地球流体電脳ライブラリ

らくらく DCL

地球流体電脳倶楽部

1995 年 6 月 9 日 (DCL-5.0)

目 次

1 はじめに **1**
1.1 地球流体電脳ライブラリ (DCL) . . . 1
1.2 DCL の背景 . . . 1
1.3 DCL の特徴と将来 . . . 2
1.4 DCL の構成 . . . 3
1.5 本書の構成 . . . 4

2 とりあえず使ってみよう **6**
2.1 とりあえず折れ線 . . . 6
2.2 とりあえず等高線図 . . . 8
2.3 とりあえず 2 次元ベクトル場 . . . 11

3 描画の基本 (1) **14**
3.1 仮想直角座標系 (V-座標系) での基本描画 . . . 14
3.2 ポリラインプリミティブ . . . 16
3.3 ポリマーカープリミティブ . . . 17
3.4 テキストプリミティブ . . . 19
3.5 トーンプリミティブ . . . 21

4 描画の基本 (2) **24**
4.1 ユーザー座標系 (U-座標系) での基本描画 . . . 24
4.1.1 基本概念 . . . 24
4.1.2 ウインドウとビューポート . . . 26
4.1.3 正規化変換の変換関数 . . . 28
4.2 もっとポリライン . . . 30
4.3 もっとテキスト . . . 33
4.4 アローとラインのサブプリミティブ . . . 35

5 レイアウト **39**
5.1 1 ページに複数の図形 . . . 39
5.2 マージンに文字列を書く . . . 41
5.3 紙を一杯に使う . . . 42

6 USPACK を一工夫 **45**
6.1 タイトルを描く . . . 45
6.2 USGRPH の分解 . . . 46
6.3 複数のデータを 1 つのグラフに描く . . . 47
6.4 等間隔データをおまかせにする . . . 48

7 座標軸 **52**
7.1 線形座標軸 . . . 52
7.2 対数座標軸 . . . 53
7.3 これはうれしい日付軸 . . . 54
7.4 中途半端なウインドウ . . . 56

8 もっと座標軸 **58**
8.1 注文の多い目盛りうち . . . 58
8.2 右も左も日付軸 . . . 59
8.3 好みの場所に好みの軸を . . . 61
8.4 同じ側に何本もの軸を . . . 62

9 2 次元量の表示 **65**
9.1 コンターラインをコントロール . . . 65
9.2 格子点が不等間隔の場合 . . . 68
9.3 配列の一部分だけを描く . . . 69

10 もっと 2 次元量表示 **72**
10.1 トーンパターンを指定する . . . 72
10.2 ベクトル場のスケーリング . . . 73
10.3 等高線とベクトル場の重ね書き . . . 75

11 欠損値処理いろいろ **78**
11.1 ポリライン ・ポリマーカーにおける欠損値処理 . . . 78
11.2 2 次元量表示における欠損値処理 . . . 80

12 カラーグラフィクス **82**
12.1 カラーマップ . . . 82
12.2 カラーライン . . . 83
12.3 カラートーン . . . 84

13 地図投影や 3 次元図形も **86**
13.1 いろいろな地図投影法 . . . 86
13.2 3 次元透視変換 . . . 91

14 グラフィクス以外にも **93**
14.1 MATH1 . . . 93
14.2 MATH2 . . . 94
14.3 MISC1 . . . 94
14.4 ETC . . . 95

15 付録 **96**
15.1 GRPH パッケージ一覧 96
15.1.1 GRPH1 96
15.1.2 GRPH2 96
15.2 フォント一覧 98
15.2.1 フォントテーブル 1 98
15.2.2 フォントテーブル 2 99
15.3 トーンパターン一覧 100
15.3.1 トーンパターンテーブル 0 100
15.3.2 トーンパターンテーブル 1 101
15.3.3 トーンパターンテーブル 2 102
15.3.4 トーンパターンテーブル 3 103
15.3.5 トーンパターンテーブル 4 104
15.3.6 トーンパターンテーブル 5 105
15.3.7 トーンパターンテーブル 6 106

謝辞 **107**

第 1 章

はじめに

この章では, まず, 地球流体電脳倶楽部の地球流体電脳ライブラリとは何なのか, その背景と特徴について, また, 我々の時代認識と将来展望について述べます. また, ライブラリの構成と本書の構成を概観します.

この章の詳細は, 「地球流体電脳ライブラリ 利用者の心得」を御覧下さい. dcl-x.x/doc/kokoroe/ の下に TEX ファイルがあります.

1.1 地球流体電脳ライブラリ (DCL)

地球流体電脳ライブラリ (DCL) は, 地球流体電脳倶楽部という集まりの下で, 地球流体力学関係の研究者有志が長年にわたって開発, 改良, あるいは収集してきた「研究者による研究者のためのライブラリ」です. また, このライブラリは, 急速に変化しつつある計算機環境に対応するために, 現在でもつねに開発, 改訂が行われている「生きているライブラリ」でもあります. 情報工学などとは畑違いの者にとってこのようなライブラリを維持することは決して楽なことではありませんが, 計算機の利用が不可欠となった現在, 我々はあえてこの困難な作業に取り組み, 確実な技術に基づく地球流体力学の着実な発展をめざしています.

我々地球流体電脳倶楽部は, このライブラリが多くの人に使われ, 地球流体力学の発展に寄与することを願っています. したがって, ライブラリの使用は無料です. しかし, このライブラリの開発には多くの人々の多大な労力が注ぎ込まれており, 現在でも全くのボランティアによる開発及び維持管理作業が行われていることに留意して, 常識的なマナーは守っていただきたいと思います. また, 当然のことながら, ライブラリの品質及びこれを使った結果に関して一切の保証はありません.

1.2 DCL の背景

DCL の開発グループは, 全て地球流体関係の研究者で構成されており, いわゆる計算機の専門家は一人もいません. そのようなグループが自力でこのようなライブラリを構築してきた背景には次のような事情があります.

近年の計算機の発達により, 地球流体関係の研究は他の研究分野と同様に, 計算機なしには研究ができないほど計算機に依存するようになってきました. 今では, パソコン, ワークステーション等の個人レベルの計算機から, スーパーコンピュータまでいろいろなものがあり, 一人の人間が複数のコンピュータを扱うことは当り前のことになってしまいました. しかし, それぞれの計算機には計算機に依存したソフトウエアが用意されているのが一般的であり, 同じような仕事をするのに計算機の数だけプログラムが必要になります.

これらのコンピュータは能力の差こそあれ, 計算機であることに変わりはないわけですから, それぞれの計算機において解を求めたりグラフを描いたりすることが同じようにできれば, 能率的に仕事ができると考えられます. そのためには, どんな計算機でも共通に使える標準的なライブラリが必要です.

しかし, そのようなライブラリを誰が開発し維持していくのでしょうか. 近年, いわゆる情報科学の進歩が非常に急速で, 10–20 年前とは状況が全く異なります. 現在の情報科学の最先端は, 人工知能やファジイ理論に代表されるように, これまで人間にしかできなかった複雑な事を機械にやらせる所に主な興味があり, 単純な計算を延々と繰り返す数値計算やグラフィクスツールの開発などは, すでに研究対象としては時代遅れのものとなりつつあります. これらを専門とする情報科学者は少なくなる一方であり, 従って, 我々地球科学を専門とする研究者が 「ライブラリの整備などはその専門家に委せておけばよい」と言っておられた時代は終りを告げたのです. 我々自身が数値計算やグラフィクスの専門家として自立していかなくてはならない時代になってしまったわけで, 実際問題として, それぞれのノウハウを自らの手で蓄積していく以外に方法はないように思われます.

さらに, 地球流体力学の本質に関わる重要な背景があります. それは, 計算機に対する依存度が高くなってきた結果, 比較的理論的な仕事でさえ 「他人が得た結果」を検証することがほとんど不可能になりつつあるということです. これは 「科学」としての地球流体力学の存在基盤そのものを危うくする由々しき事態です. 解析的な論文であれば, 論文に従って紙と鉛筆と忍耐力を頼りにその結果を確認することができますが, 数値計算に関する論文では, 解析的なものに比べて途中の情報が極端に少ないので, 論文だけの情報から追試を行うことはほとんど不可能です. これを解決するためには, プログラムそのものを公開して流通させる以外に方法はないと思われます. それには, まず, プログラムそのものを 「計算機に対する命令」ではなく, 「計算過程を記述したドキュメント」として考えることが絶対に必要でしょう. さらに, この時 「標準言語」としてのライブラリがあれば, このドキュメントはより簡潔なものとなり, 可読性が向上するはずです. すなわち, 研究者仲間で同じ 「標準言語」を使うことで, プログラムそのものを情報伝達の手段として使うことが可能となるわけです.

DCL は, このような背景を踏まえて, 地球流体関係の研究にたずさわる自分達自身がその必要とする 「どんな計算機」でも使える 「標準言語」としてのライブラリを 「我々の力」で構築したライブラリなのです. また, ライブラリに収められているソフトウェアには, 全く自力で開発したもの以外に Free Wear (フリーソウェア, 無料ではあるが著作権は放棄しない, 本ライブラリもそうである) や PDS (Public Domain Softwears) から拝借してきたもの, あるいは, それに手を加えたものが存在しています. これは, すでに存在している質の高いソフトウェアを積極的に研究者仲間に広めて 「標準言語」化しようとするものです. このことにより, 世間に流通しているソフトウェアとなるべく相性の良い状態を保ち, 我々が作る種々のプログラムの 「標準」度を高め, より広いコミュニティの研究者仲間に受け入れられるようにしていこうと思います.

1.3　DCL の特徴と将来

地球流体力学という分野における, 特に大学での研究教育活動の現状を反映して, DCL は次のような特徴を持っています.

1. 「標準言語」としてライブラリの体系がわかりやすく, 柔軟である.
2. プログラムが適当に構造化され, 可読性が高い.
3. 複数の計算機上で同じプログラムが実行できる.
4. 標準的に FORTRAN 言語を使う.

5. 高品質な 2 次元の図形出力がサブルーチンコールの形でできる.

6. 欠損値の存在するデータ (観測値) を, そのまま扱える.

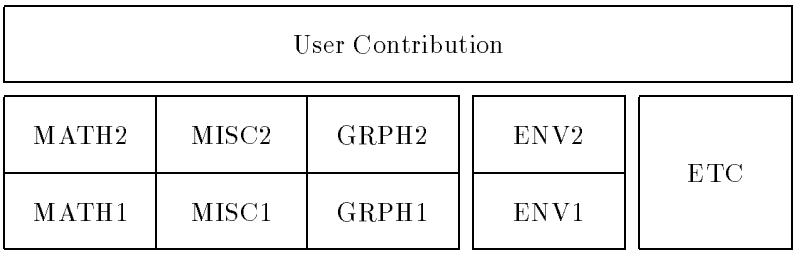
このうち 1, 2 の特徴は大学で使うという事情を反映したものであり, 3, 4 は現在の我々を取り巻く計算機環境に関係しています. また, 5, 6 は, 常に計算機の能力いっぱいの仕事が存在するという地球流体力学の特徴に関係するものです.

DCL は IBM 型メインフレームと共に育った世代の手によるものです. したがって, メインフレーム+FORTRAN という組み合せにおいては, 洗練された完成度の高いライブラリであると自負しています. 現在, DCL はワークステーション+FORTRAN という組み合せを標準的な計算機利用方法として想定していますが, これが将来にわたって標準的な形態であるとは思えません. FORTRAN の新しい規格である FORTRAN90 が FORTRAN77 に取って代るほど普及するかどうかはわかりません. また, ワークステーション+C という組み合せによる計算機利用スタイルが急速に普及しつつある現在, 世の中の計算機事情に応じて, ライブラリ自身も変化していかなければならないでしょう.

もっとも, 完成度の高いライブラリを書くためには, その言語の特質を知りつくしていなければならないので, 急に DCL が変化していくとは思えませんが, 時代の流れは注視していく必要があると考えています. 我々はあくまでも計算機を使う側の人間であり, 計算機技術に関して流れを変えることはできないのでしょうから.

1.4 DCL の構成

DCL は, 下図のようにいくつかの箱に分けて管理されています.



ここで, 左下の 6 つの箱 (MATH1, MATH2, MISC1, MISC2, GRPH1, GRPH2) は DCL の本体部分です. ライブラリに含まれるサブルーチンや関数の FORTRAN プログラムは全てこの中にあります. この本体部分は 「機能」により横に 3 つ, 処理の複雑さによる 「レベル」により上下に 2 つ, 計 6 つの箱にわかれています.

機能による分類で, MATH というのは数学的な処理, MISC は I/O 処理等その他の処理, GRPH は図形処理をさします. レベル分けの基準となる 「複雑」さはあくまで相対的なものですが, 以下のような基準を満たすように分類されています.

- レベル 1 のルーチンはレベル 2 のルーチンを呼んではならない. (下克上禁止の原則)
- レベル 2 のルーチンは他の機能のレベル 2 のルーチンを呼んではならない. (機能独立の原則)
- レベル 2 のルーチンは機種依存してはならない. (汎用性の原則)

これらの基準により, レベル 1 のルーチンはレベル 2 のルーチンがなくともその動作が保証され, また, レベル 2 のルーチンは他の機能のレベル 2 のルーチンと独立に扱うことが可能になります. さらに, 特定の計算機に移植する際には, レベル 1 のルーチンが移植できれば, 自動的にレベル 2 のルーチンも動作します. さらに, レベル 1 の箱の中では, MATH1 < MISC1 < GRPH1 のような形でパッケージ間の 「格付け」がなされており, 箱どうしの依存関係ができるだけ簡単になるように分類されています.

本体部分の隣にある ENV1, ENV2 は電脳ライブラリ本体を使用する際の環境を整えるために用意されています. ENV1 には主としてインストールに必要となる道具類や基本データが収められており, ENV2 には DCL を使ったユーザープログラムを実行する時に使うようなユーティリティプログラムが収められています. また, ETC には, 必要不可欠なものではありませんが, あると便利な道具類が入っています.

DCL のマニュアルは, 箱とレベルで分類されたパッケージごとに分冊化されています. つまり, MATH1, MATH2, MISC1, MISC2, GRPH1, GRPH2, それに, ETC があります. 環境設定のためのパッケージ ENV1, ENV2 に収められているソフトウェアは機種依存する部分が多いので, これらのマニュアルはまとめて 「機種別手引」に記載されています. また, 全体にわたるものとして, 「利用者の心得」「サンプル集」および初心者向けの 「ごくらく DCL」と 「らくらく DCL」があります.

これらのドキュメントはそれぞれ dcl-x.x/doc/ の下に分けて置かれています. すべての文章は LaTeX で書かれており, 簡単に再出力することができます. ただし, ETC に収められている地球流体電能倶楽部の標準スタイル `dennou.sty` を使っていますので, TeX でコンパイルする際にはそれを取り込むことが必要です.

1.5　本書の構成

本書は, DCL がすでにインストールされている環境で, このライブラリを使い始めようとする人を念頭に置いて書かれています. サンプルプログラムはすべて FORTRAN で書かれており, その知識を前提としています.

本書ではグラフィクス部分だけを解説しています. 第 2 章では, DCL グラフィクスの雰囲気を概観し, 第 3 ～ 5 章では, GRPH1 の基本的な部分について解説します. また, 第 6 ～ 10 章では, 上位ルーチン群である GRPH2 の各パッケージを解説します. さらに, 第 11 ～ 13 章では欠損値処理やカラーグラフィクスなどの応用的な部分を紹介します.

グラフィクス以外の部分については最後の章で簡単に触れます. MATH 部分はこれからどんどん充実させていきたいところであり, やがて, 「らくらく MATH」が出るくらいに取り組んでいきたいものです.

計算機のなまり 1

FORTRAN の方言

計算機言語には各国の標準機関 (日本では JIS) 及び国際標準機関 (ISO) により定められた規格があります. FORTRAN も例外ではなく, いわゆる FORTRAN77 の規格があります.

FORTRAN コンパイラは通常,　標準語」としての FORTRAN77 規格 $+\alpha$ の機能を持っています. この $+\alpha$ が拡張機能と呼ばれるものです. このような FORTRAN77 上位互換コンパイラは, コンパイラとしての機能は高いのですが, 逆に, $+\alpha$の機能を使ったプログラムは, そのコンパイラでないとコンパイルできないことになります. そういうわけで, ここでは$+\alpha$の部分を　方言」と呼びます.

例えば, INCLUDE 文や 4 倍精度の実数, 行末コメントなどは, 良く知られた　方言」です. このような FORTRAN の　方言」は特定の計算機を使いこなすには便利な場合も多いのですが, そのプログラムが他の計算機で動く保証はどこにもありません. プログラムの可搬性 (他の計算機への移植のしやすさ) を高めるには, そのプログラムを　標準語」化する必要があります.

しかし, どんなコンパイラでも大なり小なり　方言」を持っており,　方言」を含まないプログラムを書くことは容易なことではありません. そもそも, ある 1 つの計算機しか使ったことがなければ, 何が　標準語」で何が　方言」かを見分けることすら困難です.

DCL は, 極力この　方言」を避けるように努めています. どうしても　方言」を使わなければならないところでも直接　方言」は使わないで,　電脳標準語」を定義して各プログラムの中ではこの　電脳標準語」を使うようにしています. この　標準語」化の努力が DCL の最大の特徴であり, DCL の可搬性を生み出しているのです.

第 2 章

とりあえず使ってみよう

DCL グラフィクスではサブルーチンを数行呼ぶだけで, 折れ線図や等高線図, 2 次元ベクトル場などの作図が可能です. とりあえず, 気楽に使ってみましょう.

2.1　とりあえず折れ線

データ解析でも数値計算でも一刻も早く計算結果が見たいものです. 面倒な FORMAT を考えて, WRITE 文でたくさんの数字列をアウトプットして, それをじっくり眺めて···というようなことをやっていた時代もそんなに昔のことではないのですが, そんな時, DCL を用いるとわずか数行でデータをグラフ化できます.

まず, 例題として, カオスを生み出す簡単なロジスティク模型を考えてみましょう.

$$y_{n+1} = r y_n (1 - y_n). \tag{2.1}$$

$r > r_\infty = 3.5700$ でカオス解となりますが, 次の FORTRAN プログラム QUICK1 は, パラメータ $r = 3.7$, 初期値 $y_0 = 0.5$ として, y_{50} まで求めて, グラフ化しようというものです. 今は「おまじない」である GROPN 等を含めても 14 行めからの 4 行で, 自動的にスケールを決めて作図してくれます.

```
1         PROGRAM QUICK1
2
3         PARAMETER( NMAX=50 )
4         REAL X(0:NMAX), Y(0:NMAX)
5
6   *-- ロジスティク模型 ----
7         R    = 3.7
8         X(0) = 0.0
9         Y(0) = 0.5
10        DO 10 N=0,NMAX-1
11          X(N+1) = X(N) + 1.0
12          Y(N+1) = R*Y(N)*(1.-Y(N))
13    10  CONTINUE
14
15  *-- グラフ ----
16        CALL GROPN( 1 )
17        CALL GRFRM
18        CALL USGRPH( NMAX+1, X, Y )
19        CALL GRCLS
20
21        END
```

UNIX システムで DCL が標準的にインストールされている場合には,

```
% dclfrt -o quick1 quick1.f
```

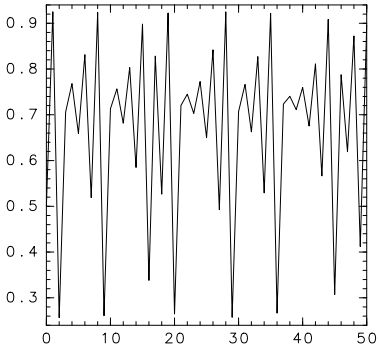
によって quick1 という実行ファイルが作られます. X ウインドウシステムが起動されている状態で

```
% quick1
```

といれると, ウインドウがひとつ現れて, ウインドウの位置を確定すると描画がはじまり, 下のようなグラフが得られます. このとき, 次の警告メッセージが出ることが多いと思いますが, 特に気にする必要はありません.

```
*** WARNING (STSWTR) *** WORKSTATION VIEWPORT WAS MODIFIED.
```

図形表示の終了はマウスクリックで行ないます.



quick1.f: frame1

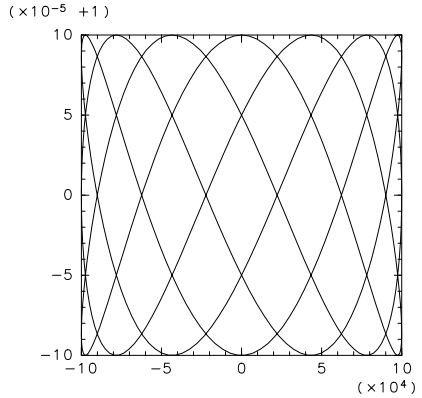
「おまじない」の GROPN および GRFRM サブルーチンで図形出力装置を準備し, 作画領域を設定します. 実際にスケーリングを行なって折れ線や座標軸を描いたのは, 自動スケーリングパッケージ USPACK のなかの USGRPH というサブルーチンです. このサブルーチンは, (x, y) 座標のデータとその個数 (NMAX+1) を与えると, それらを実線で結んでプロットします. 描画終了の「おまじない」が GRCLS ルーチンです. これらの「おまじない」については, 第 3, 6章で説明します.

つぎに, USGRPH ルーチンの自動スケーリング機能の実力を示すために, プログラム QUICK2 で意地の悪いデータを与えて作図してみましょう.

```
1         PROGRAM QUICK2
2
3         PARAMETER( NMAX=400 )
4         REAL X(NMAX), Y(NMAX)
5
6   *-- リサジューの図 ----
7         DT = 3.14159 / (NMAX-1)
8         DO 10 N=1,NMAX
9           T = DT*(N-1)
10          X(N) = 1.E 5*SIN( 6.*T)
11          Y(N) = 1.E-4*COS(14.*T) + 1.
12      10 CONTINUE
13
14  *-- グラフ ----
15        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
16        CALL SGPWSN
17        READ (*,*) IWS
18
19        CALL GROPN( IWS )
20        CALL GRFRM
21        CALL USGRPH( NMAX, X, Y )
```

```
22          CALL GRCLS
23
24          END
```

この例ではデータの変動幅が極端なのですが, 結果はどうでしょうか. x 軸のラベルが重なったり, y軸に 1.00005 というような不細工なラベルを付けて, それが描画範囲を越えてしまったりという様なことは起こりません. 与えられたデータから適当な目盛間隔やラベルの間隔を求めて座標軸を描いているのですが, ラベルの文字数が大きすぎる場合には, この例のようにファクター値 $(10^4, 10^{-5})$ やオフセット値 (1) を適当に選んで軸の端に表示し, ラベルが適当な文字数以内におさまるようにします.



quick2.f: frame1

QUICK2 のグラフ部分では 15 行めからの 3 行が増えていますが, これで図形の出力先を実行時に決められるようになります. 実行ファイルを作りこれを実行すると, 標準的な場合には

```
WORKSTATION ID (I) ? ;
1:X, 2:PS, 3:Tek ;
```

ときいてきます. GRPH1 のサブルーチン SGPWSN を呼んで, 今の環境で利用可能な 『ワークステーション名」のリストを書き出しています. この場合, 3 つの出力先が可能で, X ウインドウシステムが起動されている状態で 1 を入力すると, QUICK1 の例と同様にウインドウが現れます. 一方, 2 (PS) を指定すると, カレントディレクトリに dcl.ps というポストスクリプトファイルができます. そこで,

```
% lpr dcl.ps
```

と入力すれば, ポストスクリプトプリンタに結果が出力されます. また, 3 (Tek) を指定するとテクトロ端末で描画ができます.

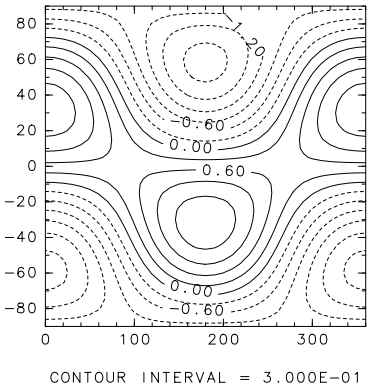
自動スケーリング機能の USPACK サブルーチンを使って, さらに手のこんだ 1 次元データの作図が可能です. それは, 第6章で説明することにしましょう.

2.2 とりあえず等高線図

2 次元のスカラー場を手早くコンタリングして見たいというときには, 等高線描画パッケージ UDPACK にあるサブルーチン UDCNTR を 1 つ呼ぶだけで十分です. 例題として, 球面調和関数の重ね合わせ $P_2^1(\sin\phi)\exp(i\lambda)-$

$P_2(\sin\phi)$ の実部を描いてみましょう (QUICK3).

```
1         PROGRAM QUICK3
2
3         PARAMETER( NX=37, NY=37 )
4         PARAMETER( XMIN=0, XMAX=360, YMIN=-90, YMAX=90 )
5         PARAMETER( PI=3.14159, DRAD=PI/180 )
6         REAL P(NX,NY)
7
8     *-- 球面調和関数 ----
9         DO 20 J=1,NY
10          DO 10 I=1,NX
11            ALON = ( XMIN + (XMAX-XMIN)*(I-1)/(NX-1) ) * DRAD
12            ALAT = ( YMIN + (YMAX-YMIN)*(J-1)/(NY-1) ) * DRAD
13            SLAT = SIN(ALAT)
14            P(I,J) = 3*SQRT(1-SLAT**2)*SLAT*COS(ALON) - 0.5*(3*SLAT**2-1)
15      10    CONTINUE
16      20  CONTINUE
17
18    *-- グラフ ----
19        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
20        CALL SGPWSN
21        READ (*,*) IWS
22
23        CALL GROPN( IWS )
24        CALL GRFRM
25
26        CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
27        CALL GRSVPT(  0.2,  0.8,  0.2,  0.8 )
28        CALL GRSTRN( 1 )
29        CALL GRSTRF
30
31        CALL USDAXS
32        CALL UDCNTR( P, NX, NX, NY )
33
34        CALL GRCLS
35
36        END
```



quick3.f: frame1

UDCNTR は等高線図を描くだけで, 枠や座標軸を描いてくれませんから, この例ではまず, いくつかの「おまじない」(GRSWND, GRSVPT, GRSTRN, GRSTRF) で座標系を決め, USPACK のルーチン USDAXS を使っておまかせの座標軸を描画したのちに, UDCNTR を呼んで等高線図を描いています.

UDCNTR の引数は, 最初の P が作画しようとする場を与える 2 次元配列, 2 番目の NX が実際に宣言した配列の第 1 次元寸法, 3, 4 番目の NX, NY がそれぞれ, 作画に使う配列の第 1 次元および第 2 次元寸法です. この例のように, 宣言した配列をフルに用いて作画するときは, 2 番目と 3 番目の引数は同じになりますが,

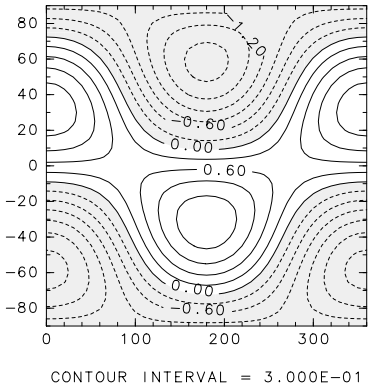
第 1 次元目の寸法をこのように 2 つの引数で指定するといいことがあります. その利点は第 9.3節で説明します.

この例のように UDCNTR だけを呼ぶ場合には, 等間隔の格子点を設定してコンタリングを行ないます. つまり, 座標軸の目盛に関係なく, P(1,1) が左下隅, P(NX,NY) が右上隅にくるような等間隔の格子点座標が設定されます. また, コンターレベルも自動的に決定されて, 図の下にそのコンター間隔が表示されます.

GRPH1 の ユーザーインターフェイス SGPACK には, 折れ線を描いたり文字列を描いたりする機能の他にも, 多角形閉領域のぬりつぶしを行なう機能があります. UEPACK はこれに対応するトーンぬりつぶしの上位ルーチンですが, QUICK3 の例で UETONE サブルーチンを 1 つ呼ぶだけで, とりあえず負の領域に斜線のハッチをつけることができます (QUICK4). ここで UETONE を呼ぶのが 31 行めで, 座標軸やコンターを描くよりも前であることに注意しておきましょう. これらの順序を変えると, 出力装置によっては結果が異なります. それまでに描いた部分がぬりつぶされ消されてしまうことがあるのです. この点については, 第 3.5節, 第 10.1節で詳しく説明します.

UDPACK や UEPACK, UWPACK の諸機能を使うことによって, 高度な等高線図が描けるようになります. カラーグラフィクスが利用できる環境では, このような等高線図を色の塗り分け (カラー グラデーション) で表現することも可能になります. これらは, 第 9, 10, 12章で説明することにしましょう.

```
1         PROGRAM QUICK4
2
3         PARAMETER( NX=37, NY=37 )
4         PARAMETER( XMIN=0, XMAX=360, YMIN=-90, YMAX=90 )
5         PARAMETER( PI=3.14159, DRAD=PI/180 )
6         REAL P(NX,NY)
7
8   *-- 球面調和関数 ----
9         DO 20 J=1,NY
10          DO 10 I=1,NX
11            ALON = ( XMIN + (XMAX-XMIN)*(I-1)/(NX-1) ) * DRAD
12            ALAT = ( YMIN + (YMAX-YMIN)*(J-1)/(NY-1) ) * DRAD
13            SLAT = SIN(ALAT)
14            P(I,J) = 3*SQRT(1-SLAT**2)*SLAT*COS(ALON) - 0.5*(3*SLAT**2-1)
15      10    CONTINUE
16      20  CONTINUE
17
18  *-- グラフ ----
19        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
20        CALL SGPWSN
21        READ (*,*) IWS
22
23        CALL GROPN( IWS )
24        CALL GRFRM
25
26        CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
27        CALL GRSVPT(  0.2,  0.8,  0.2,  0.8 )
28        CALL GRSTRN( 1 )
29        CALL GRSTRF
30
31        CALL UETONE( P, NX, NX, NY )
32        CALL USDAXS
33        CALL UDCNTR( P, NX, NX, NY )
34
35        CALL GRCLS
36
37        END
```



quick4.f: frame1

2.3　とりあえず 2 次元ベクトル場

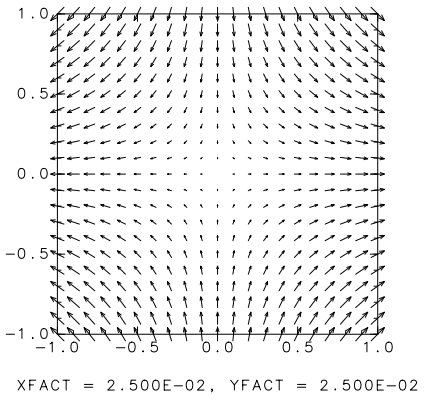
今度は, 2 次元のベクトル場を手早く矢印で見たいというときの例題です. 次の QUICK5 は簡単な変形場を描くプログラムですが, GRPH2 の UGPACK にあるサブルーチン UGVECT 1 つを呼ぶだけで十分です. 前節の等高線図の場合と同様に, おまかせの座標軸を描画したあとで UGVECT ルーチンを呼んでベクトル場を描いています.

この例では, やはり, 等間隔の格子点を設定して, それぞれの格子点でのベクトルを矢印で表現します. おまかせ描画のときには, ベクトルの長さが格子点間隔を越えないようにスケーリングファクターが決定され, それを乗じてベクトルが描かれます. この場合, x 成分と y成分のスケーリングファクターは同じになっていて, 図の下部マージンにはその値が表示されています.

ベクトル場も UGPACK のルーチンを使うことによって高度な作図ができるようになります. これも第 10 章で説明することにしましょう.

```
1         PROGRAM QUICK5
2
3         PARAMETER ( NX=21, NY=21 )
4         PARAMETER ( XMIN=-1, XMAX=1, YMIN=-1, YMAX=1 )
5         REAL U(NX,NY), V(NX,NY)
6
7   *-- 2 次元ベクトルデータ: 変形場 ----
8         DO 20 J=1,NY
9           DO 10 I=1,NX
10            X = XMIN + (XMAX-XMIN)*(I-1)/(NX-1)
11            Y = YMIN + (YMAX-YMIN)*(J-1)/(NY-1)
12            U(I,J) =  X
13            V(I,J) = -Y
14      10    CONTINUE
15      20  CONTINUE
16
17  *-- グラフ ----
18        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
19        CALL SGPWSN
20        READ (*,*) IWS
21
22        CALL GROPN( IWS )
23        CALL GRFRM
24
25        CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
26        CALL GRSVPT(  0.2,  0.8,  0.2,  0.8 )
```

```
27          CALL GRSTRN( 1 )
28          CALL GRSTRF
29
30          CALL USDAXS
31          CALL UGVECT( U, NX, V, NX, NX, NY )
32
33          CALL GRCLS
34
35          END
```



quick5.f: frame1

計算機のなまり 2

実数の内部表現

整数の内部表現は, ほとんどの機種で同一の表現となっているのに対して, 実数型変数に関してはいくつかの規格が存在します.

一般に, 実数型の変数は浮動小数点の形で表現されます. すなわち, β進法を使った場合, 実数は

$$\pm(0.f_1f_2f_3\cdots f_m)_\beta \times \beta^{\pm E}$$

という形で表されています. ここで,

$$\pm(0.f_1f_2f_3\cdots f_m)_\beta = \pm(f_1 \times \beta^{-1} + f_2 \times \beta^{-2} + f_3 \times \beta^{-3} + \cdots f_m \times \beta^{-m})$$

は仮数部で, f_iは 0 から$\beta - 1$ までの整数 $(f_1 \neq 0)$ です. また, $\beta^{\pm E}$は指数部で, Eは 0 または正の整数です.

主要な実数表現として, IBM 形式と IEEE(アイ ・トリプル ・イーと読む) 形式とがあります. どちらも, 32 ビットを 1 語とする点では同じですが, 採用されている進法が異なり, 表現できる実数の範囲や精度に違いがあります.

IBM 形式は 16 進演算が基本になっていて, 各ビットの使い方は以下の通りです.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
±	$E+64$							f_1				f_2				f_3				f_4				f_5				f_6			

IEEE 形式は 2 進法の浮動小数点形式で, IBM 形式に比べて相対精度が高いという特徴があります. 各ビットの使い方は以下の通りです.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
±	$E+126$								$f_2 \cdot\cdot f_4$			$f_5 \cdots f_8$				$f_9 \cdots f_{12}$				$f_{13} \cdots f_{16}$				$f_{17} \cdots f_{20}$				$f_{21} \cdots f_{24}$			

このように, 同じビット数で実数を表現しても, 実数として表現できる範囲や精度はシステムによって異なります. IBM 形式で表される 0 でない実数は, 絶対値が $5.397605 \times 10^{-79} \sim 7.237005 \times 10^{75}$の範囲であり, 相対誤差は $6 \times 10^{-8} \sim 10^{-6}$ 程度です. 一方, IEEE 形式では, 絶対値が $1.40129846 \times 10^{-45} \sim 3.40282347 \times 10^{38}$ の範囲の実数が表現できて, 相対誤差は $3 \times 10^{-8} \sim 6 \times 10^{-8}$ 程度です.

