The GNU Plotting Utilities

ベクトル・グラフィクスとデータ・プロットのためのプログラムおよびライブラリ
Version 2.6

Robert S. Maier (とみながだいすけ訳)

1 はじめに

GNU plotting utilities (plotutils) には８つのコマンドライン用のプログラムがある。画像を生成する `graph`、`plot`、`pic2plot`、`tek2plot`、`plotfont` と、数値処理を行う `spline`、`ode`、`double` である。GNU plotting utilities は、これらのプログラムと、それらが内部で利用している描画関数のライブラリ `libplot` をあわせたパッケージである。`libplot` は特定のデバイスに依存しない二次元のベクトル形式画像を描くことを目的としたライブラリであり、X Window システム上でアニメーションを行う機能もある。C と and C++ の両方の言語バインディングが用意されている。

これらのプログラムと `libplot` は、以下の画像形式を扱うことができる。

X 出力形式に X が指定されているときは、画像データはファイルや標準出力には出力されない。画像は X Window の画面上のウィンドウに描かれる。

PNG PNG (“portable network graphics”) は現在 web 上で普及しつつある画像形式である。GIF と違って、うっとうしい特許の問題はない。PNG 形式の画像ファイルは、たとえばフリーの `ImageMagick` パッケージのプログラム `display` などで表示させることができる。

PNM PNM はベクター、ビットマップ両方の画像形式を扱える “portable anymap” 形式であり、三種類ある。PBM (白黒画像用の portable bitmap)、PGM (portable graymap)、PPM (カラー画像用の portable pixmap) である。画像ファイルの形式はこの中から選ばれる。PNM は例えば `netpbm` パッケージを使って他の形式に変換したり、`display` で表示したりすることができる。

GIF plotutil が扱うのは本物の GIF ではなく、疑似 GIF 形式である。本物の GIF と違って LZW 圧縮を使わないため、Unisys の LZW 特許に触れていない。しかし疑似 GIF 形式は たとえば `display` など GIF 形式をサポートするソフトウェアならおおよそどれでも、表示したり編集したりできる。

SVG Scalable Vector Graphics、通称 SVG 形式である。これはウェブ上でベクトル形式の画像を扱うための、XML ベースの画像形式である。詳細は W3 Consortium (`http://www.w3.org`) にある。また開発者のための情報はそのウェブサイトの Graphics Activity (`http://www.w3.org/Graphics`) にある。

AI Adobe の Illustrator の画像形式。AI 形式は Illustrator のバージョン 5 以降、および他のソフトウェアで表示、編集できる。

PS `idraw` で編集できる PostScript である。この形式のファイルは PostScript プリンタで印刷でき、また他の文書に取り込んだり、フリーのドロー系ソフトである `idraw` で編集できる。Section E.1 [idraw], page 163 参照。

CGM Computer Graphics Metafile 形式。CGM プラグインがあればどのウェブ・ブラウザでも表示でき、また他のアプリケーションで取り込むことができる。デフォルトでは WebCGM にしたがった CGM version 3 の形式のバイナリデータが生成される。WebCGM はウェブベースのベクトル形式画像の標準である。CGM Open Consortium (`http://www.cgmopen.org`) で詳細が紹介されている。

Fig フリーのドロー系ソフト `xfig` で表示、編集できる形式。Section E.2 [xfig], page 163 参照。

PCL 5 ヒューレット・パッカードのプリンタの制御言語の機能強化版である。この形式のファイルは LaserJet およびその互換のプリンタで印刷できる (ただ、多くのインクジェットプリンタは PCL 5 非対応である)。

HP-GL ヒューレット･パッカードの画像記述言語である。デフォルトでは新しい HP-GL/2 形式で出力される。HP-GL および HP-GL/2 形式のファイルは、文書に取り込んだり、プロッタに出力できたりする。

ReGIS DEC 端末 (VT340、VT330、VT241、VT240) やそのエミュレータで表示できる画像形式。DECwindows の端末エミュレータ `dxterm` でも表示できる。

Tek Tektronix 4014 端末の画像形式。Tektronix 4014 端末は `xterm` や `kermit` の MS-DOS 版などに実装されている。

Metafile 特定の出力装置に依存することのない GNU metafile 形式。plotutils の `plot` コマンドで、この形式のファイルを上記の他の形式に変換できる。

plotutils のプログラムの中で、もっともよく使われるのは `graph` だろう。これは科学技術分野で 2D のデータプロットを行うことを想定したプログラムである。`graph` は、プロットするデータセットを一つまたは複数のデータファイルから読み込み、一つのプロットを生成する。上記の画像形式はそれぞれ、`graph -T X`、`graph -T pnm`、`graph -T gif`、`graph -T ai`、`graph -T ps`、`graph -T fig`、`graph -T pcl`、`graph -T hpgl`、`graph -T tek`、`graph` として指定できる。'`-T`' オプションをつけない場合は、GNU metafile 形式で出力する (生の `graph` 形式、raw `graph` format とも言う)。

入力データはテキスト (ascii plain text) でもバイナリでもよい。また `gnuplot` の 'table' 形式の出力も読み込める。出力されるプロットでの座標軸やラベルの有無を指定できる。ラベル、各座標軸の範囲、プロットのサイズとディスプレイ上での位置をそれぞれ指定できる。ラベルの文字には下付き、上付き、またギリシャ文字などの記号が使え、さらに (ロシア語などの) キリル文字や日本語も使える。プロットにおいては、各データセットごとにプロットに使う記号、その記号間を結ぶ線 (があればそ) の種類や太さを指定できる。領域の塗りつぶしもでき、エラーバーも付けられる。複数のプロットを同時に一つの画像ファイルとして行うことは、まったく問題なくできる。`graph` を一回実行するだけで、複数のプロットを並べた画像や、重ねた (inset) 画像を得ることができる。データセット、座標軸の設定は各プロットでそれぞれ別にできる。

`graph -T X`、`graph -T tek`、`graph -T regis` およびオプションなしの `graph` では、ほかのプロットソフトウェアにはあまりない機能として、入力を標準入力からパイプで受け取りながら、リアルタイムで出力をプロットすることができる。そのためには `graph` がデータを最後まで読み込まなくてもプロット領域を決められるように、両方の座標軸の範囲を指定しておく。

`plot` は一種の「プロット・フィルタ」とも言うべきものである。`plot` は GNU graphics metafile 形式 (たとえば `graph` の生の出力) を、対応している他の形式に変換して出力する。各形式への変換はそれぞれ、`plot -T X`、`plot -T png`、`plot -T pnm`、`plot -T gif`、`plot -T svg`、`plot -T ai`、`plot -T ps`、`plot -T cgm`、`plot -T fig`、`plot -T pcl`、`plot -T hpgl`、`plot -T regis`、`plot -T tek`、`plot` で行える。`plot` は、`graph` での描画は一回だけにしたいが複数の形式の画像ファイルがほしい、という場合に役に立つ。または、いくつかのシステムにある GNU 版でない `graph` コマンドのような、昔の 'plot(5)' の形式のファイルを変換したいときにも使える。

`pic2plot` コマンドは pic 言語で表現されている画像ファイルを対応している他の形式に変換して出力する。pic 言語はベル研で開発された言語で、技術文書や解説書など

によくあるような長方形と矢印からなる図を記述できる。各形式への変換はそれぞれ、`pic2plot -T X`、`pic2plot -T png`、`pic2plot -T pnm`、`pic2plot -T gif`、`pic2plot -T ai`、`pic2plot -T ps`、`pic2plot -T cgm`、`pic2plot -T fig`、`pic2plot -T pcl`、`pic2plot -T hpgl`、`pic2plot -T regis`、`pic2plot -T tek`、`pic2plot` で行える。

`tek2plot` は Tektronix 形式の画像データを対応している他の形式に変換して出力する。各形式への変換はそれぞれ、`tek2plot -T X`、`tek2plot -T png`、`tek2plot -T pnm`、`tek2plot -T gif`、`tek2plot -T svg`、`tek2plot -T ai`、`tek2plot -T ps`、`tek2plot -T cgm`、`tek2plot -T fig`、`tek2plot -T pcl`、`tek2plot -T hpgl`、`tek2plot -T regis`、`tek2plot` で行える。Tektronix 形式でしか出力しないような古いプログラムがあるときに重宝する。

`plotfont` は `graph`、`plot`、`pic2plot`、`tek2plot` で使えるフォントのキャラクタ・マップを出力するユーティリティである。‘`-T X`’、‘`-T ai`’、‘`-T ps`’、‘`-T cgm`’、‘`-T fig`’ オプションのどれかが指定されたときは、35 種類の PostScript の標準フォントが使える。‘`-T ai`’、‘`-T pcl`’、‘`-T hpgl`’ オプションが指定されたときは、45 種類の PCL 5 の標準フォント (つまり “LaserJet” フォント) が使える。‘`-T pcl`’ と ‘`-T hpgl`’ では同様に、多数の Hewlett–Packard のベクトル・フォントが使える。22 種類の Hershey ベクトル・フォントのセット (キリル文字フォントや日本語のフォントを含む) は常に使える。X Window システムでディスプレイに描画するときは、plotutils のプログラムはどれもスケーラブルな X のフォントを使える。

描画を行わない (数学的な計算を行う) プログラムのうち、`spline` はスカラー値またはベクトル値のデータからスプライン関数による補間曲線を求める。通常は三次 (cubic) スプラインかテンション付き指数スプラインを求めるが、`graph` のように特定の環境のもとではリアルタイム・フィルタとして使える。このコマンドは与えられるデータから d 次元空間内に指定される任意の点を通るスプライン曲線を求める (境界条件は自然または周期的のいずれでもよい)。`ode` コマンドは、常微分方程式あるいは常微分方程式系の各方程式が陽関数形式で書かれている時、それを数値積分する。`gnuplot` と似ているように思えるが、`gnuplot` では関数は描画できるが微分方程式はあつかえない。`ode` コマンドはそれを補うことができる。`double` コマンドは、バイナリおよびテキスト形式のデータの入出力ストリームを加工し、変換、スケーリング、データの取捨選択を行うプログラムである。これは開発途中なので、ここでは解説しない。

上述した描画を行うコマンドは内部で `libplot` ライブラリを使っている。`libplot` ライブラリについてはあとで詳しく解説するが、C および C++ のプログラムに線、折れ線、多角形、(真円および楕円の) 円弧、二次および三次のベジェ曲線、円と楕円、点 (ピクセル)、記号および文字列を描画する機能を提供する。点と記号と文字列以外に対しては、塗りつぶし機能が実装されている (線の色と同様に塗りつぶしの色も任意に設定できる)。文字列に使えるフォントは非常にたくさんある。X Window システム、SVG、Illustrator、PostScript、CGM、`xfig` の各ドライバでは 35 種類の PostScript フォント、SVG、Illustrator、PCL 5、HP-GL/2 のの各ドライバでは 45 種類の PCL 5 フォントが使える。PCL 5、HP-GL/2 のドライバはヒューレット・パッカードのベクトル・フォントもサポートする。PNG、PNM、GIF、ReGIS、Tektronix、metafile を含むすべてのドライバで 22 種類の Hershey ベクトル・フォントのセットが使える。

文字列描画の機能も豊富である。上付き、下付き文字が使え、一つの書体に含まれる複数のフォントが一つの文字列中に混在できる。多くの非アルファベット文字も使える。米 Naval Surface Weapons Center の Allen V. Hershey が電子化した 1700 個以上の ‘Hershey glyphs’ (これには興味深い記号が多数含まれている) が `libplot` に組み込まれている。EUC-JP (日本語の Extended Unix Code) エンコードの文字列もあつかえる。これらについては平仮名とカタカナに加えて漢字も使える。漢字は、第一水準の 2965 文字のうちよく使われる 596 文字を含む 603 文字が `libplot` に組み込まれている。

2 graph コマンド

graph は起動されると、コマンドラインで指定する一つまたは複数のデータファイル、または標準入力からデータセットを読み込み、一つのプロット画像を生成する。プロットをどう描画するか、細かく指示するためのコマンドライン・オプションが多数用意されている。指定できるオプションについては Section 2.6 [graph Invocation], page 13 を参照のこと。以下の節に、よく使われるオプションとその例を説明する。

2.1 graph の使い方

デフォルトでは graph コマンドは指定されるファイルから、指定がないときは標準入力からテキスト形式のデータを読み込む。データは以下のように二つの数値の組としてあたえられ、それがデータ点の x および y の値として扱われる。簡単な例を示す。

```
0.0  0.0
1.0  0.2
2.0  0.0
3.0  0.4
4.0  0.2
5.0  0.6
```

データ点は必ずしも各行に一組である必要はなく、また x と y が同じ行にある必要もない。しかし空行はデータセットの区切りとして扱われるので、一つのデータセットの中には空行があってはならない。

ファイル 'datafile' に上の例のデータがあるとして、それを graph でプロットするには、以下のようにする。

```
graph -T ps datafile > plot.ps
```

もしくは以下のようにしても、同じ結果が得られる。

```
graph -T ps < datafile > plot.ps
```

この実行例では、EPS (encapsulated PostScript) 形式のファイル plot.ps が生成される。この形式の画像はプリンタに直接送信して印刷したり、生成したプロットを他の文書に取り込んだり、gv などの PostScript を表示するソフトウェアで画面に表示したり、フリーのドロー系ソフト idraw で編集したりできる。'--page-size' オプション、または環境変数の PAGESIZE でプロット画像を置く用紙のサイズを指定できる。用紙サイズのデフォルトは "letter" で、8.5 in x 11 in である。他に "a4" などの ISO または ANSI に定める用紙サイズが指定できる。詳細は Appendix C [Page and Viewport Sizes], page 159 を参照のこと。

また、以下のようにすると、

```
graph -T svg < datafile > plot.svg
graph -T cgm < datafile > plot.cgm
```

SVG や WebCGM プラグインをインストールしたウェブブラウザで表示できる画像が生成される。また

```
graph -T fig < datafile > plot.fig
```

とするとファイル plot.fig が生成される。このファイルはフリーのドロー系ソフト xfig で編集できる Fig 形式である。さらに、

```
graph -T ai < datafile > plot.ai
```

とすると、Adobe Illustrator 形式のファイル plot.ai が生成される。以下のコマンド

```
graph -T hpgl < datafile > plot.plt
```

では Hewlett–Packard Graphics Language (HP-GL/2) 形式のファイル plot.plt が生成される。これはヒューレット・パッカードのプロッタにそのまま送信して描画することができる。ここで graph -T pcl とすると PCL 5 形式になる。これは LaserJet などのレーザープリンタで印刷できる。

graph -T X オプションを付けて実行すると、X Window の画面上にウィンドウが開かれ、そこにプロットが表示される。コマンドは以下のようになる。

```
graph -T X < datafile
```

この場合、ファイルはなにも生成されない。ウィンドウが表示されるだけである。キーボードから type 'q' を入力するか、ウィンドウ内をマウスでクリックすると、ウィンドウが消える。

graph -T png では PNG 形式の、graph -T pnm では PNM ("portable anymap") 形式の、graph -T gif では疑似 GIF 形式のプロット画像が生成される。フリーの画像表示ソフトウェアである display が使える環境なら、以下の 3 つのコマンド、

```
graph -T png < datafile | display
graph -T pnm < datafile | display
graph -T gif < datafile | display
```

でプロット画像が表示される。

また graph -T tek を使うと Tektronix 4014 端末をエミュレートするデバイスでプロットを表示できる。X Window システムの端末エミュレータである xterm がその一つである。xterm のウィンドウ内で以下のようにすると、ウィンドウが開かれてそこに画像が表示される。

```
graph -T tek < datafile
```

xterm は普段は VT100 端末をエミュレートしているが、xterm のウィンドウ内で 上のコマンドを実行すると、別のウィンドウ ('Tektronix window') が開かれて、その中にプロットが描画される。日本語端末エミュレータの kterm でも、正しくインストールされていれば、xterm と同じはずである。他に、MS-DOS 版の kermit が Tektronix 4014 端末のエミュレート機能を持っている。

同様に ReGIS をサポートしているグラフィック端末エミュレータ上では graph -T regis でプロット画像を表示できる。DECwindows の端末エミュレータ dxterm がその一つである。いくつかの DEC 端末 (特に VT340、VT330、VT241、VT240) でも ReGIS 形式をサポートしている。

graph コマンドの挙動は、実行する環境にあわせて変えることができる。上に述べた PAGESIZE 環境変数は、graph -T svg、graph -T ai、graph -T ps、graph -T cgm、graph -T fig、graph -T pcl、graph -T hpgl に対して有効である。同様に環境変数 BITMAPSIZE が graph -T X、graph -T png、graph -T pnm、graph -T gif のときに有効である。また DISPLAY 環境変数は graph -T X、TERM 環境変数は graph -T tek のとき有効である。この他に、graph -T pcl と graph -T hpgl のときにだけ有効な環境変数がいくつかある。詳細は Section 2.7 [graph Environment], page 25 を参照のこと。なお以下に述べる点は、どの出力形式でも共通の事項である。

デフォルトでは、一つのデータセットの中で連続しているデータ点は、実線の線分で結ばれ、その結果、多角形または折れ線が描かれる。これをここでは単に「線」と呼ぶ。描画に使う線の種類 (線種、線のモード) は '-m' オプションで指定できる。

```
graph -T ps -m 2 < datafile > plot.ps
```

上の例では '-m 2' を指定することで、線種 2 でプロットするよう指示している。プロットが白黒で描画される場合 (デフォルト) は、線は 5 種類の線種のうちのどれか一つで描かれる。

線種 1 から 5 はそれぞれ、実線、点線、一点鎖線、細かい破線、長めの破線、である。5 より上の番号で線種を指定していくと、この 5 種類が順番に繰り返し指定されることになる。'-C' オプションを使うと線の色を指定できる。この場合、線種は 25 種類になる。線種 1 から 5 は赤、緑、青、ピンク (マゼンタ)、水色 (シアン) である。これは全部実線である。線種 6 から 10 は同じ 5 色の点線である。線種 11 から 16 は同じ 5 色の一点鎖線である。25 よりも大きな数字の線種は、25 までを繰り返すことになる。線種 0 は白黒でもカラーでもなく、その線種を指定されたデータを描画しないことを指示する。

線で囲まれた多角形を (薄くした色や原色で) 塗りつぶしたいときは '-q' オプションを使えばよい。たとえば

```
echo 1 1 1 9 9 9 9 1 1 1 | graph -T ps -C -m 1 -q 0.3 > plot.ps
```

このようにすると 4 つの頂点 (1, 1)、(1, 9)、(9, 9)、(9, 1) で囲まれた領域が描かれる。データセットには数値が 10 個、つまりデータ点が 5 つあって、最初と最後の点が同じ (1, 1) であることに注意してほしい。こうすることで、閉じた多角形として四角形が描画される。オプションで線種 1、カラー描画が指定されているので、4 つの各辺は赤い線で描かれる。オプション '-q 0.3' は閉じた多角形の内側を濃さ 30% (の赤) で塗りつぶすことを指定している。濃さに 0 を指定すると白で、1.0 を指定すると線と同じ色 (原色) で塗りつぶされる。濃さに負の値を指定すると塗りつぶしは行われず、透明が設定される (これがデフォルト)。

線の太さ ('width') は塗りつぶしの有無に関係なく '-W' オプションで指定できる。たとえば '-W 0.01' とすると、描画領域の大きさに対する割合で 0.01 の太さの線が描かれる。またプロット上でデータ点を示す記号を、以下のようにして指定できる。

```
graph -T ps -S 3 0.1 < datafile > plot.ps
```

ここでは '-S' の最初の引数 3 が記号の種類を指定している。その次の引数 0.1 は記号のプロットされる大きさを「プロット領域 (plotting box)」に対する割合で指定しており、これは省略してもよい。プロット領域とは、プロットが描かれる正方形の領域のことである。記号は番号で指定され、1 が点、2 がプラス記号、3 がアスタリスク、4 が円 (丸)、5 がバツ印、で以下に続く (Section A.5 [Marker Symbols], page 155 参照)。1 番から 31 番までの記号はどの出力デバイスでも同じである。また記号の描画される色は、その記号と一緒に描かれる線と同じになる。

場合によっては、各データ点を結ぶ線は描きたくないが、各データ点の記号は表示したいことがあるかもしれない。その場合は線種に「負の番号 (negative linemode)」を指定すればよい。線種が負の線は描画されないが、その線上のデータ点の記号は、線種の番号の符号を反転して正にした場合の色で描かれる。たとえば

```
graph -T ps -C -m -3 -S 4 < datafile > plot.ps
```

とすると青い円でデータ点が描かれる。各データ点を結ぶ線は描かれない。'-S' の 2 番目の引数に、記号の大きさを指定することもできる。

graph では、'-a' を使って横軸 (x 軸) の値を自動的に生成することができる。このときはデータファイル中で横軸の値を与えてはならない。データの横軸上の距離は等間隔であると見なされる。'-a' には、最初のデータ点の横軸の値と、データの間隔を指定する。指定のない場合は、それぞれ 1.0 と 0.0 を指定したことと同じになる。例えば次のコマンド、

```
echo 0 1 0 | graph -T ps -a > plot.ps
```

は以下と全く同じプロットを描画する。

```
echo 0 0 1 1 2 0 | graph -T ps > plot.ps
```

graph に '-I e' オプションを指定することで、エラーバーを描画できる。この場合データセットでは各データ点につき、2 つ (x、y) ではなく、3 つ (x、y、$error$) の数値が必要にな

る。各データ点で、データ点の上下に入力された数値の幅を持つ垂直なエラーバーが描かれる。'-S' で記号を指定すればエラーバーとともにデータ点上に記号が描かれる。記号の種類は、一つのデータセットの各点で同じになる。'-a' オプションも '-I e' と組み合わせて使える。その場合、データセット中には横軸 (x 軸) の値を含めてはならない。

デフォルトでは、x 軸と y 軸の範囲、各軸上のラベル (文字列) の付く目盛りの間隔は自動的に計算されるが、以下のように '-x' および '-y' オプションを使えば、指定することもできる。

```
echo 0 0 1 1 2 0 | graph -T ps -x -1 3 -y -1 2 > plot.ps
```

これにより x 軸は -1 から 3 の範囲に、y は -1 から 2 の範囲に広げられる。graph のデフォルトでは各軸で数値の付けられる目盛りは 6 カ所程度だが、'-x' または '-y' の 3 つめの引数で目盛りの間隔を指定できる。たとえば、'-y -1 2' ではなく '-y -1 2 1' を指定すると、y 軸上の -1、0、1、2 のところで目盛りに数値のラベルが描かれる。デフォルトでは graph が自動的に -1、-0.5、0、0.5、1、1.5、2 に数値を描くように設定する。一般的に、3 つめの引数が指定されたときはその値の整数倍のところに数値が描かれる。

対数プロットをしたいときは、'-l' を指定する。たとえば、

```
echo 1 1 2 3 3 1 | graph -T ps -l x > plot.ps
```

とすると、x 軸は対数でプロットされる。y 軸は通常のままである。両軸とも対数にしたいときは '-l x -l y' を指定する。デフォルトでは対数軸の上下限は 10 のべき乗に合わせられ、軸上の目盛りも 10 のべき乗とその整数倍の点に描かれる。10 のべき乗の点では目盛りに数値が描かれる。描画範囲が 5 桁以上の範囲になるときは、10 のべき乗の整数倍の点の目盛りは省かれる。

対数プロットの座標軸の範囲が狭くて、目盛りの数を増やしたい場合は、目盛りの間隔をオプションで指定する。たとえば '-l x -x 1 9 2' とすると、x 軸の描画範囲は 1 から 9 になる。数値は 2 の整数倍のところ、2、4、6、8 の目盛りに描かれる。

x 軸および y 軸にはそれぞれ、'-X' および '-Y' オプションでラベルを付けることができる。たとえば、

```
echo 1 1 2 3 3 1 | graph -T ps -l x -X "A Logarithmic Axis" > plot.ps
```

とすると、一つ前のプロット例の対数軸にラベルを付ける。デフォルトでは '-Q' を付けない限り、y 軸のラベルは (もしあれば) 90 度回転して描かれる (X Window システムには、新しくもて古くてもラベルの回転をサポートしていないものがあり、その場合は '-Q' オプションを明示的に与える必要がある)。'-L' オプションで以下のようにすると、プロット全体のタイトルのラベルを指定できる。

```
echo 1 1 2 3 3 1 | graph -T ps -l x -L "A Simple Example" > plot.ps
```

こうすると、図の上部にタイトルのついた図が生成される。

x 軸および y 軸のラベルの文字の大きさを '-f' オプションで、図のタイトルの文字の大きさを '--title-font-size' オプションで指定できる。たとえば、

```
echo 1 1 2 3 3 1 | graph -T ps -X "Abscissa" -f 0.1 > plot.ps
```

とすると、x 軸のラベルと目盛りの数値が非常に大きく (プロットの描かれる正方形の領域に対する割合で 0.1) 描かれる。

'-X'、'-Y'、'-L' オプションで指定されるラベルに使われるフォントは '-F' オプションで指定できる。たとえば '-F Times-Roman' とするとラベルのフォントがデフォルト (Helvetica、ただし '-T png'、'-T pnm'、'-T gif'、'-T pcl'、'-T hpgl'、'-T tek' のときは異なる) の代わりに Times-Roman が使われる。フォント名では大文字、小文字を区別しないので、'-F

times-roman' でも同じである。使えるフォントは、35 種類の PostScript の標準フォント (ただし graph -T png、graph -T pnm、graph -T gif、graph -T pcl、graph -T hpgl、graph -T tek 以外の場合)、45 種類の PCL 5 フォント (ただし graph -T svg、graph -T ai、graph -T pcl、graph -T hpgl の場合のみ)、多数のヒューレット・パッカードのベクトル・フォント (ただし graph -T pcl、graph -T hpgl の場合のみ)、および 22 種類の Hershey ベクトル・フォントである。Hershey フォントにはロシア語のキリル文字 HersheyCyrillic と、日本語の HersheyEUC がある。利用できるフォントについての詳細は Section A.1 [Text Fonts], page 136 を参照のこと。plotfont コマンドを使えば、利用できるフォントのキャラクタ・マップが得られる (Chapter 6 [plotfont], page 51 参照)。

'-X'、'-Y'、'-L' オプションで描かれるラベルのフォントは、指定の仕方が複雑に感じられるかもしれない。また上述の例に加えて、上付き、下付き、ルート記号、フォントの切り替えが指定できるが、これまでに出てきた例には示していない。詳細は Section A.4 [Text String Format], page 144 を参照のこと。

ここまでの例では、グリッドはデフォルトのままであった (目盛りとラベルが下および左に付いていて、プロットは正方形)。これもオプションを使っていくつかの種類が設定できる。'-g 0'、'-g 1'、'-g 2'、'-g 3' を指定すると、この順に装飾が増える。それぞれ、グリッド省略、目盛りと数値の付いた一組の軸、目盛りと数値の付いた正方形、目盛りと数値の付いた正方形とグリッド、である。見て分かる通り、デフォルトは '-g 2' である。'-g 4' も指定できるが、これは他のと違って、原点を通る二本の軸を描画する。このオプションはデータ点の x または y の値が正および負の両方にわたっているときに使う。

2.2 プロットの回転、および長方形のプロット

プロットの形を長方形にしたいときや、プロットの位置を画面や用紙上で指定したいときは、以下のようにすればよい。

```
graph -T ps -h .3 -w .6 -r .1 -u .1 < datafile > plot.ps
```

ここで '-h' と '-w' オプションはプロット領域の高さと幅を指定している。'-r' と '-u' オプションはプロット領域の置かれる位置 (プロット領域の左下隅が、どれだけ上および右か) を指定している。それぞれの単位はプロット領域の大きさ (正方形) に対する割合である。デフォルトでは、プロット領域の大きさは 0.6 で、場所は上方向と右方向にそれぞれ 0.2 ずつ動いた場所である。である。したがって、上の例では高さが通常の 1/2 のプロットが生成される。また通常に比べると、画像は少し下かつ左寄りに描画される。

一部のコマンドライン・オプションでは、サイズをプロット領域に対する割合で指定する。たとえば '-S 3 .01' ではデータ点に描かれる記号に対して、種類は 3 番でフォントサイズは 0.01、つまりプロット領域の縦あるいは横のどちらか短い方の 0.01 倍を指定する。'-h' または '-w' でプロット領域の大きさが変更された場合、他のオプションで指定されているさまざまなサイズも、それにつれて変更される。おそらくはこれが正しい、分かりやすい動作と言えよう。

プロット画像を 90 度反時計回りに回転させるには、graph のコマンドラインで '--rotation 90' オプションを指定する。そこに '--rotation 180' を指定すると、逆さまのプロットになる。好きな角度を指定できるが、0、90、180、270 以外の角度は前衛芸術家くらいしか使わないだろう。'--rotation' は '-h'、'-w'、'-r'、-u と組み合わせて使うこともできる。その場合、まずプロット領域が回転させられる。これは '--rotation' は描画領域全体の回転する角度を指定し、他のオプションはその描画領域内でのプロット領域の位置を指定するからである。この二つは、論理的に意味が異なる指示である。

描画領域 (グラフィクス・ディスプレイ、または 'viewport') は抽象的な概念である。graph -T X ではこれは X display の画面に開かれたウィンドウである。graph -T pnm および graph -T gif では正方形あるいは長方形のビットマップである。この 3 つの場合は描画領域の大きさは '--bitmap-size' オプションまたは BITMAPSIZE 環境変数で指定できる。graph -T tek では描画領域は Tektronix 端末の画面の中央に位置する正方形の領域である (Tektronix 端末の画面は縦横比 3:4 の横長である)。graph -T regis では描画領域は ReGIS 端末の画面の中央に位置する正方形の領域である。graph -T ai、graph -T ps、graph -T pcl、graph -T fig ではデフォルトでは、8.5 in x 11 in の用紙 (US letter サイズ) の中央の 8 インチ四方の領域である。graph -T hpgl でも 8 インチ四方の領域だが、置かれているのは中央ではない。graph -T svg および graph -T cgm ではデフォルトでは 8 インチ四方の領域だが、出力ファイルがウェブブラウザで表示されるときには任意の大きさに拡大・縮小される。

graph -T svg、graph -T ai、graph -T ps、graph -T cgm、graph -T fig、graph -T pcl、graph -T hpgl ではデフォルトの描画領域のサイズが用紙の大きさによって決まるが、これは '--page-size' オプションまたは PAGESIZE 環境変数で指定できる。たとえば用紙サイズを "a4" に設定すれば A4 サイズ (21 cm x 29.7 cm) にあわせて描画領域が設定され、適切な大きさの画像が生成される。描画領域の大きさは、縦、横、あるいはそのどちらかを指定できる。たとえば"letter,xsize=4in" や "a4,xsize=10cm,ysize=15cm" といった指定ができる。描画領域の高さ、横幅には、負の値も指定できる (そのときは鏡像になる)。

描画領域の用紙上のデフォルトの位置に対する相対的な位置をオプションで指定できる。たとえばまず "letter,yoffset=1.2in" や "a4,xoffset=−5mm,yoffset=2.0cm" のようにすると、用紙の大きさを指定できる。そして描画領域の左下隅の位置を、用紙の左下隅に対する相対的な位置で指定する。たとえば "letter,xorigin=2in,yorigin=3in" や "a4,xorigin=0.5cm,yorigin=0.5cm" といった指定ができる。

上述したオプションは、組み合わせて使うことができる。しかし graph -T svg および graph -T cgm では "xoffset"、"yoffset"、"xorigin"、"yorigin" は無効である。SVG と WebCGM 形式では、画像が最終的にウェブブラウザ上のどこに表示されるかは、画像そのものからは指定できないからである。"xsize" と "ysize" オプションは、画像のデフォルトサイズを指定するものと解釈される (ウェブブラウザ上では画像の大きさは任意に拡大・縮小されるので、この指定は単にデフォルト値を与えるにすぎない)。

用紙サイズおよび描画領域の大きさについての詳細は、Appendix C [Page and Viewport Sizes], page 159 を参照のこと。

2.3 複数のデータセット

複数のデータセットを、まとめて一つのプロットにしたいことは多い。そのために、各データセットごとに違う線種を使って区別を付けたり、違う種類の記号でプロットすることができる。

以下に、少しだけ複雑な例を挙げる。たとえば、実験観測データのファイルと、数値が細かい刻みで連続している、データの理論値を示すファイルがあるとする。後者のファイルは、それ自身で一つのデータセットである。理論値の方は連続するデータ点を線で結んで、理論曲線としてプロットしたいが、前者の観測データは線で結ばず、記号でデータ点を示すだけにしたいとする。実際には各点のデータ点には記号が描かれる。

この例では他と同様、データセットのプロットに 7 種類の属性を指定して、データセットをどのように描画するかを指示する。属性には以下の種類があり、コマンドライン・オプションで指定できる。

1. 色の有無 (color/monochrome)

2. 線種 (linemode)
3. 線幅 (linewidth)
4. 記号の種類 (symbol type)
5. 記号の大きさ (symbol size)
6. 記号のフォント名 (symbol font name)
7. 塗りつぶしの濃さ (fill fraction)

色の有無 (color/monochrome のどちらか。両方を同時に指定することはできない) は、見た通りの意味である。'-C' オプションで指定できる。指定はコマンドライン上でトグル動作になる。線種 (linemode) は連続するデータ点を結ぶ線分の種類である。これは '-m' オプションで指定できる。線種に 0 番を指定するとまったく描かないという意味になる。線幅 (linewidth) は線の太さである。これは '-W' オプションで指定できる。記号の種類 (symbol type) と記号の大きさ (symbol size) は各データセットのデータ点に描かれる記号を指定する。これは '-S' オプションで指定できる。記号のフォント名 (symbol font name) では、32 番より後の記号をどのフォントから取るかを指定する。これはなにか図章的な記号ではなく、文字を指定することになる。このオプションは '--symbol-font-name' オプションと組み合わせて、'-S' で特殊な記号などを指定して使う。最後に、データ点を結んで多角形として描かれるデータセットでは、その多角形の内側を原色や薄い色で塗りつぶすことができる。「塗りつぶしの濃さ」は '-q' オプションで指定できる。濃さの値が負の場合は塗りつぶしを行わず、透明化するという指示になる。0 のときは白で、1.0 のときは原色で塗りつぶされる。

この 7 種類の属性で、データセットのプロットの描き方を指定できる。またファイルからの読み込み方もデータセットごとに指定できる。たとえば一つのファイルの中のデータがエラーバーのデータかそうでないか、などである。もしファイル中のデータをエラーバーのデータとして扱う場合、ファイル名の前に '-I e' オプションを指定して、それを示さなければならない ('-I' オプションはそれに続くファイルの入力の形式を指定する)。

3 つの異なるファイルのデータを、どうやって一緒にプロットするかを以下に示す。

```
graph -T ps -m 0 -S 3 file1 -C -m 3 file2 -C -W 0.02 file3 > output.ps
```

file_1 のデータセットは線種 0 番で描かれる。つまり各データ点は結ばれないが、記号 3 (アスタリスク) が各データ点に描かれる。file_2 のデータセットはカラーで、線種 3 番で描かれる。カラーの場合、線種 3 番は青い実線である。コマンドラインでの二つ目の '-C' は、file_3 ではカラーモードをオフにする、という指定になる。3 つめのデータセットは描画領域の縦、横のサイズのいずれか短い方の 0.02 倍の太さの黒い線でデータ点が結ばれる。

'-q' や '-I' オプションを各ファイルの前に付けると、上の例はもっと複雑になる。またコマンドラインでは '-x'、'-y' で座標軸の範囲を指定することも実際にはある。そういう、各データセットに対してではなく、プロット全体に対するオプションは、最初に指定されるファイルよりも前に置くべきである。たとえば出力形式の指定は以下のように、

```
graph -T ps -x 0 1 0.5 -m 0 -S 3 file1 -C -m 3 file2 > output.ps
```

とする。コマンドラインでは、標準入力を表す特殊なファイル名として '-' が使える。これを使うと、一部はファイルから、他の一部を他のプログラムから graph へのパイプで渡されるデータから入力して、プロットを生成することができる。

一つのファイルに複数のデータセットが入っていてもよい。その場合、コマンドラインでそのファイルの前に指定されるオプションは、そのファイル中のデータセットすべてに適用されるが、例外が二つある。デフォルトではデータセットごとにラインモードが一つずつ後のものになる。通常は、これが便利なはずである。たとえば以下のようにすると、

```
graph -T ps -m 3 file1 > output.ps
```

file_1 の最初のデータセットは線種 3 で、二つ目のデータセットは線種 4 で、のようになる。以下のようにして線種の指定を省くと実際、

```
graph -T ps file1 file2 ... > output.ps
```

データセットの入力される順にその線種が 1、2、 . . . に設定される。もし違うようにしたいときは、'-B' オプションでこれをオフにするか、トグルさせることができる。

データファイルの中で直接、線種と記号の種類を指定することもできる。その場合コマンドラインではなく、ファイル中に指定子を使って記述する。例えば以下の行が、テキスト形式の入力ファイル中にあるとき、

```
#m=-5,S=10
```

これは、これの後のデータセットの線種を $\#-5$ に切り替え、記号の種類を 10 番にする。将来的には、7 種の属性をすべてこの方法で指定できるようにする予定である。

2.4 多重プロット

複数のプロットを一つのページあるいは描画領域の中に描画する、つまりプロットを合成することをここでは多重プロットと呼ぶ。小さなプロットを大きなプロットにはめ込む、または複数のプロットを並べる、といったことである。

graph は、多数の小さなプロットを任意の数だけ多重プロットできる。このとき graph は小さな各プロットを、それぞれの「仮想描画領域」に描画する。通常のプロットでは、仮想描画領域と物理的な実際のディスプレイ上の本当の描画領域は同じである。多重プロットのときは仮想描画領域はより小さな正方形の領域である。以下の 2 つの例でその考え方を示す。

```
graph -T X datafile1 --reposition .35 .35 .3 datafile2
```

この例では datafile1 は通常通りに描かれる。'--reposition' オプションで datafile2 を描く仮想描画領域を指定している。'--reposition' オプションでの仮想描画領域の位置の指定は、物理的なディスプレイ上の、全体プロットの描画領域は正方形で、左下隅が (0.0, 0.0) で右上隅が (1.0, 1.0) であるとして相対的な値を指定する。上の例では仮想描画領域のサイズは一辺が 0.3 の正方形で、左上隅が物理的なディスプレイ上の描画領域の (0.35,0.35) に置かれる。そうして二つ目の小さなプロットが物理的なディスプレイ内の、最初の大きなプロット内に置かれる。

物理的なディスプレイ上の描画領域のサイズと場所が '-w'、'-h'、'-r'、'-u' で指定できるが、これらのオプションは仮想描画領域のサイズと場所を指定するのにも使える。たとえば、

```
graph -T X datafile1 --reposition .35 .35 .3 -w .4 -r .3 datafile2
```

とすると、二つ目の小さなプロットをかなり変更した多重プロットが行われる。そのプロット領域の横幅は仮想描画領域に対する割合で 0.4 しかない。しかしプロット領域は仮想描画領域の中央に置かれる。それはプロット領域の左端と仮想描画領域の左端の距離が、仮想描画領域の横幅の 0.3 倍だからである。

最初の小さなプロットの時以外は、各プロットを描画する前にプロット領域を囲む「消去領域」が削除される (これは白、または背景色があればそれで塗りつぶされる)。これにより小さなプロットが重なってしまったりして、多重プロットの結果が醜くなったりするのを避けている。デフォルトでは消去領域の縦横の大きさは、それぞれの小さなプロット領域の 1.3 倍である。これは、そのプロットが小さくて、かつすでにプロットがあるところに後から描画される場合にはよいが、複数の小さなプロットを縦横に並べるような多重プロットをしたい場合にはよくない。そういったときは '--blankout' オプションでその大きさを調整

できる。たとえば '--blankout 1.0' と指定すると消去領域と描画領域が同じ大きさになる。'--blankout 0.0' とすると余白を取らなくなる。消去領域は複数の小さなプロットについてそれぞれ独立に指定できる。

多重プロットされるそれぞれの小さなプロットは、それぞれ独立している。通常のオプション ('-m'、'-S'、'-x'、'-y' など) はそれぞれの小さなプロットに、それぞれ指定できる。それぞれの小さなプロットに対するオプションは、graph のコマンドラインで、'--reposition' オプションのすぐ後ろに指定しなければならない。それぞれの小さなプロットは一つ、あるいは複数のデータセットをプロットできる。それぞれの小さなプロットのデータファイルは (もしあるなら)、'--reposition' のすぐ前に指定する。

2.5 読み込めるデータ形式

graph はデフォルトではテキスト形式のデータを読み込むようになっている。しかし 3 種類のバイナリ形式 (単精度浮動小数点、倍精度浮動小数点、整数) も読み込むことができる。バイナリ形式で読み込むときはその形式を '-I f'、'-I d'、'-I i' オプションで指定する。それぞれ単精度、倍精度、整数である。

バイナリ形式には二つの利点がある。ひとつは、graph の実行がかなり速くなることである。これはテキスト形式のデータをバイナリに変換する必要がなくなるからである。もうひとつは入力ファイルのサイズがずっと小さくなることである。もし大量のデータを扱うなら、バイナリ形式の方が保存する記憶容量、実行時間ともに有利である。

たとえば C 言語のプログラムから以下のようにして単精度実数を出力するとする。

```
#include <stdio.h>
void write_point (float x, float y)
{
  fwrite(&x, sizeof (float), 1, stdout);
  fwrite(&y, sizeof (float), 1, stdout);
}
```

こうして作られたデータファイルは、以下のようにするとプロットできる。

```
graph -T ps -I f < binary_datafile > plot.ps
```

一つのバイナリ形式のファイルに複数のデータセットが入っていてもよい。その場合、そのバイナリ形式で表現できる数値の最大値がデータセットの区切りになる。単精度実数の場合は FLT_MAX、倍精度実数の場合は DBL_MAX、整数の場合は INT_MAX で表される数値がそれになる。多くの計算機環境では FLT_MAX は約 3.4×10^{38}、DBL_MAX は 1.8×10^{308}、INT_MAX は $2^{31} - 1$ である。

複数のファイルからデータを読み込むときに、各ファイルの形式は異なっていてもよい。たとえば、

```
graph -T ps -I f binary_datafile -I a ascii_datafile > plot.ps
```

とすると binary_datafile からは 'f' (単精度実数) で、ascii_datafile からは 'a' (普通のテキスト形式) で読み込む。

現在の所、エラーバーのデータはバイナリ形式では読み込めない。エラーバーのデータはテキスト形式で '-I e' オプションを指定して graph に入力せねばならない。

graph は、gnuplot が出力する「表形式 ('table' format)」のテキストデータも読み込める。これは '-I g' で指定する。これにはデータセットが複数含まれていてもよい。

