。

F3 キー　右ボタンクリックと同じ動作をします (CNTL+F)。

F4 キー　白色以外の図形が全て黒色になります (CNTL+W)。

F5 キー　背景色を黒色に，黒色以外の描画図形全てを白色にします (CNTL+R)。

F8 キー　X ウィンドウへの描画を停止し，以後はファイルへの保存だけにする (CNTL+Z)。

リサイズ　描画ウィンドウは拡大縮小が可能です。これは通常のマウスによるウィンドウリサイズ操作で行います。ただし，ウィンドウの縦横比は変わりません。拡大縮小に伴って図形も表示文字サイズも拡大縮小されます。

クローズ　クローズボタンを押すなどのウィンドウ操作によりウィンドウを閉じる（クローズする）と，F8 キーと同じように以後の描画を停止し，ファイル保存だけのモードに移行します。

背景色を黒色にするモードは，背景色を黒色に，黒色の描画図形を白色にします。それ以外の色の図形は変化しません。好みに応じて切り替えてください。

A.3　その他の注意点

FRAMES は，NEC の PC-9801 用が原点です。このため基本色の指定などは 98 用時代の遺産を引き継いで 8 色しかありませんが，おかげで指定が簡単になっているという利点もあります。描画ルーチンも，計算・描画共に遅く，少ないメモリ空間しかないマシンで使っていた時代に作ったものをそのまま残しているため，高速かつ広大なメモリ空間を持つ最近のマシンで使う時には若干注意が必要です。ここで，気がついた注意点を列挙します。

1. X ウィンドウでは，プログラムが終了する際に表示ウィンドウをクローズします。これを避けるため `fr_gend` は，終了化する前にプログラムを一時停止状態にします (A.2 参照)。このことから，`fr_gend` のコールは `stop` 文の直前が望ましいと考えられます。

2. 図形ウィンドウがアクティブな状態では `CNTL+C` による強制終了はできません。アプリケーションを起動させたウィンドウをアクティブにしてから `CNTL+C` キーを押してください。強制終了すると図形ウィンドウはクローズします。

3. 強制終了をすると出力ファイルが不完全になるため，再描画の際に読み込みエラーになる可能性があります。もし，計算させながら描画している途中で描画だけをストップして計算は続行させたい時には F8 (または `CNTL+Z`) キーで描画だけをストップさせて下さい。ただし，`fr_gpause` ルーチンで描画を一時停止させた時でないとこの機能は働きません。

4. *FRAMES* で用いている文字は以下の標準固定フォントを使用しています。

基準ポイント	8	10	12	14	16	18	20	22	24	26
フォント名	5x8	6x10	6x12	7x14	8x16	8x16	10x20	10x20	12x24	12x24

5. 通常は描画空間のサイズはディスプレイウィンドウのドット数に一致します。もし `fr_gwsize` ルーチンを用いて描画空間のサイズをディスプレイのドット数より大きくなるように指定した場合には適当に縮尺されたウィンドウが表示されますが，描画空間座標は指定したとおりになります。また，マウスなどを使ってウインドウサイズを変更してもそれに従って描画空間も拡大・縮小しますから，描画空間座標は変わりません。つまり図形も拡大・縮小されます。

6. *FRAMES* のウィンドウは自由に拡大・縮小できますが，それに従ってフォントサイズを計算し，画面のサイズを変更しても，ほぼ指定された場所に文字を描くようになっています。ただし，X サーバによってはこれらの中のいくつかは添付されていないかもしれません。添付されていないときは，`fonts.alias` ファイルなどを修正して，上記のフォント名を登録する必要があります。

付録 B　動作が確認されていない機種での *FRAMES* ライブラリ作成

1.4 で述べたように，動作確認済みの機種については `makefile.`*TARGET*（*TARGET* の部分は機種に依存）という名称のメーク設定ファイルを用意してあるので，これをお使いの OS やコンパイラ環境に合わせて修正して `make` することが可能です。しかし，*FRAMES* は一般的な Fortran と C 言語，および X ウインドウライブラリを使っているだけなので，設定ファイルを用意していないマシンで使いたい場合や，他の市販のコンパイラを利用して使いたい場合でも，さほど問題なくコンパイル・リンクができると思います。ただし，独自でコンパイルを試みる際にはいくつか注意が必要です。

FRAMES は C 言語と Fortran の基本的な文法しか使っていないのですが，どうしても機種に依存せざるをえないものにデータの並び順があります。これは CPU が仮定している整数値の並びが上位バイトから並んでいる “BigEndian” 型と下位から並んでいる “LittleEndian” 型が存在するためです。例えば，PowerPC や HP PA-RISC は前者であり，Pentium や Alpha は後者です。*FRAMES* のファイル保存形式はこの機種依存性をなくすようにしているため，逆にコンパイル時に型に合わせた読み書きルーチンを選択する必要があります。C 言語でのコンパイルオプション `-DLITTLE` がその選択指定で，指定した時は LittleEndian，しない時には BigEndian になります。

もしお使いのマシンがどちらの Endian 型か不明の時は，付属のプログラム `misc/bltest.c` をそのマシン上の適当な C コンパイラでコンパイル・実行してみて下さい。実行すると，LittleEndian か BigEndian かを表示してくれます。

もう一つ問題があります。*FRAMES* は C 言語と Fortran を混在させているので C 言語から Fortran のサブルーチンをコールしたり，その逆をしています。ところが，Fortran コンパイラによってはコンパイル時にサブルーチン名にアンダースコア ‘_’ を名前の前や後ろに自動的に付加するものがあり，このとき C 言語から Fortran サブルーチン名でコールするとリンクエラーになります。これを回避するために C コンパイラのオプションに，`-DNAMEU`, `-DNAMEUU`, `-DUNAME`, `-DUNAMEU`, を指定することで使い分けるようになっています。これらは以下の場合につけます。何もオプションをつけない場合にはアンダースコアを付加しないそのままの名前になります。

`-DNAMEU`	名前の最後に ‘_’ を 1 個付ける
`-DNAMEUU`	名前の最後に ‘_’ を 1 個付ける，ただしサブルーチン名に ‘_’ を含む場合には 2 個付ける
`-DUNAME`	名前の最初に ‘_’ を 1 個付ける
`-DUNAMEU`	名前の最初と最後に ‘_’ を 1 個ずつ付ける

なお，どのオプションを使えばいいかわからないときは，とりあえずオプション無しでコンパイル・リンクしてみて，リンク時のエラーメッセージに表示された足りないサブルーチンの名前から推測すれば良いでしょう。

付録 C　再描画アプリケーション tplot

3.2 の項で述べたように *FRAMES* には精度をあまり落とさずに描画データを保存する plt ファイル出力機能があります。*FRAMES* は物理座標のデータを直接サブルーチンに与えて図形を描くようになっているため，これを独自の方式で精度を落として保存したのが plt ファイルです。独自形式であるため，汎用の図形ソフトでは処理ができません。そこで plt ファイルを利用するために２つの処理方法を用意してあります。

一つは *FRAMES* ライブラリ作成の際に同時に作成される付属の再描画アプリケーション tplot を利用することです。代表的な tplot の機能を以下に示します。

- *FRAMES* で描いた図形の再描画
- ページ切り替え機能によるアニメーション (再生速度設定可能)
- 図形の拡大・縮小
- ２次元図形の座標測定
- ３次元図形の視点変換
- ３次元図形の自動回転表示
- 等高線の間隔などの各種パラメータ変更
- 等値面の閾値の変更
- 疑似ライティングの設定変更
- PostScript 出力
- GIF イメージ，GIF アニメーション出力
- 簡単なデータ解析 (線形回帰など)
- 外部コマンド利用による自動連続フィルター機能

plt ファイルは *FRAMES* を含むプログラムを実行した機種に依存しないように作られていますので，速いマシンで plt ファイルを作り，遅いマシンで tplot を起動させて再描画することも可能です。図形描画には時間がかかるので，プログラムが一旦完成すればシミュレーションをさせながら図形を描画することがわずらわしくなってきます。この時は，描画の途中で F8 キーによりディスプレイ表示を停止して，plt ファイルを作成するだけにするか，あらかじめ *CANVAS* の設定を変更して，描画せずに plt ファイルに出力する指定にすればいいでしょう。

なお，tplot は，描画の単位を描画ページで制御しています (2.4 参照)。つまり，プログラムで fr_ginit，fr_gclear，fr_canvas から次の fr_gclear，fr_canvas，fr_gend までの間に描かれた図形が「１ページ」で，tplot で加えた変更は基本的にこの描画ページ内でのみ有効です。

もう一つの plt ファイル利用手段は，ユーザーが独自に処理プログラムを作成することです。tplot にはない図形加工機能を作成したり，AVS の様にユーザープログラムによるカスタムモジュールが作成できるソフトから plt ファイルを利用する時にはこの方法を使います。詳細は付録 D を参照して下さい。

C.1　tplot による再描画

tplot は UNIX で実行するアプリケーションで，基本的な起動方法は

```
tplot
```

です。起動すると，plt ファイルの選択ダイアログが出てきますから，カーソルキーやマウスで再描画したい図形ファイルを選択して下さい。ファイル名を引数で指定することも可能です。

```
tplot <plt ファイル名>
```

plt ファイル名の拡張子は '.plt' がデフォルトになっていますので，拡張子を省略したときは '.plt' が付加されます。

ファイルを選択すると，その plt ファイルを読み込んで１ページ目の図形を表示します。図形表示ウィンドウは３つのエリアに分割されています。一番上が操作パネル，中央が図形描画エリア，一番下が文字列表示エリアです。

中央の図形描画エリアの大きさは plt ファイル作成時に `fr_gwsize` で指定した描画サイズによって変化します。また一番下の文字列表示エリアはユーザーの要求に応じた文字列を表示する領域です。例えば，表示されている図形が２次元図形の場合には，図形描画エリアをクリックするとその点の物理座標が表示されます。また，`fr_gnotes` (3.2.11) で指定した文字列の先頭文字が '`#`' の場合には，その文字列が表示されます。

一番上の操作パネルは，図 C.1 のようにページ表示領域と四角形で囲まれた操作ボタン・選択メニューからなっています。このボタンやメニューを使って `tplot` を操作します。