DCL では, このようなシステム依存の定数を MATH1/SYSLIB の GLRSET/GLRGET で管理しています. また, システムに依存する実数表現を解釈する道具が REALLIB です.

詳細は, MISC1 のマニュアル (dcl-x.x/doc/misc1/gaiyou/real.tex) を御覧下さい.

第 3 章

描画の基本 (1)

これからの 2 章では, いくつかの簡単なプログラム例を用いて, GRPH1 の中のユーザーインターフェイスパッケージ SGPACK の基本的な機能を説明しましょう. まず, この章では, 各種出力プリミティブとその属性について述べ, 次章では, 座標系 ・座標変換と出力プリミティブのちょっと進んだ内容について述べることにします.

出力例の図は縮小されているので線の太さなどの見た目が実際の出力結果と異なる場合があります. ご注意下さい. FORTRAN プログラムが dcl-x.x/demo/rakuraku/kihon/ にありますので, 実際に試してみて下さい.

3.1 仮想直角座標系 (V-座標系) での基本描画

まず最初のプログラム KIHON1 を見て下さい.

```
1          PROGRAM KIHON1
2
3          PARAMETER( NMAX=50 )
4          REAL X(0:NMAX), Y1(0:NMAX), Y2(0:NMAX), Y3(0:NMAX)
5
6          DO 10 N=0,NMAX
7            X (N) = REAL(N)/REAL(NMAX)
8            Y1(N) = X(N)**3
9            Y2(N) = X(N)**2
10           Y3(N) = SQRT(X(N))
11       10 CONTINUE
12
13         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
14         CALL SGPWSN
15         READ (*,*) IWS
16
17         CALL SGOPN( IWS )
18         CALL SGFRM
19         CALL SLPVPR( 1 )
20
21         CALL SGPLV( NMAX+1, X, Y1 )
22         CALL SGPMV( NMAX+1, X, Y2 )
23         CALL SGTXV( 0.5, 0.5, 'SGTXV' )
24         CALL SGLSET( 'LSOFTF', .TRUE. )
25         CALL SGTNV( NMAX+1, X, Y3 )
26
27         CALL SGCLS
28
29         END
```

図形を描くためには, まずディスプレイやプリンタなど図形出力装置 (デバイス) の準備をします. この準備をする時の操作を 「オープン」といい, SGOPN というサブルーチンを呼びます. ここで, 引数 IWS はワークステーション番号で, 各デバイスに割り当てられた番号です. IWS>0 のときは画面を横長に使い, IWS<0

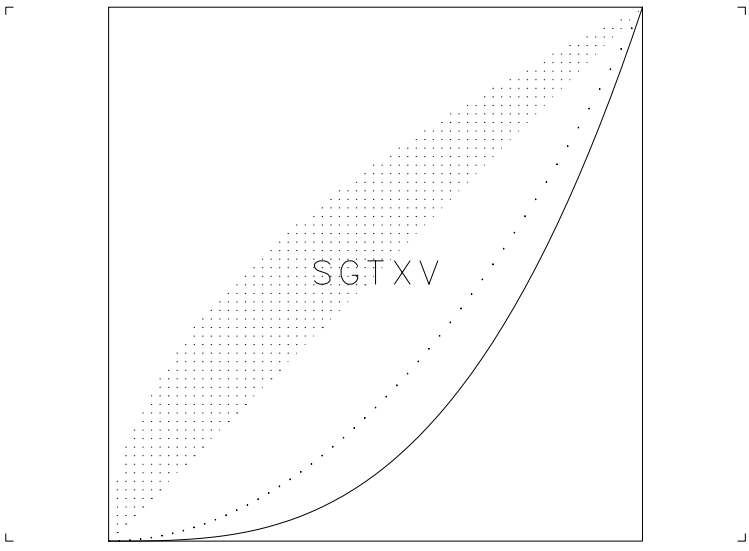
のときは画面を縦長に使って図形出力は 90 度回転して表示します. 14 行めのサブルーチン SGPWSN を呼ぶと, それぞれの環境で利用可能なワークステーション名のリストが書き出されます.

第 2章では GROPN だったのが, SGOPN になっています. 以下の SGFRM や SGCLS なども同様ですが, SGxxx が GRPH1 のコントロールルーチンであるのに対して, GRxxx はそれぞれに対応する GRPH2 のコントロールルーチンなのです. GRxxx ではいくつかの初期化処理も同時に行なっているので, 第 2章のように GRPH2 のルーチン (例えば, QUICK1 では USGRPH) を使う場合には, こちらを使います. 一方, この章では GRPH1 で閉じたルーチン群で説明をするので, SGxxx を用いています.

つぎに, SGFRM で新しい作画領域を設定します. DCL ではいくつかの座標系と座標変換が用意されていて, それぞれの座標系で作図ができます. しかし, それらの説明は次章にまわすことにして, ここでは $[0,1]\times[0,1]$ の仮想直角座標系 (これから V-座標系と略記します) だけを陽に考えましょう. SGFRM を呼ぶことにより, 各デバイスの作画できる領域に最大内接するようにこの仮想直角座標系が設定され, この範囲がとりあえず 「ビューポート」となります. SLPVPR ルーチンでこのビューポートの枠を描いています.

図形を構成する基本要素を出力プリミティブといいます. GRPH1 の出力プリミティブには, ポリライン (折れ線), ポリマーカー (マーカー列), テキスト (文字列), およびトーン (多角形のぬりつぶし) の 4 つがあり, 補助的に, アロー (矢印) とライン (線分) のサブプリミティブがあります. 結果の図を参照すれば明らかですが, SGPLV ルーチンで 3 次関数の折れ線が実線で描かれ, SGPMV ルーチンでは 2 次関数が ・のマーカー列として描かれます. また, SGTXV ルーチンで文字列 'SGTXV' がビューポートの真ん中に描かれます. さらに, SGTNV ルーチンでは, $y=\sqrt{x}$ と $y=x$ で囲まれた領域が点々でぬりつぶされます. とりあえず, 24 行めのサブルーチンコールはこのトーンの 「おまじない」と思って下さい. これらのプリミティブにはそれぞれいくつかの属性があり, それらを陽に指定することにより, 多種多様な作図が可能となります.

そして, 最後に描画を終了する時の操作を 「クローズ」といい, SGCLS ルーチンを呼びます. すべての図形は, デバイスをオープンしてからクローズするまでの間に描かれることになります.



kihon1.f: frame1

デバイスに出力される図形が複数 「ページ」にわたることがありますが, DCL では, ページという言葉の代わりに 「フレーム」という言葉を使います. これは, デバイスによってはページという概念のないもの (例

えば, 巻物のような長い紙に出力するようなデバイス) があったり, 後述のように物理的な 1 ページの中に複数のフレームを設定することもあるからです.

3.2　ポリラインプリミティブ

折れ線を描くポリラインプリミティブには, 次の 2 つの属性があります.

- ラインインデクス (折れ線の太さ ・色)
- ラインタイプ (折れ線の線種)

カラーグラフィクスについてはまとめて第 12章で説明しますので, ここでは線の太さだけが変えられる出力装置を念頭に置いて説明しましょう.

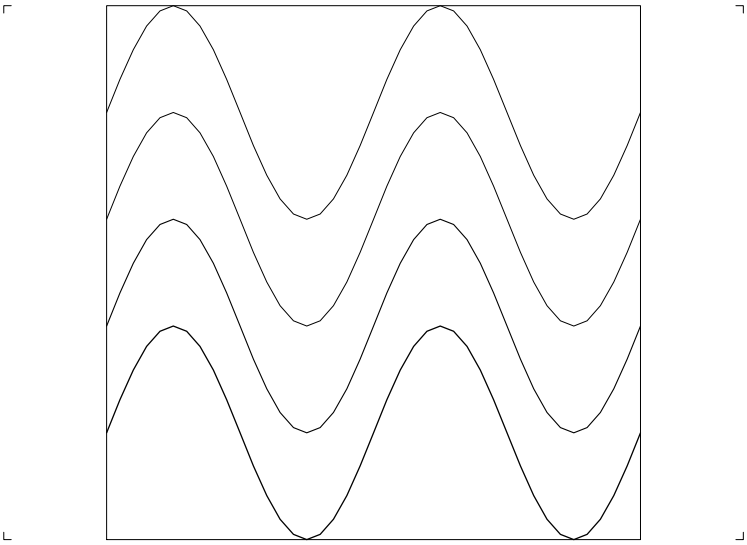
プログラム KIHON2 では, これらの属性がどのように指定できるかを示します. 正弦曲線を SGPLV ルーチンで描いていますが, 一番上の線がラインインデクスが初期値 (1) の場合です (24 行め). 次からの 3 本は, 27 行めの SGSPLI ルーチンでこの値を 4, 6, 8 と変えて描いた結果です. ラインインデクスが大きくなるにつれて, 線が太くなります.

34 行めで SGFRM ルーチンを呼んで次のフレームを設定し, 今度はラインタイプを変えて描いています. ラインタイプとは, 実線, 破線などの線種ですが, 1 から 4 までの番号にはあらかじめ次のタイプが決められています. 1: 実線, 2: 破線, 3: 点線, 4: 1 点鎖線. 初期値は 1 で, 単に SGPLV ルーチンを呼ぶと実線で折れ線を描きます. そういえば, frame1 の結果も実線でした. 40 行めの SGSPLT ルーチンでこの値を 2, 3, 4 と変えて描いた結果が frame2 です.

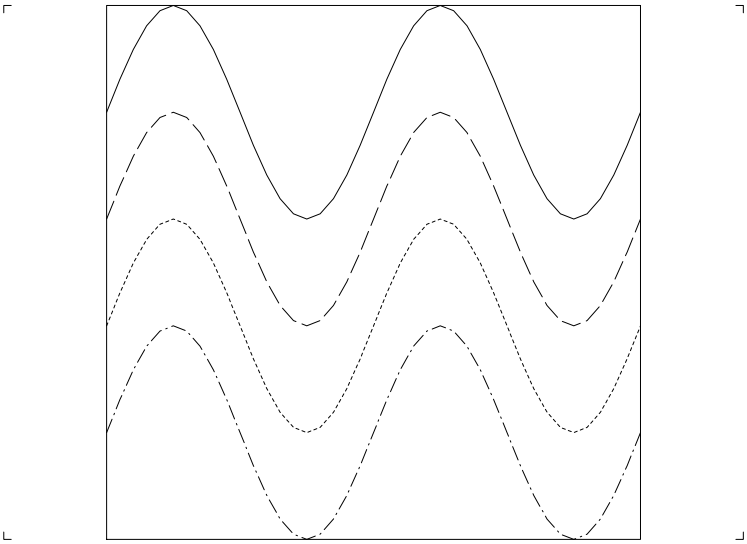
ラインタイプには, まだまだいろんな種類があり, ユーザーがそれぞれの好みの線種を指定することが可能です. それは第 4.2節で詳しく説明することにしましょう.

```
1         PROGRAM KIHON2
2
3         PARAMETER( NMAX=40, IMAX=4 )
4         REAL X(0:NMAX), Y(0:NMAX,IMAX)
5
6         DT = 4.* 3.14159 / NMAX
7         DO 20 N=0,NMAX
8           X(N) = REAL(N)/REAL(NMAX)
9           DO 10 I=1,IMAX
10            Y(N,I) = 0.2*SIN(N*DT) + 1. - 0.2*I
11      10  CONTINUE
12      20 CONTINUE
13
14        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
15        CALL SGPWSN
16        READ (*,*) IWS
17
18        CALL SGOPN( IWS )
19
20    *-- ラインインデクス: frame 1 --
21        CALL SGFRM
22        CALL SLPVPR( 1 )
23
24        CALL SGPLV( NMAX+1, X, Y(0,1) )
25
26        DO 30 I=2,IMAX
27          CALL SGSPLI( 2*I )
28          CALL SGPLV( NMAX+1, X, Y(0,I) )
29      30 CONTINUE
30
31        CALL SGSPLI( 1 )
32
33    *-- ラインタイプ: frame 2 --
34        CALL SGFRM
35        CALL SLPVPR( 1 )
```

```
36
37          CALL SGPLV( NMAX+1, X, Y(0,1) )
38
39          DO 40 I=2,IMAX
40            CALL SGSPLT( I )
41            CALL SGPLV( NMAX+1, X, Y(0,I) )
42       40 CONTINUE
43
44          CALL SGCLS
45
46          END
```



kihon2.f: frame1



kihon2.f: frame2

3.3　ポリマーカープリミティブ

マーカー列を描くポリマーカープリミティブには, 次の 3 つの属性があります.

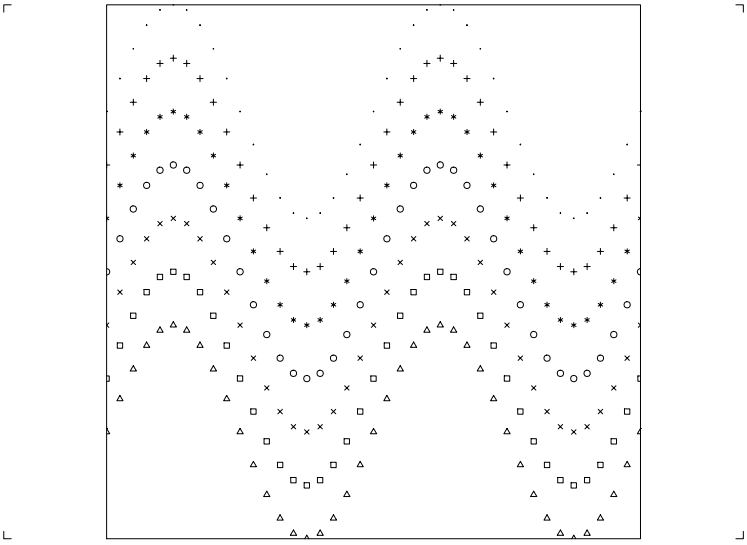
- マーカータイプ (種類)
- マーカーサイズ (大きさ)

- マーカーを描く線のラインインデクス

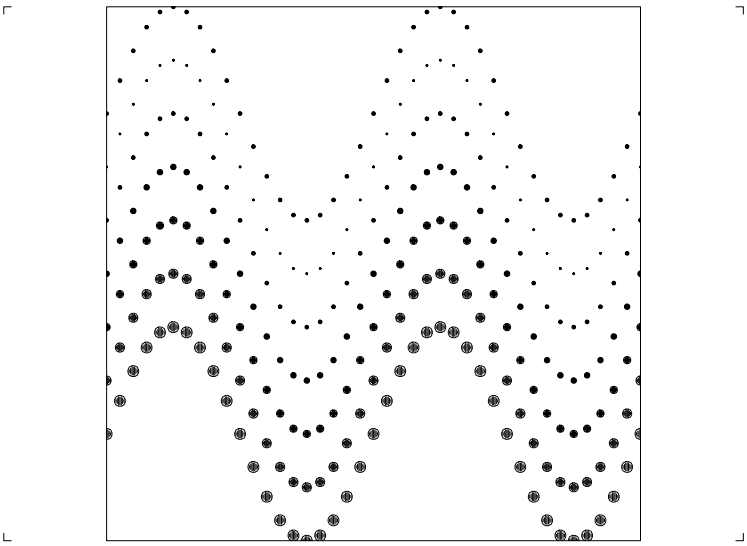
プログラム KIHON3 は, マーカータイプとマーカーサイズの属性を変えて描く例です.

まず最初のフレームですが, 第 3.1節で見たように, 単に SGPMV ルーチンを呼ぶと初期値である ・のマーカー列が描かれます. 28 行めの SGSPMT ルーチンでマーカータイプを 2 から 7 まで変えると, $+$, $*$, $\circ$ と順に異なるマーカー列が上から下へと描かれていきます. 巻末付録のフォントテーブルにあるマーク ・記号 ・文字などには, それぞれ DCL 文字番号が与えられていますが, ここでは SGSPMT で与える DCL 文字番号に対応するマーカー列が描かれているのです. たとえば ITYPE=152 のとき, α のマーカー列を描きます.

次のフレームではマーカーの大きさを変えています. 初期値は V-座標系における単位で 0.01 です. 2 番めからは, SGSPMS ルーチンで大きさを $0.005, 0.01, 0.015, \cdots, 0.03$ と変えた結果です. 塗りつぶしたようなマーカーもどんどん大きくしていくと, いくつかの線分で構成されていることが見えてきます. このような時には, SGSPMI ルーチンで描くマーカーのラインインデクスを大きくしておくと, 線が太くなって塗りつぶした雰囲気が出てきます.



kihon3.f: frame1



kihon3.f: frame2

```
1         PROGRAM KIHON3
2
3         PARAMETER( NMAX=40, IMAX=7 )
4         REAL X(0:NMAX), Y(0:NMAX,IMAX)
5
6         DT = 4.* 3.14159 / NMAX
7         DO 20 N=0,NMAX
8           X(N) = REAL(N)/REAL(NMAX)
9           DO 10 I=1,IMAX
10            Y(N,I) = 0.2*SIN(N*DT) + 0.9 - 0.1*I
11      10  CONTINUE
12      20 CONTINUE
13
14        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
15        CALL SGPWSN
16        READ (*,*) IWS
17
18        CALL SGOPN( IWS )
19
20    *-- マーカータイプ: frame 1 --
21        CALL SGFRM
22        CALL SLPVPR( 1 )
23
24        CALL SGPMV( NMAX+1, X, Y(0,1) )
25
26        DO 30 I=2,IMAX
27          ITYPE = I
28          CALL SGSPMT( ITYPE )
29          CALL SGPMV( NMAX+1, X, Y(0,I) )
30      30 CONTINUE
31
32    *-- マーカーサイズ: frame 2 --
33        CALL SGFRM
34        CALL SLPVPR( 1 )
35
36        CALL SGSPMT( 10 )
37        CALL SGPMV( NMAX+1, X, Y(0,1) )
38
39        DO 40 I=2,IMAX
40          RSIZE = 0.005*(I-1)
41          CALL SGSPMS( RSIZE )
42          CALL SGPMV( NMAX+1, X, Y(0,I) )
43      40 CONTINUE
44
45        CALL SGCLS
46
47        END
```

3.4 テキストプリミティブ

文字列を描くテキストプリミティブには, 次のような属性があります.

- 文字を描く線のラインインデクス
- 文字の大きさ
- 文字列のセンタリングオプション
- 文字列の回転角

プログラム KIHON4 で, これらの属性を 1 つずつ順に変えて文字列を書いてみましょう.

まず, SGTXV だけを呼んで, デフォルトで 'SGTXV' と書いてみました. ラインインデクスの初期値は 1, 文字の大きさの初期値は V-座標系における単位で 0.05 で, 指定した座標に文字列の中心が来ます.

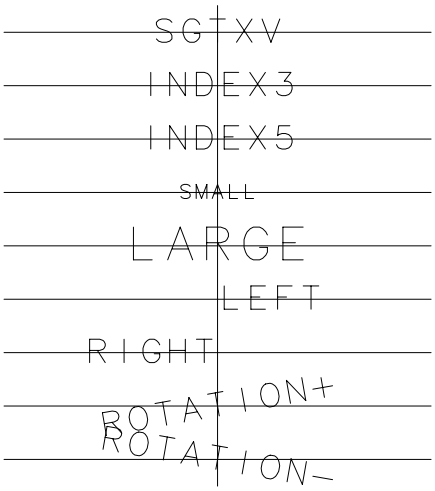
次に, SGSTXI ルーチンで文字を描く線のラインインデクスを変えて, 'INDEX3', 'INDEX5' と書いてみました. 大きな値にすると太字になります. このラインインデクスはポリラインなどのラインインデクスとは独立に管理されており, 他のラインインデクスに影響を与えたりしません. マーカーのラインインデクスも同様です.

また, 文字の大きさは SGSTXS ルーチンで変えることができます ('SMALL' と 'LARGE').

文字列の位置ですが, 文字列の左端の座標や, 右端の座標で指定することもできます. SGSTXC で -1 を指定すると左あわせ ('LEFT'), 1 で右あわせ ('RIGHT') となります. 初期値は 0 で, 中央あわせとなっています. この引数は整数に限られることに注意して下さい.

文字列を回転させるためには, SGSTXR ルーチンで回転角を指定します. 単位は度で, 指定した位置を中心に反時計回りに回転します. やはり, このルーチンでも引数は整数値に限られます。

```
1         PROGRAM KIHON4
2
3         PARAMETER( NMAX=9, X1=0.1, XC=0.5, X2=0.9 )
4         REAL Y(NMAX)
5
6         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
7         CALL SGPWSN
8         READ (*,*) IWS
9
10        CALL SGOPN( IWS )
11        CALL SGFRM
12
13    *-- 罫線 ----
14        DO 10 N=1,NMAX
15          Y(N) = 0.1*(10-N)
16          CALL SGLNV( X1, Y(N), X2, Y(N) )
17      10  CONTINUE
18
19        CALL SGLNV( XC, 0.05, XC, 0.95 )
20
21    *-- デフォルト ----
22        CALL SGTXV( XC, Y(1), 'SGTXV' )
23
24    *-- 文字のラインインデクス ----
25        CALL SGSTXI( 3 )
26        CALL SGTXV( XC, Y(2), 'INDEX3' )
27
28        CALL SGSTXI( 5 )
29        CALL SGTXV( XC, Y(3), 'INDEX5' )
30
31        CALL SGSTXI( 1 )
32
33    *-- 文字の大きさ ----
34        CALL SGSTXS( 0.03 )
35        CALL SGTXV( XC, Y(4), 'SMALL' )
36
37        CALL SGSTXS( 0.07 )
38        CALL SGTXV( XC, Y(5), 'LARGE' )
39
40        CALL SGSTXS( 0.05 )
41
42    *-- 文字列のセンタリングオプション ----
43        CALL SGSTXC( -1 )
44        CALL SGTXV( XC, Y(6), 'LEFT' )
45
46        CALL SGSTXC( 1 )
47        CALL SGTXV( XC, Y(7), 'RIGHT' )
48
49        CALL SGSTXC( 0 )
50
51    *-- 文字列の回転角 ----
52        CALL SGSTXR( 10 )
53        CALL SGTXV( XC, Y(8), 'ROTATION+' )
54
55        CALL SGSTXR( -10 )
56        CALL SGTXV( XC, Y(9), 'ROTATION-' )
57
58        CALL SGCLS
59
60        END
```



kihon4.f: frame1

3.5 トーンプリミティブ

これまでの出力プリミティブはすべて線分で構成されていましたが, トーンプリミティブは出力装置の能力に応じて, ハードフィルまたはソフトフィルを切替えて多角形のぬりつぶしを行ないます. ハードフィルでは実際にデバイスに依存した 〝領域のぬりつぶし」を行ないますが, ソフトフィルではそこにドットや線分を描くことによって 〝ぬりつぶし」を実現します.

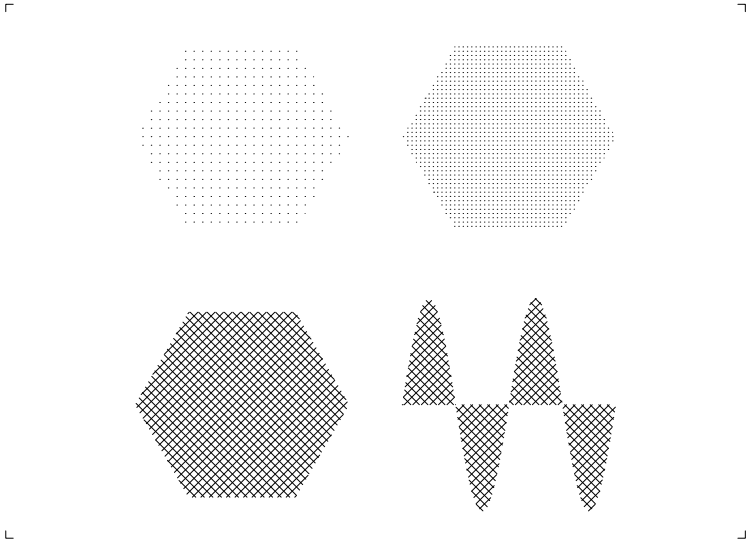
トーンプリミティブには, トーンパターンという属性があります. トーンパターンには色とパターンの 2 つの情報が含まれていますが, カラーグラフィクスについては第 12章で説明しますので, ここではドットや横線, 斜線などのパターンだけが変えられる出力装置を念頭に置いて説明しましょう.

まず, パターンの種類ですが, 3 けたのパターン番号で指定します. 具体的には巻末付録のトーンパターンテーブルを参照して下さい. パターン番号の 1 桁め (0~6) は, 次のパターンの種類をあらわします. 0: ドット, 1: 横線, 2: 斜線 (右上がり), 3: 縦線, 4: 斜線 (左上がり), 5: 格子 (縦横), 6: 格子 (斜め). 2 桁めは, ドットの大きさや斜線の太さで, 0 から 5 まで値が大きくなるにつれてドットは大きくなり斜線は太くなります. 3 桁めはドットや斜線の密度で, 0 から 5 まで値が大きくなるにつれて密度が高くなります. ただし 3 桁めが 0 のときは何も描かれません. パターン番号として 999 を指定するとべたぬりとなります. これらの組合せ以外の整数値はパターンが定義されていません.

さて, KIHON5 のプログラム例ですが, 27 行めで SGLSET ルーチンを用いてトーンプリミティブに関する論理型内部変数 'LSOFTF' を.TRUE. としてソフトフィルを選んでいます. そして SGTNV ルーチンで多角形の内部をぬりつぶします. 引数の与え方は SGPLV と似ていますが, SGTNV では多角形を定義する頂点の座標を与えます. 配列の最初と最後 (ポリラインでは始点と終点) が一致する必要はありません. この例の左上では 6 角形をトーンパターンの初期値 001(小さく疎なドット) でぬりつぶしました. 右上では, SGSTNP ルーチンでトーンパターン番号を 3 に指定してぬりつぶしています. 003 ですから, 同じ大きさのドットでより密なパターンです. 左下は 601 で斜め格子の例です.

SGTNV で指定する多角形は単連結領域である必要はありません. 右下のように多角形の辺同志が交差する 〝ねじれた多角形」でも, その図形によってできる閉領域を全てぬりつぶします.

```
1         PROGRAM KIHON5
2
3         PARAMETER( NMAX=40, IMAX=4 )
4         REAL X(0:NMAX,IMAX), Y(0:NMAX,IMAX), XC(IMAX), YC(IMAX)
5
6         DATA XC/0.25, 0.75, 0.25, 0.75/
7         DATA YC/0.75, 0.75, 0.25, 0.25/
8
9         DT = 2.* 3.14159 / 6
10        DO 10 N=0,5
11        DO 10 I=1,IMAX-1
12          X(N,I) = 0.2*COS(N*DT) + XC(I)
13          Y(N,I) = 0.2*SIN(N*DT) + YC(I)
14    10  CONTINUE
15
16        DT = 4.* 3.14159 / NMAX
17        DO 20 N=0,NMAX
18          X(N,IMAX) = 0.4*REAL(N)/REAL(NMAX) - 0.2 + XC(IMAX)
19          Y(N,IMAX) = 0.2*SIN(N*DT)                + YC(IMAX)
20    20  CONTINUE
21
22        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
23        CALL SGPWSN
24        READ (*,*) IWS
25
26        CALL SGOPN( IWS )
27        CALL SGLSET( 'LSOFTF', .TRUE. )
28        CALL SGFRM
29
30  *-- デフォルト ----
31        CALL SGTNV( 6, X(0,1), Y(0,1) )
32
33  *-- トーンパターン ----
34        CALL SGSTNP( 3 )
35        CALL SGTNV( 6, X(0,2), Y(0,2) )
36
37        CALL SGSTNP( 601 )
38        CALL SGTNV( 6, X(0,3), Y(0,3) )
39
40  *-- ねじれた多角形 ----
41        CALL SGTNV( NMAX+1, X(0,4), Y(0,4) )
42
43        CALL SGCLS
44
45        END
```



kihon5.f: frame1

DCL のしくみ 1

SGPACK ルーチンの名付け方

SGPACK の各出力プリミティブを定義するサブルーチン名の名付け方は以下のようになっています.

1. 先頭の 2 文字はすべて SG で始まる.
2. 属性を設定/参照する場合, 3 文字目は S (set)/ Q (query) である.
3. 各プリミティブの名前がそれに続く. それぞれ,
   - PL : ポリラインプリミティブ.
   - PM : ポリマーカープリミティブ.
   - TX : テキストプリミティブ.
   - TN : トーンプリミティブ.
   - LA : アローサブプリミティブ.
   - LN : ラインサブプリミティブ.
4. プリミティブを定義 (出力) するルーチンについて, 属性を同時に指定する場合は Z が続く.
5. さらにプリミティブをどこに定義するかによって, U (U-座標系で定義する), V (V-座標系で定義する) または R (R-座標系で定義する) が続く.
6. プリミティブの属性を設定/参照する場合, 6 文字目はその属性を示す.
   - T : ラインタイプ, マーカータイプ.
   - I : ラインインデクス.
   - S : マーカーサイズ, 文字の高さ.
   - R : 文字列の回転角.
   - C : 文字列のセンタリングオプション.
   - P : トーンパターン.

第 4 章

描画の基本 (2)

これまでは $[0,1]\times[0,1]$ の仮想直角座標系 (V-座標系) だけを用いて各出力プリミティブを説明してきました. しかし, 一般に扱っているデータは $[0,1]$ で正規化されているわけではありませんから, つねに V-座標系で作図するのは能率的なことではありません. そこで, ユーザーが実際に使っている単位の座標系, ユーザー座標系 (U-座標系) を導入し, U-座標系から V-座標系への変換 (正規化変換) を定義して, U-座標系で作図することを考えましょう.

また, 出力プリミティブのちょっと進んだ内容についても紹介しましょう.

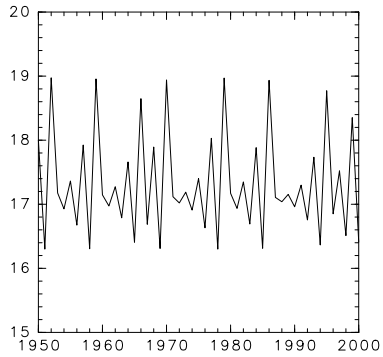
4.1　ユーザー座標系 (U-座標系) での基本描画

4.1.1　基本概念

ユーザーが実際のデータをもとに作図する過程を KIHON6 のプログラム例をもとに考えてみましょう. ある地点の年平均気温のデータが 1950 年から 2000 年まであり, およそセ氏 16 度から 19 度の範囲内で変動していた (第 2章のロジスティク模型の結果をちょっと細工しただけですが) としましょう. このデータを元に平均気温の年々変動の折れ線グラフを描こうと思います. x 軸を時間軸にとり 1950 年から 2000 年まで, y軸には 15 度から 20 度を範囲として, このデータを折れ線で描くことにします.

```
1         PROGRAM KIHON6
2
3         PARAMETER( NMAX=50 )
4         REAL      X(0:NMAX), Y(0:NMAX)
5
6         R    = 3.7
7         X(0) = 1950
8         Y(0) = 0.5
9         DO 10 N=0,NMAX-1
10          X(N+1) = X(N) + 1
11          Y(N+1) = R*Y(N)*(1.-Y(N))
12      10 CONTINUE
13
14        DO 20 N=0,NMAX
15          Y(N) = -4*Y(N) + 20.
16      20 CONTINUE
17
18        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
19        CALL SGPWSN
20        READ (*,*) IWS
21
22        CALL GROPN( IWS )
23        CALL GRFRM
24
25        CALL GRSWND( 1950., 2000., 15., 20. )
26        CALL GRSVPT(   0.2,   0.8, 0.2, 0.8 )
27        CALL GRSTRN( 1 )
```

```
28          CALL GRSTRF
29
30          CALL USDAXS
31          CALL SGPLU( NMAX+1, X, Y )
32
33          CALL GRCLS
34
35          END
```



kihon6.f: frame1

ユーザーの使っている座標系 (ここでは時間 [年] と温度 [度]) を, ユーザー座標系 (U-座標系) と呼びます. U-座標系でグラフに描きたい範囲を 「ウインドウ」と呼び, それぞれの軸の最小値と最大値で設定します. この例では, (UXMIN, UXMAX, UYMIN, UYMAX) = (1950, 2000, 15, 20) で, 25 行めの GRSWND ルーチンで設定しています. このプログラムでは 30 行めで GRPH2 の USDAXS ルーチンを用いて座標軸を描くために GRxxxx を使っていることに注意して下さい.

次に, このウインドウを V-座標系のどの範囲に対応させるかを考えて, これを 「ビューポート」と呼びます. ビューポートとは, V-座標系で通常座標軸が描かれる矩形の領域のことです. ここでは, GRSVPT ルーチンを用いて (VXMIN, VXMAX, VYMIN, VYMAX) = (0.2, 0.8, 0.2, 0.8) と設定しました.

これで, ウインドウとビューポートの四隅は対応させることができましたが, ウインドウ内の各点をビューポート内の点に対応させる必要があります. これを 「正規化変換」と呼びます. 線形に対応させるか, 対数をとって対応させるかなどの任意性がありますから, 具体的に変換関数を決めなければなりません. SGPACK(GRPACK) ではいくつかの変換関数が用意されており, それぞれに変換関数番号が付けられています. ここでは, GRSTRN ルーチンで関数番号 1 を指定しています. これは両軸ともに線形に対応させるもので, 直角一様座標となります.

このように設定されたパラメータの値は, 変換関数を確定するルーチン GRSTRF を呼ぶことで有効になります. U-座標系で描画するためには, GRFRM ルーチンを呼んだ後でかつ描画をはじめる前に変換関数を確定する必要があります.

さて, U-座標系での描画ですが, V-座標系での各種描画ルーチンに対応するものが, すべて用意されています. この例では, U-座標系でのポリラインプリミティブ SGPLU で折れ線を描いています. 引数の与え方は全く同じで, データ X, Y が U-座標系の単位で用意されています. V-座標系での描画ルーチンのサブルーチン名が V で終っていたのに対して, U-座標系での描画ルーチンは U で終るところが違っているだけで, 全

く同じ機能になっています.