以上をまとめると、読み込めるデータ形式は 6 種類である。'a' (普通のテキスト形式)、'e' (エラーバーのデータを含むテキスト形式)、'g' (gnuplot が出力する「表形式」のテキスト

データ)、'f' (単精度実数のバイナリ形式)、'd' (倍精度実数のバイナリ形式)、'i' (整数のバイナリ形式) である。入力ファイルはこの６つのうちのどれかであればよい。

2.6 graph のコマンドライン・オプション

graph プログラムは、コマンドラインで指定されたファイルまたは標準入力から、一つまたは複数のデータセットを読み込み、プロットを描画する。プロット画像の形式の種類は '-T' オプションで指定する。

デフォルトでは graph はコマンドラインで指定されたファイルから、テキスト形式のデータを読み込む。データは二つの数値の組で、それぞれデータ点の x および y 座標であると解釈される。コマンドラインでファイルが指定されていないか、または '-' というファイル名が指定されたときは、標準入力からデータが読み込まれる。出力は、'-T X' オプションが指定された時以外は、標準出力に書き出される。'-T X' オプションが指定されているときは、プロットは X Window の画面上に描画され、画像データは標準出力には書き出されない。

プロットの細部を指示するコマンドライン・オプションが多数用意されているが、そういったオプションを使うときには、コマンドライン上でのオプションと入力ファイルの指定される順番が重要である。ファイル名よりも前に指定されているオプションだけが、そのファイルのデータのプロットに適用される。

以下にすべてのオプションを解説する。引数を取るオプションでは、説明の最初にカッコ書きでその引数の型とデフォルト値を示している。オプションは以下の５種類に分けられる。

1. プロット全体を制御するオプション (Section 2.6.1 [Plot Options], page 13 参照)。
2. 一つのプロット内の各データセットの読み込み、描画を制御するオプション (Section 2.6.2 [Dataset Options], page 20 参照)。
3. 多重プロット (複数のプロットを一つにまとめる) を制御するオプション (Section 2.6.3 [Multiplot Options], page 23 参照)。
4. graph の生出力のデータ ('-T' で何も指定されないときの出力形式) に関するオプション (Section 2.6.4 [Raw graph Options], page 24 参照)。
5. 情報を表示するためのオプション ('--help' など) (Section 2.6.5 [Info Options], page 24 参照)。

2.6.1 プロット全体を制御するオプション

以下のオプションのは、プロットの描画全体に対する制御を行う。指定されるときは、多くの場合コマンドライン上で一回だけであり、どのデータ・ファイルよりも前に指定される。もし多重プロットを行うときは、('-T' オプションを除いて) 複数回指定されうる。そのときは、２回目またはそれ以降のオプションは '--reposition *x y*' オプションのすぐ後ろに置かなければならない。

'-T *type*'
'--output-format *type*'
: (文字列、デフォルトは "meta") 出力形式を *type* に設定する。これには "X"、"png"、"pnm"、"gif"、"svg"、"ai"、"ps"、"cgm"、"fig"、"pcl"、"hpgl"、"regis"、"tek"、"meta" のいずれかの文字列を指定する。それぞれ、X Window システム、PNG 形式、portable anymap (PBM/PGM/PPM) 形式、疑似 GIF 形式、XML ベースの Scalable Vector Graphics 形式、Adobe Illustrator 形式、idraw で編集できる PostScript 形式、ウェブ用のベクトル画像形式である WebCGM 形式、xfig で編集できる形式、ヒューレット・

パッカードの PCL 5 プリンタ言語、ヒューレット・パッカード・グラフィクス言語 (デフォルトでは HP-GL/2)、DEC で開発された ReGIS (remote graphics instruction set) 形式、Tektronix 形式、特定のデバイスに依存しない GNU graphics metafile 形式である。'`--display-type`' は古いので '`--output-format`' を使うべきである。

'`-E` *x*|*y*'
'`--toggle-axis-end` *x*|*y*'
指定した軸を、通常描かれる場所から、描画領域の反対側に移動する。たとえば '`-E y`' を指定すると y 軸がプロットの左側 (これがデフォルト) ではなく、右側に描かれる。同様に '`-E x`' とすると x 軸が描画領域の下部ではなく、上部に描かれる。x 軸を上部に描くと、プロットのタイトルラベルが表示されなくなる (ラベルの場所がなくなるため)。

'`-f` *size*'
'`--font-size` *size*'
(実数、デフォルトは 0.0525) 座標軸と目盛りのラベルに使う文字列のフォントサイズを *size* に指定する。数値は、プロット領域の縦または横の長さの小さい方に対する割合として指定する。

'`-F` *font_name*'
'`--font-name` *font_name*'
(文字列、デフォルトは "Helvetica"、ただし `graph -T pcl` では"Univers"、`graph -T png`、`graph -T pnm`、`graph -T gif`、`graph -T hpgl`、`graph -T regis`、`graph -T tek`、生の `graph` の場合は "HersheySerif") 座標軸と目盛りのラベル、およびプロットのタイトル (があればそれ) に使うフォントを *font_name* に指定する。プロットのタイトルのフォントは '`--title-font-name`' オプションによる指定 (後述) が優先される。フォント名では大文字と小文字は区別されない。もし指定されたフォントが使えない場合、デフォルトのフォントが使われる。使えるフォントは、'`-T`' オプションで何を指定するかによって異なる。すべてのフォントのリストは Section A.1 [Text Fonts], page 136 を参照のこと。`plotfont` ユーティリティ・プログラムで、使えるフォントのキャラクタ・マップを表示することができる。Chapter 6 [plotfont], page 51 参照。

'`-g` *grid_style*'
'`--grid-style` *grid_style*'
(0...4 の範囲の整数、デフォルトは 2) 引数 *grid_style* でグリッドの描き方を指定する。0 から 3 までは、だんだんと描かれるものが増えて行く。4 では違った描き方になる。

0. グリッド、目盛り、ラベルを描かない。
1. 両座標軸と、目盛りとラベルを描く。
2. 描画領域の周囲の四辺と目盛りとラベルを描く。
3. 描画領域の周囲の四辺と目盛りとラベルを描く。グリッドも描かれる。
4. 両座標軸を原点を通るように描き、目盛りとラベルも描く。

'`-h` *height*'
'`--height-of-plot` *height*'
(実数、デフォルトは 0.6) 描画領域 (または、多重プロットの場合は仮想描画領域) の縦のサイズに対する割合として、プロット領域の縦のサイズを指定する。

その値が 1.0 のときは、プロット領域の大きさは指定できる最大サイズになる。目盛りとラベルはプロット領域の外側なので、普通は 1.0 よりも小さな値にする。

‘-H’
‘--toggle-frame-on-top’
プロットのすぐ上に、下に描かれているのと同じフレームを描くよう指示する。これはプロットの線が太かったり記号が大きかったりして、他のデータセットが隠れてしまうときに指定する。

‘-k *length*’
‘--tick-size *length*’
(実数、デフォルトは .02) 両座標軸上の目盛りの長さを *length* で指定する。その値が 1.0 のときは、プロット領域の縦か横の短い方と同じ長さになる。*length* に負の値を指定すると、目盛りはプロットの内側向きではなく、外側に向けて描かれる。

‘-K *clip_mode*’
‘--clip-mode *clip_mode*’
(整数、デフォルトは 1) クリップ・モードを *clip_mode* にする。クリップ・モードは、データ点が線で結ばれているが、塗りつぶされてはいないときにだけ有効である (塗りつぶされた領域は別の方法でクリッピングされる)。

３種類のクリップ・モードが、0、1、2 で指定できる。これは gnuplot のクリップ・モードの番号と同じである。モード 0 では、二つのデータ点を結ぶ線は、どちらの点もプロット領域の中にある時にだけ結ばれる。モード 1 では、どちらか一つが描画領域内にあれば線が結ばれる。モード 2 では、どちらも描画領域の外にある場合でも、線が結ばれる。しかしいずれの場合でも、結ばれた線のプロット領域の中にある部分だけが描かれる。

‘-l *x|y*’
‘--toggle-log-axis *x|y*’
指定された座標軸を対数軸にするかどうかを切り替える。デフォルトでは、どちらも対数軸ではない。

‘-L *top_label*’
‘--top-label *top_label*’
(文字列、デフォルトは空) 指定された文字列 *top_label* (プロットのタイトル) をプロットの上部に置く。文字列にはエスケープ・シーケンスが使える (Section A.4 [Text String Format], page 144 参照)。‘--title-font-size’ オプションでフォントのサイズを指定できる。フォントは通常は、‘-F’ オプションで指定される目盛りや座標軸のラベルのフォントと同じだが、‘--title-font-name’ でこの文字列のフォントを別に指定できる。

‘-N *x|y*’
‘--toggle-no-ticks *x|y*’
指定された座標軸での目盛りの有無を切り替える。‘-g *grid_style*’ での指定 1、2、3、4 でも目盛りとそのラベルの有無を制御できる。

‘-Q’
‘--toggle-rotate-y-label’
y 軸のラベル (‘-Y’ オプションで指定される) を置く向きを、水平、垂直で切り替える。デフォルトではラベルは回転していて y 軸と平行になる。しかし一部の

出力機器 (昔の X Window や、新しくてもバグのあるもの) ではラベルを回転させられない。そんなときは '-T X' するときに '-Q' も指定する必要があるかもしれない。

'-r *right*'
'--right-shift *right*'
(実数、デフォルトは 0.2) プロット領域を右に移動する。指定する移動量 *right* は描画領域 (多重プロットのときは仮想描画領域) の横幅に対する割合である。これによってプロット領域の左側にあける余白を制御できる。この値を 0.5 にすると描画領域の左半分が余白になる。その余白に目盛りとラベルが描かれることになる。

'-R *x*|*y*'
'--toggle-round-to-next-tick *x*|*y*'
指定された座標軸において、プロット範囲の上下限が数値の付けられている目盛りの整数倍に一致させるかどうかを切り替える。

このオプションは、'-x' および '-y' オプションのどちらか、または両方を指定するときに特に便利である。もしどちらも指定されない場合、「整数倍」でプロット範囲が決められる。

'-s'
'--save-screen'
画面を消さない。このオプションを指定すると、graph が描画をはじめるときに、前に描いたプロットを消さないようになる。

このオプションは graph -T tek と生の graph 形式でのみ有効である。Tektronix 端末とそのエミュレータでは、一度描いた図形はそのままずっと画面上に残っている。なので、graph -T tek -s を繰り返し実行することで、多重プロットを組みたてることができる。

'-t'
'--toggle-transpose-axes'
縦軸と横軸を入れ替えるかどうかを切り替える。各軸に対して指定したオプションは、それぞれ入れ替わった軸に対して適用される。これはデータが (y, x) の形式で読み込まれ、'-x' や '-X' オプションがx 軸ではなく y 軸に対して適用されてしまうような場合に使える。もし '-I e' が指定されていれば、データはエラーバーの値を含めて読み込まれ、エラーバーの向きも垂直と水平が切り替わる。

'-u *up*'
'--upward-shift *up*'
(実数、デフォルトは 0.2) プロット領域を上に移動する。指定する移動量 *up* は描画領域 (多重プロットのときは仮想描画領域) の縦の高さに対する割合である。これによって描画領域の下側にあける余白を制御できる。この値を 0.5 にすると描画領域の下半分が余白になる。その余白に目盛りとラベルが描かれることになる。

'-w *width*'
'--width-of-plot *width*'
(実数、デフォルトは 0.6) プロット領域の横幅を指定する。指定する横幅の長さは *width* は描画領域 (多重プロットのときは仮想描画領域) の横幅に対する割合

である。その値が 1.0 のときは、描画領域の横幅と同じになる。目盛りとラベルはプロット領域の外側なので、普通は 1.0 よりも小さな値にする。

‘-x [*lower_limit* [*upper_limit* [*spacing*]]]’
‘--x-limits [*lower_limit* [*upper_limit* [*spacing*]]]’
(実数) *lower_limit* と *upper_limit* で x 軸の範囲を、*spacing* で軸上でラベルの付く目盛りの間隔を指定する (この指定はあってもなくてもよい)。3 つの引数のうちどれか一つでも欠けている場合、または入力が ‘-’ (一つのハイフン) で与えられた場合、欠けている部分はデータから自動的に計算される。graph をリアルタイムでプロットするフィルタとして使うときは、*lower_limit* と *upper_limit* の両方を指定しなければならない。

デフォルトでは、この指定を他より優先する。しかし ‘-R x’ が指定されたときは、指定された値にもっとも近い整数値の倍数として、ラベルの付く目盛りの間隔が決められる。下限は指定された値よりも下の、上限は上の整数値になる。

‘-X *x_label*’
‘--x-label *x_label*’
(文字列、デフォルトは空) x 軸のラベルとして文字列 *x_label* を指定する。文字列にはエスケープ・シーケンスが使える (Section A.4 [Text String Format], page 144 参照)。‘-F’ と ‘-f’ で使われるフォントとそのサイズを指定できる。

‘-y [*lower_limit* [*upper_limit* [*spacing*]]]’
‘--y-limits [*lower_limit* [*upper_limit* [*spacing*]]]’
(実数) x 軸 の場合 (上述) と同様に、*lower_limit* と *upper_limit* で y 軸の範囲を、*spacing* で軸上でラベルの付く目盛りの間隔を指定する。(この指定はあってもなくてもよい)。graph をリアルタイムでプロットするフィルタとして使うときは、*lower_limit* と *upper_limit* の両方を指定しなければならない。

デフォルトでは、この指定を他より優先する。しかし ‘-R y’ が指定されたときは、指定された値にもっとも近い整数値の倍数として、ラベルの付く目盛りの間隔が決められる。下限は指定された値よりも下の、上限は上の整数値になる。

‘-Y *y_label*’
‘--y-label *y_label*’
(文字列、デフォルトは空) y 軸のラベルとして文字列 *y_label* を指定する。文字列にはエスケープ・シーケンスが使える (Section A.4 [Text String Format], page 144 参照)。文字列は 90 度回転して、座標軸と平行に描画されるが、‘-Q’ オプションが指定されているときは回転されない (昔の X Window システムではラベルの回転がサポートされていないことがある。なので ‘-T X’ を指定するときは ‘-Q’ オプションを使うとよいことがある)。‘-F’ と ‘-f’ で使われるフォントとそのサイズを指定できる。

‘--bg-color *name*’
(文字列、デフォルトは "white") 背景色を *name* に指定する。これは graph -T X、graph -T png、graph -T pnm、graph -T gif、graph -T cgm、graph -T regis、graph -T meta のときにだけ有効である。無効な名前が色名に指定されたときは、デフォルトの色が使われる。使える色名に付いては Appendix B [Color Names], page 158 を参照のこと。環境変数 BG_COLOR でも同様に背景色を指定できる。

‘-T png’ か ‘-T gif’ オプションのどちらかが使われているときは、環境変数 TRANSPARENT_COLOR に背景色を指定することで、透過 PNG または透過疑似

GIF 形式の画像を生成できる。Section 2.7 [graph Environment], page 25 参照。‘-T svg’ か ‘-T cgm’ オプションのどちらかが使われているときは、背景色に "none" を指定することで、背景色のない画像を生成できる。

‘--bitmap-size *bitmap_size*’
(文字列、デフォルトは "570x570") 描かれる描画領域のサイズを、ピクセル単位で *bitmap_size* に設定する。これは graph -T X、graph -T png、graph -T pnm、graph -T gif のときにだけ有効である。これらの出力形式では描画領域のサイズがピクセルで表されるからである。環境変数 BITMAPSIZE でも同様に指定できる。

graph -T X のときは描画のために開かれたウィンドウが描画領域になる。X Window の画面上でのこのウィンドウの位置も、コマンドライン・オプションで指定できる。たとえば *bitmap_size* を "570x570+0+0" と指定すると、ウィンドウは画面の左上隅に開かれる。

もし描画領域を正方形ではなく長方形にすると、描画に使われるフォントは縦と横で違う倍率で拡大縮小されることになる。このように拡大縮小できないフォントが指定されているときは、Hershey ベクトル・フォントの "HersheySerif" などのデフォルトのスケーラブル・フォントが使われる。

過去のバージョンとの互換性を確保するため、‘--bitmap-size’ オプションや BITMAPSIZE 環境変数の代わりに、 X のリソースの Xplot.geometry でもウィンドウのサイズが指定できるようになっている。

‘--emulate-color *option*’
(文字列、デフォルトは "no") *option* が "yes" のとき、カラーの出力画像をグレイスケールにして出力する。このオプションは、‘graph -T pcl’ で PCL 5 対応機器への出力を行っているときでなければ、使うことはあまりないかもしれない。(白黒の LaserJet プリンタなどの白黒 PCL 5 機器が搭載しているカラーからグレイスケールへの変換機能は、HP-GL/2 の 7 種類のペンの標準色を黒に置き換えるだけ (黄色も黒になる) で、あまりいいとは言えない)。環境変数 EMULATE_COLOR を "yes" に設定しても同じ指定ができる。

‘--frame-color *name*’
(文字列、デフォルトは "black") プロットの外枠と、白黒のデータセット (があればそ) のプロットの色に *name* を指定する。無効な名前が色名に指定されたときは、デフォルトの色が使われる。使える色名に付いては Appendix B [Color Names], page 158 を参照のこと。

‘--frame-line-width *frame_line_width*’
(実数、デフォルトは −1.0) プロットの外枠の線の太さを *frame_line_width* にする。数値は描画領域の縦か横の短い方に対する割合として指定する。負の値を指定すると、libplot ライブラリで設定しているデフォルト値が使われる。通常、デフォルト値は描画領域の 1/850 だが、‘-T X’、‘-T png’、‘-T pnm’、-T gif のときは 0 になる。0 になったときは、そのデバイスで描けるもっとも細い線が使われる。これはどのデバイスでもそうである。しかし idraw や xfig では、太さ 0 の線は表示されない。

graph -T tek および graph -T regis では、線の太さはデフォルト値以外は使えない。graph -T hpgl でも環境変数 HPGL_VERSION に 2 よりも小さな値をセットしているときは、同様である (2 がデフォルト値)。

‘--max-line-length *max_line_length*’
: (整数、デフォルトは 500) 一つのデータセットから描かれる一本の折れ線について、一度に描ける最大のデータ点数を *max_line_length* に指定する。データ点の数がこの数に達したときは、折れ線は複数回に分けて描画される。この時、特に通知やメッセージなどは行われない。これは、‘-q’ オプションで塗りつぶしが指定されているときは行われない。

  この指定は、いくつかの出力デバイス (昔の PostScript プリンタや HP-GL プロッタ) では出力バッファの大きさに制限があることによる。この最大点数は、環境変数 MAX_LINE_LENGTH でも指定できる。graph -T tek や生の graph 形式では、各データ点ごとにリアルタイムで描画が行われるので、このオプションは無効である。

‘--page-size *pagesize*’
: (文字列、デフォルトは "letter") プロットが描かれる用紙の大きさを指定する。このオプションは graph -T svg、graph -T ai、graph -T ps、graph -T cgm、graph -T fig、graph -T pcl、graph -T hpgl でだけ有効である。"letter" で指定される大きさは 8.5 in x 11 in である。"a0"...."a4" の ISO の用紙サイズおよび "a"...."e" の ANSI のサイズも指定できる ("letter" は "a"、"tabloid" は "b" と同じである)。"legal"、"ledger"、"b5" も指定できる。用紙サイズは、環境変数 PAGESIZE でも同様に指定できる。

  graph -T ai、graph -T ps、graph -T pcl、graph -T fig ではプロットが置かれる描画領域 (または ‘viewport’) は、デフォルトでは指定されたページの中央に置かれる正方形の領域である。graph -T hpgl でも同様だが、ページの中央ではない。この描画領域の縦、横の長さは、たとえば *pagesize* を "letter,xsize=4in" や "a4,xsize=10cm,ysize=15cm" のようにすることで指定できる。この長さは負の値になってもよい (負のときは鏡像になる)。

  描画領域の位置は、デフォルトの位置に対する相対座標で指定できる。たとえば *pagesize* を "letter,yoffset=1.2in" や"a4,xoffset=−5mm,yoffset=2.0cm" のようにするとよい。または用紙の左下隅からの相対座標として描画領域の左下隅の座標を指定することで、より正確に指定することもできる。それにはたとえば *pagesize* を"letter,xorigin=2in,yorigin=3in" や "a4,xorigin=0.5cm,yorigin=0.5cm" ようにするとよい。これらのオプションは取り混ぜて使ってもよい。

  graph -T svg と graph -T cgm では"xoffset"、"yoffset"、"xorigin"、"yorigin" は無視される。これらの形式では、画像が最終的にウェブブラウザ上のどこに表示されるかは、画像そのものから指定できないからである。しかし "xsize" と "ysize" オプションは無視される訳ではない。Appendix C [Page and Viewport Sizes], page 159 を参照のこと。

‘--pen-colors *colors*’
: (文字列、デフォルトは "1=red:2=green:3=blue:4=magenta:5=cyan") プロットに使われる色の番号を *colors* で指定する。各数字に割り当てられている色は、デフォルトの例で示される通りである。無効な色名が指定されたときは、その番号はデフォルトの色に設定される。使える色名については Appendix B [Color Names], page 158 を参照のこと。

‘--rotation *angle*’
(整数、デフォルトは 0) 描画領域中のプロットを角度 *angle* だけ回転する。その数字の角度だけ、反時計回りに回転する。環境変数 ROTATION に値を設定することでも、同じ指定ができる。

角度に 0 または 90 を指定することでそれぞれ、ポートレートまたはランドスケープ (用紙が縦、または横) モードになる。前衛芸術家にも便利に使ってもらえるオプションである。

‘--title-font-name *font_name*’
(文字列、デフォルトは "Helvetica"、ただし graph -T pcl では"Univers"、graph -T png、graph -T pnm、graph -T gif、graph -T hpgl、graph -T regis、graph -T tek の場合は "HersheySerif") プロットのタイトルに使われるフォントを *font_name* に設定する。通常はタイトルのフォントは、座標軸や目盛りのラベルと同じであり、‘-F’ オプションで設定されるが、‘--title-font-name’ が指定されたときはこれが優先される。フォント名では大文字と小文字は同一視される。使えるフォントの種類は、‘-T’ で何が指定されているかによって変わる。すべてのフォントのリストは Section A.1 [Text Fonts], page 136 を参照のこと。plotfont ユーティリティ・プログラムで、使えるフォントのキャラクタ・マップを表示することができる。Chapter 6 [plotfont], page 51 参照。

‘--title-font-size *size*’
(実数、デフォルトは 0.07) ‘-L’ オプションで指定されるプロットのタイトルの文字の大きさを *size* に指定する。文字の大きさは、プロット領域の縦または横の短い方に対する割合で指定する。

2.6.2 データセットのオプション

データセットは、ファイルから読み込まれて、それぞれがプロットの一部として描かれる。この節では各データセットに対する指定を行うオプションを解説する。この節のオプションは、指定対象とするデータセットのファイルよりも、コマンドライン上で前に指定しなければならない。入力ファイルが複数のときは、複数回指定されるオプションもある。

以下の 3 つのオプションは、ファイルからのデータの読み込まれ方を指定する。

‘-I *data-format*’
‘--input-format *data-format*’
このオプションに続くファイル中の、データの形式を指定する。

‘a’ テキスト形式。入力ファイル中には、x と y 座標の二つの実数値の組が表すデータ点が並んで格納されていると解釈される。データ点の x と y 座標は同じ行に書かれている必要はなく、また各点が違う行になっている必要もない。しかし空行 (連続する二つの改行文字がファイル中にあったとき) は、そこがデータセットの終わりで、次の行から別のデータセットであると解釈される。

‘e’ エラーバーのデータを含むテキスト形式。‘a’ 形式と同じだが、(x, y) という二つの数値の組ではなく、x、y、*error* の 3 つの数値の組で一つのデータ点が表されている、と解釈される。

‘g’ gnuplot が出力する、テキスト形式の「表形式 (‘table’ format)」。

‘f’ 単精度実数のバイナリ形式。入力ファイル中には、x と y 座標の二つの単精度浮動小数点実数で表されるデータ点が、並んで格納されていると解釈される。データセットの区切りは一つの FLT_MAX (単精度浮動小数点実数の取りうる最大値) の値を持つ単精度実数で表される。ほとんどのシステムではこの値は 3.4×10^{38} である。

‘d’ 倍精度実数のバイナリ形式。入力ファイル中には、x と y 座標の二つの倍精度浮動小数点実数で表されるデータ点が、並んで格納されていると解釈される。データセットの区切りは一つの DBL_MAX (倍精度浮動小数点実数の取りうる最大値) の値を持つ倍精度実数で表される。ほとんどのシステムではこの値は 1.8×10^{308} である。

‘i’ 整数のバイナリ形式。入力ファイル中には、x と y 座標の二つの整数で表されるデータ点が並んで格納されていると解釈される。データセットの区切りは一つの INT_MAX (整数の取りうる最大値) の値を持つ整数で表される。ほとんどのシステムはこの値は $2^{31} - 1$ である。

‘-a [*step_size* [*lower_limit*]]’
‘--auto-abscissa [*step_size* [*lower_limit*]]’
(実数、デフォルトは 1.0 および 0.0) 横軸 (x 軸) の値を自動生成する。データの入力形式 (‘a’、‘e’、‘f’、‘d’、‘i’) がなんであっても、このオプションを指定すると、入力ファイル中には横軸 (x 軸) の値は入っておらず、数値はすべて縦軸 (y 軸) の値であると解釈される。x の値の増分が *step_size* で、x の値の開始値が *lower_limit* で指定される。ファイル中に横軸の値も含まれているものとして読み込むように戻すときは、‘-a 0’ を指定すれば自動生成が無効になり、同時に *step_size* と *lower_limit* がデフォルト値に戻る。

‘-B’
‘--toggle-auto-bump’
線種の番号 (後述の *-m* オプションを参照のこと) はデフォルトではデータセットの開始点で一つずつ自動的に増えていくが、このオプションでそれをするかどうかを切り替えられる。

以下のオプションは、プロットの一部をなす各個別のデータセットを、どのように描くかを指定する。これらのオプションで 6 種類の「属性」 (記号の種類 symbol type、記号のフォント symbol font、線種 linemode、線の太さ line thickness、塗りつぶしの濃さ fill fraction、カラー/白黒 color/monochrome) を指定できる。

‘-m *line_mode*’
‘--line-mode *line_mode*’
(整数、デフォルトは 1) このオプションに続くデータットを描く線のモード (線種) を *line_mode* で指定する。線種 0 では線は全然描かれず、データ点の記号だけになる。データセットが白黒で描かれるときは、各 *line_mode* の意味する線種は以下のようになる。

1. 実線
2. 点線
3. 一点鎖線
4. 細かい破線

5. 長い破線

これより大きな数字 (5 以上の *line_mode* など) では上の 5 つが上の順で繰り返される。白黒の場合はしたがって線種は 0 を除いて 5 種類である。カラーの場合 ('-C' オプションで指定される) は、1 から 5 の各線種は以下のように解釈される。

1. 赤、実線
2. 緑, 実線
3. 青, 実線
4. 赤紫, 実線 (マゼンタ)
5. 青緑, 実線 (シアン)

線種 6 から 10 では色は上と同じだが、実線ではなく点線になる。11 から 15 では一点鎖線に、16 から 20 では細かい破線に、21 から 25 では長い破線になる。したがってカラーモードでは、0 を除いて 25 種類の線種があることになる。負の線種を指定すると線は描かれないが、データ点に置かれる記号は、その線種 (の符号を取った線種) の色で描かれる。

'-S [*symbol_number* [*symbol_size*]]'
'--symbol [*symbol_number* [*symbol_size*]]'

(整数と実数、デフォルトは 0 と 0.03) データ点に記号を描く。*symbol_number* で記号の種類を、*symbol_size* で記号の大きさを指定する。記号の大きさはプロット領域の縦あるいは横の短い方の長さに対する割合で指定する。カラーモードでは、記号はそのデータセットでデータ点を結ぶラインと同じ色で描かれる。

'-m' オプションに負の数で線種を指定することで、記号の色を指定し、かつデータ点を線で結ばないように指定できる (上述)。

以下の表に、最初のいくつかの記号を示す (記号 0 はまったく記号を描かないことを表す)。

1. 点 ($\cdot$)
2. プラス記号 ($+$)
3. アスタリスク ($*$)
4. 円 ($\circ$)
5. バツ印 ($\times$)

0 番から 31 番までの記号は libplot ライブラリに組み込まれている。Section A.5 [Marker Symbols], page 155 参照。32 番以降の記号は、symbol フォントの文字を指定していると解釈される。これは '--symbol-font-name' オプションで指定できる (後述)。

'-W *line_width*'
'--line-width *line_width*'

(実数、デフォルトは -1.0) データセット中で連続するデータ点を結ぶ線の太さを *line_width* にする。その数値は、描画領域の縦又は横の短い方に対する割合で指定する。負の値を指定すると、libplot ライブラリのデフォルト値が使われる。デフォルト値は描画領域の一辺の長さの 1/850 だが、'-T X'、'-T png'、'-T pnm'、-T gif では 0 になる。太さに 0 を指定するとどの出力デバイスでも、そのデバイスでもっとも細いラインが描かれるが、idraw や xfig は太さ 0 の線は表示しないようになっている。

graph -T tek および graph -T regis ではデフォルトの太さ以外の線は描けない。graph -T hpgl でも、環境変数 HPGL_VERSION に 2 よりも小さな値を指定したときは (デフォルト値は 2)、デフォルトの太さ以外の線は描けない。

‘-q *fill_fraction*’
‘--fill-fraction *fill_fraction*’
(実数、デフォルトは −1.0) 一つのデータセット中の連続するデータ点が線で結ばれる設定になっているとき、線が作る多角形を塗りつぶす色の濃さを *fill_fraction* で指定する。*fill_fraction*=1.0 にするともっとも濃くなる (線の色と同じ濃さになる)。*fill_fraction*=0.0 にすると、多角形の内側は白くなる。*fill_fraction* の値を負に設定すると、多角形の内側には透明が設定される。

もし多角形どうしが重なる部分がある場合、「偶奇塗りつぶし則 (even-odd fill rule)」でどこがその多角形の内側か、あるいは外側かが決められる。*PostScript Language Reference Manual* に具体的な説明がある。

‘-q’ オプションは graph -T tek では無効である。また graph -T hpgl では環境変数 HPGL_VERSION に 2 よりも小さな値を指定したときは (デフォルト値は 2)、機能が制限される。

‘-C’
‘--toggle-use-color’
そのデータセットの描画がカラーか白黒かを切り替える。線種に指定されている数値の示す意味が、描画がカラーか白黒かで変わる。上述の ‘-m’ オプションを参照のこと。

‘--symbol-font-name *symbol_font_name*’
(文字列、デフォルトは "ZapfDingbats"、ただし ‘-T png’、‘-T pnm’、‘-T gif’、‘-T pcl’、‘-T hpgl’、‘-T regis’、-T tek では "HersheySerif") 32 番以降の数字で指定された記号をどのフォントから取るかを *symbol_font_name* に指定する。フォント名では大文字と小文字は区別されない。もし無効なフォントが指定された場合は、デフォルトのフォントが使われる。使えるフォントは ‘-T’ オプションで何が指定されているかによって変わる。たとえば ‘-T pcl’ や ‘-T hpgl’ では、通常は Wingdings フォントが有効になり、これに含まれる記号を使えるようになる。すべてのフォントのリストは Section A.1 [Text Fonts], page 136 を参照のこと。plotfont ユーティリティ・プログラムで、使えるフォントのキャラクタ・マップを表示することができる。Chapter 6 [plotfont], page 51 参照。

2.6.3 多重プロットのオプション

以下のオプションは、多重プロット (複数のプロットを一つのウィンドウや用紙にまとめる) に関するものである。コマンドライン上のオプションは、それの後に指定されるデータ・ファイルをプロットするときに適用されるが、‘--reposition’ をその区切りとして使うことができる。

‘--reposition *x y size*’
(実数、デフォルトは 0.0、0.0、1.0) 次のプロットを描く「仮想描画領域」を、一辺の長さが *size* の正方形にし、左下隅が (x, y) になるように配置する。座標値は、物理的な用紙上の全体プロットの描画範囲上で (0, 0) がその左下隅、(1, 1) がその右上隅になるように正規化した値で与える。仮想描画領域の中に描かれるプロット領域の大きさは ‘-h’ と ‘-w’ オプションで、プロット領域の場所は ‘-u’ と ‘-w’ オプションで指定できる。‘--reposition’ あとでは、これらの 4 つのオ

プションの数値は物理的な全体プロットの描画領域に対してではなく、仮想描画領域中のプロット範囲に対する数値であると解釈される。

‘--blankout *blankout_fraction*’
(実数、デフォルトは 1.3) 多重プロットに追加するプロットを描くとき、そのプロットを置く場所が空いていなければならない。*blankout_fraction*=1.0 を指定すると、新たにプロットを置こうとする場所にある画像が消去される。*blankout_fraction*=1.3 を指定すると、そのプロットよりも 30% だけ大きな領域で画像が消去される。プロットを横に並べようとするときは、1.0 がちょうどよい。

graph -T tek では画像の消去はできない。graph -T hpgl では環境変数 HPGL_VERSION と HPGL_OPAQUE_MODE がデフォルト値 (それぞれ "2" と "yes") でないときには、画像の消去はできない。

2.6.4 graph の生出力データのオプション

以下のオプションは graph 生出力の形式に関するものであり、‘-T’ による出力デバイスの形式の指定を何もしなかった時に有効である。この場合 graph は graphics metafile 形式で出力を行うが、これを plot プログラムで他の形式に変換することもできる。以下のオプションは、コマンドラインでどのファイル名よりも先に指定され、生成されるプロット (あるいは多重プロット) 全体に作用する。

‘-O’
‘--portable-output’
GNU metafile 出力を、バイナリ形式 (デフォルト) ではなく、ポータブル形式 (テキスト形式、人間の読める形式) で行う。これは環境変数 META_PORTABLE を "yes" に設定することでも指定できる。

2.6.5 情報を表示するオプション

以下のオプションで、必要な情報を表示させることができる。

‘--help’ コマンドライン・オプションのリストを表示させ、そのまま実行を終了する。

‘--help-fonts’
利用可能なフォントのリストを表示して、そのまま終了する。出力される内容は、‘-T’ オプションで指定される出力デバイスによって変わる。graph -T X、graph -T svg、graph -T ai、graph -T ps、graph -T cgm、graph -T fig では PostScript 標準の 35 種類のフォントが使える。graph -T svg、graph -T ai、graph -T pcl、graph -T hpgl では 45 種類の PCL 5 標準フォントが使える。graph -T pcl と graph -T hpgl ではヒューレット・パッカードのベクトル・フォントが使える。上に挙げたすべての出力形式、および graph -T png、graph -T pnm、graph -T gif、graph -T regis、graph -T tek では 22 種類の Hershey ベクトル・フォントが使える。graph の生出力では原理的にこれらのフォントがすべて使える。どの形式にも、graph 生出力から plot で変換できるからである。plotfont ユーティリティ・プログラムで、使えるフォントのキャラクタ・マップを表示することができる。Chapter 6 [plotfont], page 51 参照。

‘--list-fonts’
‘--help-fonts’ と同じだが、他のプログラムにパイプで渡して処理しやすいように、フォントのリストを 1 カラムで出力する。‘-T’ オプションでの出力デバイスの指定が何もないときは、サポートされているすべてのフォントのリストが表示される。

‘--version’
: graph および GNU plotting utilities パッケージのバージョン番号を表示して、そのまま終了する。

2.7 graph の環境変数

graph の動作には、環境変数を読んでその値により制御されるものがある。BITMAPSIZE、PAGESIZE、BG_COLOR、EMULATE_COLOR、MAX_LINE_LENGTH、ROTATION については上述したとおりである。これらの環境変数はそれぞれ、コマンドライン・オプションの‘--bitmap-size’、‘--page-size’、‘--bg-color’、‘--emulate-color’、‘--max-line-length’、‘--rotation’と同じ働きを持つ。他にも、特定の出力デバイスに対してだけ有効な環境変数がある。

graph -T X では X Window の画面にウィンドウが一つ開かれ、そこにプロットが描画されるが、このとき DISPLAY 環境変数が設定されていれば、それで示されるディスプレイにウィンドウが開かれる。

PNG または疑似 GIF 形式を生成する graph -T png または graph -T gif に対しては、二つの環境変数が用意されている。INTERLACE が "yes" のときは、出力される画像はインターレース形式になる。TRANSPARENT_COLOR に色名を指定しておくと、出力される画像のその色の部分が透明に設定され、PNG や GIF を扱う多くのアプリケーションで透明として扱われる。使える色名については Appendix B [Color Names], page 158 を参照のこと。

graph -T pnm では Portable Anymap (PBM/PGM/PPM) 形式で出力されるが、環境変数 PNM_PORTABLE が "yes" のとき、出力は PBM、PGM、PPM のポータブル形式 (人間が読めるテキスト形式) で行われる。そうでない場合はバイナリ形式 (これがデフォルト) で行われる。

ウェブ用のベクトル画像の形式である WebCGM プロファイルにしたがった CGM 形式のファイルを graph -T cgm で生成する場合に有効な環境変数が二つある。デフォルトでは、生成される CGM ファイルはバージョン 3 の規格に沿っているが、マイクロソフト・オフィスなどの古い CGM 表示ソフトウェアはほとんど、CGM のバージョン 1 しか扱えない。そこで環境変数 CGM_MAX_VERSION を"1"、"2"、"3"、"4" のどれかに設定することで、出力される CGM のバージョンの最大値を指定できる ("4" がデフォルト)。また CGM_ENCODING で CGM ファイルのエンコーディングを指定できる。これには"clear_text" (人が読めるテキスト形式) と "binary" が指定できる ("binary" がデフォルト)。WebCGM プロファイルではバイナリ形式が要求される。

graph -T pcl ではヒューレット・パッカードのプリンタまたはプロッタの出力形式である PCL 5 形式で出力されるが、このとき PCL_ASSIGN_COLORS 環境変数が有効である。PCL 5 出力をカラープリンタなどに出力する際に指定する。これにより色の再現性を正確にするために、内部で出力デバイスに、各色を描画するための「論理ペン」が任意に設定できるようになる。"no" にセットされるとあらかじめ設定されている色のペンだけが用いられ、他の色は濃淡の変化 (シェイディング) によって擬似的に描画される。世の中の PCL 5 デバイスは、カラーより白黒が多いため、デフォルトでは "no" であり、色はシェイディング (グレイスケール) に置き換えられる。

graph -T hpgl では HP-GL (ヒューレット・パッカード・グラフィクス・ランゲージ) 形式の出力が生成されるが、これにもいくつかの環境変数が用意されている。もっとも重要なのは HPGL_VERSION で、これには "1"、"1.5"、"2" のいずれかが指定できる ("2" がデフォルト)。"1" は純正の HP-GL、"1.5" は HP7550A グラフィクス・プロッタおよび HP758x、HP7595A、HP7596A ドラフティング・プロッタに出力できる形式 (HP-GL にいくつかの HP-GL/2 拡張を加えたもの)、"2" は新しい HP-GL/2 でそれぞれ描画するよう指示する。

もしバージョンが "1" または "1.5" にセットされていると、使えるフォントはベクトル・フォントのみで、線はすべてデフォルトの太さで描かれる (‘-W’ は無効になる)。さらにバージョンが "1" のときは、任意の多角形を線の色で塗りつぶす機能が使えなくなる (‘-q’ オプションは座標軸に沿って置かれた円や長方形を塗りつぶすときに使える)。

graph -T hpgl のときの用紙上の描画領域は HPGL_ROTATE 環境変数を"yes" に設定すると反時計回りに 90 度回転させることができる。--rotation オプションでは描画領域を回転させ、その左下隅の位置を用紙の左下隅からの相対位置で指定するため、これとは異なった効果になる。HPGL_ROTATE 環境変数に設定できる値は、"no" と "yes" に加えて、"0"、"90"、"180"、および "270" である。"no" と "yes" は "0" と "90" とそれぞれ同じである。"180" と "270" は HPGL_VERSION が "2" のとき (デフォルトがそうである) にだけ有効である。

不透明化機能による、白い線や白い色による塗りつぶしは、HPGL_VERSION が "2" (デフォルト) で環境変数 HPGL_OPAQUE_MODE が "yes" のときに可能になる。もしこれが "no" のときは不透明化による白い色での塗りつぶしは行われず、白色の線 (通常は 0 番のペンで描かれる) も描かれない。この機能は昔の HP-GL/2 デバイスに適している。たとえば HP-GL/2 ペン・プロッタは不透明化や 0 番のペンによる白色の線には対応していない。また古い HP-GL/2 デバイスの中には不透明化機能が不十分なものもある。

デフォルトでは、graph -T hpgl では使えるペンはあらかじめ決められているものだけである。描画時点でどのペンを使うかは、HPGL_PENS 環境変数で指定される。HPGL_VERSION が "1" のときは HPGL_PENS のデフォルトのペンは"1=black" である。HPGL_VERSION が "1.5" または "2"のときは、HPGL_PENS のデフォルト値は "1=black:2=red:3=green:4=yellow:5=blue:6=magenta:7=cyan" である。指定の仕方は見ての通りである。HPGL_PENS を指定することで 1...31 番のどのペンにも色を割り当てることができる。指定できる色名については Appendix B [Color Names], page 158 を参照のこと。1 番のペンはいつでも使える。その色は黒である必要はない。2...31 番の他のペンはあってもなくてもよい。

HPGL_VERSION が "2" のときは graph -T hpgl では HPGL_ASSIGN_COLORS も有効である。もしこれが "yes" にセットされていれば、graph -T hpgl で使えるペンの色は HPGL_PENS による制約を受けない。1...31 番の「論理ペン」に必要に応じて色が割り当てられる。LaserJet プリンタや DesignJet 以外の、他の HP-GL/2 機器でこの機能が使えるものはあまり多くはないため、デフォルトでは "no" である。特に HP-GL/2 ペン・プロッタは非対応である。

graph -T tek では Tektronix 端末のための出力が行われるが、環境変数 TERM が有効である。もし TERM が "xterm"、"nxterms"、"kterm" のいずれかであればそれぞれ、graph がその時実行されている環境は X Window システム上の VT100 端末エミュレータの xterm、nxterm、kterm であると判断される。このとき、graph -T tek では描画が行われる前に、xterm に付属していて通常は隠れている Tektronix のウィンドウを、前面にポップアップさせるエスケープ・シーケンスが送られる。また描画が行われた後には、元の VT100 端末に制御を戻すためのエスケープ・シーケンスが送られる。その際、Tektronix のウィンドウは画面に残ったままになる。