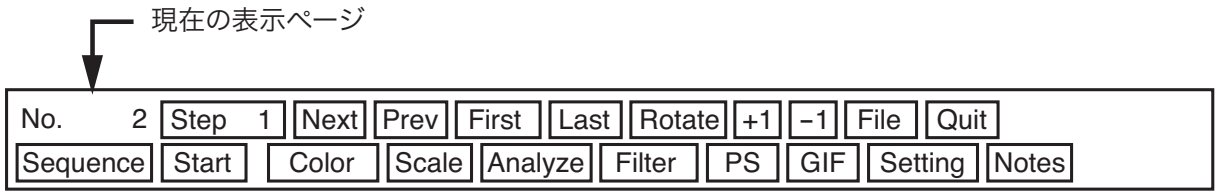


図 C.1 `tplot` の操作パネル

図に示してあるように上段の一番左は現在の表示ページ番号です。
上段にあるボタンの動作は以下の通りです。

`Step` n	`Next` または `Prev` の際のページ送り数表示および各種移動指定ダイアログ表示
`Next`	`Page Step` で指定したページ進む
`Prev`	`Page Step` で指定したページ戻る
`First`	先頭ページへ戻る
`Last`	最終ページへ進む
`Rotate`	３次元図形を一周回転させる
`+1`	３次元図形を 1 ステップ正に回転する
`-1`	３次元図形を 1 ステップ負に回転する
`File`	別の plt ファイルをオープンする
`Quit`	`tplot` を終了する

`Step` n ボタンはページの送りステップや表示ページを変更するためのもので，このボタンを押すと図 C.2 のような Step ダイアログが表示されます。

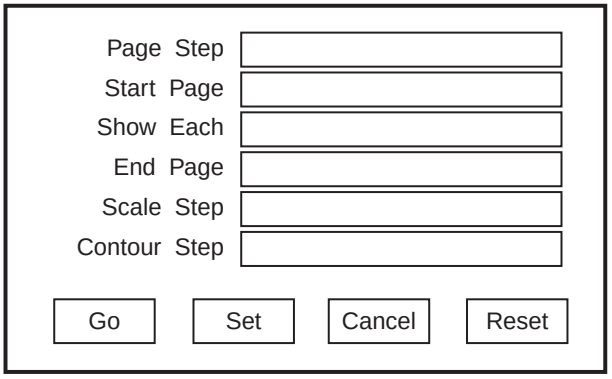


図 C.2 Step ダイアログ

必要な欄の数値を変更して Set ボタンを押すと変更が完了します。また， Go ボタンを押すかリターンキーを押すと変更が完了すると同時に Start Page にジャンプします。
各欄の意味は以下の通りです。

Page Step	Next ボタンや Prev で進むページステップ指定
Start Page	開始ページ指定 (指定ページへのジャンプ)
Show Each	ページステップ毎に表示する枚数指定 ((2)-(d) 項参照)
End Page	連続動作する時の最後のページ指定
Scale Step	スケール変更するページステップ指定
Contour Step	等高線の指定を変更するページステップ指定

スケール変更や等高線指定の変更は，操作パネル下段の Scale ボタンを押すと現れる Scale ダイアログで指定しますが，最後の 2 つは，このスケールや等高線の数などの変更を有効にするページのステップです。例えば，X 座標指定を変更して Scale Step を 3 にすれば，その変更は現在の表示ページから $3n$ ページ進んだページのみに有効になり，その他のページには影響を与えません。

操作パネル下段にあるボタンの動作は以下の通りです。

Sequence	連続描画モードを指定するメニュー
Start	連続描画モードを開始または停止する
Color	背景色の白黒を入れ替える
Scale	座標スケールを変更したり視点を変更したりするダイアログ表示
Analyze	２次元 fr_graph2d のデータ解析ダイアログ表示
Filter	現在の表示図形に対し，フィルターを実行する (注 2)
PS	現在の表示図形を PostScript コードに変換して出力する (C.3 参照)
GIF	現在の表示図形を GIF イメージに変換して出力する (C.4 参照)
Setting	各種設定変更ダイアログ表示
Notes	fr_gnotes で埋め込んだ文字列表示

ここで,「連続描画」というのは，複数ページからなる図形に対し，描画を連続的にすることでアニメーションにしたり，画面のハードコピーを連続的に取ったりすることを意味します。 Sequence メニューは設定だけで，右横の Start ボタンで連続描画が開始します。
連続描画できるものには以下があります。

Display	連続ディスプレイ出力 (簡易アニメーション)
Filter	連続フィルター実行 (連続画面ダンプ，連続ハードコピーなど，注 2)
PS output	連続 PostScript 出力 (C.3 参照)
GIF image	連続 GIF イメージ出力 (C.4 参照)
GIF anime	GIF アニメーション (C.4 参照)

Start ボタンを押して「連続描画」が開始すると， Start ボタンは Stop ボタンに変わります。「連続描画」は Step 指定したページ毎に描画して，最後のページになったら停止します。もし途中でやめたくなったときには Stop ボタンを押します。

PostScript 出力と GIF 出力の詳細はそれぞれ，C.3 と C.4 を参照してください。その他のダイアログの詳細は省略します。

C.2 tplot の描画指定オプション

以上は基本的な tplot の使用方法です。tplot を起動してから各種のダイアログによって指定すれば，拡大・縮小や視点回転など様々な図形の加工ができます。さらに，図形を PostScript にしてプリンタに出力したり GIF アニメーションを作成することもできます。しかし，加工手順が固定化するといちいちダイアログを引っ張り出して変更するのが面倒になってきます。そこで，オプションを指定することで，起動直後から加工して表示することや自動的に全てのページのハードコピーを取ることなどができます。C.3 の PostScript 出力や C.4 の GIF 出力では，ディスプレイに表示しなくても図形をプリンタに出力したり，GIF イメージにすることができますが，その時には特に有効になります。

tplot にオプションを付ける時は，

```
tplot [オプション] <plt ファイル名>
```

と指定します。plt ファイル名の拡張子は‘.plt’がデフォルトになっていますので，拡張子を省略したときは‘.plt’が付加されます。オプションだけで，plt ファイル名指定を省略するとファイル選択ダイアログが出てきます。

なお，オプション‘-help’を指定すると各種オプションのヘルプの一覧表が表示されますので，その中から必要に応じたヘルプ出力をして下さい。なお，オプションは指定した順に処理しますので，後から指定したものが優先されます。また，‘.tplotrc’という名のファイルを用意して，その中にあらかじめよく使うオプションを書いておくこともできます (注 1)。さらに，オプションにスケール指定や等高線指定をスクリプト形式で加えたプリファレンスファイルを作成し，tplot 起動時にそのファイルをオプションで指定することによって tplot 起動時の動作を制御することもできます (C.5)。

なお，オプション指定の際に数字と組み合わせるものは，オプションと数字との間にスペースを入れないで下さい (例：-g5，-o10 等)。

(1) 描画図形の縮尺および描画領域の指定オプション

描画時の図形の縮尺と描画空間座標での描画領域を指定します。オプションは 4 種類あり，このうち，-r，-w，-h は縮尺を，-vw は描画領域の指定に使います。-w，-h での縮尺指定では規定値のウィンドウサイズを 640×480 ドットとして，これを 1 とします。*FRAMES* は，幅 320 ドットを最小値としているので，これ以下になるような指定はできません。また，初期においてはディスプレイ画面のドット数の $\frac{3}{5}$ を最大値としているので，これ以上の指定もできません。

(a) -rx　(例，-r0.5，-r2)
　これは縮尺比を与えます。x は実数を指定します。
　規定値は-r1 です。

(b) -wn　(例，-w400，-w500)
　これはウィンドウの横ドット数 n を与えて縮尺を決めます。n は整数を指定します。
　規定値は-w640 です。なお，320 より小さいと 320 となります。縦ドット数は横ドット数の縮尺で決定されます。

(c) -wn_1xn_2　(例，-w600x300，-w500x500)
　これはウィンドウの横ドット数 (n_1) と縦ドット数 (n_2) を与えます。n_1，n_2 は整数で指定します。
　規定値は-w640x480 です。もし横ドット数が 320 より小さいときは 320 に，縦ドット数が 240 より小さいときは 240 になります。作成図形の大きさは横ドット数の縮尺で決定されます。即ち，-wn_1 と指定した場合の図形と同じ大きさになります。縦方向は図形の中心部を取り出すので，このオプションは，上下の余白を取りたいときに使います。

(d) -hn　(例，-h200，-h300)
これはウィンドウの縦ドット数 n を与えて縮尺を決めます。n は整数を指定します。
規定値は-h480 です。なお，240 より小さいと 240 となります。横ドット数は縦ドット数の縮尺で決定されます。

(e) -hn_1xn_2　(例，-h300x600，-h500x500)
これはウィンドウの縦ドット数 (n_1) と横ドット数 (n_2) を与えます。n_1，n_2 は整数で指定します。
規定値は-h480x640 です。もし縦ドット数が 240 より小さいときは 240 に，横ドット数が 320 より小さいときは 320 になります。作成図形の大きさは縦ドット数の縮尺で決定されます。即ち，-hn_1 と指定した場合の図形と同じ大きさになります。横方向は図形の中心部を取り出すので，このオプションは，左右の余白を取りたいときに使います。

(f) -vwn_1xn_2+x_0[または-x_0]+y_0[または-y_0]　(例，-vw500x300+20+20)
これは描画領域を描画空間座標の ($\pm x_0$,$\pm y_0$) から $n_1 \times n_2$ の大きさに限定します。規定値は
-vw640x480+0+0 です。ウィンドウは描画領域の大きさと-r オプションで設定した比率で決まります。
x 以降は省略してもかまいません。省略した場合には規定値を用います。また，n_1xn_2 を省略すると，オフセット移動になります。