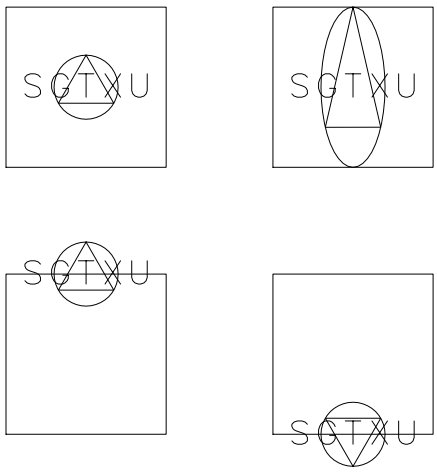
4.1.2　ウインドウとビューポート

次のプログラム KIHON7 では, U-座標系でのポリラインやテキストプリミティブの出力結果が, ウインドウとビューポートの設定の仕方によってどのようになるかを示しましょう.

```
      PROGRAM KIHON7

      WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
      CALL SGPWSN
      READ (*,*) IWS

      CALL SGOPN( IWS )
C      CALL SGLSET( 'LCLIP', .TRUE. )
*-- frame 1 ----
      CALL SGFRM
*                  XMIN,  XMAX,  YMIN, YMAX
      CALL SGSWND( -100., 100., -100., 100. )
      CALL SGSVPT(  0.0,   1.0,   0.0,  1.0 )
      CALL SGSTRN( 1 )
      CALL SGSTRF

      CALL APLOT

*-- frame 2 ----
      CALL SGFRM
*-- 左上 ----      XMIN,  XMAX,  YMIN, YMAX
      CALL SGSWND( -100., 100., -100., 100. )
      CALL SGSVPT(   0.1,  0.4,   0.6,  0.9 )
      CALL SGSTRN( 1 )
      CALL SGSTRF

      CALL APLOT

*-- 右上 ----       XMIN, XMAX, YMIN, YMAX
      CALL SGSWND( -100., 100., -40.,  40. )
      CALL SGSVPT(   0.6,  0.9,  0.6,  0.9 )
      CALL SGSTRN( 1 )
      CALL SGSTRF

      CALL APLOT

*-- 左下 ----       XMIN, XMAX,  YMIN, YMAX
      CALL SGSWND( -100., 100., -200.,   0. )
      CALL SGSVPT(   0.1,  0.4,   0.1,  0.4 )
      CALL SGSTRN( 1 )
      CALL SGSTRF

      CALL APLOT


*-- 右下 ----       XMIN, XMAX, YMIN,  YMAX
      CALL SGSWND( -100., 100.,   0., -200. )
      CALL SGSVPT(   0.6,  0.9,  0.1,   0.4 )
      CALL SGSTRN( 1 )
      CALL SGSTRF

      CALL APLOT

      CALL SGCLS

      END
*--------------------------------------------------------------------
      SUBROUTINE APLOT

      PARAMETER( NMAX=40 )
      REAL X(0:NMAX), Y(0:NMAX)

      CALL SLPVPR( 1 )

*-- 円形 ----
      DT = 2.*3.14159 / NMAX
      DO 10 N=0,NMAX
        X(N) = 40.*SIN(N*DT)
        Y(N) = 40.*COS(N*DT)
   10 CONTINUE

```

```
72          CALL SGPLU( NMAX+1, X, Y )
73
74    *-- 三角形 ----
75          DT = 2.*3.14159 / 3
76          DO 20 N=0,3
77            X(N) = 40.*SIN(N*DT)
78            Y(N) = 40.*COS(N*DT)
79       20 CONTINUE
80
81          CALL SGPLU( 4, X, Y )
82
83    *-- 文字列 ----
84          CALL SGTXU( 0., 0., 'SGTXU' )
85
86          RETURN
87          END
```

kihon7.f: frame1



kihon7.f: frame2

1 フレームめが基本形で, まず SLPVPR ルーチンでビューポートの枠を描き, U-座標系描画ルーチンの SGPLU で円と正三角形を描き, SGTXU で文字列 'SGTXU' を描いています (サブルーチン APLOT).

2 フレームめの左上には, 同じ APLOT サブルーチンでビューポートだけを小さくして描いてみました. ビューポートが小さくなったのに対応して SGPLU で描いた円と三角形は小さくなりましたが, 文字の大きさは 1

フレームめと変わりません. U-座標系でのテキストプリミティブでも, 文字の大きさは V-座標系の単位で指定するので, ビューポートの大きさが変わっても文字の大きさは変わらないのです.

次に, ウインドウの y軸の範囲を変えてみました (右上). 範囲を 40%に狭めたので, 図形は縦に引き延ばされて, 円が楕円になりました. この場合にも, 文字は変わりません.

左下の例では, ウインドウの縦横比を元に戻して, UYMIN=-200, UYMAX=0 と y軸負方向に平行移動させました. その結果, 設定したビューポートをはみ出して図形を描いています.

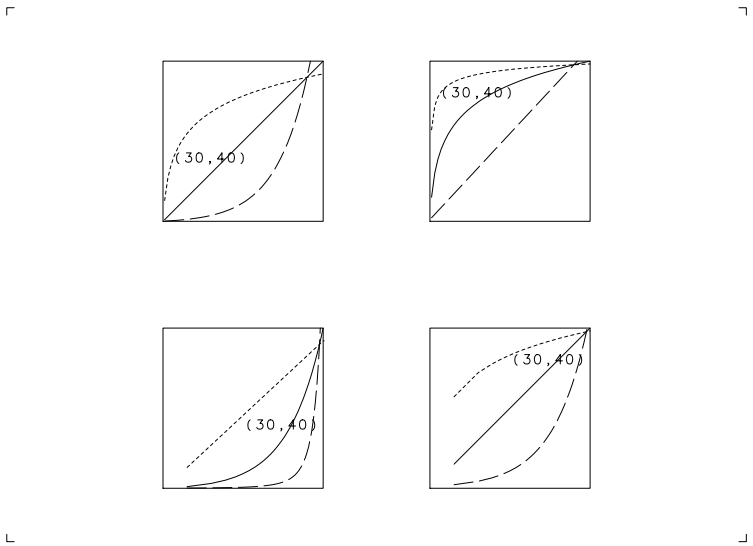
最後に, 右下の例はウインドウの y軸を逆転させた場合です. UYMIN=0, UYMAX=-200 として, 左下の場合と逆にしました. この時の三角形を見れば明らかなように, 図形は上下に折り返されています. ただし, 文字列は正立のままで折り返されません.

下の 2 つの例のように, 設定したビューポートをはみ出して図形を描く場合, はみ出した部分を描かないようにすることも出来ます. SGLSET ルーチンを用いてクリッピングに関する内部変数 'LCLIP' を.TRUE. にしてやると, ビューポートをはみ出した部分はラインでも文字でも描かれません. 8 行めのコメント行をもとに戻せばクリッピングを行ないます.

4.1.3　正規化変換の変換関数

変換関数を変えることによってさまざまな座標系での描画が可能です. SGPACK で扱える座標系には, 大別して, 直交直線座標系 (1~4), 直交曲線座標系 (5~7), 地図投影座標系 (10 番台~30 番台), ユーザー定義座標系 (99), の 4 種類があります. 括弧内の数字は変換関数番号です. KIHON8 のプログラム例では, よく使う直交直線座標系の 4 つを見ておきましょう.

サブルーチン BPLOT では, ビューポートの枠を描き, 一次関数を実線で, 指数関数を破線で, 対数関数を点線で描き, (X=30,Y=40) を中心に SGTXU で文字列 '(30,40)' を描きます. SGSTRN ルーチンで変換関数番号を 1 と指定すると, 直角一様座標 (左上) となります. 2 ならば y 軸が対数の片対数座標 (右上), 3 ならば x 軸が対数の片対数座標 (左下), 4 ならば両対数座標 (右下) となります.



kihon8.f: frame1

```
      PROGRAM KIHON8

      PARAMETER( XMIN=1., XMAX=100., YMIN=1., YMAX=100. )

      WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
      CALL SGPWSN
      READ (*,*) IWS

      CALL SGOPN( IWS )
      CALL SGLSET( 'LCLIP', .TRUE. )

      CALL SGFRM
*-- 直角一様座標 (左上) ----
      CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
      CALL SGSVPT(  0.1,  0.4,  0.6,  0.9 )
      CALL SGSTRN( 1 )
      CALL SGSTRF

      CALL BPLOT

*-- 片対数 (y) 座標 (右上) ----
      CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
      CALL SGSVPT(  0.6,  0.9,  0.6,  0.9 )
      CALL SGSTRN( 2 )
      CALL SGSTRF

      CALL BPLOT

*-- 片対数 (x) 座標 (左下) ----
      CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
      CALL SGSVPT(  0.1,  0.4,  0.1,  0.4 )
      CALL SGSTRN( 3 )
      CALL SGSTRF

      CALL BPLOT

*-- 対数座標 (右下) ----
      CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
      CALL SGSVPT(  0.6,  0.9,  0.1,  0.4 )
      CALL SGSTRN( 4 )
      CALL SGSTRF

      CALL BPLOT

      CALL SGCLS

      END
*---------------------------------------------------------------------
      SUBROUTINE BPLOT

      PARAMETER( NMAX=50 )
      REAL X(NMAX), Y(NMAX)

      CALL SLPVPR( 1 )

*-- 一次関数 ----
      DO 10 N=1,NMAX
        X(N) = 2.*N
        Y(N) = X(N)
   10 CONTINUE

      CALL SGSPLT( 1 )
      CALL SGPLU( NMAX, X, Y )

*-- 指数関数 ----
      DO 20 N=1,NMAX
        Y(N) = EXP(0.05*X(N))
   20 CONTINUE

      CALL SGSPLT( 2 )
      CALL SGPLU( NMAX, X, Y )

*-- 対数関数 ----
      DO 30 N=1,NMAX
        Y(N) = 20.*LOG(X(N))
   30 CONTINUE

      CALL SGSPLT( 3 )
      CALL SGPLU( NMAX, X, Y )

*-- 文字列 ----
      CALL SGSTXS( 0.02 )
      CALL SGTXU( 30., 40., '(30,40)' )
```

```
84
85          RETURN
86          END
```

4.2 もっとポリライン

U-座標系で折れ線を描くルーチンは SGPLU です. これを使ってポリラインプリミティブをもっと深く探ってみましょう (KIHON9, KIHONA). 属性の 1 つがラインタイプですが, ユーザーの指定により, 多種多様なラインタイプをつくることができます.

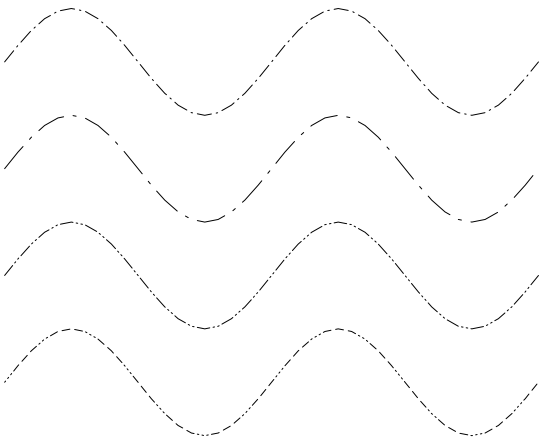
まず, KIHON9 のプログラムですが, 一番上にラインタイプが 4 の折れ線 (一点鎖線) を描いています. SGRSET ルーチンでポリラインプリミティブに関する実数型内部変数の 'BITLEN' を変更すると, パターンの 1 サイクルの長さを変えることができます. この初期値は V-座標系の単位で 0.003 ですが, 倍の 0.006 に設定し直したのが 2 本めの折れ線です. ビューポートのとり方次第で, パターンが間延びしたり, 逆にパターンが潰れて判別できなくなったりすることがありますが, そのようなときには 'BITLEN' を調節しましょう.

SGPACK では, 実線, 破線, 点線, および一点鎖線以外のパターンのラインタイプも指定できます. 実は, SGSPLT ルーチンの引数に 5 以上の整数を指定すると, その 2 進表現のパターンを指定したものと見なされます. すなわち, 整数の下位 16 bit のうち 1 の部分に線を描き, 0 の部分は空白とするようなパターンを設定します. 自分の描きたいパターンに対応する整数を求めるには, MISC1 の BITLIB パッケージ中の BITPCI ルーチン (47 行め) を使うと便利です. '1111111100100100' のように文字型でビットパターンを与えると, それに対応する整数値 ITYPE が返されて, そのラインタイプで描いた二点鎖線が 3 本めです.

さらに複雑なパターンを指定したいときには, SGISET で内部変数 'NBITS' を変更することにより, パターンのビット長を 32 bit まで長くすることができます. 4 本めの折れ線はこれを 32 bit にして, 3 本めと同様に BITPCI ルーチンで新しいパターンを作っています.

```
1           PROGRAM KIHON9
2
3           PARAMETER( NMAX=40 )
4           PARAMETER( PI=3.14159 )
5           PARAMETER( XMIN=0., XMAX=4*PI, YMIN=-1., YMAX=1. )
6           REAL X(0:NMAX), Y(0:NMAX)
7
8           DT = XMAX/NMAX
9           DO 10 N=0,NMAX
10            X(N) = N*DT
11            Y(N) = SIN(X(N))
12        10 CONTINUE
13
14          WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
15          CALL SGPWSN
16          READ (*,*) IWS
17
18          CALL SGOPN( IWS )
19          CALL SGFRM
20
21    *--   ラインタイプ = 4 (デフォルト) ----
22          CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
23          CALL SGSVPT(   0.,   1.,  0.7,  0.9 )
24          CALL SGSTRN( 1 )
25          CALL SGSTRF
26
27          CALL SGSPLT( 4 )
28          CALL SGPLU( NMAX+1, X, Y )
29
30    *--   ラインタイプ = 4 (BITLEN*2) ----
31          CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
32          CALL SGSVPT(   0.,   1.,  0.5,  0.7 )
33          CALL SGSTRN( 1 )
34          CALL SGSTRF
```

```
35
36          CALL SGRSET( 'BITLEN', 0.006 )
37          CALL SGPLU( NMAX+1, X, Y )
38
39          CALL SGRSET( 'BITLEN', 0.003 )
40
41    *--   ビットパターン ----
42          CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
43          CALL SGSVPT(   0.,   1.,  0.3,  0.5 )
44          CALL SGSTRN( 1 )
45          CALL SGSTRF
46
47          CALL BITPCI( '1111111100100100', ITYPE )
48          CALL SGSPLT( ITYPE )
49          CALL SGPLU( NMAX+1, X, Y )
50
51    *--   ビットパターン (倍長) ----
52          CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
53          CALL SGSVPT(   0.,   1.,  0.1,  0.3 )
54          CALL SGSTRN( 1 )
55          CALL SGSTRF
56
57          CALL SGISET( 'NBITS', 32 )
58          CALL BITPCI( '10010010011111000111110001111100', ITYPE )
59          CALL SGSPLT( ITYPE )
60          CALL SGPLU( NMAX+1, X, Y )
61
62          CALL SGCLS
63
64          END
```



kihon9.f: frame1

次のプログラム KIHONA は, ラベルつきの折れ線を描く例です.

SGLSET ルーチンで内部変数 'LCHAR' を .TRUE. にすると , ポリラインプリミティブはラベルつき折れ線を描きます. ラベル付き折れ線とは, 描くべき線分のある長さを 1 サイクルとして, その一部分に空白域をとり, そこに指定した文字列を描くものです. 描く文字列は SGSPLC ルーチンで指定します. この例では, まず 'A' のラベルをつけて折れ線を描きました.

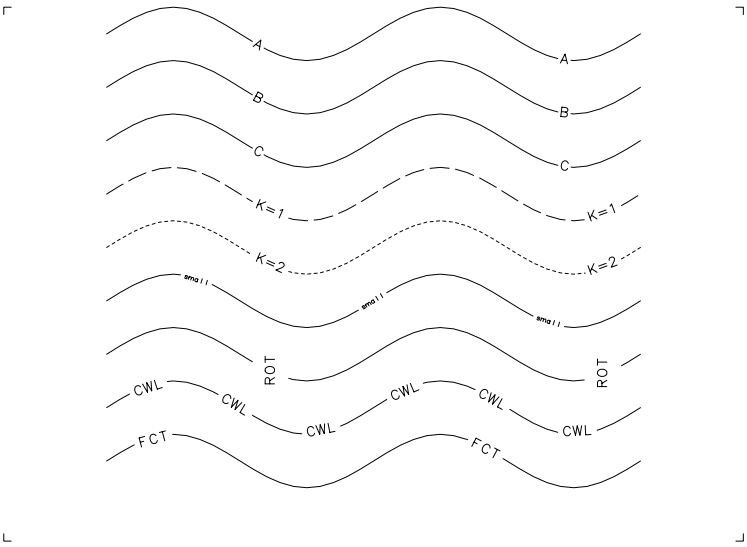
次に, 42 行めで SGNPLC ルーチンを呼ぶと, 設定されている文字列の最後の文字の文字番号が 1 つ増えます. そこで, 2 本め, 3 本めの折れ線のラベルが 'B', 'C' と変わります. また, 文字列として 'K=1' と指定すると, まず 'K=1' というラベルを描き, 次の呼び出しでは'K=2' というラベルを描きます (4 本めと 5 本め). これらの折れ線では, SGSPLT ルーチンで線分のラインタイプも変えています.

ラベルの文字列の高さは, SGSPLS ルーチンで指定できます. 初期値は V-座標系の単位で 0.02 ですが, 6 本

めの例ではこれを 0.01 として小さめのラベルにしています. さらに, ラベル付折れ線に関するパラメータを陽に設定すると, さまざまな変形が可能です. 内部変数 'LROT' を .TRUE. にして, 'IROT' で回転角を整数値で指定すると, 一定の回転角でラベルを付けます. これが .FALSE. の場合 (初期値) には, 線分に沿ってラベルを描きます. また, ラベルの間隔は内部変数 'CWL' で, ラベルの書き始めは内部変数 'FFCT' で, それぞれ調節できます.

```
      PROGRAM KIHONA

      PARAMETER( NMAX=40 )
      PARAMETER( PI=3.14159 )
      PARAMETER( XMIN=0., XMAX=4*PI, YMIN=-1., YMAX=1. )
      REAL X(0:NMAX), Y(0:NMAX)

      DT = XMAX/NMAX
      DO 10 N=0,NMAX
        X(N) = N*DT
        Y(N) = SIN(X(N))
   10 CONTINUE

      WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
      CALL SGPWSN
      READ (*,*) IWS

      CALL SGOPN( IWS )
      CALL SGFRM

      CALL SGLSET( 'LCHAR', .TRUE. )

*-- ラベルつき折れ線 ----
      CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
      CALL SGSVPT(   0.,   1.,  0.9,  1.0 )
      CALL SGSTRN( 1 )
      CALL SGSTRF

      CALL SGSPLC( 'A' )
      CALL SGSPLT( 1 )
      CALL SGPLU( NMAX+1, X, Y )

*-- 順序ラベル: A,B,C,... ----
      DO 20 I=1,2
        VYMIN = 0.9 - 0.1*I
        VYMAX = VYMIN + 0.1
        CALL SGSWND( XMIN, XMAX,  YMIN,  YMAX )
        CALL SGSVPT(   0.,   1., VYMIN, VYMAX )
        CALL SGSTRN( 1 )
        CALL SGSTRF

        CALL SGNPLC
        CALL SGPLU( NMAX+1, X, Y )
   20 CONTINUE

*-- 順序ラベル: K=1,K=2,... ----
      CALL SGSPLC( 'K=1' )

      DO 30 I=3,4
        VYMIN = 0.9 - 0.1*I
        VYMAX = VYMIN + 0.1
        CALL SGSWND( XMIN, XMAX,  YMIN,  YMAX )
        CALL SGSVPT(   0.,   1., VYMIN, VYMAX )
        CALL SGSTRN( 1 )
        CALL SGSTRF

        CALL SGSPLT( I-1 )
        CALL SGPLU( NMAX+1, X, Y )
        CALL SGNPLC
   30 CONTINUE

        CALL SGSPLT( 1 )

*--  ラベルの文字列の高さ ----
      CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
      CALL SGSVPT(   0.,   1.,  0.4,  0.5 )
      CALL SGSTRN( 1 )
      CALL SGSTRF

      CALL SGSPLS( 0.01 )
      CALL SGSPLC( 'small' )
```

```
72          CALL SGPLU( NMAX+1, X, Y )
73
74          CALL SGSPLS( 0.02 )
75
76    *--   ラベルの角度 ----
77          CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
78          CALL SGSVPT(   0.,   1.,  0.3,  0.4 )
79          CALL SGSTRN( 1 )
80          CALL SGSTRF
81
82          CALL SGLSET( 'LROT', .TRUE. )
83          CALL SGISET( 'IROT', 90 )
84          CALL SGSPLC( 'ROT' )
85          CALL SGPLU( NMAX+1, X, Y )
86
87          CALL SGLSET( 'LROT', .FALSE. )
88
89    *--   ラベルの間隔 ----
90          CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
91          CALL SGSVPT(   0.,   1.,  0.2,  0.3 )
92          CALL SGSTRN( 1 )
93          CALL SGSTRF
94
95          CALL SGRSET( 'CWL', 5. )
96          CALL SGSPLC( 'CWL' )
97          CALL SGPLU( NMAX+1, X, Y )
98
99          CALL SGRSET( 'CWL', 30. )
100
101   *--   ラベルの書き始め ----
102         CALL SGSWND( XMIN, XMAX, YMIN, YMAX )
103         CALL SGSVPT(   0.,   1.,  0.1,  0.2 )
104         CALL SGSTRN( 1 )
105         CALL SGSTRF
106
107         CALL SGRSET( 'FFCT', 0.9 )
108         CALL SGSPLC( 'FCT' )
109         CALL SGPLU( NMAX+1, X, Y )
110
111         CALL SGCLS
112
113         END
```



kihona.f: frame1

4.3　もっとテキスト

プログラム KIHONB で, ちょっと進んだ文字出力をしてみましょう.

SGPACK においては '|', '_', '"' の 3 文字が添字の制御に使われるのですが, 何も指定しなければ, これらの文字も他の文字と同様に文字として出力されます (最初の例). SGLSET ルーチンでテキストプリミティブ

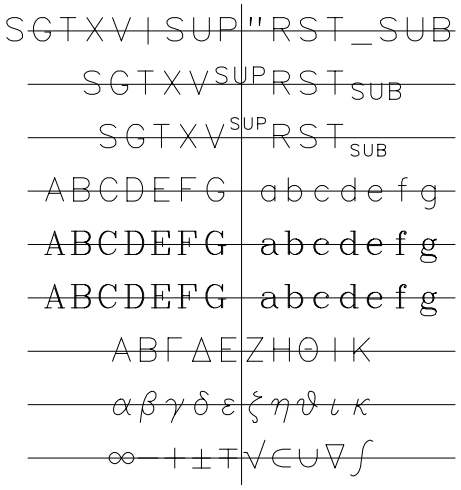
に関する内部変数 'LCNTL' を .TRUE. にすると, 2 番めの例のように, 上付や下付の添字を出力することができます. '|' と'_' は, それぞれ, 上付添え字と下付添え字のモードの始まりをしめす制御文字で, '"' は上付および下付添え字のモードの終わりをしめす制御文字です. ここで注意すべきことは, 文字列が添字で終る場合でも, 添字の後に必ず '"' を入れて通常の文字モードに戻すことです. こうしないと文字列の長さが正確に求まらないので, センタリングや右よせ等の処理がうまくいきません. また, 添字の大きさや上下の移動量は, それぞれ SGRSET ルーチンで内部変数 'SMALL', 'SHIFT' を指定することにより変更できます (3 番め).

巻末付録に示したように SGPACK は 2 種類のフォントを持っています. SGISET ルーチンで内部変数 'IFONT' を 2 とすると, フォント番号 2 の高品位フォントを出力することができます. 高品位フォントは複数の線を引いて線の太さを調節していますから, SGSTXI ルーチンで文字を描く線の太さを太くすると空白が潰れてそれらしい文字になります. また, 当然のことながら高品位フォントを使うと出力量が増えますから, 出力に時間がかかるようになります.

フォントテーブルを見ればわかるように, SGPACK は普通のアルファベットだけでなく, ギリシャ文字や特殊記号のフォントも持っています. プログラムの 50 行め以降に示したように, これらのフォント番号を DCL 文字関数 (SGPACK にある関数) の 1 つ CSGI で文字コードに変換して, それをテキストとして SGTXV ルーチン等に渡せば, 出力できます. 5 行めの CHARACTER 文で CSGI*1 を忘れないで下さい.

```
1         PROGRAM KIHONB
2
3         PARAMETER( NMAX=9 )
4         REAL Y(NMAX)
5         CHARACTER GREEK1*10, GREEK2*10, SYMBOL*10, CSGI*1
6
7         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
8         CALL SGPWSN
9         READ (*,*) IWS
10
11        CALL SGOPN(IWS)
12        CALL SGFRM
13
14        X1 = 0.1
15        X2 = 0.9
16        XC = 0.5
17
18  *-- 罫線 ----
19        DO 10 N=1,NMAX
20          Y(N) = 0.1*(10-N)
21          CALL SGLNV( X1, Y(N), X2, Y(N) )
22    10  CONTINUE
23
24        CALL SGLNV( XC, 0.05, XC, 0.95 )
25
26  *-- デフォルト ----
27        CALL SGTXV( XC, Y(1), 'SGTXV|SUP"RST_SUB"' )
28
29  *-- 添字 ----
30        CALL SGLSET( 'LCNTL', .TRUE. )
31        CALL SGTXV( XC, Y(2), 'SGTXV|SUP"RST_SUB"' )
32
33        CALL SGRSET( 'SMALL', 0.5 )
34        CALL SGRSET( 'SHIFT', 0.5 )
35        CALL SGTXV( XC, Y(3), 'SGTXV|SUP"RST_SUB"' )
36
37  *-- フォント ----
38        CALL SGTXV( XC, Y(4), 'ABCDEFG abcdefg' )
39
40        CALL SGISET( 'IFONT', 2 )
41        CALL SGTXV( XC, Y(5), 'ABCDEFG abcdefg' )
42
43        CALL SGSTXI( 3 )
44        CALL SGTXV( XC, Y(6), 'ABCDEFG abcdefg' )
45
46        CALL SGISET( 'IFONT', 1 )
47        CALL SGSTXI( 1 )
48
```

```
*-- ギリシャ文字 ----
      GREEK1 = CSGI(128)//CSGI(129)//CSGI(130)//CSGI(131)//CSGI(132)//
     +         CSGI(133)//CSGI(134)//CSGI(135)//CSGI(136)//CSGI(137)
      GREEK2 = CSGI(152)//CSGI(153)//CSGI(154)//CSGI(155)//CSGI(156)//
     +         CSGI(157)//CSGI(158)//CSGI(159)//CSGI(160)//CSGI(161)

      CALL SGTXV( XC, Y(7), GREEK1 )
      CALL SGTXV( XC, Y(8), GREEK2 )

*-- 特殊記号 ----
      SYMBOL = CSGI(189)//CSGI(190)//CSGI(191)//CSGI(192)//CSGI(193)//
     +         CSGI(210)//CSGI(211)//CSGI(212)//CSGI(217)//CSGI(218)

      CALL SGTXV( XC, Y(9), SYMBOL )

      CALL SGCLS

      END
```



kihonb.f: frame1

4.4　アローとラインのサブプリミティブ

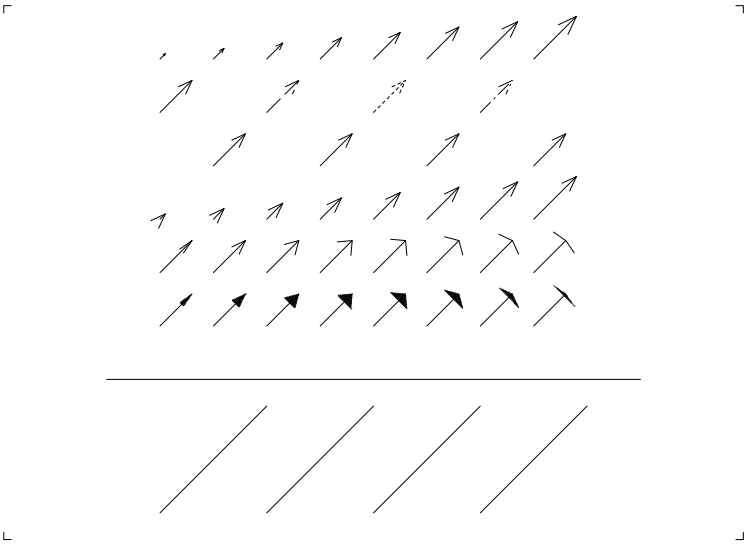
出力プリミティブには, ポリライン ・ポリマーカー ・テキスト ・トーンの他に, 補助的にアロー (矢印) とライン (線分) のサブプリミティブがあります. アローサブプリミティブには, 矢印の軸部分のラインインデクスとラインタイプの 2 つの属性があり, ラインサブプリミティブにはラインインデクスの属性があります.

プログラム KIHONC で, いろいろな矢印や線分を描いてみましょう. U-座標系での矢印描画は, SGLAU ルーチンで行ないます. 引数は, (X1,Y1) が始点の座標, (X2,Y2) が終点の座標です. 線分の終点から対称な 2 本の線分を付け加えて矢じり部分とします. 最初の例のように, デフォルトでは, 本体部分が長くなるのに比例して矢じり部分も大きくなります.

```
      PROGRAM KIHONC

      WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
      CALL SGPWSN
      READ (*,*) IWS

      CALL SGOPN( IWS )
      CALL SGFRM

      CALL SGSWND( 0.0, 10.0, 0.0, 10.0 )
      CALL SGSVPT( 0.0,  1.0, 0.0,  1.0 )
      CALL SGSTRN( 1 )
      CALL SGSTRF
```

```
14
15    *-- デフォルト ----
16          Y1 = 9.0
17          DO 10 I=1,8
18            X1 = I
19            X2 = X1 + 0.1*I
20            Y2 = Y1 + 0.1*I
21            CALL SGLAU( X1, Y1, X2, Y2 )
22       10 CONTINUE
23
24    *-- 線分のラインタイプ ----
25          Y1 = 8.0
26          Y2 = 8.6
27          DO 20 I=1,4
28            X1 = 2*I - 1
29            X2 = X1 + 0.6
30            CALL SGSLAT( I )
31            CALL SGLAU( X1, Y1, X2, Y2 )
32       20 CONTINUE
33
34          CALL SGSLAT( 1 )
35
36    *-- 線分のラインインデクス ----
37          Y1 = 7.0
38          Y2 = 7.6
39          DO 30 I=1,4
40            X1 = 2*I
41            X2 = X1 + 0.6
42            CALL SGSLAI( I )
43            CALL SGLAU( X1, Y1, X2, Y2 )
44       30 CONTINUE
45
46          CALL SGSLAI( 1 )
47
48    *-- 矢じり部分の長さ ----
49          CALL SGLSET( 'LPROP', .FALSE. )
50          CALL SGRSET( 'CONST', 0.03 )
51
52          Y1 = 6.0
53          DO 40 I=1,8
54            X1 = I
55            X2 = X1 + 0.1*I
56            Y2 = Y1 + 0.1*I
57            CALL SGLAU( X1, Y1, X2, Y2 )
58       40 CONTINUE
59
60          CALL SGLSET( 'LPROP', .TRUE. )
61
62    *-- 矢じり部分の角度 ----
63          Y1 = 5.0
64          Y2 = 5.6
65          DO 50 I=1,8
66            X1 = I
67            X2 = X1 + 0.6
68            CALL SGRSET( 'ANGLE', 10.0*I )
69            CALL SGLAU( X1, Y1, X2, Y2 )
70       50 CONTINUE
71
72    *-- 矢じり部分のぬりつぶし ----
73          CALL SGLSET( 'LSOFTF', .TRUE. )
74          CALL SGLSET( 'LATONE', .TRUE. )
75          CALL SGISET( 'IATONE', 655 )
76
77          Y1 = 4.0
78          Y2 = 4.6
79          DO 60 I=1,8
80            X1 = I
81            X2 = X1 + 0.6
82            CALL SGRSET( 'ANGLE', 10.*I )
83            CALL SGLAU( X1, Y1, X2, Y2 )
84       60 CONTINUE
85
86    *-- ラインサブプリミティブ ----
87    *-- デフォルト ----
88            CALL SGLNU( 0., 3., 10., 3. )
89
90    *-- 線分のラインインデクス ----
91          Y1 = 0.5
92          Y2 = 2.5
93          DO 70 I=1,4
94            X1 = 2*I - 1
95            X2 = X1 + 2.0
96            CALL SGSLNI( I )
```

```
97            CALL SGLNU( X1, Y1, X2, Y2 )
98       70 CONTINUE
99
100         CALL SGCLS
101
102         END
```



kihonc.f: frame1

SGSLAT ルーチンでは矢印を描く線分のラインタイプを, SGSLAI ルーチンではそのラインインデクスを設定できます (2 番めと 3 番め). また, アローサブプリミティブに関する内部変数を設定し直すことにより, 矢じり部分の形状を変えることもできます. 4 番めの例では, 内部変数 'LPROP' を.FALSE. として矢じり部分の長さが一定値となるようにし, その長さを内部変数 'CONST' で陽に与えました. さらに, 矢じり部分の線分と本体部分の線分のなす角を変化させたり (5 番め), 矢じり部分を定義する三角形の領域を塗りつぶしたり (6 番め) することにより, 多種多様な矢印が可能です.

最後に, U-座標系での線分描画は, SGLNU ルーチンで同様に行ないます. しかし, これはアローサブプリミティブの特殊な場合 (矢じり部分がない場合) と考えられますので, 将来的には削除されるかも知れません.

DCL のしくみ 2

xxpGET/xxpSET で内部変数管理

地球流体電脳ライブラリでは, 例えば, SGLGET/SGLSET のような内部変数管理ルーチンが多く使われています. 内部変数管理ルーチンとは, 設定された変数の値を保持し, 問い合わせに答えて値を返す機能を持ったルーチンで, 「掲示版」のような役目をするものです. この様なルーチンを使う理由として, 次のようなことがあげられます.

- 複数のサブルーチンで情報を共有するため
- サブルーチンの引数の個数を最小限にするため

複数のサブルーチンで情報を共有するには, COMMON BLOCK を使うこともできますが, 大きなパッケージでこれを多用するとプログラムの可読性を落とすことにつながります. また, サブルーチンの引数を少くすると融通が効かなくなりますが, かといって, むやみに引数の数を増やすとかえって使いにくいものです.

このような問題を解決するのが内部変数管理ルーチンです. もともと「パッケージの内部で使われる変数」という意味で, 内部変数という言葉を使っていますが, その変数はパッケージ外からも参照/設定できるので, その有効範囲からすると C 言語の「外部変数」に似た性格を持つものです.

内部変数管理ルーチンは xxpGET, xxpSET という名前です. xx は通常パッケージの先頭 2 文字で, p は変数の型によって, I(整数型), R(実数型), L(論理型), C(文字型) のうちのひとつになります. 内部変数は, あらかじめシステムが用意した値 (初期値) をデフォルトで保持しています. この値は xxpGET ルーチンによって参照し, xxpSET ルーチンによって変更することがでます. ユーザーが何も指定しなければ初期値を使うことになります.