TERM が "kermit"、"ansi.sys"、"nansi.sys" のいずれかの時は、MS-DOS 版の kermit 上の VT100 エミュレータ上であるとみなされる。このとき、graph -T tek では描画が行われる前に、VT100 エミュレータから Tektronix のウィンドウに切り替えるエスケープ・シーケンスが送られる。また出力には Tektronix 制御コードは、kermit 特有のものが使われる。このとき、使える色数には制限がある (ansi.sys の 16 色はサポートされる)。また描画が行われた後には、元の VT100 端末に制御を戻すためのエスケープ・シーケンスが送られる。キーボードで ‘ALT’ - ‘-’ をタイプすると、VT100 モードと Tektronix モードの切り替えが手動でできる。

3 plot コマンド

3.1 plot コマンドの使い方

GNU プロット・フィルタ plot は、GNU graphics metafile 形式の画像ファイルを画面に表示したり、他の形式に変換したりするコマンドである。入力はコマンドラインでファイル名を指定するか、標準入力から与える。出力形式は '-T' で指定する。指定できる形式は、"X"、"png"、"pnm"、"gif"、"svg"、"ai"、"ps"、"cgm"、"fig"、"pcl"、"hpgl"、"regis"、"tek"、"meta" で、"meta" がデフォルトである。

metafile 形式は、特定の出力デバイスに依存しないベクトル形式の画像フォーマットである。デフォルトではバイナリ形式だが、人間に読めるテキスト形式で出力させることもできる (Appendix D [Metafiles], page 161 参照)。graph、pic2plot、tek2plot、plotfont の各コマンドは '-T' オプションによる出力形式の指定がないときは metafile 形式で出力する。libplot ライブラリでも metafile 形式の出力が生成できる。一般に metafile 形式では複数ページの画像が記述できるが、graph コマンドで生成できるのは単ページの画像だけである。

plot コマンドと metafile という画像形式はどちらも、画像ファイルをベクトル形式で保存しておいたり、様々なソフトウェアで表示、編集したりするのに便利に使える。以下にその例を示す。

テキストファイル中で x と y が交互に並んでいるデータをプロットするには、graph で以下のようにすればよい。

```
graph < datafile > test.meta
```

生成されるファイル 'test.meta' は単ページの metafile 形式の画像である。同様に、以下のようにしても一つの画像だけからなるプロットを metafile 形式で作ることができる。

```
echo 0 0 1 1 2 0 | spline | graph > test.meta
```

生成したプロットを X Window の画面上に表示するには、以下のようにする。

```
plot -T X test.meta
```

または

```
plot -T X < test.meta
```

としてもよい。また以下のようにすれば、生成したプロットを PostScript プリンタで印刷できるだろう。

```
plot -T ps < test.meta | lpr
```

フリーのドロー系ソフトウェア idraw で編集するには、以下のようにする。

```
plot -T ps < test.meta > test.ps
idraw test.ps
```

PNG 形式にするには、以下のようにする。

```
plot -T png < test.meta > test.png
```

"portable anymap" 形式 (PBM、PGM、PPM 形式のから選ばれる適切な形式) を生成するには、以下のようにする。

```
plot -T pnm < test.meta > test.pnm
```

疑似 GIF 形式を生成するには、以下のようにする。

```
plot -T gif < test.meta > test.gif
```

ウェブブラウザで表示できる SVG または WebCGM 形式の画像を生成するには、以下のようにする。表示するには、ウェブブラウザがこれらの形式をサポートしている必要がある。

```
plot -T svg < test.meta > test.svg
plot -T cgm < test.meta > test.cgm
```

アドビの Illustrator で表示、編集できる形式で出力するには、以下のようにする。

```
plot -T ai < test.meta > test.ai
```

フリーのドロー系ソフトウェア xfig で編集できる画像形式にし、xfig で開くには、以下のようにする。

```
plot -T fig < test.meta > test.fig
xfig test.fig
```

plot -T pcl、plot -T hpgl、plot -T regis、plot -T tek で、その他の形式も生成できる。

plot コマンドの挙動は、実行する環境にあわせて変えることができる。PAGESIZE 環境変数は、plot -T svg、plot -T ai、plot -T ps、plot -T cgm、plot -T fig、plot -T pcl、plot -T hpgl に対して有効である。同様に環境変数 BITMAPSIZE は plot -T X、plot -T png、plot -T pnm、plot -T gif のときに有効である。また DISPLAY 環境変数は plot -T X に、TERM 環境変数は plot -T tek のとき有効である。この他に、plot -T pcl と plot -T hpgl のときにだけ有効な環境変数がいくつかある。詳細については Section 3.3 [plot Environment], page 33 を参照のこと。

3.2 plot のコマンドライン・オプション

GNU プロット・フィルタ plot は、GNU graphics metafile 形式のファイルを表示したり、他の形式に変換したりするコマンドである。出力形式は '-T' で指定する。GNU metafile 形式は GNU graph、pic2plot、tek2plot、plotfont の各コマンドや、GNU libplot ライブラリを使った他のアプリケーションで生成できる。GNU metafile 形式の技術的な詳細に付いては Appendix D [Metafiles], page 161 を参照のこと。

入力ファイルの名前をコマンドライン上で指定できる。その順番やコマンドライン・オプションの順番は、意味を持たない。ファイル名が何も指定されないか、'-' というファイル名が指定されたときは、標準入力から読み込まれる。出力は、'-T X' が指定されない限りは標準出力に行われる。'-T X' が指定されたときは X Windows の画面上のウィンドウに描画され、標準出力にはなにも出力されない。

以下に、指定できるコマンドライン・オプションのリストを載せる。コマンドライン・オプションは 4 種類に分けられる。

1. 描画パラメータを指定するオプション。
2. plot 生出力のデータ ('-T' で何も指定されないときの出力形式) に関するオプション。
3. 入力される metafile 形式を指定する (後方互換性を維持するためだけに用意されている) オプション。
4. 情報を表示するためのオプション ('--help' など)。

引数を取るオプションでは、説明の最初にカッコ書きでその引数の型とデフォルト値を示している。

以下のオプションは、描画パラメータを指定する。

'-T *type*'
'--output-format *type*'
: (文字列、デフォルトは "meta") ディスプレイの種類、あるいは出力形式を *type* で指定する。指摘できる文字列は"X"、"png"、"pnm"、"gif"、"svg"、"ai"、"ps"、

"cgm"、"fig"、"pcl"、"hpgl"、"regis"、"tek"、"meta" のいずれかである。それぞれ、X Window システム、PNG 形式、portable anymap (PBM/PGM/PPM) 形式、疑似 GIF 形式、XML ベースの Scalable Vector Graphics 形式、Adobe Illustrator 形式、idraw で編集できる PostScript 形式、ウェブ用のベクトル画像形式である WebCGM 形式、xfig で編集できる形式、ヒューレット・パッカードの PCL 5 プリンタ言語、ヒューレット・パッカード・グラフィクス言語 (デフォルトでは HP-GL/2)、DEC で開発された ReGIS (remote graphics instruction set) 形式、Tektronix 形式、特定のデバイスに依存しない GNU graphics metafile 形式である。'--display-type' は古いので '--output-format' を使うべきである。

'-p *n*'
'--page-number *n*'
(正の整数) 変換される metafile、または複数の metafile データの中から、*n* ページ目だけを表示する。

metafile 形式では複数のページを保持でき、先頭ページのページ番号は 1 である。また各ページには一つまたは複数の「フレーム」を置くことができる。plot -T X、plot -T regis、plot -T tek ではプロットはリアルタイムで行われ、各フレームを描くごとに前に描かれたフレームは消去される。plot -T png、plot -T pnm、plot -T gif、plot -T svg、plot -T ai、plot -T ps、plot -T cgm、plot -T fig、plot -T pcl、plot -T hpgl ではプロットはリアルタイムでは行われず、複数のフレームが置かれたページでは、そのページの最後のフレームだけが描画される。

'-p' オプションが指定されない場合、デフォルトではすべてのページが描画される。たとえば plot -T X では各ページをそれぞれ別のウィンドウに描画する。もし '-T png'、'-T pnm'、'-T gif'、'-T svg'、'-T ai'、'-T fig' オプションのいずれかが指定されているときは、最初のページだけを出力するのがデフォルトである。PNG 形式、PNM 形式、疑似 GIF 形式、SVG 形式、AI 形式、Fig 形式は、単ページしかサポートしていないからである。

GNU plotting utilities が生成する metafile 形式 (たとえば graph の生出力) はほとんどの場合、二つのフレームを持つ単ページのプロットである。1 つ目のフレームは描画領域内の画像を消去するためのもので、二つ目のフレームに実際の描画内容が含まれている。

'-s'
'--merge-pages'
各ページのフレームを一つにまとめ、またすべてのページを一つにまとめる。

これは、別々に作られた単ページのプロットを一つにまとめるときに使う。たとえば、plot を複数回実行して作られた複数のファイルなどである。この機能を使って多重プロットをすることもできる (Section 2.4 [Multiplotting], page 11 参照)。

'--bitmap-size *bitmap_size*'
(文字列、デフォルトは "570x570") 描画領域の大きさをピクセル単位で *bitmap_size* に設定する。このオプションは plot -T X、plot -T png、plot -T pnm、plot -T gif のときにだけ有効である。これらの出力形式では描画領域のサイズがピクセル単位で表現されているからである。環境変数 BITMAPSIZE でも同様に指定できる。

plot -T X のときは描画のために開かれたウィンドウが描画領域になる。X Window の画面上でのこのウィンドウの位置も、コマンドライン・オプションで指定できる。たとえば *bitmap_size* を "570x570+0+0" と指定すると、開かれたウィンドウは画面の左上隅に置かれる。

もし描画領域を正方形ではなく長方形にすると、描画に使われるフォントは縦と横で違う倍率で拡大縮小されることになる。そのような拡大縮小ができないフォントが指定されているときは、Hershey ベクトル・フォントの "HersheySerif" などのデフォルトのスケーラブル・フォントが使われる。

後方互換性のために、X resource の Xplot.geometry を設定することで、'--bitmap-size' や BITMAPSIZE の代わりにディスプレイのサイズを設定できる。

'--emulate-color *option*'

(文字列、デフォルトは "no") *option* が "yes" のとき、カラーの出力画像をグレイスケールにして出力する。このオプションは、'plot -T pcl' で PCL 5 対応機器への出力を行っているときでなければ、使うことはあまりないかもしれない。(白黒の LaserJet プリンタなどの白黒 PCL 5 機器が搭載しているカラーからグレイスケールへの変換機能は、HP-GL/2 の 7 種類のペンの標準色を黒に置き換えるだけ (黄色も黒になる) で、あまりいいとは言えない)。環境変数 EMULATE_COLOR を "yes" に設定しても同じ指定ができる。

'--max-line-length *max_line_length*'

(整数、デフォルトは 500) 一つのデータセットから描かれる一本の折れ線について、一度に描ける最大のデータ点数を *max_line_length* に指定する。データ点の数がこの数に達したときは、折れ線は複数回に分けて描画される。この時、特に通知やメッセージなどは行われない。'-q' オプションで塗りつぶしが指定されているときは、これは行われない。

この指定は、いくつかの出力デバイス (昔の PostScript プリンタや HP-GL プロッタ) では出力バッファの大きさに制限があることによる。この最大点数は、環境変数 MAX_LINE_LENGTH でも指定できる。plot -T tek や plot の生出力では、各データ点ごとにリアルタイムで描画が行われるので、このオプションは無効である。

'--page-size *pagesize*'

(文字列、デフォルトは "letter") プロットが描かれる用紙の大きさを指定する。このオプションは plot -T svg、plot -T ai、plot -T ps、plot -T cgm、plot -T fig、plot -T pcl、plot -T hpgl でだけ有効である。"letter" で指定される大きさは 8.5 in x 11 in である。"a0"...."a4" の ISO の用紙サイズおよび "a"...."e" の ANSI のサイズも指定できる ("letter" は "a"、"tabloid" は "b" と同じである)。"legal"、"ledger"、"b5" も指定できる。用紙サイズは、環境変数 PAGESIZE でも同様に指定できる。

plot -T ai、plot -T ps、plot -T pcl、plot -T fig ではプロットが置かれる描画領域 (または 'viewport') は、デフォルトでは指定されたページの中央に置かれる正方形の領域である。plot -T hpgl でも同様だが、ページの中央ではない。この描画領域の縦、横の長さは、たとえば *pagesize* を "letter,xsize=4in" や "a4,xsize=10cm,ysize=15cm" のようにすることで指定できる。この長さは負の値になってもよい (負のときは鏡像になる)。

描画領域の位置は、デフォルトの位置に対する相対座標で指定できる。たとえば *pagesize* を"letter,yoffset=1.2in" や "a4,xoffset=−5mm,yoffset=2.0cm" のようにするとよい。または用紙の左下隅からの相対座標として描画領域の左下隅の座標を指定することで、より正確に指定することもできる。それにはたとえば *pagesize* を"letter,xorigin=2in,yorigin=3in" や "a4,xorigin=0.5cm,yorigin=0.5cm" ようにするとよい。これらのオプションは取り混ぜて使ってもよい。

`plot -T svg` と `plot -T cgm` では"xoffset"、"yoffset"、"xorigin"、"yorigin" は無視される。これらの形式では、画像が最終的にウェブブラウザ上のどこに表示されるかは、画像そのものから指定できないからである。しかし "xsize" と "ysize" オプションは無視される訳ではない。Appendix C [Page and Viewport Sizes], page 159 を参照のこと。

他にもいくつかの描画設定を、以下のオプションで指定することができる。これらの中には、入力として与えられた metafile 形式のデータの中の指定の方が優先されるものもある。GNU 版以前の 'plot(5)' の metafile にはそういった指定法がなかったので、以下のオプションは、そういう古いデータをプロットするときに使うとよい。

'`--bg-color` *name*'
: (文字列、デフォルトは "white") 背景色を *name* に指定する。これは `plot -T X`、`plot -T png`、`plot -T pnm`、`plot -T gif`、`plot -T cgm`、`plot -T regis`、`plot -T meta` のときにだけ有効である。無効な名前が色名に指定されたときは、デフォルトの色が使われる。使える色名に付いては Appendix B [Color Names], page 158 を参照のこと。環境変数 `BG_COLOR` でも同様に背景色を指定できる。

  '`-T png`' か '`-T gif`' オプションのどちらかが使われているときは、環境変数 `TRANSPARENT_COLOR` に背景色を指定することで、透過 PNG または透過疑似 GIF 形式の画像を生成できる。Section 3.3 [plot Environment], page 33 参照。'`-T svg`' か '`-T cgm`' オプションのどちらかが使われているときは、背景色に "none" を指定することで、背景色のない画像を生成できる。

'`-f` *font_size*'
'`--font-size` *font_size*'
: (実数、デフォルトは出力形式によって異なる) 文字列のフォントサイズの初期値を *font_size* で指定する。数値は、描画領域の横の長さに対する割合として指定する。

'`-F` *font_name*'
'`--font-name` *font_name*'
: (文字列、デフォルトは "Helvetica"、ただし `plot -T pcl` では"Univers"、`plot -T png`、`plot -T pnm`、`plot -T gif`、`plot -T hpgl`、`plot -T regis`、`plot -T tek`、`plot` の生出力の場合は "HersheySerif") 文字列 ('ラベル') に使うフォントを *font_name* に指定する。フォント名では大文字と小文字は区別されない。もし指定されたフォントが使えない場合、デフォルトのフォントが使われる。使えるフォントは、'`-T`' オプションで何を指定するかによって異なる。すべてのフォントのリストは Section A.1 [Text Fonts], page 136 を参照のこと。`plotfont` ユーティリティ・プログラムで、使えるフォントのキャラクタ・マップを表示することができる。Chapter 6 [plotfont], page 51 参照。

‘-W *line_width*’
‘--line-width *line_width*’
(実数、デフォルトは −1.0) プロットの外枠の線の太さを *line_width* にする。数値は描画領域の縦か横の短い方に対する割合として指定する。負の値を指定すると、libplot ライブラリで設定しているデフォルト値が使われる。通常、デフォルト値は描画領域の 1/850 だが、‘-T X’、‘-T png’、‘-T pnm’、-T gif のときは 0 になる。0 になったときは、そのデバイスで描けるもっとも細い線が使われる。これはどのデバイスでもそうである。しかし idraw や xfig では、太さ 0 の線は描かれない。

plot -T tek および plot -T regis では、線の太さはデフォルト値以外は使えない。plot -T hpgl でも環境変数 HPGL_VERSION に 2 よりも小さな値をセットしているときは、同様である (2 がデフォルト値)。

‘--pen-color *name*’
(String, デフォルトは "black") プロットに使われる「ペンの色」を *name* で指定する。無効な名前が色名に指定されたときは、デフォルトの色が使われる。使える色名に付いては Appendix B [Color Names], page 158 を参照のこと。

以下のオプションは、plot の生出力に対してだけ、つまり ‘-T’ をつけずに plot を使うときにだけ有効である。その場合の plot の出力は GNU metafile 形式であり、これをまた plot に渡すことで他の形式に変換できる。

‘-O’
‘--portable-output’
GNU metafile 出力を、バイナリ形式 (デフォルト) ではなく、ポータブル形式 (テキスト形式、人間の読める形式) で行う。これは環境変数 META_PORTABLE を "yes" に設定することでも指定できる。

plot コマンドは入力された GNU matafile 形式のデータがバイナリ (デフォルトの形式) がテキスト形式かを自動で判別する。バイナリ形式は、計算機のアーキテクチャに依存した形式である。Appendix D [Metafiles], page 161。

昔の plot との互換性を持たせるために、GNU 版より前の ‘plot(5)’ 形式の入力もサポートしている。普通はこれはバイナリ形式で、整数を 2 バイトで表現しているが、そのバイトオーダーは計算機のアーキテクチャに依存している。そういったファイルを現在の plotutils のコマンドで扱いたいとき、バイトオーダーが分かっていればそれをコマンドライン・オプション ‘-h’ (“high byte first”) または ‘-l’ (“low byte first”) で明示的に指定できる。また GNU でないシステムでは plot(5) 形式でテキスト形式をサポートしているものがある。そういったデータに対しては ‘-A’ でその旨を明示できる。そういった独自拡張のあるものを除けば、plot(5) 形式では一つのファイルで扱えるページ数は 1 である。

‘-h’
‘--high-byte-first-input’
入力が古い ‘plot(5)’ metafile 形式で、整数の上位バイトが先に来る形式 (ビッグエンディアン) であることを指定する。そういうデータは多くない。

‘-l’
‘--low-byte-first-input’
入力が古い ‘plot(5)’ metafile 形式で、整数の下位バイトが先に来る形式 (リトルエンディアン) であることを指定する。こちらの方が多数派である。

‘-A’
‘--ascii-input’
入力が古い ‘plot(5)’ metafile 形式で、テキスト形式であることを指定する。そういうデータは、かなり少ない。plottoa という名前のプログラムがこの形式を出力していた。

以下のオプションで、必要な情報を表示させることができる。

‘--help’ コマンドライン・オプションのリストを表示させ、そのまま実行を終了する。

‘--help-fonts’
利用可能なフォントのリストを表示して、そのまま終了する。出力される内容は、‘-T’ オプションで指定される出力デバイスによって変わる。plot -T X、plot -T svg、plot -T ai、plot -T ps、plot -T cgm、plot -T fig では PostScript 標準の 35 種類のフォントが使える。plot -T svg、plot -T ai、plot -T pcl、plot -T hpgl では 45 種類の PCL 5 標準フォントが使える。plot -T pcl と plot -T hpgl ではヒューレット・パッカードのベクトル・フォントが使える。上に挙げたすべての出力形式、および plot -T png、plot -T pnm、plot -T gif、plot -T regis、plot -T tek では 22 種類の Hershey ベクトル・フォントが使える。plot の生出力では原理的にこれらのフォントがすべて使える。どの形式にも、plot 生出力から plot で変換できるからである。plotfont ユーティリティ・プログラムで、使えるフォントのキャラクタ・マップを表示することができる。Chapter 6 [plotfont], page 51 参照。

‘--list-fonts’
‘--help-fonts’ と同じだが、他のプログラムにパイプで渡して処理しやすいように、フォントのリストを 1 カラムで出力する。‘-T’ オプションでの出力デバイスの指定が何もないときは、サポートされているすべてのフォントのリストが表示される。

‘--version’
plot および GNU plotting utilities パッケージのバージョン番号を表示して、そのまま終了する。

3.3 plot の環境変数

plot の動作には、環境変数を読んでその値により制御されるものがある。BITMAPSIZE、PAGESIZE、BG_COLOR、EMULATE_COLOR、MAX_LINE_LENGTH、ROTATION については上述したとおりである。これらの環境変数はそれぞれ、コマンドライン・オプション の ‘--bitmap-size’、‘--page-size’、‘--bg-color’、‘--emulate-color’、‘--max-line-length’、‘--rotation’ と同じ働きを持つ。他にも、特定の出力デバイスに対してだけ有効な環境変数もある。

plot -T X では X Window の画面にウィンドウが一つ開かれ、そこにプロットが描画されるが、このとき DISPLAY 環境変数が設定されていれば、それで示されるディスプレイにウィンドウが開かれる。

PNG または疑似 GIF 形式を生成する plot -T png または plot -T gif に対しては、二つの環境変数が用意されている。INTERLACE が "yes" のときは、出力される画像はインターレース形式になる。TRANSPARENT_COLOR に色名を指定しておくと、出力される画像のその色の部分が透明に設定され、PNG や GIF を扱う多くのアプリケーションで透明として扱われる。使える色名については Appendix B [Color Names], page 158 を参照のこと。

plot -T pnm では Portable Anymap (PBM/PGM/PPM) 形式で出力されるが、環境変数 PNM_PORTABLE が "yes" のとき、出力は PBM、PGM、PPM のポータブル形式 (人間が読めるテキスト形式) で行われる。そうでない場合はバイナリ形式 (これがデフォルト) で行われる。

ウェブ用のベクトル画像の形式である WebCGM プロファイルにしたがった CGM 形式のファイルを plot -T cgm で生成する場合に有効な環境変数が二つある。デフォルトでは、生成される CGM ファイルはバージョン 3 の規格に沿っているが、マイクロソフト・オフィスなどの古い CGM 表示ソフトウェアはほとんど、CGM のバージョン 1 しか扱えない。そこで環境変数 CGM_MAX_VERSION を 1"、"2"、"3"、"4" のどれかに設定することで、出力される CGM のバージョンの最大値を指定できる ("4" がデフォルト)。また CGM_ENCODING で CGM ファイルのエンコーディングを指定できる。これには"clear_text" (人が読めるテキスト形式) と "binary" が指定できる ("binary" がデフォルト)。WebCGM プロファイルではバイナリ形式が要求されている。

plot -T pcl ではヒューレット・パッカードのプリンタまたはプロッタの出力形式、PCL 5 形式で出力されるが、このとき PCL_ASSIGN_COLORS 環境変数が有効である。PCL 5 出力をカラープリンタなどに出力する際に指定する。これにより色の再現性を正確にするために、内部で出力デバイスに、各色を描画するための「論理ペン」が任意に設定できるようになる。"no" にセットされるとあらかじめ設定されている色のペンだけが用いられ、他の色は濃淡の変化 (シェイディング) によって擬似的に描画される。世の中の PCL 5 デバイスは、カラーより白黒が多いため、デフォルトでは "no" であり、色はシェイディング (グレイスケール) に置き換えられる。

plot -T hpgl では HP-GL (ヒューレット・パッカード・グラフィクス・ランゲージ) 形式の出力が生成されるが、これにもいくつかの環境変数が用意されている。もっとも重要なのは HPGL_VERSION で、これには "1"、"1.5"、"2" のいずれかが指定できる ("2" がデフォルト)。"1" は純正の HP-GL、"1.5" は HP7550A グラフィクス・プロッタおよび HP758x、HP7595A、HP7596A ドラフティング・プロッタに出力できる形式 (HP-GL にいくつかの HP-GL/2 拡張を加えたもの)、"2" は新しい HP-GL/2 でそれぞれ描画するよう指示する。もしバージョンが "1" または "1.5" にセットされていると、使えるフォントはベクトル・フォントのみで、線はすべてデフォルトの太さで描かれる ('-W' は無効になる)。さらにバージョンが "1" のときは、任意の多角形を線の色で塗りつぶす機能が使えなくなる ('-q' オプションは座標軸に沿って置かれた円や長方形を塗りつぶすのに使われる)。

plot -T hpgl のときの用紙上の描画領域は HPGL_ROTATE 環境変数を"yes" に設定すると反時計回りに 90 度回転させることができる。--rotation オプションでは描画領域を回転させ、その左下隅の位置を用紙の左下隅からの相対位置で指定するため、異なった効果になる。HPGL_ROTATE 環境変数に設定できる値は、"no" と "yes" に加えて、"0"、"90"、"180"、および "270" である。"no" と "yes" は "0" と "90" とそれぞれ同じである。"180" と "270" は HPGL_VERSION が "2" のとき (デフォルトがそうである) にだけ有効である。

不透明化機能による、白い線や白い色による塗りつぶしは、HPGL_VERSION が "2" (デフォルト) で環境変数 HPGL_OPAQUE_MODE が "yes" のときに可能になる。もしこれが "no" のときは不透明化による白い色での塗りつぶしは行われず、白色の線 (通常は 0 番のペンで描かれる) も描かれない。この機能は昔の HP-GL/2 デバイスに適している。たとえば HP-GL/2 ペン・プロッタは不透明化や 0 番のペンによる白色の線には対応していない。また古い HP-GL/2 デバイスの中には不透明化機能が不十分なものもある。

デフォルトでは、plot -T hpgl では使えるペンはあらかじめ決められているものだけである。描画時点でどのペンを使うかは、HPGL_PENS 環境変数で指定され

る。HPGL_VERSION が "1" のときは HPGL_PENS のデフォルトのペンは"1=black" である。HPGL_VERSION が "1.5" または "2"のときは、HPGL_PENS のデフォルト値は "1=black:2=red:3=green:4=yellow:5=blue:6=magenta:7=cyan" である。指定の仕方は見ての通りである。HPGL_PENS を指定することで 1...31 番のどのペンにも色を割り当てることができる。指定できる色名については Appendix B [Color Names], page 158 を参照のこと。1 番のペンはいつでも使える。その色は黒である必要はない。2...31 番の他のペンはあってもなくてもよい。

HPGL_VERSION が "2" のときは plot -T hpgl では HPGL_ASSIGN_COLORS も有効である。もしこれが "yes" にセットされていれば、plot -T hpgl で使えるペンの色は HPGL_PENS による制約を受けない。1...31 番の「論理ペン」に必要に応じて色が割り当てられる。これをサポートする HP-GL/2 は LaserJet プリンタや DesignJet だが、あまり多くはないため、デフォルトでは "no" である。特に HP-GL/2 ペン・プロッタは非対応である。

plot -T tek では Tektronix 端末のための出力が行われるが、環境変数 TERM が有効である。もし TERM が "xterm"、"nxterms"、"kterm" のいずれかであればそれぞれ、plot がその時実行されている環境は X Window システム上の VT100 端末エミュレータの xterm、nxterm、kterm であると判断される。このとき、plot -T tek では描画が行われる前に、xterm に付属していて通常は隠れている Tektronix のウィンドウを、前面にポップアップさせるエスケープ・シーケンスが送られる。また描画が行われた後には、元の VT100 端末に制御を戻すためのエスケープ・シーケンスが送られる。その際、Tektronix のウィンドウは画面に残ったままになる。

TERM が "kermit"、"ansi.sys"、"nansi.sys" のいずれかの時は、MS-DOS 版の kermit 上の VT100 エミュレータ上であるとみなされる。このとき、plot -T tek では描画が行われる前に、VT100 エミュレータから Tektronix のウィンドウに切り替えるエスケープ・シーケンスが送られる。また出力には Tektronix 制御コードは、kermit 特有のものが使われる。このとき、使える色数には制限がある (ansi.sys の 16 色はサポートされる)。また描画が行われた後には、元の VT100 端末に制御を戻すためのエスケープ・シーケンスが送られる。キーボードで 'ALT' - '-' をタイプすると、VT100 モードと Tektronix モードの切り替えが手動でできる。

4 pic2plot コマンド

4.1 pic2plot の使い方

pic2plot は、pic 言語で書かれたデータを一つまたは複数のファイルから読み込み、X Window の画面に表示したり、それらの図を取り込んだファイルを出力したりできる。多くの画像フォーマットがサポートされている。

pic は、技術文書や教科書などでよくあるような四角形と矢印からなる図を描くための、ベル研で開発されたコンパクトな仕様の言語である。pic2plot は、pic 言語で記述された図を含むディレクトリとともに公開、配布できるライセンスになっている pic2plot をインストールすると、ほとんどのシステムで '/usr/share/pic2plot' または '/usr/local/share/pic2plot' ディレクトリに、ブライアン・カーニハンによる pic 言語の原論文、エリック・S・レイモンドによるその GNU 実装の入門書、W. リチャード・スティーヴンス による pic マクロのサンプルもインストールされる。

pic 言語は元々、troff 文書整形プログラムのために作られたものである。当時、pic 言語で書かれた図は pic、あるいはその GNU 版である gpic に読み込まれるという状況が想定されていた。この pic あるいは gpic については詳細な解説書があるため、この節では入力ファイルの例と pic2plot に実装された拡張機能について説明するにとどめる。

pic 言語では、一つのファイルに一つまたは複数の、四角形と矢印からなる図を入れることができる。各図は .PS という行と .PE という行に記述される。.PS....PE の間ではない部分にある記述されているものは無視される。各図は幾何学図形 (長方形、円、楕円、1/4 円 (「円弧」)、折れ線、スプライン) から構成される。円弧、折れ線、スプライン曲線は端点に矢印がつけられる。各オブジェクトには、ラベルを付けられる。ラベルは複数行になってもよい。

各オブジェクトの位置は絶対座標ではなく、そのオブジェクトの前に置かれたオブジェクトに対する相対座標で与えられる。以下に例を示す。

```
.PS
box "START"; arrow; circle dashed filled; arrow
circle diam 2 thickness 3 "This is a" "big, thick" "circle" dashed; up
arrow from top of last circle; ellipse "loopback" dashed
arrow dotted from left of last ellipse to top of last box
arc cw radius 1/2 from top of last ellipse; arrow
box "END"
.PE
```

上の例をファイルに保存し 'pic2plot -T X' に入力として与えると、X Window の画面上に、図が描画されたウィンドウが開かれる。同様に 'pic2plot -T ps' にこのデータを入力すると、同じ図が描画された PostScript データが標準出力から得られる。その PostScript データはドロー・エディタ idraw で編集できる。他に PNG 形式、PNM 形式、疑似 GIF 形式、SVG 形式、WebCGM 形式、Fig 形式 (ドロー・エディタ xfig で編集できる形式) にも同様に出力できる。それぞれ、'-T png'、'-T pnm'、'-T gif'、'-T svg'、'-T cgm'、'-T fig' オプションで得られる。

上の例には pic 言語の特徴がよく現れている。デフォルトでは、繋がっているオブジェクトが連続して描かれる。そして描画はデフォルトでは左から右の方に向けて進められる。上の例の中の 'up' 命令ではこの向きを下から上の方向に変更する。すると次のオブジェクト (大きな円の上部から伸びる矢印) は右ではなく、上を向くことになる。

オブジェクトには、大きさなどの属性がある。属性には図全体のものと、個別のオブジェクトのものとがある。たとえば円には直径、円弧には半径がある。円弧に 'cw' 属性を設定すると、円弧の向きが反時計回りではなく、時計回りになる。多くのオブジェクトでは、'dashed' や 'dotted' 属性でそのオブジェクトを描く線種を設定できる。また各オブジェクトにはラベルが付けられるが、これは一つまたは複数の文字列であり、オブジェクトの属性として設定される。文字列にはエスケープ・シーケンスが使え、フォントの切り替え、上付きや下付き、ASCII 文字ではない数学記号などが使える。Section A.4 [Text String Format], page 144 参照。

ほとんどのサイズや位置は「仮想インチ」単位で指定される。これは pic2plot 特有の方法である。pic2plot が描画を行う「描画領域」は、一辺が 8 仮想インチの正方形が想定されている。出力のページサイズに "letter" サイズ (PostScript のデフォルト) が設定された時、仮想インチは実際のインチと同じになる。しかし他のページサイズも、たとえば '--page-size a4' オプションなどで指定できる。そうした場合、1 仮想インチは単に描画領域の幅の 1/8 になる。A4 サイズを指定すると描画領域は一辺が 19.81 cm の正方形になる。

デフォルトでは各図は描画領域の中央に置かれる。この機能をオフにして '-n' オプションで図を置く絶対座標を指定することもできる。たとえば '座標 (8,8) から (4,4) に引かれた矢印' というオブジェクトだけからなる図は、図を中央に置く機能を使わない場合、矢印の端点が描画領域の中央になる。もう一方の端は右上隅になる。

線の太さは仮想インチでは指定されない。gpic との互換性を持たせるため、これは仮想ポイントで指定される。上の例では 'thickness' 属性が指定されているオブジェクトがその例である。72 仮想ポイントで 1 仮想インチである。

複数の図がある場合、X windows システム上では複数のウィンドウが開かれ、他の形式ではそれぞれのページに出力される。しかし一部の出力形式 (PNG、PNM、疑似 GIF、SVG、Illustrator、Fig) では単ページの出力しかサポートされないため、このうちのどれかが出力形式に指定されているときは、最初の図だけが出力される。現在は pic2plot ではアニメーション GIF は生成できない。

pic 言語は、変数、制御ループなど、実質的にプログラミング言語としての機能を完備しているにも関わらず、あまり評価されていない。そういった先進的な機能を使うことで pic 言語では繰り返し部分のあるような巨大な図を簡単に記述することができる。

4.2 pic2plot のコマンドライン・オプション

pic 言語は、技術文書や教科書などでよくあるような四角形と矢印からなる図を描くための、ベル研で開発されたコンパクトな仕様の言語であり、pic2plot コマンドは pic 言語で書かれた図を他の画像形式に変換するプログラムである。変換する形式は '-T' オプションで指定する。変換可能な画像形式は GNU graph および plot の両コマンドと同じである。

入力ファイルの名前をコマンドライン上で指定できる。その順番やコマンドライン・オプションの順番は、意味を持たない。ファイル名が何も指定されないか、'-' というファイル名が指定されたときは、標準入力から読み込まれる。出力は、'-T X' が指定されない限りは標準出力に行われる。'-T X' が指定されたときは X Windows の画面上のウィンドウに描画され、標準出力にはなにも出力されない。

以下に、指定できるコマンドライン・オプションのリストを載せる。コマンドライン・オプションは 3 種類に分けられる。

1. 一般的なオプション。
2. pic2plot 生出力のデータ ('-T' で何も指定されないときの出力形式) に関するオプション。

3. 情報を表示するためのオプション (‘--help’ など)

引数を取るオプションでは、説明の最初にカッコ書きでその引数の型とデフォルト値を示している。

まず、一般的なオプション。

‘-T *type*’
‘--output-format *type*’
(文字列、デフォルトは "meta") ディスプレイの種類、あるいは出力形式を *type* で指定する。指摘できる文字列は"X"、"png"、"pnm"、"gif"、"svg"、"ai"、"ps"、"cgm"、"fig"、"pcl"、"hpgl"、"regis"、"tek"、"meta" のいずれかである。それぞれ、X Window システム、PNG 形式、portable anymap (PBM/PGM/PPM) 形式、pseudo-GIF 形式、XML ベースの Scalable Vector Graphics 形式、Adobe Illustrator 形式、`idraw` で編集できる PostScript 形式、ウェブ用のベクトル画像形式である WebCGM 形式、`xfig` で編集できる形式、ヒューレット・パッカードの PCL 5 プリンタ言語、ヒューレット・パッカード・グラフィクス言語 (デフォルトでは HP-GL/2)、DEC で開発された ReGIS (remote graphics instruction set) 形式、Tektronix 形式、特定のデバイスに依存しない GNU graphics metafile 形式である。‘--display-type’ は古いので‘--output-format’ を使うべきである。

‘-d’
‘--precision-dashing’
点線および破線を描くときに、線を構成する短い線分や点をそれぞれ独立したオブジェクトとすることで、正確に描画する。デフォルトは、`pic2plot` が内部で使っている GNU `libplot` が破線と点線に関してこの機能をサポートしているかどうかによる。

このオプションを使うと、破線と点線の見栄えがよくなる。しかし、なんらかの編集可能な形式 (たとえば Illustrator、PostScript、Fig 形式など) で得られた出力を思い通りに編集するのは困難になるだろう。

‘-f *font_size*’
‘--font-size *font_size*’
フォントのサイズを、描画領域の幅に対する割合で *font_size* に指定する。*font_size*.