(g) -rw$n$　(例，-rw400)
-vw で指定した後，切り取った描画領域の横幅を n になるように縮尺します。n は整数を指定します。

(h) -rh$n$　(例，-rh300)
-vw で指定した後，切り取った描画領域の縦幅を n になるように縮尺します。n は整数を指定します。

(2) 描画開始ページ・描画ステップなどの画像送り制御オプション

tplot はダイアログとメニューの操作により各種の操作ができますが，慣れて来ると最初のページを飛ばして 3 枚目から表示したいとか，5 枚毎に表示したいとかの動作制御を起動時に指定したい場合が出て来ます。特に，C.3 の PostScript 出力のように画面描画せずに出力する時には，必要なページだけを出力するのに制御が必要です。そこで，以下に述べる画像送り制御オプションを用意しています。

(a) -gn (描画開始ページ指定)　(例：-g5)
複数ページを含んだ描画ファイルの場合，n 枚目のページから描画を開始するときに使います。

(b) -om (ページ枚数指定)　(例：-o1，-o10)
描画ページ数を m 枚にします。

(c) -sm (ページステップ指定)　(例：-s2，-s5)
ページステップを m にします。

(d) -em (ページステップ毎に表示する枚数指定)　(例：-e2，-e5)
上記のページステップ指定によりステップを指定したときに，ステップ毎の表示枚数 m を指定します。ページステップを m にします。例えば，-s10 -e3 と指定すると，表示ページは，1,2,3,11,12,13,21,22,23... となります。

(e) -ss$m$ (スケール設定ステップ指定)　(例：-ss2，-ss5)
スケール設定指定のステップを m にします。

(f) -cs$m$ (等高線設定ステップ指定)　(例：-cs2，-cs5)
等高線設定指定のステップを m にします。

(3) 描画手法の変更オプション

隠線消去法や色分割などにおいて，計算時に高速な手法を取り，再描画時に別の描画手法に変更することができます。

(a) -z
fr_surface(3.6.17) において高度な隠線処理 (MODE=3 または 4) をする場合には，メモリは必要ですし，時間もかかります。このオプションを指定すると，元の隠線処理手法にかかわらず，再描画の際に高度な隠線処理 (mode=4) にモード変更して描画します。

(b) `-rv` (背景色白黒交換指定)
FRAMES はデフォルトの背景色は白色ですが，このオプションを指定すると，背景を黒色にすることができます。

(c) `-k`
このオプションを指定すると，等高線ルーチンでの描画方法を強制的に HLS 色指定モードに切り替えます。ただし，すべての等高線図が置き換わります。

(d) `-k`*nd* [,nc_1,nc_2]
このオプションを指定すると，等高線ルーチンでの描画方法を強制的に HLS 色指定から通常色指定モードに切り替えます。*nd* は線の数，nc_1 は第 1 種色指定，nc_2 は第 2 種色指定です。色の指定は省略可能で，省略すると $nc_1 = 5$，$nc_2 = 7$ となります。

(4) 自動連続再描画モード指定オプション

C.1 で述べたように，`tplot` のメニューで連続描画モードにすれば，図形表示をアニメーションにしたり，連続的にフィルターを実行させることができますが，次のオプションによりこれらを起動時から行うことができます。これを自動連続再描画モードといいます。自動連続再描画モードにすると，操作パネルや文字列表示エリアは現れず，図形描画エリアのみが現れて指定した図形を次々に表示します。全ての図形を表示し終わったら `tplot` の実行は終了します。このため，図形の加工をしたい時は上記のオプションや C.5 のプリファレンスファイルを併用する必要があります。なお，自動連続再描画モード指定は C.3 の PostScript オプションや C.4 の GIF オプションとの共存はできません。

(a) `-a` (Animation)
画面切替を利用して連続ディスプレイ表示をします。すなわちアニメーション表示になります。

(b) `-f` (Filter)
1 ページの描画が終わり次第，指定したフィルターを実行します。フィルター実行が終了すると次の描画に入ります (注 2)。

(c) `-h` (Hardcopy)
1 ページの描画が終わり次第，ハードコピーコマンドを実行します。ハードコピーが終了すると次の描画に入ります (注 3)。

注 1) `.tplotrc` ファイルの書き方は上記のオプション指定どおりです。ただし，1 行に 1 オプションで 1 カラム目から書く必要があります。それ以外は無視されますから，コメントを書くことも可能です。なお，`.tplotrc` ファイルはカレントディレクトリになければホームディレクトリを探し，それもなければデフォルトを使用します。また，`tplot` 起動時にオプションを指定した場合にはそちらが優先されます。

注 2) フィルターはプログラム内部で `system` ルーチンにより `filter_dump` というスクリプトを実行することで実現しています。このため `Filter` ボタンの動作を有効にしたり，自動連続再描画モードでフィルターを指定するには `filter_dump` を用意しておく必要があります。このスクリプトに対し，`tplot` は 4 個の引数を与えています。
1 番目はウィンドウの番号，2 番目はページ番号，3 番目はフィルターをかけた順番，4 番目は描画ファイル名からディレクトリ名と拡張子 (`.plt`) を除いた文字列です (2 番と 3 番の数字は 4 桁で 0001〜9999 です)。このフィルターはウィンドウデータを一定の形式に変換するときに使用します。例えば，描画画面をダンプして `pbmplus` を使用して PICT 形式に変換したファイルにする時は次のようなスクリプトを用意します。

```
xwd -id $1 | xwdtopnm | ppmtopict > $4$2.pict
```

このように 1 番目の引数 (`$1`) は `xwd` の`-id` オプションの引数として用い，2 番目 (`$2`) または 3 番目 (`$3`) の引数と 4 番目の引数 (`$4`) はファイル名を指定するのに使います。全ての引数を使う必要はありません。注 3 のスクリプトにすれば，ハードコピーを取ることもできます。

なおオプションによりデフォルトのフィルター名 filter_dump は変更可能です。

-f:*filtername*

のようにオプション-f に続けてセミコロンとフィルター名 (*filtername*) を書いて下さい。

注 3) 自動連続再描画モードでハードコピーを取る場合は，フィルターとは別に引数 1 個のスクリプト hard_copy を用意します。このハードコピーコールはプログラム内部で system ルーチンにより hard_copy を実行することで実現しています。tplot はこのスクリプトの引数にウィンドウの ID 番号を与えますので xwd ルーチンを使用するときに利用できます。筆者の研究室の IBM 25T ですと次のような内容のスクリプトになっています。

```
xwd -bitmap -nobdrs -id $1 | xpr -rv -device jprinter | qprt -w132
```

このように 1 番目の引数 ($1) は xwd の-id オプションの引数として用います。xpr を別のフィルターにし，qprt を lpr コマンドなどにすれば他の機種でも利用できるでしょう。

C.3 tplot による PostScript 出力

tplot は図形を PostScript (PS) に変換してファイルに出力したり，パイプを通して PS プリンタに出力することができます。tplot を立ち上げると，C.1 で述べたように PS ボタンがあり，出力したいページでこのボタンを押せば，その画像が PostScript に変換されて出力されます。出力の前に座標スケールの変更などによって図形を加工しておけば，加工された図形が出力されます。

出力先はオプションを指定しなければ，ファイルに出力されます。作成されるファイル名は，指定した plt ファイルの名前からディレクトリ名を取り，拡張子 '.plt' を，'.ps' に変えたものになります。例えば，graph.plt からは，graph.ps ができます。ただし， PS ボタンを押して個々の図形を指定したときは plt ファイル名を name.plt とすると，出力順に name_0001.ps，name_0002.ps... という名前になります。なお，この名前の部分は-pn オプションを用いて変更することが可能です。また， File ボタンにより新たなファイルをオープンしても PostScript 出力ファイルは切り替わらず，最初の plt ファイルの名前で開いたファイルに続けて出力します。

以下に PostScript 出力の際のオプションを説明します。なお，PostScript は PS プリンタに出力して図形を印刷することが目的なので，図形は用紙サイズを基準に縮尺されます。このため以下の説明は印刷を前提にしていることにご留意下さい。

(1) 出力ファイル名指定，出力パイプ指定オプション

出力ファイルを別の名前にしたいときには，-pn オプションを使って名前を指定します。

-pn:ファイル名

例えば，-pn:abcde を付ければ，plt ファイルの名前が何であっても，できるファイルの名前は abcde.ps になります。なおこのように，拡張子 '.ps' は自動的に付加されますから，オプション指定の際は付けないで下さい。

ファイルではなく，プリンタに直接出力したいときには，パイプ指定をします。パイプ指定は，-p オプションで行います。

-p:パイプ名 または -p:"パイプフィルターコマンド"

例えば，

```
tplot -p:"lpr -Plp2" [他のオプション] <plt ファイル名>
```

とすれば，lpr コマンドを利用して lp2 プリンタに PostScript コードを送出します。

(2) ダイレクト出力オプション

-ps オプションを用いれば，ディスプレイ表示を省略して全ての図形を PostScript 出力することができます。これをダイレクト出力といいます。tplot の再描画機能を使って必要な図形のみを印刷する機能は便利なのですが，慣れてきて出力したいページが分かっていると逆に煩わしくなってきます。また，遠隔で操作するときは必ずしも X 端末で描画できるとは限りません。-ps でダイレクト出力を指定すると，plt ファイルから PostScript への変換プログラムとして動作します。例えば，

```
tplot -ps -p:"lpr -Plp2" [他のオプション] <plt ファイル名>
```

と入力すればディスプレイへの表示は一切せずにプリンタへ出力されます。出力ページの指定をしたい時には，C.2(2) のページ送り指定オプションを使います。さらに画像の拡大などの加工を行いたいときには C.5 のプリファレンスファイルを使います。

いくつかの描画ファイルに入っているデータを続けて出力したいときは tplot をファイル毎に使用すれば良いのですが，用紙が別になります。そこで-ps オプションのときに限り複数のファイルを同時に指定して，1 枚の紙に出力することができます。

```
tplot -ps [オプション 1] <plt ファイル名 1> [オプション 2] <plt ファイル名 2>...
```

このとき，オプション指定はそれ以降のファイル描画に対してのみ有効です。ただし，オプション 1 以外での出力サイズ指定オプションは無視されます。

(3) 出力サイズ指定オプション

図形の出力サイズは，用紙の種類，縦か横か，縦横いくつずつ並べるかで決定します。デフォルトでは A4 縦の用紙を縦に 2 分割して，それぞれを描画領域として図形を描きます。つまり 1 枚の用紙に 2 ページの図形を描きます。これを変更するには，以下のオプションがあります。