第 5 章

レイアウト

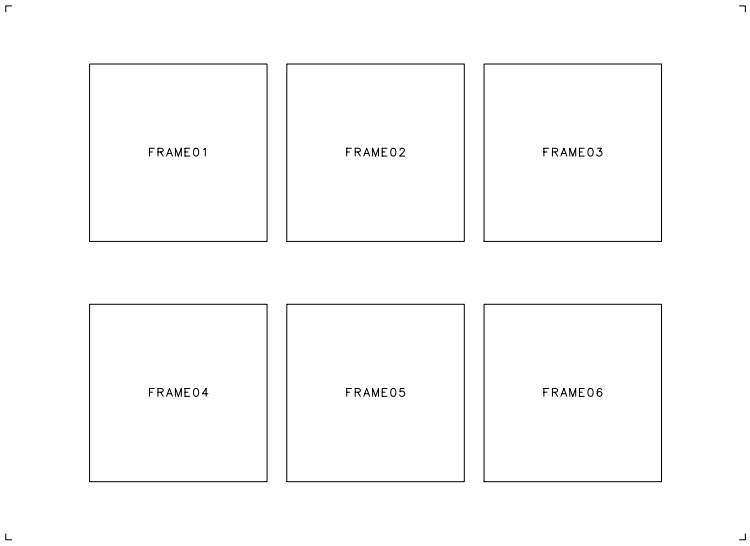
GRPH1 の SLPACK は, 図形の外側にマージン (余白部分) をとったり, 複数の図を 1 つのページにまとめたりする機能を持つパッケージです. マージン部分に文字列 (タイトル) を描くこともできます. SLPACK のルーチンは, 原則として SGOPN と最初に現れる SGFRM の間で呼ばれなければなりません.

5.1　1 ページに複数の図形

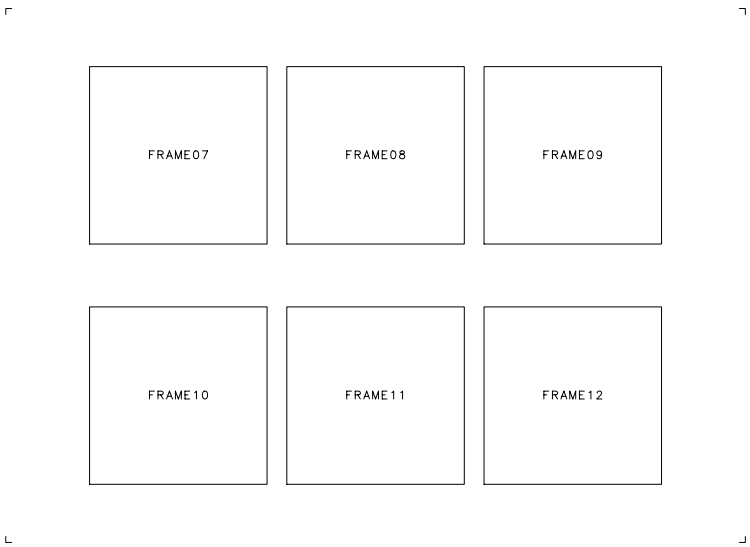
同じ様な図形を沢山並べたい時, SLPACK を使うと非常に簡単にできます. 必要に応じて,「改ページ」の操作も自動的に行なわれます.

最初のプログラム LAY1 では, SGOPN のあと, SLMGN ルーチンを呼び, 第 1 レベルめ (用紙全体) のフレームでマージン (余白部分) をとるように指定しています. ここで, 引数は, 順に左辺, 右辺, 下辺, 上辺のマージンで, それぞれの全幅を 1 とする比率で与えます. 次に, マージンを除いた部分を SLDIV を用いて横 3, 縦 2 に分割し, 次のレベルのフレームを定義します. 最初の引数では順に割り付ける方向を指定します. 'Y' ならば横方向に, 'T' ならば縦方向に割り付けられます. さらに, 14 行めでは分割された第 2 レベルめのフレームでさらに 5%ずつのマージンをとっています. SLDIV は 2 回まで呼ぶことができて, 分割されたフレームをさらにもう一度分割することが可能です.

```
1         PROGRAM LAY1
2
3         CHARACTER*7  CTXT
4         DATA CTXT/'FRAME??'/
5
6         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
7         CALL SGPWSN
8         READ (*,*) IWS
9
10        CALL SGOPN( IWS )
11
12        CALL SLMGN( 0.1, 0.1, 0.05, 0.05 )
13        CALL SLDIV( 'Y', 3, 2 )
14        CALL SLMGN( 0.05, 0.05, 0.05, 0.05 )
15
16        DO 10 I=1,12
17          CALL SGFRM
18          CALL SLPVPR( 1 )
19          WRITE(CTXT(6:7),'(I2.2)') I
20          CALL SGTXV( 0.5, 0.5, CTXT )
21     10 CONTINUE
22
23        CALL SGCLS
24
25        END
```



lay1.f: page1



lay1.f: page2

分割された領域は縦横比が 1 ではありませんが, SGFRM が 1:1 のフレームを設定しています. SLRAT ルーチンを用いると, 今のレベルのフレームすべてについて, 縦横比を指定してフレームが最大内接するようにマージンをとることができます.

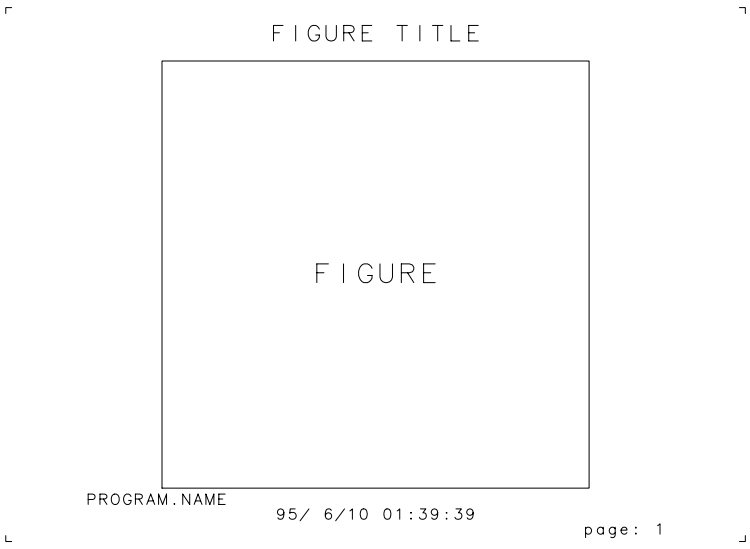
このように分割されたフレームを GRPH1 ではあたかも 1 枚の紙のように扱い, SGFRM の実行により, 次のフレームに自動的に移っていきます. プログラムの DO ループの中では, 普通に改ページをしながら描画するのと同じように, SLPVPR を呼んで各フレームのビューポートの枠を描き, その真中に SGTXV で 'FRAME01', 'FRANE02', などの文字列を書いています. 第 3.1節の出力結果と比較して, 文字の大きさが分割されたフレームの大きさに応じて小さくなっていることに注意して下さい.

このように, 同じ図形を規則的に並べることは SLPACK を使うと非常に簡単にできます. しかし, 大きな図形の横に小さな図形を並べるというようなことは, SLPACK を使うより SGSVPT で陽にビューポートの設定をした方がよいでしょう.

5.2　マージンに文字列を書く

沢山の図を出力すると, いつ, どのプログラムで描いた, 何の図だったのかわからなくなってしまいます. そんな時, SLPACK の機能を使って, タイトル, その図を出力したプログラム名, 使用したデータ名等々をマージンに書き込んでおくと, あとの整理が楽になります.

```
1         PROGRAM LAY2
2
3         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
4         CALL SGPWSN
5         READ (*,*) IWS
6
7         CALL SGOPN( IWS )
8
9         CALL SLMGN( 0.1, 0.1, 0.1, 0.1 )
10        CALL SLSTTL( 'FIGURE TITLE', 'T',  0.,  0., 0.03, 1 )
11        CALL SLSTTL( 'PROGRAM.NAME', 'B', -1.,  1., 0.02, 2 )
12        CALL SLSTTL( '#DATE #TIME',  'B',  0.,  0., 0.02, 3 )
13        CALL SLSTTL( 'page:#PAGE',   'B',  1., -1., 0.02, 4 )
14
15        CALL SGFRM
16        CALL SLPVPR( 1 )
17        CALL SGTXV( 0.5, 0.5, 'FIGURE' )
18
19        CALL SGCLS
20
21        END
```



lay2.f: frame1

LAY2 の例にあるように, タイトルなどを書くには SLSTTL ルーチンで書きたい文字列を指定しておきます. 文字列は, 最大 5 つまでの番号をつけて指定し (最後の引数), それぞれの書くべき位置を設定します. この番号を指定することで, 例えば右下の文字列だけをページごとに変える, ということが可能になります. その場合には, SLSTTL を SGFRM を呼んだあとにも呼ぶことになります. 位置の設定ですが, 第 2 番めの引数で 'T' (トップマージン) または 'B'(ボトムマージン) を指定し, 第 3 番めと第 4 番めの引数で, マージン内における文字列の位置を -1.0(左寄せまたは下寄せ) から$+1.0$(右寄せまたは上寄せ) までの実数値で指定します. 0.0 とすると中央合わせになります. 第 5 番めの引数は文字の高さです.

SLSTTL ルーチンでタイトル等を書くときには, そのためにあらかじめ第 1 レベルのマージンをとっておく必要があります. この時, 文字列が書かれるのは, フレームの分割の有無に関わらず, 第 1 レベルのマージンに対してだけです. ここで注意すべきことは, マージンは各レベルごとの最大作画領域に対する比率で指定されるのに対して, 文字の大きさの単位は第 1 レベルにおける最大作画領域の長辺を 1 とするような単位

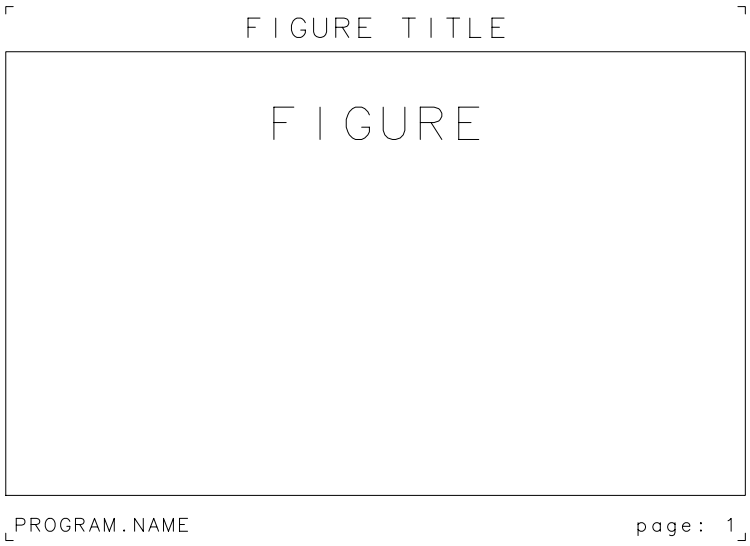
となることです . たとえば , ここで描いたタイトルと次の作画例におけるタイトルでは文字の高さの単位が違うことに注意しましょう.

なお, 12 ・13 行めで用いたように, #DATE, #TIME, #PAGE という予約変数があって, それぞれ, 日付, 時刻, ページ数にあたる文字列に置き換えられます.

5.3　紙を一杯に使う

用紙の形は長方形なのに, そこに内接する正方形の領域をとって, その中だけに図を描くのではもったいない, 紙一杯に図を描きたい, という時にも SLPACK を活用しましょう.

```
1         PROGRAM LAY3
2
3         WRITE(*,*) 'WORKSTATION ID ? '
4         CALL SGPWSN
5         READ (*,*) IWS
6
7         CALL SGOPN( IWS )
8
9         CALL SGLSET( 'LFULL', .TRUE. )
10        CALL SLMGN( 0., 0., 0.08, 0.08 )
11        CALL SLRAT( 1., 0.6 )
12        CALL SLSTTL( 'FIGURE TITLE', 'T',  0.,  0., 0.03, 1 )
13        CALL SLSTTL( 'PROGRAM.NAME', 'B', -1., -1., 0.02, 2 )
14        CALL SLSTTL( 'page:#PAGE',   'B',  1., -1., 0.02, 3 )
15
16        CALL SGFRM
17        CALL SLPVPR( 1 )
18        CALL SGTXV( 0.5, 0.5, 'FIGURE' )
19
20        CALL SGCLS
21
22        END
```



lay3.f: frame1

物理的な描画範囲一杯に作画したい時は, SGPACK の SGLSET を用いて論理型内部変数 'LFULL' を .TRUE. にします. これが, .FALSE.(初期値) の時には, 最大内接する正方形が作画可能な範囲となります. 物理的な描画範囲はデバイスによって違いますから , 'LFULL' を .TRUE. にしたときは, 異なるデバイスに出力する際にエラーを起こして出力できなくなる可能性があります. そこで, 11 行めのように SLRAT を使って　私は 1×0.6 の領域に図を描きたいのだ」と宣言しておくことをお勧めします. SLRAT で縦横比が指定される

と, このフレームが最大内接するように描画領域を設定するので, どんなデバイスに出力してもエラーは起こりません.

この例では, 作画領域の上下に 8% ずつのマージンをとってタイトル等のスペースを確保した後に, 1×0.6 の領域を宣言しています. まだ, 上下に余裕がありますので, 左右にはいっぱいにフレームが確保できました. 上下のマージンを大きくとると, この縦横比を保つために, 左右にもマージンが自動的にできるようになります.

SLPACK では基本的に最大描画領域に対する比率でマージンなどをとるようになっていますが, 目的によっては絶対的な長さ (例えば 10cm) を指定したい場合もあるでしょう. SLSIZE または SLFORM で第 1 レベルめのフレームを再設定しておけば, その範囲が物理的に描画できる範囲内である限り, 異なったデバイスでも同じ大きさの図が出力できます. SLFORM は描画範囲を A4, B5 等の規格の大きさで, また, SLSIZE は cm 単位で指定します. 通常 A4 の用紙の最大描画範囲は用紙そのものの大きさよりも小さいので, A4 の紙に SLSIZE で A4 を指定するとエラーとなりますから, ご注意下さい. また, コンソールディスプレイなど, 物理的な大きさがはっきりしないデバイスに対しては, 適当な大きさが仮定されています.

FORTRAN のひけつ 1

不定の概念

FORTRAN 規格の基本的な概念に 「不定」という概念があります. これは最も重要な概念の一つでありながら, 最もわかりにくい概念でもあります. 岩波 FORTRAN 辞典には,

> プログラム実行中に, 変数, 配列要素または部分列が想定できるような値を持たない定義状態.

とありますが, なかなかイメージが湧きません. 簡単にいえば, 「○ ○ の時, 変数× × は不定となる.」という文章は, コンパイラをつくる人に対しては, 「○ ○ の時, 変数× × が占めていた記憶領域をどのように使ってもよい.」ということになり, 我々プログラマに対しては, 「コンパイラをつくる人に, 上のように言ってしまったので, × × がどうなるかわからない.」ということになります.

「不定」とは逆に, 規格によって値が保証されている状態を 「確定」といいます. 規格上, プログラムの中で引用できる変数は 「確定」された変数だけです. 変数が 「不定」となる例に次のようなものがあります.

- 変数が未定義のとき
- サブルーチンの中の RETURN 文または END 文を実行したとき
- 異なる型の変数が結合しているとき

2 番めの状況を回避するには SAVE 文を使います. SAVE 文は, サブルーチンの実行が終ってもサブルーチン内の変数の値を保持するように指示する宣言文です. コンパイラによっては (というより多くのコンパイラでは) サブルーチンの中で使われる局所変数を保存しますが, これは 「方言」です. ただし, DATA 文で指定された変数は, その値が書き換えられない限り, RETURN 文または END 文を実行しても 「不定」にはなりません.

因みに, DCL の `xxpGET/xxpSET` ルーチンが, 掲示板の役目を果たせるのは, この DATA 文と SAVE 文のおかげです.

第 6 章

USPACK を一工夫

この章から, 上位ルーチン群である GRPH2 の各パッケージについて解説します.

まず, 自動スケーリングルーチンパッケージ USPACK ですが, すでに第 2.1節などで見たように USGRPH ルーチンでデータを簡単にグラフ化することができます. このパッケージの主な機能は, 次の 2 つです.

1. データの最大値と最小値を調べて, 適切な正規化変換を設定する.
2. 現在設定されている正規化変換に対して, 適切な目盛及びラベル間隔を計算して, 座標軸を描く.

座標軸を描くルーチンは独立に呼ぶこともできるので, 簡単に座標軸を描くユーティリティとして使うことも可能です. 第 2.2節で紹介したようにコンターを描く UDPACK などと組み合せて使うと便利です.

なお, 実際の折れ線グラフの描画は UUPACK, 座標軸やタイトルの描画は UXPACK/UYPACK/UZPACK に依存しています. ここで, DCL 全体で使用する内部変数　朱定義値」をうまく使うと, より簡単に一次元図が描けるようになります.

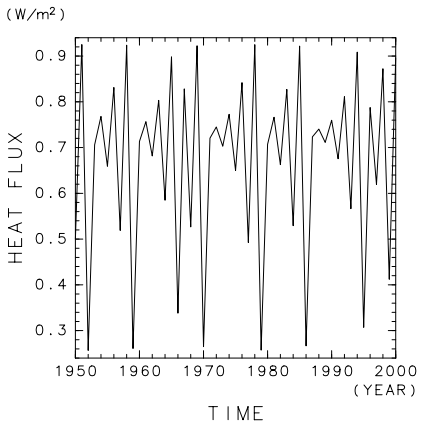
6.1　タイトルを描く

第 2.1節などで見たように, USGRPH ルーチンを使うと, 容易に折れ線グラフが描けます. しかし, 座標軸にタイトルや単位をつけておかないと, 後で何のグラフだったのかわからなくなってしまいます. 次の USPAC1 のプログラム例のように, USSTTL を使えば, 座標軸のタイトルや単位を簡単に描くことができます. 4 つの引数は, 順に, x 座標のタイトル, 単位, y 座標のタイトル, 単位, です. これらを付けたくない時には, 1 文字以上の空白文字を指定します.

なお, これらのタイトルや単位は GRFRM, GRFIG により初期化されます. 同じタイトルで沢山の図を描きたい時には USISET で内部変数'IRESET' を 0 にしておけば, タイトルの設定は 1 度で済みます.

```
1          PROGRAM USPAC1
2
3          PARAMETER( NMAX=50 )
4          REAL X(0:NMAX), Y(0:NMAX)
5
6          R    = 3.7
7          X(0) = 1950.
8          Y(0) = 0.5
9          DO 10 N=0,NMAX-1
10           X(N+1) = X(N) + 1.
11           Y(N+1) = R*Y(N)*(1.-Y(N))
12       10 CONTINUE
```

```
13
14          WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
15          CALL SGPWSN
16          READ (*,*) IWS
17
18          CALL GROPN( IWS )
19          CALL GRFRM
20
21          CALL USSTTL( 'TIME', 'YEAR', 'HEAT FLUX', 'W/m|2"' )
22          CALL USGRPH( NMAX+1, X, Y )
23
24          CALL GRCLS
25
26          END
```



uspac1.f(uspac3.f): frame1

6.2　USGRPH の分解

ここで, プログラム USPAC1 で呼んだ USGRPH サブルーチンがどのようなルーチンで構成されているか, 見てみましょう. 実は,

```
CALL USGRPH( N, X, Y )
```

というサブルーチン ・コールは, 次の 5 つのサブルーチンを順に呼ぶことと同じなのです.

```
CALL USSPNT( N, X, Y )
CALL USPFIT
CALL GRSTRF
CALL USDAXS
CALL UULIN( N, X, Y )
```

データを自動的にスケーリングするためには, まず, 描きたいデータすべてのなかから最大値と最小値を見つける必要があります. サブルーチン USSPNT がこれを行ないます. つぎの USPFIT ルーチンでは, これらのデータの最大値 ・最小値を切りの良い数値に丸めてウインドウを決め, ほかの正規化変換のパラメータも 「おまかせ」で決めます. そして, GRSTRF ルーチンで正規化変換を確定します.

ここで, 「おまかせ」の中身を見てみましょう.

まず, MATH1 の SYSLIB パッケージが管理する内部変数 RUNDEF のお話です. これは DCL 全体で使用する内部変数のひとつで, GLRGET/GLRSET ルーチンによって参照/変更できます. RUNDEF は 「ユーザーが陽に指定していない」ことを表す実数値で, 「未定義値」と呼びます. 初期値は $-999.$ です.

さて, この未定義値 RUNDEF ですが, GRFRM ルーチンで新しい作画領域を設定する際に使われます. GRFRM を呼ぶと, ウインドウ, ビューポート, および変換関数番号の正規化変換に関する変数に RUNDEF が代入され, これらが未定義状態になります. ここで, USSPNT と USPFIT を呼ぶと, ウインドウについては最大値・最小値を切りの良い数値に丸めて設定し, それ以外の未定義状態にある変数には次の初期値を設定します:

ビューポート:　(0.2, 0.8, 0.2, 0.8)
変換関数番号:　1 (直角一様座標)

当然ながら, USPFIT ルーチンの前に GRSWND, GRSVPT, または GRSTRN でこれらの変数を設定していると, それは未定義状態ではなくなっているので, その値のまま GRSTRF ルーチンで確定されます.

GRSTRF ルーチンの次の USDAXS は「おまかせ」で座標軸を描くルーチンです. 次章以降で説明する座標軸ルーチンを使っているのですが, すでに見たように座標軸ラベルがデータに合わせて自動的に付けられます (第 2.1節, quick2.f).

折れ線を描いているのは, UUPACK の UULIN ルーチンです. 基本的には, SGPACK のポリラインプリミティブに対応するものですが, 次節以降で具体的に見るように別の機能が付け加えられています.

6.3　複数のデータを 1 つのグラフに描く

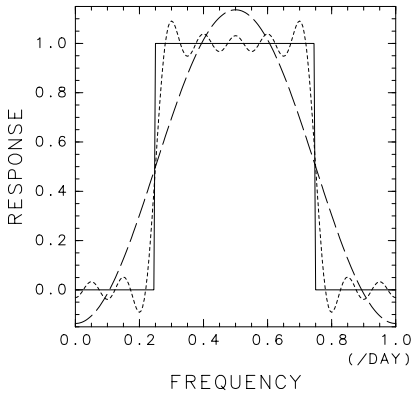
この章のはじめに紹介した USPAC1 のプログラムでは, 一つのグラフに一つのデータをプロットしましたが, 複数のデータを折れ線の属性を変えながら 1 つのグラフに描いてみましょう (USPAC2).

まず, 前節で紹介した USSPNT ルーチンを 3 回呼んで X と Y0, Y1, Y2 のデータのなかから x と y の最大値と最小値を見つけ, USPFIT と GRSTRF ルーチンで正規化変換を確定します. そして, つぎの USSTTL と USDAXS で座標軸を描きます.

最後に, UUSLNT と UUSLNI のサブルーチンで折れ線の属性 (線種と線の太さ) を変更しながら, UULIN ルーチンで折れ線を描きます. これら UUPACK のサブルーチンの動作は, この場合には SGPACK の SGPLU ルーチンなどと同じです. (便利な機能は次節の例で示します.)

```
1          PROGRAM USPAC2
2
3          PARAMETER( NMAX=201, IMAX=5 )
4          REAL X(NMAX), Y0(NMAX), Y1(NMAX), Y2(NMAX), A(IMAX)
5
6          PI = 3.14159
7          DO 10 I=1,IMAX
8            II = 2*I - 1
9            A(I) = (-1)**I *2./(II*PI)
10       10 CONTINUE
11
12         DO 20 N=1,NMAX
13           X(N)  = 1.*(N-1)/(NMAX-1)
14           T     = 2.*PI*X(N)
15           IF(T.LT.PI/2. .OR. T.GE.PI*3./2.) THEN
16             Y0(N) = 0.
17           ELSE
18             Y0(N) = 1.
19           END IF
20           Y1(N) = 0.5 + A(1)*COS(T)
21           Y2(N) = 0.5
22           DO 30 I=1,IMAX
23             II = 2*I - 1
24             Y2(N) = Y2(N) + A(I)*COS(II*T)
25       30    CONTINUE
26       20 CONTINUE
27
28         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
```

```
29          CALL SGPWSN
30          READ (*,*) IWS
31
32          CALL GROPN( IWS )
33          CALL GRFRM
34
35          CALL USSPNT( NMAX, X, Y0 )
36          CALL USSPNT( NMAX, X, Y1 )
37          CALL USSPNT( NMAX, X, Y2 )
38          CALL USPFIT
39          CALL GRSTRF
40
41          CALL USSTTL( 'FREQUENCY', '/DAY', 'RESPONSE', ' ' )
42          CALL USDAXS
43
44          CALL UULIN( NMAX, X, Y0 )
45          CALL UUSLNT( 2 )
46          CALL UUSLNI( 3 )
47          CALL UULIN( NMAX, X, Y1 )
48          CALL UUSLNT( 3 )
49          CALL UULIN( NMAX, X, Y2 )
50
51          CALL GRCLS
52
53          END
```



uspac2.f: frame1

6.4　等間隔データをおまかせにする

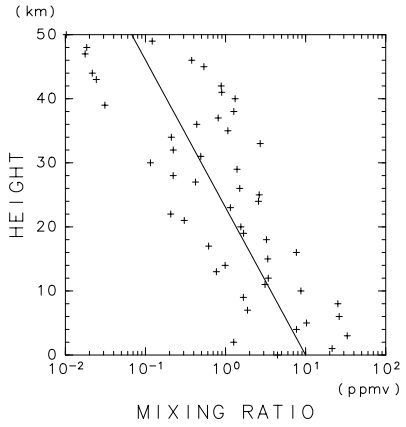
これまでのプログラム USPAC1 や USPAC2 では, x 座標値の配列を宣言して等間隔に値を代入し, $f(x)$ 型の一次元図を描きましたが, これはちょっと大げさな気もします. ここで, 未定義値 RUNDEF をうまく使うと, プログラムが簡単になります. 次のプログラム USPAC3 は USPAC1 と同じ図を描くプログラムですが, x 座標値は等間隔なので, UUPACK におまかせします.

まず, 19 行めで RUNDEF の値を参照し, 22 行めの USGRPH ルーチンで X を指定するかわりに RUNDEF を指定します. このように, X は定義されていないと宣言すると, USGPRH は x 座標値がウインドウの幅いっぱいに等間隔にならんでいるものと解釈してグラフを描きます. このとき, USGRPH が呼ばれる前に x 方向のウインドウは決まっていないといけませんから, GRSWND ルーチンで x 方向だけを陽に与えています. ここでも, y 方向は未定義にして, 正規化変換の確定は USGPRH ルーチンにおまかせしています. 実行結果は, 前の USPAC1 の場合と全く同じです.

```
      PROGRAM USPAC3

      PARAMETER( NMAX=50, XMIN=1950, XMAX=2000 )
      REAL Y(0:NMAX)

      R    = 3.7
      Y(0) = 0.5
      DO 10 N=0,NMAX-1
        Y(N+1) = R*Y(N)*(1.-Y(N))
   10 CONTINUE

      WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
      CALL SGPWSN
      READ (*,*) IWS

      CALL GROPN( IWS )
      CALL GRFRM

      CALL GLRGET( 'RUNDEF', RUNDEF )
      CALL GRSWND( XMIN, XMAX, RUNDEF, RUNDEF )
      CALL USSTTL( 'TIME', 'YEAR', 'HEAT FLUX', 'W/m|2"' )
      CALL USGRPH( NMAX+1, RUNDEF, Y )

      CALL GRCLS

      END
```

次のプログラム USPAC4 では, USSPNT や UULIN などのルーチンで Y を指定するかわりに RUNDEF を指定して, y 方向に等間隔なデータを描きます. ここで, UUMRK ルーチンはポリマーカープリミティブに対応するもので, UUSMKT, UUSMKI, UUSMKS でこのパッケージで使うマーカーの属性 (種類, 描く線の太さ, 大きさ) を設定できます. USSPNT ルーチンを使ってウインドウを決め, ビューポートの設定は初期値に頼りますので, GRSTRF ルーチンの前に USPFIT を呼んでいることを再確認しておきましょう.

```
      PROGRAM USPAC4

      PARAMETER( NMAX=50, YMIN=0., YMAX=50. )
      REAL X1(0:NMAX), X2(0:NMAX)

      ISEED = 1
      DO 10 N=0,NMAX
        Y = YMIN + (YMAX-YMIN)*N/NMAX
        X1(N) = 10.*(EXP(-Y/20))**2 * EXP((RNGU3(ISEED)-0.5)*2)**2
        X2(N) = 10.*(EXP(-Y/20))**2
   10 CONTINUE

      WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
      CALL SGPWSN
      READ (*,*) IWS

      CALL GROPN( IWS )
      CALL GRFRM

      CALL GLRGET( 'RUNDEF', RUNDEF )
      CALL GRSWND( RUNDEF, RUNDEF, YMIN, YMAX )
      CALL USSPNT( NMAX+1, X1, RUNDEF )
      CALL USSPNT( NMAX+1, X2, RUNDEF )
      CALL GRSTRN( 3 )
      CALL USPFIT
      CALL GRSTRF

      CALL USSTTL( 'MIXING RATIO', 'ppmv', 'HEIGHT', 'km' )
      CALL USDAXS

      CALL UUSMKT( 2 )
      CALL UUMRK( NMAX+1, X1, RUNDEF )
      CALL UULIN( NMAX+1, X2, RUNDEF )

      CALL GRCLS

      END
```



uspac4.f: frame1

この章で紹介した正規化変換のおまかせ確定は,「らくらく DCL」の他の章では使っていません. 教育的な配慮を優先したからで, 正規化変換の内容を陽に意識してもらいたいからです. しかし, この章で「おまかせ」の意味がはっきり分かったら, これを積極的に使って何も問題ありません. より簡単にプログラムが書けることでしょう.

DCL のしくみ 3

根回し型ルーチンと上意下達型ルーチン

SGPACK の各出力プリミティブにはそれぞれ 2 種類の描画ルーチンがあります.

例えば, ポリラインプリミティブでは, SGPLV, SGPLU ルーチンで折れ線を描くように説明しました. ポリラインの属性は SGSPLT, SGSPLI などのルーチンで前もって設定しておきました. これらは, いろいろ属性を決めておいてから描画ルーチンを呼ぶという「根回し型」のルーチンです.

これに対して, 属性も同時に指定してポリラインを描画するルーチンも用意されています. 呼び出し方法は, それぞれ,

```
CALL SGPLZV(N,VPX,VPY,ITYPE,INDEX)
CALL SGPLZU(N,UPX,UPY,ITYPE,INDEX)
```

です. 引数の ITYPE でラインタイプを与え, INDEX でラインインデクスを与えます. これらは, サブルーチンひとつで一度に描いてしまう「上意下達型」のルーチンです.

同様に, 他の出力プリミティブも上意下達型のルーチンが用意されています. 引数の説明は省きますが, 次のように呼び出します.

```
CALL SGPMZV(N,VPX,VPY,ITYPE,INDEX,RSIZE)
CALL SGPMZU(N,UPX,UPY,ITYPE,INDEX,RSIZE)
CALL SGTXZV(VX,VY,CHARS,RSIZE,IROTA,ICENT,INDEX)
CALL SGTXZU(UX,UY,CHARS,RSIZE,IROTA,ICENT,INDEX)
CALL SGTNZV(N,VPX,VPY,ITPAT)
CALL SGTNZU(N,UPX,UPY,ITPAT)
CALL SGLAZV(VX1,VY1,VX2,VY2,ITYPE,INDEX)
CALL SGLAZU(UX1,UY1,UX2,UY2,ITYPE,INDEX)
CALL SGLNZV(VX1,VY1,VX2,VY2,INDEX)
CALL SGLNZU(UX1,UY1,UX2,UY2,INDEX)
```

他のグラフィクスパッケージでも, GKS の流れは前者の型で, CALCOMP 系のものは後者の型です. これまでにどちらかの型に馴染んでいるユーザーは, 好みの方をお使い下さい.

第 7 章

座標軸

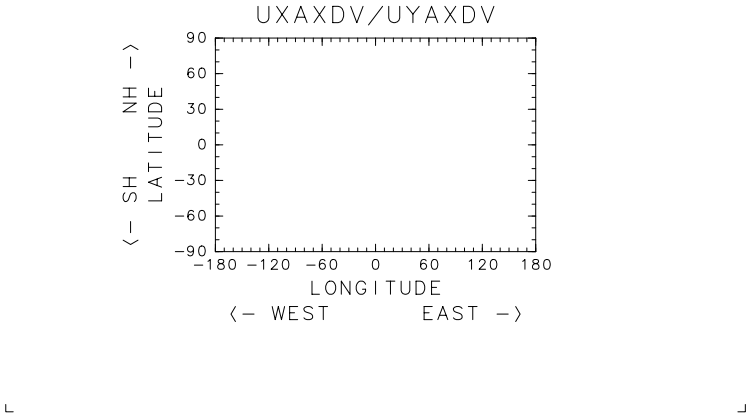
ここでは GRPH2 のパッケージのうち座標軸描画に関連する U[XYZ]PACK/ULPACK/UCPACK の基本的な機能を紹介しましょう. これらのパッケージを使うと数行のサブルーチンコールで多種多様な座標軸を描くことができます. これらは序の口で, さらに凝った座標軸を描くこともできますが, それらについては次章で説明しましょう.

なお, 以下のプログラム例では, GRPACK を用いて初期化や改ページなどの操作をおこなっていますから, 初期化ルーチン UZINIT は陽に呼ばなくてもよいようになっています.

7.1 線形座標軸

まず, 最も簡単なプログラム例 UXYZ1 を見てみましょう. UXPACK/UYPACK のサブルーチンを使って, x 軸, y軸ともに線形座標軸 (均等な目盛り) を描き, それぞれの軸には目盛りとラベル (目盛りのところにつける数字), およびタイトル (軸の説明) をつけます. また, 図のタイトルもつけることにします.

```
1         PROGRAM UXYZ1
2
3         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
4         CALL SGPWSN
5         READ (*,*) IWS
6
7         CALL GROPN( IWS )
8         CALL GRFRM
9
10        CALL GRSWND( -180., 180., -90., 90. )
11        CALL GRSVPT(   0.2,  0.8,  0.3, 0.7 )
12        CALL GRSTRN( 1 )
13        CALL GRSTRF
14
15        CALL UXAXDV( 'B', 10., 60. )
16        CALL UXAXDV( 'T', 10., 60. )
17        CALL UXSTTL( 'B', 'LONGITUDE', 0. )
18        CALL UXSTTL( 'B', '<- WEST      EAST ->', 0. )
19
20        CALL UYAXDV( 'L', 10., 30. )
21        CALL UYAXDV( 'R', 10., 30. )
22        CALL UYSTTL( 'L', 'LATITUDE', 0. )
23        CALL UYSTTL( 'L', '<- SH    NH ->', 0. )
24
25        CALL UXMTTL( 'T', 'UXAXDV/UYAXDV', 0. )
26
27        CALL GRCLS
28
29        END
```



uxyz1.f: frame1

まず, 座標軸関連のパッケージを使用するときには, ウインドウとビューポート (第 4.1.2節) が適切に設定されていなければなりません. そして, 座標軸の作画を行なうにあたって最も基本的なことは, 1 本の座標軸 (1 本の軸とそれに付ける目盛りおよびラベルによって構成される) を描くために作画ルーチンを 1 回呼ぶということです. この例では, 上下左右あわせて 4 回作画ルーチン (x 軸は UXAXDV, y軸は UYAXDV) を呼んでいます. 最初の引数 'B', 'T', 'L', 'R' によって, それぞれ, 下, 上, 左, 右側の軸を描画することを指定します. 座標軸を描くこれらの場所は, ウインドウとビューポートを設定した矩形領域のちょうど境界線上になります. 次の 2 つの引数では, 短い目盛りと長い目盛りをどんな間隔で打つかを U-座標系の値で指定します. ラベルは, 長い目盛りのところにだけ描かれます.