‘-F *font_name*’
‘--font-name *font_name*’
(文字列、デフォルトは "Helvetica"、ただし `plot -T pcl` では"Univers"、`plot -T png`、`plot -T pnm`、`plot -T gif`、`plot -T hpgl`、`plot -T regis`、`plot -T tek`、`plot` の生出力の場合は "HersheySerif") 文字列に使うフォントを *font_name* に指定する。フォント名では大文字と小文字は区別されない。もし指定されたフォントが使えない場合、デフォルトのフォントが使われる。使えるフォントは、‘-T’ オプションで何を指定するかによって異なる。すべてのフォントのリストは Section A.1 [Text Fonts], page 136 を参照のこと。`plotfont` ユーティリティ・プログラムで、使えるフォントのキャラクタ・マップを表示することができる。Chapter 6 [plotfont], page 51 参照。

‘-n’
‘--no-centering’
自動的に図をセンタリングする機能をオフにする。このオプションが指定された時は、各オブジェクトの置かれる場所を絶対座標で指定できる。たとえば "line from (0,0) to (4,4)" は描画領域の左下隅から中央までの線分を描く (描画領域は一辺が 8 仮想インチの正方形だから)。

‘-W *line_width*’
‘--line-width *line_width*’
(実数、デフォルトは −1.0) 線の太さを *line_width* にする。その数値は、描画領域の縦又は横の短い方に対する割合で指定する。負の値を指定すると、libplot ライブラリのデフォルト値が使われる。デフォルト値は描画領域の一辺の長さの 1/850 だが、‘-T X’、‘-T png’、‘-T pnm’、-T gif では 0 になる。太さに 0 を指定するとどの出力デバイスでも、そのデバイスでもっとも細いラインが描かれるが、idraw や xfig は太さ 0 の線は表示しないようになっている。

pic2plot -T hpgl では、環境変数 HPGL_VERSION に 2 よりも小さな値を指定したときは (デフォルト値は 2)、デフォルトの太さ以外の線は描けない。

‘--bg-color *name*’
(文字列、デフォルトは "white") 背景色を *name* に設定する。このオプションは pic2plot -T X、pic2plot -T png、pic2plot -T pnm、pic2plot -T gif、pic2plot -T cgm、pic2plot -T regis、pic2plot -T meta でだけ有効である。無効な色名が指定されたときは、デフォルトの色が使われる。使える色名については Appendix B [Color Names], page 158 を参照のこと。環境変数 BG_COLOR でも同様に指定できる。

‘-T png’ か ‘-T gif’ オプションのどちらかが使われているときは、環境変数 TRANSPARENT_COLOR に背景色を指定することで、透過 PNG または透過疑似 GIF 形式の画像を生成できる。Section 4.3 [pic2plot Environment], page 42 参照。‘-T svg’ か ‘-T cgm’ オプションのどちらかが使われているときは、背景色に "none" を指定することで、背景色のない画像を生成できる。

‘--bitmap-size *bitmap_size*’
(文字列、デフォルトは "570x570") 描画領域の大きさをピクセル単位で *bitmap_size* に設定する。このオプションは pic2plot -T X、pic2plot -T png、pic2plot -T pnm、pic2plot -T gif のときにだけ有効である。これらの出力形式では描画領域のサイズがピクセル単位で表現されているからである。環境変数 BITMAPSIZE でも同様に指定できる。

pic2plot -T X のときは描画のために開かれたウィンドウが描画領域になる。X Window の画面上でのこのウィンドウの位置も、コマンドライン・オプションで指定できる。たとえば *bitmap_size* を "570x570+0+0" と指定すると、開かれたウィンドウは画面の左上隅に置かれる。

もし描画領域を正方形ではなく長方形にすると、描画に使われるフォントは縦と横で違う倍率で拡大縮小されることになる。このように拡大縮小できないフォントが指定されているときは、Hershey ベクトル・フォントの "HersheySerif" などのデフォルトのスケーラブル・フォントが使われる。

後方互換性のために、X resource の Xplot.geometry を設定することで、‘--bitmap-size’ や BITMAPSIZE の代わりにディスプレイのサイズを設定できる。

‘--emulate-color *option*’
: (文字列、デフォルトは "no") *option* が "yes" のとき、カラーの出力画像をグレイスケールにして出力する。このオプションは、‘pic2plot -T pcl’ で PCL 5 対応機器への出力を行っているときでなければ、使うことはあまりないかもしれない。(白黒の LaserJet プリンタなどの白黒 PCL 5 機器が搭載しているカラーからグレイスケールへの変換機能は、HP-GL/2 の 7 種類のペンの標準色を黒に置き換えるだけ (黄色も黒になる) で、あまりいいとは言えない)。環境変数 EMULATE_COLOR を "yes" に設定しても同じ指定ができる。

‘--max-line-length *max_line_length*’
: (整数、デフォルトは 500) 一つのデータセットから描かれる一本の折れ線について、一度に描ける最大のデータ点数を *max_line_length* に指定する。データ点の数がこの数に達したときは、折れ線は複数回に分けて描画される。この時、特に通知やメッセージなどは行われない。‘-q’ オプションで塗りつぶしが指定されているときは、これは行われない。

  この指定は、いくつかの出力デバイス (昔の PostScript プリンタや HP-GL プロッタ) では出力バッファの大きさに制限があることによる。この最大点数は、環境変数 MAX_LINE_LENGTH でも指定できる。pic2plot の生出力では、各データ点ごとにリアルタイムで描画が行われるので、このオプションは無効である。

‘--page-size *pagesize*’
: (文字列、デフォルトは "letter") プロットが描かれる用紙の大きさを指定する。このオプションは pic2plot -T svg、pic2plot -T ai、pic2plot -T ps、pic2plot -T cgm、pic2plot -T fig、pic2plot -T pcl、pic2plot -T hpgl でだけ有効である。"letter" で指定される大きさは 8.5 in x 11 in である。"a0"...."a4" の ISO の用紙サイズおよび "a"...."e" の ANSI のサイズも指定できる ("letter" は "a"、"tabloid" は "b" と同じである)。"legal"、"ledger"、"b5" も指定できる。用紙サイズは、環境変数 PAGESIZE でも同様に指定できる。

  pic2plot -T ai、pic2plot -T ps、pic2plot -T pcl、pic2plot -T fig ではプロットが置かれる描画領域 (または ‘viewport’) は、デフォルトでは指定されたページの中央に置かれる正方形の領域である。pic2plot -T hpgl でも同様だが、ページの中央ではない。この描画領域の縦、横の長さは、たとえば *pagesize* を "letter,xsize=4in" や "a4,xsize=10cm,ysize=15cm" のようにすることで指定できる。この長さは負の値になってもよい (負のときは鏡像になる)。

  描画領域の位置は、デフォルトの位置に対する相対座標で指定できる。たとえば *pagesize* を "letter,yoffset=1.2in" や"a4,xoffset=−5mm,yoffset=2.0cm" のようにするとよい。または用紙の左下隅からの相対座標として描画領域の左下隅の座標を指定することで、より正確に指定することもできる。それにはたとえば *pagesize* を"letter,xorigin=2in,yorigin=3in" や "a4,xorigin=0.5cm,yorigin=0.5cm" ようにするとよい。これらのオプションは取り混ぜて使ってもよい。

  pic2plot -T svg と pic2plot -T cgm では"xoffset"、"yoffset"、"xorigin"、"yorigin" は無視される。これらの形式では、画像が最終的にウェブブラウザ上のどこに表示されるかは、画像そのものから指定できないからである。しかし "xsize" と "ysize" オプションは無視される訳ではない。Appendix C [Page and Viewport Sizes], page 159 を参照のこと。

‘--pen-color *name*’
: (String, デフォルトは "black") プロットに使われる「ペンの色」を *name* で指定する。無効な名前が色名に指定されたときは、デフォルトの色が使われる。使える色名に付いては Appendix B [Color Names], page 158 を参照のこと。

‘--rotation *angle*’
: (実数、デフォルトは 0.0) 描画領域を角度 *angle* だけ回転する。その数字の角度だけ、反時計回りに回転する。環境変数 ROTATION に値を設定することでも、同じ指定ができる。

  角度に 0 または 90 を指定することでそれぞれ、ポートレートまたはランドスケープ (用紙が縦、または横) モードになる。前衛芸術家にも便利に使ってもらえるオプションである。

以下のオプションは、pic2plot の生出力に対してだけ、つまり ‘-T’ をつけずに pic2plot を使うときにだけ有効である。その場合の pic2plot の出力は GNU metafile 形式であり、これを plot に渡すことで他の形式に変換できる。

‘-O’
‘--portable-output’
: GNU metafile 出力を、バイナリ形式 (デフォルト) ではなく、ポータブル形式 (テキスト形式、人間の読める形式) で行う。これは環境変数 META_PORTABLE を "yes" に設定することでも指定できる。

以下のオプションで、必要な情報を表示させることができる。

‘--help’ コマンドライン・オプションのリストを表示させ、そのまま実行を終了する。

‘--help-fonts’
: 利用可能なフォントのリストを表示して、そのまま終了する。出力される内容は、‘-T’ オプションで指定される出力デバイスによって変わる。pic2plot -T X、pic2plot -T svg、pic2plot -T ai、pic2plot -T ps、pic2plot -T cgm、pic2plot -T fig では PostScript 標準の 35 種類のフォントが使える。pic2plot -T svg、pic2plot -T ai、pic2plot -T pcl、pic2plot -T hpgl では 45 種類の PCL 5 標準フォントが使える。pic2plot -T pcl と pic2plot -T hpgl ではヒューレット・パッカードのベクトル・フォントが使える。上に挙げたすべての出力形式、および pic2plot -T png、pic2plot -T pnm、pic2plot -T gif、pic2plot -T regis、pic2plot -T tek では 22 種類の Hershey ベクトル・フォントが使える。pic2plot の生出力では原理的にこれらのフォントがすべて使える。どの形式にも、pic2plot 生出力から plot で変換できるからである。plotfont ユーティリティ・プログラムで、使えるフォントのキャラクタ・マップを表示することができる。Chapter 6 [plotfont], page 51 参照。

‘--list-fonts’
: ‘--help-fonts’ と同じだが、他のプログラムにパイプで渡して処理しやすいように、フォントのリストを 1 カラムで出力する。‘-T’ オプションでの出力デバイスの指定が何もないときは、サポートされているすべてのフォントのリストが表示される。

‘--version’
: pic2plot および GNU plotting utilities パッケージのバージョン番号を表示して、そのまま終了する。

4.3 pic2plot の環境変数

pic2plot の動作には、環境変数を読んでその値により制御されるものがある。BITMAPSIZE、PAGESIZE、BG_COLOR、EMULATE_COLOR、MAX_LINE_LENGTH、ROTATION については上述したとおりである。これらの環境変数はそれぞれ、コマンドライン・オプションの‘--bitmap-size’、‘--page-size’、‘--bg-color’、‘--emulate-color’、‘--max-line-length’、‘--rotation’と同じ働きを持つ。他にも、特定の出力デバイスに対してだけ有効な環境変数もある。

pic2plot -T X では X Window の画面にウィンドウが一つ開かれ、そこにプロットが描画されるが、このとき DISPLAY 環境変数が設定されていれば、それで示されるディスプレイにウィンドウが開かれる。

PNG または疑似 GIF 形式を生成する pic2plot -T png または pic2plot -T gif に対しては、二つの環境変数が用意されている。INTERLACE が "yes" のときは、出力される画像はインターレース形式になる。TRANSPARENT_COLOR に色名を指定しておくと、出力される画像のその色の部分が透明に設定され、PNG や GIF を扱う多くのアプリケーションで透明として扱われる。使える色名については Appendix B [Color Names], page 158 を参照のこと。

pic2plot -T pnm では Portable Anymap (PBM/PGM/PPM) 形式で出力されるが、環境変数 PNM_PORTABLE が "yes" のとき、出力は PBM、PGM、PPM のポータブル形式 (人間が読めるテキスト形式) で行われる。そうでない場合はバイナリ形式 (これがデフォルト) で行われる。

ウェブ用のベクトル画像の形式である WebCGM プロファイルにしたがった CGM 形式のファイルを pic2plot -T cgm で生成する場合に有効な環境変数が二つある。デフォルトでは、生成される CGM ファイルはバージョン 3 の規格に沿っているが、マイクロソフト・オフィスなどの古い CGM 表示ソフトウェアはほとんど、CGM のバージョン 1 しか扱えない。そこで環境変数 CGM_MAX_VERSION を 1"、"2"、"3"、"4" のどれかに設定することで、出力される CGM のバージョンの最大値を指定できる ("4" がデフォルト)。また CGM_ENCODING で CGM ファイルのエンコーディングを指定できる。これには"clear_text" (人が読めるテキスト形式) と "binary" が指定できる ("binary" がデフォルト)。WebCGM プロファイルではバイナリ形式が要求される。

pic2plot -T pcl ではヒューレット・パッカードのプリンタまたはプロッタの出力形式、PCL 5 形式で出力されるが、このとき PCL_ASSIGN_COLORS 環境変数が有効である。PCL 5 出力をカラープリンタなどに出力する際に指定する。これにより色の再現性を正確にするために、内部で出力デバイスに、各色を描画するための「論理ペン」が任意に設定できるようになる。"no" にセットされるとあらかじめ設定されている色のペンだけが用いられ、他の色は濃淡の変化 (シェイディング) によって擬似的に描画される。世の中の PCL 5 デバイスは、カラーより白黒が多いため、デフォルトでは "no" であり、色はシェイディング (グレイスケール) に置き換えられる。

plot -T hpgl では HP-GL (ヒューレット・パッカード・グラフィクス・ランゲージ) 形式の出力が生成されるが、これにもいくつかの環境変数が用意されている。もっとも重要なのは HPGL_VERSION で、これには "1"、"1.5"、"2" のいずれかが指定できる ("2" がデフォルト)。"1" は純正の HP-GL、"1.5" は HP7550A グラフィクス・プロッタおよび HP758x、HP7595A、HP7596A ドラフティング・プロッタに出力できる形式 (HP-GL にいくつかの HP-GL/2 拡張を加えたもの)、"2" は新しい HP-GL/2 でそれぞれ描画するよう指示する。もしバージョンが "1" または "1.5" にセットされていると、使えるフォントはベクトル・フォントのみで、線はすべてデフォルトの太さで描かれる (‘-W’ は無効になる)。さらにバージョンが "1" のときは、任意の多角形を線の色で塗りつぶす機能が使えなくなる (‘-q’ オプションは座標軸に沿って置かれた円や長方形を塗りつぶすのに使われる)。

pic2plot -T hpgl のときの用紙上の描画領域は HPGL_ROTATE 環境変数を"yes" に設定すると反時計回りに 90 度回転させることができる。--rotation オプションでは描画領域を回転させ、その左下隅の位置を用紙の左下隅からの相対位置で指定するため、異なった効果になる。HPGL_ROTATE 環境変数に設定できる値は、"no" と "yes" に加えて、"0"、"90"、"180"、および "270" である。"no" と "yes" は "0" と "90" とそれぞれ同じである。"180" と "270" は HPGL_VERSION が "2" のとき (デフォルトがそうである) にだけ有効である。

不透明化機能による、白い線や白い色による塗りつぶしは、HPGL_VERSION が "2" (デフォルト) で環境変数 HPGL_OPAQUE_MODE が "yes" のときに可能になる。もしこれが "no" のときは不透明化による白い色での塗りつぶしは行われず、白色の線 (通常は 0 番のペンで描かれる) も描かれない。この機能は昔の HP-GL/2 デバイスに適している。たとえば HP-GL/2 ペン・プロッタは不透明化や 0 番のペンによる白色の線には対応していない。また古い HP-GL/2 デバイスの中には不透明化機能が不十分なものもある。

デフォルトでは、pic2plot -T hpgl では使えるペンはあらかじめ決められているものだけである。描画時点でどのペンを使うかは、HPGL_PENS 環境変数で指定される。HPGL_VERSION が "1" のときは HPGL_PENS のデフォルトのペンは"1=black" である。HPGL_VERSION が "1.5" または "2"のときは、HPGL_PENS のデフォルト値は "1=black:2=red:3=green:4=yellow:5=blue:6=magenta:7=cyan" である。指定の仕方は見ての通りである。HPGL_PENS を指定することで 1...31 番のどのペンにも色を割り当てることができる。指定できる色名については Appendix B [Color Names], page 158 を参照のこと。1 番のペンはいつでも使える。その色は黒である必要はない。2...31 番の他のペンはあってもなくてもよい。

HPGL_VERSION が "2" のときは pic2plot -T hpgl では HPGL_ASSIGN_COLORS も有効である。もしこれが "yes" にセットされていれば、pic2plot -T hpgl で使えるペンの色は HPGL_PENS による制約を受けない。1...31 番の「論理ペン」に必要に応じて色が割り当てられる。これをサポートする HP-GL/2 は LaserJet プリンタや DesignJet だが、あまり多くはないため、デフォルトでは "no" である。特に HP-GL/2 ペン・プロッタは非対応である。

pic2plot -T tek では Tektronix 端末のための出力が行われるが、環境変数 TERM が有効である。もし TERM が "xterm"、"nxterms"、"kterm" のいずれかであればそれぞれ、pic2plot がその時実行されている環境は X Window システム上の VT100 端末エミュレータの xterm、nxterm、kterm であると判断される。このとき、pic2plot -T tek では描画が行われる前に、xterm に付属していて通常は隠れている Tektronix のウィンドウを、前面にポップアップさせるエスケープ・シーケンスが送られる。また描画が行われた後には、元の VT100 端末に制御を戻すためのエスケープ・シーケンスが送られる。その際、Tektronix のウィンドウは画面に残ったままになる。

TERM が "kermit"、"ansi.sys"、"nansi.sys" のいずれかの時は、MS-DOS 版の kermit 上の VT100 エミュレータ上であるとみなされる。このとき、pic2plot -T tek では描画が行われる前に、VT100 エミュレータから Tektronix のウィンドウに切り替えるエスケープ・シーケンスが送られる。また出力には Tektronix 制御コードは、kermit 特有のものが使われる。このとき、使える色数には制限がある (ansi.sys の 16 色はサポートされる)。また描画が行われた後には、元の VT100 端末に制御を戻すためのエスケープ・シーケンスが送られる。キーボードで 'ALT' - '-' をタイプすると、VT100 モードと Tektronix モードの切り替えが手動でできる。

5 tek2plot コマンド

5.1 tek2plot の使い方

GNU tek2plot は Tektronix 形式の画像データを変換するプログラムであり、Tektronix データを画面に表示したり、他の形式に変換したりするコマンドである。どの形式に変換するかは '-T' で指定する。指定できる形式は、"X"、"png"、"pnm"、"gif"、"svg"、"ai"、"ps"、"cgm"、"fig"、"pcl"、"hpgl"、"regis"、"tek"、"meta" で、"meta" がデフォルトであり、GNU graph、plot、pic2plot がサポートする形式と同じである。入力は plot コマンドと同様、コマンドラインでファイル名を指定するか、標準入力から与える。

Tektronix 形式の画像ファイルを生成するソフトウェアは、昔はたくさんあった。たとえば SKYMAP (http://tdc-www.harvard.edu/software/skymap) (強力な天文学画像ソフトウェア) がそうである。GNU plotting utilities のパッケージには Tektronix 形式の画像ファイルのサンプルが同梱されているので、試してみるとよい。ほとんどのシステムでは '/usr/share/tek2plot' または '/usr/local/share/tek2plot' ディレクトリにインストールされている。

Tektronix 形式は Tektronix 4010/4014 端末で描画するための、非対話型の画像フォーマットとして定義されている。詳細は Tektronix Inc. による 1974 年の文書 *4014 Service Manual* (Tektronix Part #070-1648-00) に述べられている。tek2plot では、グラフィクス入力モード ("GIN mode") やステータス表示のような対話モードは用意されていない。しかし MS-DOS 版 kermit が実装している Tektronix エミュレータのカラー拡張など、一般的になった Tektronix エミュレータの拡張機能を実装している。

5.2 tek2plot のコマンドライン・オプション

tek2plot は多くの古いソフトウェアで生成された Tektronix 形式のデータを、他の画像フォーマットに変換する。出力形式やディスプレイの種類は '-T' オプションで指定できる。指定できる形式は GNU graph、plot、pic2plot と同じである。

入力ファイルの名前をコマンドライン上で指定できる。その順番やコマンドライン・オプションの順番は、意味を持たない。ファイル名が何も指定されないか、'-' というファイル名が指定されたときは、標準入力から読み込まれる。出力は、'-T X' が指定されない限りは標準出力に行われる。'-T X' が指定されたときは X Windows の画面上のウィンドウに描画され、標準出力にはなにも出力されない。

以下に、指定できるコマンドライン・オプションのリストを載せる。コマンドライン・オプションは 3 種類に分けられる。

1. 一般的なオプション。
2. tek2plot 生出力のデータ ('-T' で何も指定されないときの出力形式) に関するオプション。
3. 情報を表示するためのオプション ('--help' など)。

引数を取るオプションでは、説明の最初にカッコ書きでその引数の型とデフォルト値を示している。

まず、一般的なオプション。

'-T *type*'
'--output-format *type*'
: (文字列、デフォルトは "meta") ディスプレイの種類、あるいは出力形式を *type* で指定する。指摘できる文字列は"X"、"png"、"pnm"、"gif"、"svg"、"ai"、"ps"、

"cgm"、"fig"、"pcl"、"hpgl"、"regis"、"tek"、"meta" のいずれかである。それぞれ、X Window システム、PNG 形式、portable anymap (PBM/PGM/PPM) 形式、pseudo-GIF 形式、XML ベースの Scalable Vector Graphics 形式、Adobe Illustrator 形式、`idraw` で編集できる PostScript 形式、ウェブ用のベクトル画像形式である WebCGM 形式、`xfig` で編集できる形式、ヒューレット・パッカードの PCL 5 プリンタ言語、ヒューレット・パッカード・グラフィクス言語 (デフォルトでは HP-GL/2)、DEC で開発された ReGIS (remote graphics instruction set) 形式、Tektronix 形式、特定のデバイスに依存しない GNU graphics metafile 形式である。'`--display-type`' は古いので '`--output-format`' を使うべきである。

'`-p` *n*'
'`--page-number` *n*'
(正の整数) 入力として与えられる一つあるいは複数の Tektronix データファイルから、*n* ページ目だけを取り出して変換する。一つの Tektronix データファイルには複数のページを持つことができるが、ページ番号は 0 から始まる。

'`-p`' オプションが指定されない場合、デフォルトでは空でないすべてのページを順に表示する。たとえば `tek2plot -T X` では各ページを、それぞれのウィンドウを X window 上に開いて表示する。'`-T png`'、'`-T pnm`'、'`-T gif`'、'`-T svg`'、'`-T ai`'、'`-T fig`' のいずれかが指定されているときは、デフォルトでは最初のページだけが描画される。それは PNG、 PNG、疑似 GIF、SVG、AI、Fig 形式では単ページの画像しかサポートされないからである。

Tektronix データの多くは、ページ数が 1 (ページ 0) か、2 (空のページ (0 ページ目) と、1 ページ目) である。GNU plotting utilities の Tektronix 形式の出力 (`graph -T tek` などによる) は通常、後者である。

'`-F` *font_name*'
'`--font-name` *font_name*'
(文字列、デフォルトは "Courier"、ただし `tek2plot -T png`、`tek2plot -T pnm`、`tek2plot -T gif`、`tek2plot -T hpgl`、`tek2plot -T regis`、`tek2plot` の生出力の場合は "HersheySerif") 文字列に使うフォントを *font_name* に指定する。フォント名では大文字と小文字は区別されない。Courier 系以外のフォントが指定された場合は '`--position-chars`' を使う (後述)。すべてのフォントのリストは Section A.1 [Text Fonts], page 136 を参照のこと。もし指定されたフォントが使えない場合、デフォルトのフォントが使われる。

`tek2plot -T pcl` で PCL 5 形式の画像を生成して LaserJet III プリンタで出力したい場合は Courier 以外のフォントを指定した方がよい。LaserJet III プリンタはヒューレット・パッカードの最初の PCL 5 対応プリンタであり、スケーラブルな Courier 書体をサポートしていない。このプリンタでサポートされている PCL 5 は CGTimes と Univers ファミリの計 8 種類だけである。詳細は Section A.1 [Text Fonts], page 136 のこと。

'`-W` *line_width*'
'`--line-width` *line_width*'
(実数、デフォルトは −1.0) プロットの外枠の線の太さを *line_width* にする。数値は描画領域の縦か横の短い方に対する割合として指定する。負の値を指定すると、`libplot` ライブラリで設定しているデフォルト値が使われる。通常、デフォルト値は描画領域の 1/850 だが、'`-T X`'、'`-T png`'、'`-T pnm`'、`-T gif` のときは 0 になる。0 になったときは、そのデバイスで描けるもっとも細い線が使われる。

これはどのデバイスでもそうである。しかし idraw や xfig では、太さ 0 の線は描かれない。

tek2plot -T regis では、線の太さはデフォルト値以外は使えない。tek2plot -T hpgl でも環境変数 HPGL_VERSION に 2 よりも小さな値をセットしているときは、同様である (2 がデフォルト値)。

‘--bg-color *name*’

(文字列、デフォルトは "white") 背景色を *name* に設定する。このオプションは tek2plot -T X、tek2plot -T png、tek2plot -T pnm、tek2plot -T gif、tek2plot -T cgm、tek2plot -T regis、tek2plot -T meta でだけ有効である。無効な色名が指定されたときは、デフォルトの色が使われる。使える色名については Appendix B [Color Names], page 158 を参照のこと。環境変数 BG_COLOR でも同様に指定できる。

‘-T png’ か ‘-T gif’ オプションのどちらかが使われているときは、環境変数 TRANSPARENT_COLOR に背景色を指定することで、透過 PNG または透過疑似 GIF 形式の画像を生成できる。Section 5.3 [tek2plot Environment], page 49 参照。‘-T svg’ か ‘-T cgm’ オプションのどちらかが使われているときは、背景色に "none" を指定することで、背景色のない画像を生成できる。

‘--bitmap-size *bitmap_size*’

(文字列、デフォルトは "570x570") 描画領域の大きさをピクセル単位で *bitmap_size* に設定する。このオプションは tek2plot -T X、tek2plot -T png、tek2plot -T pnm、tek2plot -T gif のときにだけ有効である。これらの出力形式では描画領域のサイズがピクセル単位で表現されているからである。環境変数 BITMAPSIZE でも同様に指定できる。

tek2plot -T X のときは描画のために開かれたウィンドウが描画領域になる。X Window の画面上でのこのウィンドウの位置も、コマンドライン・オプションで指定できる。たとえば *bitmap_size* を "570x570+0+0" と指定すると、開かれたウィンドウは画面の左上隅に置かれる。

もし描画領域を正方形ではなく長方形にすると、描画に使われるフォントは縦と横で違う倍率で拡大縮小されることになる。このように拡大縮小できないフォントが指定されているときは、Hershey ベクトル・フォントの "HersheySerif" などのデフォルトのスケーラブル・フォントが使われる。

後方互換性のために、X resource の Xplot.geometry を設定することで、‘--bitmap-size’ や BITMAPSIZE の代わりにディスプレイのサイズを設定できる。

‘--emulate-color *option*’

(文字列、デフォルトは "no") *option* が "yes" のとき、カラーの出力画像をグレイスケールにして出力する。このオプションは、‘tek2plot -T pcl’ で PCL 5 対応機器への出力を行っているときでなければ、使うことはあまりないかもしれない。(白黒の LaserJet プリンタなどの白黒 PCL 5 機器が搭載しているカラーからグレイスケールへの変換機能は、HP-GL/2 の 7 種類のペンの標準色を黒に置き換えるだけ (黄色も黒になる) で、あまりいいとは言えない)。環境変数 EMULATE_COLOR を "yes" に設定しても同じ指定ができる。

‘--max-line-length *max_line_length*’

(整数、デフォルトは 500) 一つのデータセットから描かれる一本の折れ線について、一度に描ける最大のデータ点数を *max_line_length* に指定する。データ点の

数がこの数に達したときは、折れ線は複数回に分けて描画される。この時、特に通知やメッセージなどは行われない。

この指定は、いくつかの出力デバイス (昔の PostScript プリンタや HP-GL プロッタ) では出力バッファの大きさに制限があることによる。この最大点数は、環境変数 `MAX_LINE_LENGTH` でも指定できる。`tek2plot` の生出力では、各データ点ごとにリアルタイムで描画が行われるので、このオプションは無効である。

‘`--page-size` *pagesize*’

(文字列、デフォルトは "letter") プロットが描かれる用紙の大きさを指定する。このオプションは `tek2plot -T svg`、`tek2plot -T ai`、`tek2plot -T ps`、`tek2plot -T cgm`、`tek2plot -T fig`、`tek2plot -T pcl`、`tek2plot -T hpgl` でだけ有効である。"letter" で指定される大きさは 8.5 in x 11 in である。"a0"...."a4" の ISO の用紙サイズおよび "a"...."e" の ANSI のサイズも指定できる ("letter" は "a"、"tabloid" は "b" と同じである)。"legal"、"ledger"、"b5" も指定できる。用紙サイズは、環境変数 `PAGESIZE` でも同様に指定できる。

`tek2plot -T ai`、`tek2plot -T ps`、`tek2plot -T pcl`、`tek2plot -T fig` ではプロットが置かれる描画領域 (または ‘viewport’) は、デフォルトでは指定されたページの中央に置かれる正方形の領域である。`tek2plot -T hpgl` でも同様だが、ページの中央ではない。この描画領域の縦、横の長さは、たとえば *pagesize* を "letter,xsize=4in" や "a4,xsize=10cm,ysize=15cm" のようにすることで指定できる。この長さは負の値になってもよい (負のときは鏡像になる)。

描画領域の位置は、デフォルトの位置に対する相対座標で指定できる。たとえば *pagesize* を "letter,yoffset=1.2in" や"a4,xoffset=−5mm,yoffset=2.0cm" のようにするとよい。または用紙の左下隅からの相対座標として描画領域の左下隅の座標を指定することで、より正確に指定することもできる。それにはたとえば *pagesize* を"letter,xorigin=2in,yorigin=3in" や "a4,xorigin=0.5cm,yorigin=0.5cm" ようにするとよい。これらのオプションは取り混ぜて使ってもよい。

`tek2plot -T svg` と `tek2plot -T cgm` では"xoffset"、"yoffset"、"xorigin"、"yorigin" は無視される。これらの形式では、画像が最終的にウェブブラウザ上のどこに表示されるかは、画像そのものから指定できないからである。しかし "xsize" と "ysize" オプションは無視される訳ではない。Appendix C [Page and Viewport Sizes], page 159 を参照のこと。

‘`--pen-color` *name*’

(String, デフォルトは "black") プロットに使われる「ペンの色」を *name* で指定する。無効な名前が色名に指定されたときは、デフォルトの色が使われる。使える色名に付いては Appendix B [Color Names], page 158 を参照のこと。

‘`--position-chars`’

文字列中の各文字を、一文字ずつ独立に描画する。文字のフォントが courier ファミリーではなく、さらに等幅でない場合はこのオプションを指定した方がよい。これを指定すると、出力された画像ファイルを `xfig` や `idraw` で編集するのが煩雑にはなるが、文字列の見栄えが良くなる。

‘`--rotation` *angle*’

(実数、デフォルトは 0.0) 描画領域中のプロットを角度 *angle* だけ回転する。その数字の角度だけ、反時計回りに回転する。環境変数 `ROTATION` に値を設定することでも、同じ指定ができる。

角度に 0 または 90 を指定することでそれぞれ、ポートレートまたはランドスケープ (用紙が縦、または横) モードになる。前衛芸術家にも便利に使ってもらえるオプションである。

‘--use-tek-fonts’
Tektronix 4010/4014 で使われていたオリジナルのフォントを使う。よりオリジナルに忠実な描画を再現できる。このオプションは tek2plot -T X のときにだけ有効である。オリジナルの 4 種類の Tektronix フォントをビットマップにしたものを GNU plotting utilities のパッケージに tekfont0...tekfont3 という名前で同梱しており、簡単に X Window システムにインストールすることができる。このオプションを指定するときは、‘--bitmap-size 1024x1024’ オプションを指定するか、または X のリソース Xplot.geometry でウィンドウをのサイズを 1024x1024 ピクセルにしないと、正確な描画ができない。ビットマップ・フォントの拡大・縮小をしないように制限しているためである。

Tektronix のオリジナルフォントが文字列で使われているような Tektronix 形式のファイルがすでにあるような状況では、このオプションは意味があるだろう。しかし GNU plotting utilities で出力する Tektronix 形式 (graph -T tek の出力など) ではこのオリジナルのフォントは使わない。

以下のオプションは、tek2plot の生出力に対してだけ、つまり ‘-T’ をつけずに tek2plot を使うときにだけ有効である。その場合の tek2plot の出力は GNU metafile 形式であり、これを plot に渡すことで他の形式に変換できる。

‘-O’
‘--portable-output’
GNU metafile 出力を、バイナリ形式 (デフォルト) ではなく、ポータブル形式 (テキスト形式、人間の読める形式) で行う。これは環境変数 META_PORTABLE を "yes" に設定することでも指定できる。

以下のオプションで、必要な情報を表示させることができる。

‘--help’ コマンドライン・オプションのリストを表示させ、そのまま実行を終了する。

‘--help-fonts’
利用可能なフォントのリストを表示して、そのまま終了する。出力される内容は、‘-T’ オプションで指定される出力デバイスによって変わる。tek2plot -T X、tek2plot -T svg、tek2plot -T ai、tek2plot -T ps、tek2plot -T cgm、tek2plot -T fig では PostScript 標準の 35 種類のフォントが使える。tek2plot -T svg、tek2plot -T ai、tek2plot -T pcl、tek2plot -T hpgl では 45 種類の PCL 5 標準フォントが使える。tek2plot -T pcl と tek2plot -T hpgl ではヒューレット・パッカードのベクトル・フォントが使える。上に挙げたすべての出力形式、および tek2plot -T png、tek2plot -T pnm、tek2plot -T gif、tek2plot -T regis、tek2plot -T tek では 22 種類の Hershey ベクトル・フォントが使える。tek2plot の生出力では原理的にこれらのフォントがすべて使える。どの形式にも、tek2plot 生出力から plot で変換できるからである。plotfont ユーティリティ・プログラムで、使えるフォントのキャラクタ・マップを表示することができる。Chapter 6 [plotfont], page 51 参照。

‘--list-fonts’
‘--help-fonts’ と同じだが、他のプログラムにパイプで渡して処理しやすいように、フォントのリストを 1 カラムで出力する。‘-T’ オプションでの出力デバイ

スの指定が何もないときは、サポートされているすべてのフォントのリストが表示される。

‘--version’
tek2plot および GNU plotting utilities パッケージのバージョン番号を表示して、そのまま終了する。

5.3 tek2plot の環境変数

graph や plot の動作を制御するための環境変数は、tek2plot の動作の制御にも使われる。重複になるが、以下にそれを挙げる。

BITMAPSIZE、PAGESIZE、BG_COLOR、EMULATE_COLOR、MAX_LINE_LENGTH、ROTATION については上述したとおりである。これらの環境変数はそれぞれ、コマンドライン・オプションの ‘--bitmap-size’、‘--page-size’、‘--bg-color’、‘--emulate-color’、‘--max-line-length’、‘--rotation’ と同じ働きを持つ。他にも、特定の出力デバイスに対してだけ有効な環境変数もある。

tek2plot -T X では X Window の画面にウィンドウが一つ開かれ、そこにプロットが描画されるが、このとき DISPLAY 環境変数が設定されていれば、それで示されるディスプレイにウィンドウが開かれる。

PNG または疑似 GIF 形式を生成する tek2plot -T png または tek2plot -T gif に対しては、二つの環境変数が用意されている。INTERLACE が "yes" のときは、出力される画像はインターレース形式になる。TRANSPARENT_COLOR に色名を指定しておくと、出力される画像のその色の部分が透明に設定され、PNG や GIF を扱う多くのアプリケーションで透明として扱われる。使える色名については Appendix B [Color Names], page 158 を参照のこと。

tek2plot -T pnm では Portable Anymap (PBM/PGM/PPM) 形式で出力されるが、環境変数 PNM_PORTABLE が "yes" のとき、出力は PBM、PGM、PPM のポータブル形式 (人間が読めるテキスト形式) で行われる。そうでない場合はバイナリ形式 (これがデフォルト) で行われる。

ウェブ用のベクトル画像の形式である WebCGM プロファイルにしたがった CGM 形式のファイルを tek2plot -T cgm で生成する場合に有効な環境変数が二つある。デフォルトでは、生成される CGM ファイルはバージョン 3 の規格に沿っているが、マイクロソフト・オフィスなどの古い CGM 表示ソフトウェアはほとんど、CGM のバージョン 1 しか扱えない。そこで環境変数 CGM_MAX_VERSION を 1"、"2"、"3"、"4" のどれかに設定することで、出力される CGM のバージョンの最大値を指定できる ("4" がデフォルト)。また CGM_ENCODING で CGM ファイルのエンコーディングを指定できる。これには"clear_text" (人が読めるテキスト形式) と "binary" が指定できる ("binary" がデフォルト)。WebCGM プロファイルではバイナリ形式が要求される。

tek2plot -T pcl ではヒューレット・パッカードのプリンタまたはプロッタの出力形式、PCL 5 形式で出力されるが、このとき PCL_ASSIGN_COLORS 環境変数が有効である。PCL 5 出力をカラープリンタなどに出力する際に指定する。これにより色の再現性を正確にするために、内部で出力デバイスに、各色を描画するための「論理ペン」が任意に設定できるようになる。"no" にセットされるとあらかじめ設定されている色のペンだけが用いられ、他の色は濃淡の変化 (シェイディング) によって擬似的に描画される。世の中の PCL 5 デバイスは、カラーより白黒が多いため、デフォルトでは "no" であり、色はシェイディング (グレイスケール) に置き換えられる。

tek2plot -T hpgl では HP-GL (ヒューレット・パッカード・グラフィクス・ランゲージ) 形式の出力が生成されるが、これにもいくつかの環境変数が用意されている。もっとも重要な

のは HPGL_VERSION で、これには "1"、"1.5"、"2" のいずれかが指定できる ("2" がデフォルト)。"1" は純正の HP-GL、"1.5" は HP7550A グラフィクス・プロッタおよび HP758x、HP7595A、HP7596A ドラフティング・プロッタに出力できる形式 (HP-GL にいくつかの HP-GL/2 拡張を加えたもの)、"2" は新しい HP-GL/2 でそれぞれ描画するよう指示する。もしバージョンが "1" または "1.5" にセットされていると、使えるフォントはベクトル・フォントのみで、線はすべてデフォルトの太さで描かれる ('-W' は無効になる)。

tek2plot -T hpgl のときの用紙上の描画領域は HPGL_ROTATE 環境変数を"yes" に設定すると反時計回りに 90 度回転させることができる。--rotation オプションでは描画領域を回転させ、その左下隅の位置を用紙の左下隅からの相対位置で指定するため、異なった効果になる。HPGL_ROTATE 環境変数に設定できる値は、"no" と "yes" に加えて、"0"、"90"、"180"、および "270" である。"no" と "yes" は "0" と "90" とそれぞれ同じである。"180" と "270" は HPGL_VERSION が "2" のとき (デフォルトがそうである) にだけ有効である。

不透明化機能による、白い線や白い色による塗りつぶしは、HPGL_VERSION が "2" (デフォルト) で環境変数 HPGL_OPAQUE_MODE が "yes" のときに可能になる。もしこれが "no" のときは不透明化による白い色での塗りつぶしは行われず、白色の線 (通常は 0 番のペンで描かれる) も描かれない。この機能は昔の HP-GL/2 デバイスに適している。たとえば HP-GL/2 ペン・プロッタは不透明化や 0 番のペンによる白色の線には対応していない。また古い HP-GL/2 デバイスの中には不透明化機能が不十分なものもある。

デフォルトでは、tek2plot -T hpgl では使えるペンはあらかじめ決められているものだけである。描画時点でどのペンを使うかは、HPGL_PENS 環境変数で指定される。HPGL_VERSION が "1" のときは HPGL_PENS のデフォルトのペンは"1=black" である。HPGL_VERSION が "1.5" または "2"のときは、HPGL_PENS のデフォルト値は "1=black:2=red:3=green:4=yellow:5=blue:6=magenta:7=cyan" である。指定の仕方は見ての通りである。HPGL_PENS を指定することで 1...31 番のどのペンにも色を割り当てることができる。指定できる色名については Appendix B [Color Names], page 158 を参照のこと。1 番のペンはいつでも使える。その色は黒である必要はない。2...31 番の他のペンはあってもなくてもよい。

HPGL_VERSION が "2" のときは tek2plot -T hpgl では HPGL_ASSIGN_COLORS も有効である。もしこれが "yes" にセットされていれば、tek2plot -T hpgl で使えるペンの色は HPGL_PENS による制約を受けない。1...31 番の「論理ペン」に必要に応じて色が割り当てられる。これをサポートする HP-GL/2 は LaserJet プリンタや DesignJet だが、あまり多くはないため、デフォルトでは "no" である。特に HP-GL/2 ペン・プロッタは非対応である。

6 plotfont コマンド

6.1 plotfont の使い方

plotfont は GNU plotting utilities の libplot ライブラリを内部で利用している graph、plot、pic2plot、tek2plot プログラムで使えるフォントのキャラクタ・マップを表示する、単純なユーティリティ・プログラムである。キャラクタ・マップは X Window システムのディスプレイ上に表示、または他の画像フォーマットに出力できる。'-T' オプションで、出力する画像フォーマットを指定できる。指定できるのは"X"、"png"、"pnm"、"gif"、"svg"、"ai"、"ps"、"cgm"、"fig"、"pcl"、"hpgl"、"regis"、"tek"、"meta" ("meta" がデフォルト) である。

どのフォントが使えるかは、'-T' オプションで何が指定されているかによって変わる。使えるフォントのリストを表示させるには、'--help-fonts' オプションを使う。たとえば、

```
plotfont -T ps --help-fonts
```

とすると出力が PostScript 形式のときに使えるフォントのリストが表示される。その中には "Times-Roman" があるが、以下のようにすると、

```
plotfont -T ps Times-Roman > map.ps
```

このフォントの後半、表示に使える (制御文字でない) アスキー文字のキャラクタ・マップが表示される。キャラクタ・マップは 12 x 8 の表で、各セルの中央に文字が表示される。'-2' オプションを付けるとフォントの前半のマップが表示される。

ほとんどの組み込みフォントは ISO-Latin-1 エンコードである。それらのフォントの前半は ISO-Latin-1 でエンコードされるが、たとえば"HersheyCyrillic" はそうでない。以下のようにすると、

```
plotfont -T ps -2 HersheyCyrillic > map.ps
```

この場合のエンコード、KOI8-R エンコードでどうなるかが表示される。KOI8-R エンコードは昔のソヴィエト連邦の Unix およびネットワーク・アプリケーションで標準だったエンコードである。"dingbats" フォントとも呼ばれる "ZapfDingbats" や "Wingdings" なども独自のエンコードを使っている。GNU plotting utilities をインストールしてもほとんどの場合、PostScript 形式を出力する際に Wingdings フォントは使えない。しかし PCL 5 および HP-GL/2 形式では Wingdings フォントが使える。

```
plotfont -T hpgl Wingdings > map.plt
```

上のコマンドを実行すると HP-GL/2 形式で Wingdings フォントのキャラクタ・マップが得られる。同様に plotfont -T pcl Wingdings とすると、LaserJet や他の PCL 5 プリンタで印刷できる PCL 5 形式で Wingdings フォントのキャラクタ・マップが得られる。

GNU plotting utilities には合計で 100 種類以上のフォントが組み込まれている。Section A.1 [Text Fonts], page 136 参照。出力デバイスに X display を使うときは、組み込みフォント以外のフォントも使える。以下のようにすると、

```
plotfont -T X --help-fonts
```

利用できる組み込みフォントのリストが表示される。それには Hershey フォントも PostScript フォントも含まれている。これに加えて、表示に使われている X display で利用できるフォントも使える。xlsfonts コマンドで利用中の X display で使えるフォントのリストが表示される。フォント名の多くは XLFD 形式と呼ばれる形式で表示される。plotting utilities は X の font 名を XLFD 形式の短縮形で表示する。たとえば、"CharterBT-Roman" という

フォントは多くの X display で使えるが、これの XLDF 名は"-bitstream-charter-medium-r-normal–0-0-0-0-p-0-iso8859-1" で、短縮形は"charter-medium-r-normal" である。以下のようにすると、

```
plotfont -T X charter-medium-r-normal
```

X window の画面上にウィンドウが開いて、このフォントのキャラクタ・マップが表示される。

'-T X' オプションを指定するとき、'--bitmap-size' を使えば開かれるウィンドウのサイズを指定できる。最近の X display では縦、横方向で異なる倍率でフォントがスケーリングできる。たとえば上の例のコマンドに '--bitmap-size 600x300' を追加すると、キャラクタ・マップが表示され、その中の CharterBT-Roman の文字が X Window の機能により拡大縮小される。

6.2 plotfont のコマンドライン・オプション

plotfont は GNU plotting utilities の libplot ライブラリを内部で利用している graph、plot、pic2plot、tek2plot プログラムで使えるフォントのキャラクタ・マップを表示するユーティリティである。キャラクタ・マップは X Window システムのディスプレイ上に表示、または他の画像形式に出力できる。'-T' オプションで出力する画像形式を指定できる。

キャラクタ・マップを出力するフォント名は、plotfont のコマンドライン上のどの位置でもよい。つまりフォント名やコマンドライン・オプションの順序関係には意味はない。'-T X' オプションが指定されない限り、キャラクタ・マップは標準出力に書き出される。'-T X' の場合、X Window の画面上の一つのウィンドウにキャラクタ・マップが描画され、ファイルには何も書き出されない。

以下に指定できるコマンドライン・オプションを列挙する。コマンドライン・オプションには、3種類ある。

1. 一般的なオプション。
2. plotfont の生出力のデータ ('-T' で何も指定されないときの出力形式) に関するオプション。
3. 情報を表示するためのオプション ('--help' など)。

引数を取るオプションでは、説明の最初にカッコ書きでその引数の型とデフォルト値を示している。

まず、一般的なオプション。

'-1'
'--lower-half'
: 指定されたフォントの前半 (lower half) のキャラクタ・マップを表示する。これがデフォルトである。

'-2'
'--upper-half'
: 指定されたフォントの後半 (upper half) のキャラクタ・マップを表示する。

'-o'
'--octal' 十進数ではなく八進数でキャラクタの番号を表示する (十進数がデフォルト)。

'-x'
'--hexadecimal'
: 十進数ではなく十六進数でキャラクタの番号を表示する (十進数がデフォルト)。

‘--box’ 各キャラクタの左右を横幅を示す点線で囲む。デフォルトではそうしないようになっている。

‘-j *row*’
‘--jis-row *row*’
JIS [Japanese Industrial Standard] X0208 エンコードの日本語フォントの指定された行 (*row*) のキャラクタ・マップを表示する。これは現在、HersheyEUC [Extended Unix Code] フォントでだけ有効である。もしこれが指定された場合、‘-1’ および ‘-2’ は無視され、このオプションが優先される。

指定できる行番号は 1...94 である。JIS X0208 ではローマ字は 3、日本語の表音文字 (平仮名とカタカナ) は 4 と 5 である。ギリシャ文字とキリル文字は 6 と 7 である。日本語の表意文字 (漢字) は 16...84 である。行番号 16...47 には、よく使われる JIS 第一水準の漢字が入っている。漢字は音読み (古代の中国語読み) の順に並べられている。行番号 48...84 は JIS 第二水準の (さほど使われない) 漢字が入っている。

HersheyEUC フォントには、JIS 第一水準の 2965 文字のうち 596 文字と、第二水準から 7 文字が実装されている。これらは 8-bit EUC-JP でエンコードされている。これは多バイトエンコードで、JIS X0208 と同じくアスキー文字も含まれている。アスキー文字は普通、最高位ビットが立っていない 1 バイトで表現されている。一方 JIS X0208 の各文字は 2 バイトで表現され、どちらのバイトも最高位ビットが立っている。第一バイトは行番号に 32 を加えた値、第二バイトがその文字の番号である。