(a) -l[n_1[xn_2]]　(例：-l2，-l1x2)
A4 用紙を横にして縦 n_1 個，横 n_2 個の図形を描きます。n_2 を指定しなければ縦に n_1 個描き，n_1 も指定しなければ 1 個だけ描きます。

(b) -m[n_1[xn_2]]　(例：-m2，-m2x3)
A4 用紙を縦にして縦 n_1 個，横 n_2 個の図形を描きます。n_2 を指定しなければ縦に n_1 個描き，n_1 も指定しなければデフォルトと同じで縦に 2 個描きます。

(c) -s[n_1[xn_2]]　(例：-s，-s3x2)
-l オプションと動作は同じですが，n_1 も n_2 も指定しなければ縦に 2 個，横に 2 個描きます。

(d) -a4l
用紙を A4 横にします。

(e) -us
用紙を US Letter にします。

(f) -mz
Ghostscript を用いてイメージファイルにする際など，左と下のマージンを 0 にしたい時に指定します。

(4) PostScript コード出力における色指定

デフォルトの PostScript はカラー出力ですが，白黒の PostScript プリンタに出力する場合はグレイスケールに変換する必要があります。以下に図形をグレイスケールに変換するためのオプションを示します。

(a) -c0
全ての図形を同一の黒色にします。

(b) -c1
下記の対応により，図形に応じて色指定をグレイスケールで置き換えます。

1. 直線，曲線　　→ 7 段階のグレイスケール
2. 塗り潰し図形　→ 7 段階のグレイスケール
3. 文字　　→ 7 段階のグレイスケール
4. 矢印　　→ 7 段階のグレイスケール
5. HLS 色モード　→ 文字，矢印を除く全図形に対し HLS 色設定数段階のグレイスケール

(c) `-c2`
直線，曲線を点線，破線などの線種を変えることで表現します ((h) 参照)。それ以外の図形は (b) の置き換えになります。

(d) `-c3` (デフォルト)
カラー PostScript を出力します。

(e) `-v0`
矢印を指定色に関係なく黒色で出力します。

(f) `-t0`
文字を指定色に関係なく黒色で出力します。

(g) `-x`n_1`[`$n_2n_3n_4n_5n_6n_7n_0$`]` (デフォルトは `-x12345670`)
`-c1` オプションの際，8 色モードのグレイスケールの順序を変更します。ここで，n_1... は 0〜7 の数字または * です。デフォルトでは番号の小さいものから順に濃くなって，IC=7(黒色) が真黒に対応しています。これを変更するのが本オプションです。ただし，通常 0 番色指定 (背景色) は使用しませんから，1 番から順に指定するようになっていることに注意してください。また，2 番以降は無視しても結構です。省略すると，デフォルト値が使われます。特に最後の 0 番色指定はあまり使わないと思います。逆に，前の順序にある番号の色にデフォルト値を使いたいときは * を指定してください。もし描画したくない色の図形があるときはその色を 0 に指定することで取り除くことができます。

(h) `-y`n_1`[`$n_2n_3n_4n_5n_6n_7n_0$`]` (デフォルトは `-y12345670`)
`-c2` オプションの際，8 色モードの線種の順序を変更します。ここで，n_1... は 0〜7 の数字または * です。デフォルトでは

0	描画せず	4	長破線
1	点線	5	1 点鎖線
2	短破線	6	2 点鎖線
3	中破線	7	実線

です。これらを変更するのが本オプションです。ただし，通常 0 番色指定 (背景色) は使用しませんから，1 番から順に指定するようになっていることに注意してください。また，2 番以降は無視しても結構です。省略すると，デフォルト値が使われます。特に最後の 0 番色指定はあまり使わないと思います。逆に，前の順序にある番号の色にデフォルト値を使いたいときは * を指定してください。もし描画したくない色の線があるときはその色を 0 に指定することで取り除くことができます。

C.4 `tplot` による GIF 出力

`tplot` は 1 ページの図形を GIF イメージにしたり，複数ページの図形を GIF アニメーションにして出力する機能があります。`tplot` を立ち上げると，C.1 で述べたように `GIF` ボタンがあり，出力したいページでこのボタンを押せば，その図形が GIF イメージに変換されてファイルに出力されます。出力の前に座標スケールの変更などによって図形を加工しておけば，加工された図形が出力されます。また，PostScript 出力と同様に画面描画せずに直接 GIF イメージにすることも可能です。GIF イメージは容量が少ないため，遠隔のマシンで作成した plt ファイルから GIF イメージを作成してこれを FTP で転送して結果を見る，というような使い方ができます。

`GIF` ボタンを押したときに作成される GIF イメージのファイル名は，plt ファイル名を `name.plt` とすると，出力順に `name_0001.gif`，`name_0002.gif`... となります。また，連続描画モードで GIF アニメーションを作る

と，ファイル名は name.gif となります。なお，この name の部分は-gn オプションを用いて変更することが可能です。

以下に GIF 出力の際のオプションを示します。

(a) -gif
ディスプレイ表示を省略して，ダイレクト出力で全ての図形を GIF イメージにします。作成される GIF ファイル名は，plt ファイル名を name.plt とすると，1 番目のページが name0001.gif，2 番目のページが name0002.gif... というようになります。name の部分は-gn オプションを用いて変更することが可能です

(b) -gifa
ディスプレイ表示を省略して，ダイレクト出力で全ての図形から作成した GIF イメージを一つのファイルにして GIF アニメーションにします。作成される GIF アニメーションのファイル名は，name.gif となります。name の部分は-gn オプションを用いて変更することが可能です

(c) -gn:ファイル名
GIF 出力ファイル名を変更します。例えば，-gn:abcde を付ければ，plt ファイルの名前が何であっても，できるファイルの名前は abcdexxxx.gif になります。なおこのように，拡張子 '.gif' は自動的に付加されますから，オプション指定の際は付けないで下さい。なお，このオプションを付けると，PostScript 出力ファイル名も abcde.ps になります。ただし，-gn と-pn を同時に使用すれば，それぞれ別の名前を付けることができます。

(d) -gi
GIF を Interlace にします。

(e) -gt$n$
GIF Transparent の基本カラーの番号を n にします。(n=1〜7)

(f) -gd$n$
GIF アニメーションの遅延時間を n にします。(単位は $\frac{1}{100}$ 秒，デフォルトは 10)

(g) -glc
GIF アニメーションのカラーマップを個別にします (デフォルト)。

(h) -ggc
GIF アニメーションのカラーマップを共通にします。

(i) -grv
GIF イメージの白と黒を入れ替えます。

(j) -gr$x$
GIF のウィンドウ比を x にします。

(k) -gw$n$
GIF のウィンドウ幅を n にします。

(l) -gh$n$
GIF のウィンドウ高さを n にします。

C.5 プリファレンスファイルでのスクリプト

tplot は PostScript や GIF をインタラクティブに出力することもできれば，ディスプレイ表示無しにダイレクトに出力してファイルコンバータのように使うこともできます。インタラクティブな出力ではダイアログによる設定によって図形のスケールを変更して出力することが可能ですが，コンバータとして使うときにはダイアログは使えません。そこで，プリファレンスファイルを作成し，その中にスクリプト形式でスケール指定などを埋め込めるようになっています。プリファレンスファイルには，オプションも入れることが可能ですから初期オプション指定ファイル.tplotrc を変更せずに，描画図形ファイルに応じたオプション指定をすることができます。また，

プリファレンスファイルはダイレクト出力の図形加工に使えるだけではなく，インタラクティブモードで初期スケールを変更するのにも使えます。

プリファレンスファイルの指定は

```
-on:プリファレンスファイル名
```

です。例えば，ファイル名が abcd.pref の時は，

```
tplot -on:abcd.pref [他のオプション] <plt ファイル名>
```

です。

スクリプトの記述方法は以下のような代入形式で与えます。

```
keyword=value
```

スクリプトは 1 行に 1 文だけしか書けません。

keyword の形式は，name または name_*n*d または name(*nn*) または name_*n*d(*nn*) です。ここで，name は以下のものが用意されています。

xmin，xmax	X 方向の最小値と最大値（2 次元または 3 次元）
ymin，ymax	Y 方向の最小値と最大値（2 次元または 3 次元）
zmin，zmax	Z 方向の最小値と最大値（3 次元のみ）
theta，phi	視点の緯度と経度（3 次元のみ）
fmin，fmax	等高線の最小値と最大値（2 次元または 3 次元）
fdelta	等高線の値間隔（2 次元または 3 次元）
ncontour	等高線の数（2 次元または 3 次元）

ただし，fdelta と ncontour を同時に指定すると fdelta の指定が優先します。

また，*n*d は 2d または 3d の指定ができます。これを指定しないときには 2 次元と 3 次元の両方のスケールが同時に指定されます。

(*nn*) の指定は *nn* ページ目の図形のスケールを指定するのに用います。ただし，-ss または-cs オプションによりスケールステップ指定したときのみ有効です。プリファレンスファイル中にはオプション指定も書き込むことができるので，(*nn*) を指定するときは指定する前に-ss または-cs オプションを同時に書き込んでおけばいいでしょう。

また，行の先頭が ‘%’，‘#’，‘$’，‘!’ のどれかの場合には，その行は無視されるのでコメント行として使うことができます。また，上記の文法に則っていないスクリプトを書いた場合にも無視されます。

例をいくつか示します。

xmin=50	2 次元図形または 3 次元図形の X 方向の最小値を 50 にする
ymax_3d=100	3 次元図形の Y 方向の最大値を 100 にする
ymin_2d(3)=-5	3 ページ目の 2 次元図形の Y 方向の最小値を −5 にする
theta=60	視点の緯度を 60 度にする

付録 D　plt ファイルからデータを抽出する C 言語ルーチン

plt ファイルは，*FRAMES* 独自の形式を持つベクトルデータファイルであり，基本的には `tplot` を用いて，再描画して図形を加工したり，PostScript に変換してプリンタに出力するために使用します。しかし再描画に最低限必要な程度まで精度を落として保存するのでデータ量が小さい上に，動作機種に依存しないように作られていますから，大容量のデータを plt 形式で保存し，他のプログラムから利用することができれば便利です。

plt 形式は，ヘッダデータに続いて，各描画ルーチンに対応して

- 描画図形指定データ (1 バイト)
- 各種指定値 (整数)
- 各種データ値 (実数)
- .....