軸につけるタイトルおよび図のタイトルも, それぞれの軸についてタイトル描画ルーチンを呼びます. 小さめの文字でタイトルを描きたいときは UXSTTL, UYSTTL ルーチンを, 大きめの文字で描きたいときには UXMTTL, UYMTTL ルーチンを用います. 最初の引数によってタイトルをつける軸の場所を指定します. 2 番目の引数はタイトルとして描く文字列です. 最後の引数では, タイトルを描く位置を-1.0 から 1.0 までの実数値で指定します. たとえば x 軸については, -1.0: 左寄せ, 0.0: 中央合わせ, 1.0: 右寄せとなり, また y 軸については, -1.0: 下寄せ, 0.0: 中央合わせ, 1.0: 上寄せとなります. この例の下側 x 軸や左側 y軸のように, 同じ軸に対して 2 回以上タイトル描画ルーチンを呼んだときには, タイトルが重ならないようにだんだんと外側の方へずらして描かれます.

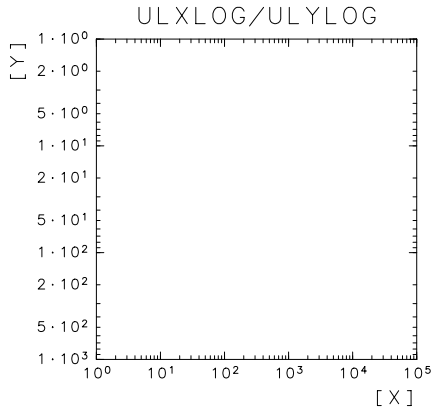
言われるままに作画してみて, 『はて, タイトルやラベルの大きさ, 目盛の長さなどは一体どのように決まっているのだろか?』と, 不思議に思うかもしれません. じつはこれらの属性は, UZPACK の管理する内部変数を参照して決められているのです. ふつうの用途には, これらで充分に満足できる座標軸が作画できるでしょう. タイトルの大きさやその描く向きなどを指定して独自の座標軸を作ることもできますが, それは次章にまわしましょう.

7.2　対数座標軸

次のプログラム UXYZ2 は対数座標軸の例です. 対数座標軸の描画は ULPACK が担当しています.

```
1           PROGRAM UXYZ2
2
```

```
3         WRITE(*,*) ' WORKSTATION ID (I) ? ;'
4         CALL SGPWSN
5         READ (*,*) IWS
6
7         CALL GROPN( IWS )
8         CALL GRFRM
9
10        CALL GRSWND( 1.E0, 1.E5, 1.E3, 1.E0 )
11        CALL GRSVPT(  0.2,  0.8,  0.2,  0.8 )
12        CALL GRSTRN( 4 )
13        CALL GRSTRF
14
15        CALL ULXLOG( 'B', 1, 9 )
16        CALL ULXLOG( 'T', 1, 9 )
17        CALL UXSTTL( 'B', '[X]', 1. )
18
19        CALL ULYLOG( 'L', 3, 9 )
20        CALL ULYLOG( 'R', 3, 9 )
21        CALL UYSTTL( 'L', '[Y]', 1. )
22
23        CALL UXMTTL( 'T', 'ULXLOG/ULYLOG', 0. )
24
25        CALL GRCLS
26
27        END
```



uxyz2.f: frame1

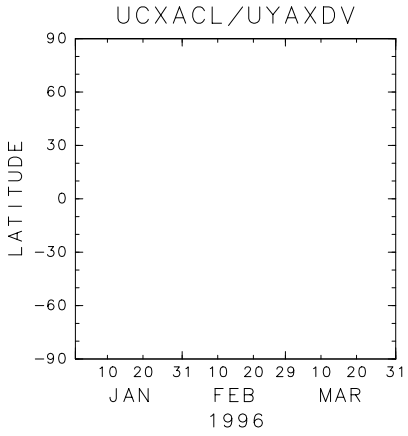
対数座標軸を描くには, まず GRSTRN ルーチンによって対数変換をあらわす変換関数番号を設定します. 変換関数番号についてはすでに第 4.1.3節で説明しましたが, 1: 直角一様座標 (線形座標), 2: 片対数 (y軸) 座標, 3: 片対数 (x 軸) 座標, 4: 両対数座標です.

対数座標軸を描くサブルーチンは ULXLOG, ULYLOG です. ここでも, 最初の引数で座標軸を描く場所を指定します. 2 番めの引数は, 1 桁の範囲に描くラベルの数であり, x 軸については 1 となっていて, 10^n のところにのみラベルが描かれます. また, y軸については 3 となっていて, 10^n 以外にも 2×10^n, 5×10^n にラベルが描かれてます. この引数が 2 ならば, 10^n と 2×10^n にラベルが描かれます. 最後の引数は, 1 桁の範囲に描く目盛りの数です. ここでは 1 から 9 まですべての目盛を打つように, 9 を指定しています. 8 以下の場合には, 目盛間隔の狭いところから省かれます.

7.3 これはうれしい日付軸

長年にわたるデータを解析した人なら, 日付軸をつけることの面倒くささは十分ご承知でしょう. UCPACK を用いると, 閏年まで考慮した完璧な日付軸を容易に作画することができます (UXYZ3).

```
1         PROGRAM UXYZ3
2
3         PARAMETER( ID0=19960101, ND=90, RND=ND )
4
5         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
6         CALL SGPWSN
7         READ (*,*) IWS
8
9         CALL GROPN( IWS )
10        CALL GRFRM
11
12        CALL GRSWND( 0.0, RND, -90., 90. )
13        CALL GRSVPT( 0.2, 0.8,  0.2, 0.8 )
14        CALL GRSTRN( 1 )
15        CALL GRSTRF
16
17        CALL UCXACL( 'B', ID0, ND )
18        CALL UCXACL( 'T', ID0, ND )
19
20        CALL UYAXDV( 'L', 10., 30. )
21        CALL UYAXDV( 'R', 10., 30. )
22        CALL UYSTTL( 'L', 'LATITUDE', 0. )
23
24        CALL UXMTTL( 'T', 'UCXACL/UYAXDV', 0. )
25
26        CALL GRCLS
27
28        END
```



uxyz3.f: frame1

まず, 作画しようとする座標軸については, 日数を単位として正規化変換を設定する必要があります. また, 座標軸の作画は, U-座標系において 0 に相当する位置からおこなわれますから, この例の場合のように 1996 年 1 月 1 日から 3 月 31 日まで 91 日分をビューポートいっぱいに割り当てたいときには, ウインドウの両端値を UXMIN が 0.0, UXMAX が RND(= 90) と指定します (12 行め).

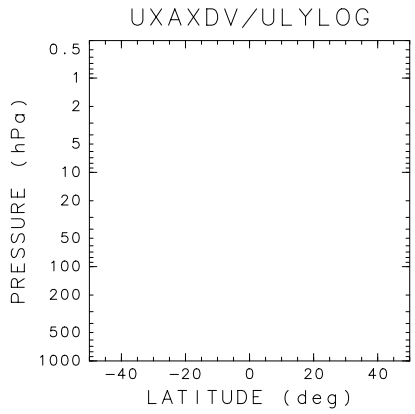
日付に関する座標軸は, サブルーチン UCXACL, UCYACL で簡単に描けます. 最初の引数では, これまで通り, 座標軸を描く場所を指定します. 2 番めの引数 ID0 は起日, つまり座標軸を描きはじめる最初の日です. 3 行めのパラメータ文をみれば明らかなように, 8 桁の整数 ($yyyymmdd$) で指定します. $yyyy$が年, mm が月, dd が日で, この場合 ID0=19960101 ですから, 起日は 1996 年 1 月 1 日となります. 最後の引数では, 何日間分を描くかを指定します.

7.4 中途半端なウインドウ

ウインドウの最小値 ・最大値は必ずしも切りのよい値でなくても大丈夫です. UXYZ4 のプログラム例では, x 軸については-50 から 50 の範囲で, y軸については 1000 から 0.4 の範囲でウインドウを設定しています. このとき, UXAXDV はきざみ値の整数倍となるところに目盛とラベルを描きます. また, ULYLOG は 10^n から 10^{n+1}を 1 サイクルとして目盛とラベルを描きます. このように, 目盛やラベルは切りのよいサイクルを単位として作画がおこなわれます. さらに, 好みの間隔で目盛をうち, 好みのところにラベルを描きたい場合は, 次章を見て下さい.

なお, この例では, ULYLOG を用いて対数座標軸を描画する際に, ULISET ルーチンで整数型内部変数 'IYTYPE' を 3 に変更してラベルの書式を変えました. 対数軸に付けるラベルの書式は全部で 4 種類用意されています.

```
1         PROGRAM UXYZ4
2
3         WRITE(*,*) ' WORKSTATION ID (I) ? ;'
4         CALL SGPWSN
5         READ (*,*) IWS
6
7         CALL GROPN( IWS )
8         CALL GRFRM
9
10        CALL GRSWND( -50., 50., 1.E3, 0.4 )
11        CALL GRSVPT(  0.2, 0.8,  0.2, 0.8 )
12        CALL GRSTRN( 2 )
13        CALL GRSTRF
14
15        CALL UXAXDV( 'B', 5., 20. )
16        CALL UXAXDV( 'T', 5., 20. )
17        CALL UXSTTL( 'B', 'LATITUDE (deg)', 0. )
18
19        CALL ULISET( 'IYTYPE', 3 )
20        CALL ULYLOG( 'L', 3, 9 )
21        CALL ULYLOG( 'R', 3, 9 )
22        CALL UYSTTL( 'L', 'PRESSURE (hPa)', 0. )
23
24        CALL UXMTTL( 'T', 'UXAXDV/ULYLOG', 0. )
25
26        CALL GRCLS
27
28        END
```



uxyz4.f: frame1

計算機のなまり 3

文字の内部表現

文字の内部表現も実数表現と同様に機種に依存しています. DCL の CHGLIB と CHKLIB のサブパッケージは, 機種依存する文字処理を規格化するためのものです.

FORTRAN では以下の FORTRAN 文字集合が定められており, プログラムは, 注釈行や文字型データの中は例外として, これらの文字だけで書かなければなりません.

英字:　ABCDEFGHIJKLMNOPQRSTUVWXYZ
数字:　0123456789
特殊文字:　空白 '()*+,-./:= 通貨記号

したがって, 小文字で FORTRAN プログラムを書くのは厳密に言えば文法違反になります.

これらの FORTRAN 文字を含めて, 文字の内部表現方法は FORTRAN の規格では特に規定されていません. 文字の内部表現方法は実数の場合と同様に, IBM 規格である EBCDIC と, アメリカの標準規格である ASCII とに大別されます.

EBCDIC は, IBM によって定められた拡張 2 進化 10 進情報交換用コード (Extended Binary Coded Decimal Interchange Code) です. 富士通, 日立などのいわゆる IBM 互換の汎用機でも採用されていますが, 各社で微妙に定義が異なります. 富士通のコードには, IBM 規格以外の制御コードも定義されており, 日立の EBCDIK(最後の K はカナ) ではアルファベットの小文字のコードが異なります.

ASCII コードはアメリカ規格協会 (ANSI) で規定された文字コード体系です. UNIX, MS-DOS などで採用されている文字コードです. 日本では, ほぼ同じものが JIS X0201 として規定されています. ASCII コードは 7 ビットで, 最上位桁は 0 ですが, JIS には 8 ビット全部使ってカタカナまで規定した 8 単位符号表があります.

詳細は, MISC1 のマニュアル (dcl-x.x/doc/misc1/gaiyou/char.tex) を御覧下さい.

第 8 章

もっと座標軸

座標軸描画ルーチンのさらに進んだ機能を見てみましょう. おそらく, 誰の要求も満たしてくれることと思います.

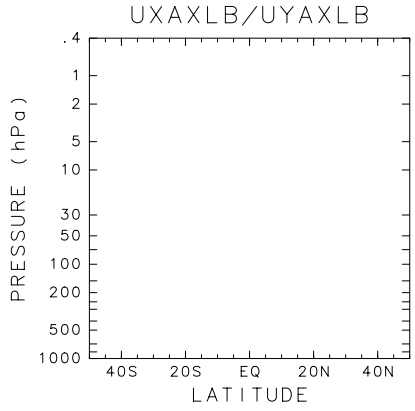
8.1 注文の多い目盛りうち

好みの間隔で目盛りをうち, 好みのところにラベルを描く. しかも, ラベルには数字だけでなく文字列も使いたい.」 そのような要求にも U[XYZ]PACK のパッケージは柔軟に対応できます.

スケーリングは前章の例 UXYZ4 と同じで, 緯度のラベルは文字列で表現し, また圧力の目盛はデータが存在するレベルだけに打つというプログラムが, UXYZ5 です.

```
1         PROGRAM UXYZ5
2
3         PARAMETER( NX1=21, NX2= 5 )
4         PARAMETER( NY1= 0, NY2=18 )
5         REAL      RX1(NX1), RX2(NX2), RY2(NY2)
6         CHARACTER CX2(NX2)*4, CY2(NY2)*4
7
8         DATA      RX1/-50,-45,-40,-35,-30,-25,-20,-15,-10, -5,  0,
9        +               5, 10, 15, 20, 25, 30, 35, 40, 45, 50/
10        DATA      RX2/  -40 ,  -20 ,    0 ,   20 ,   40 /
11        DATA      CX2/'40S ','20S ','EQ  ','20N ','40N '/
12        DATA      RY2/ 1000 , 850  , 700  , 500  , 400  , 300  ,
13       +              250  , 200  , 150  , 100  , 70   , 50   ,
14       +              30   , 10   , 5    , 2    , 1    , 0.4  /
15        DATA      CY2/'1000','    ','    ','500 ','    ','    ',
16       +             '    ','200 ','    ','100 ','    ','50  ',
17       +            '30  ','10  ','5   ','2   ','1   ','.4  '/
18
19        WRITE(*,*) ' WORKSTATION ID (I) ? ;'
20        CALL SGPWSN
21        READ (*,*) IWS
22
23        CALL GROPN( IWS )
24        CALL GRFRM
25
26        CALL GRSWND( -50., 50., 1.E3, 0.4 )
27        CALL GRSVPT(  0.2, 0.8,  0.2, 0.8 )
28        CALL GRSTRN( 2 )
29        CALL GRSTRF
30
31        CALL UXAXLB( 'B', RX1, NX1, RX2, CX2, 4, NX2 )
32        CALL UXAXLB( 'T', RX1, NX1, RX2, CX2, 4, NX2 )
33        CALL UXSTTL( 'B', 'LATITUDE', 0. )
34
35        CALL UYAXLB( 'L', DUMMY, NY1, RY2, CY2, 4, NY2 )
36        CALL UYAXLB( 'R', DUMMY, NY1, RY2, CY2, 4, NY2 )
37        CALL UYSTTL( 'L', 'PRESSURE (hPa)', 0. )
38
39        CALL UXMTTL( 'T', 'UXAXLB/UYAXLB', 0. )
```

```
40
41          CALL GRCLS
42
43          END
```



uxyz5.f: frame1

目盛とラベルを描く場所を指定して座標軸を描くサブルーチンとして, UXAXNM, UYAXNM があり, さらに, 描くラベルも指定するルーチンに UXAXLB, UYAXLB があります. この例では, 後者を使っています. 最初の引数は場所を指定する引数です. 2, 3 番目の引数では, 小さめの目盛りをうつ場所に関する情報を指定します. 2 番目の引数は, 小さい目盛りをうつ場所の U 座標系での値を与える実数型配列で, 3 番目はその配列の長さです. 4 番目から 7 番目までの引数では, 同様にして, 大きめの目盛りについての情報を与えます; 目盛りをうつ場所を指定する配列, ラベルを指定する文字型配列, ラベルの文字数, 配列の長さ, の順に与えます. これらの情報は, 8 行めからのデータ文で用意しています.

なお, ラベルとして与える文字列の有効な長さは, 後方のブランクを無視して数えます. 具体的には, x 軸の 'EQ□□' のように, DATA 文では後方に 2 文字のブランクを含んで文字列を与えても, 先行する有効な 2 文字のみがラベルの作画対象となり, 2 文字分がセンタリングされて作画されます. y軸に関しても同様に, 文字列 '.4□□' なども後方のブランクを無視して右寄せして描かれます.

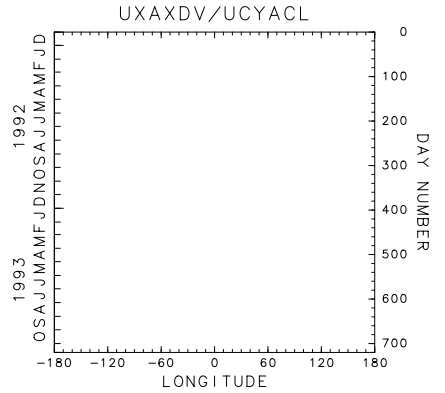
小さい目盛りを描きたくない場合には, この例の UYAXLB のように, 小さい目盛りの配列を与えるべきところに適当な変数名 (ここでは DUMMY) を書いておいて, その配列の大きさを 0 と指定すれば大丈夫です.

8.2　右も左も日付軸

これまで, 同じ座標軸作画ルーチンを呼んでも, 下側と左側にはラベルを描き, 上側と右側は目盛だけでラベルは描きませんでした. 左右両方に異なったラベルの座標軸を描きたい時にはどうすれば良いのでしょう (UXYZ6).

座標軸のラベルを描くか描かないかは, UZPACK の管理する内部変数によって制御されています. この例のように, 右側の座標軸に関しては 'UZLSET' ルーチンで 'LABELYR' という内部変数を .TRUE. にすることによって, 右側の座標軸についてもラベルが描けます. 左側は 'LABELYL', x 軸の上と下はそれぞれ, 'LABELXT', 'LABELXB' によって制御されています.

```
1         PROGRAM UXYZ6
2
3         PARAMETER( ID0=19911201, ND=720, RND=ND )
4
5         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
6         CALL SGPWSN
7         READ (*,*) IWS
8
9         CALL GROPN( IWS )
10        CALL UZFACT( 0.8 )
11        CALL GRFRM
12
13        CALL GRSWND( -180., 180., RND, 0.0 )
14        CALL GRSVPT(   0.2,  0.8, 0.2, 0.8 )
15        CALL GRSTRN( 1 )
16        CALL GRSTRF
17
18        CALL UXAXDV( 'B', 10., 60. )
19        CALL UXAXDV( 'T', 10., 60. )
20        CALL UXSTTL( 'B', 'LONGITUDE', 0. )
21
22        CALL UCYACL( 'L', ID0, ND )
23
24        CALL UZLSET( 'LABELYR', .TRUE. )
25        CALL UYAXDV( 'R', 20., 100. )
26        CALL UZISET( 'IROTCYR', -1 )
27        CALL UYSTTL( 'R', 'DAY NUMBER', 0. )
28
29        CALL UXMTTL( 'T', 'UXAXDV/UCYACL', 0. )
30
31        CALL GRCLS
32
33        END
```



uxyz6.f: frame1

右側の座標軸は UYAXDV によって描きました. 13 行めで UYMIN が RND(=720), UYMAX が 0.0 とウインドウを指定したので, ラベルの値が上から下に増えています. また, UZISET ルーチンで内部変数 'IROTCYR' を −1 にしたので, UYSTTL で描いた 'DAY NUMBER' という座標軸のタイトルも 180 度回転しています. この回転角の値は, 90 度を単位とする整数値で指定します.

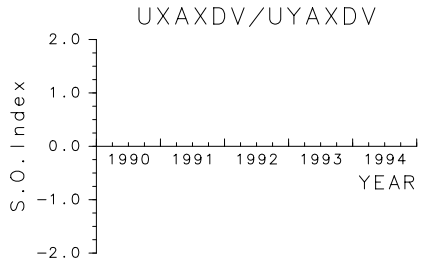
左側の座標軸は UCYACL ルーチンで描いたのですが, この例からわかるように, 日数のラベルが描けないような場合には, 適当に判断して月と年の目盛だけが描かれます. なお, UCXACL/UCYACL の下位ルーチンを呼ぶことによって, 日, 月, 年の座標軸を別々に描くこともできます.

なお, この例では 10 行めで UZFACT を呼んで, 文字や目盛のサイズを全体的に小さく (デフォルトの 0.8 倍) して座標軸を描いています.

8.3　好みの場所に好みの軸を

座標軸は 'B', 'T', 'L', 'R' で指定する以外のところにも描くことができます. また, 軸の表現もいろいろ変えられます (UXYZ7).

```
1         PROGRAM UXYZ7
2
3         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
4         CALL SGPWSN
5         READ (*,*) IWS
6
7         CALL GROPN( IWS )
8         CALL GRFRM
9
10        CALL GRSWND( 1990., 1995., -2.0, 2.0 )
11        CALL GRSVPT(   0.2,   0.8,  0.3, 0.7 )
12        CALL GRSTRN( 1 )
13        CALL GRSTRF
14
15        CALL UZRSET( 'UYUSER', 0. )
16        CALL UZLSET( 'LBTWN', .TRUE. )
17        CALL UXSFMT( '(I4)' )
18        CALL UXAXDV( 'U', 0.25, 1. )
19        CALL UZLSET( 'LBTWN', .FALSE. )
20        CALL UXSTTL( 'U', 'YEAR', 1. )
21
22        CALL UZISET( 'INNER', -1 )
23        CALL UYSFMT( '(F4.1)' )
24        CALL UYAXDV( 'L', 0.25, 1. )
25        CALL UYSTTL( 'L', 'S.O.Index', 0. )
26
27        CALL UXMTTL( 'T', 'UXAXDV/UYAXDV', 0. )
28
29        CALL GRCLS
30
31        END
```



uxyz7.f: frame1

座標軸をウインドウ ・ビューポートの境界線以外のところへ描きたいときには, 場所をあらわす引数としてユーザー指定の座標軸 'U' を指定します. この例では UXAXDV と UXSTTL のサブルーチンで 'U' を指定して, 座標軸とタイトルを描いています. 具体的にどこに軸を描くかは, UZRSET ルーチンを用いて, x 軸については内部変数 'UYUSER' の値を, y軸については内部変数 'UXUSER' の値を指定ます. この例の場合, 'UYUSER' を 0.0 と指定することによ って, U-座標系でみた y 座標の値が 0.0 のところにユーザー指定の x 軸を描きます.

また, この例の x 軸では, 目盛と目盛の間にラベルを描いてみました. UZLSET ルーチンで内部変数 'LBTWN' を .TRUE. とすれば, 目盛と目盛の間にラベルを描きます. ただし, 注意すべきことは, ほかの座標軸を描くときにもこの設定が影響しますので, ラベルを描き終ったら .FALSE.(初期値) に戻しておく必要があります.

さらに, 17 行めの UXSFMT ルーチンによってフォーマットを '(I4)' と陽に指定しています. これは, デフォルトでは有効数字 3 桁で数値を表現するフォーマットとなっているためで, 4 桁目までちゃんと表現する必要があるときには, このようにユーザーがフォーマットを指定しなければなりません.

y軸では, UZISET で内部変数 'INNER' を-1 に設定して, 目盛を外向きにつけました. (初期値は 1 で, 内向きに目盛をつけます.) また, UYSFMT ルーチンによってフォーマットを '(F4.1)' と陽に指定しました.

8.4　同じ側に何本もの軸を

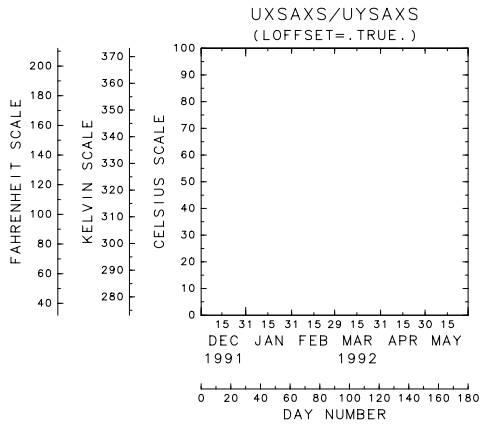
最後のプログラム UXYZ8 では, 同じ側に 2 本以上の座標軸を別の目盛で描こうという時の実例です.

軸を外側へずらしたい, または, すでに描いた軸の外側にもう一本軸を描きたいというときは, x 軸については UXSAXS ルーチンを, y軸については UYSAXS ルーチンを呼ぶだけです. 引数は, 場所を指定するおなじみの引数です. これらのルーチンを呼ぶと, 次の軸は内側の軸と重ならない程度に適度に外側に描かれます. これを何回も使えば, 簡単にいくつもの軸を一つの側に描かせることができます.

1 つの側に複数の軸を描こうとする時, そのためにいちいちウインドウを設定し直す必要はありません. この例では, y座標のウインドウとしてセ氏温度で 0 度から 100 度の範囲で設定したのですが, 換算の便のためにケルビンやカ氏の目盛りもあわせて描いています. ウインドウ設定に影響を与えず, 目盛りだけを変えて複数の座標軸を描きたいときは, オフセット機能を用います. そのためにまず UZLSET ルーチンで内部変数 'LOFFSET' を .TRUE. にしておきます. そして, 必要なところで, UZRSET ルーチンで内部変数 'YOFFSET' と 'YFACT' (x 軸については 'XOFFSET' と 'XFACT') を設定すると, 目盛の位置が 'YOFFSET' だけ平行移動し, 目盛の間隔が 'YFACT' 倍されます.

```
1         PROGRAM UXYZ8
2
3         PARAMETER( ID0=19911201, ND=180, RND=ND )
4
5         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
6         CALL SGPWSN
7         READ (*,*) IWS
8
9         CALL GROPN( IWS )
10        CALL UZFACT( 0.7 )
11        CALL UZLSET( 'LOFFSET', .TRUE. )
12        CALL GRFRM
13
14        CALL GRSWND( 0.0, RND, 0.0, 100. )
15        CALL GRSVPT( 0.4, 0.9, 0.3,  0.8 )
16        CALL GRSTRN( 1 )
17        CALL GRSTRF
18
19        CALL UCXACL( 'B', ID0, ND )
20        CALL UCXACL( 'T', ID0, ND )
21        CALL UXSAXS( 'B' )
22        CALL UXAXDV( 'B', 10., 20. )
23        CALL UXSTTL( 'B', 'DAY NUMBER', 0. )
24
25        CALL UYAXDV( 'L', 5., 10. )
26        CALL UYAXDV( 'R', 5., 10. )
27        CALL UYSTTL( 'L', 'CELSIUS SCALE', 0. )
28
29        CALL UYSAXS( 'L' )
30        CALL UZRSET( 'YOFFSET', 273.15 )
```

```
31          CALL UZRSET( 'YFACT  ', 1. )
32          CALL UYAXDV( 'L', 5., 10. )
33          CALL UYSTTL( 'L', 'KELVIN SCALE', 0. )
34
35          CALL UYSAXS( 'L' )
36          CALL UZRSET( 'YOFFSET', 32. )
37          CALL UZRSET( 'YFACT  ', 1.8 )
38          CALL UYAXDV( 'L', 10., 20. )
39          CALL UYSTTL( 'L', 'FAHRENHEIT SCALE', 0. )
40
41          CALL UXSTTL( 'T', '(LOFFSET=.TRUE.)', 0. )
42          CALL UXMTTL( 'T', 'UXSAXS/UYSAXS', 0. )
43
44          CALL GRCLS
45
46          END
47
```



uxyz8.f: frame1

計算機のなまり 4

ファイルの構造

ファイルの形式も機種によりかなり異なります. FORTRAN 規格におけるファイルの属性は, 基本的に, 書式のありなし (FORMATTED/UNFORMATTED) とアクセス方法 (SEQUENTIAL/DIRECT) の組み合せで決ります. 書式のありなしは, 例えば WRITE 文を実行するときに, 内部表現を文字に変換するか, 内部表現をそのまま出力するかということを指定するものであり, いわば 「ファイルの中身」を指定するものです. 一方, アクセス方法の指定は文字どおり解釈すれば, 順番に読み書きするか, ランダムに読み書きするかという指定ですが, 実際には 「アクセス方法」よりも, そのアクセス方法を実現する 「ファイルの構造」の方が問題となります.

規格上, DIRECT ファイルではレコード長を指定する RECL 指定子を書かなければならず, SEQUENTIAL ファイルの場合には, これを書いてはいけません. つまり, 「アクセス方法」の指定は, DIRECT ファイルは固定長レコードのファイルであり, SEQUENTIAL ファイルは可変長レコードのファイルであるという 「ファイル構造」を暗黙のうちに指定することになるのです.

FORTRAN プログラム上の論理的なレコードが, 実際に記録媒体の上でどのように記録されるかということに関しても, 大きく分けてメインフレーム系 (IBM 系) と UNIX 系の 2 種類あり, それぞれファイルの扱い方がかなり異なります.

詳細は, MISC1 のマニュアル (dcl-x.x/doc/misc1/gaiyou/file.tex) を御覧下さい.

第 9 章

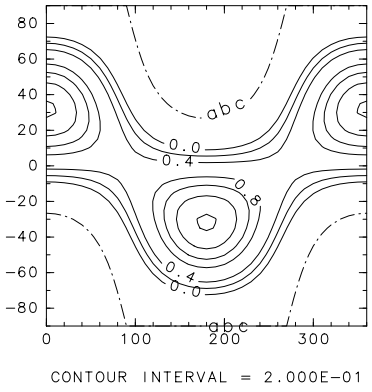
2 次元量の表示

第 2章で見たように, 2 次元データを手早くコンタリングしたいというときには, サブルーチン UDCNTR を呼ぶだけで, とりあえず等高線図が得られました. これは, GRPH2 の 2 次元等高線図描画ルーチンパッケージ UDPACK に含まれるサブルーチンです. ここでは, UDPACK のいくつかの機能を簡単なプログラム例で示します.
なお, UDPACK は U-座標系で作画していますから, 第 13章で簡単にふれる地図投影にも対応できます.

9.1　コンターラインをコントロール

第 2.2節で扱ったデータを用いて, コンターラインをユーザーの好みで制御することを考えてみましょう (U2D1).

```
1         PROGRAM U2D1
2
3         PARAMETER( NX=37, NY=37 )
4         PARAMETER( XMIN=0, XMAX=360, YMIN=-90, YMAX=90 )
5         PARAMETER( PI=3.14159, DRAD=PI/180 )
6         REAL P(NX,NY)
7
8         DO 10 J=1,NY
9         DO 10 I=1,NX
10          ALON = ( XMIN + (XMAX-XMIN)*(I-1)/(NX-1) ) * DRAD
11          ALAT = ( YMIN + (YMAX-YMIN)*(J-1)/(NY-1) ) * DRAD
12          SLAT = SIN(ALAT)
13          P(I,J) = 3*SQRT(1-SLAT**2)*SLAT*COS(ALON) - 0.5*(3*SLAT**2-1)
14      10 CONTINUE
15
16        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
17        CALL SGPWSN
18        READ (*,*) IWS
19
20        CALL GROPN( IWS )
21        CALL GRFRM
22
23        CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
24        CALL GRSVPT(  0.2,  0.8,  0.2,  0.8 )
25        CALL GRSTRN( 1 )
26        CALL GRSTRF
27
28        CALL USDAXS
29        CALL UDSFMT( '(F4.1)' )
30        CALL UDGCLA( 0., 1.4, 0.2 )
31        CALL UDSCLV( -1., 3, 4, 'abc', 0.028 )
32        CALL UDDCLV( 0.6 )
33        CALL UDCNTR( P, NX, NX, NY )
34
35        CALL GRCLS
36
37        END
```



u2d1.f: frame1

デフォルトではラベル付きの太線が 1 本おきに引かれますが, これをメジャーラインとよびます. このメジャーラインに付いているコンターラベルのフォーマットは, UDSFMT ルーチンで設定できます. 29 行めのように, 指定するフォーマットを文字型で与えます. コンターラインにつけるラベルは, コンターレベルを決定するルーチンの中で生成されますから, UDSFMT ルーチンはコンターラインを生成するルーチン (UDGCLA や UDGCLB) の前に呼ばなければなりません.

コンターラインの生成ですが, ここでは, UDGCLA ルーチンで等間隔のコンターレベルを生成します. 最初の 2 つの引数がコンターレベルの最小値と最大値で, 30 行めの例では負のコンターは描かれません. 最後の引数は刻み幅ですが, これが負の時にはその絶対値程度の本数のコンターレベルを生成します.

これらのルーチンでは, 等間隔のコンターレベルしか生成されません. もしも, 不等間隔のコンターレベルを指定したい場合や, 特定のコンターレベルを追加したい場合は, 31 行めのように UDSCLV ルーチンで 1 本 1 本のコンターレベルを生成します. 最初の引数がコンターレベルの値で, 残りの 4 つがコンターラインの属性です. 順に, ラインインデクス, ラインタイプ, ラベル, V-座標系におけるラベルの大きさ, です. 逆に, ある 1 本のコンターレベルを無効にするには UDDCLV ルーチンを呼びます. 引数はコンターレベルの値で, この例では, 0.6 の等高線を描いていません. また, すべてのコンターレベルを無効にするには UDICLV ルーチンを呼びます.

あとは, 同じ UDCNTR ルーチンを呼ぶだけです. このプログラムを実行すると, 次のようなメッセージが出るはずです.

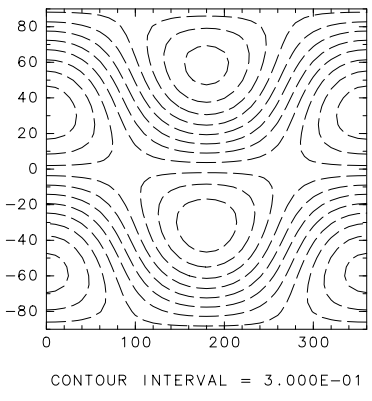
```
*** MESSAGE (UDCNTR) *** INAPPROPRIATE DATA WILL BE MODIFIED INTERNALLY.
** MESSAGE (-CNT.-) *** Z( 1, 1)= -1.00000000 ===> -1.00000119
```

UDCNTR では, ある格子点での値がコンターレベルと等しい時に, 格子点値をわずかにずらして作画し, このようなメッセージを出力します. たいていの場合, このメッセージを気にする必要はありません.