‘-T *type*’
‘--output-format *type*’
(文字列、デフォルトは "meta") ディスプレイの種類、あるいは出力形式を *type* で指定する。指摘できる文字列は"X"、"png"、"pnm"、"gif"、"svg"、"ai"、"ps"、"cgm"、"fig"、"pcl"、"hpgl"、"regis "、"tek"、"meta" のいずれかである。それぞれ、X Window システム、PNG 形式、portable anymap (PBM/PGM/PPM) 形式、pseudo-GIF 形式、XML ベースの Scalable Vector Graphics 形式、Adobe Illustrator 形式、`idraw` で編集できる PostScript 形式、ウェブ用のベクトル画像形式である WebCGM 形式、`xfig` で編集できる形式、ヒューレット・パッカードの PCL 5 プリンタ言語、ヒューレット・パッカード・グラフィクス言語 (デフォルトでは HP-GL/2)、DEC で開発された ReGIS (remote graphics instruction set) 形式、Tektronix 形式、特定のデバイスに依存しない GNU graphics metafile 形式である。‘--display-type’ は古いので ‘--output-format’ を使うべきである。

PNG、PNM、疑似 GIF、SVG、AI、Fig の各形式では複数ページの画像をサポートしていないので、‘-T png’、g‘-T pnm’、the ‘-T gif’、the ‘-T svg’、the ‘-T ai’、the ‘-T fig’ のいずれかが指定されているときは、最初に指定されたフォントのキャラクタ・マップだけが出力される。

‘--bg-color *name*’
(文字列、デフォルトは "white") 背景色を *name* に設定する。このオプションは `plotfont -T X`、`plotfont -T png`、`plotfont -T pnm`、`plotfont -T gif`、`plotfont -T cgm`、`plotfont -T regis`、`plotfont -T meta` でだけ有効である。無効な色名が指定されたときは、デフォルトの色が使われる。使える色名については Appendix B [Color Names], page 158 を参照のこと。環境変数 `BG_COLOR` でも同様に指定できる。

‘-T png’ か ‘-T gif’ オプションのどちらかが使われているときは、環境変数 TRANSPARENT_COLOR に背景色を指定することで、透過 PNG または透過疑似 GIF 形式の画像を生成できる。Section 6.3 [plotfont Environment], page 56 参照。‘-T svg’ か ‘-T cgm’ オプションのどちらかが使われているときは、背景色に "none" を指定することで、背景色のない画像を生成できる。

‘--bitmap-size *bitmap_size*’
(文字列、デフォルトは "570x570") 描画領域の大きさをピクセル単位で *bitmap_size* に設定する。このオプションは plotfont -T X、plotfont -T png、plotfont -T pnm、plotfont -T gif のときにだけ有効である。これらの出力形式では描画領域のサイズがピクセル単位で表現されているからである。環境変数 BITMAPSIZE でも同様に指定できる。

plotfont -T X のときは描画のために開かれたウィンドウが描画領域になる。X Window の画面上でのこのウィンドウの位置も、コマンドライン・オプションで指定できる。たとえば *bitmap_size* を "570x570+0+0" と指定すると、開かれたウィンドウは画面の左上隅に置かれる。

もし描画領域を正方形ではなく長方形にすると、描画に使われるフォントは縦と横で違う倍率で拡大縮小されることになる。

後方互換性のために、X resource の Xplot.geometry を設定することで、‘--bitmap-size’ や BITMAPSIZE の代わりにディスプレイのサイズを設定できる。

‘--emulate-color *option*’
(文字列、デフォルトは "no") *option* が "yes" のとき、カラーの出力画像をグレイスケールにして出力する。このオプションは、‘plotfont -T pcl’ で PCL 5 対応機器への出力を行っているときでなければ、使うことはあまりないかもしれない。(白黒の LaserJet プリンタなどの白黒 PCL 5 機器が搭載しているカラーからグレイスケールへの変換機能は、HP-GL/2 の 7 種類のペンの標準色を黒に置き換えるだけ (黄色も黒になる) で、あまりいいとは言えない)。

‘--numbering-font-name *font_name*’
(文字列、デフォルトは "Helvetica"、ただし plotfont -T pcl では "Univers"、plotfont -T png、plotfont -T pnm、plotfont -T gif、plotfont -T hpgl、plotfont -T regis、plotfont -T tek では "HersheySerif") キャラクタ・マップ中で各文字に付けられている番号に使うフォントを *font_name* にする。

‘--page-size *pagesize*’
(文字列、デフォルトは "letter") プロットが描かれる用紙の大きさを指定する。このオプションは plotfont -T svg、plotfont -T ai、plotfont -T ps、plotfont -T fig、plotfont -T pcl、plotfont -T hpgl でだけ有効である。"letter" で指定される大きさは 8.5 in x 11 in である。"a0"...."a4" の ISO の用紙サイズおよび "a"...."e" の ANSI のサイズも指定できる ("letter" は "a"、"tabloid" は "b" と同じである)。"legal"、"ledger"、"b5" も指定できる。用紙サイズは、環境変数 PAGESIZE でも同様に指定できる。

For plotfont -T ai、plotfont -T ps、plotfont -T pcl、plotfont -T fig ではプロットが置かれる描画領域 (または ‘viewport’) は、デフォルトでは指定されたページの中央に置かれる正方形の領域である。tek2plot -T hpgl でも同様だが、ページの中央ではない。この描画領域の縦、横の長さは、たとえば *pagesize*

を "letter,xsize=4in" や "a4,xsize=10cm,ysize=15cm" のようにすることで指定できる。この長さは負の値になってもよい (負のときは鏡像になる)。

描画領域の位置は、デフォルトの位置に対する相対座標で指定できる。たとえば *pagesize* を "letter,yoffset=1.2in" や"a4,xoffset=−5mm,yoffset=2.0cm" のようにするとよい。または用紙の左下隅からの相対座標として描画領域の左下隅の座標を指定することで、より正確に指定することもできる。それにはたとえば *pagesize* を"letter,xorigin=2in,yorigin=3in" や "a4,xorigin=0.5cm,yorigin=0.5cm" ようにするとよい。これらのオプションは取り混ぜて使ってもよい。

`plotfont -T svg` および `plotfont -T cgm` では"xoffset"、"yoffset"、"xorigin"、"yorigin" は無視される。これらの形式では、画像が最終的にウェブブラウザ上のどこに表示されるかは、画像そのものから指定できないからである。しかし "xsize" と "ysize" オプションは無視される訳ではない。Appendix C [Page and Viewport Sizes], page 159 を参照のこと。

‘`--pen-color` *`name`*’
(String, デフォルトは "black") 描画に使われる「ペンの色」を *name* で指定する。無効な名前が色名に指定されたときは、デフォルトの色が使われる。使える色名に付いては Appendix B [Color Names], page 158 を参照のこと。

‘`--rotation` *`angle`*’
(実数、デフォルトは 0.0) 描画領域中のプロットを角度 *angle* だけ回転する。その数字の角度だけ、反時計回りに回転する。環境変数 `ROTATION` に値を設定することでも、同じ指定ができる。

角度に 0 または 90 を指定することでそれぞれ、ポートレートまたはランドスケープ (用紙が縦、または横) モードになる。前衛芸術家にも便利に使ってもらえるオプションである。

‘`--title-font-name` *`font_name`*’
(文字列) 各キャラクタ・マップののタイトルに使われるフォントを *font_name* に設定する。通常は、キャラクタ・マップが表示されているフォントをタイトルの文字列に使う。たとえば "ZapfDingbats" とか "Wingdings" のような、普通の文字ではないフォントのキャラクタ・マップを表示させるときに、このオプションを使う。

以下のオプションは、`plotfont` の生出力に対してだけ、つまり ‘`-T`’ をつけずに `plotfont` を使うときにだけ有効である。その場合の `plotfont` の出力は GNU metafile 形式であり、これを `plot` に渡すことで他の形式に変換できる。

‘`-O`’
‘`--portable-output`’
GNU metafile 出力を、バイナリ形式 (デフォルト) ではなく、ポータブル形式 (テキスト形式、人間の読める形式) で行う。これは環境変数 `META_PORTABLE` を "yes" に設定することでも指定できる。

以下のオプションで、必要な情報を表示させることができる。

‘`--help`’ コマンドライン・オプションのリストを表示させ、そのまま実行を終了する。

‘`--help-fonts`’
利用可能なフォントのリストを表示して、そのまま終了する。出力される内容は、‘`-T`’ オプションで指定される出力デバイスによって変わる。`plotfont`

-T X、plotfont -T svg、plotfont -T ai、plotfont -T ps、plotfont -T cgm、plotfont -T fig では PostScript 標準の 35 種類のフォントが使える。plotfont -T svg、plotfont -T ai、plotfont -T pcl、plotfont -T hpgl では 45 種類の PCL 5 標準フォントが使える。plotfont -T pcl と plotfont -T hpgl ではヒューレット・パッカードのベクトル・フォントが使える。上に挙げたすべての出力形式、および plotfont -T png、plotfont -T pnm、plotfont -T gif、plotfont -T regis、plotfont -T tek では 22 種類の Hershey ベクトル・フォントが使える。plotfont の生出力では原理的にこれらのフォントがすべて使える。どの形式にも、plotfont 生出力から plot で変換できるからである。

‘--list-fonts’
‘--help-fonts’ と同じだが、他のプログラムにパイプで渡して処理しやすいように、フォントのリストを 1 カラムで出力する。‘-T’ オプションでの出力デバイスの指定が何もないときは、サポートされているすべてのフォントのリストが表示される。

‘--version’
plotfont および GNU plotting utilities パッケージのバージョン番号を表示して、そのまま終了する。

6.3 plotfont の環境変数

graph や plot の動作を制御するための環境変数は、plotfont の動作の制御にも使われる。重複になるが、以下にそれを挙げる。

BITMAPSIZE、PAGESIZE、BG_COLOR、EMULATE_COLOR、MAX_LINE_LENGTH、ROTATION については上述したとおりである。これらの環境変数はそれぞれ、コマンドライン・オプションの ‘--bitmap-size’、‘--page-size’、‘--bg-color’、‘--emulate-color’、‘--max-line-length’、‘--rotation’ と同じ働きを持つ。他にも、特定の出力デバイスに対してだけ有効な環境変数もある。

plotfont -T X では X Window の画面にウィンドウが一つ開かれ、そこにプロットが描画されるが、このとき DISPLAY 環境変数が設定されていれば、それで示されるディスプレイにウィンドウが開かれる。

PNG または疑似 GIF 形式を生成する plotfont -T png または plotfont -T gif に対しては、二つの環境変数が用意されている。INTERLACE が "yes" のときは、出力される画像はインターレース形式になる。TRANSPARENT_COLOR に色名を指定しておくと、出力される画像のその色の部分が透明に設定され、PNG や GIF を扱う多くのアプリケーションで透明として扱われる。使える色名については Appendix B [Color Names], page 158 を参照のこと。

plotfont -T pnm では Portable Anymap (PBM/PGM/PPM) 形式で出力されるが、環境変数 PNM_PORTABLE が "yes" のとき、出力は PBM、PGM、PPM のポータブル形式 (人間が読めるテキスト形式) で行われる。そうでない場合はバイナリ形式 (これがデフォルト) で行われる。

ウェブ用のベクトル画像の形式である WebCGM プロファイルにしたがった CGM 形式のファイルを plotfont -T cgm で生成する場合に有効な環境変数が二つある。デフォルトでは、生成される CGM ファイルはバージョン 3 の規格に沿っているが、マイクロソフト・オフィスなどの古い CGM 表示ソフトウェアはほとんど、CGM のバージョン 1 しか扱えない。そこで環境変数 CGM_MAX_VERSION を 1"、"2"、"3"、"4" のどれかに設定することで、出力

される CGM のバージョンの最大値を指定できる ("4" がデフォルト)。また CGM_ENCODING で CGM ファイルのエンコーディングを指定できる。これには"clear_text" (人が読めるテキスト形式) と "binary" が指定できる ("binary" がデフォルト)。WebCGM プロファイルではバイナリ形式が要求される。

plotfont -T pcl ではヒューレット・パッカードのプリンタまたはプロッタの出力形式、PCL 5 形式で出力されるが、このとき PCL_ASSIGN_COLORS 環境変数が有効である。PCL 5 出力をカラープリンタなどに出力する際に指定する。これにより色の再現性を正確にするために、内部で出力デバイスに、各色を描画するための「論理ペン」が任意に設定できるようになる。"no" にセットされるとあらかじめ設定されている色のペンだけが用いられ、他の色は濃淡の変化 (シェイディング) によって擬似的に描画される。世の中の PCL 5 デバイスは、カラーより白黒が多いため、デフォルトでは "no" であり、色はシェイディング (グレイスケール) に置き換えられる。

plotfont -T hpgl (ヒューレット・パッカード・グラフィクス・ランゲージ) 形式の出力が生成されるが、これにもいくつかの環境変数が用意されている。もっとも重要なのは HPGL_VERSION で、これには "1"、"1.5"、"2" のいずれかが指定できる ("2" がデフォルト)。"1" は純正の HP-GL、"1.5" は HP7550A グラフィクス・プロッタおよび HP758x、HP7595A、HP7596A ドラフティング・プロッタに出力できる形式 (HP-GL にいくつかの HP-GL/2 拡張を加えたもの)、"2" は新しい HP-GL/2 でそれぞれ描画するよう指示する。もしバージョンが "1" または "1.5" にセットされていると、使えるフォントはベクトル・フォントのみになる。

plotfont -T hpgl のときの用紙上の描画領域は HPGL_ROTATE 環境変数を"yes" に設定すると反時計回りに 90 度回転させることができる。--rotation オプションでは描画領域を回転させ、その左下隅の位置を用紙の左下隅からの相対位置で指定するため、異なった効果になる。HPGL_ROTATE 環境変数に設定できる値は、"no" と "yes" に加えて、"0"、"90"、"180"、および "270" である。"no" と "yes" は "0" と "90" とそれぞれ同じである。"180" と "270" は HPGL_VERSION が "2" のとき (デフォルトがそうである) にだけ有効である。

デフォルトでは、plotfont -T hpgl では使えるペンはあらかじめ決められているものだけである。描画時点でどのペンを使うかは、HPGL_PENS 環境変数で指定される。HPGL_VERSION が "1" のときは HPGL_PENS のデフォルトのペンは"1=black" である。HPGL_VERSION が "1.5" または "2"のときは、HPGL_PENS のデフォルト値は "1=black:2=red:3=green:4=yellow:5=blue:6=magenta:7=cyan" である。指定の仕方は見ての通りである。HPGL_PENS を指定することで 1...31 番のどのペンにも色を割り当てることができる。指定できる色名については Appendix B [Color Names], page 158 を参照のこと。1 番のペンはいつでも使える。その色は黒である必要はない。2...31 番の他のペンはあってもなくてもよい。

HPGL_VERSION が "2" のときは plotfont -T hpgl では HPGL_ASSIGN_COLORS も有効である。もしこれが "yes" にセットされていれば、plotfont -T hpgl で使えるペンの色は HPGL_PENS による制約を受けない。1...31 番の「論理ペン」に必要に応じて色が割り当てられる。これをサポートする HP-GL/2 は LaserJet プリンタや DesignJet だが、あまり多くはないため、デフォルトでは "no" である。特に HP-GL/2 ペン・プロッタは非対応である。

plotfont -T tek では Tektronix 端末のための出力が行われるが、環境変数 TERM が有効である。もし TERM が "xterm"、"nxterms"、"kterm" のいずれかであればそれぞれ、pic2plot がその時実行されている環境は X Window システム上の VT100 端末エミュレータの xterm、nxterm、kterm であると判断される。このとき、pic2plot -T tek では描画が行われる前に、xterm に付属していて通常は隠れている Tektronix のウィンドウを、前面にポップアップさせるエスケープ・シーケンスが送られる。また描画が行われた後には、元の VT100 端末に制

御を戻すためのエスケープ・シーケンスが送られる。その際、Tektronix のウィンドウは画面に残ったままになる。

TERM が "kermit"、"ansi.sys"、"nansi.sys" のいずれかの時は、MS-DOS 版の kermit 上の VT100 エミュレータ上であるとみなされる。このとき、pic2plot -T tek では描画が行われる前に、VT100 エミュレータから Tektronix のウィンドウに切り替えるエスケープ・シーケンスが送られる。また出力には Tektronix 制御コードは、kermit 特有のものが使われる。このとき、使える色数には制限がある (ansi.sys の 16 色はサポートされる)。また描画が行われた後には、元の VT100 端末に制御を戻すためのエスケープ・シーケンスが送られる。キーボードで 'ALT' - '-' をタイプすると、VT100 モードと Tektronix モードの切り替えが手動でできる。

7 spline コマンド

7.1 spline の使い方

spline は一つまたは複数のデータセットに対して、データ点を補間するプログラムである。各データセットは独立変数と従属変数からなり、従属変数は、次元数が固定で、それが明示されていれば、ベクトルでもよい。ここでは独立変数と従属変数はそれぞれ 't' と 'y' で表す。注意: t はスカラー値の変数だが、従属変数 y は一般にベクトルであってよい。

もっとも簡単なデータは、入力はテキスト形式の一つのファイルで、従属変数ベクトル y が 1 次元の場合である。これがデフォルトである。たとえば、入力ファイルの内容が以下のようになっている場合、

```
0.0  0.0
1.0  1.0
2.0  0.0
```

このデータでは、データ点が (t, y) 座標で (0,0)、(1,1)、(2,0) であると解釈される。各データ点は各行に分かれている必要はなく、また t と y が同じ行である必要もない。しかしデータセットが一つであるとする場合には、その中に空行が含まれていてはならない。またデフォルトでは、独立変数の値は単調増加である必要がある。したがって y は t の関数である、と見なすことができる。

データセット中の各点を通るスプライン曲線 (「補間関数」のプロット・データ) は、以下のようにすると得ることができる。

```
spline input_file > output_file
```

このデータの PostScript 形式のプロットを graph で描画するには、以下のようにする。

```
spline input_file | graph -T ps > output.ps
```

X Window の画面にプロットを表示するには、以下のようにする。

```
echo 0 0 1 1 2 0 | spline | graph -T X
```

最後の例では、入力ファイルをなにも指定していない。spline コマンドは、入力ファイルが何も指定されないか、または特殊ファイル名 '-' が指定されたときは、標準入力からデータを読み込む。

spline が行っていることは、正確には、まずデータ点に対する曲線 (補間関数のプロット) のフィッティングである。次に独立変数 t の取る範囲を 100 個の小区間に分割し、y の値が各小区間の端点 (合計 101 点) で取る値を計算する。そしてその値を (t, y) として出力する。これはつまり、与えられたデータセットの点を通る曲線上の 101 個の点の座標である。入力中に (空行で区切られた) 複数のデータセットがある場合は、各データセットが別々に補間される。

'-n' オプションを使えば、上述の 100 を別の正の整数に変更できる。また '-t' オプションで補間する範囲をデフォルト値 (独立変数の取っている範囲) とは異なった値に変更できる。たとえば以下のコマンド

```
echo 0 0 1 1 2 0 | spline -n 20 -t 1.0 1.5 > output_file
```

を実行すると、t の値が (0.0 から 2.0 ではなく) 1.0 から 1.5 までで、等間隔の (101 点ではなく) 21 点のデータを出力する。出力されるデータ点は (0,0)、(1,1)、(2,0) の 3 点を通る曲線上の点である。この例では放物線になる。

一般的に、補間に使われる関数は区分的な三次のスプライン関数である。この関数では、隣接する節点 (knot、与えられるデータセット中にある点) の間で y は t の三次関数であり、y がどの節点の間にあるかで関数の形が変わる。各節点では、節点の両側の関数の勾配と曲率が一致するようになっている。数学的に表現すると、補間関数は節点において二階微分可能であるということである。

spline コマンドでは補間曲線に「テンション」を持たせることができる。'-T' オプションで非零の値のテンションを指定することができる。たとえばある値のテンションを持つスプライン曲線を表示させるには、以下のようにする。

```
echo 0 0 1 0 2 0 | spline -T 10 | graph -T X
```

テンションの値を大きくすると、スプライン曲線はだんだんと折れ線に近づいていく。外見上、強く張られた線のように見える。テンションの値は、正の値と同様に負の値も指定できる。そのときはスプライン曲線は外に膨らむ形になり、正確には正弦曲線的に振動する。しかし負の無限大に近づけていくと、振動していたスプライン曲線はまた折れ線に収束していく。

テンションが正の数の時、その逆数は、スプライン曲線が「よく近似している」と言える独立変数 t の範囲は最大でどこまでか、を表す。テンションを 0 から大きくして行くと、スプライン曲線は短い曲線部分 (データ点がその中心である) と、ほとんど直線の部分で構成されることになる。つまりテンションは「次元」のような量を表す。もしテンションに 0 でない値が指定された場合、まず独立変数のすべて値にはある共通の正の値が乗じられる。テンションの値はその共通の値で除され、元のスプライン曲線をスケーリングしたスプライン曲線が得られる。もしテンションが 0 の場合 (これがデフォルト。つまり三次のプラインの場合) は、データを線形にスケーリングしてもスプライン曲線は変化しない。

数式で表現すると、テンションのあるスプライン関数は、連続する二つの節点の間で以下の微分方程式を満たす。

$$d^4y/dt^4 = \mathrm{sgn}(tension)tension^2\, d^2y/dt^2$$

テンションが 0 のとき (デフォルトの場合) y の t に関する四階微分は各点で 0 になる。この場合 t の関数としての y は連続する二つの節点の間で三次の多項式になる。テンションが 0 でない場合は、y はt の多項式では表されない。指数関数などになるだろう。

テンションがあってもなくても、t に関して周期的なスプライン関数を得るには、'-p' オプションを使えばよい。これを使うときはデータセット中の最初と最後の点で y の値が同じでなければならない。その値が異なる場合は、周期的な補間曲線を計算することはできない。

場合によって、別のプログラムで生成される点を、その生成と同時に補間していけると便利なことがある。つまり、spline コマンドをリアルタイム・フィルタとして使えれば便利である、ということである。spline は通常はフィルタとしては動作しない。可能な限り滑らかな曲線をデータ全体から得ることが一般的な目的だからである。spline をフィルタとして動作させるには '-f' オプションを指定する。この場合、補間をデータ全体からでなく局所的に行うために、異なるアルゴリズム (三次のベッセル補間) が用いられる。'-f' を指定するときは普通、'-p' は指定しない。または '-f' を指定するときは、補間する範囲 (変数 t の範囲) を '-t' オプションで明示的に指定しなければならない。

三次のベッセル補間は原理的に、データ全体から補間を行う三次のスプライン関数ほど滑らかではない。もし '-f' オプションが指定された場合、各節点におけるスプライン関数の勾配は、その節点とその両側の節点の三点から求められた二次曲線の勾配になる。二つの隣り合う補間曲線の勾配は、その両方に共通する節点において一致するが、曲率は一般に一致しない。数学的に表現すると、補間曲線は一階微分可能だが、一般に二階微分可能とは言えない。spline コマンドでは、リアルタイム・フィルタとして動作するためには、微分可能性が犠牲になるということである。

7.2 spline の他の機能

前節で spline がスカラー変数 t の関数 y (関数値もスカラー) をどのように補間するかを解説した。この節では様々な他の補間方法について解説する。多次元の補間、一価の関数ではない一般の曲線のスプライン補間などである。

spline コマンドは次元が指定されていれば y がベクトルでも扱える。次元数は '-d' オプションで指定する。たとえば以下のような多次元のデータセットがあるとする。

```
0.0  0.0  1.0
1.0  1.0  0.0
2.0  0.0  1.0
```

これは 3 点のデータ点 (t, y) の座標がそれぞれ (0,0,1)、 (1,1,0)、(2,0,1) であると解釈される。データセット中の各点を通るスプライン曲線 (補間関数のプロット) は、以下のようにすると得ることができる。

```
spline -d 2 input_file > output_file
```

この例では従属変数 y が二次元ベクトルなので、オプション '-d 2' を指定している。y ベクトルの各要素はそれぞれ独立に補間され、その結果得られる補間関数から計算される値が出力される。

多次元補間を行う場合でも、一次元の場合と同じオプションが指定できる。たとえば '-f' を指定するとリアルタイムで三次のベッセル補間を行う。一次元の場合と同様、'-f' が指定されたときは '-t' オプションで補間区間 (t の値の取りうる範囲) を指定しなければならない。-p オプションではベクトル関数で周期スプラインが得られる。この場合 y の値はデータセットの最初と最後で一致していなければならない。

spline では、平面上に任意に置かれた点を通る曲線を得ることもできる。これは一般の d 次元空間内についても可能である。しかしこれは、少なくとも従来の一般的なスプライン関数と定義とは異なっている。「スプライン補間」とは、(一価の) 関数を組み合わせることであり、関数かそうでないかを問わないような一般の曲線を補間するものではない。一般的に、曲線は必ずしも関数として表現される訳ではない。

以下に「一般の曲線のスプライン補間」の例を示す。

```
echo 0 0 1 0 1 1 0 1 | spline -d 2 -a -s | graph -T X
```

ここでは二次元平面内の 4 点 (0,0)、(1,0)、(1,1)、(0,1) を通る曲線が求められ X Window システムのディスプレイに表示される。'-d 2' オプションで従属変数 y の次元数が 2 であることを指定している。'-a' オプションで、入力中には t の値がない、したがってそれは自動生成することを指示している。デフォルトでは t の値の初期値は 0 で、その次の値は 1、のようになる。'-s' オプションで、t の値は出力しなくてよいことを指示している。

閉曲線も同様にして描くことができる。たとえば以下のようにすると、

```
echo 0 0 1 0 0 1 0 0 | spline -d 2 -a -s -p | graph -T X
```

3 つの点 (0,0)、(1,0)、(0,1) を通る薬用のど飴のような形の曲線が描かれる。データ点の最初と最後が同じ (0,0) であることと、'-p' (つまり '--periodic') オプションを指定することで、閉曲線を描くことができる。

閉曲線でもそうでなくても、一般の曲線をスプライン補間で描く場合、'-a' オプションの代わりに '-A' オプションを使っても、入力には t の値がなくそれを自動生成するよう指定できる。この場合は t の値の増分は 1 ではなく、y の増分と同じなる。各データ点を通る曲線を求めるときにこのようにして t の値を決めるのは、FITPACK という有名なライブラリと同じ方法である。連続するデータ点間の距離が激しく振動するような場合は、このようにするのがおそらく最良の方法である。

二次元平面内のデータ点を通る曲線は、閉曲線でもそうでなくても、自分自身と交差する可能性がある。その場合にテンションを負の値にすると、興味深い画像が得られる。たとえば以下のコマンドを実行すると、

```
echo 0 0 1 0 1 1 0 0 | spline -d 2 -a -s -p -T -14 -n 500 | graph -T X
```

3つの点 (0,0)、(1,0)、(0,1) を渦巻き様になって通る曲線が得られる。'-n 500' は、この曲線は曲がっているところが多いために指定している。これにより合計で 501 点が生成されるように指示しているが、これだけあれば十分に滑らかな線が描ける。

7.3 spline のコマンドライン・オプション

spline コマンドはスカラー変数 t に関してベクトル値を取る関数、または d 次元空間内の曲線を補間する。spline が内部で使っているアルゴリズムは D. Kincaid と [E.] W. Cheney による *Numerical Analysis* (2nd ed., Brooks/Cole, 1996) の section 6.4、および C. de Boor による *A Practical Guide to Splines* (Springer-Verlag, 1978) の Chapter 4 のアルゴリズムと同じである。

入力ファイルの名前はコマンドライン上のどこでも指定できる。その順番やコマンドライン・オプションの順番は、意味を持たない。ファイル名が何も指定されないか、'-' というファイル名が指定されたときは、標準入力から読み込まれる。

一つの入力ファイルに複数のデータセットを記述しておくことができる。'-a' または '-A' オプション (後述) が指定されない限りは、各データセットはデータ点の連続するもので、t と y の値が交互に並べられたものであるとされる。t はスカラー値の独立変数であり、y はベクトルで表される従属変数であり、y の次元数は '-d' オプションで指定される (デフォルトでは 1 である)。

入力ファイルがテキストファイルである時 (これがデフォルト)、複数のデータセットは空行で区切られる。また入力ファイル中には任意の行数のコメント行が書ける。行頭が # という文字の行がコメント行として扱われる。コメント行は無視されるが、空行として扱われる訳ではない。そのため、データセットの途中にあっても構わない。

以下に spline に指定できるコマンドライン・オプションを列挙する。コマンドライン・オプションには、3種類ある。

1. 各データセットに適用される補間法を指定するオプション。
2. 入出力の形式を指定するオプション。
3. 情報を表示させるオプション (たとえば '--help')。

引数を取るオプションでは、説明の最初にカッコ書きでその引数の型とデフォルト値を示している。

まず以下に、各データセットに適用される補間法を指定するオプションを列挙する。

'-f'
'--filter'

局所的補間アルゴリズム (三次のベッセル補間アルゴリズム) を使い、spline がリアルタイム・フィルタとして動作できるようにする。補間曲線の傾きは、データセット中の各データ点において、その点と両隣の点を通る二次曲線から決められる。'-t' オプションは通常は省略してもよいが、'-f' オプションが指定されたときは必須である。また '-f' オプションが指定されるときは、'-k'、'-p'、'-T' の各オプションを使うことはないだろう。

'-f' が指定されないときは、別の (データセット全体を一度に使う) アルゴリズムが使われる。

‘-k *k*’
‘--boundary-condition *k*’
(実数、デフォルトは 1.0) 作り出されたそれぞれのスプライン曲線に対して、境界条件のパラメータを k にする。各スプライン曲線は、境界条件として $y''[0] = ky''[1]$ および $y''[n] = ky''[n-1]$ を満たすが、ここで $y[0]$ および $y[1]$ は与えられるデータセットの最初の 2 点での従属変数ベクトル y の特定の要素の値であり、$y[n-1]$ と $y[n]$ は最後の 2 点での、その要素の値である。k を 0 にすると与えられるデータセットの両端で曲率が 0 になり、自然スプラインとなる。‘-k’ オプションは ‘-f’ または ‘-p’ オプションが指定されたときには使われないだろう。

‘-n *n*’
‘--number-of-intervals *n*’
(正の整数、デフォルトは 100) 補間を行う区間を n 個の小区間に分割する。計算されるデータ点の数と出力される点の数は $n+1$ になる。

‘-p’
‘--periodic’
周期的スプライン補間を行う。このオプションを指定するときは、各データセットにおいて最初と最後のデータ点が同じでなければならない。‘-p’ オプションを指定するときは、‘-f’ または ‘-k’ オプションは使われないだろう。

‘-T *tension*’
‘--tension *tension*’
(実数、デフォルトは 0.0) 各補間曲線のテンションの強さを *tension* に設定する。データセット中の二つの連続するデータ点の間で、ベクトルの各要素について、以下の微分方程式を満たすスプライン関数が計算される。

$$d^4y/dt^4 = \mathrm{sgn}(tension)tension^2\, d^2y/dt^2$$

tension が 0 のときはスプライン関数は区分的な三次多項式になる。*tension* の値を正の方向に大きくすると、スプライン関数は折れ線に収束する。‘-T’ を ‘-f’ オプションと同時に使うことはないだろう。

‘-t *tmin tmax* [*tspacing*]’
‘--t-limits *tmin tmax* [*tspacing*]’
各データセットについて、補間する区間を *tmin* から *tmax* にする。*tspacing* が指定されていなければ、補間区間全体は ‘-n’ オプションで指定される個数の小区間に分割される。もし ‘-t’ オプションが指定されないときは、入力されるデータセット中で独立変数が取っている範囲が補間対象となる。‘-f’ オプションを使って spline コマンドをフィルタとして使うときには、‘-t’ オプションを指定する必要がある (上述)。

以下のオプションは、入出力ファイルの形式を指定する。

‘-d *dimension*’
‘--y-dimension *dimension*’
(整数、デフォルトは 1) 入力および出力における従属変数 y の次元を *dimension* に指定する。

‘-I *data-format*’
‘--input-format *data-format*’
(文字、デフォルトは ‘a’) 入力ファイル中のデータの形式を、以下の中から *data-format* に指定する文字で指定する。指定できる文字 (と形式) は以下の通りである。

‘a’ テキスト形式。ファイル中では浮動小数点実数が並んでおり、連続するデータ点の t と y が交互に並んでいるものと解釈される。y が d ベクトルのとき、各データ点は $d+1$ 個の数値で表される。各点の t と y は必ずしも同じ行に書かれている必要はなく、また各点を違う行に書く必要もない。しかし空行 (2 個連続する改行文字) はデータセットの区切りであると見なされ、次の行から次のデータセットが始まるものと解釈される。

‘f’ 単精度実数バイナリ形式。ファイル中では浮動小数点実数が並んでおり、連続するデータ点の t と y が交互に並んでいるものと解釈される。y が d 次元ベクトルのとき、各データ点は $d+1$ 個の数値で表される。データセットの区切りは一つの FLT_MAX の値の数値で表される。これは単精度実数で表現できる最大値である。その値はほとんどのプラットフォームでは 3.4×10^{38} である。

‘d’ 倍精度実数バイナリ形式。ファイル中では浮動小数点実数が並んでおり、連続するデータ点の t と y が交互に並んでいるものと解釈される。y が d 次元ベクトルのとき、各データ点は $d+1$ 個の数値で表される。データセットの区切りは一つの DBL_MAX の値の数値で表される。これは倍精度実数で表現できる最大値である。その値はほとんどのプラットフォームでは 1.8×10^{308} である。

‘i’ 整数バイナリ形式。ファイル中では整数が並んでおり、連続するデータ点の t と y が交互に並んでいるものと解釈される。y が d 次元ベクトルのとき、各データ点は $d+1$ 個の数値で表される。データセットの区切りは一つの INT_MAX の値の数値で表される。これは整数で表現できる最大値である。その値はほとんどのプラットフォームでは $2^{31}-1$ である。

‘-a [*step_size* [*lower_limit*]]’
‘--auto-abscissa [*step_size* [*lower_limit*]]’
(実数、デフォルトは 1.0 と 0.0) 独立変数 (t) の値を自動で決定する。データの形式 (‘a’、‘f’、‘d’、または ‘i’) とは関係なく、このオプションを指定すると、入力ファイル中には独立変数 (t) の値はないものとして扱われ、入力されるデータはすべて従属変数 (y) である、と解釈される。したがって y の次元が d のとき、各データ点には d 個の数値があればよい。t の各点での増分は *step_size* で指定され、最初のデータ点の t の値は *lower_limit* で指定される。

‘-A’
‘--auto-dist-abscissa’
独立変数 (t) の値を自動で決定する。‘-a’ と似ているが、最初のデータ点の t の値を 0.0 にし、それ以降の増分を y の増分と同じにする。関数とは言えない一般の曲線を補間するときに使う (Section 7.2 [Advanced Use of spline], page 61 参照)。

‘-O *data-format*’
‘--output-format *data-format*’
(文字、デフォルトは ‘a’) 出力するデータの形式を *data-format* に指定する。指定できるものとその意味は ‘-I’ オプションと同じである。

‘-P *significant-digits*’
‘--precision *significant-digits*’
(正の整数、デフォルトは 6) t と y を出力するときの精度 (桁数) を *significant-digits* にする。出力形式が ‘a’、テキスト形式の場合にのみ有効である。

‘-s’
‘--suppress-abscissa’
独立変数 t の値を出力せず、各点において従属変数 y の値のみを出力する。y が d 次元のとき、各点で $d+1$ 個ではなく、d 個の数値が出力される。関数とは言えない一般の曲線を補間するときに使う (Section 7.2 [Advanced Use of spline], page 61 参照)。

以下のオプションは、情報を表示させるためのものである。

‘--help’ コマンドライン・オプションのリストを表示、そのまま終了する。

‘--version’
spline とこの GNU plotting utilities パッケージのバージョン番号を表示して、そのまま終了する。

8 ode コマンド

GNU ode コマンドは、一階常備分方程式 (ODE) の初期値問題の数値解を計算するプログラムである。n' 次の微分方程式を n 連立の一階微分方程式に単純に変換すれば ode で高次の微分方程式を解くこともできる。ode の出力はパイプで graph に渡して、一つあるいは複数の計算された解曲線をプロットすることができる。

求解アルゴリズムは複数実装されている。Runge–Kutta–Fehlberg 法 (デフォルト) と Adams–Moulton 法 と Euler 法である。Runge–Kutta–Fehlberg 法 と Adams–Moulton 法では、ステップ幅の適応制御が行われる。

8.1 数学的基礎

まず、標準的な定義から解説する。微分方程式は、形が明示的に示されない関数とその導関数を含む方程式である。微分方程式は、未知関数の独立変数が一つだけのとき常微分方程式と呼ばれる。ここでは独立変数を t で表す。微分方程式の次数とは、方程式に含まれるもっとも高階の導関数の次数である。複数の方程式が連立しているときは、微分方程式系と呼ばれる。ある方程式が他の方程式に従属している場合は、対をなしていると言われる。解とは、方程式を満たす関数のことである。初期値問題とは、独立変数がある値のときのすべての未知関数とその導関数の値がわかっている、という条件が与えられることである。このような「初期条件」が与えられれば、解が一意に定まる。解の存在性と一意性についての、定義や解説は多くの入門書にある (Birkhoff と Rota による *Ordinary Differential Equations* の Chapter 1 など。ode に関係のある他の文献については Section 8.9 [ODE Bibliography], page 83 を参照のこと)。

現実の問題では、微分方程式の解は初等関数によっては表されないことが多い。したがって、数値解を求めることになる。

初期値問題の数値解法は、関数の評価と算術演算だけを使った近似解法である。ode は、以下の形式の一階の初期値問題を解くことができる。

$$\begin{aligned} dx/dt &= f(t,x,y,\ldots,z) \\ dy/dt &= g(t,x,y,\ldots,z) \\ dz/dt &= h(t,x,y,\ldots,z) \end{aligned}$$

独立変数 t の初期値における各従属変数の初期値は、以下のようにして与えられる。

$$\begin{aligned} x(a) &= b \\ y(a) &= c \\ &\vdots \\ z(a) &= d \\ t &= a \end{aligned}$$

ここで $a, b, c, \ldots, d$ は定数である。

ode で数値解が得られるようにするには、関数 $f, g, \ldots, h$ が一般的な演算子 (加算、減算、乗算、除算、べき乗) と ode が理解できる基本的な関数を使って表現されている必要があるが、ode で使える関数はプロット・プログラムの gnuplot が理解するものと同じである。また ode は一階導関数が陰関数として表現されているような系は扱えないので、関数 $f, g, \ldots, h$ は陽関数として表現されていなければならない。

実装されている数値解法はどれも、独立変数 t の離散的な点上における計算を行う。離散点の間隔はステップ幅 (連続する二つの t の値の距離) と呼ばれ、通常は h で表されるが、そ

の値は固定、あるいは適応的に決定される。一般的に、ステップ幅を小さくすると解の精度が向上する。ode ではステップ幅を明示的に与えて計算させることもできる。

8.2 ode の使用例

以下の例で ode でどのようにして初期値問題を解くかを示す。もしもう少し複雑なことがやりたい場合は、Section 8.8 [Input Language], page 79 を参照するとよい。その節に ode 言語の仕様が述べられている。また GNU plotting utilities の配布パッケージに ode の入力サンプルを集めたディレクトリがある。ほとんどのシステムではそれは、'/usr/share/ode' か '/usr/local/share/ode' にインストールされている。

最初の例は、以下の単純なものである。

$$y'(t) = y(t)$$

初期条件は以下である。

$$y(0) = 1$$

この微分方程式の解析解は、以下である。

$$y(t) = e^t$$

数値計算の精度が 7 桁の場合は、$t = 1$ で以下のようになる。

$$y(1) = e^1 = 2.718282$$

コマンドラインで以下のように入力、実行すると、ode が計算する数値解が得られる。

```
ode
y' = y
y = 1
print t, y
step 0, 1
```

数値が 2 列で出力されるはずである。各行には独立変数 t と変数 y の値が表示される。t は 0 から 1 まで、ある幅で刻みながら進んでいるはずである。出力の最後の行は

```
1 2.718282
```

となるだろう。計算精度は '-p' オプションで変更できる。たとえばもし、単に 'ode' とするのではなく 'ode -p 10' とすると、7 桁 (デフォルトの精度) ではなく 10 桁で数値が出力される。

ここまで出力した後、ode は次の入力を待つモードになる。そこで以下のように入力すると、

```
step 1, 0
```

また t と y の値からなる 2 列の数値が出力される。ただし t の値が 1 から 0 に 進んでいるはずである。また、

```
step 1, 2
```

と入力すると、t が減少するのではなく、増加する。ode の実行を終了するには、'.'、つまり一つのピリオドとリターンを入力すればよい。もしくは入力ストリームに EOF (end-of-file) があったときも ode は終了する。これは端末上で control-D を入力すればよい。

上の例では、入力の各行の意味は見て分かる通りである。step 文は独立変数 (ここでは t) の取る範囲の始点と終点を設定し、ode はその範囲で数値解を計算する。初期値は最初の step 文 (つまり 0) と以下のような代入文

```
y = 1
```

が示すことになる。この場合、$y(0) = 1$ という初期条件を示したことになる。'y' = y' と 'y = 1' という二つの文は、全く違った意味を持つ。'y' = y' は y の導関数の計算法を与え、'y = 1' は y の初期値を与える。'step' 文が入力に現れたところで、ode は与えられた範囲内で独立変数のステップを進めて求解を行い始める。ステップごとにどの変数を表示するかは、もっとも後に入力された 'print' 文による。たとえば、

```
print t, y, y'
```

とすると独立変数 t の値と、変数 y とその導関数の値を、ステップごとに表示する。

ode は入力全体あるいは入力の最初の部分をファイルから受け取ることができる。その例として、まず以下の内容のファイルがあるとする。

```
# an ode to Euler
y  = 1
y' = y
print t, y, y'
```

ファイル名を 'euler' とする ('#' の行はコメント行で、入力のどこにあってもよい。'#' のある場所から行末までは、無視される)。このファイルを ode に読み込ませるには、以下のようにする。

```
ode -f euler
step 0, 1
```

最初の行で ode はファイル 'euler' を読み込み、次の行でステップを開始する。この例では、0 から 1 までの t の各値について、3 つの数値 (t、y、y') が表示される。ステップが最後まで進むと、ode は端末から次の文が入力されるのを待つ。これは次の例で示す。このコマンド

```
ode -f euler
```

と、次のコマンド

```
ode < euler
```

では、ode の動作は異なる。後者では入力はすべてファイル 'euler' から行われることになり、前者ではファイルを読み込んだ後、端末からの入力を待つ。後者の場合は、ファイルの最後に 'step' 文がないと結果が出力されないが、'.' をファイルに書いておく必要はない。'.' は端末からの入力を終了するときにだけ用いられる。