のように保存されています。この保存データ並びは最初に来る 1 バイトの描画図形指定データで決定されるため，データ保存形式の詳細がわからなければ抽出できません。そこで，C 言語の構造体を用いて抽出するルーチンを用意してあります。

D.1　データを抽出する C 言語ルーチンの使用方法の概略

ここでは，データを抽出する C 言語ルーチンを使用するプログラムの 1 例を示して，それについての簡単な説明をすることにとどめ，各ルーチンの詳細な仕様は省略します。

```
/*************************************************************************
 *   test_plt.c     Test for Data Restoring Routine from PLT File        *
 *************************************************************************/
#include <stdio.h>
#include "frames_plt.h"    ..................................................... (1)

main()
{
    int com,i,j,n;
    frames_data dat;    ..................................................... (2)

    if (open_plt_file("graph")) {    ..................................... (3)
        printf("Cannot open plt file !\n");
        exit(1);
    }
    dat.com=0;    ..................................................... (4)
    while (1) {    ..................................................... (5)
        com=get_frames_command(1);    ..................................... (6)
        if (com==0) break;    ..................................... (7)
        if (com<0) printf("clear stop\n");
        if (com==FRAMES_CONTOUR) {    ..................................... (8)
            printf("command = %04d\n",com);
            get_frames_data(&dat);    ..................................... (9)
            printf("%d %d\n",dat.nx,dat.ny);
            for (j=0,n=0;j<dat.ny;j++) {
               for (i=0;i<dat.nx;i++,n++) printf("%d %d %lf\n",i,j,dat.X[n]);
            }
          }
          else if (com==FRAMES_GRAPH2D) {
            get_frames_data(&dat);
            if (dat.nz==100) {
                printf("command = %04d\n",com);
                printf("%d\n",dat.n);
                for (i=0;i<dat.n;i++) {
                    printf("%d %lf %lf\n",i,dat.X[i],dat.Y[i]);
                }
            }
          }
          else if (com==FRAMES_GRAPH3D) {
            get_frames_data(&dat);
            if (dat.nz==1) {
```

```
            printf("command = %04d\n",com);
            printf("%d\n",dat.n);
            for (i=0;i<dat.n;i++) {
                printf("%d %lf %lf %lf\n",i,dat.X[i],dat.Y[i],dat.Z[i]);
            }
        }
      }
      else if (com==FRAMES_FILEINT) {     ..... (10)
        get_frames_data(&dat);
        if (dat.n==1) {
            printf("command = %04d\n",com);
            printf("%d\n",dat.nx);
        }
      }
      else if (com==FRAMES_FILEDBL) {
        get_frames_data(&dat);
        if (dat.n==1) {
            printf("command = %04d\n",com);
            printf("%lf\n",dat.x1);
        }
      }
      else get_frames_data(NULL);     ..... (11)
    }
    free_frames_data(&dat);     ..... (12)
}
```

プログラム中の番号で示した点を説明します。

(1) 必ず，frames_plt.h をインクルードして下さい。

(2) データ抽出用の構造体は frames_data という名前で宣言します。

(3) ファイルのオープンは open_plt_file を用います。引数はファイル名で，拡張子をつけなければ '.plt' が仮定されます。戻り値は

正常終了	0
異常終了 (ファイルが無いときなど)	1

です。

(4) その構造体 (この例では dat) の com という要素を最初必ず 0 にセットします。ただし，一度セットした後は，(12) の free_frames_data を実行しない限り 0 にしないで下さい。

(5) データ抽出は無限ループの中に入れます。

(6) 描画図形指定データは get_frames_command という関数で受け取ります。引数は 1 にして下さい。

(7) get_frames_command の結果が 0 の時には fr_gend，即ち，保存データが無くなったときであり，負の時には fr_gclear で次のページに入ったことを表します。

(8) get_frames_command の結果が正の時には次に保存されている描画図形の指定データになっています。描画図形指定データは 1 バイトの数ですが，サブルーチン名のプレフィックス 'FR_' を 'FRAMES_' に置き換えた名前のマクロが frames_plt.h の中で定義されていますのでこれを使って判定できます。

(9) 描画図形のその他の保存データは get_frames_data 関数を用いて抽出します。ここで引数は抽出するための構造体へのポインタです。ここでは，2 次元データの fr_contour，1 次元データの fr_graph2d，3 次元データの fr_graph3d，でのデータ抽出方法が例として入っています。データ格納に必要なメモリは自動的に割り当てられます。

(10) fr_fileint などのデータ埋め込みルーチンで埋め込んだデータはここで役に立ちます (D.2 参照)。

(11) 必要なデータ以外は get_frames_data の引数に NULL を与えることにより読み飛ばすことができます。ただし，get_frames_data(NULL) をコールしなくても，次に get_frames_command 関数を呼べば自動的にコールしますのでこの部分は不要です。

(12) 不要になった構造体は free_frames_data 関数でメモリの解放をして下さい。

以上述べた関数は libfrX.a の中に入っていますので，libfrX.a をリンクしてもらえば使えます。例えば，上記の test_plt.c は

```
cc test_plt.c -o test_plt libfrX.a
```

とコンパイル・リンクします。図形描画はしないので libgdfr.a や-lX11 は不要です。

この他の注意事項としては,

(a) それぞれの描画コマンドにはレベルがあり，指定したレベルより大きい描画コマンドは自動的に読みとばされます。デフォルトでは，fr_graph2d，fr_contour などの，実際に描画データを埋め込んだルーチンのみ抽出するレベル (INLAY_LEBEL) になっています。このレベルは change_frames_level 関数で変更できます。レベルの名前は frames_plt.h を参照して下さい。

(b) 抽出用の構造体は読み込むデータ量に応じてメモリの割り当てを行います。一度割り当てたメモリの量が変化した場合には割り当てをやり直しますのでいくつかの要素数の違う配列を必要とするときにはその数の構造体を用意した方がいいでしょう。

(c) get_frames_command で読み込んだ描画図形指定データが期待していたものと異なる場合には，unget_frames_command にて差し戻すことができます。これは次に get_frames_command をコールしたときに再度抽出可能になります。

D.2 データ埋め込みのための *FRAMES* ルーチン

plt ファイルを作るプログラムにおいて，描画ルーチンに与えるデータは図形の形状を表すものだけです。しかし，データを保存するという視点に立つと，入力パラメータなどの描画に関係のない情報を同時に埋め込んでおくことができれば便利です。そこで，*FRAMES* ライブラリには描画に関係のない文字列や数値を埋め込むルーチンを用意してあります。これらは描画図形に一切影響を与えません。なお，文字列埋め込みルーチン fr_filechar は 3.2.11 の fr_gnotes と機能はほぼ同じですが，埋め込んだ文字列は tplot では無視されます。

◎ 文字列の埋め込み (fr_filechar)

書式　call fr_filechar(ch)
引数　character ch
動作　文字列 ch を埋め込む
注意　☆ 文字列の長さは 256 までである。それ以上埋め込みたいときは 256 文字ずつに分割すること。
☆ 本ルーチンを用いて埋め込んだ文字列は描画図形には何の影響も与えない。
☆ 本ルーチンは，データ埋め込みルーチン fr_gnotes (3.2.11) と機能はほぼ同じであるが，埋め込んだ文字列は tplot で表示されない。すなわち無視される。

◎ 整数の埋め込み (fr_fileint)

書式　call fr_fileint(k,n)
引数　integer k(n),n
動作　n 個の要素からなる整数配列 k(n) を埋め込む
注意　☆ n=1 の時には，k は変数でなくても良い。

◎ 実数の埋め込み (fr_filedble)

書式　call fr_filedble(x,n)
引数　real*8 x(n)
integer n
動作　n 個の要素からなる倍精度実数配列 x(n) を埋め込む
注意　☆ n=1 の時には，x は変数でなくても良い。
☆ 描画ファイルへの保存方法と同じルーチンを用いているため，保証される精度は倍精度より低い。

付録 E　C 言語から *FRAMES* を利用するには

FRAMES は，Fortran と C 言語が混在したライブラリですから，C 言語から利用することも可能です。実際，`tplot` は C 言語で書かれています。ただし，プログラミングの際，いくつか注意が必要です。

1. サブルーチン名は全て，`void` 関数として小文字で指定してください。なお，関数のプロトタイプ宣言用に `frames.h` というヘッダーファイルを用意していますのでこれを必ずインクルードして下さい。
2. Fortran は全て Call by Reference ですので，配列を指定するとき以外も全て変数に値を代入して，その変数のポインタを引数に与える必要があります。
3. 1 次元配列はそのまま与えても結構ですが，2 次元配列は第 1 引数と第 2 引数の順序を交換する必要があります。例えば，Fortran で `f(i,j)` と与えるには，C 言語では `f[j][i]` となります。3 次元配列も順序を逆にします。例えば，Fortran で `f(i,j,k)` と与えるには，C 言語では `f[k][j][i]` となります。
4. 文字列を与える場合にはそのまま与えても結構です。すなわち，Fortran で`'Data'` と与えるときは，C 言語では`"Data"`と与えます。