もう少しコンターラインにこだわってみましょう. 次のプログラム U2D2 では, 内部変数を変更することにより, コンターラインをコントロールしています. UDPACK サブルーチンパッケージで使用する内部変数の参照と変更は, UDpGET と UDpSET のルーチンで行ないます. ここで, p は, 内部変数の型によって L(論

理型), I(整数型), R(実数型) のどれかをとります. この例では, 内部変数 'LABEL' を .FALSE. にして, メジャーラインにラベルを付けないようにします. 'LDASH' も .FALSE. にして正も負も同じラインタイプでコンターラインを描くことにします. また, その時のラインタイプを 'ISOLID' で 2(破線) と指定しています. さらに, 'ICYCLE' を 4 として, 4 本に 1 本の割合でメジャーラインを引くことにします. 結果は, 正負の区別もつかず, わけのわからない等高線図となってしまいましたが, このような機能もあるというデモンストレーションです.

```
1         PROGRAM U2D2
2
3         PARAMETER( NX=37, NY=37 )
4         PARAMETER( XMIN=0, XMAX=360, YMIN=-90, YMAX=90 )
5         PARAMETER( PI=3.14159, DRAD=PI/180 )
6         REAL P(NX,NY)
7
8         DO 10 J=1,NY
9         DO 10 I=1,NX
10          ALON = ( XMIN + (XMAX-XMIN)*(I-1)/(NX-1) ) * DRAD
11          ALAT = ( YMIN + (YMAX-YMIN)*(J-1)/(NY-1) ) * DRAD
12          SLAT = SIN(ALAT)
13          P(I,J) = 3*SQRT(1-SLAT**2)*SLAT*COS(ALON) - 0.5*(3*SLAT**2-1)
14      10 CONTINUE
15
16        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
17        CALL SGPWSN
18        READ (*,*) IWS
19
20        CALL GROPN( IWS )
21        CALL GRFRM
22
23        CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
24        CALL GRSVPT(  0.2,  0.8,  0.2,  0.8 )
25        CALL GRSTRN( 1 )
26        CALL GRSTRF
27
28        CALL USDAXS
29        CALL UDLSET( 'LABEL', .FALSE. )
30        CALL UDLSET( 'LDASH', .FALSE. )
31        CALL UDISET( 'ISOLID', 2 )
32        CALL UDISET( 'ICYCLE', 4 )
33        CALL UDCNTR( P, NX, NX, NY )
34
35        CALL GRCLS
36
37        END
```



u2d2.f: frame1

9.2　格子点が不等間隔の場合

UWPACK は 2 次元の格子点座標に関する情報を管理するサブルーチンパッケージです. UDCNTR ルーチンを呼ぶ前に UWPACK のサブルーチンを呼んで格子点の座標値をあらかじめ指定しておけば, 不等間隔な格子点でもコンタリングが可能です (U2D3).

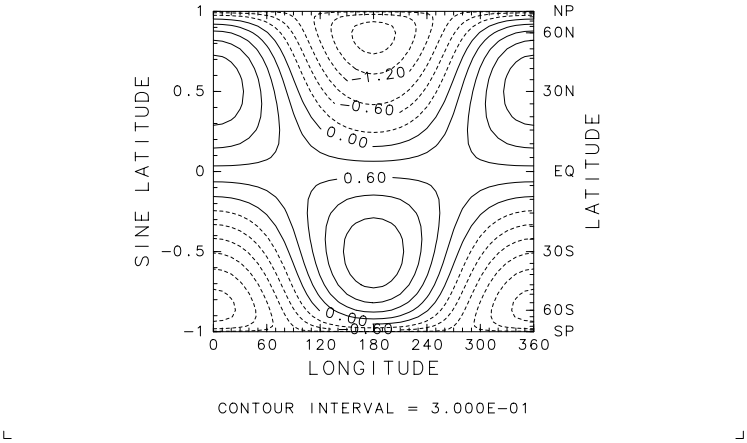
```
      PROGRAM U2D3

      PARAMETER( NX=37, NY=37, MY=7 )
      PARAMETER( XMIN=0, XMAX=360, YMIN=-90, YMAX=90 )
      PARAMETER( PI=3.14159, DRAD=PI/180 )
      REAL      P(NX,NY), UY1(NY), UY2(MY)
      CHARACTER CH(MY)*3

      DATA      CH/ 'SP ', '60S', '30S', 'EQ ', '30N', '60N', 'NP ' /

      DO 10 J=1,NY
      DO 10 I=1,NX
        ALON = ( XMIN + (XMAX-XMIN)*(I-1)/(NX-1) ) * DRAD
        ALAT = ( YMIN + (YMAX-YMIN)*(J-1)/(NY-1) ) * DRAD
        SLAT = SIN(ALAT)
        UY1(J) = SLAT
        P(I,J) = 3*SQRT(1-SLAT**2)*SLAT*COS(ALON) - 0.5*(3*SLAT**2-1)
   10 CONTINUE

      DO 20 J=1,MY
        UY2(J) = SIN( ( YMIN + (YMAX-YMIN)*(J-1)/(MY-1) ) * DRAD )
   20 CONTINUE

      WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
      CALL SGPWSN
      READ (*,*) IWS

      CALL GROPN( IWS )
      CALL GRFRM

      CALL GRSWND( XMIN, XMAX, -1.,  1. )
      CALL GRSVPT(  0.2,  0.8, 0.2, 0.8 )
      CALL GRSTRN( 1 )
      CALL GRSTRF

      CALL UXAXDV( 'B', 10., 60. )
      CALL UXAXDV( 'T', 10., 60. )
      CALL UXSTTL( 'B', 'LONGITUDE', 0. )

      CALL UYAXDV( 'L', 0.1, 0.5 )
      CALL UYSTTL( 'L', 'SINE LATITUDE', 0. )

      CALL UZLSET( 'LABELYR', .TRUE. )
      CALL UYAXLB( 'R', UY1, NY, UY2, CH, 3, MY )
      CALL UYSTTL( 'R', 'LATITUDE', 0. )

      CALL UWSGXB( XMIN, XMAX, NX )
      CALL UWSGYA( UY1, NY )

      CALL UDCNTR( P, NX, NX, NY )

      CALL GRCLS

      END
```

この例では, これまでと同じ 2 次元配列データですが, y方向にはサイン緯度 (-1 から 1 の値を取る) の座標系を設定してコンタリングをおこないます. まず, 配列 UY1 に格子点の座標を U-座標系の値で用意します. UY2 は, UYAXLB ルーチンで y座標軸のラベルを描くために, その座標値を用意する配列です. 40 行めでは UYAXDV ルーチンでサイン緯度の座標軸を等間隔に刻んで描きます. 次に, 右側の y座標軸にラベルが描けるようにして, UYAXLB ルーチンでデータのある格子点に目盛をつけ, 緯度 30 度毎にラベルをつけます.

次に, x 方向については, UWSGXB ルーチンを用いて, 最小値, 最大値および格子点数を指定することによって, 格子点座標に関する情報を設定します. ただし, ここでは格子点をウインドウいっぱいに設定するようにしているだけなので, UWSGXB ルーチンを呼ばなくても結果は同じです. そして, UWSGYA ルーチンを用い

て不等間隔の格子点を設定しています. 引数は, 座標値を指定する配列名 UY1 とその長さです. x 軸に不等間隔の格子点を設定するには, UWSGXA ルーチンを用います.

あとは, UDCNTR ルーチンを呼ぶと, 格子点が不等間隔の場合でも御覧のように等高線図が描かれます.



u2d3.f: frame1

9.3　配列の一部分だけを描く

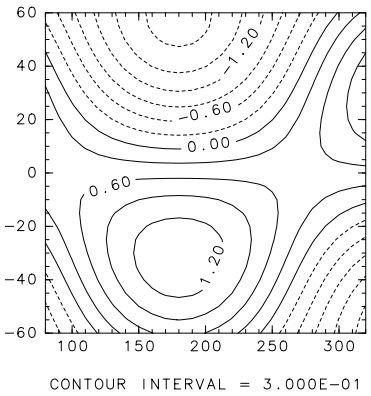
配列の一部分だけを作画するにはどうしたらよいのでしょう? これまでと同じデータで, その一部分だけを描くことを考えてみましょう (U2D4).

経度 $80°$ から $320°$, 緯度 $-60°$ から $60°$ の部分だけを描くことにします. 6 行めのパラメータ文で与えているように, 格子点の範囲は x 方向に $9 \leq i \leq 33$, y方向に $7 \leq j \leq 31$ となります. ウインドウの設定をこの範囲とし, 36 行めの UDCNTR ルーチンで等高線図を描くのですが, ここで一工夫が必要です. 最初の引数としては, この矩形領域の左下に対応する配列要素 P(KXMN,KYMN) を陽に与え, 2 番めには実際に宣言した配列の第 1 次元寸法 NX, 3, 4 番めには実際の作画に使う配列の第 1 次元および第 2 次元寸法 (KXMX-KXMN+1,KYMX-KYMN+1) を与えることによって配列の一部分だけを作画することができます.

この辺の事情を正しく理解するためには, FORTRAN において, 2 次元の配列要素が記憶領域上でどのように順序づけられ, サブルーチンの引数にある配列がどのように扱われているかを知る必要があります. FORTRAN マニュアルなどを参照してください.

```
1         PROGRAM U2D4
2
3         PARAMETER( NX=37, NY=37 )
4         PARAMETER( XMIN=0, XMAX=360, YMIN=-90, YMAX=90 )
5         PARAMETER( PI=3.14159, DRAD=PI/180 )
6         PARAMETER( KXMN=9, KXMX=33, KYMN=7, KYMX=31 )
7         REAL P(NX,NY), ALON(NX), ALAT(NY)
8
9          DO 10 I=1,NX
10           ALON(I) = XMIN + (XMAX-XMIN)*(I-1)/(NX-1)
11      10 CONTINUE
12         DO 20 J=1,NY
13           ALAT(J) = YMIN + (YMAX-YMIN)*(J-1)/(NY-1)
14      20 CONTINUE
15
```

```
16          DO 30 J=1,NY
17            SLAT = SIN(ALAT(J)*DRAD)
18          DO 30 I=1,NX
19            P(I,J) = 3*SQRT(1-SLAT**2)*SLAT*COS(ALON(I)*DRAD)
20         +         - 0.5*(3*SLAT**2-1)
21       30 CONTINUE
22
23          WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
24          CALL SGPWSN
25          READ (*,*) IWS
26
27          CALL GROPN( IWS )
28          CALL GRFRM
29
30          CALL GRSWND( ALON(KXMN), ALON(KXMX), ALAT(KYMN), ALAT(KYMX) )
31          CALL GRSVPT(        0.2,        0.8,        0.2,        0.8 )
32          CALL GRSTRN( 1 )
33          CALL GRSTRF
34
35          CALL USDAXS
36          CALL UDCNTR( P(KXMN,KYMN), NX, KXMX-KXMN+1, KYMX-KYMN+1 )
37
38          CALL GRCLS
39
40          END
```



u2d4.f: frame1

FORTRAN のひけつ 2

多次元配列の記憶順序

FORTRAN77 では多次元 (7 次元まで) の配列を定義できますが, 実際の記憶装置が多次元構造を持っているわけではなくて, 1 次元的に管理されています. 多次元配列を 1 次元的に展開する順序は, FORTRAN77 の規格で定めてありますから, 多次元配列を 1 次元配列と結合することが可能です.

例えば

```
REAL X(6), Y(2,3)
COMPLEX Z(3)
EQUIVALENCE (X,Y,Z)
```

という場合, 変数 X, Y, Z はどれも長さが 6 語の配列で, EQUIVALENCE 文により同じ記憶領域を占めます. この時, それぞれの変数の並び方は

X(1)	X(2)	X(3)	X(4)	X(5)	X(6)
Y(1,1)	Y(2,1)	Y(1,2)	Y(2,2)	Y(1,3)	Y(2,3)
Re(Z(1))	Im(Z(1))	Re(Z(2))	Im(Z(2))	Re(Z(3))	Im(Z(3))

という様に, 2 次元配列 Y は添字の左側の数字から変るように 1 次元的に展開されます. また, 複素数データ Z は 2 つの実数が並んだ形で展開されます. したがって, X(3) の数値は, Y(1,2), Re(Z(2)) と同じになります.

この規則は多くのプログラムで積極的に使われており, 多次元配列を 1 次元配列として扱ったり, 複素数データを実数データと見なして処理したりすることが行われています. この規則は, 計算機のハードウェアに近い部分の規則なので, 計算機によって異なるように思えるかもしれませんが, FORTRAN77 規格で定められた 「標準語」です. 例えば, 「岩波 FORTRAN 辞典」(西村恕彦/酒井俊夫・高田正之 著) の 「記憶列の結合」を御覧下さい.

第 10 章

もっと 2 次元量表示

ここでは UEPACK と UGPACK の進んだ使い方を見てみましょう.

UEPACK は U-座標系で作画しているので, 後で述べる地図投影にも対応できますが, UGPACK は V-座標系で作画しているので地図投影には対応していません .

10.1　トーンパターンを指定する

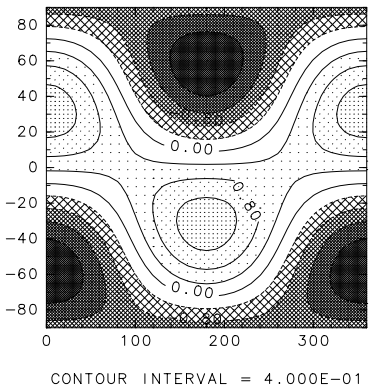
第 2.2節の例のように, ただ負の領域に斜線のハッチをつけるだけでは芸がありません. UEPACK を活用すれば, トーンをつけるレベルやパターンを指定することができるようになります (U2D5).

トーンの指定は, 必要なレベルの分だけ UESTLV ルーチンを呼ぶことによっておこないます (37 行め). 最初の 2 つの引数によってぬりわけるレベルの下限値と上限値を指定し, 最後の引数でトーンパターン番号を指定します. トーンパターンは巻末付録を御覧下さい. UESTLN というルーチンを用いれば, 複数の塗り分けるレベルとパターンを配列で一度に指定することもできます. また, UDGCLA, UDGCLB ルーチンと同様のトーンレベル生成ルーチン UEGTLA, UEGTLB もあります.

この例も, 第 2.2節と同じようにコンターとの重ね書きをおこなっていますが, プログラム構成で異なるところがあります. それは, SGLSET ルーチンで内部変数 'LSOFTF' を .TRUE. として, ソフトフィルを指定しているために, 座標軸の描画やコンタリングのあとで UETONE ルーチンを呼んでも問題ないということです. UEPACK のハッチングは SGPACK のトーンプリミティブを用いていますが, トーンプリミティブは出力装置の能力に応じてハードフィルとソフトフィルとを切替えることができます. ハードフィルによるぬりわけを行なう場合, 出力装置によっては先に描かれた図形が消えてしまうことがあるので, QUICK4 のプログラム例では, UETONE を最初に呼んでいるわけです. しかし, この例ではソフトフィルを指定したので, 描く順番を気にしなくてもよいわけです.

```
1         PROGRAM U2D5
2
3         PARAMETER( NX=37, NY=37 )
4         PARAMETER( XMIN=0, XMAX=360, YMIN=-90, YMAX=90 )
5         PARAMETER( PI=3.14159, DRAD=PI/180 )
6         REAL P(NX,NY)
7
8         DO 10 J=1,NY
9         DO 10 I=1,NX
10          ALON = ( XMIN + (XMAX-XMIN)*(I-1)/(NX-1) ) * DRAD
11          ALAT = ( YMIN + (YMAX-YMIN)*(J-1)/(NY-1) ) * DRAD
12          SLAT = SIN(ALAT)
13          P(I,J) = 3*SQRT(1-SLAT**2)*SLAT*COS(ALON) - 0.5*(3*SLAT**2-1)
```

```
14      10 CONTINUE
15
16         WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
17         CALL SGPWSN
18         READ (*,*) IWS
19
20         CALL GROPN( IWS )
21         CALL SGLSET( 'LSOFTF', .TRUE. )
22         CALL GRFRM
23
24         CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
25         CALL GRSVPT(  0.2,  0.8,  0.2,  0.8 )
26         CALL GRSTRN( 1 )
27         CALL GRSTRF
28
29         DO 20 K=-5,3
30           TLEV1 = 0.4*K
31           TLEV2 = TLEV1 + 0.4
32           IF(K.LE.-1) THEN
33             IPAT = 600 + ABS(K+1)
34           ELSE
35             IPAT =  30 + K
36           END IF
37           CALL UESTLV( TLEV1, TLEV2, IPAT )
38      20 CONTINUE
39
40         CALL USDAXS
41         CALL UDGCLB( P, NX, NX, NY, 0.4 )
42         CALL UDCNTR( P, NX, NX, NY )
43         CALL UETONE( P, NX, NX, NY )
44
45         CALL GRCLS
46
47         END
```



u2d5.f: frame1

10.2　ベクトル場のスケーリング

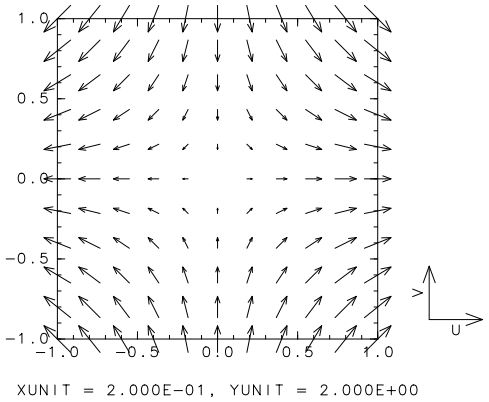
2 次元ベクトル場を描くサブルーチン UGVECT では, ベクトルの各成分を V-座標系における単位で表現しています. ただし実際には, V-座標系の単位に変換した配列を与える必要はなく, スケーリングファクターを与えてやれば十分です. このスケーリングファクターは, 内部変数 'LNRMAL' が .TRUE. (初期値) ならば内部的に決定され, .FALSE. ならば内部変数 'XFACT1', 'YFACT1' を参照します. したがって, このデモプログラム (U2D6) のように, スケーリングファクターを陽に指定する場合は, UGLSET ルーチンで 'LNRMAL' を .FALSE. として UGRSET で 'XFACT1' と 'YFACT1' を設定します. ここでは, U と V の値が 1 桁違いますから, XFACT1=0.5, YFACT1=0.05 としました. たとえば, ベクトル (U,V)=(0.1,1) は, V- 座標系単位で (0.05, 0.05) の大きさのベクトルとして表示されます. つまり, スケーリングファクターとは, 次元量と

して与えられたベクトルの成分を V-座標系の単位で表現される矢印の成分に変換するための比例係数なのです.

またこの例では, 内部変数 'LUNIT' を .TRUE. としてユニットベクトルを描いています. (この初期値は .FALSE. で, ユニットベクトルを描きません.) 表示するユニットベクトルの大きさは, V-座標系の単位を用いて内部変数 'VXUNIT', 'VYUNIT' をそれぞれ 0.1 に設定しています. もしも次元量で指定したいならば, 内部変数 'UXUNIT', 'UYUNIT' を設定することでユニットベクトルを描くこともできます. UGSUT ルーチンを用いると, このユニットベクトルにタイトルをつけることができます. x 成分は 'U', y成分は 'V' としました. なお, 図の下部に表示されているユニットベクトルの大きさ XUNIT, YUNIT は次元量です. この例のように, ユニットベクトルの大きさを V-座標系の単位で指定した場合は, それぞれをスケーリングファクターで割った値となります.

UGPACK が図の下に書くメッセージについてですが, ユニットベクトルを描かないときにはスケーリングファクターが表示され, ユニットベクトルを描くときにはユニットベクトルの大きさが表示されます. ユニットベクトルを描くときにも, 内部変数 'LUMSG' を .FALSE. とすることによってスケーリングファクターの表示に切替えることもできます. また, メッセージを何も描かせたくないときには, 内部変数 'LMSG' を .FALSE. とします.

```
1         PROGRAM U2D6
2
3         PARAMETER( NX=11, NY=11 )
4         PARAMETER( XMIN=-1, XMAX=1, YMIN=-1, YMAX=1 )
5         REAL U(NX,NY), V(NX,NY)
6
7         DO 10 J=1,NY
8         DO 10 I=1,NX
9           X = XMIN + (XMAX-XMIN)*(I-1)/(NX-1)
10          Y = YMIN + (YMAX-YMIN)*(J-1)/(NY-1)
11          U(I,J) =  X * 0.1
12          V(I,J) = -Y
13      10 CONTINUE
14
15        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
16        CALL SGPWSN
17        READ (*,*) IWS
18
19        CALL GROPN( IWS )
20        CALL GRFRM
21
22        CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
23        CALL GRSVPT(  0.2,  0.8,  0.2,  0.8 )
24        CALL GRSTRN( 1 )
25        CALL GRSTRF
26
27        CALL USDAXS
28
29        CALL UGLSET( 'LNRMAL', .FALSE. )
30        CALL UGRSET( 'XFACT1', 0.5 )
31        CALL UGRSET( 'YFACT1', 0.05 )
32        CALL UGLSET( 'LUNIT' , .TRUE. )
33        CALL UGRSET( 'VXUNIT', 0.1 )
34        CALL UGRSET( 'VYUNIT', 0.1 )
35        CALL UGRSET( 'VXUOFF', 0.06 )
36        CALL UGSUT( 'X', 'U' )
37        CALL UGSUT( 'Y', 'V' )
38        CALL UGVECT( U, NX, V, NX, NX, NY )
39
40        CALL GRCLS
41
42        END
```



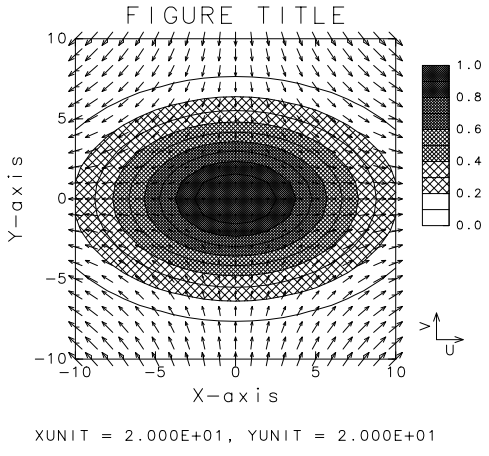
u2d6.f: frame1

10.3　等高線とベクトル場の重ね書き

2 次元量表示の最後のプログラム例 U2D7 は, 等高線とベクトル場の重ね書きです. これまでに見てきた内容の描画を 1 つのフレームにまとめたものです. 'UDLSET' ルーチンで等高線図に関する内部変数 'LABEL' と 'LMSG' を .FALSE. にして, ラベルとメッセージを描かないように設定し, そのかわり, 右側にトーンバーをつけるようにしました. 61 行め以降で, もう一度正規化変換を行ない, UDCNTR と UETONE ルーチンでトーンバーを描いています.

```
1         PROGRAM U2D7
2
3         PARAMETER( NX=21, NY=21 )
4         PARAMETER( XMIN=-10, XMAX=10, YMIN=-10, YMAX=10 )
5         PARAMETER( DX1=1, DX2=5, DY1=1, DY2=5 )
6         PARAMETER( KMAX=5, PMIN=0, PMAX=1 )
7         REAL U(NX,NY), V(NX,NY), P(NX,NY), PI(2,KMAX+1)
8
9         DO 10 J=1,NY
10        DO 10 I=1,NX
11          X = XMIN + (XMAX-XMIN)*(I-1)/(NX-1)
12          Y = YMIN + (YMAX-YMIN)*(J-1)/(NY-1)
13          U(I,J) =  X
14          V(I,J) = -Y
15          P(I,J) = EXP( -X**2/64 -Y**2/25 )
16      10 CONTINUE
17
18        WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
19        CALL SGPWSN
20        READ (*,*) IWS
21
22        CALL GROPN( IWS )
23        CALL SGLSET( 'LSOFTF', .TRUE. )
24        CALL GRFRM
25
26        CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
27        CALL GRSVPT(  0.2,  0.8,  0.2,  0.8 )
28        CALL GRSTRN( 1 )
29        CALL GRSTRF
30
31        CALL UXAXDV( 'B', DX1, DX2 )
32        CALL UXAXDV( 'T', DX1, DX2 )
33        CALL UXSTTL( 'B', 'X-axis', 0. )
34        CALL UXMTTL( 'T', 'FIGURE TITLE', 0. )
35
36        CALL UYAXDV( 'L', DY1, DY2 )
37        CALL UYAXDV( 'R', DY1, DY2 )
38        CALL UYSTTL( 'L', 'Y-axis', 0. )
```

```
39
40          CALL UGLSET( 'LUNIT' , .TRUE. )
41          CALL UGRSET( 'VXUOFF', 0.04 )
42          CALL UGSUT( 'X', 'U' )
43          CALL UGSUT( 'Y', 'V' )
44          CALL UGVECT( U, NX, V, NX, NX, NY )
45
46          CALL UDLSET( 'LABEL', .FALSE. )
47          CALL UDLSET( 'LMSG' , .FALSE. )
48          CALL UDGCLB( P, NX, NX, NY, 0.1 )
49          CALL UDCNTR( P, NX, NX, NY )
50
51          DP = (PMAX-PMIN)/KMAX
52          DO 20 K=1,KMAX
53            TLEV1 = (K-1)*DP
54            TLEV2 = TLEV1 + DP
55            IPAT  = 600 + K - 1
56            CALL UESTLV( TLEV1, TLEV2, IPAT )
57       20 CONTINUE
58          CALL UETONE( P, NX, NX, NY )
59
60      *-- トーンバー ----
61          CALL GRSWND(  0.0, 1.0, PMIN, PMAX )
62          CALL GRSVPT( 0.85, 0.9, 0.45, 0.75 )
63          CALL GRSTRN( 1 )
64          CALL GRSTRF
65
66          DO 30 K=1,KMAX+1
67            PI(1,K) = PMIN + (K-1)*DP
68            PI(2,K) = PMIN + (K-1)*DP
69       30 CONTINUE
70
71          CALL UDCNTR( PI, 2, 2, KMAX+1 )
72          CALL UETONE( PI, 2, 2, KMAX+1 )
73
74          CALL SLPVPR( 3 )
75          CALL UZLSET( 'LABELYR', .TRUE. )
76          CALL UZFACT( 0.8 )
77          CALL UYSFMT( '(F4.1)' )
78          CALL UYAXDV( 'R', DP, DP )
79
80          CALL GRCLS
81
82          END
```



u2d7.f: frame1

FORTRAN のひけつ 3

サブルーチンにおける引数の結合

関数引用やサブルーチンの引用で引数によってデータの結合をおこなうとき, 配列の記憶順序を意識したテクニックがよく使われます. たとえば, 第3章のプログラム KIHON2 では, 何の説明も無しにつぎのように使っています.

```
CALL SGPLV( NMAX+1, X, Y(0,1) )
・・・・・・・・・・・・・・・・・
CALL SGPLV( NMAX+1, X, Y(0,I) )
```

FORTRAN では, 配列の結合は次のように定められています :

> 仮引数が配列名である場合の実引数は, 実引数配列の記憶列のうちで仮引数と結合される部分の先頭の記憶単位を指定する.

この例のように, 実引数が配列要素名であれば, その配列要素の記憶単位から配列の最後の記憶単位までの記憶列が仮引数と結合されます.

つまり, 配列がサブルーチンに渡されるとき, メインプログラムの配列の中身がサブルーチンの配列の中身にコピーされるのではなくて, メインプログラムの中の配列の番地だけが渡されて, サブルーチンの方はその番地を基準にしてデータ列の処理を行なうのです.

このようなことを意識してプログラムを書くと, 少ない引数で柔軟な処理が可能になります. 詳細は,　「岩波 FORTRAN 辞典」などをお読み下さい.

第 11 章

欠損値処理いろいろ

地球流体の分野では, 測器にトラブルがあったり観測出来なかったりして, データに欠損が生じることがしばしば起こります. その時, 適当な欠損値を与えてデータの継続性を保つことが一般的に行なわれますが, そのようなデータを解析するのは結構面倒なものです. DCL には, 強力な欠損値処理機能があります. どのように欠損値が処理できるのか, この章ではグラフィクスルーチンに限って説明しましょう.

11.1 ポリライン ・ポリマーカーにおける欠損値処理

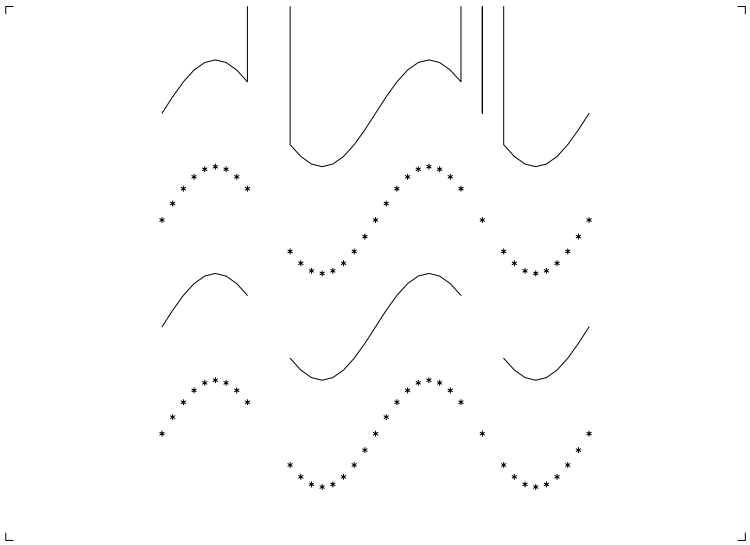
データの中に欠損がある場合には, MATH1 の SYSLIB サブルーチンパッケージにある GLLSET ルーチンで欠損値処理の指定に関する内部変数 'LMISS' を .TRUE. にして, 欠損値処理を有効にします. 欠損値処理の制御は SGPACK だけでなく DCL 全体で統一的に行われるために, 欠損値処理に関する内部変数の管理は SGpGET/SGpSET ではなく GLpGET/GLpSET で行われています (p は I, R, L または C).

このように指定すると, データの値が 999. であるものは欠損値と見なして, これから見るような欠損値処理を行ないます. 欠損値 999. が実際のデータ範囲に入っていて, 別の値にしたい場合には, GLRSET ルーチンによって実数の欠損値をあらわす内部変数 'RMISS' を変更します.

まず, MISS1 プログラムを例に, ポリラインとポリマーカーのプリミティブで欠損値処理がどのように行なわれるか見てみましょう. 第 3 章で用いた正弦関数のデータに欠損値 999. を埋め込みます. データの 1/4 のところでは 3 点連続して欠損値とし, 3/4 のところでは, 有効な点が欠損値データに挟まれて 1 点だけ孤立しているように与えます.

```
1          PROGRAM MISS1
2
3          PARAMETER( NMAX=40 )
4          PARAMETER( PI=3.14159 )
5          PARAMETER( XMIN=0., XMAX=4*PI, YMIN=-1., YMAX=1. )
6          REAL X(0:NMAX), Y(0:NMAX)
7
8          DT = XMAX/NMAX
9          DO 10 N=0,NMAX
10           X(N) = N*DT
11           Y(N) = SIN(X(N))
12       10 CONTINUE
13
14         N1 = NMAX/4
15         Y(N1-1) = 999.
16         Y(N1  ) = 999.
17         Y(N1+1) = 999.
18
19         N2 = N1*3
20         Y(N2-1) = 999.
21         Y(N2+1) = 999.
```

```
22
23          WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
24          CALL SGPWSN
25          READ (*,*) IWS
26
27          CALL GROPN( IWS )
28          CALL GRFRM
29
30    *-- 欠損値処理なし ----
31          CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
32          CALL GRSVPT(  0.1,  0.9,  0.7,  0.9 )
33          CALL GRSTRN( 1 )
34          CALL GRSTRF
35
36          CALL SGPLU( NMAX+1, X, Y )
37
38          CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
39          CALL GRSVPT(  0.1,  0.9,  0.5,  0.7 )
40          CALL GRSTRN( 1 )
41          CALL GRSTRF
42
43          CALL SGSPMT( 3 )
44          CALL SGPMU( NMAX+1, X, Y )
45
46    *-- 欠損値処理 ----
47          CALL GLLSET( 'LMISS', .TRUE. )
48
49          CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
50          CALL GRSVPT(  0.1,  0.9,  0.3,  0.5 )
51          CALL GRSTRN( 1 )
52          CALL GRSTRF
53
54          CALL SGPLU( NMAX+1, X, Y )
55
56          CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
57          CALL GRSVPT(  0.1,  0.9,  0.1,  0.3 )
58          CALL GRSTRN( 1 )
59          CALL GRSTRF
60
61          CALL SGPMU( NMAX+1, X, Y )
62
63          CALL GRCLS
64
65          END
```



miss1.f: frame1

欠損値処理をしない (デフォルト) で SGPLU と SGPMU のルーチンを呼んだ結果が上の 2 つです. 他のデータと比べて 999. は非常に大きな値ですから, ポリラインではほぼ真上に線を引いて最大作画範囲を飛び出しています. また, ポリマーカーでもこれらの点は作画範囲を飛び出してしまって描かれません.

GLLSET ルーチンで内部変数 'LMISS' を .TRUE. にして, 欠損値処理を行なって描いた結果が下の 2 つで

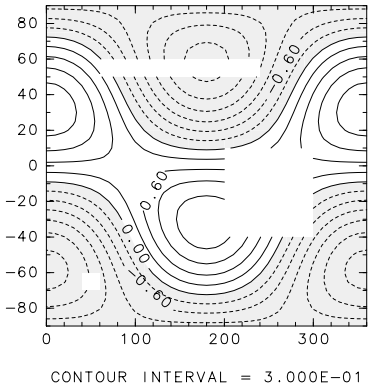
す. ポリラインでは欠損値の前後を線で結びません. 有効な点が 1 点だけ孤立している場合には, その点を結ぶことができないので, そこには何も描かれません. また, ポリマーカーでは欠損値の点にはマーカーを打ちません. この例では, ポリマーカーの結果はどちらも同じですが, 欠損値を小さくして 2. とすると違いは明らかです.

11.2　2 次元量表示における欠損値処理

格子点の値が欠損値のときでも, 欠損値処理の指定をすることによって, UDPACK や UEPACK, UGPACK にある 2 次元量表示サブルーチンでも欠損値処理をして作図できます.