次に、二階の微分方程式の数値解を求める例を示す。以下の初期値問題を考える。

$$\begin{aligned} y''(t) &= -y(t) \\ y(0) &= 0 \\ y'(0) &= 1 \end{aligned}$$

この解析解は以下である。

$$y(t) = \sin(t)$$

ode でこの問題を解くには、二階の微分方程式を二つの一階の微分方程式に直す必要がある。そのためには独立変数 t の別の関数 (ここでは yp とする) を新たに導入し、以下のようにする。

$$\begin{aligned} y' &= yp \\ yp' &= -y \end{aligned}$$

この微分方程式系は、一つの微分方程式だった問題と等価である。このようにして n 次の方程式を一階の問題に書き直すのは、よく行われるテクニックである。

変数 t の関数として y をプロットするには、以下のような内容のファイルを作るとよい。

```
# sine : y''(t) = -y(t), y(0) = 0, y'(0) = 1
sine' = cosine
cosine' = -sine
sine = 0
cosine = 1
print t, sine
```

(*y* と *yp* をそれぞれ *sine* と *cosine* に書き直している。解析解がそうなるからである。) このファイルを 'sine' とする。生成されたデータを X Window システムのディスプレイで表示するには、以下のようにする。

```
ode -f sine | graph -T X -x 0 10 -y -1 1
step 0, 2*PI
.
```

上の例の ode の行を入力すると、graph -T X によりウィンドウが開かれる。その次の行の 'step' 文を入力すると、そのウィンドウ内にデータが描画される。'-x 0 10' および '-y -1 1' オプションは両座標軸の範囲を指定する。リアルタイム・プロットを行うときにはプロットされる点が後から入力されるため、このオプションが必要になる。もし座標軸の描画範囲がコマンドラインで指定されなかったときは、graph -T X はすべてのデータが入力され終わるのを待ってからプロット範囲を決定して、描画を行う。

上の例を少し変更して、ode にどのようにして複数のデータセットを生成させて graph にプロットさせるかを示す。端末上で、以下の入力をするとする。

```
ode -f sine | graph -T X -C -x 0 10 -y -1 1
step 0, PI
step PI, 2*PI
step 2*PI, 3*PI
.
```

すると、正弦曲線が 3 回に分けて描かれる。'step' 文による出力の最後には空行があるため、graph -T X はそれぞれの 'step' 文による出力を別のデータセットであると見なす。カラーディスプレイ上では、3 つの各部分が異なった色で表示されるだろう。これは graph の、データセットが読み込まれるごとに線種を変更していく機能によるものである。この機能を無効にしたいときは、graph -T X の代わりに graph -T X -B とするとよい。

上の例の graph -T X の代わりに graph の他の出力デバイスを使ってもよい。たとえば以下のように graph -T ps とすると EPS (Encapsulated PostScript) 形式の出力が得られる。

```
ode -f sine | graph -T ps > plot.ps
step 0, 2*PI
.
```

graph の出力デバイスのうち、graph -T png、graph -T pnm、graph -T gif、graph -T svg、graph -T ai、graph -T ps、graph -T cgm、graph -T fig、graph -T pcl、graph -T hpgl では、たとえ '-x' および '-y' オプションを指定しても、リアルタイム・プロットはできない。したがってこれらの出力デバイスのどれかを使うときは、ode コマンドに '.' を入力してすべての入力が終わってからしか、プロットは生成されない。

上の例では従属変数の導関数は比較的簡単な形であった。しかし任意の、もっと複雑に従属変数と独立変数が組合わさった形でもよい。それには ode に組み込まれている関数が使える。abs、sqrt、exp、log、log10、sin、cos、tan、asin、acos、atan、sinh、cosh、tanh、asinh、acosh、and atanh などが組み込まれている。ほかにもあまり多くは使われないが、besj0、besj1、besy0、besy1、erf、erfc、inverf、lgamma、gamma、norm、invnorm、

ibeta、igamma も組み込まれている。これらはプロット・プログラムの gnuplot の同名の関数と同じ定義である (引数を 3 つ取る ibeta、2 つ取る igamma 以外はすべて引数は一つである)。また ode は上で示したように、'PI' の値も正しく理解する。上に列挙した 'cos' や 'sin' などの関数の名前は予約語であり、したがって変数名としては使えない。

予約名とキーワードによる制限以外は、変数名は任意に付けることができる。英字で始まる英数字からなる文字列が名前として使える。最初の 32 文字だけが名前として有効である。非常に重要なことは、ode は微分方程式系の定義から、微分方程式や初期値の代入文の左辺に現れない唯一の変数が独立変数であると見なす、ということである。そういう変数が複数あると、ステップを進められず、エラーメッセージが表示される。またそういった変数がない場合は、ode の内部で代わりに '(indep)' という名前のダミーの独立変数が使われる。

8.3 一歩進んだ ode の使い方

すべての機能を網羅できるわけではないが、以下に ode の他の機能の使用例を示す。コマンドライン・オプションのリストについては Section 8.4 [ode Invocation], page 72 を参照のこと。

ode を使って、非常に美しいプロットをさほど大した労力もなく得ることができる。以下にストレンジ・アトラクタ、またはローレンツ・アトラクタをプロットする例を示す。ファイル 'lorenz' に以下の内容が書き込んであるとする。

```
# The Lorenz model, a system of three coupled ODE's with parameter r.
x' = -3*(x-y)
y' = -x*z+r*x-y
z' = x*y-z

r = 26
x = 0; y = 1; z = 0

print x, y
step 0, 200
```

そこで以下のコマンドを実行すると、

```
ode < lorenz | graph -T X -C -x -10 10 -y -10 10
```

ローレンツ・アトラクタのプロットが表示される (厳密に言うと、このプロットはそれを二次元に投影したものである)。以下のようにすれば、このプロットを PostScript に変換して、プリンタで印刷できる。

```
ode < lorenz | graph -T ps -x -10 10 -y -10 10 -W 0 | lpr
```

'-W 0' (太さを 0 にする) オプションで graph -T ps にもっとも細い線で描画するように指示している。プリンタなどの PostScript デバイスでは、こうするときれいに表示できる。

ローレンツ・アトラクタの例のセミコロンの部分を見ると、リアルタイムで生成されるデータから見た目にインパクトのあるプロットをする一方で、複数の入力文をどのようにすれば一行で入力できるかがわかる。この例ではまた、定数を記号で表す方法も示されている。ode の入力としては、パラメータ r は x、y、および z といった他の変数と同じである。しかしそれらと違って、ステップが進んでも r の値は一定のままである。それは導関数 r' の式が与えられていないからである。

次の例では対話的に「位相平面 (phase portrait)」を描く方法を示す。それは、異なる初期値から得られる解曲線の集合である。位相平面は、微分方程式の性質を見る上でもっともよい方法であり、外見的にも美しい。

ode に与える入力には ‘step’ 文はいくつあってもよいため、ode を一度実行するだけで複数の解曲線を得られる。各 ‘step’ 文の前、最後に読み込まれた ‘print’ 文による指定がその ‘step’ による出力で使われる。位相平面を描くとき実際には、どの解曲線を描くべきかを丁寧に、慎重に選ぶ必要がある。そのためにはリアルタイム・プロットを使って重要そうな解をすべてプロットし、試行錯誤を行って決める必要があるだろう。

例として、被食者と捕食者の関係のモデルに使われるロトカ・ヴォルテラ・モデルを使う。このモデルでは湖に A (被食者) および B (捕食者) という二種類の魚がいるとする。被食者は豊富に生えている植物を食べて生きていて、捕食者は被食者を食べている。$x(t)$ が被食者の個体数、$y(t)$ が捕食者の個体数で、ともに時間 t の関数とする。A と B の単純な関係のモデルは、以下の式で与えられる。

$$\begin{aligned} x' &= x(a - by) \\ y' &= y(cx - d) \end{aligned}$$

a, b, c, d は正の定数である。この微分方程式系の位相平面を描くには、以下のようにすればよい。

```
ode | graph -T X -C -x 0 5 -y 0 5
x' = (a - b*y) * x
y' = (c*x - d) * y
a = 1; b = 1; c = 1; d = 1;
print x, y
x = 1; y = 2
step 0, 10
x = 1; y = 3
step 0, 10
x = 1; y = 4
step 0, 10
x = 1; y = 5
step 0, 10
.
```

各 ‘step’ 文ごとに１つの曲線が、合計４本の曲線が連続して描かれる。これらは周期的である。その周期性は、被食者と捕食者の個体数が実際の自然環境下で増減する周期と似ている。カラーディスプレイ上では各曲線は異なる色で描かれる。これは、graph がデフォルトではデータセットごとに違う線種を使うからである。この機能を無効にしたいときは、graph -T X ではなく graph -T X -B とすればよい。

ode と graph で離散点をプロットする時、それらを線で結ばない方がよいような場合もある。３つ目の例として、そのやり方を示す。ファイル ‘atwoods’ に以下の内容が書き込んであるとする。

```
m = 1
M = 1.0625
a = 0.5; adot = 0
l = 10; ldot = 0

ldot' = ( m * l * adot * adot - M * 9.8 + m * 9.8 * cos(a) ) / (m + M)
l'    = ldot
adot' = (-1/l) * (9.8 * sin(a) +  2 * adot * ldot)
a'    = adot
```

```
print l, ldot
step 0, 400
```

最初の数行は揺れるアトウッドの器械と呼ばれる、一種の振り子を表現する関数の記述である。通常のアトウッドの器械は、滑車に掛けられた、たるみのない糸の両端に重りがそれぞれぶら下がっている、というものである。通常、重い方の重りの質量が M、軽い方が m と表され、軽い方が上に引っ張られていく。揺れるアトウッドの器械は、軽い方の重りが、上下方向に動いていくときに前後にも揺れる、としたものである。

‘print l, ldot’ 文は、軽い方の重りの上下方向の位置と速度を各ステップごとに表示させる。以下のコマンドを実行すると、

```
ode < atwoods | graph -T X -x 9 11 -y -1 1 -m 0 -S 1 -X l -Y ldot
```

リアルタイム・プロットが得られる。‘-m 0’ オプションで連続するデータ点が線で結ばれないようにしている。‘-S 1’ オプションで記号 1 (点) で各データ点をプロットするように指示している。このコマンドを実行すると分かるように、重い方の重りが軽い方に「勝つ」わけではく、非周期的に振動を続ける。周期的ではないため、プロットは線で結ばないほうが見やすい。

最後に、何かおかしなことが起こっていて様子を見たい、というようなときに便利な、いくつかの ode の機能を示す。その一つは ‘examine’ 文である。これは微分方程式系の変数について、その情報を直接得るのに役に立つだろう。詳細は Section 8.8 [Input Language], page 79 を参照のこと。

また ‘print’ を使って、単なる変数の値だけでなくもっと詳しい情報を表示させることもできる。既に例に出ているが、変数名に ‘’’ を付けるとその変数の導関数値が表示される。同様に変数名に ‘?’、‘!’、‘~’ を付けるとそれぞれ 1 ステップでの相対誤差、1 ステップでの絶対誤差、累積誤差が表示される (累積誤差は現在、まだ実装されていない) 。これらについては Section 8.6 [Numerical Error], page 75 参照のこと。

前出の例で示したように、‘print’ 文ではもっと複雑な表示もできる。その一般的な書式は以下である。

```
print <pr-list> [every <const>] [from <const>]
```

‘[...]’ のカッコに囲まれた部分は、必ずしも指定しなくてもよい。ここまでの例では ‘every’ 節と ‘from’ 節のことは出てきていない。<pr-list> はよく使うが、これはコンマで区切られた変数のリストである。たとえば以下の文では

```
print t, y, y' every 5 from 1
```

<pr-list> は <t, y, y’> である。‘every 5’ と ‘from 1’ いう節は、変数値の表示を 5 ステップおきに、また独立変数 t の値が 1 になってから 行うことを指示している。‘every’ 節は ‘step’ 文による出力をすっきりさせたいときに、‘from’ 節は解曲線の最後の方だけが必要なときに、それぞれ便利である。

8.4 ode のコマンドライン・オプション

以下に ode に指定できるコマンドライン・オプションを列挙する。コマンドライン・オプションには、3 種類ある。

1. 入力法に関するオプション。
2. 出力形式に関するオプション。
3. 求解アルゴリズムと、そのアルゴリズムでの許容誤差の指定方法に関するオプション。

4. 情報を表示させるオプション。

以下のオプションで、入力の方法を指定する。

‘-f *filename*’
‘--input-file *filename*’
標準入力から読み込む前に、*filename* から読み込む。

以下のオプションで、出力形式に関する指定を行う。

‘-p *significant-digits*’
‘--precision *significant-digits*’
(正の整数、デフォルトは 6) 数値解を表示するときの精度を *significant-digits* で指定する。このオプションが指定されているときは、結果は指数表記される。

‘-t’
‘--title’ 出力の最初にタイトル行として、各カラムの意味を表示する。このオプションが指定されているときは、結果は指数表記される。

以下のオプションは、求解アルゴリズムに関するものである。３つの基本オプション ‘-R’、‘-A’、‘-E’ のうちの一つだけを指定できる。デフォルトは ‘-R’ (Runge–Kutta–Fehlberg) である。

‘-R [*stepsize*]’
‘--runge-kutta [*stepsize*]’
５次の Runge–Kutta–Fehlberg 法を使う。固定のステップ幅 *stepsize* を指定しない限り、ステップ幅は適応的に決定される。固定のステップ幅を指定し、許容誤差の指定も行わない場合、古典的な４次の Runge–Kutta 法が使われる。

‘-A [*stepsize*]’
‘--adams-moulton [*stepsize*]’
Adams–Moulton の４次の予測子–修正子法を使う。固定のステップ幅 *stepsize* を指定しない限り、ステップ幅は適応的に決定される。通過した点が「悪い」と判定されたとき (があれば)、Runge–Kutta–Fehlberg 法を使う。

‘-E [*stepsize*]’
‘--euler [*stepsize*]’
高速だが粗いオイラー法を使う。ステップ幅は固定である。*stepsize* のデフォルト値は 0.1 である。精度を必要とする場合にはお勧めできない。

許容誤差を指定するオプション ‘-r’ と ‘-e’ (後述) は、‘-E’ が指定されているときには使ってもあまり意味はないだろう。

‘-h *hmin* [*hmax*]’
‘--step-size-bound *hmin* [*hmax*]’
ステップ幅の下限を *hmin* にする。どの求解アルゴリズムを使っていても、ステップ幅が *hmin* より小さくなることはなくなる。デフォルトでは、ステップ幅はそのプラットフォームの精度限界、つまり倍精度浮動小数点実数の、非零であるようなもっとも小さな値まで小さくなれるようになっている。オプションに *hmax* も指定された場合は、ステップ幅の上限も指定される。これにより、詳しく見たい領域を求解アルゴリズムが粗く飛ばしていってしまうことを避けられる。

以下のオプションは、許容誤差の指定を行うことを指示するものである。

‘-r *rmax* [*rmin*]’
‘--relative-error-bound *rmax* [*rmin*]’
‘-e *emax* [*emin*]’
‘--absolute-error-bound *emax* [*emin*]’
‘-r’ オプションは 1 ステップでの最大許容相対誤差を指定する。‘-r’ オプションが指定されているときは、どの従属変数でもその相対誤差が 1 ステップで *rmax* (デフォルトでは 10^{-9}) を超えることがないようにされる。誤差がそれより大きくなったときには、その解は捨てられ、エラーメッセージが表示される。ステップ幅が固定でないときは「適応的に」ステップ幅が縮められる。それにより、相対誤差が指定値よりも小さくすることができる。したがって、許容誤差を小さく指定すると、それだけステップ幅も小さくなる。下限値 *rmin* も指定できる。これにより、ステップ幅をある程度の大きさにしておくことができる (デフォルトでは *rmin* は *rmax*/1000 である)。‘-e’ オプションは ‘-r’ とほとんど同じだが、相対誤差ではなく絶対誤差を指定する。

‘-s’
‘--suppress-error-bound’
ステップ誤差 (ステップを一つ進めたときに、そのステップで生じたと推定される計算誤差) の許容限度の指定をなくす。これにより、ステップ誤差が許容範囲を超えてしまうような場合でも、ode は求解を続けることができる。これを指定すると、数値計算の誤差が大きくなることがある。

最後に、情報を表示するためのオプションを挙げる。

‘--help’ コマンドライン・オプションのリストを表示、そのまま終了する。

‘--version’
パッケージのバージョン番号を表示して、そのまま終了する。

8.5 診断メッセージ

ode は実行中、常に以下の二種類の状態 (モード) のどちらかにある。

- 入力を読み込む。入力されるのは、微分方程式系の定義、数値的に解くアルゴリズムの指定、‘print’ 文、‘step’ 文などである。
- 微分方程式系を数値的に解き、結果を出力する。

ode は、‘step’ 文が入力されたときに最初の状態から二つ目の状態に遷移する。そして出力が終わった時点で最初の状態に戻る。「読み込み」および「求解」のどちらの状態でも、エラーが発生したら計算を中断するか、ode の実行を終了することがある。以下に生じうるエラーについて説明する。

ode が入力を読み込んでいるときには、構文エラーが発生し得る。これは入力に文法の間違いがあって、ode が構文解析を行えないことを示す (構文の詳細については Section 8.8 [Input Language], page 79 を参照のこと)。この場合、以下のエラーメッセージが表示される。

```
ode::nnn: syntax error
```

‘nnn’ は文法の間違いがあるとされる行の番号である。入力がキーボードからではなく、‘-f filename’ で入力ファイルを指定しているときは、そのファイル内にあるエラーに対して、以下のようにメッセージが表示される。

```
ode:filename:nnn: syntax error
```

ファイルからの読み込みが終わって標準入力からの読み込みに切り替わると、表示される行番号はまた 1 からはじまる。

入力に文法の間違いがあった場合、それを修正して動作を回復するような動作は一切行わない。しかしできるだけ無駄にモードが切り替わらないようにはするので、入力ファイル中に複数の間違いがあっても、それらはできる範囲で一度に表示される。

また数値計算上、致命的な演算例外 (零除算や浮動小数点のオーバーフローなど) が ode での入力中に発生することもある。そういったときは ode は "Floating point exception" エラーメッセージを表示し、終了する。演算例外はプラットフォーム依存である。いくつかのプラットフォームでは、以下の行

```
y = 1/0
```

は演算例外を発生する。またいくつかのプラットフォームでは (上のと同じプラットフォームとは限らない)、以下の行

```
y = 1e100
z = y^4
```

も演算例外を発生する。多くのプラットフォームでは、ode が内部で使う倍精度実数の大きさが約 1.8×10^{308} に制限されているためである。

ode が「求解」状態にあるとき、たとえば数値解を計算しているとき、やはり同様の演算例外が発生しうる。その場合求解は中断され、以下のようなメッセージが表示される。

```
ode: arithmetic exception while calculating y'
```

しかし ode の実行自体は終了しない。例外は「キャッチ」される。ode 自身はいくつかの例外を認識し、キャッチする。負の数の平方根、非正数の対数、非正数の負の数乗、である。ode はこれらの演算が実行されてしまう前にキャッチし、生じた不正な演算はどれかを示すエラーメッセージを表示する。

```
ode: square root of a negative number while calculating y'
```

上がその代表例 (負の数の平方根) である。

ode を実行しているプラットフォームで 'matherr' 機能が有効で、標準の数学関数の計算で生じたエラーを表示できるような場合、これが使われる。この機能を使うと、'log' や 'gamma' といった関数の評価時に、それが実際に評価される前にドメイン・エラー (引数の値が許されない値になっている) や範囲エラー (オーバーフロー、アンダーフロー、精度落ち) が報告される。'matherr' 機能が有効なら、エラーメッセージではもっと多くの情報が表示される。たとえば

```
ode: range error (overflow) in lgamma while calculating y'
```

というエラーメッセージでは、与えられた引数で対数ガンマ関数 'lgamma' を評価したら、その値が大きくなりすぎたであろうことを示している。アンダーフローの警告以外のエラーメッセージが表示されたときは、求解を中断する。

他の種類のエラーも、求解中には発生しうる。'-r' か '-e' オプションが指定されているとき、切り上げられた誤差が許容範囲を超えてしまうような場合である。これが発生した数値解は捨てられ、ode のステップはエラーが発生する前の点に戻る。

8.6 数値誤差

以下の議論は、何らかの結論や答えを断定的に導くものではなく、問題を理解することを目的としている。議論の各主題 (浮動小数点エラーなど) について、それぞれ参考になる書籍があるが、ここでは中庸を目指すことにする。最初に、ode で実装されている誤差解析の基本

的な考え方について、次に、よくある落とし穴を避ける方法について述べる。以下の議論を読む前に、数値解析の本 (たとえば Atkinson の *Introduction to Numerical Analysis* など) を読んでおくとよい。

最初にいくつかの定義をしておく。もっとも問題になる誤差は、真の解と近似的な数値解法による値との差異である。これは累積誤差と呼ばれる。求解の各ステップで積み重ねられていくからである。困ったことに、普通は真の解が分からないので、累積誤差を推定することもできない。しかし、なんとか誤差の見当をつける方法はいくつかある。特に、求解を 1 ステップ進めたときに生じる真の解と近似数値解の誤差、つまりステップ誤差は、最初のステップにおいては正しく求められると仮定できる。

相対ステップ誤差は、ステップ誤差をその点の近似数値解で除した値である。真の解の値は正確には分からないので、除算には近似解をつかうしかない。ステップ幅が適応的に決められるアルゴリズムを使う時、ode は相対ステップ誤差に基づいてステップ幅を決定する。デフォルトでは、各ステップで 8 桁の精度を維持できるようにステップ幅を選ぶ。これはつまり、相対ステップ誤差が許容誤差 10^{-9} を超えない程度にはステップ幅を広く取る、ということである。これは '-r' オプションで調整することができる。

数値誤差の生じる原因には、2 種類ある。一つ目は、計算機での数値演算では精度が有限である、ということである。すべての計算機は、数学の本物の実数の代わりに浮動小数点実数を使って計算を行う。これは実数を近似するものでしかない。しかし ode の内部の演算にはすべて倍精度実数を使っているので、誤差は比較的小さいと言っていいだろう。それでも、'-p 17' オプションを使って 17 桁で計算結果を出力すれば、本物の実数と浮動小数点実数の違いを見ることができる。ほとんどの計算機では、倍精度実数の精度は 17 桁だからである。

二つ目は理論打ち切り誤差と呼ばれるものである。これは真の解と、求解に用いられるアルゴリズムに起因する近似値との差異である。たとえば平方根の値を求めるアルゴリズムではほとんどの場合、無限級数展開を行う。計算機では無限級数をそのままは扱えないので、数値解法では級数を、または項の計算を途中で打ち切る必要がある。ステップ誤差はこの打ち切り誤差と浮動小数点誤差の二つをあわせたものである。しかし現実的には浮動小数点誤差はあまり問題にならないので、ode は打ち切り誤差だけからステップ誤差の値を推定する。

ある種の初期値問題では、ステップ幅を小さくすれば、求解を行う区間全体にわたる誤差の累積値を小さくすることができる。少なくともある範囲内のステップ幅でそうできるのであれば、そのときこの求解アルゴリズムは「安定」であると言われる。この意味で、Runge–Kutta–Fehlberg ('-R') 法と Adams–Moulton ('-A') 法は多くの場合、(原理的にではなく経験的に) 安定である。

これらの議論をふまえて、数値的な求解において累積誤差や不安定性の生じる主な原因を以下に挙げる。累積誤差が大きくなったり不安定になったりする系は普通、「悪い点」に近いところでステップ誤差が大きくなることに起因する。

1. 特異点。

   ode での求解は、特異点を含む区間で行うべきではない。解こうとしている微分方程式系の特異点や未定義である点をまたぐようなステップを、ode でとろうとするべきではない。

   特異点、正則な特異点、正則でない特異点などの定義は、多くの微分方程式に関する本に解説されている。いい本が見つからなければ、Birkhoff と Rota による *Ordinary Differential Equations* の Chapter 9 を読むとよい。ode で求解をしようとする前に、どんな系でも、まず特異点があるかどうか、あるならの場所と種類をよく調べるべきである。

2. 不良設定問題。

   対象とする区間全体において ode で精密な解を得ようとするなら、その区間において真の解が定義されていて、かつその振る舞いの性質がよいことが求められる。また真の解は実数でなければならない。これらの条件のどれかが成立しない場合、その問題は「不良設定」である、と言われる。また、区間内での振る舞いがよくても不良設定となる場合もある。たとえばステップ幅が突然、精度限界近くまで小さくなったり、近似数値解の値が急上昇したりといったおかしな計算結果が得られる場合も、不良設定問題であることによる場合がある。

   以下の、一見問題のなさそうな系が不良設定問題 (解が定義されないところがある) のよい例である。

   この解は $t > 0$ で定義される、以下の関数である。

   $$y(t) \;=\; -1/t.$$

   この問題では数値解は $t = 0$ を含む区間では計算できない。'step' 文を以下のように入力すると、それがわかる。

   ```
   step 1, -1
   ```

   ode がどのように振る舞うか、わかっただろうか？

   もうひとつ不良設定問題の例を挙げる。以下の系

   $$y' = 1/y$$

   は $y = 0$ で未定義となる。一般解は

   $$y \;=\; \pm\ (2(t - C))^{1/2},$$

   であり、$y(2) = 2$ という条件を与えると解は $(2t)^{1/2}$ となる。'step' 文で指定される区間に t が負の領域が含まれていると、その領域では求解は正しく行われない。

   一般的に、ステップ幅を固定するときは、悪い点をまたぐことのないようにするべきである。ステップ幅の適応制御を行うときは、ode が悪い点を見つけることもあるが、いつも見つけられるとは限らない。

3. 臨界点。

   微分方程式の右辺に明示的に独立変数を含まないような系を自励系と呼ぶ。自励系の臨界点とは、右辺がすべて 0 になる点のことである。たとえば以下の系

   $$\begin{aligned} y' &= 2x \\ x' &= 2y \end{aligned}$$

   では、$(x, y) = (0, 0)$ の一点だけが臨界点である。

   臨界点はまた、よどみ点とも呼ばれる。それは、系がその近くの点にあると激しい挙動を示すことがあるが、その点上にある系はずっとそのままだからである。環境によっては、臨界点の近くを通ると累積誤差が大きくなることがある。

   実際に上の系を ode を使って、 $x(0) = y(0) = 0$ という初期条件の元で解いてみると、時間が進んでも解は一定値のままである。初期値を臨界点の近くの別の点に動かしてみると、どうなるだろうか。

   ode で求解をしようとする前に、どんな系でも、まず臨界点の場所をよく調べるべきである。臨界点にはいくつもの種類 (平衡点、渦点、不安定点、安定点、その他) があり、種類が分かると有益なことがある。臨界点についての詳細は、微分方程式の性質についての本をみると解説してある (たとえば Birkhoff と Rota の *Ordinary Differential Equations*, Chapter 6 など)。

4. 不適切なアルゴリズム。

   ode の計算結果が不安定な挙動を示したり、ステップ誤差を抑えるためにステップ幅が極端に小さくなったりしたときは、単に求解アルゴリズムがその問題に適していないのかもしれない。たとえば、「硬い」系をうまく解けるような数値解法は現在、ode には実装されていない。

   以下の例で、硬い系とはどのようなものが見ることができる。

   $$y' = -100 + 100t + 1$$
   $$y(0) = 1$$

   区間は $[0, 1]$ とする。厳密解は

   $$y(t) = e^{-100t} + t.$$

   である。ode で求解アルゴリズム、ステップ幅、許容相対ステップ誤差を様々に変えて、得られる解曲線と厳密解を比較してみるとよい。

ode の計算した近似数値解の精度を、手軽におおざっぱに確認してみる方法も、いくつかある。順番に解説する。

1. 解曲線の安定性を見る: 曲線は収束していっているか？

   これは、ステップ幅の調整が解曲線にどう影響しているかを見るということである。ステップ幅を小さくすると曲線は収束していくはずである。そうならない場合は、ステップ幅が十分に小さくなっていないか、求解アルゴリズムが適切でないということである。実際には、以下のようにするといいだろう。

   - ステップ幅の適応制御を行っている場合は、ステップ誤差の許容範囲を段階的に小さくして (‘-r 1e-9’、‘-r 1e-11’、‘-r 1e-13’、...) 解曲線を描いてみる。それで解曲線が収束したら、解は区間全体で安定に振る舞い、それは精度は十分であるということである。
   - ステップ幅を固定して計算している場合、ステップ幅をその半分にして、上と同様に解がどう変わるかを見てみる。

   これを試す例を以下に示す。以下の内容を含む ‘qcd’ というファイルがあるとする。

   ```
   # an equation arising in QCD (quantum chromodynamics)
   f'   = fp
   fp'  = -f*g^2
   g'   = gp
   gp'  = g*f^2
   f = 0; fp = -1; g = 1; gp = -1

   print t, f
   step 0, 5
   ```

   次に、以下の内容を持つ ‘stability’ というファイルを作る。

   ```
   # sserr is the bound on the relative single-step error
   for sserr
   do
   ode -r $sserr < qcd
   done | spline -n 500 | graph -T X -C
   ```

   ファイル ‘stability’ は「シェル・スクリプト」であり、許容相対ステップ誤差の指定値を変えながら、解曲線を重ねて描いていく。これを以下のように実行する。

```
sh stability 1 .1 .01 .001
```

指定された許容誤差の場合の解曲線がそれぞれ描かれる。どう収束していくのか、つまり安定性を見て取ることができる。

2. 系の中の不変量: それは定数か？

   不変量を含む系は多数ある。たとえば物理学においてなんらかの「保存則」の数理モデルがあるとするとエネルギー (その系の従属変数の関数) は時間の経過によっては変化しない。一般に、従属変数の振る舞いがどういった性質なのかが分かっていれば、解の性質を調べるのに有利である。

3. 解曲線の「族」: それらは発散していくか？

   問題としている系の初期値を様々に変えることで、解曲線の曲線族が得られるが、それを見ると誤差がどのように累積していくか、おおざっぱに見ることができる。それがはっきりと発散する場合、つまり非常に近い初期値から始まっているのに、区間の終端ではかなり離れた値になる場合、その解曲線はおかしいのかもしれない。一方で解曲線があまり離れていかない場合、区間全体に渡って誤差は、あらゆる可能性において、その桁が変わるほどには増えていないということである。そういった、近い点から出発した解がさほど大きく離れていかないような系は、良設定であると言われることがある。

8.7 実行時間

ode が微分方程式系を数値的に解くのにかかる時間は、多くの要因に左右される。そのうちのいくつかを挙げると、含まれる方程式の数、方程式の複雑さ (演算子の個数や性質)、求解アルゴリズムがとるステップの数 (ステップ幅が適応制御される場合、ステップ幅を決めるのには非常に複雑な計算が必要である) などである。時間を無駄にせずに求解にかかる時間を見積もるには、短い区間で、あるいは低い精度で計算時間をはかってみるのがよい。それは、ステップを 2 回進めるのにかかる時間は 1 回の時間の 2 倍程度であろうこと、および指定された許容相対誤差とステップ数にはなんからの関係があるだろう、ということから考えられる。そのなんらかの関係は、求解アルゴリズムや系の「硬さ」、また許容相対誤差自体によって決まると考え、注意深く何回か実行してみれば、その関係の見当をつけることができる。また、区間全体で要求精度で計算するとどの程度の時間がかかるかを反映するような、精度の低い求解をその区間でやってみることができるだろう。最後に、データが大量に出力されるときは、解の計算自体よりも画面への出力に時間がかかっている、というのもありがちである。

8.8 ode の記述言語

以下に ode が入力として受け付ける言語の仕様を、バッカス・ナウア記法で示す。文法における非終端記号は <カッコ> で示す。終端記号は大文字のみで示す。カッコのない語や記号は、そのままを表す。

```
<program>    ::=          ... empty ...
               |  <program> <statement>
```

```
<statement>  ::=  SEP
                |  IDENTIFIER = <const> SEP
                |  IDENTIFIER ' = <expression> SEP
                |  print <printlist> <optevery> <optfrom> SEP
                |  step <const> , <const> , <const> SEP
                |  step <const> , <const> SEP
                |  examine IDENTIFIER SEP


<printlist>  ::=  <printitem>
                |  <printlist> , <printitem>


<printitem>  ::=  IDENTIFIER
                |  IDENTIFIER '
                |  IDENTIFIER ?
                |  IDENTIFIER !
                |  IDENTIFIER ~


<optevery>   ::=        ... empty ...
                |  every <const>


<optfrom>    ::=        ... empty ...
                |  from <const>


<const>      ::=  <expression>


<expression> ::=  ( <expression> )
                |  <expression> + <expression>
                |  <expression> - <expression>
                |  <expression> * <expression>
                |  <expression> / <expression>
                |  <expression> ^ <expression>
                |  FUNCTION ( <expression> )
                |  - <expression>
                |  NUMBER
                |  IDENTIFIER
```

上に示した文法には、また曖昧さが残っている。そこで演算子の結合の優先順位を以下の表に示す。表の上の演算子ほど優先して結合される。

```
Class           Operators    Associativity

Exponential         ^            right
Multiplicative      * /          left
Additive            + -          left
```

上述の文法中に示されているように、記述対象が自明でない文が 6 種類ある。各種類の「意味付け」を順番に説明する。

1. IDENTIFIER ’ = <expression>

   一階の微分方程式を定義する。IDENTIFIER の導関数が <expression> で指定される。動的変数がこの形式の左辺に現れない場合は、その導関数は 0 のままであると仮定される。その場合、それは記号定数となる。

2. IDENTIFIER = <expression>

   IDENTIFIER の値を <expression> の現在の値にする。この文で初期化されていない動的変数は、0 になる。

3. step <const> , <const>

4. step <const> , <const> , <const>

   ‘step’ 文で求解アルゴリズムが実行される。最初の <const> は独立変数の初期値である。二つ目は求解を行う区間の終端の独立変数値である。三つ目はステップ幅である。ステップ幅が与えられたときは、コマンドラインで指定されていたステップ幅をこの指定で上書きする。ステップ幅を指定することはあまりなく、求解アルゴリズムによって適応的に決定させることが多い。

5. print <printlist> [every <const>] [from <const>]

   ‘print’ 文ではどの数値をいつ出力するかを指定する。<printlist> はコンマで区切られた IDENTIFIER のリストで、各 IDENTIFIER の後ろに ‘’’ を付けて導関数であることを、‘?’ を付けて相対ステップ誤差であることを、‘!’ を付けて絶対ステップ誤差であることを、‘~’ を付けて累積誤差であることを示すことができる (最後のは未実装)。指定された値は、指定された順序で出力される。‘every’ 節と ‘from’ 節は指定してもしなくてもよい。‘every’ 節を指定すると <const> ごとに指定された値が出力される。デフォルトは毎ステップ (つまり ‘every 1’) である。最初と最後のステップでの値は、かならず表示される。もし ‘from’ 節があった場合、独立変数の値が <const> を超えてからはじめて出力が行われる。デフォルトでは、ステップ開始と同時に出力が始まる。

   ‘print’ 文が現れない場合は、与えられた微分方程式系の独立変数とすべての従属変数が表示される。最初に表示されるのは独立変数である。従属変数は、方程式系で定義されている順に表示される。

6. examine IDENTIFIER

   ‘examine’ 文は、実行されると名前の与えられた変数に関する詳しい情報が表の形で表示される。たとえば、どこかで出てきた ‘ode to Euler’ の例を実行したときに ‘examine y’ とすると、以下のような情報を表示する。

```
"y" is a dynamic variable
value:2.718282
prime:2.718282
sserr:1.121662e-09
aberr:3.245638e-09
acerr:0
 code:  push "y"
```

'dynamic variable' (動的変数) という言葉は、y の挙動を記述する微分方程式が系の中にある、という意味である。また表示されている数値の意味はそれぞれ、以下の通りである。

value　現在のその変数の値。

prime　現在のその変数の導関数値。

sserr　最後のステップにおける、その変数の相対ステップ誤差。

aberr　最後のステップにおける、その変数の絶対ステップ誤差。

acerr　最後に実行された 'step' 文による求解での累積誤差。この機能は現在はまだ実装されていない。

'code' で表示されている部分は y の導関数値を計算するために必要な一連のスタック操作のリストである (往年の逆ポーランド式電卓のようなものである)。ここをみれば、導関数値の計算が正確にはどのように行われるのか、その計算にかかるメモリや時間はどの程度か、が分かる。'push "y"' は y の値をスタックに積むことを表す。この系の場合は、導関数値を計算するのに必要な操作はこれですべてである。

ode 入力言語の文法には、４種類の終端記号がある。FUNCTION、IDENTIFIER、NUMBER、および SEP である。以下にその意味を示す。

1. FUNCTION

   abs、sqrt、exp、log、ln、log10、sin、cos、tan、asin、acos、atan、sinh、cosh、tanh、asinh、acosh、atanh、floor、ceil、besj0、besj1、besy0、besy1、erf、erfc、inverf、lgamma、gamma、norm、invnorm、ibeta、igamma のうちの一つである。これらの定義はプロット・プログラムの gnuplot の同名の関数と同じである。ibeta は３つの、igamma は２つの引数を取るが、他の関数の引数はすべて１つである。三角関数の引数はラジアン単位で与える。atan 関数は −PI/2 以上 PI/2 以下の値を返すようになっている。

2. IDENTIFIER

   英字で始まり英数字からなる文字列である。最初の 32 文字だけが有効である。大文字と小文字は別のものとして扱われる。この識別子名の中では、下線 (アンダースコア) は英字の一種として扱われる。関数名やキーワード、および 'PI' は識別子名に使わない。

3. NUMBER

   数字からなる空でない文字列。小数点、指数部を含んでいてもよい。指数部は 'e' または 'E' に 1、2、3 個のいずれかの個数の数字を続けたものである。一つ目の数字の前には符号がついていてもよい。数字はすべて十進数として扱われる。数値としては空白を含んでいてはならない。'PI' は数値である。

4. SEP

   区切り文字。セミコロンか、エスケープされてない改行。

ode 入力言語では大文字と小文字は別の文字である。'#' から行末まではコメントとして無視される。長い行は分割できる。その際、上になった行の行末にはバックスラッシュ ('\') を置く。バックスラッシュと改行が連続している時、その二つをまとめて一つの空白文字と等価であると見なされる。

入力中で識別子、数値、キーワードを他のものと分離するために、空白文字とタブを使う。空白類は区切り文字と同様、例外的に扱われる (つまり無視される)。

8.9 微分方程式について

- K. E. Atkinson, *An Introduction to Numerical Analysis*, Wiley, 1978. Chapter 6 には常備分方程式の数値解法の参考文献について述べられている。
- G. Birkhoff and G. Rota, *Ordinary Differential Equations*, 4th ed., Wiley, 1989.
- N. B. Tufillaro, T. Abbott, and J. Reilly, *An Experimental Approach to Nonlinear Dynamics and Chaos*, Addison–Wesley, 1992. Appendix C に初期の ode について述べられている。
- N. B. Tufillaro, E. F. Redish, and J. S. Risley, "ode: A numerical simulation of ordinary differential equations," pp. 480–481 in *Proceedings of the Conference on Computers in Physics Instruction*, Addison–Wesley, 1990.