コンパイル・リンクするときには，以下の注意が必要です。

1. Fortran コンパイラによっては，コンパイル時にサブルーチン名にアンダースコア '_' を名前の前後に付けるため，C 言語から同じ名前でコールしてもリンクエラーになる場合があります。これを回避するために，C 言語プログラムの最初に，必ず `frames.h` をインクルードし，アンダースコアの付き方に応じたコンパイルオプションを付けて下さい。オプションは付録 B で示した通りです。例えば，`fr_graph_`のように名前の後ろに 1 個だけ付く場合には`-DNAMEU` というオプションを付けてコンパイルします。さらに，付属の `name.h` を使用しますので，これを `frames.h` と同じディレクトリに入れておく必要があります。(ユーザープログラムでインクルードする必要はありません)
2. 文字列は，Fortran と C で長さの定義が異なります。*FRAMES* でこれが問題になるのは文字列処理サブルーチンの入った `chr_for.f` というプログラムだけですので，これを `chr_c.f` に置き換えれば使えます。実際には `chr_c.f` を Fortran でコンパイルしてできる `chr_c.o` を `libfrX.a` より前に並べてリンクすればＯＫです。`chr_c.o` はライブラリ作成の際に `LIB=forc` を指定すれば同時に作成されます (1.4 参照)。
3. C 言語がメインルーチンの場合にはリンク時に Fortran ライブラリを明示的に付ける必要があり，これはコンパイラによって異なります。付属のメーク設定ファイル (`makefile.`*TARGET*) には機種やコンパイラに応じたライブラリリストが `LFLAGS` という名の環境変数で示されているので，それを参考にして下さい。ただし，`g77` は C のプログラムもコンパイルできるので，`g77` を使ってコンパイルすれば明示的に付加する必要はありません。このあたりもメーク設定ファイルを参照して下さい。

以下に 1 例を示します。

```
/*************************************************************
 *   gctest.c    Test for Frames called from C program       *
 *************************************************************/
#include <stdio.h>
#include <math.h>

#include "frames.h"
#define N           5
#define IMAX       20
#define JMAX       20

main()
{
    int n0=0,i,j,nn,n=N+1,ind=10,ix1=100,ix2=550,iy1=70,iy2=390;
    double x1[2],y1[N+1],y2[N+1];
    double xmin=0.0,xmax=2.0,ymin=-1.5,ymax=1.5;
    double x,y,dx,dy,dd,yy,x0=1,y0=0;
    double f[JMAX+1][IMAX+1];
```

```
    fr_ginit();
    fr_gfile("gtest_c",&n0);
    fr_frame2d(&xmin,&xmax,&ymin,&ymax,&ind);
    fr_xyname("X-axis","Y-datas");

    dx = (xmax-xmin)/N;
    for (i=0;i<=N;i++) y2[i]=0.1*sin(10*dx*i);

    nn=7;          dy=2.0/nn;
    x1[0]=xmin;    x1[1]=xmax;
    for (i=1;i<=nn;i++) {
        yy=dy*i-1.0;
        dd=20*dy*i;  fr_setcircle(&dd);
        for (j=0;j<=N;j++) y1[j]=y2[j]+yy;
        ind=101;     fr_graph2d(x1,y1,&n,&i,&ind);
        ind=103;     fr_graph2d(x1,y1,&n,&i,&ind);
    }

    fr_gpause();     fr_gclear();

    xmin=-3;     xmax=3;        ymin=-2;   ymax=2;
    dx=(xmax-xmin)/IMAX;        dy=(ymax-ymin)/JMAX;
    for (i=0;i<=IMAX;i++) {
        for (j=0;j<=JMAX;j++) {
            x=dx*i+xmin;    y=dy*j+ymin;
            f[j][i]=exp(-0.5*((x-x0)*(x-x0)+(y-y0)*(y-y0)))
                      -exp(-1.2*((x-1.5)*(x-1.5)+(y-0.5)*(y-0.5)));
        }
    }

    fr_view2d(&ix1,&iy1,&ix2,&iy2);
    x=1;       y=1;    ind=1;    fr_aspect2d(&x,&y,&ind);
    ind=0;    fr_frame2d(&xmin,&xmax,&ymin,&ymax,&ind);
    fr_xyname("Re","Im");
    ix1=IMAX+1;     iy1=JMAX+1;
    n=20;           ind=-2;
    fr_contour(f,&ix1,&iy1,&n,&ind);

    fr_gend();
}
```

このプログラム gctest.c を Linux でコンパイル・リンクするときは,

```
cc -DNAMEUU gctest.c -o gctest chr_c.o libfrX.a -L/usr/X11R6/lib -lX11 libgdfr.a -lg2c
```

とします。Linux/FreeBSD では名前の後ろにアンダースコアが 2 個付くので-DNAMEUU オプションを付けていること，chr_c.o を libfrX.a の前に付けていること，-lg2c オプションを付けることで Fortran ライブラリをリンクしていることに注意して下さい。

g77 を使うのであれば，Fortran ライブラリを明示的に付加する必要はありません。

```
g77 -DNAMEUU gctest.c -o gctest chr_c.o libfrX.a -L/usr/X11R6/lib -lX11 libgdfr.a
```

で OK です。

付録 F　サブルーチンの短縮名について

FRAMES v6.2 からライブラリサブルーチン名を，全てプレフィックス fr_がついたものに変更しました。これは v6.1 以前のサブルーチン名が「Fortran のサブルーチン名は 6 文字以内」という古い規約を守って付けていたため，VIEW とか DRAW とか誰もが付けそうな名前があって，たまにリンクエラーになったからです。しかし，長い名前は英語に慣れている人には覚えやすくても，そうでない人には返ってつづり間違いを起こす可能性があります。そこで，特にコンフリクトが起こらない限り「短縮名」のつもりで，旧バージョン名も使えるようになっています。また，一部のサブルーチンに関しては旧名称を変更したり，新設サブルーチンにもかかわらず短縮名を用意したものもあります。このため，必ずしも 6 文字以内という規約は守っていません。

なお，短縮名はバイパスルーチンによって実現していますので，もしお使いのマシンでコンフリクトが起きるようであれば，コンパイル時のオプションでライブラリから抜くこともできます。ライブラリから抜く方法は筆者までお問い合わせ下さい。

以下にサブルーチンの短縮名対応表を示します。

(1) 描画動作制御と *CANVAS* 関係のサブルーチン

短縮名	正式名	短縮名	正式名	短縮名	正式名
GINIT	fr_ginit	GEND	fr_gend	GWSIZE	fr_gwsize
GWSTEP	fr_gwstep	GPAUSE	fr_gpause	GCLEAR	fr_gclear
GREQUEST	fr_grequest	GUPDATE	fr_delayupdate	GNOTES	fr_gnotes
HLSSET	fr_hlsset	HLSRECALL	fr_hlsrecall	HLSQUERY	fr_hlsquery
GFILE	fr_gfile	GIFFILE	fr_gfilegif	FILESW	fr_filesw
GCANVAS	fr_canvas	GOUTLET	fr_outlet	GIFOPTION	fr_gifoption
PLTFILE	fr_pltfile	GIFIMAGE	fr_gifimage	GIFANIME	fr_gifanimation

(2) 2 次元図形描画のサブルーチン

短縮名	正式名	短縮名	正式名	短縮名	正式名
VIEW	fr_view2d	ASPECT	fr_aspect2d	FRAME	fr_frame2d
GRAPH	fr_graph2d	DRAW	fr_draw2d		
VECTOR	fr_vector2d	XYNAME	fr_xyname	STAXIS	fr_setaxis
CONTOR	fr_contour	CONTSQR	fr_sqrcontour	CONTTRI	fr_tricontour
FLOWLIN	fr_flow2dline	FLOWRGN	fr_flow2drgn	FLOWSPC	fr_flowspace

(3) 1 パラメータ図形と 3 次元図形描画のサブルーチン

短縮名	正式名	短縮名	正式名	短縮名	正式名
PMVIEW	fr_pmview	PMFRAME	fr_pmframe	PMGRAPH	fr_pmgraph
PMDRAW	fr_pmdraw	PMSET	fr_pmset	HCLEAR	fr_hclear
XYZNAME	fr_xyzname	PROFIL	fr_profile	VIEW3D	fr_view3d
ANGLE3D	fr_angle3d	ASPECT3D	fr_aspect3d	PROJMD	fr_project
FRAME3D	fr_frame3d	ROTATE	fr_rotate	GRAPH3D	fr_graph3d
DRAW3D	fr_draw3d	VECTOR3D	fr_vector3d	PROF3D	fr_profile3d
PROFTR	fr_triprofile	CONT3D	fr_contour3d	ISOSURF	fr_isosurface
FL3DLIN	fr_flow3dline	FL3DRGN	fr_flow3drgn	POLY3D	fr_3dpolygon

(4) 基礎図形・設定変更・その他のサブルーチン

短縮名	正式名	短縮名	正式名	短縮名	正式名
FPLINE	fr_fpline	FPCIRC	fr_fpcircle	FPPOLY	fr_fppolygon
FPMARK	fr_fpmark	FPDOTS	fr_fpdots	FPCHRS	fr_fpchars
G2MOVE	fr_2dmove	G3MOVE	fr_3dmove	COLBAR	fr_colorbar
STCIRC	fr_setcircle	STARROW	fr_setarrow	STMARK	fr_setmark
STSTEP	fr_setstep	STCONT	fr_setcontour	STBAR	fr_setbar
ST2DBOX	fr_set2dbox	ST3DBOX	fr_set3dbox		
ST2DRGN	fr_set2drgn	ST3DRGN	fr_set3drgn		

付録 G　*FRAMES*バージョン履歴

- 第 1 版 (1990 年某月)—
  (1) C 言語ルーチンの基幹部を Fortran 用に移植した。

- 第 2 版 (1991 年 5 月)—
  (1) １パラメータ図形群用の簡易３次元表示ルーチンを追加した。
  (2) ３次元形状表示用に等高線ルーチン (CONTOR) と鳥瞰図ルーチン (PROFIL) を追加した。
  (3) その他いくつかの便利なサブルーチンを追加した。
  (4) 全体的に高速化を図った。
  (5) いくつかのルーチンに対し自動スケーリングを付加した。

- 第 3.1 版 (1992 年 6 月)—
  (1) 描画データを自動的にディスクに落とす機能を追加した。
  (2) ディスクに落としたデータから再描画するアプリケーションを加えた。

- 第 4.0 版 (1992 年 12 月)—
  (1) ３次元図形描画グラフィックルーチンを追加した。
  (2) １次元や２次元配列の描画ルーチン (GRAPH，PROF3D，etc.) に対し，描画範囲指定ルーチンを追加した。
  (3) ２次元描画領域のアスペクト比指定を追加した。
  (4) 小円や矢印のサイズを変更するルーチンを追加した。
  (5) いくつかのルーチンのパラメータの定義を拡張した。
  (6) 描画一時停止ルーチンを追加した。