プログラム MISS2 の例では, 基本的には第 2.2節の QUICK4 と同じプログラム構成で, ただし格子点値の一部分を欠損値 RMISS (この値は GLRGET ルーチンで設定されている実数欠損値を参照しました) として, 欠損値処理の指定をおこなって作画しています. 欠損値処理の指定は, やはり GLLSET ルーチンによって内部変数 'LMISS' を .TRUE. とすることによっておこないます. この実行結果からもわかるように, 欠損値のまわりの格子点を境界としてコンターやトーンが描かれません.

```
1         PROGRAM MISS2
2
3         PARAMETER( NX=37, NY=37 )
4         PARAMETER( XMIN=0, XMAX=360, YMIN=-90, YMAX=90 )
5         PARAMETER( PI=3.14159, DRAD=PI/180 )
6         REAL P(NX,NY)
7
8         CALL GLRGET( 'RMISS', RMISS )
9         CALL GLLSET( 'LMISS', .TRUE. )
10
11        DO 10 J=1,NY
12        DO 10 I=1,NX
13          ALON = ( XMIN + (XMAX-XMIN)*(I-1)/(NX-1) ) * DRAD
14          ALAT = ( YMIN + (YMAX-YMIN)*(J-1)/(NY-1) ) * DRAD
15          SLAT = SIN(ALAT)
16          P(I,J) = 3*SQRT(1-SLAT**2)*SLAT*COS(ALON) - 0.5*(3*SLAT**2-1)
17          IF( I.EQ.6 .AND. J.EQ.6 ) THEN
18            P(I,J) = RMISS
19          END IF
20          IF( ( 8.LE.I .AND. I.LE.24) .AND. J.EQ.30 ) THEN
21            P(I,J) = RMISS
22          END IF
23          IF( (22.LE.I .AND. I.LE.30) .AND.
24       +      (12.LE.J .AND. J.LE.20)       ) THEN
25            P(I,J) = RMISS
26          END IF
27       10 CONTINUE
28
29          WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
30          CALL SGPWSN
31          READ (*,*) IWS
32
33          CALL GROPN( IWS )
34          CALL GRFRM
35
36          CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
37          CALL GRSVPT(  0.2,  0.8,  0.2,  0.8 )
38          CALL GRSTRN( 1 )
39          CALL GRSTRF
40
41          CALL UETONE( P, NX, NX, NY )
42          CALL USDAXS
43          CALL UDCNTR( P, NX, NX, NY )
44
45          CALL GRCLS
46
47          END
```



miss2.f: frame1

2 次元ベクトル場を作画する UGPACK でも, 同様の欠損値処理を行ないます. UGVECT ルーチンでは, ベクトルの少なくとも 1 成分が欠損値の時にはベクトルを描きません. UGLSET ルーチンで UGPACK に関する内部変数'LMISSP' を .TRUE. にすると, 欠損値ベクトルの格子点にマーカー × を描きます. この印のマーカータイプやマーカーサイズは, やはり, UGISET, UGRSET ルーチンでそれぞれ対応する内部変数'ITYPE1', 'RSIZEM' を設定することにより変更できます.

第 12 章

カラーグラフィクス

いろいろな色が思う存分に使えたらグラフィクスの表現力は格段に増します. GRPH1 の出力プリミティブはすべてカラー対応していますから, ちょっとしたプログラムの追加・変更でカラーグラフィクスが可能です.

12.1　カラーマップ

GRPH1 の出力プリミティブのうち, トーンプリミティブ以外はすべて線分で構成されています. 線分の属性のうち, ラインインデクスには色の情報が含まれています. また, トーンプリミティブには, トーンパターン番号という属性があり, やはり色の情報が含まれています.

GRPH1 で描かれる線分のラインインデクスは 3 桁の整数 (nnm) で指定します. 線の太さと色の両方が変えられるようなシステムでは, 上位 2 桁 ($nn = 0 \sim 99$) が色番号, 下位 1 桁 ($m = 0 \sim 9$) が線の太さを表します. (これまでの章では, 下位 1 桁だけを使ってきました.) 色番号は, 1 から 5 までは標準的に, 1: 白または黒 (フォアグラウンド), 2: 赤, 3: 緑, 4: 青, 5: 黄, と決められていますが, それより大きな番号に関しては カラーマップファイルの定義によります.

カラーマップファイルとは, 色番号と実際の色を対応づけるデータファイルであって, 内部変数 'CLRMAP' が示すファイルです. 標準的なライブラリにおいては, X 用として .x11, PS 用として .psx のサフィックスをつけると, それらが優先的に検索されます. 色は RGB の強さで指定されており, それらの値の範囲は 0 から 65535(16bit) です. 機種によってはこれだけの階調が表現できない場合もありますが, 適当な階調に読み変えますから, 異なるシステムでもこのファイルはそのまま有効です.

トーンパターン番号には色の情報とパターンの情報が両方含まれており, 5 桁の整数 ($nnlll$) で指定します. 上位 2 桁が上に述べた色番号, 下位 3 桁がパターン番号です. パターン番号は第 3.5節で説明しましたが, パターン番号として 999 を指定するとべたぬりとなります.

次の COLOR1 プログラムは, カラーマップのサンプルテーブルを作るプログラムです. 38 行めでトーンパターン番号を代入していますように, 100 色を描き出します. このように 1 つの作図では高々100 色しかつかえませんが, カラーマップファイルを取り替えるとさまざまな色が使えます. dcl-x.x/demo/grph2/rakuraku/color/ にも何種類かのカラーマップファイルを置いています. これをカレントディレクトリにコピーしてカラー描画の実行ファイルを実行すると, コピーしたカラーマップが使われるようになります. 例えば, それぞれの RGB の値を大きめにシフトさせるとパステルカラー調のカラーマップとなります.

```
1         PROGRAM COLOR1
2
3         PARAMETER( NP=4, NT=10 )
4         PARAMETER( DD=0.07, DS=0.003 )
5         PARAMETER( DX=DD+DS*2, DY=1.0/NT )
6
7         REAL      XBOX(NP), YBOX(NP)
8         CHARACTER CHR*5
9
10        DATA      XBOX/ 0., 1., 1., 0. /
11        DATA      YBOX/ 0., 0., 1., 1. /
12
13        WRITE(*,*) ' WORKSTATION ID (I) ? ; '
14        CALL SGPWSN
15        READ (*,*) IWS
16
17        CALL GLISET( 'MAXMSG', 300 )
18
19        CALL SGOPN( -ABS(IWS) )
20
21        CALL SGLSET( 'LFULL' , .TRUE. )
22        CALL SLRAT( 0.85, 1. )
23        CALL SLMGN( 0., 0., 0.05, 0.1 )
24        CALL SLSTTL( 'TEST OF COLORMAP', 'T', 0., -0.5, 0.025, 1 )
25
26        CALL SGFRM
27        CALL SGSWND( 0., 1., 0., 1. )
28        CALL SGSTRN( 1 )
29
30        CALL SGSTXS( 0.01 )
31        CALL SGSTXC( 1 )
32        DO 10 J=0,9
33        DO 10 I=0,9
34          X1 = DX*I     + DS + (1-DX*NT)/2
35          Y1 = DY*(9-J) + DS
36          X2 = X1 + DD
37          Y2 = Y1 + DD
38          ITPAT = (I+J*10)*1000 + 999
39
40          CALL SGSVPT( X1, X2, Y1, Y2 )
41          CALL SGSTRF
42
43          CALL SLPVPR( 1 )
44          CALL SGSTNP( ITPAT )
45          CALL SGTNU( NP, XBOX, YBOX )
46          WRITE(CHR,'(I5)') ITPAT
47          CALL SGTXV( X2, Y2+(DY-DD)/2, CHR )
48      10 CONTINUE
49
50          CALL SGCLS
51
52          END
```

12.2 カラーライン

ポリライン, ポリマーカー, テキストなどの線分による出力プリミティブは, 個別にラインインデクスを指定することができましたから, それぞれ, フルに 3 桁の値で指定すると, カラーのライン, マーカー, テキストとなります. また, GRPH2 の各パッケージも, 用いられている線分のラインインデクスに関する内部変数を変更できるようになっていますから, 容易にカラー化できます. 次の COLOR2 プログラムで, カラー線画の実例を見てみましょう.

UZpGET/UZpSET ルーチンで座標軸関連の内部変数の管理を行ないますが, そのなかの 'INDEXT1', 'INDEXT2', 'INDEXL1', 'INDEXL2' がラインインデクスの設定に関するものです. 'INDEXT1' と 'INDEXT2' は座標軸および目盛を描く線分の, 'INDEXL1' と'INDEXL2' はラベルおよびタイトルの文字を描く線分のラインインデクスを指定する内部変数です. 最後が '1' のほうが小さめの目盛やラベル, タイトルを描く時に用いられ, '2' のほうは大きめのものを描く時に用いられます.

この例では, 標準のカラーマップを用いた時, 目盛は淡青で描き, 小さめの文字は濃青で, 大きめの文字は紫で描きます. 太さは, 小さめの目盛は細線で, それ以外はやや太くなります.

折れ線を SGPLU ルーチンで描くのですが, そのラインインデクスは SGSPLI ルーチンで 24, 赤色の太い線と指定しています.

```
      PROGRAM COLOR2

      PARAMETER( NMAX=50 )
      REAL      X(0:NMAX), Y(0:NMAX)
      CHARACTER CSGI*1, CTTL*32

      R    = 3.7
      X(0) = 1950
      Y(0) = 0.5
      DO 10 N=0,NMAX-1
        X(N+1) = X(N) + 1
        Y(N+1) = R*Y(N)*(1.-Y(N))
   10 CONTINUE
      YAVE = RAVE( Y, NMAX+1, 1 )
      DO 20 N=0,NMAX
        Y(N) = -4*(Y(N)-YAVE)
   20 CONTINUE

      WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
      CALL SGPWSN
      READ (*,*) IWS

      CALL GROPN( IWS )
      CALL GRFRM

      CALL GRSWND( X(0), X(NMAX), -1.5, 2.0 )
      CALL GRSVPT(  0.2,     0.9,  0.2, 0.9 )
      CALL GRSTRN( 1 )
      CALL GRSTRF

      CALL SGISET( 'IFONT', 2 )
      CALL UZISET( 'INDEXT1', 402 )
      CALL UZISET( 'INDEXT2', 404 )
      CALL UZISET( 'INDEXL1', 303 )
      CALL UZISET( 'INDEXL2', 123 )
      CALL UXAXDV( 'B', 2., 10. )
      CALL UXAXDV( 'T', 2., 10. )
      CALL UXSTTL( 'B', 'YEAR', 0. )

      CALL UYAXDV( 'L', 0.1, 0.5 )
      CALL UYAXDV( 'R', 0.1, 0.5 )
      CTTL = CSGI(131) // 'T [K]'
      CALL UYSTTL( 'L',CTTL, 0. )

      CALL UXMTTL( 'T', 'INTERANNUAL VARIATION', 0. )

      CALL SGSPLI( 24 )
      CALL SGPLU( NMAX+1, X, Y )
      CALL GRCLS

      END
```

12.3　カラートーン

COLOR3 プログラムはトーンプリミティブを用いたカラー塗り分けの例です. 第 10.3節の U2D7 プログラムではパターンだけで塗り分けていたのを, ここでは色で塗り分けて見ましょう. DO 20 のループでトーンパターン番号を各レベルに割り当てています. トーン番号が 30999 から 85999 までの 56 色 (濃青から, 緑, 黄, 橙, 赤まで) です. 下 3 桁を 999 としてべたぬりを指定しています. コンターは引かずに, トーンバーを右側につけています. ハードフィル (初期値) でぬりつぶしますから, 各ルーチンを呼ぶ順番に気をつける必要があります.

```
      PROGRAM COLOR3

      PARAMETER ( NX=21, NY=21 )
      PARAMETER ( XMIN=-10, XMAX=10, YMIN=-10, YMAX=10 )
      PARAMETER ( DX1=1, DX2=5, DY1=1, DY2=5 )
      PARAMETER ( KMAX=56, PMIN=0, PMAX=1 )
      REAL P(NX,NY), PI(2,KMAX+1)
```

```
 8
 9          DO 10 J=1,NY
10          DO 10 I=1,NX
11            X = XMIN + (XMAX-XMIN)*(I-1)/(NX-1)
12            Y = YMIN + (YMAX-YMIN)*(J-1)/(NY-1)
13            P(I,J) = EXP( -X**2/64 -Y**2/25 )
14       10 CONTINUE
15
16          WRITE(*,*) ' WORKSTATION ID (I)  ? ;'
17          CALL SGPWSN
18          READ (*,*) IWS
19
20          CALL GROPN( IWS )
21          CALL GRFRM
22
23          CALL GRSWND( XMIN, XMAX, YMIN, YMAX )
24          CALL GRSVPT(  0.2,  0.8,  0.2,  0.8 )
25          CALL GRSTRN( 1 )
26          CALL GRSTRF
27
28          DP = (PMAX-PMIN)/KMAX
29          DO 20 K=1,KMAX
30            TLEV1 = (K-1)*DP
31            TLEV2 = TLEV1 + DP
32            IPAT  = (29+K)*1000 + 999
33            CALL UESTLV( TLEV1, TLEV2, IPAT )
34       20 CONTINUE
35
36          CALL UETONE( P, NX, NX, NY )
37          CALL USDAXS
38
39    *-- トーンバー ----
40          CALL GRSWND(  0.0, 1.0, PMIN, PMAX )
41          CALL GRSVPT( 0.85, 0.9,  0.2,  0.8 )
42          CALL GRSTRN( 1 )
43          CALL GRSTRF
44
45          DO 30 K=1,KMAX+1
46            PI(1,K) = PMIN + (K-1)*DP
47            PI(2,K) = PMIN + (K-1)*DP
48       30 CONTINUE
49
50          CALL UETONE( PI, 2, 2, KMAX+1 )
51
52          CALL SLPVPR( 3 )
53          CALL UZLSET( 'LABELYR', .TRUE. )
54          CALL UZFACT( 0.8 )
55          CALL UYSFMT( '(F4.1)' )
56          CALL UYAXDV( 'R', 0.1, 0.2 )
57
58          CALL GRCLS
59
60          END
```

第 13 章

地図投影や 3 次元図形も

いま, DCL グラフィクスで精力的に開発・整備されているパッケージに, 地図投影と 3 次元グラフィクスがあります. これらのサブルーチンはこれから変更されていく可能性があるので, ここではいくつかの出力結果を紹介するに留め, 具体的な解説はいつか将来に致しましょう.

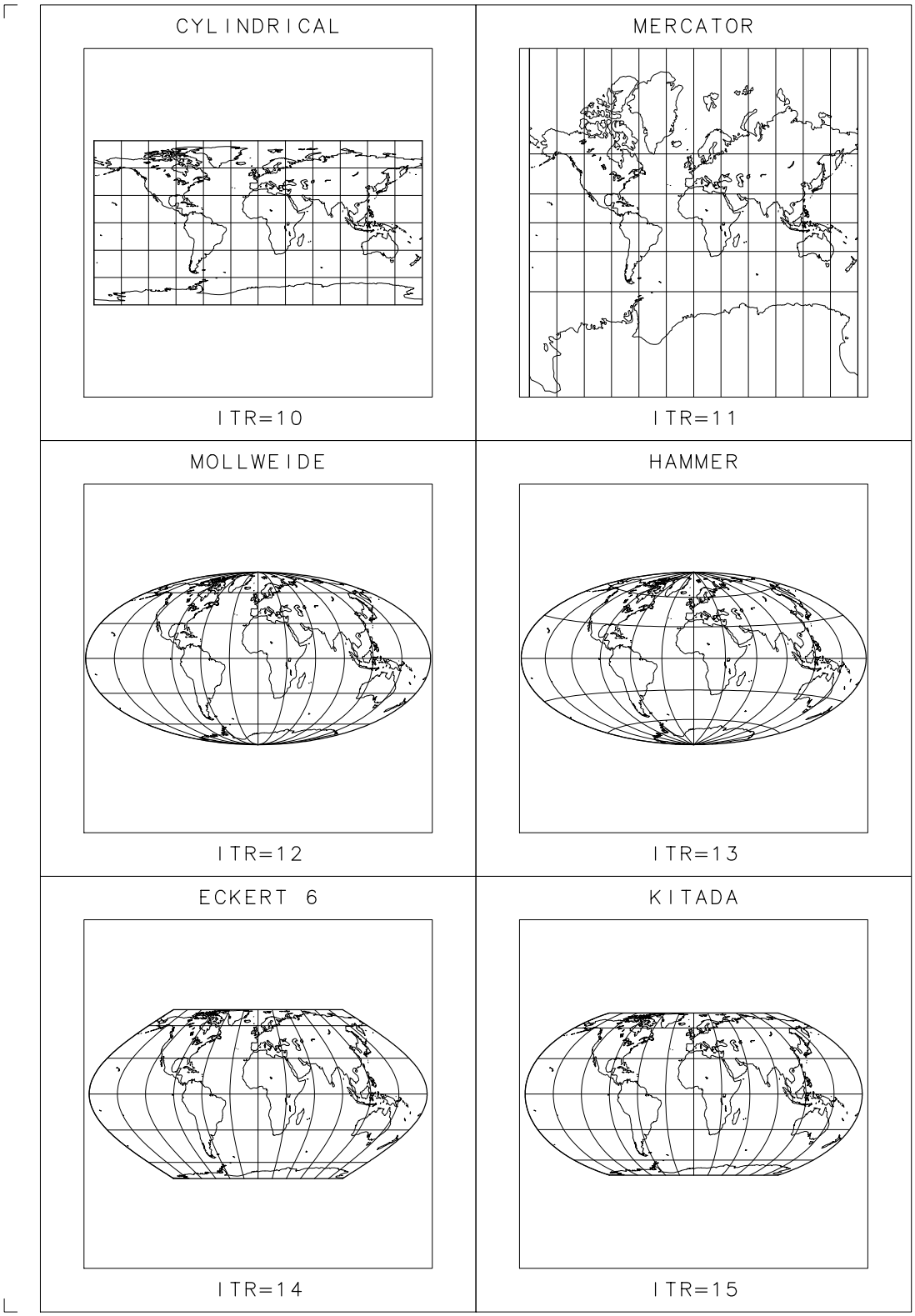
さっそく使ってみたいという人は, GRPH1, GRPH2 のマニュアルの該当するところをお読み下さい. また, この章で示す図は, dcl-x.x/demo/rakuraku/map3d/ にサンプルプログラムがあります. また, dcl-x.x/demo/grph2/umpack と dcl-x.x/demo/grph1/scpack のディレクトリにあるテストプログラムなども覗いてみて下さい.

13.1　いろいろな地図投影法

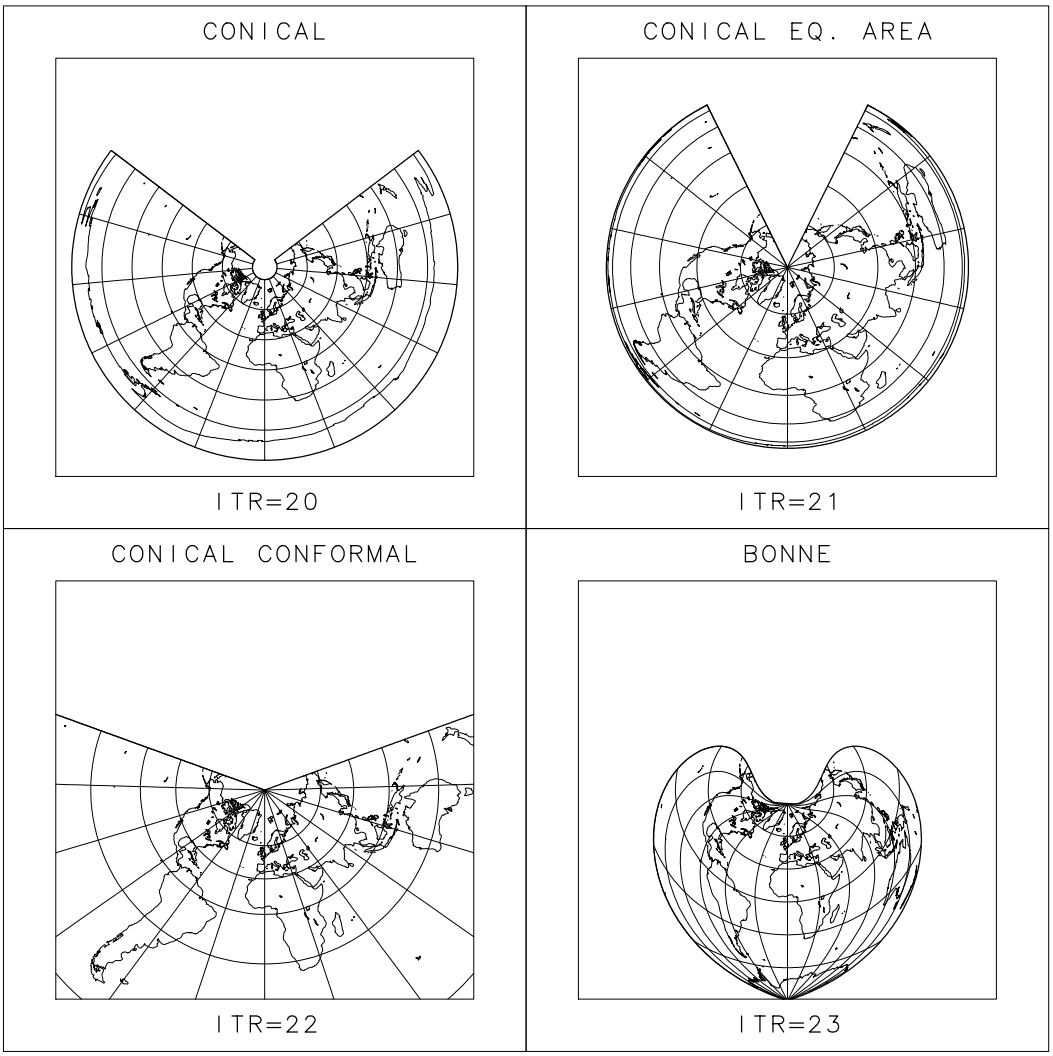
GRPH1 で扱える座標系としては , 直角直線座標系などのほかに 14 種類の地図投影座標系があります. U-座標系から V-座標系への正規化変換でこれらの座標系を設定するれば, それぞれの地図座標系で描画が可能となります.

これらの地図投影変換を用いて, 全球 (いくつかの例外あり) を表示した結果を map3d1.f の出力図に示します. 変換関数番号 (ITR) が 10 から 33 までの 14 種類の投影法で海岸線情報と緯度・経度線を描きます. 第 1 フレームは円筒図法 (6 種), 第 2 フレームは円錐図法 (4 種), 第 3 フレーム方位図法 (4 種) の地図投影法です.

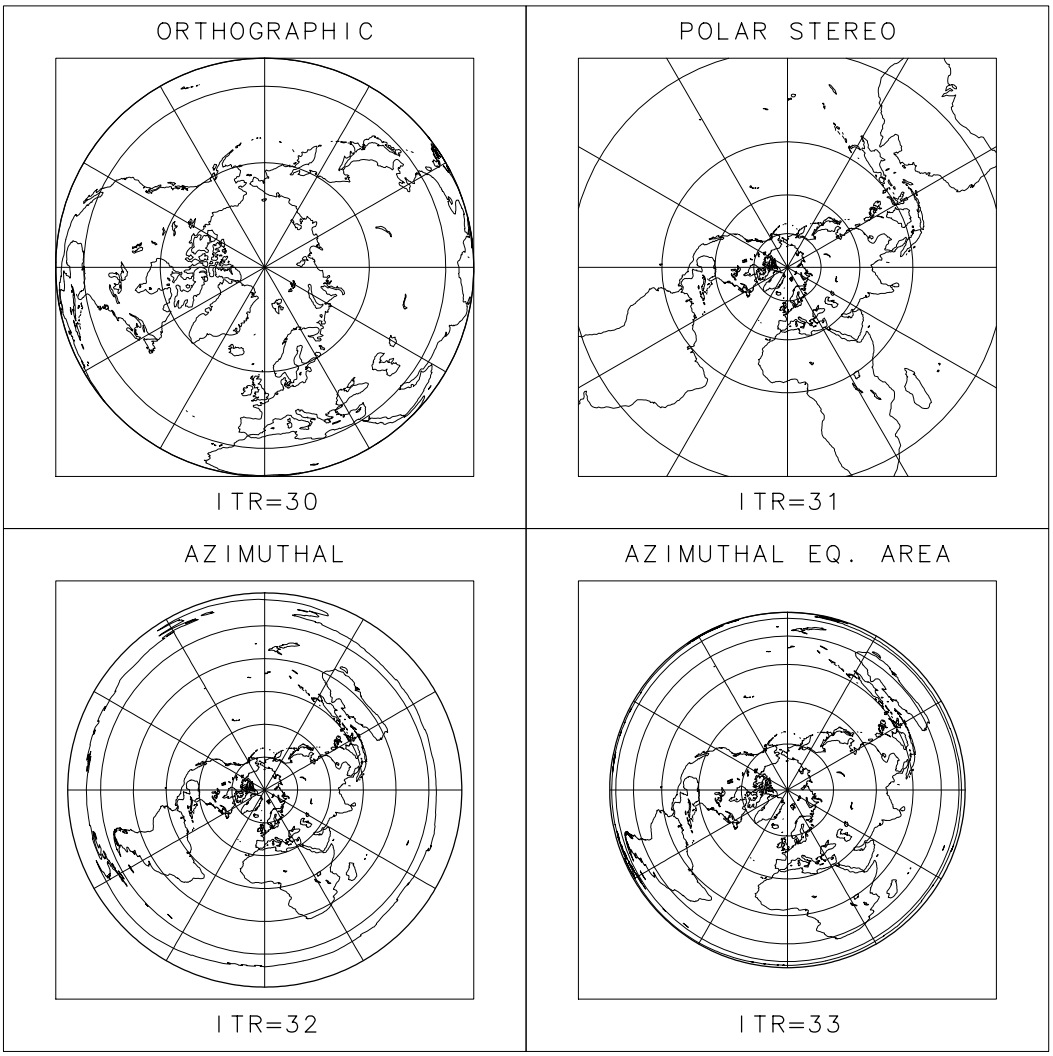
DCL で指定できる地図情報としては, 海岸線, 国境, 米国の州境, 日本の県境, プレート境界などがあります.



map3d1.f: frame1



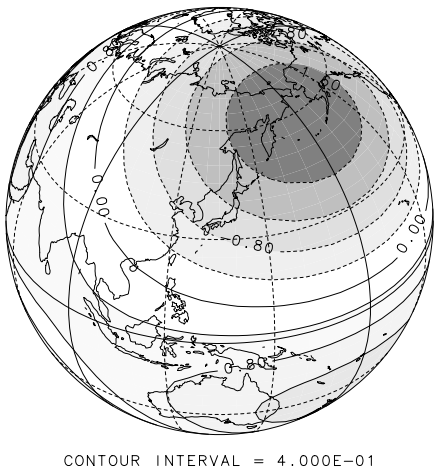
map3d1.f: frame2



map3d1.f: frame3

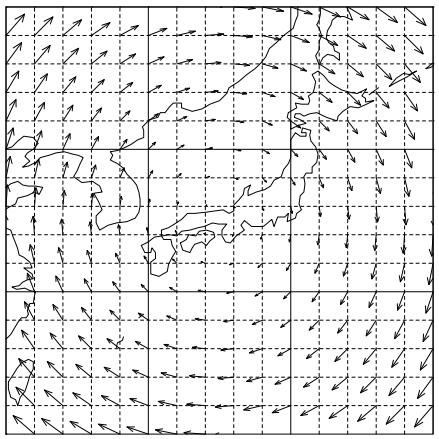
地図情報や緯線 ･経線を描くのと同時に, 等高線図やトーンによる塗りわけをおこなったり, ベクトル場を描いたりすることが可能です. 等高線図作画パッケージ UDPACK とトーンによる塗りわけパッケージ UEPACK はともに U-座標系で作画するルーチン群ですから, 正規化変換で地図投影変換を指定していてもこれまでと全く同じように利用できます.

プログラム map3d2.f では, これまで描いてきた球面調和関数の重ね合わせを正射図法で描きます. 第 10 章のプログラム U2D5 と同じように等高線とハッチが地図上で作画され, 地球の裏側の部分は描かれないようになっています.



map3d2.f: frame1

ベクトル場作画パッケージ UGPACK は V-座標系上で作画しますから, 残念ながら, 現在のところ地図投影には対応していません. しかしながら, SGPACK のアローサブプリミティブを使って, 緯度 ･経度の単位を持った矢印を地図上に描くことは簡単にできます. map3d3.f の例のように, 限られた地域だけでベクトル場を描くならば, 緯度 ･経度の単位で矢印をスケーリングしてもそれほど気にならないでしょう.

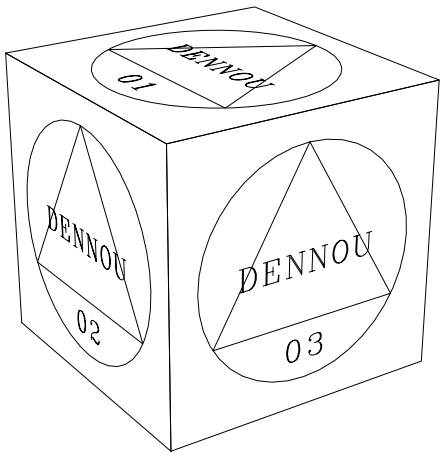


map3d3.f: frame1

13.2　3 次元透視変換

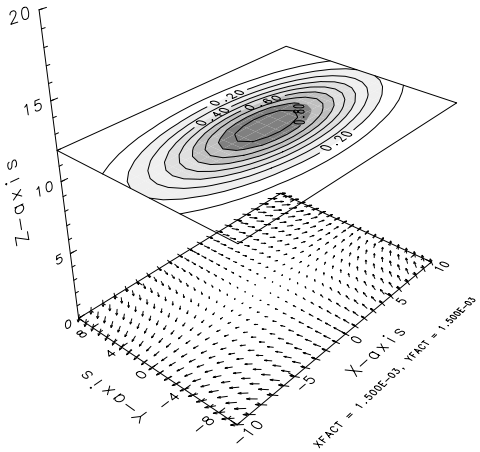
3 次元透視変換とは, 3 次元空間内に置かれた 3 次元図形を任意の視点から眺めて 2 次元平面に投影する変換で, 遠近感が出る変換です. 3 次元グラフィクスをうまく使うと, これまでに見てこなかった新しい表現法が可能となります.

まず, 3 次元空間内で三軸に平行に置かれた立方体の表面に, 2 次元の作図をしてみましょう (map3d4.f). 作画する 2 次元平面をちゃんと指定すれば, これまでの描画サブルーチンがそのまま使えます.



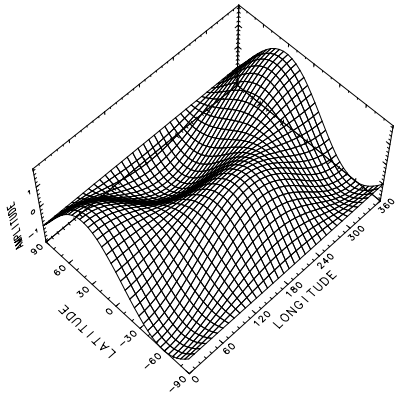
map3d4.f: frame1

また, GRPH2 のパッケージもそのまま使えます. map3d5.f のプログラムでは, 第 10.3節で描いた等高線とベクトル場を 2 段重ねにして表現してみました.



map3d5.f: frame1

さらに, 3 次元空間内で線を引いたり, 面を定義してトーンをつけたりすることも可能です. map3d6.f の図は, これまで何度も描いてきた球面調和関数をワイヤフレームで表現するものです. この 2 次元スカラー場を図示するのに, 縦横に張ったワイヤの凹凸として 3 次元的に表現しています. データの値の大小がワイヤを張った面の高低となって, 視覚的にデータを把握しやすくなります.



map3d6.f: frame1

第 14 章

グラフィクス以外にも

DCL はグラフィクスだけではありません. GRPH1, GRPH2 の他にも MATH1, MATH2, MISC1, MISC2(今は空です), ETC などさまざまな機能を持つパッケージ群があります. うまく使うと非常に機能的にプログラムできるようになります.

ここでは概観するだけにしますが, それぞれのマニュアルを参照していろいろ使ってみましょう. この辺をカバーする 「らくらく DCL Part II」をそのうち出せれば良いのですが.

14.1 MATH1

数学処理下位パッケージ MATH1 は, DCL のなかでもっとも基本的なサブルーチンおよび関数を集めたパッケージです.

MATH1 は, 次のサブパッケージから構成されています.

- **SYSLIB :** 内部変数管理, メッセージ出力
- **OSLIB :** システム依存ルーチン
- **FNCLIB :** 基本関数 (最大整数, 剰余など)
- **SUBLIB :** 基本サブルーチン (自然数列の生成など)
- **CHRLIB :** 文字列の左・右詰め, 反転, 空白処理
- **LRLLIB :** 実数値の比較
- **BLKLIB :** 実数値と順序列
- **GNMLIB :** きりのよい打ち切り数
- **INTLIB :** 実数に近い整数
- **INDXLIB :** 配列要素の検索
- **IFALIB :** 整数の欠損値処理付最大最小など
- **RFALIB :** 実数の欠損値処理付最大最小など

- **RFBLIB :** 実数列の内積, 共分散, 相関係数
- **VIALIB :** 1 つの整数型配列への作用素
- **VIBLIB :** 2 つの整数型配列への作用素
- **VRALIB :** 1 つの実数型配列への作用素
- **VRBLIB :** 2 つの実数型配列への作用素
- **CTRLIB :** 座標変換/回転
- **MAPLIB :** 地図投影変換

これらの中の多くのパッケージでは,「欠損値の処理」や「実数の誤差を考慮した大小関係の判定」ができるようになっています. なお, これらのパッケージの中でソースレベルでの機種依存性があるのは OSLIB のみです. SYSLIB はシステムに依存する定数を管理しているだけです.

14.2 MATH2

数学処理上位パッケージ MATH2 は, 地球流体の様々な現場で標準的に用いられる数値計算のための基本ツール群として計画されています. しかし残念ながら, あまり整備が進んでいないというのが現状です.

MATH2 に対応するパッケージには, 既存のソフトウェア (例えば IMSL 等) を参照すればソースコードが手に入るようなものが多いのですが, その精度等の信頼度は必ずしも我々にとって十分なものとは言えません. また, これらのソフトウェアには PDS(Public Domain Softwares) でないものが多いので, 自分たちのソフトウェアに組み込んで自由に配布することができないという難点があります.

MATH2 は, 現在, 次のサブパッケージから構成されています.

- **FFTLIB :** 高速フーリエ変換
- **ODELIB :** 常微分方程式 (ルンゲクッタ)
- **SHTLIB :** 球面調和関数
- **VSTLIB :** ベクトルデータの統計処理
- **INTRLIB :** 補間
- **RNMLIB :** 移動平均

MATH2 パッケージの整備目標は, 差分の基本スキーム, 基本統計パッケージ, 特種関数, 固有値問題, ... 等々を自力開発する, あるいは, PDS から精選するということです. このために今も継続的な努力がなされています.

14.3 MISC1

その他の基本処理下位パッケージ MISC1 は, ビットパターン・文字の処理や, ファイルの入出力などシステムに依存するライブラリ, および日付・時間など, 数学的取り扱いとはやや異なるライブラリなどを含んでいます.

MISC1 は, 次のサブパッケージから構成されています.

- **BITLIB:** ビットパターンの処理 * ☆ C
- **CHGLIB:** 大文字 ・小文字の変換 * ☆ FC
- **CHKLIB:** 文字種の判別 *
- **CHNLIB:** 文字列の置換 *
- **FMTLIB:** 数値の文字列化 *
- **DATELIB:** 日付の取り扱い * ☆ D
- **TIMELIB:** 時刻の取り扱い * ☆ D
- **MISCLIB:** 雑多な関数 ・サブルーチン * ☆ FC
- **CLCKLIB:** CPU 時間の取り扱い ☆
- **FIOLIB:** ファイルの入出力 ☆
- **RANDLIB:** 疑似乱数 ☆ FC
- **HEXLIB:** 16 進定数の処理
- **REALLIB:** 実数の変換

ここで, *, ☆, D, F, C の各印は, 次のような意味を持っています.