9 libplot ライブラリ

9.1 libplot プログラミング

GNU libplot バージョン 4.4 は、ベクトル形式の二次元画像を描くためのフリーのライブラリである。二重バッファリングを使った滑らかなアニメーションを X Windows システム上に表示することができ、また画像を様々なフォーマットに書き出すことができる。出力する画像形式やディスプレイの種類によらない API (application programming interface) を実装しており、その意味で、このライブラリは特定の出力デバイスに依存しない、と言える。

plotutils のライブラリは C、C++ およびその他の言語から利用することができる。もっともよく使われるのは C バインディングであり、これを libplot と呼んでいる。また C++ バインディングには (libplot と分けて呼ばねばならないときは) libplotter と呼ばれるものがある。この章では libplot という名前はバインディングとは関係なくこのライブラリそのものをさす名前として用いる。

libplot で描画できるオブジェクトには、曲線、円と楕円、点、記号、サイズを調整したラベル (適切な大きさの文字列) がある。ここで、線分がつながったもの、円弧、楕円の弧、二次のベジェ曲線、三次のベジェ曲線がつながった線をパスとよぶ。パスには、閉曲線とそうでないものが描ける。閉曲線、円、楕円については、塗りつぶしができる。塗りつぶしの色はペンの色と同様に指定でき、また塗りつぶしのパターンも指定できる。

PostScript 言語風の、描画状態 (グラフィクス・コンテキスト) のスタックを管理する機能もあり、ある時点での描画に関する様々な属性をまとめて、スタックで管理することができる。パスに付けられる属性には、線の太さ、線種、両端の形、線のつなげ方、変換行列があり、文字列の属性にはフォント名やフォントサイズ、テキストの回転角、変換行列がある。

libplot が提供する基礎的な属性の一つに、*Plotter* の属性がある。Plotter はオブジェクトであり、いわゆる従来のペン・プロッタと同様の使い方で、これに対して操作を行うことでベクトル形式の画像を描画できる。Plotter にはいろいろな種類の出力形式のものがある。一つのアプリケーション中に、Plotter オブジェクトはどの種類のものがいくつ保持されていてもよい。

どの種類の Plotter オブジェクトでも、できる描画操作は同じであり、デバイスの種類に依存しないようになっている。したがって libplot を利用するアプリケーションは、libplot がサポートするどの画像形式にも簡単に出力できる。

現在、以下の種類の Plotter オブジェクトが実装されている。

- X の Plotter。このデバイスはオープンされるときに X window のディスプレイ上にウィンドウを開き、以降の描画はその中に行われる。オブジェクトに対してクローズ操作を行うと、ウィンドウ内の画像が消える。そのあと再びオープンされた時には、新しいウィンドウがまた作られる。画像が消えてもウィンドウそのものはディスプレイ上に残っているが、そのウィンドウ内で 'q' を入力するかマウスクリックをすると、ウィンドウは閉じられる。将来的には、閉じられた X の Plotter がまたオープンされたときには、既にあるウィンドウを再利用するようになる予定である。
- X Drawable の Plotter。これは一つまたは二つの「X Drawable」を X Windows システムのディスプレイ上に表示する。X Drawable とは、X のウィンドウまたはビットマップ画像のことである。それらを Plotter デバイスにパラメータとして渡すことで描画が行われる (Section 9.5 [Plotter Parameters], page 128 参照)。
- PNG の Plotter。これは出力を 1 ページの PNG (Portable Network Graphics) 形式の画像として生成し、ファイルまたは標準出力に書き出す。出力されたファイルは、たと

えばフリーの ImageMagick パッケージの display などで表示、あるいは編集することができる。

- PNM の Plotter。これは出力を 1 ページの “portable anymap” 形式の画像として生成し、ファイルまたは標準出力に書き出す。PNM には、PBM (portable bitmap、白黒画像)、PGM (portable graymap、グレイスケール画像)、PPM (portable pixmap、カラー画像) の形式がある。出力されるファイルは、この 3 つの形式のうちのどれかもっとも適切な形式になる。出力されたファイルは、たとえばフリーの display などで表示、あるいは編集することができる。
- GIF の Plotter。これは出力を 1 ページの疑似 GIF 形式の画像として生成し、ファイルまたは標準出力に書き出す。本物の GIF 形式と違って、疑似 GIF 形式では LZW 圧縮ではなくランレングス圧縮を行う。そのため LZW 圧縮の利用を制限している Unisys の特許には触れない。それでも、出力されたファイルは、たとえばフリーの display などで表示、あるいは編集することができる。アニメーション疑似 GIF の生成もサポートしている。
- SVG の Plotter。これは出力を 1 ページの Scalable Vector Graphics 形式の画像として生成し、ファイルまたは標準出力に書き出す。SVG はウェブ上でベクトル形式の画像を扱うための、XML ベースの画像形式である。詳細は W3 Consortium (http://www.w3.org) にある。また開発者のための情報はそのウェブサイトの Graphics Activity (http://www.w3.org/Graphics) にある。libplot では SVG の 2000 年 3 月 3 日版にしたがった出力を行う。
- Illustrator の Plotter。これは出力を 1 ページのアドビの Illustator 形式の画像として生成し、ファイルまたは標準出力に書き出す。出力されたファイルは、アドビ Illustrator (バージョン 5 以上) などのソフトウェアで表示、あるいは編集することができる。
- PostScript の Plotter。これは出力を PostScript 形式の画像として生成し、ファイルまたは標準出力に書き出す。もし画像が 1 ページだけであれば、このデバイスは EPS (encapsulated PostScript) を出力する。これは他の文書に取り込んだりすることができる。これはドロー・ソフトウェアの idraw で表示、あるいは編集することができる。Section E.1 [idraw], page 163 を参照のこと。
- CGM の Plotter。これは出力を Computer Graphics Metafile 形式の画像として生成し、ファイルまたは標準出力に書き出す。デフォルトではバイナリ形式の CGM 形式バージョン 3 で出力される。CGM 形式出力はウェブ対応のベクトル画像形式である WebCGM プロファイルに対応しているため、WebCGM に対応しているウェブ・ブラウザならどれでも表示できる。WebCGM については CGM Open Consortium (http://www.cgmopen.org/) を参照のこと。
- Fig の Plotter。これは出力を 1 ページの Fig 形式の画像として生成し、ファイルまたは標準出力に書き出す。これはドロー・ソフトウェアの xfig で表示、あるいは編集することができる。xfig を使えば、様々な画像フォーマットに書き出して、他の文書に取り込んだりすることができる。Section E.2 [xfig], page 163 を参照のこと。
- PCL の Plotter。これは出力を PCL 5 形式の画像として生成し、ファイルまたは標準出力に書き出す。PCL 5 はヒューレット・パッカードの強力なプリンタ制御言語で、ベクトル形式画像をサポートしている。PCL 5 をサポートしているのは LaserJet プリンタや比較的高級なインクジェットプリンタなどであり、この Plotter の出力はそういったプリンタで印刷することができる。
- HP-GL の Plotter。これは出力を HP-GL (ヒューレット・パッカード・グラフィクス・ランゲージ) 形式の画像として生成し、ファイルまたは標準出力に書き出す (デフォルトでは

HP-GL/2)。この形式の画像は他のソフトウェアで読み込んだり、プロッタで描いたりすることができる。

- ReGIS の Plotter。これは出力を ReGIS (remote graphics instruction set) 形式の画像として生成し、ファイルまたは標準出力に書き出す。この形式の画像は ReGIS 形式に対応した端末やエミュレータで表示することができる。DEC の VT340、VT330、VT241、VT240 端末や dxterm などである。
- Tektronix の Plotter。これは出力を Tektronix 4014 形式の画像として生成し、ファイルまたは標準出力に書き出す。この出力は、Tektronix 4014 エミュレータならどれでも表示させることができる。X Window のターミナル・エミュレータの xterm にその機能がある。MS-DOS 版の kermit でも可能である。
- Metafile の Plotter。これは出力を GNU のグラフィクス・メタファイル graphics metafile 形式の画像として生成し、ファイルまたは標準出力に書き出す。この形式はいくつかの OS に搭載されている 'plot(5)' の形式を拡張したものである (Appendix D [Metafiles], page 161 参照)。この形式は GNU plot コマンドを使って他の形式に変換することができる (Chapter 3 [plot], page 27 参照)。

上述の Plotter デバイスの違いはまず、X と X Drawable を除くと、ファイルまたは他の出力先に書き出す形式の違いである。X の Plotter はウィンドウを新しく開き、X Drawable は一つか二つの X Drawable の中に画像を描く。

次に、最初の 5 つの Plotter デバイス (X、X Drawable、PNG、PNM、GIF) はビットマップ画像を、他のものはベクトル形式の画像を描くという違いがある。ビットマップ画像では描かれているオブジェクトの構造などの情報は失われるが、ベクトル形式では保存されている。

また、X、X Drawable、ReGIS、Tektronix、Metafile の Plotter デバイスはリアルタイム・デバイスである。これは、デバイスに対する描画操作が全部終了していなくても、各操作が行われるたびに出力が逐次行われるということである。他のデバイスはリアルタイム・デバイスではないので、必要な関数がすべて呼ばれた後でないと出力は行われない。PNM と GIF の各 Plotter では、そうでないとビットマップ画像を生成できないからである。Illustrator と PostScript の Plotter ではまず最初に、画像のサイズ ('bounding box') を出力する必要がある。Fig の Plotter では色の定義をまず最初に出力せねばならない。

どの Plotter デバイスに対しても行えるもっとも重要な操作は、openpl と closepl である。これによってそのデバイスをオープン、クローズする。画像の描画や様々な属性の設定などは、すべて openpl...closepl の間に行う必要がある。一対の openpl...closepl の間に行われた描画操作から、ひとつのページの画像が構築される。原則的に、Plotter デバイスは何度もオープン／クローズできる。X の Plotter は各ページをそれぞれのウィンドウで表示し、PostScript、PCL、HP-GL の Plotter は各ページの画像をそれぞれ、物理的に別の紙に描画する。X Drawable と Tektronix の Plotter では、複数のページを一つの X Drawable またはディスプレイ上に順に表示する。リアルタイムでない Plotter (PNG、PNM、GIF、Illustrator、PostScript、CGM、Fig、PCL、HP-GL の Plotter) は、その Plotter オブジェクトが破棄されるのを待ち、そのときに描画されたすべてのページを出力する。

現在の libplot では PostScript と CGM の Plotter ではこの出力に遅延がある。PCL と HP-GL の Plotter では、たとえば closepl が呼ばれるなどして各ページが作られるたびに出力する。PNG、PNM、GIF、SVG、Illustrator、Fig の各 Plotter でも同様だが、最初のページだけしか出力されない。PNG、PNM、GIF、SVG、Illustrator、Fig では複数ページの画像はサポートされていないからである。

画像が描かれる「描画領域」('viewport') は、その Plotter デバイスの出力形式における正方形、あるいは長方形の領域である。どの Plotter デバイスでも、各デバイスが持っている座標系ではなく、プログラム側で座標系を自由に設定でき、デバイスの種類とは関係なく、設定したその座標系上で描画操作を行うことができる。各 Plotter デバイスは、内部でその指定された座標系からデバイス上の座標系にアフィン変換を行う。

`openpl` で Plotter デバイスをオープンしたあと、すぐ `space` を呼んで、Plotter、つまりこの出力形式変換オブジェクトを初期化する。`space` は、プログラム側座標系の中に長方形の領域 (「ウィンドウ」) を想定し、そのデバイスの座標系とのマッピングをつくる。それにより設定されるアフィン変換は、`space` または `fconcat` を呼べばいつでも再設定あるいは設定の変更ができる。`fconcat` は現在設定されているアフィン変換と、この関数で指定するアフィン変換の合成変換を設定する。こういった変換の合成は、たとえば PostScript などでよく使われている機能である。

それぞれの Plotter デバイスは描画状態 (グラフィクス・コンテキスト) を PostScript 風のスタックで管理できる。描画状態には現在設定されている、指定座標系からデバイス上の座標系へのアフィン変換が含まれる。またその時点での描画カーソルの位置、線種、線の太さ、ペンや塗りつぶしの色、文字列のフォントなども含まれる。もしまた描き終わっていないパス (つなげられた線) があれば、その状態 (描き終わってないこと) も含まれる。パスは一部分 (線分や円弧、ベジェ曲線) を一つ一つ順番につなげて作られるからである。

注意: PostScript でも現在の描画状態は `savestate` を呼ぶことでスタックに積まれ、`restorestate` でスタックから取りだされる (ポップされる)。しかし `libplot` と PostScript の描画モデルはかなり異なっている。`libplot` では `savestate` によって、状態を保存すると同時に新しく生成される描画状態はパスをなにも含まない。したがって新しいパスはその中で新たに置かれ、描画される。その後、前の描画コンテキストのパスは `restorestate` が呼ばれたときに復元され、場合によってはそこからパスの構築が続けられる。また `libplot` では、新たにパスを作り始めるときに '`newpath`' を呼ぶ必要がないのも PostScript と違う点である。単に新しくパスを構成する要素の描画をはじめればよいだけである。少なくとも機能としては、パスの構築を終了するときには明示的に `endpath` を呼ぶことで、描画領域に実際に描画が行われる。しかし実際には `endpath` の呼び出しは省かれることもある。構築中のパスがあるときに `restorestate` を呼ぶとその内部で `endpath` が呼び出され、そのときの描画状態でのパスの構築は終了する。

ベクトル形式の画像でのアニメーションは、画像の各ページを複数のフレームに分けることで可能になる。`erase` を呼ぶことで、現在のフレームの作成を終了し、次のフレームをはじめることができる。リアルタイム描画を行う Plotter デバイス (X、X Drawable、ReGIS、Tektronix、Metafile) では、これによってグラフィクス・ディスプレイ上の画像が消去され、次のフレームが描画できる状態になる。一連のフレームを次々に表示することで、滑らかなアニメーションのように見える。

ほとんどのリアルタイム描画のできない Plotter デバイス (PNG、PNM、SVG、Illustrator、PostScript、CGM、Fig、PCL、HP-GL) では、`erase` で内部のバッファ上にあるすべての描画オブジェクトが削除される。そのためリアルタイムでないほとんどの Plotter は、複数フレームのページがあっても、最後のフレームだけしか表示しない。

GIF Plotter もリアルタイム描画はできないが、他のと違ってアニメージョン疑似 GIF のような複数画像からなる出力を、複数フレームのページから生成できる。上述したように、GIF Plotter の生成する疑似 GIF では、画像の最初のページだけのファイルしか生成できない。しかしそのページに複数のフレームがあるときは、`erase` を呼べば、デフォルトではそこが二つのフレームの切れ目になる。

9.2 libplot を使った C プログラミング

9.2.1 C の API

libplot にはいくつかの言語バインディングがある。どのバインディングが使われていても、libplot の概念 (Plotter デバイスと、どのデバイスでも使える共通の描画操作) は同じである。しかし Plotter の実際の操作の仕方 (生成、描画対象としての選択、破棄) は言語によって異なる。この節では C バインディングについて説明する。以前の古い形式の C バインディングについては Section 9.2.2 [Older C APIs], page 89 を参照のこと。

C バインディングでは、Plotter オブジェクトは抽象化されたデータ型 plPlotter として実装されており、その型へのポインタを使って操作される。様々な描画操作を行う関数には、第一引数として plPlotter 型のインスタンスへのポインタを渡す。関数 pl_newpl_r と pl_deletepl_r がそれぞれ plPlotter 型オブジェクトのコンストラクタとデストラクタである。pl_newpl_r 関数には、最後の引数に plPlotterParams オブジェクトへのポインタを渡す。これで Plotter のさまざまなパラメータを指定する。pl_newpl_r は plPlotter 型オブジェクトへのポインタを返す。

Plotter に対して、したい操作をすべて行ったら、pl_deletepl_r を呼ぶ。一般に、リアルタイムでの描画を行わない Plotter (特に PostScript と CGM) は、pl_deletepl_r が呼ばれたときにはじめて、実際の出力を行う。

以下に、C バインディングでの Plotter を操作する関数を挙げる。

plPlotter * pl_newpl_r (const char *type, FILE *infile, FILE *outfile, FILE *errfile, plPlotterParams *params);

出力形式が *type* の Plotter オブジェクトを生成する。*type* は"X"、"Xdrawable"、"png"、"pnm"、"gif"、"svg"、"ai"、"ps"、"cgm"、"fig"、"pcl"、"hpgl"、"regis"、"tek"、"meta" のいずれかである。Plotter への入出力には、標準入力を *infile* に、標準出力を *outfile* に、標準エラー出力を *errfile* にそれぞれ指定できるが、どれか、または 3 つともを NULL にしてもよい。現時点での実装では Plotter は出力するのみで入力を受け付けないので、*infile* は指定しても無視される。また X と X Drawable の Plotter は標準出力ではなく X Window システムの画面に表示を行うので、*type* が "X" または "Xdrawable" のときは *outfile* が無視される。*errfile* が NULL でなければ、なにかエラーが発生したときにはそこに出力される。

Plotter のパラメータはすべて plPlotterParams から *params* が指す先のオブジェクトにコピーされる。Plotter のオブジェクトが生成できなかったときには NULL が返される。

int pl_deletepl_r (plPlotter *plotter);

引数で渡された Plotter オブジェクトを破棄する。破棄に失敗したときは負の値を返す。

また、plPlotter とは別に plPlotterParams という型を用意している。関数 pl_newplparams、pl_deleteplparams、pl_copyplparams がそれぞれ、plPlotterParams のコンストラクタ、デストラクタ、複製関数である。関数 pl_setplparam は、plPlotterParams オブジェクト中の Plotter のパラメータを一つだけ設定する。

plPlotterParams * pl_newplparams ();
int pl_deleteplparams (plPlotterParams *plotter_params);
plPlotterParams * pl_copyplparams (const plPlotterParams *params);
int pl_setplparam (plPlotterParams *params, const char *parameter, void *value);

params が指すオブジェクト中のパラメータ *parameter* の値を *value* に設定する。ほとんどのパラメータでは *value* は char *、つまり文字列である。*value* に NULL を指定すると、そのパラメータは未設定の状態になる。

指定できるパラメータとその意味のリストは Section 9.5 [Plotter Parameters], page 128 を参照のこと。無効なパラメータを指定したときは、単に無視される。

基本的に Plotter の操作は Plotter の種類にはよらないようなってはいるが、種類によってはその生成時になんらかの特定の動作をさせたいことがあるかもしれない。それを可能にするために plPlotterParams をわざわざ独立した型にしている。plPlotterParams オブジェクト中のパラメータが設定されている時、生成される Plotter はその値を参照する。文字列の値を取るパラメータについては、それが設定されていないときは同名の環境変数が参照される。環境変数が設定されていればそれを、環境変数も設定されていないときはデフォルトの値を使う。これにより実行時にも柔軟に設定を行うことができる。アプリケーションの動作に致命的な影響を与えるパラメータ以外については、環境変数で設定するようにしてもよい。

C バインディングでは Plotter に対して行われる描画操作はそれぞれ、"pl_" で始まり"_r" で終わる名前の関数を呼ぶことで行われる。たとえば Plotter に対して openpl 操作を行う関数の名前は pl_openpl_r であり、関数の最初に引数として操作対象の plPlotter へのポインタを渡す。

9.2.2 古い形式の C 言語の API

現在の C 言語 API (application programming interface) は古い API を改良したものである。この改良でスレッド・セーフになった。現在の API の関数名の末尾に "_r" がついているのは、改良版またはスレッド・セーフであること (revised または reentrant) を示すためである。

古い C 言語 API では、操作対象とする Plotter は、操作を行う関数の引数として渡すようにはなっていなかった。代わりにまず Plotter を「選択」しておいてから操作関数を呼ぶ必要があった。たとえば、Plotter に新しくページをはじめる操作を行う関数 pl_openpl は、現在選択されている Plotter に対してその操作を行う。

この古い API の使用は推奨されないが、現在でもサポートはされている。これには、以下の４つの関数がある。

int pl_newpl (const char *type, FILE *infile, FILE *outfile, FILE *errfile);

出力形式が *type* の Plotter オブジェクトを生成する。*type* は"X"、"Xdrawable"、"png"、"pnm"、"gif"、"svg"、"ai"、"ps"、"cgm"、"fig"、"pcl"、"hpgl"、"regis"、"tek"、"meta" のいずれかである。Plotter への入出力には、標準入力を *infile* に、標準出力を *outfile* に、標準エラー出力を *errfile* にそれぞれ指定できる。返り値は、生成された Plotter オブジェクトを表す非負の整数で、これをハンドルと呼ぶ。Plotter を生成できなかったときは負の値を返す。

int pl_selectpl (int *handle*);

は引数で示される Plotter を選択し、操作できる状態にする。この関数で選択されている一つの Plotter だけが操作対象になる。選択に失敗したときは負の値を返す。選択できたときは、それまでに選択されていた Plotter のハンドルを返す。

プログラムの実行開始時に、標準出力に metafile 形式の画像を書き出す Plotter オブジェクト (ハンドル '0') が一つだけ生成され、選択された状態になるようになっている。

int pl_deletepl (int *handle*);
ハンドルで指定された Plotter を破棄する。破棄する Plotter は選択されていてはならない。破棄に失敗したときは負の値を返す。

int pl_parampl (const char **parameter*, void **value*);
Plotter のパラメータ *parameter* の値を *value* に設定する。それ以降に生成されるどの Plotter も、設定されたパラメータの値を Plotter オブジェクト内にコピーする。

この古い API では、pl_selectpl による Plotter の選択と pl_parampl によるパラメータ値の設定はどちらも「大域的」な操作であり、したがってスレッド・セーフではない。

もっと古い C 言語 API では、libplot の関数名から "pl_" を取った名前になっていた。"pl_" は、以前の GNU 版でない libplot と GNU 版のものを区別するために付けられたものである。GNU 版でもそうでないものでも、古い libplot を使ったプログラムをコンパイルする必要があるときは、ヘッダファイル plotcompat.h を使うとよい。このヘッダファイル中では openpl を pl_openpl で、と言った具合に古い関数を新しい関数で定義し直している。Section 9.2.3 [C Compiling and Linking], page 90 参照。

9.2.3 C 言語でのコンパイルとリンク

C 言語で libplot の C API (C application programming interface) を使ったグラフィクス・アプリケーションを書く場合、そのソースファイルには以下の行が必要になる。

```
#include <stdio.h>
#include <plot.h>
```

ヘッダファイル plot.h は libplot の配布パッケージに含まれている。これを、C コンパイラが見つけられる場所に置いておく。このファイルには libplot の C API の各関数のプロトタイプ宣言やその他の定義などが入っている。

libplot の現在のバージョンでは、関数名はすべて "pl_" がその先頭に、"_r" が末尾についている。たとえば Plotter に openpl 操作を行う関数は pl_openpl_r という名前で、その最初の引数が操作対象の Plotter である。これらはすべて 'plot.h' で宣言されている。

GNU libplot の 3.0 より前のバージョンでは、違う方法で Plotter に対して操作を行っていた。たとえば pl_openpl 関数は引数に Plotter を取らず、それが呼ばれたときに「選択」されていた Plotter が操作対象であった。この古い C API も 'plot.h' でサポートされている。その詳細は Section 9.2.2 [Older C APIs], page 89 を参照のこと。

GNU libplot のもっと古いバージョンと GNU 版より前のものでは "pl_" もついていなかった。もしそういった古い libplot の関数を使っているプログラムをコンパイルする必要があるときは、ヘッダファイル plotcompat.h を使うとよい。このファイル中ではたとえば、openpl という関数呼び出しが libplot の pl_openpl になるように再定義されている。

作成したプログラムを libplot とリンクするには、コンパイル時のコマンドラインで '-l' オプションをつければよい。たとえば以下のようにする。

```
-lplot -lXaw -lXmu -lXt -lXext -lX11 -lpng -lz -lm
```

最近の X Windows システムでは以下のようになるだろう。

```
-lplot -lXaw -lXmu -lXt -lSM -lICE -lXext -lX11 -lpng -lz -lm
```

上の例では、PNG サポートを有効にしてコンパイルした libplot を使うことを前提にしている。もしそうでないなら、'-lpng -lz' オプションを指定しないようにする。

(Athena ウィジェットの代わりに Motif ウィジェットを使うシステムでは '-lplot -lXm -lXt -lXext -lX11 -lpng -lz -lm'、'-lplot -lXm -lXt -lXext -lX11 -lpng -lz -lm -lc -lgen'、'-lplot -lXm -lXt -lXext -lX11 -lpng -lz -lm -lc -lPW' のいずれかになるだろう。また最近の X Window システムではオプションに '-lSM -lICE' を追加する必要があるかもしれない。さらに Motif の最近のバージョンでは '-lXp' や '-lXpm' も必要かもしれない。)

場合によっては libplot やその他のライブラリのインストールされているディレクトリを、コマンドラインで指定する必要がある。たとえば X Window のライブラリを指定するオプション '-lXaw -lXmu -lXt -lSM -lICE -lXext -lX11' を使うには '-L/usr/X11/lib' といったオプションを同時に指定せねばならないかもしれない。

多くの場合 libplot は DLL (dynamically linked library、または shared library、共有ライブラリ) としてインストールされる。この場合、リンクはコンパイル時には行われず、プログラムの実行時に行われることになる。その際ライブラリのある場所は環境変数 LD_LIBRARY_PATH によって示される。libplot を使うプログラムが正常に実行されるためには、libplot ライブラリのあるディレクトリを LD_LIBRARY_PATH に設定しておかねばならない。

9.2.4 C 言語でのサンプル描画

以下に、libplot の機能を使ってベクトル形式の画像を描画する C 言語のプログラムの例を示す。このプログラムが描くのは、美しく入り組んだ Bill Gosper の C 曲線 (Item #135 in *HAKMEM*, MIT Artificial Intelligence Laboratory Memo #239, 1972) である。数値定数 MAXORDER の値 (ここでは 12 に固定) を大きくしていくと、無数の周天円的な八角形状の形状をなす線分からなる C の字形の曲線を描いていく。

```
#include <stdio.h>
#include <plot.h>
#define MAXORDER 12

void draw_c_curve (plPlotter *plotter, double dx, double dy, int order)
{
  if (order >= MAXORDER)
    /* continue path along (dx, dy) */
    pl_fcontrel_r (plotter, dx, dy);
  else
    {
      draw_c_curve (plotter,
                    0.5 * (dx - dy), 0.5 * (dx + dy), order + 1);
      draw_c_curve (plotter,
                    0.5 * (dx + dy), 0.5 * (dy - dx), order + 1);
    }
}
```

```
int main ()
{
  plPlotter *plotter;
  plPlotterParams *plotter_params;

  /* set a Plotter parameter */
  plotter_params = pl_newplparams ();
  pl_setplparam (plotter_params, "PAGESIZE", "letter");

  /* create a Postscript Plotter that writes to standard output */
  if ((plotter = pl_newpl_r ("ps", stdin, stdout, stderr,
                             plotter_params)) == NULL)
    {
      fprintf (stderr, "Couldn't create Plotter\n");
      return 1;
    }

  if (pl_openpl_r (plotter) < 0)      /* open Plotter */
    {
      fprintf (stderr, "Couldn't open Plotter\n");
      return 1;
    }
  pl_fspace_r (plotter, 0.0, 0.0, 1000.0, 1000.0); /* set coor system */
  pl_flinewidth_r (plotter, 0.25);    /* set line thickness */
  pl_pencolorname_r (plotter, "red"); /* use red pen */
  pl_erase_r (plotter);               /* erase graphics display */
  pl_fmove_r (plotter, 600.0, 300.0); /* position the graphics cursor */
  draw_c_curve (plotter, 0.0, 400.0, 0);
  if (pl_closepl_r (plotter) < 0)     /* close Plotter */
    {
      fprintf (stderr, "Couldn't close Plotter\n");
      return 1;
    }

  if (pl_deletepl_r (plotter) < 0)    /* delete Plotter */
    {
      fprintf (stderr, "Couldn't delete Plotter\n");
      return 1;
    }
  return 0;
}
```

見て分かる通り、プログラムの動作はPlotter のパラメータを保持する plPlotterParams を生成し、PAGESIZE パラメータを設定することから始まる。それから PostScript の Plotter を生成するための pl_newpl_r が呼ばれる。

この Plotter オブジェクトではページのサイズは US letter だが、たとえば "a4" などのほかの標準規格のサイズにしてもよい。それには pl_setplparam の呼び出しを変えればよい。

PAGESIZE パラメータなどのパラメータもこの関数で設定できる。設定できるパラメータのリストは Section 9.5 [Plotter Parameters], page 128 を参照のこと。

Plotter オブジェクトを生成したら、それをオープンしてから、再帰呼び出しを行う関数で “C” 曲線を描いている。曲線の描画は、pl_fmove_r を呼んでカーソルの位置を調整してから、draw_c_curve によって行われる。この関数は繰り返し pl_fcontrel_r を呼び出す。pl_fcontrel_r はそれまで描かれてきたパスに線分を追加するが、その線分の終端は、その直前のカーソル位置に対する相対座標で指定されている。“C” 曲線が描かれたら、Plotter をクローズする。pl_deletepl_r が呼ばれると、内部で自動的に pl_endpath_r が呼ばれパスの描画が完結させられる。それから pl_deletepl_r を呼ぶと PostScript 形式の画像ファイルを標準出力に書き出し、この Plotter オブジェクトは破棄される。

"ps" の代わりに"png"、"pnm"、"gif"、"svg"、"ai"、"cgm"、"fig"、"pcl"、"hpgl"、"regis"、"tek"、"meta" のいずれかを pl_newpl_r の最初の引数として指定すると、その指定された形式で Plotter は画像を標準出力に書き出す。PAGESIZE パラメータは"svg"、"ai"、"cgm"、"fig"、"pcl"、"hpgl" で有効であり、他の形式が指定されているときには無視される。出力された画像の形式の変換を繰り返したくない場合には、"meta" を指定しておくと良い。この形式のデータは、plot コマンドの標準入力に渡して、他の形式に変換したり X Window のディスプレイに表示できたりする。Chapter 3 [plot], page 27 参照。

pl_newpl_r の最初の引数に "X" を指定すると、X window のウィンドウが一つ開かれ、そこに曲線が描かれる。X window システムのどのディスプレイにウィンドウが出てくるかは、DISPLAY パラメータで指定される。それが設定されていないときは環境変数 DISPLAY で指定される。表示されるウィンドウのサイズは BITMAPSIZE パラメータで指定される。それが設定されていないときは環境変数 BITMAPSIZE で指定される。デフォルトのサイズは "570x570" である。BITMAPSIZE は "png"、"pnm"、"gif" のときも同様に指定できる。

Plotter の出力形式には "X drawable"も指定できるが、これを理解するには若干の X Window システム・プログラミングの知識が必要である。少なくとも一つの X drawable (ウィンドウまたはビットマップ画像) を生成し、pl_newpl_r の前に pl_setparampl を呼んでパラメータ XDRAWABLE_DRAWABLE1 または XDRAWABLE_DRAWABLE2 を設定し、描画をそのどちらにするかを指定する。X Drawable の Plotter に指定できるパラメータについては、Section 9.5 [Plotter Parameters], page 128 を参照のこと。

以下の C プログラムも libplot の機能を使ってベクトル形式の画像を生成する。この例では "GNU libplot!" という文字列を楕円で囲んで、それを螺旋状に置いていくものである。画像は標準出力に PostScript 形式で書き出される。

```
#include <stdio.h>
#include <plot.h>
#include <math.h>
#define SIZE 100.0   /* nominal size of user coordinate frame */
#define EXPAND 2.2   /* expansion factor for elliptical box */
```

```
void draw_boxed_string (plPlotter *plotter,
                        char *s, double size, double angle)
{
  double true_size, width;

  pl_ftextangle_r (plotter, angle);      /* set text angle (degrees) */
  true_size = pl_ffontsize_r (plotter, size);  /* set font size */
  width = pl_flabelwidth_r (plotter, s); /* compute width of string */
  pl_fellipserel_r (plotter, 0.0, 0.0,
                    EXPAND * 0.5 * width, EXPAND * 0.5 * true_size,
                    angle);              /* draw surrounding ellipse */
  pl_alabel_r (plotter, 'c', 'c', s);    /* draw centered text string */
}

int main()
{
  plPlotter *plotter;
  plPlotterParams *plotter_params;
  int i;

  /* set a Plotter parameter */
  plotter_params = pl_newplparams ();
  pl_setplparam (plotter_params, "PAGESIZE", "letter");

  /* create a Postscript Plotter that writes to standard output */
  if ((plotter = pl_newpl_r ("ps", stdin, stdout, stderr,
                             plotter_params)) == NULL)
    {
      fprintf (stderr, "Couldn't create Plotter\n");
      return 1;
    }

  if (pl_openpl_r (plotter) < 0)      /* open Plotter */
    {
      fprintf (stderr, "Couldn't open Plotter\n");
      return 1;
    }
  /* specify user coor system */
  pl_fspace_r (plotter, -(SIZE), -(SIZE), SIZE, SIZE);
  pl_pencolorname_r (plotter, "blue");     /* use blue pen */
  pl_fillcolorname_r (plotter, "white");   /* set white fill color */
  pl_filltype_r (plotter, 1);   /* fill ellipses with fill color */
  /* choose a Postscript font */
  pl_fontname_r (plotter, "NewCenturySchlbk-Roman");
```

```
  for (i = 80; i > 1; i--)      /* loop through angles */
    {
      double theta, radius;

      theta = 0.5 * (double)i;  /* theta is in radians */
      radius = SIZE / pow (theta, 0.35);  /* this yields a spiral */
      pl_fmove_r (plotter, radius * cos (theta), radius * sin (theta));
      draw_boxed_string (plotter, "GNU libplot!", 0.04 * radius,
                         (180.0 * theta / M_PI) - 90.0);
    }

  if (pl_closepl_r (plotter) < 0)        /* close Plotter */
    {
      fprintf (stderr, "Couldn't close Plotter\n");
      return 1;
    }
  if (pl_deletepl_r (plotter) < 0)       /* delete Plotter */
    {
      fprintf (stderr, "Couldn't delete Plotter\n");
      return 1;
    }
  return 0;
}
```

この例では文字列を描画に入れる方法を示している。まず最初に、使うフォントを指定する。フォントは pl_fontname_r、pl_fontsize_r、pl_textangle_r の 3 つが呼ばれて初めて指定できたことになる。数値を実数で指定したいときは pl_ffontname_r、pl_ffontsize_r、pl_ftextangle_r を呼ぶ。これらはどんな順序で呼んでもよい。呼び出し側の利便を図るため、これらの関数は設定しようとするフォントのサイズを返すようになっている。その値は pl_fontsize_r や pl_ffontsize_r の最後の呼び出しで指定された値と若干異なっていることがある。多くの Plotter では使えるフォントに制約があるためである。上の例では文字列のフォントは "NewCenturySchlbk-Roman" で、これは PostScript の Plotter で使えるフォントである。Section A.1 [Text Fonts], page 136 参照。

上の例で、pl_newpl_r の引数の "ps" を "X" に置き換えると、PostScript の Plotter の代わりに X の Plotter が生成され、文字列の渦巻きは X Window 画面上に開かれるウィンドウに描かれる。描画する X のディスプレイで "NewCenturySchlbk-Roman" フォントが使えないときは、広く使われているスケーラブルな "charter-medium-r-normal" などのコア X フォントや、場合によっては昔からある画面出力用の固定幅フォントで置き換えられる。このフォント名の意味するところについては Section A.3 [Text Fonts in X], page 143 を参照のこと。指定されたフォントを X の Plotter から利用できない場合、最初にまずデフォルトのスケーラブル・フォント ("Helvetica"。X Window でのフォント名は "helvetica-medium-r-normal") を使おうとする。それもダメだったときにはデフォルトの Hershey ベクトルフォント ("HersheySerif") を試す。Hershey フォントは線分の集合からできていて、どの種類の Plotter でも組み込みの Hershey フォントが使えるようになっている。

古い X Window システム (X11R6 より前) のディスプレイでは、スケーラブル・フォントを利用するときに長い実行時間を要することがある。そういった環境では、上の例を実行するとき、描画の前にまずフォントの準備を行うため、それに時間がかかるかもしれない。

この例では各文字列がそれぞれ違う角度で傾いて表示されているからである。X11R6 では、フォント全体を処理せず、各文字をそれぞれ独立に描画できる。その機能が使える場合には X の Plotter の内部で自動的にそれを利用し、実行時間の短縮を図っている。

9.2.5 単純パスと複合パス

libplot で実装している描画オブジェクトのうち、もっとも洗練されたものが「パス」である、と言えるだろう。ここではパスの構成法の詳細に付いて説明する。他の 3 種類のオブジェクト (文字列、記号、点 (つまりピクセル) については後述)。

PostScript の Plotter では単純パスと複合パスの両方が使える。単純パスは、線分、円弧、楕円弧、二次ベジェ曲線、三次ベジェ曲線から成る。複合パスは一つ、または複数の単純パスがネストしているものである。複合パスを構成している単純パス同士は交差してはならない。つまり *PostScript* よりも制限がきびしい。

libplot の描画方法は PostScript に比べると大きく違っていて、ユーザーフレンドリーである。libplot の各種の操作でパスを描くにあたって、その前に特になにかの関数を呼んでおく必要はないが、描くパスの属性を指定する関数を呼んでいてもよい。指定できる属性は、ペンの色、線種、線の太さ、線の端の形、線のつなぎ目の形、角の尖り具合、である。塗りつぶしが指定されたパスに対しては、塗りつぶしの色と塗りつぶし方も指定できる。これらの属性は「モード」としての性質を持っていて、つまり明示的に切り替えられない限りは、次のパスも同じ属性になる。

原理的には endpath 操作を行うことで、構築中のパスを完成させ、描画領域にそのパスを描くことができる。しかし、パスの属性を変更する操作、カーソル位置を移動する move 操作が行われるとき、または他の描画オブジェクトが構築、描画されるに先立って、自動的に endpath 操作が行われる。また closepl 操作でページが終了させられるときも自動的に endpath が呼ばれる。したがって通常は endpath を明示的に呼ぶ必要はない。PostScript ではパスを描画し、あるいは塗りつぶすためのコマンドを明示的に呼ぶ必要があり、libplot とは異なっている。

複合パスの構築法でも libplot と PostScript は異なっている。libplot では複合パスを構成する各単純パスを完結させるためには endsubpath 操作を呼び、各単純パスがすべて完成してから、全体を複合パスとして描画するために endpath 操作を呼び出す。endsubpath を呼んだ後は、次の単純パスの開始点に move 操作でカーソルを移動することができる。endsubpath の直後に move を呼んだ場合は、endpath が自動的に実行されることはない。

以下に示す簡単な例では、PostScript の Plotter を生成して PostScript 形式の出力を行う。このプログラムでは、17 の単純パスからなる複合パスを描く。最初の単純パスは大きな長方形である。この中に 7 つの円からなる同心円を描き、それとは別にもう一つ 7 つの円からなる同心円を描く。それぞれの同心円の中に、二つのつながった線分からなる単純パスを描く。複合パスは緑色で描かれ、塗りつぶされる。塗りつぶしの色は明るい青である。

```
#include <stdio.h>
#include <plot.h>

int main ()
{
  int i, j;
  plPlotter *plotter;
  plPlotterParams *plotter_params;
```

```
/* set a Plotter parameter */
plotter_params = pl_newplparams ();
pl_setplparam (plotter_params, "PAGESIZE", "letter");
/* create a Postscript Plotter that writes to standard output */
plotter = pl_newpl_r ("ps", stdin, stdout, stderr, plotter_params);
/* open Plotter, i.e. begin a page of graphics */
pl_openpl_r (plotter);

pl_fspace_r (plotter, 0.0, 0.0, 1000.0, 1000.0); /* set coor system */
pl_flinewidth_r (plotter, 5.0);  /* set line thickness */
pl_pencolorname_r (plotter, "green");
pl_fillcolorname_r (plotter, "light blue");
pl_filltype_r (plotter, 1);      /* do filling, full strength */
pl_erase_r (plotter);            /* erase graphics display */

/* draw a compound path consisting of 17 simple paths */

/* draw the first simple path: a large box */
pl_orientation_r (plotter, 1);
pl_fbox_r (plotter, 50.0, 50.0, 950.0, 950.0);
pl_endsubpath_r (plotter);
for (i = 0; i < 2; i++)
  /* draw 8 simple paths that are nested inside the box */
  {
    /* first, draw 7 simple paths: nested circles */
    for (j = 9; j >= 3; j--)
      {
        pl_orientation_r (plotter, j % 2 ? -1 : 1);
        pl_fcircle_r (plotter, 250.0 + 500 * i, 500.0, j * 20.0);
        pl_endsubpath_r (plotter);
      }
    /* draw an open simple path comprising two line segments */
    pl_fmove_r (plotter, 225.0 + 500 * i, 475.0);
    pl_fcont_r (plotter, 250.0 + 500 * i, 525.0);
    pl_fcont_r (plotter, 275.0 + 500 * i, 475.0);
    pl_endsubpath_r (plotter);
  }
/* formally end the compound path (not actually necessary) */
pl_endpath_r (plotter);

/* close Plotter, i.e. end page of graphics */
pl_closepl_r (plotter);
```

```
  /* delete Plotter */
  if (pl_deletepl_r (plotter) < 0)
    {
      fprintf (stderr, "Couldn't delete Plotter\n");
      return 1;
    }
  return 0;
}
```

見て分かるように、このプログラムを実行すると、同心円の間の塗りつぶしが交互に行われ、単純にべったりと塗りつぶすようにはならないようになっている。これは libplot のデフォルトの塗りつぶし規則が "even-odd" (偶奇塗りつぶし則) になっているからである。複合パスを構成する単純パスが包含関係 (ネスト) にある場合、どの部分が偶数番目で、どの部分が奇数番目かは簡便な規則で決められる。libplot では後者 (奇数番目) の領域が塗りつぶされる。

上のプログラムでは向きに関する orientation 操作が何度も行われる。'orientation' 属性はモード的な性質があり (1 は反時計回り、-1 は時計回りを意味する)、描かれる長方形、円、楕円に対して適用される。塗りつぶし規則が "even-odd" のときは、この属性は意味を持たない。しかし fillmod 操作を最初に呼ぶときに複合パスの塗りつぶし規則を"nonzero-winding" に設定すると、同心円の間が交互に塗りつぶされることは"even-odd" 設定のときと同様だが、具体的にどこが塗りつぶされるかは orientation の設定によって変わる。

上の説明がいったい何を言っているのかわからないときは、PostScript プログラミングのいい入門書か、'winding number' (巻き数) という言葉を調べるとよい。

9.2.6 実際のページの描画

libplot では通常、描画領域 ('viewport') に対して描画を行うが、物理的な用紙のページ全体に対しても描画を行うことができる。

Illustrator、PostScript、Fig、PCL の各 Plotter のデフォルトの viewport は、ページの中央に置かれた正方形の領域である。デフォルトの viewport の大きさは PAGESIZE パラメータによる。PAGESIZE は "letter" や "a4" などであるが、詳細については Appendix C [Page and Viewport Sizes], page 159 を参照のこと。たとえば用紙サイズがレターサイズ、つまり幅 8.5 in で高さ 11 in のときは、viewport は一辺が 8 in の正方形である。

しかし viewport のサイズや位置は他の値も指定できる。ページ全体を指定することもできる。これは PAGESIZE を、例えば"letter,xsize=8.5in,ysize=11in,xorigin=0in,yorigin=0in" と指定することでできる。"xorigin" と "yorigin" は viewport の左下隅の座標を用紙の左下隅からの相対値として指定する。

viewport をこのように指定すると、原理的には用紙全体に描画できることになる。この場合プログラム側で指定する座標を、そのまま用紙の左下隅を (0,0) とする座標にして、座標軸のスケールが実際のインチあるいは cm 単位にすると便利である。そのためには Plotter オブジェクトに対して適切な引数で space 操作を行う。以下にその例を示す。

```
#include <stdio.h>
#include <plot.h>

int main()
{
  plPlotter *plotter;
  plPlotterParams *plotter_params;
```

```
  /* set page size parameter, including viewport size and location */
  plotter_params = pl_newplparams ();
  pl_setplparam (plotter_params, "PAGESIZE",
                 "letter,xsize=8.5in,ysize=11in,xorigin=0in,yorigin=0in");

  /* create a Postscript Plotter with the specified parameter */
  plotter = pl_newpl_r ("ps", stdin, stdout, stderr, plotter_params);

  pl_openpl_r (plotter);                /* begin page of graphics */
  pl_fspace_r (plotter,
               0.0, 0.0, 8.5, 11.0);   /* set user coor system */

  pl_fontname_r (plotter, "Times-Bold");
  pl_ffontsize_r (plotter, 0.5);        /* font size = 0.5in = 36pt */

  pl_fmove_r (plotter, 1.0, 10.0);
  pl_alabel_r (plotter, 'l', 'x', "One inch below the top");
  pl_fline_r (plotter, 1.0, 10.0, 7.5, 10.0);

  pl_fmove_r (plotter, 7.5, 1.0);
  pl_alabel_r (plotter, 'r', 'x', "One inch above the bottom");
  pl_fline_r (plotter, 1.0, 1.0, 7.5, 1.0);

  pl_closepl_r (plotter);               /* end page of graphics */
  pl_deletepl_r (plotter);              /* delete Plotter */
  return 0;
}
```

このプログラムでは、下線のある文字列を二つ描く。最初の文字列は座標 (1.0, 10.0) に、つまり用紙の上端から 1 インチ下に左揃えで描く。二つ目は座標 (7.5, 1.0) に、つまり用紙の下端の 1 インチ上に右揃えで描く。どちらの場合も pl_alabel_r の 'x' 引数で縦方向の位置を指定する。この例では文字列のベースライン (文字の上端、下端ではなく) をその時点での縦座標の位置にすることを指示している。