- 第 4.5 版 (1993 年 12 月)—
  (1) 内部ルーチンを全面的に書き換え，移植性を高めた。
  (2) 基礎図形ルーチンを追加した。
  (3) 対数軸ルーチンを加えた。

- 第 5.0 版 (1994 年 2 月)—
  (1) ３次元図形描画ルーチンに透視図法を追加した。

- 第 5.1 版 (1994 年 12 月)—
  (1) 256 色色指定モードを追加した。
  (2) 楕円描画を追加した。
  (3) 長方形，楕円，多角形の塗り潰し指定ができるようにした。
  (4) plot ルーチンを大幅に書き換えて，PostScript 出力の際，線種指定やグレースケール指定ができるようにした。また，ハードコピーやフィルターなどもかけられるようにした。

- 第 5.2 版 (1995 年 5 月)—
  (1) ３角形メッシュデータによる等高線作図ルーチンを加えた。
  (2) フォントの種類と取り扱い方法を変更した。

- 第 5.2.3 版 (1996 年 4 月)—
  (1) 等高線ルーチン (CONTOR，CONSQR，CONTRI，CONT3D) の色指定 ic=-1 の場合の色分割数を 16 色に変更した (以前は 6 色)。
  (2) ２次元４角形メッシュデータによる等高線作図ルーチンを加えた。
  (3) 等高線のデータ保存形式を変更した。
  (4) plot ルーチンにカラー PostScript 出力を加えた。

- 第 5.3.2 版 (1997 年 12 月)—
  (1) 256 色色指定モードを HLS 色指定に変更し，カラーマップを変更できるとともにデフォルト色数を 128 とした。
  (2) ２次元流線描画ルーチンを加えた。
  (3) ３角形メッシュデータによる３次元曲面の線画図形描画を加えた。
  (4) Toolkit 型 plot の tplot を付属した。

- 第 6.0 版 (1999 年 5 月)—
  (1) GCLS ルーチンに代わる GCLEAR ルーチンを加えた。
  (2) GWSIZE ルーチンにより，ウィンドウサイズを変更できるようにした。
  (3) STCIRC，STAROW の引数を実数に変更した。また，STCIRC を直径指定にした。
  (4) GRAPH，PMGRAF，GRAF3D に等間隔配列表示モードを新設した。
  (5) PROFIL 等の 1 パラメータ図形群描画と３次元図形描画の両方で使えるルーチンのデフォルトを３次元図形にした。
  (6) plt 形式ファイル抽出用にデータ埋め込みルーチンを新設した。
  (7) tplot と plot ルーチンを統合した。
  (8) plt 形式ファイルからデータを抽出する C 言語関数を付加した。
  (9) C 言語からコールするためのプログラムファイルを付加した。

- 第 6.1 版 (2000 年 5 月)—
  (1) tplot を大幅に改良し，描画時のスケール変換や視点変換，各種パラメータの変更などができるようにした。
  (2) tplot での PostScript コード出力を従来の標準出力送出からファイルまたはパイプ出力に変更した。
  (3) ３次元流線描画ルーチンを加えた。
  (4) CONT3D ルーチンの仕様を変更し，３次元データの断面等高線図が描けるようにした。
  (5) 通常描画時にキーボード操作で画面出力を中断してファイル出力のみにすることができるようにした。
  (6) 実座標指定による基準点変更ルーチン，G2MOVE および G3MOVE を新設し，これに合わせて，基礎図形ルーチンを変更した。
  (7) FPCHRS の接続指定方法を変更した。

- 第 6.2 版 (2001 年 6 月)—
  (1) 全てのライブラリサブルーチンの名称を fr_をプレフィックスとするものに置き換えた。
  (2) fr_gfile を改良し，複数の plt ファイルを切り換えて書き込むことが可能になった。
  (3) HLS 色設定ルーチンの指定を記録し，そのシリアル番号を呼び出すだけで HLS カラーテーブルを変更できる fr_hlsrecall ルーチンとその補助用の fr_hlsquery ルーチンを新設した。
  (4) tplot の操作パネルに文字列を表示できるノート機能 fr_gnotes ルーチンを新設した。
  (5) tplot にファイル選択ダイアログを付けた。
  (6) tplot に fr_gnotes ルーチンに対応する Notes ダイアログを付けた。
  (7) コンパイル時のオプションにより GIF ファイル作成機能を持たせることができるようになった。
  (8) tplot の-on オプションによりプリファレンスファイルを指定し，その中でデフォルトのスケールなどを変更できるようになった。

- 第 6.3 版 (2002 年 3 月)—
  (1) 等値面描画ルーチン fr_isosurface を新設した。
  (2) ３角形メッシュデータから曲面を描く fr_triprofile ルーチンに擬似ライティング表現を付加した。
  (3) 対数軸描画方式を変更し，軸指定ルーチン fr_setaxis を新設した。
  (4) 小円の代わりに正多角形や星形が描画できるようにした。
  (5) ３次元多角形描画ルーチン fr_3dpolygon を新設した。
  (6) fr_graph2d ルーチンなどでスプライン曲線，棒グラフが描けるようにした。

- 第 7.0 版 (2005 年 1 月)—
  (1) 仮想描画空間 *CANVAS* という概念を導入し，図形は *CANVAS* を通して出力装置に伝達される形式にした。これにより，複数の装置を自由に付けたり外したりすることができるようになった。
  (2) *CANVAS* の導入に伴い，GIF イメージ出力を *FRAMES* ライブラリから直接作成できるようにした。
  (3) `fr_pagesw`，`fr_gpage` ルーチンは廃止し，`fr_delayupdate` による描画ページ切り替えモードの変更という形式で簡易アニメーションを実現できるようにした。
  (4) 等高色分布図において，グリッドの頂点の色のみでそのグリッドの色を決定するモードを加えた。
  (5) 等高色分布図において最低値に対応する色は描かないモードを付加した。
  (6) HLS 色指定で同じ描画ページで異なる色指定をして重ね塗りをしてもハードウェアが描画可能な範囲であれば正常に色を付けることができるようにした。
  (7) `fr_xyname`，`fr_xyzname` ルーチンにおいて，描画ボックスを確定する前にコールしても正常に働くようにした。
  (8) いくつかの基礎図形ルーチンにおいて物理座標指定で図形を描けるようにした。

- 第 7.1 版 (2005 年 5 月)—
  (1) ウィンドウの処理を改良し，マルチウインドウの取り扱いができるようになった。これを用いて異なる *CANVAS* に異なるウィンドウを割り当てるようにした。
  (2) マルチウインドウを利用するために，ディスプレイのみの *OUTLET* を持つ簡易型 *CANVAS* 定義ルーチン `fr_window` を新設した。
  (3) *CANVAS* 定義にて背景色を白色にする白黒反転モードが指定できるようになった。
  (4) 基本色指定を 10 種類 (0-9) にし，絶対黒色と絶対白色設定ができるようにした。
  (5) ライティングの手法を改良し，`fr_profile3d`，`fr_triprofile`，`fr_isosurface` において，強度補間を用いたスムーズシェーディング表現ができるようにした。
  (6) ライティングパラメータを変更するための `fr_setlight`，`fr_setsurface` ルーチンを新設した。
  (7) `fr_profile3d`，`fr_triprofile` において点の高さに応じた色分布を付けることができるようになった。
  (8) `tplot` で 3 次元図形を再描画する時に，自動的に回転して見せる `Rotate` ボタンを付加した。

- 第 7.2 版 (2006 年 12 月)—
  (1) デフォルトの背景色を白にした。
  (2) 色の処理を改良し，1 ページ内で複数の HLS 色を指定したときに切り替えて動作できるようにした。
  (3) 鳥瞰図を描き，それを別のデータによって色分けをすることができる `fr_mapprofile` ルーチンを新設した。
  (4) `fr_profile3d`，`fr_triprofile` を改良し，メモリ割り当てを自動化してワーク領域を不要にした。但し，従来のバージョンとの互換性を保つため，これらのルーチンは残し，名称を変更してそれぞれ，`fr_surface`，`fr_trisurface` とした。
  (5) 3 次元曲面を描き，それを別のデータによって色分けをすることができる `fr_mapsurface` ルーチンを新設した。
  (6) 3 次元線分描画ルーチン `fr_3dline` を新設した。
  (7) `fr_graph3d` ルーチンに，多数の小球を遠近に応じて描画するモードを追加した。

あとがき

FRAMES は摂南大学の卒研生用に作成したもので，作り始めてから 10 年以上経過しています。もっとも，それ以前から研究補助用に PC-9801 用の C 言語ライブラリを作っていましたので，それを含めるとさらに長い年月手掛けていることになります。「簡単に２次元グラフや等高線が描けるパソコン用ライブラリが欲しい」,「簡単な３次元表示も欲しい」程度の動機で作り始めたのですが，シミュレーションの開発が進むにつれて,「もっとリアルな３次元が欲しい」,「対数座標も描きたい」などと要望が増え，サブルーチンがどんどん増えました。

FRAMES は 98 用でも MS-DOS 上の MS-FORTRAN 用とトランスピュータ用があり，サブルーチンが少なかった頃は，表面上は同じでも内部は個々の性能に合わせてできるだけ高速化しようとしていたものです。しかし，ライブラリが大きくなったので，描画空間座標で直線や文字を描く基本描画部（コアルーチン）と隠線処理や等高線処理などの図形処理ルーチンとを分離し，移植の際に生じる変更点をできるだけ減らすように改良しました。これにより，パソコン上での描画速度は若干犠牲になりましたが，別の OS への移植が容易になりました。私は Macintosh ファンなので Macintosh 用も作りました。

91 年にワークステーションを導入してから UNIX で計算することが中心になったので，X ウィンドウ上で動くものを作り始めました。初めて購入した IBM ワークステーションには graPHIGS というグラフィックライブラリがあり，最初はこれで動かしていました。graPHIGS は X ウィンドウ以外でも使える汎用の図形描画ライブラリで，X そのものを知らなくてもウィンドウへの描画プログラムを作ることができたのです。しかし graPHIGS では IBM 以外の X 端末から実行するとエラーになる場合があって，動作する X サーバーが限定されていました。