***** : MISC1 以外のライブラリの実行に必要なパッケージ

☆ : 機種依存性のあるサブルーチンを含む

D : 機種依存性のあるサブルーチンをダミールーチンとしても, 致命的な支障はないもの

F : 機種依存性のあるサブルーチンに対して, FORTRAN のプロトタイプ (機種依存性がないと思われるコード) が用意されているもの

C : 機種依存性のあるサブルーチンに対して, C の関数が用意されているもの

14.4 ETC

ETC は, DCL を使いこなす上で便利な道具たちを集めたものです. その中には電脳倶楽部で作成したもの以外のものも入っています.

ETC は, 次のライブラリから構成されています.

- LaTeX マクロライブラリ :
- **PS** 周辺ライブラリ :

前者は, LaTeX を用いたドキュメントの作成に便利なマクロ定義スタイルファイル集です. DCL のマニュアルもこのマクロを用いて書かれています. 後者は, DCL の PostScript(PS) 出力を加工するためのフィルタコマンド集です.

第 15 章

付録

この付録では, まず, GRPH1 と GRPH2 のパッケージの一覧表を示します.

また, フォント, トーンパターンに関する各種テーブルを示します. これらのテーブルを生成するプログラムは dcl-x.x/demo/grph1/sgpack/ の中にあります. フォントについては sgfont.f, トーンパターンについては sgtone.f によって対応するテーブルを生成できます.

15.1 GRPH パッケージ一覧

15.1.1 GRPH1

図形処理下位パッケージ GRPH1 は, 座標変換をしたり, 線分などの基本的な図形を描画するパッケージです. このパッケージは, MATH1 と MISC1 のライブラリを使用しています.

GRPH1 は, 次のサブパッケージから構成されています.

- **SGPACK :** ユーザーインターフェイス
- **SLPACK :** 複数の図の割り付けのための基本ルーチン
- **SZPACK :** 線分やトーンなどの図形を描画する基本ルーチン
- **STPACK :** 座標変換の基本関数
- **SWPACK :** 実際の描画デバイスに出力する下請けルーチン

通常, ユーザーが使うのは主として SGPACK と SLPACK のルーチン群で, その他のパッケージを使わなければならない場合は少ないでしょう. しかし, 実際の描画動作は SZPACK がおこないますから, 基本的には SZPACK だけでプログラムを書くことも可能です. SGPACK は各種パラメーター設定などのオーバーヘッドにかなりの時間が使われる場合がありますから, SZPACK を使えばその分だけ高速化できるはずです.

なお, これらのパッケージのうちで, 機種依存部分を含むのは SWPACK だけです.

15.1.2 GRPH2

図形処理上位パッケージ GRPH2 は, GRPH1 の折れ線や文字といった単純な図形要素を組み合せて, 座標軸や等高線図などを描くルーチンを集めたものです.

GRPH2 はレベル 2 のライブラリであり機種に依存しませんが, GRPH1, MISC1, MATH1 が正しくインストールされている必要があります.

GRPH2 は, 次のサブパッケージから構成されています.

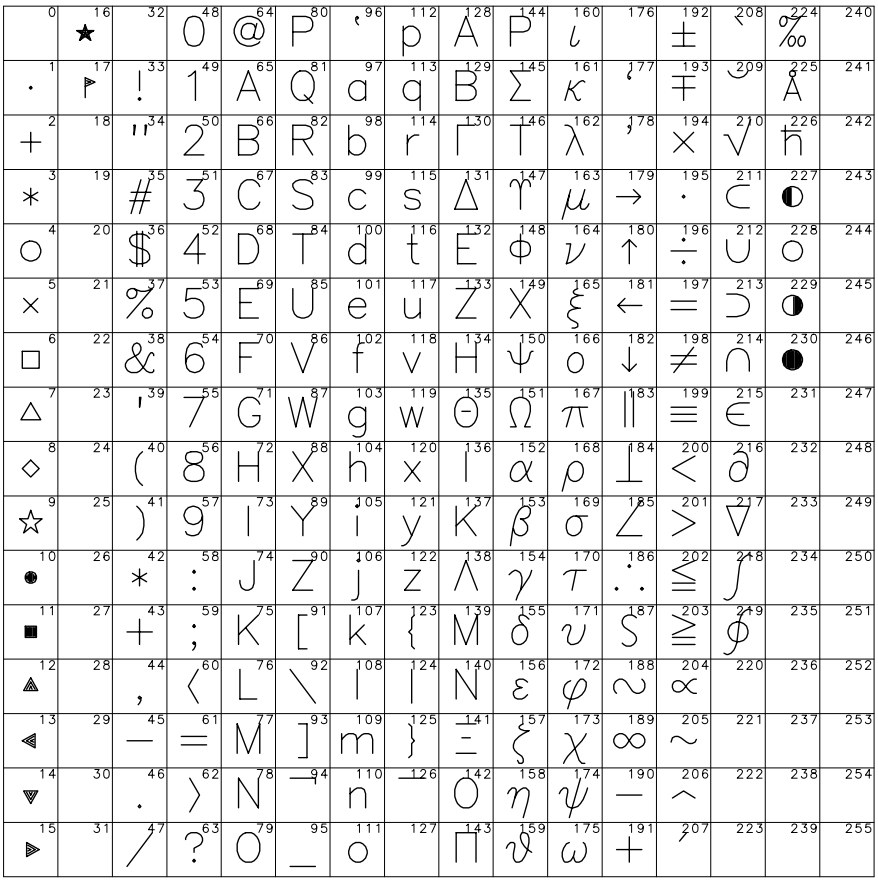
- **GRPACK：** 上位コントロールルーチン
- **U[XYZ]PACK：** 直交座標軸ルーチン
- **ULPACK：** 対数座標軸ルーチン
- **UCPACK：** 日付に関する座標軸ルーチン
- **USPACK：** 自動スケーリングルーチン
- **U[UVH]PACK：** 1 次元量のグラフ表示ルーチン
- **UDPACK：** 等高線ルーチン
- **UEPACK：** トーン塗りつぶしルーチン
- **UGPACK：** ベクトル場表示ルーチン
- **UWPACK：** 格子点情報の管理ルーチン
- **UMPACK：** 地図投影ルーチン

15.2 フォント一覧

15.2.1 フォントテーブル 1

次のテーブルは, フォント番号 1 のフォントテーブルです.

FONT NO. = 1

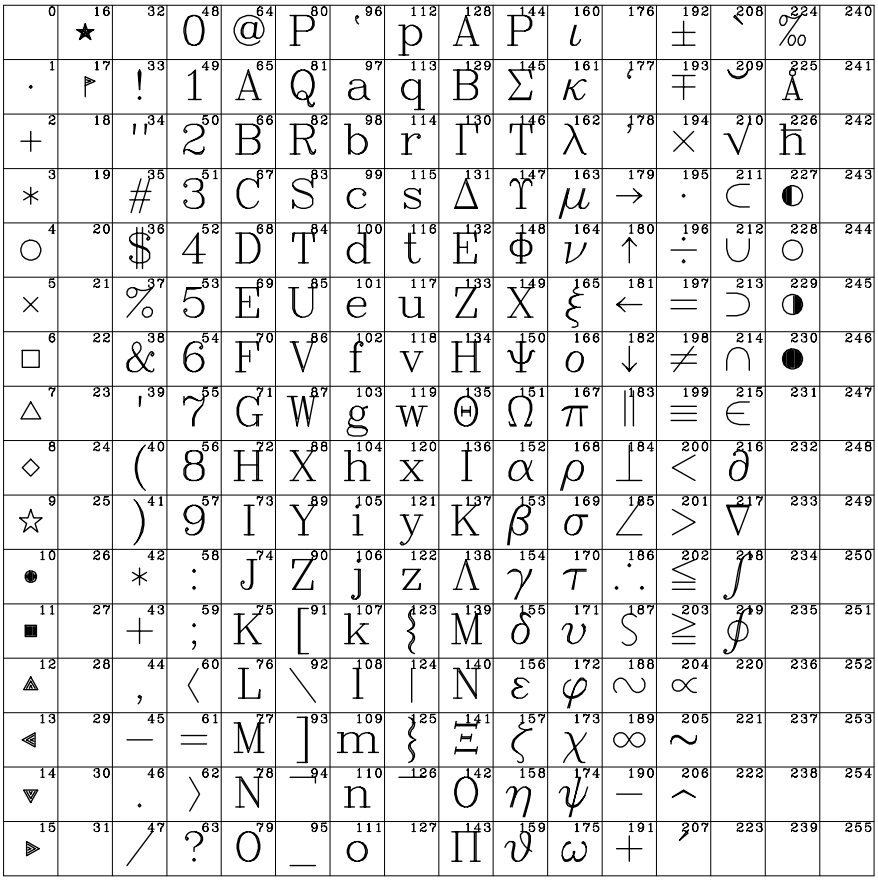


sgfont.f: frame1

15.2.2　フォントテーブル 2

次のテーブルは、フォント番号 2 のフォントテーブルです.

FONT NO. = 2

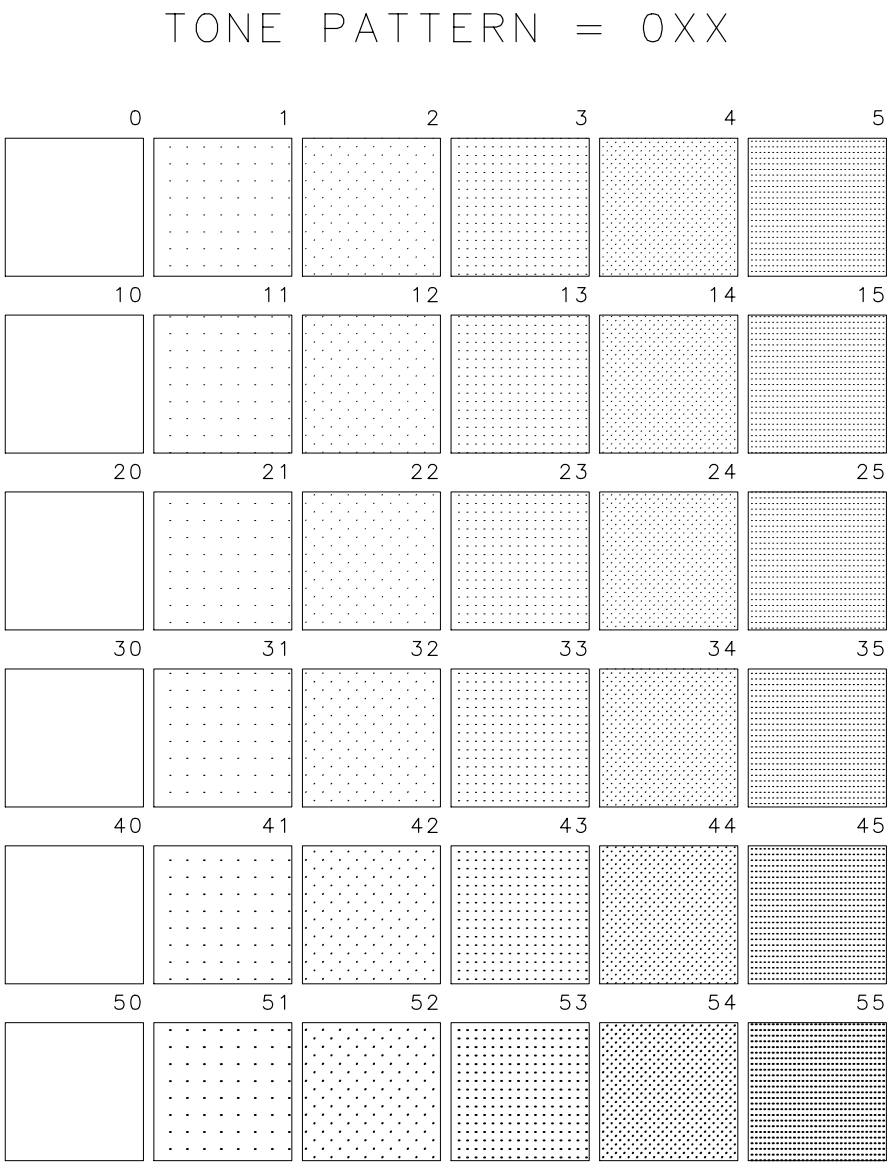


sgfont.f: frame2

15.3 トーンパターン一覧

15.3.1 トーンパターンテーブル 0

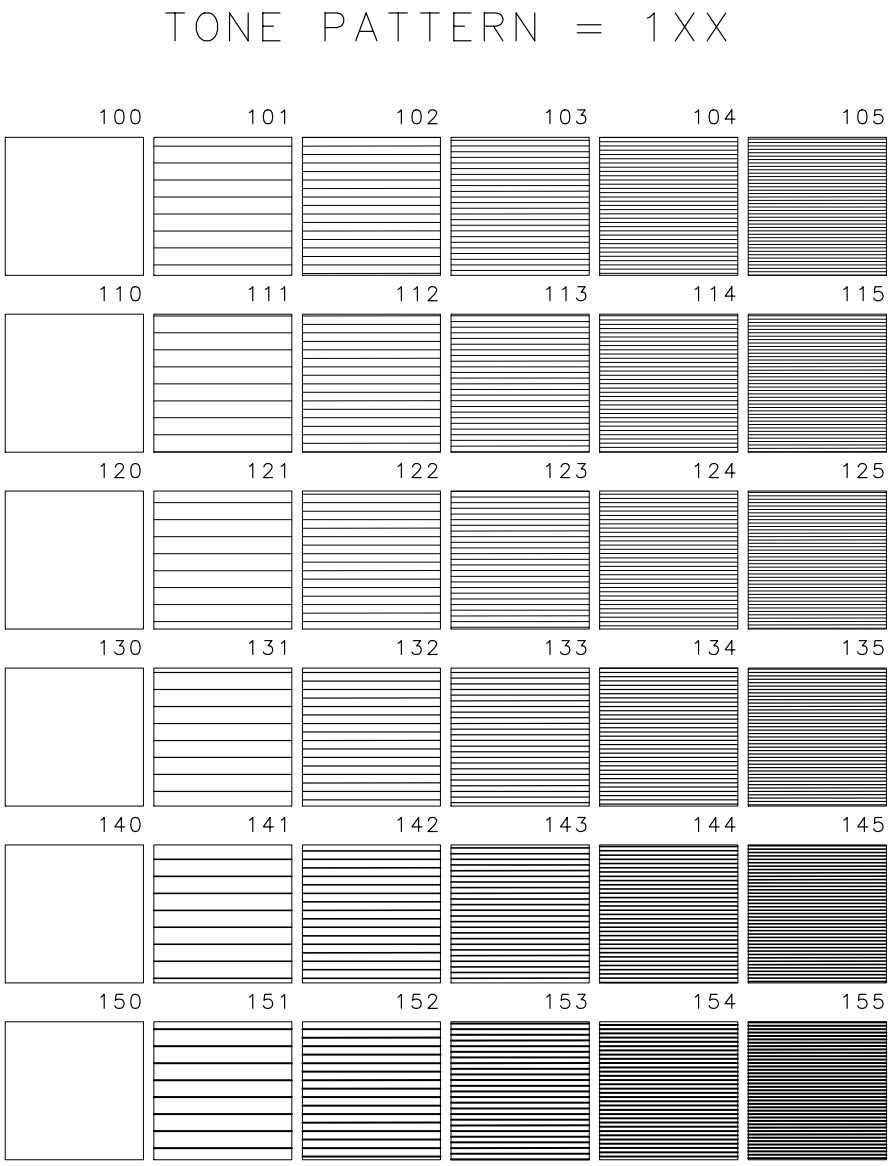
次のテーブルは, パターン番号の百の位が 0 のトーンパターンテーブルです.



sgtone.f: frame1

15.3.2　トーンパターンテーブル 1

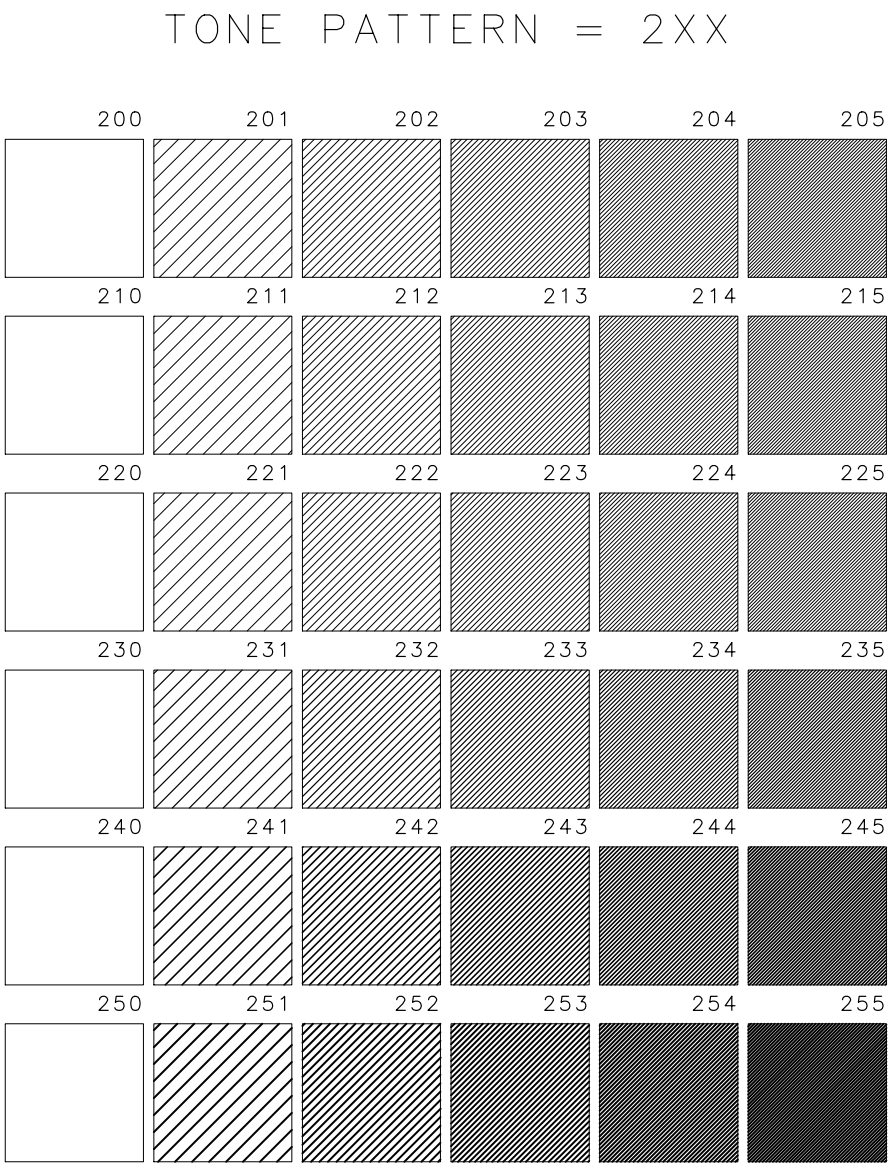
次のテーブルは, パターン番号の百の位が 1 のトーンパターンテーブルです.



sgtone.f: frame2

15.3.3 トーンパターンテーブル 2

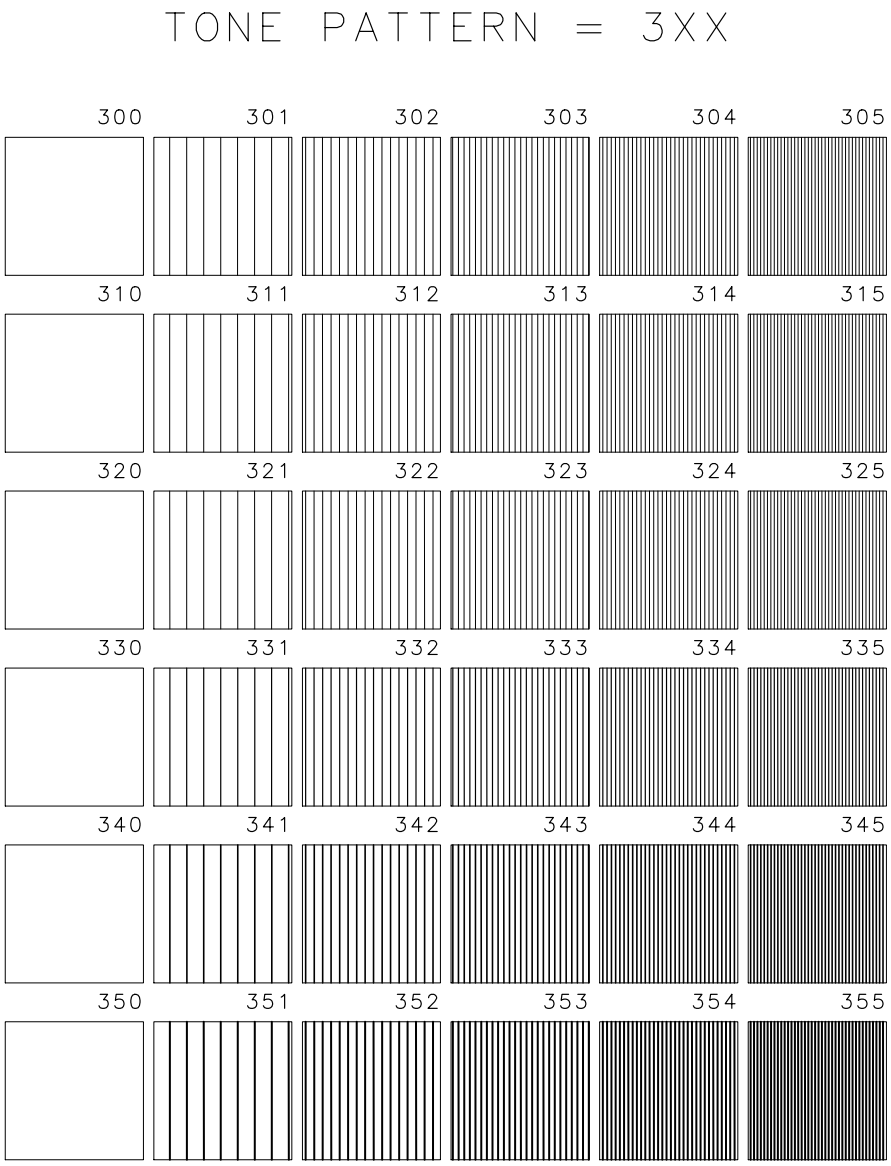
次のテーブルは, パターン番号の百の位が 2 のトーンパターンテーブルです.



sgtone.f: frame3

15.3.4　トーンパターンテーブル 3

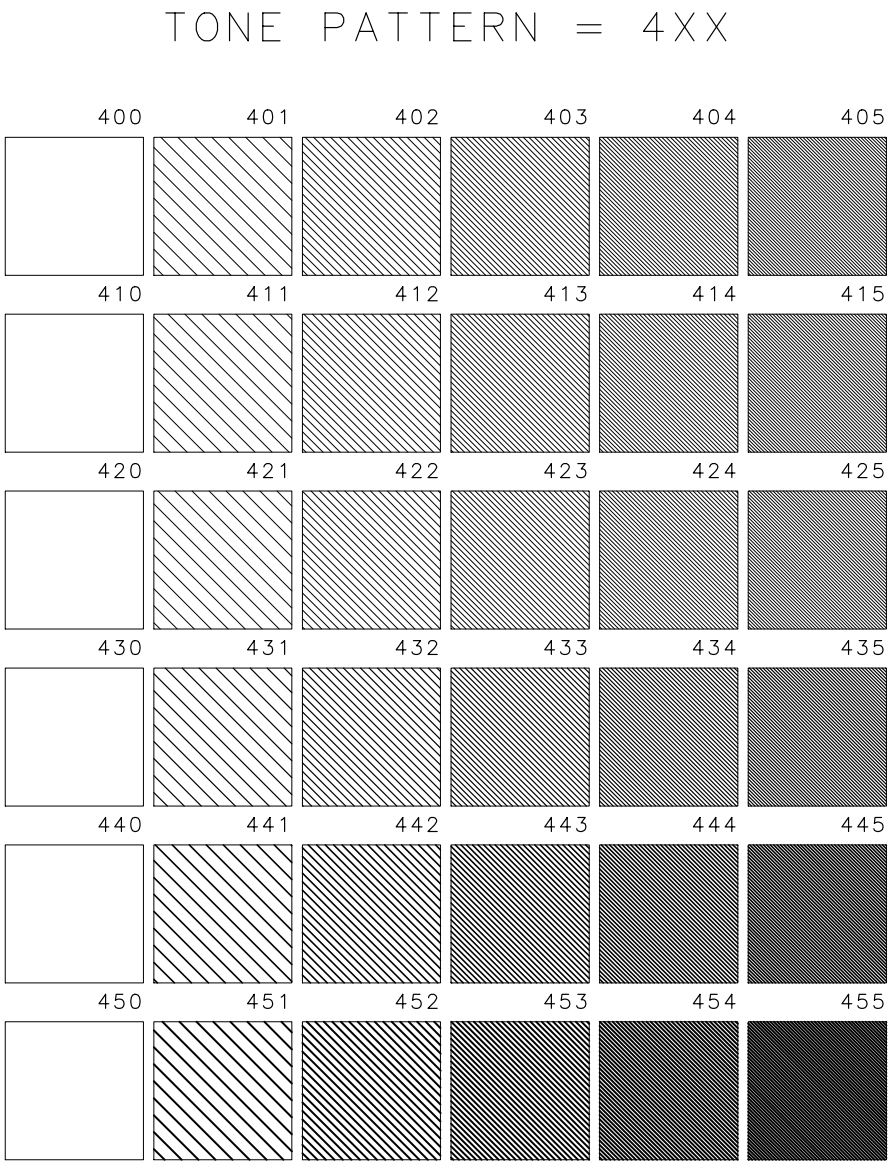
次のテーブルは, パターン番号の百の位が 3 のトーンパターンテーブルです.



sgtone.f: frame4

15.3.5 トーンパターンテーブル4

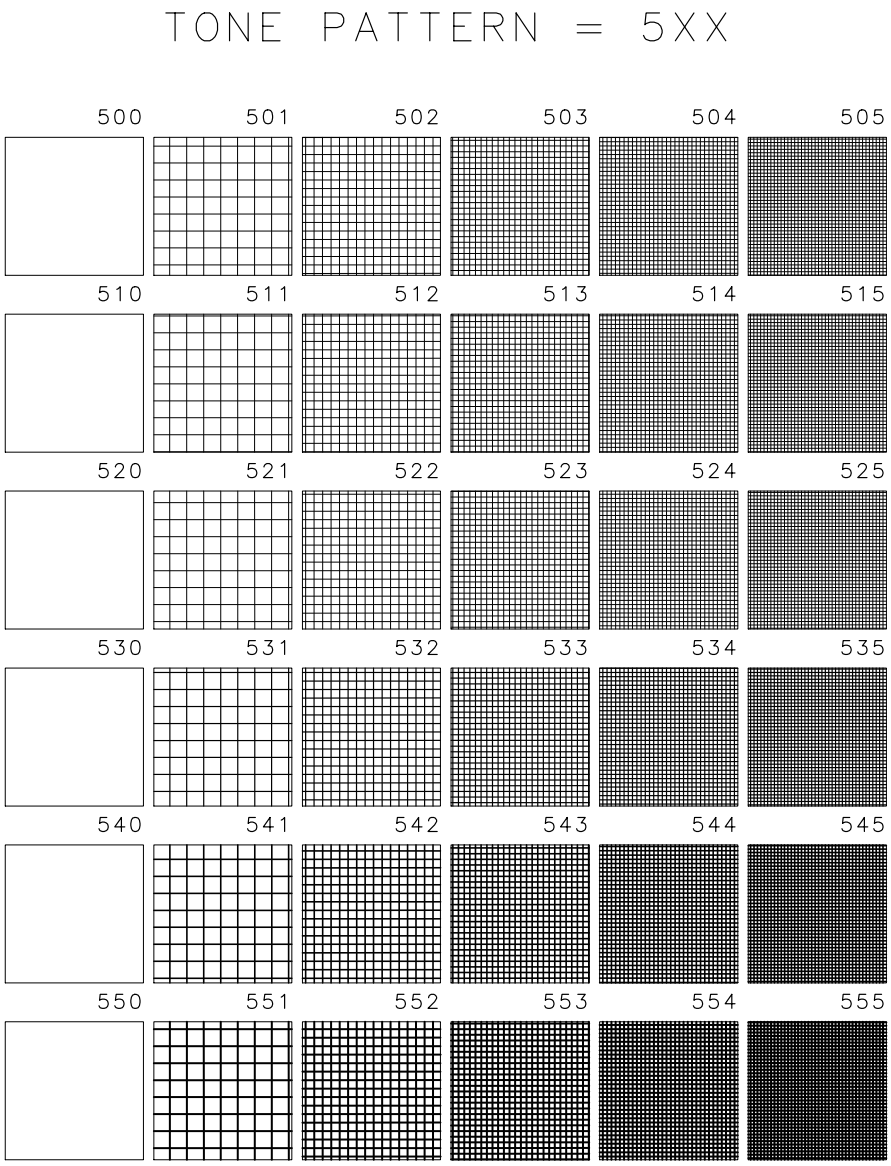
次のテーブルは, パターン番号の百の位が 4 のトーンパターンテーブルです.



sgtone.f: frame5

15.3.6　トーンパターンテーブル 5

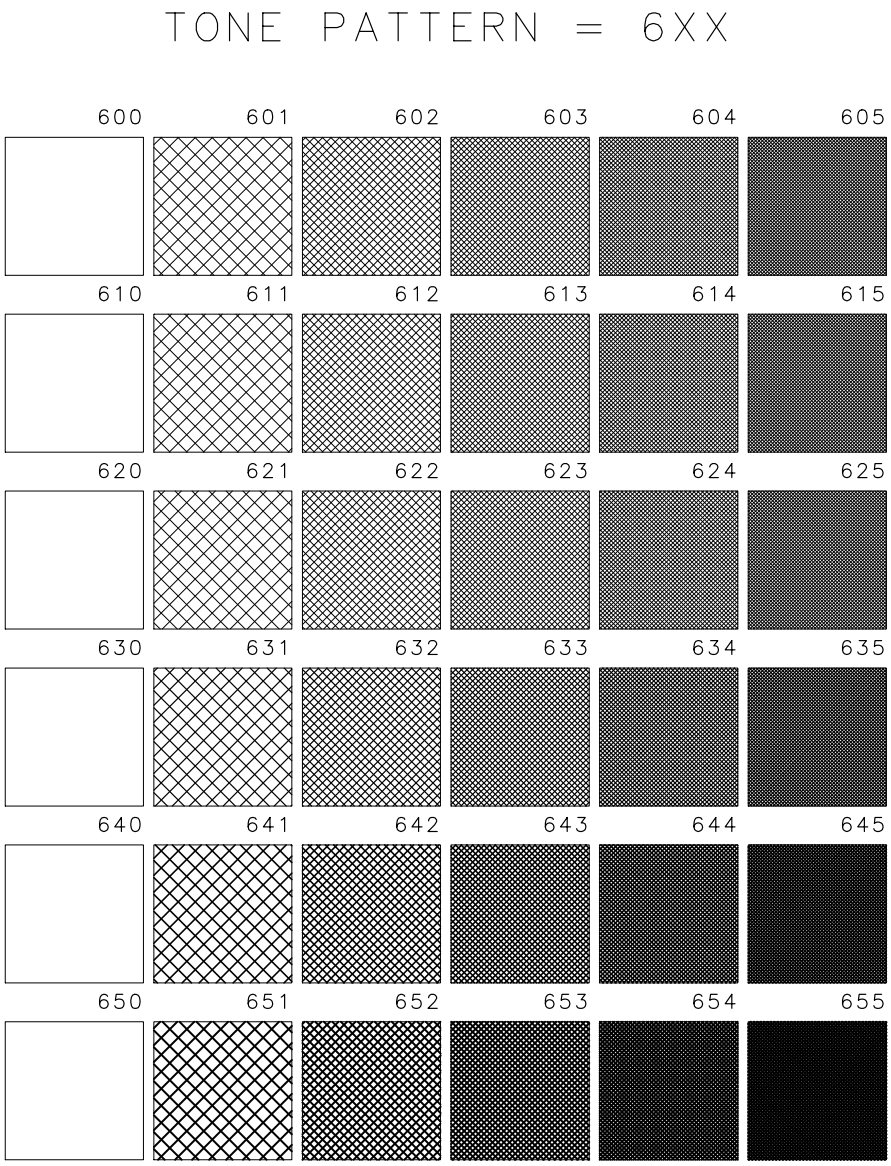
次のテーブルは, パターン番号の百の位が 5 のトーンパターンテーブルです.



sgtone.f: frame6

15.3.7 トーンパターンテーブル6

次のテーブルは, パターン番号の百の位が 6 のトーンパターンテーブルです.



sgtone.f: frame7

謝辞

資源	**dcl-5**
プログラム製作	石岡圭一, 酒井敏, 佐藤薫, 塩谷雅人, 沼口敦
プログラム作成協力	石渡正樹, 竹広真一, 中島健介, 西憲敬, 林祥介, 保坂征宏, 余田成男
デバッグ協力	石田明生, 石渡正樹, 市川香, 沖大幹, 沖理子, 桜井隆博, 佐藤正樹, 竹広真一, 中島健介, 沼口敦, 根田昌典, 野村真佐子, 藤尾伸三, 保坂征宏
文章	酒井敏, 塩谷雅人, 余田成男
文章協力	石渡正樹, 佐藤薫, 佐藤正樹, 竹広真一, 中島健介, 西憲敬, 沼口敦, 林祥介, 保坂征宏
データソース	NCAR [フォント情報, 地図情報]
参考資源	Layout [二木徹], XYGRAPH [佐藤亨], Numerical Recipes [Willam H. Press et al.], NCARG [NCAR], FFTPACK [Paul Swarztrauber]

版権

本資源の版権は地球流体電脳倶楽部に属する. 資源の利用にあたっては地球流体電脳倶楽部の定める規定にしたがっていただきたい. 原則として, 教育的目的の場合には自由に使用・改変して良いものとしている. 地球流体電脳倶楽部が定める規定の詳細は地球流体電脳倶楽部サーバー

```
dennou.gaia.h.kyoto-u.ac.jp
```

上の

```
~ftp/saloon/dennou/To.Users
```

というファイルに収められているので参照されたい. (このファイルは電脳ライブラリを収めたファイルシステムのトップディレクトリにも置かれている.) 上記のアクセスができない場合は, 電子メールにて地球流体電脳倶楽部管理グループ

```
dennou_admin@gaia.h.kyoto-u.ac.jp
```

へ連絡されたい.

引用について

「地球流体電脳ライブラリ」の著作者は「地球流体電脳倶楽部」, その英語名は 'GFD-DENNOU Library', 'GFD-DENNOU Club' である. 地球流体電脳ライブラリを用いて作成された著作物等には, 英語では例えば "The figures were produced by GFD-DENNOU Library." 日本語では例えば「図の作成には地球流体電脳ライブラリをもちいた.」のように引用していただければ幸いである.