そこで 92 年に，当時卒研生だった鮫嶋君に「X ウィンドウを勉強しないか」と持ちかけ，プリミティブな X ウィンドウライブラリ (Xlib) を用いたコアルーチンを作ってもらうことにしました。彼は半年ほどかけ，改良を重ねてページングやイベント取得も含めたルーチンを作ってくれました。この X ルーチンのおかげで X 端末での問題はなくなり，より多くのマシンから同時に使うことができるようになったのです。なお，彼はこの X ウィンドウコアルーチンを作ることで卒論を書いたのではなく，ちゃんと別に卒業研究をしていましたから今考えてもよくできた学生でしたね。このコアルーチンは，その後色々と拡張しましたが，基本的な部分は同じです。彼とは卒業以来会っていませんが，きっといい仕事をしていることでしょう。

ということで，できあがった X ウィンドウ用 *FRAMES* ですが，最初は IBM のマシンしか無かったので，それほど意識していなかった他の UNIX マシンへの移植性を高めるきっかけを与えてくれたのが，姫路工業大学の坂上君 (現在，核融合科学研究所) でした。彼が X ウィンドウで動く Fortran 用のグラフィックルーチンを探していたので，*FRAMES* を送ったところ，自分の Alpha マシンで使えるように改良してくれたのです。これを機会に色々なマシンでのコンパイル・リンクを試みて，移植性の低い部分を改善していきました。また，彼は色々と厳しい要求をしてくるので，それに合わせて，HLS 色指定を作ったり `Makefile` を用意しました。おかげで随分勉強させられましたが，機能がアップすると同時に，他の UNIX マシンへの移植性が向上しました。

私は昔から Mac ファンであり，Windows は毛嫌いしている人間ですが，PC-9801 は Mac 以前から使っていたし，所有していたマシンも少なかったので UNIX マシン導入後も引き続きトランスピュータを入れて MS-DOS で使っていました。96 年に PC-9801 で動く FreeBSD の存在を知り，古い PC-9801 にインストールして X 端末として使ったのをきっかけに，それまでトランスピュータや Macintosh の Fortran も交えて卒研指導をしていたものを，全て UNIX でやることにしてしまいました。卒研生は横着な人間が多く，最初 MS-DOS で計算機を覚えてしまうとなかなかそこから脱け出せないようです。「どうせなら最初から UNIX にしてやればええやろ，エディタは `vi` だけど，どうせ他も知らんから一緒やろ」てなもんです。これ，結構正解でした。この UNIX への全面移行のおかげで，*FRAMES* の開発対象は X ウィンドウ用に限定することができ，改良もしやすくなりました。

UNIX に全面的に切り替えた当初は「UNIX は古いから卒業してから使えないのが欠点やなあ」という気分がちょっぴりあったのですが，そこへ押し寄せたのが Linux，FreeBSD の大波です。安い DOS/V マシンにフリーの UNIX を搭載すれば Windows NT よりもはるかに安定したサーバーが手に入るということで，色々なパッケージが雑誌の付録に付いてくるわ，初心者向けの本や解説記事が本屋にところ狭しと並んでいるわで,「これからは，やっぱり UNIX やで」と声高らかに言えるようになりました。学生にとっても家のパソコンにインストールすれば，C も Fortran も X ウィンドウも自由に使えるわけで，大学と全く同じ環境でプログラム開発ができるのです。

最近，坂上君は「ダイレクト可視化システム」というのを提唱しています。最近のスーパーコンピュータの進

歩は著しく，メモリ 1 TByte・演算速度 1 TFlops，というオーダーのものが利用可能になっています。ところが，「そんな大きなコンピュータで計算したとき，出力結果をどう解析するの？」というのが大問題になっています。なぜなら，計算はできてもその結果を出力して保存するのが大変だからです。数 TByte のハードディスクが必要になりますからね。また，そのような巨大なファイルを扱って画像処理するアプリケーションソフトはそんなにありません。そこで，出力は「数値データ」ではなく「GIF 等の画像イメージ」にしてしまえばよい，という考えが「ダイレクト可視化システム」です。スーパーコンピュータで直接イメージを作るわけで，GIF イメージは容量がかなり小さいので非常に有効な手段です。

坂上君のグループは，手始めに彼らが開発した GIF ライブラリ gdf を用いて `tplot` に GIF 出力機能を付加しました。この時，単一イメージだけではなく，時系列データを用いた GIF アニメーションも作成可能にしてくれました。ところが，時を同じくしてパソコンによるプレゼンテーション，いわゆる PowerPoint が流行してきます。このため作成した GIF アニメーションがプレゼンテーションにそのまま使えることになり，非常に良いタイミングでの改良になったのです。この `tplot` の GIF アニメーション作成機能にはずいぶんお世話になりました。

しかし，まず plt ファイルに出力してから GIF アニメーションに変換するため，シミュレーションデータが大きくなって中間ファイルである plt ファイル自体が ftp 転送に支障をきたすくらい大きくなると結構手間になってきました。坂上君たちは，*FRAMES* ライブラリを改良して Fortran プログラムから直接 GIF ファイルを作り出すルーチンを作成してくれたのですが，筆者はその仕様に若干不満があったため採用せず，自分で独自に直接出力ルーチンを作成しました。しかし筆者の拡張版は 1 種類の GIF ファイルしか作成できないものだったため，ファイル切り替え機能を持つ坂上バージョンに及ばず，逆に坂上グループには採用されないという 2 極化状態がしばらく続きました。

前回のバージョンアップの最大のポイントである仮想描画空間 *CANVAS* の導入は，この問題を解決すべく意を決してファイル出力部を大幅に改訂して達成したものです。元々，plt ファイルへの出力は PC-9801 用 *FRAMES* 時代からの遺産であり，とてもじゃないがこのままで色々なデバイスを使い分ける仕様に変更することはできなかったのです。逆に言えば，一番やりたくなかった改良であるとも言えます。しかし，スーパーコンピュータでのシミュレーションが日常の仕事になり，遠隔からのデータをいかに圧縮するかが重要課題になってしまったおかげで，背に腹は代えられずついにメスを入れたと言うところでしょうか。今回はさらに思い切ってウインドウシステムも改良し，*CANVAS* ごとに別のウィンドウを開くことができるようにしました。ライティングもそれなりに改良したし，*FRAMES* もそろそろ完成品に近づいてきたと思われます。

それにしても，こうして *FRAMES* の開発史を振り返ってみると，私の性格を反映してか，計画性がまるでないですね。必要に迫られて思い付くまま作っていたら，たまたま現れた良くできる学生が X 用を作ってくれたり，うるさい先生の要求に合わせて作っていたら移植性が高まったり。気がついたら結構良いものができていたというところです。しかもフリー UNIX という時流に乗れるものができていたとはラッキーとしか言いようがありません。そもそも，*FRAMES* というネーミングも大した意味はないのです。*FRAMES* の特徴が，最初に `FRAME` ルーチン (現在の `fr_frame2d`) で「枠」さえ決めれば，後は実数データをそのまま与えられるところにあったので，そのまま名前にしちゃっただけです。

最後に，Xlib 型 *FRAMES* の作成に大きな貢献をしてくれた鮫嶋君と，その厳しい要求で *FRAMES* を完成品に近づけてくれた坂上君に感謝いたします。また，バグの多いプログラムを我慢して使ってくださり，色々と助言を下さった摂南大学工学部電気電子工学科の大家先生にも感謝したいと思います。対数表示指定や `fr_isosurface` ルーチン作成は，大家先生の要望によるものです，さらに，`tplot` への GIF 追加機能を作成してくれた姫路工業大学情報工学科卒業生の藤井君にも感謝いたします。

(2005 年 5 月 10 日)

参 考 図 書

1. 「パソコン FORTRAN プログラミング」　穴吹雅敏・李 義頡・久保田美明著，東海大学出版会 1989 年

2. 「入門グラフィックス」　佐藤義雄著，アスキー出版局 1984 年

3. 「２＆３次元グラフィックス・ソフトの基礎と応用」　守川 穣著，ＣＱ出版社 1984 年

4. 別冊数学セミナー「コンピュータと数学３ 数学研究のためのコンピュータシステム」
石田晴久・唐木幸比古・米田信夫編，日本評論社 1985 年

5. 「PostScript チュートリアル&クックブック」　Adobe Systems 著，野中浩一訳，アスキー出版局 1989 年

6. 「X-Window Ver.11 プログラミング」　木下凌一著，日刊工業新聞社 1989 年

7. 「X Window ハンドブック」Oliver Jones 著，西村　亨監修，三浦明美・ドキュメントシステム訳，
アスキー出版局 1990 年

8. 「X ツールキット・イントリンシクス　プログラミング・マニュアル Second Edition」
Adrian Nye and Tim O'Reilly 著，今泉貴史監訳，ソフトバンク 1992 年

9. 「X アプリケーション・プログラミング Athena ウィジェット編」安居院　猛，永江孝規著，新紀元社 1992 年

10. 「X ウィンドウ・システムユーザーガイド第 3 巻」Valerie Quercia and Tim O'Reilly 著，大木敦雄監訳，
ソフトバンク 1993 年

11. 「X プログラミングことはじめ，Xaw の基礎③，④」瀬戸米治著，Unix User 2000 年 1 月号，2 月号，
ソフトバンク 1999，2000 年

FRAMES サブルーチンパッケージは作者の独断によりその仕様が予告なく変更されることがありますので，マニュアルは最新のものをご使用ください。

FRAMES サブルーチンパッケージは自由にコピーしてくださって結構ですが，作者の許可なく売買することは禁じます。また，*FRAMES* サブルーチンパッケージを使用した結果生じたいかなる問題に対しても作者はいっさい責任を持ちません。

質問や要望は気分次第で受け付けます。

摂南大学 理工学部 電気電子工学科　田口俊弘
E-mail: taguchi@ele.setsunan.ac.jp