用紙の方向はポートレート (縦長) がデフォルトであることは前述である。ランドスケープ (横長) での描画は少し複雑である。このためには、ROTATION パラメータを設定して viewport を用紙上で回転する必要がある。ROTATION のデフォルト値は "0" (または "no") だが、他にどんな角度を指定してもよい。用紙を横向きにするためにはこの値を "90" にする (過去の経緯から "yes" にしても "90" にしたのと同じことになる)。上のプログラムを修正して用紙を横置きにするプログラムを以下に示す。

```
#include <stdio.h>
#include <plot.h>

int main()
{
  plPlotter *plotter;
  plPlotterParams *plotter_params;
```

```
    /* set Plotter parameters */
    plotter_params = pl_newplparams ();
    pl_setplparam (plotter_params, "PAGESIZE",
                   "letter,xsize=8.5in,ysize=11in,xorigin=0in,yorigin=0in");
    pl_setplparam (plotter_params, "ROTATION", "90");

    /* create a Postscript Plotter with the specified parameters */
    plotter = pl_newpl_r ("ps", stdin, stdout, stderr, plotter_params);

    pl_openpl_r (plotter);                  /* begin page of graphics */
    pl_fspace_r (plotter,
                 0.0, 0.0, 11.0, 8.5);   /* set user coor system */

    pl_fontname_r (plotter, "Times-Bold");
    pl_ffontsize_r (plotter, 0.5);          /* font size = 0.5in = 36pt */

    pl_fmove_r (plotter, 1.0, 7.5);
    pl_alabel_r (plotter, 'l', 'x', "One inch below the top");
    pl_fline_r (plotter, 1.0, 7.5, 10.0, 7.5);

    pl_fmove_r (plotter, 10.0, 1.0);
    pl_alabel_r (plotter, 'r', 'x', "One inch above the bottom");
    pl_fline_r (plotter, 1.0, 1.0, 10.0, 1.0);

    pl_closepl_r (plotter);                 /* end page of graphics */
    pl_deletepl_r (plotter);                /* delete Plotter */
    return 0;
}
```

この例では viewport の大きさは一辺が 8 in の正方形のままで以前のプログラムと変わらないが、反時計回りに 90 度回転している。または、描画されるオブジェクトが回転されているとも言える。これの前の例と同様、`pl_fspace_r` 関数でプログラム側の座標のスケールを実際のインチに一致させているが、座標の原点は用紙の右下隅になっている。x 座標 と y 座標はそれぞれ、上方向と横方向に向かって増加する向きになっている。

9.2.7 アニメーション GIF

`libplot` を使って疑似アニメーション GIF 形式 (単なる疑似 GIF 形式の静止画像含む) の出力を得るのに、難しいことは何もない。GIF の Plotter は他の Plotter と変わるところは何もなく、同じ操作で描画が行える。しかし注意点が 2 点だけある。(1) 単ページの画像しかサポートされない。最初の `openpl`...`closepl` の間に描画されたものだけが出力される。これはリアルタイム・プロットを行わない他の Plotter も同じである。(2) その `openpl`...`closepl` の間で `erase` が呼ばれると、それまでの描画が終了して、次の画像の描画が新しく始まることになる。ただそれまでになにも描画が行われていないときには、なにもしない。

`libplot` を使ったプログラミングを行っているとき、Plotter をオープンしてすぐに `erase` を呼ぶ人は少なくない。X Drawable や Tektronix の Plotter では、描画を行うとずっとそれが残っているので、Plotter のオープン後すぐに `erase` を呼ぶのは悪いことではない。

以下に幅 150 ピクセル、縦 100 ピクセルの疑似アニメーション GIF を生成するプログラムを示す。

```
#include <stdio.h>
#include <plot.h>

int main()
{
  plPlotter *plotter;
  plPlotterParams *plotter_params;
  int i;

  /* set Plotter parameters */
  plotter_params = pl_newplparams ();
  pl_setplparam (plotter_params, "BITMAPSIZE", "150x100");
  pl_setplparam (plotter_params, "BG_COLOR", "orange");
  pl_setplparam (plotter_params, "TRANSPARENT_COLOR", "orange");
  pl_setplparam (plotter_params, "GIF_ITERATIONS", "100");
  pl_setplparam (plotter_params, "GIF_DELAY", "5");

  /* create a GIF Plotter with the specified parameters */
  plotter = pl_newpl_r ("gif", stdin, stdout, stderr, plotter_params);

  pl_openpl_r (plotter);                 /* begin page of graphics */
  pl_fspace_r (plotter,
               -0.5, -0.5, 149.5, 99.5); /* set user coor system */

  pl_pencolorname_r (plotter, "red");    /* use red pen */
  pl_linewidth_r (plotter, 5);           /* set line thickness */
  pl_filltype_r (plotter, 1);            /* objects will be filled */
  pl_fillcolorname_r (plotter, "black"); /* set the fill color */

  for (i = 0; i < 180 ; i += 15)
    {
      pl_erase_r (plotter);              /* begin new GIF image */
      pl_ellipse_r (plotter, 75, 50, 40, 20, i); /* draw an ellipse */
    }

  pl_closepl_r (plotter);                /* end page of graphics */
  pl_deletepl_r (plotter);               /* delete Plotter */
  return 0;
}
```

疑似アニメーション GIF の画像データは標準出力に書き出される。上のプログラムは 12 枚の静止画からなり、赤い縁取りの黒い楕円が反時計方向に回転する動画である。回転角度は 180 度だが、疑似 GIF のアニメーションは「ループ」するため (後述)、回転を続けるように見える。

楕円を定義するパラメータを指定する座標系はプログラム側から与える。したがって画像データ上のピクセル単位の座標ではないが、pl_fspace_r を呼ぶことでその両方の座標系を同一に設定できる。150 x 100 ピクセルの疑似 GIF 画像の左下隅の座標は、プログラム側で設定する座標系では (−0.5,−0.5)、右上隅は (149.5,99.5) である。つまりプログラム側の座標上で整数で点を指定すれば、その点がピクセルに対応する。たとえば、pl_point_r(plotter,0,0) および pl_point_r(plotter,149,99) とすると、左下隅と右上隅のピクセルの色をそのときのペンの色にする。

BITMAPSIZE と BG_COLOR の他にも重要なパラメータが疑似 GIF の Plotter にはある。それらを pl_setparampl 関数で設定している。まず、TRANSPARENT_COLOR には色名を設定できる。疑似 GIF 形式の画像上のその色のピクセルは、多くのソフトウェアで透明として扱われる。普通は画像の背景を透明にするのに使う。上のサンプル・プログラムでは背景色はオレンジに設定されているが、透明色もオレンジに設定されている。そのため画像の背景は実際には描かれない。

GIF_ITERATIONS パラメータで、複数のフレームからなる疑似 GIF 画像のループ回数を設定できる。GIF_DELAY パラメータには、1/100 秒単位でループの中の 1 枚の画像を表示する時間の長さを設定できる。

場合によっては INTERLACE が便利である。これを "yes" に設定すると、生成される疑似 GIF データはインターレース形式になる。これは静止画のときには非常に役に立つ指定である。その他パラメータについての詳細は、Section 9.5 [Plotter Parameters], page 128 を参照のこと。

9.2.8 C 言語での X Window システムでのアニメーション

libplot を使えば、リアルタイム・プロットを行う Plotter (X、X Drawable、ReGIS、Tektronix、Metafile) でベクトル形式の画像によるアニメーションを生成することができる。erase 操作を呼ぶことで、一つのページ中の「フレーム」を区切ることができる。そして各フレームごとに、各デバイスの描画領域内が消去されることになる。連続するフレーム間での違いを小さくすれば、滑らかなアニメーションになる。

以下に C 言語でのサンプル・プログラムを示す。このプログラムでは X Window システム上にアニメーションを表示する。これを実行すると「飛び去っていく目」が表示される。つまりウィンドウが新しく開き、その中を目玉が、ゆっくりと回転しながら左から右へと通り過ぎていく。目玉が 2 回通過したところで、ウィンドウは閉じられる。

```
#include <stdio.h>
#include <plot.h>

int main ()
{
  plPlotter *plotter;
  plPlotterParams *plotter_params;
  int i = 0, j;

  /* set Plotter parameters */
  plotter_params = pl_newplparams ();
  pl_setplparam (plotter_params, "BITMAPSIZE", "300x150");
  pl_setplparam (plotter_params, "VANISH_ON_DELETE", "yes");
  pl_setplparam (plotter_params, "USE_DOUBLE_BUFFERING", "yes");
```

```
    /* create an X Plotter with the specified parameters */
    if ((plotter = pl_newpl_r ("X", stdin, stdout, stderr,
                               plotter_params)) == NULL)
      {
        fprintf (stderr, "Couldn't create Plotter\n");
        return 1;
      }

    if (pl_openpl_r (plotter) < 0)         /* open Plotter */
      {
        fprintf (stderr, "Couldn't open Plotter\n");
        return 1;
      }
    pl_fspace_r (plotter,
                 -0.5, -0.5, 299.5, 149.5);  /* set user coor system */
    pl_linewidth_r (plotter, 8);           /* set line thickness */
    pl_filltype_r (plotter, 1);            /* objects will be filled */
    pl_bgcolorname_r (plotter, "saddle brown"); /* set background color */
    for (j = 0; j < 300; j++)
      {
        pl_erase_r (plotter);                  /* erase window */
        pl_pencolorname_r (plotter, "red");    /* use red pen */
        pl_fillcolorname_r (plotter, "cyan"); /* use cyan filling */
        pl_ellipse_r (plotter, i, 75, 35, 50, i);  /* draw an ellipse */
        pl_colorname_r (plotter, "black"); /* use black pen and filling */
        pl_circle_r (plotter, i, 75, 12);  /* draw a circle [the pupil] */
        i = (i + 2) % 300;                 /* shift rightwards */
      }
    if (pl_closepl_r (plotter) < 0)        /* close Plotter */
      {
        fprintf (stderr, "Couldn't close Plotter\n");
        return 1;
      }

    if (pl_deletepl_r (plotter) < 0)       /* delete Plotter */
      {
        fprintf (stderr, "Couldn't delete Plotter\n");
        return 1;
      }
    return 0;
  }
```

見て分かるように、プログラムの最初に出力デバイスのパラメータを設定する pl_setplparam が設定するパラメータの数だけ呼ばれ、それから pl_newpl_r を呼んで X の Plotter を生成している。X の Plotter のウィンドウのサイズは 300 x 150 ピクセルである。Plotter がクローズ／破棄されるときに、ウィンドウも閉じられる。VANISH_ON_DELETE

パラメータが "yes" に設定されていないときは、ウィンドウは自動的には消えず、ウィンドウ内で 'q' を押すか、ウィンドウをマウスでクリックしたときに消えるようになる。

USE_DOUBLE_BUFFERING パラメータを "yes" にすると、二重バッファリングが行われるようになる。アニメーションがコマ送りみたいにならず、スムーズに動くようにするためには重要な機能である。通常は X の Plotter はリアルタイムで画面に描画を行い、pl_erase_r が呼ばれたときに画面をクリアするが、二重バッファリングが有効になっているときは、各フレームの内容は一旦バッファに書き出され、 pl_erase_r が呼ばれたとき、またはその Plotter がクローズされたときに、ピクセル単位でウィンドウにコピーされることで描画が行われる。滑らかなアニメーションを描くためには必須の機能である。

生成された Plotter を使うためには、それを選択し、オープンしなければならない。pl_openpl_r を呼ぶとウィンドウが新しく開かれ、アニメーションの表示が始まる。for ループの中には pl_erase_r と、目玉を描く一連の libplot の描画操作がある。ループ内で線を描くときにそれぞれ 1 回ずつ、線の色と塗りつぶしの色の指定が行われる。実際に表示を行う計算機環境で、パラメータ値をどう設定するのが良いのか、試行錯誤するといいかもしれない。

このアニメーションの中で図形を描くときの座標系は、プログラム側で指定する座標系である。したがって画像データ上のピクセル単位の座標ではないが、pl_fspace_r を呼ぶことでその両方の座標系を同一に設定できる。プログラム側で設定する座標系では $(-0.5,-0.5)$、右上隅は (299.5,149.5) である。つまりプログラム側の座標上で整数で点を指定すれば、その点がピクセルに対応する。たとえば、pl_point_r(plotter,299,149) とすると、右上隅のピクセルの色をそのときのペンの色にする。

次のプログラムでは、文字 'A' が回転するアニメーションを生成する。

```
#include <stdio.h>
#include <plot.h>

int main()
{
  plPlotter *plotter;
  plPlotterParams *plotter_params;
  int angle = 0;

  /* set Plotter parameters */
  plotter_params = pl_newplparams ();
  pl_setplparam (plotter_params, "BITMAPSIZE", "300x300");
  pl_setplparam (plotter_params, "USE_DOUBLE_BUFFERING", "yes");
  pl_setplparam (plotter_params, "BG_COLOR", "blue");

  /* create an X Plotter with the specified parameters */
  plotter = pl_newpl_r ("X", stdin, stdout, stderr, plotter_params);

  /* open X Plotter, initialize coordinates, pen, and font */
  pl_openpl_r (plotter);
  pl_fspace_r (plotter, 0.0, 0.0, 1.0, 1.0);  /* use normalized coors */
  pl_pencolorname_r (plotter, "white");
  pl_ffontsize_r (plotter, 1.0);
  pl_fontname_r (plotter, "NewCenturySchlbk-Roman");
```

```
  pl_fmove_r (plotter, 0.5, 0.5);        /* move to center */
  while (1)                              /* loop endlessly */
    {
      pl_erase_r (plotter);
      pl_textangle_r (plotter, angle++); /* set new rotation angle */
      pl_alabel_r (plotter, 'c', 'c', "A"); /* draw a centered 'A' */
    }
  pl_closepl_r (plotter);                /* close Plotter */

  pl_deletepl_r (plotter);               /* delete Plotter */
  return 0;
}
```

この例では X Window システムの機能がよく示されている。最近の X11R6 のディスプレイでは、アニメーションは高速で滑らかである。これは X11R6 では、フォントから使いたい文字だけを取り出してラスタライズする機能があるからである。描画する X のディスプレイで "NewCenturySchlbk-Roman" フォントが使えないときは、広く使われているスケーラブルな"charter-medium-r-normal" などのコア X フォントや、場合によっては昔からある画面出力用の固定幅フォントで置き換えられる。このフォント名の意味するところについては Section A.3 [Text Fonts in X], page 143 を参照のこと。

指定されたフォントを X の Plotter から利用できない場合、最初にまずデフォルトのスケーラブル・フォント ("Helvetica"。X Window でのフォント名は "helvetica-medium-r-normal") を使おうとする。それもダメだったときにはデフォルトの Hershey ベクトルフォント ("HersheySerif") を試す。

Hershey フォントは線分の集合からできていて、PostScript フォントを使うよりもこれを使った方がアニメーションの描画は高速である。線分の描画はフォントのラスタライズよりも高速に行われるからである。

X の Plotter で長々とつづく描画操作が続くアニメーションを行うときは、Plotter パラメータの `X_AUTO_FLUSH` を "no" に設定するとよい。デフォルトでは X の Plotter は各描画操作が行われるたびに、それを X Window システムのウィンドウ上に反映する。つまりプログラム内で描画操作が行われたらすぐにそれが目に見えるようになるが、場合によってはそれによって描画速度が低下することがある。詳細については Section 9.5 [Plotter Parameters], page 128 を参照のこと。

9.2.9 一歩進んだ X Window システム・プログラミング

X Window システムを利用するプログラムは通常 Xt、X Toolkit ライブラリを使って書かれる。Xt では各アプリケーションは、たとえばテキスト入力フィールド、ボタン、スライダー、描画できる領域などの「ウィジェット」と呼ばれる部品を使って構築される。アプリケーションの実行が開始される時、各ウィジェットはそれぞれに対応する「イベント」(キー入力やマウスクリックなど) に反応するように設定される。その設定が終わってから Xt のイベントループに制御が渡される。

GNU `libplot` は Xt のイベントループ内でベクトル形式画像を描画するのに使うことができる。そのためには一つ、あるいは複数の X Drawable の Plotter を生成して使う。一つの X Drawable の Plotter は画面に表示されない一つのビットマップ画像に、または一つのウィジェットとして管理される画面上の一つのウィンドウに、描画することができる。

以下に一つの X Drawable の Plotter を使うサンプル・プログラムを示す。前の節でも使った 'C' 曲線を、ウィンドウを一つ生成してそこに描くアプリケーションである。通常の Xt アプリケーションに使えるコマンドライン・オプションがこれにも使える。ウィンドウの背景色の '-bg' オプション、ウィンドウの位置の '-geometry' オプションなどである。曲線は最初は赤で、マウスで1回クリックすると緑で描かれる。もう1回クリックすると赤に戻り、以下同様である。'q' をキー入力すると終了する。

```
#include <stdio.h>
#include <stdlib.h>
#include <plot.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <X11/StringDefs.h>
#include <X11/Core.h>

plPlotter *plotter;
int green = 0;                  /* draw in green, not red? */

#define MAXORDER 12
void draw_c_curve (double dx, double dy, int order)
{
  if (order >= MAXORDER)
    /* continue path along (dx, dy) */
    pl_fcontrel_r (plotter, dx, dy);
  else
    {
      draw_c_curve (0.5 * (dx - dy), 0.5 * (dx + dy), order + 1);
      draw_c_curve (0.5 * (dx + dy), 0.5 * (dy - dx), order + 1);
    }
}

void Redraw (Widget w, XEvent *ev, String *params, Cardinal *n_params)
{
  /* draw C curve */
  pl_erase_r (plotter);
  pl_pencolorname_r (plotter, green ? "green" : "red");
  pl_fmove_r (plotter, 600.0, 300.0);
  draw_c_curve (0.0, 400.0, 0);
  pl_endpath_r (plotter);
}

void Toggle (Widget w, XEvent *ev, String *params, Cardinal *n_params)
{
  green = (green ? 0 : 1);
  Redraw (w, ev, params, n_params);
}
```

```
void Quit (Widget w, XEvent *ev, String *params, Cardinal *n_params)
{
  exit (0);
}

/* mapping of events to actions */
static const String translations =
"<Expose>:      redraw()\n\
<Btn1Down>:     toggle()\n\
<Key>q:         quit()";

/* mapping of actions to subroutines */
static XtActionsRec actions[] =
{
  {"redraw",            Redraw},
  {"toggle",            Toggle},
  {"quit",              Quit},
};

/* default parameters for widgets */
static String default_resources[] =
{
  "Example*geometry:      250x250",
  (String)NULL
};

int main (int argc, char *argv[])
{
  plPlotterParams *plotter_params;
  Arg wargs[10];                /* storage of widget args */
  Display *display;             /* X display */
  Widget shell, canvas;         /* toplevel widget; child */
  Window window;                /* child widget's window */
  XtAppContext app_con;         /* application context */
  int i;
  char *bg_colorname = "white";

  /* take background color from command line */
  for (i = 0; i < argc - 1; i++)
    if (strcmp (argv[i], "-bg") == 0)
      bg_colorname = argv[i + 1];
```

```
    /* create toplevel shell widget */
    shell = XtAppInitialize (&app_con,
                             (String)"Example", /* app class */
                             NULL,              /* options */
                             (Cardinal)0,       /* num of options */
                             &argc,             /* command line */
                             argv,              /* command line */
                             default_resources,
                             NULL,              /* ArgList */
                             (Cardinal)0        /* num of Args */
                             );
    /* set default widget parameters (including window size) */
    XtAppSetFallbackResources (app_con, default_resources);
    /* map actions to subroutines */
    XtAppAddActions (app_con, actions, XtNumber (actions));
    /* create canvas widget as child of shell widget; realize both */
    XtSetArg(wargs[0], XtNargc, argc);
    XtSetArg(wargs[1], XtNargv, argv);
    canvas = XtCreateManagedWidget ((String)"", coreWidgetClass,
                                    shell, wargs, (Cardinal)2);
    XtRealizeWidget (shell);
    /* for the canvas widget, map events to actions */
    XtSetArg (wargs[0], XtNtranslations,
              XtParseTranslationTable (translations));
    XtSetValues (canvas, wargs, (Cardinal)1);

    /* initialize GNU libplot */
    plotter_params = pl_newplparams ();
    display = XtDisplay (canvas);
    window = XtWindow (canvas);
    pl_setplparam (plotter_params, "XDRAWABLE_DISPLAY", display);
    pl_setplparam (plotter_params, "XDRAWABLE_DRAWABLE1", &window);
    pl_setplparam (plotter_params, "BG_COLOR", bg_colorname);
    plotter = pl_newpl_r ("Xdrawable", NULL, NULL, stderr,
                          plotter_params);
    pl_openpl_r (plotter);
    pl_fspace_r (plotter, 0.0, 0.0, 1000.0, 1000.0);
    pl_flinewidth_r (plotter, 0.25);

    /* transfer control to X Toolkit event loop (doesn't return) */
    XtAppMainLoop (app_con);

    return 1;
}
```

X Window システムのプログラミングに詳しくなくても、このサンプル・プログラムの大まかな構造は把握できるだろう。このプログラムには３つのコールバック、Redraw、Toggle、

Quit が定義されている。それぞれ (1) ウィンドウがアクティブになった、またはマウスクリックがあった、(2) マウスクリック、(3) 'q' がキー入力された、というイベントがあったときに呼ばれる。最初に行われる 'C' 曲線 (赤い線) の描画は、生成後最初にアクティブになったときに、生成されたウィンドウにそのイベントが通知されることにより実行される。

このサンプル・プログラムは、ウィンドウのリサイズに対応するようにも簡単に書き換えられる。しかし実際には X Drawable の Plotter は画面に直接ではなく、ビットマップ画像として描画するときに用いるのが普通である。ビットマップ画像はウィンドウと違って、リサイズされない。

9.3 libplotter を使った C++ プログラミング

9.3.1 Plotter クラス

libplot の C++ バインディングは libplotter というクラス・ライブラリとして提供されている。このライブラリでは、Plotter をオブジェクトとして実装している Plotter クラスが利用できる。実際には各 Plotter は、出力形式ごとに定義される派生クラスのインスタンスとなる。現在実装されている派生クラスは XPlotter、XDrawablePlotter、PNGPlotter、PNMPlotter、GIFPlotter、AIPlotter、PSPlotter、CGMPlotter、FigPlotter、PCLPlotter、HPGLPlotter、ReGISPlotter、TekPlotter、MetaPlotter である。それぞれの出力形式は名前から分かる通りである。Plotter インスタンスに対する操作 (たとえば新しいページを作る openpl など) は Plotter クラスの public なメンバー関数として実装されている。

Plotter インスタンスを生成するときには、入力元、出力先、エラーの出力先をそれぞれパラメータとして指定しなければならない (現在の実装では、入力元は無視される。Plotter クラスはすべて書き出すのみで入力を受け付けないからである。他のパラメータも指定できるが、それらは省略してもよい)。出力先、エラーの出力先はそれぞれ、iostreams でも FILE ポインタでも指定でき、それぞれの場合について、以下のコンストラクタがある。

```
Plotter(istream& instream, ostream& outstream, ostream& errstream,
        PlotterParams &params);
Plotter(FILE *infile, FILE *outfile, FILE *errfile,
        PlotterParams &params);
```

これらはそれぞれ Plotter 基本クラス、および各派生クラスに用意されている。例えば

```
PSPlotter plotter(cin, cout, cerr, params);
```

と次の

```
PSPlotter plotter(stdin, stdout, stderr, params);
```

はどちらも、標準出力に画像データを出力する PostScript インスタンスの宣言となる。iostream を使った場合は、ストリーム・バッファが null の ostream を出力先あるいはエラーの出力先に指定すると、その出力が抑制される。FILE ポインタを使った場合は、FILE に null を指定すると同じことになる。XPlotter および XDrawablePlotter クラスのインスタンスは X のディスプレイに描画を行うため、コンストラクタの出力先を指定する引数は単に無視される。

PlotterParams クラスは複製や代入をサポートしているが、その public なメンバー関数は setplparam だけである。その仕様は以下のようになる。

int PlotterParams::setplparam (const char *_parameter_, void *_value_);
: その Plotter のパラメータ *parameter* の値を *value* に設定する。ほとんどのパラメータの値 *value* は char *、つまり文字列である。無効な値を指定した場合

は、無視される。Section 9.5 [Plotter Parameters], page 128 に指定できるパラメータ値のリストを挙げてある。

C バインディングの plPlotterParams 型と pl_setplparam 関数と同様、C++ バインディングでは PlotterParams クラスと PlotterParams::setplparam を使ってパラメータ値を設定して、その後に生成される Plotter インスタンスを制御することができる。Plotter が参照するパラメータ値は、その Plotter インスタンスが破棄されるまで定数として扱われる。Plotter の生成時に PlotterParams オブジェクト内にパラメータが設定されていない場合はデフォルト値が使われる。しかしパラメータがその値として文字列を取るもので、同名の環境変数が設定されている場合は、その環境変数の値がパラメータとして使われる。

PlotterParams::setplparam の引数 *value* に NULL を渡すことで、PlotterParams オブジェクト内に一度設定されたパラメータの設定を解除できる。これによりプログラミングの自由度を高くしている。

以下に、古いやり方での Plotter の生成方法を示す。この方法は現在もサポートはされているが推奨されない。

```
Plotter(istream& instream, ostream& outstream, ostream& errstream);
Plotter(FILE *infile, FILE *outfile, FILE *errfile);
```

上のどちらでも、PlotterParams オブジェクトを指定せずに Plotter オブジェクトを生成できる。この場合、生成される Plotter のパラメータ値は、静的に確保されている領域からコピーされる。その領域は Plotter クラスのメンバー関数 Plotter::parampl を呼ぶことで設定できる。

int PlotterParams::parampl (const char **parameter*, void **value*);

この方法はスレッド・セーフではないので、推奨されない。

9.3.2 C++ でのコンパイルとリンク

C++ で libplotter を使うプログラムを書くとき、そのソースファイルには以下の行が必要である。

```
#include <plotter.h>
```

ヘッダファイル plotter.h は libplotter の配布パッケージの一部として含まれており、C++ コンパイラが見つけられる場所に置いておかねばならない。このファイル中では Plotter クラスとその派生クラスが宣言されており、その他の色々な宣言も行われている。またこの中から <iostream.h> と<stdio.h> が include されているので、別途それらの include 文を書く必要はない。

作成したプログラムを libplotter とリンクするには、コンパイル時にコマンドライン・オプションに '-l' が必要になる。たとえば以下のようにする。

```
-lplotter -lXaw -lXmu -lXt -lXext -lX11 -lpng -lz -lm
```

最近の X Windows システムでは以下のようになるだろう。

```
-lplotter -lXaw -lXmu -lXt -lSM -lICE -lXext -lX11 -lpng -lz -lm
```

上の例では、PNG サポートを有効にしてコンパイルした libplot を使うことを前提にしている。もしそうでないなら、'-lpng -lz' オプションを指定しないようにする。

(Athena ウィジェットの代わりに Motif ウィジェットを使うシステムでは '-lplotter -lXm -lXt -lXext -lX11 -lpng -lz -lm'、'-lplotter -lXm -lXt -lXext -lX11 -lpng -lz -lm -lc -lgen'、'-lplotter -lXm -lXt -lXext -lX11 -lpng -lz -lm -lc -lPW' のいずれかになるだろう。また最近の X Window システムではオプションに '-lSM -lICE' を追加する必

要があるかもしれない。さらに Motif の最近のバージョンでは '-lXp' や '-lXpm' も必要かもしれない。)

場合によっては libplotter やその他のライブラリのインストールされているディレクトリを、コマンドラインで指定する必要がある。たとえば X Window のライブラリを指定するオプション '-lXaw -lXmu -lXt -lSM -lICE -lXext -lX11' を使うには '-L/usr/X11/lib' といったオプションを同時に指定せねばならないかもしれない。

多くの場合 libplotter は共有ライブラリ (shared library) としてインストールされる。この場合、リンクはコンパイル時には行われず、プログラムの実行時に行われることになる。その際ライブラリのある場所は環境変数 LD_LIBRARY_PATH によって示される。libplotter を使うプログラムが正常に実行されるためには、そのライブラリのあるディレクトリを LD_LIBRARY_PATH に設定しておかねばならない。

9.3.3 C++ でのサンプル・プログラム

前章では libplot の C バインディングでベクトル形式の描画を行うサンプルの C プログラムをいくつか示した (Section 9.2.4 [Sample C Drawings], page 91 参照) が、ここではそれらの C プログラムを、libplot の C++ バインディングである libplotter を使うように修正したプログラムを例に挙げる。実行方法や得られる出力は、C のものと同じである。

以下の C++ プログラムは、美しく絡まる Bill Gosper の "C" 曲線を描く。

```
#include <plotter.h>
const int maxorder = 12;

void draw_c_curve (Plotter& plotter, double dx, double dy, int order)
{
  if (order >= maxorder)
    plotter.fcontrel (dx, dy); // continue path along (dx, dy)
  else
    {
      draw_c_curve (plotter,
                    0.5 * (dx - dy), 0.5 * (dx + dy), order + 1);
      draw_c_curve (plotter,
                    0.5 * (dx + dy), 0.5 * (dy - dx), order + 1);
    }
}

int main ()
{
  // set a Plotter parameter
  PlotterParams params;
  params.setplparam ("PAGESIZE", (char *)"letter");

  PSPlotter plotter(cin, cout, cerr, params); // declare Plotter
  if (plotter.openpl () < 0)                  // open Plotter
    {
      cerr << "Couldn't open Plotter\n";
      return 1;
    }
```

```
    plotter.fspace (0.0, 0.0, 1000.0, 1000.0); // specify user coor system
    plotter.flinewidth (0.25);       // line thickness in user coordinates
    plotter.pencolorname ("red");    // path will be drawn in red
    plotter.erase ();                // erase Plotter's graphics display
    plotter.fmove (600.0, 300.0);    // position the graphics cursor
    draw_c_curve (plotter, 0.0, 400.0, 0);
    if (plotter.closepl () < 0)      // close Plotter
      {
        cerr << "Couldn't close Plotter\n";
        return 1;
      }
    return 0;
  }
```

上のプログラムは、前章の C プログラムを単純に書き直しただけである。ここでは PSPlotter クラスのインスタンスとして plotter を宣言している。このインスタンスの出力先は cout に設定されている。そのメンバー関数を呼び出すことで、描画操作を行う。

9.4 libplot の関数リスト

GNU libplot は現在、97 種類の Plotter 操作を実装している。libplot の各言語バインディングではその各操作に対応する 97 個の関数を実装している。C バインディングでは、その 97 個の関数は C の API (Application Programming Interface) である。その各関数は関数名の先頭に "pl_" が、末尾に "_r" が付けられている。C++ バインディングでは、それらは Plotter クラスの public なメンバー関数だが、関数名の先頭や末尾には特に何も付けられていない。

言語バインディングには Plotter オブジェクトの生成、選択、破棄のための関数も含まれる。たとえば C バインディングでは生成と破棄はそれぞれ pl_newpl_r、pl_deletepl_r である。Section 9.2.1 [The C API], page 88 参照。

指定する Plotter に対して様々な操作を行うこの 97 の関数は、以下の 4 種類に分けられる。

1. 制御関数: Plotter オブジェクトをオープン、初期化、破棄する関数。
2. Plotter に図形の描画を行わせる関数。
3. Plotter の描画の仕方を設定あるいは制御する関数。
4. プログラム側で設定する座標系から Plotter 内部のデバイス上の座標系に変換を行うアフィン変換を制御する関数。

多くの関数は、整数版と倍精度実数版の二種類が用意されている。libplot は内部では倍精度実数を使っている。整数版は後方互換性を保つために用意されているものである。この二種類が用意されている関数では、実数版の関数名の先頭に 'f' という文字が付けられている。

また多くの関数に、絶対値版と相対値版が用意されている。後者は相対座標 (カーソルの現在位置からの相対座標値) を使い、関数名の末尾に 'rel' が付けられている。

現在、97 種の関数のうち、返り値に意味のある関数は少ない。

9.4.1 制御関数

以下に挙げるのは libplot の「制御関数」である。これらは、すでに生成されている Plotter に対してオープン、初期化、破棄といった操作を行う。以下に、おおよそ実際に使われるときの順番で列挙する。

C バインディングでは、各関数は plPlotter 型へのポインタを最初の引数にとり、関数名の先頭には "pl_" が、末尾には"_r" がついている ("r" は revised または reentrant という意味である)。古い C バインディングについては Section 9.2.2 [Older C APIs], page 89 を参照のこと。C++ バインディングでは、これらは Plotter クラスと派生クラスの public なメンバー関数である。関数名の先頭や末尾には特になにも付けられていない。

int openpl ();
: Plotter をオープンし、画像のページを新しくはじめる。この時 Plotter の描画属性はデフォルト値に設定される。Plotter がオープンできなかったときは負の値を返す。

  X の Plotter では openpl で新しくページが開始すると X Window のディスプレイ上に新しくウィンドウが開かれる。将来的には一つのウィンドウ上に次の描画を行うようになる予定である。

int bgcolor (int *red*, int *green*, int *blue*);
: Plotter の描画領域の背景色を、48 ビット RGB で指定される色に設定する。引数の *red*、*green*、*blue* にそれぞれ背景色での赤、緑、青の濃さを指定する。濃さの指定は 0x0000...0xffff、つまり 0...65535 の範囲内の整数で行う。(0, 0, 0) を指定すると黒に、(65535, 65535, 65535) を指定すると白になる。

  bgcolor は、出力形式がビットマップ画像の Plotter、つまり X、X Drawable、PNG、PNM、GIF の各 Plotter と、CGM、ReGIS、Metafile の各 Plotter でだけ有効である。この設定の効果は単純なもので、この設定の後に呼ばれる erase 操作で、描画領域全体がこの色で塗りつぶされるだけである。

int bgcolorname (const char **name*);
: Plotter の描画領域の背景色を、*name* に設定する。無効な色名を指定したときは "white" に置き換えられる。設定できる色名については Appendix B [Color Names], page 158 を参照のこと。"#c0c0c0" のようにして 6 桁の 16 進数で 24 ビット RGB カラーを指定することもできる。

  bgcolor は、出力形式がビットマップ画像の Plotter、つまり X、X Drawable、PNG、PNM、GIF の各 Plotter と、CGM、ReGIS、Metafile の各 Plotter でだけ有効であり、その効果は単純なもので、これを設定した後に erase 操作を行うと、描画領域全体がこの色で塗りつぶされるということだけである。

  SVG、CGM の各 Plotter では背景色に "none" を指定できる。それにより背景を無効にすることができる。Web ページに表示する画像を生成したいときに使うとよい。

int erase ();
: 複数フレームを含むページにおいて、それまでに描画領域に描かれている図形をすべて消去し、背景色が設定されていればそれで全体を塗りつぶし、新しいフレームを開始する。

  openpl によって新しくページをはじめた直後に erase を呼ぶと便利なことがよくある。Plotter の種類によっては openpl...closepl の間に描かれた画像

がその後に openpl を呼んで新しくページを始めても残っているからである。現在、X Drawable と Tektronix がそういった Plotter である。将来的には、X の Plotter では画像が残るかどうかを選択できるようにする予定である。

X と X Drawable の Plotter ではデバイスドライバのパラメータ USE_DOUBLE_BUFFERING が "yes" に設定されていると、 erase の挙動が変わってくる。その場合、図形は描画領域ではなくバッファに描かれるため、erase は (1) バッファの内容をディスプレイにコピーし、(2) バッファを背景色で塗りつぶす、という作業を行う。この「二重バッファリング」によりアニメーションを滑らかにすることができる。Section 9.5 [Plotter Parameters], page 128 参照。

int space (int *x0*, int *y0*, int *x1*, int *y1*);
int fspace (double *x0*, double *y0*, double *x1*, double *y1*);

描画領域の左下隅と右上隅の、プログラム側で想定する座標系上での座標値を、二組の数値で設定する関数である。これによりプログラム側の座標値からデバイスの描画領域 (viewport) 上の座標値への変換が設定される。デフォルトではプログラム側座標系は正方形で、左下隅と右上隅はそれぞれ (0,0) と (1,1) である。

数学的に表現すると、space または fspace を呼び出すことで、プログラム側の座標系からデバイスドライバの座標系 (viewport) へのアフィン変換が設定される。これにより、各図形について、それが描画される前に適用される変換が設定される。openpl を呼んで新しくページを開始したら、すぐに space か fspace のどちらかの関数を呼ぶ。space や fspace は何度呼んでもよい。また「変換関数」は他にもある。Section 9.4.4 [Mapping Functions], page 127 を参照のこと。

viewport の大きさや位置は Plotter の種類や、Plotter が生成されるときに設定されているパラメータ値によって変わる。たとえばIllustrator、PostScript、Fig、PCL、HP-GL の各 Plotter では正方形だが、その大きさは設定されている用紙によって変わる。詳細については Appendix C [Page and Viewport Sizes], page 159 を参照のこと。

int space2 (int *x0*, int *y0*, int *x1*, int *y1*, int *x2*, int *y2*);
int fspace2 (double *x0*, double *y0*, double *x1*, double *y1*, double *x2*, double *y2*);

これらは space と fspace の拡張版である。引数で、「アフィン窓」(平行四辺形を描く) を定義する 3 つのプログラム側の座標系上の座標を指定する。3 つの座標はそれぞれ、左下隅、右下隅、右上隅である。ウィンドウはアフィン変換で描画領域、つまり図形が実際に描画される出力デバイス上の長方形領域にマッピングされる。

int havecap (const char *s);

Plotter が指定された機能を使えるかどうかを調べる。その Plotter はオープンされている必要はない。返り値は 0, 1, 2 のいずれかであり、それぞれ no/yes/maybe を意味する。指摘できない機能が指定されたときには 0 を返す。調べられる機能は"WIDE_LINES" (デフォルト以外の太さの線が描けるか)、"DASH_ARRAY" (linedash 機能を使って任意の細かさの破線が描けるか)、"SETTABLE_BACKGROUND" (背景色をカラーで設定できるか)、"SOLID_FILL" などである。"HERSHEY_FONTS"、"PS_FONTS"、"PCL_FONTS"、"STICK_FONTS" でそれぞれの形式のフォントが使えるかどうかが分かる。Section A.1 [Text Fonts], page 136 参照。

Tektronix 以外の Plotter では "SOLID_FILL" 機能により、単色で塗りつぶしができる。これができる Plotter では少なくとも "EVEN_ODD_FILL" か

"NONZERO_WINDING_NUMBER_FILL" のどちらかが使える。これにより、「曲線の内側」をどう決定するかが選べる。

Metafile の Plotter では、ほとんどの機能について返り値が 'maybe' になる。その Plotter 自身では描画を行わないからである。Metafile の Plotter の出力は `plot` を使って他の形式に変換するか、画面に表示するかせねばならない。

int `flushpl` ();
出力デバイスに、すべての描画命令を書き出す。これはリアルタイムの Plotter (X、X Drawable、ReGIS、Tektronix、Metafile の各 Plotter) でのみ意味のある操作であり、それまでに描画された図形をすべてディスプレイに描き、画面上で見えるようにするための操作である。リアルタイムでない Plotter に対してこの操作を行っても、なにも行われない。

int `closepl` ();
Plotter をクローズして、ページの描画を終了する。構築中のパスがあったときには `endpath` を呼ぶのと同じように、まずそのパスの構築が完了させられる。Plotter がクローズできなかったときには、非零の値を返す。

`libplot` の現在の実装では、`closepl` が呼ばれたときにリアルタイムでページを描画する Plotter がいくつかある。たとえば PCL と HP-GL の Plotter がそうである。同様に、単ページしか描画できない Plotter では、`closepl` が呼ばれたときにすぐにページを終了し、描画データを出力する (PNG、PNM、GIF、SVG、Illustrator、Fig の各 Plotter)。しかし PostScript や CGM の Plotter では、その Plotter が破棄されるときにすべてのページが出力される。

9.4.2 図形関数

以下に `libplot` の「図形描画関数」を説明する。これらを Plotter に対して実行した時、その描画領域に図形オブジェクト (パス、文字列、記号、点つまりピクセル) が描かれる。

パスには単純パスと複合パスがある。単純パスは、線分、円弧 (真円および楕円の円弧)、二次または三次のベジェ曲線が連結されたものである。単純パスはこれらの要素を一つずつ付け加えていくことで描かれる。単純パスは円、長方形、楕円にもなる。複合パスは複数の単純パスが入れ子 (ネスト) になることで構成される。

パスを描き始めるための命令は特に用意されていない。またすくなくとも機能としては、`endpath` 操作を呼ぶことで構築中のパスが完結し、描画領域に描かれる。しかし `endpath` は他の図形を描いたりページを終了したりするときに自動的に呼ばれる。またパスの属性を変更するときにも自動的に呼ばれる。たとえばカーソル位置を移動するための `move` を呼んだときなどである。そのため `endpath` を明示的に呼ぶ機会は決して多くはない。

複合パスを描くときは、それを構成する単純パスの構築を `endsubpath` を呼ぶことで完結し、一つの複合パスの構築を `endpath` で完結する。`endsubpath` を呼んだ後は、次の単純パスの開始点に `move` 操作でカーソルと移動することができる。`endsubpath` の直後に `move` を呼んだ場合は、`endpath` が自動的に実行されることはない。したがって、複合パスの構築中は上述の自動呼び出しが行われないことになる。

現在の C バインディングでは、これらの各関数の第一引数は `plPlotter` へのポインタである。また各関数の先頭には "pl_" が、末尾には"_r" が付いている ("_r" は 'revised' または 'reentrant' の意味である)。古い C バインディングについては Section 9.2.2 [Older C APIs], page 89 を参照のこと。C++ バインディングでは、これらは `Plotter` クラスとその派生クラスのメンバー関数であり、先頭にも末尾にも特になにもついていない。

int alabel (int *horiz_justify*, int *vert_justify*, const char *s);

３つの引数 *horiz_justify*、*vert_justify*、*s* で「整列文字列」を指定する。構築中のパスは (もしあれば) 完了させられ、文字列 *s* が指定された方法で整列させられ、描かれる。*horiz_justify* が 'l'、'c'、'r' のときそれぞれ、そのときのカーソルの位置に対して左揃え、センタリング、右揃えが行われる。*vert_justify* が 'b'、'x'、'c'、'C'、't' のときそれぞれ、そのときのカーソルの位置に文字列の、下端、ベースライン、中央、キャップ高、上端がそろえられる。左揃えが行われたときはカーソルは文字列の右端に、右揃えが行われたときは文字列の左端にカーソルは移動する。

文字列には各種のエスケープ・シーケンスを入れることができる (Section A.4 [Text String Format], page 144 を参照のこと)。しかし改行文字と復帰文字 (LF と CR) は入れられない。実際には文字列には印刷可能な文字だけ、つまりアスキーコードで言うと 0x20...0x7e と 0xa0...0xff の範囲内の文字が使える。また textangle を呼べば文字列を傾けることができる。

int arc (int *xc*, int *yc*, int *x0*, int *y0*, int *x1*, int *y1*);
int farc (double *xc*, double *yc*, double *x0*, double *y0*, double *x1*, double *y1*);
int arcrel (int *xc*, int *yc*, int *x0*, int *y0*, int *x1*, int *y1*);
int farcrel (double *xc*, double *yc*, double *x0*, double *y0*, double *x1*, double *y1*);

arc と farc の６つの引数でそれぞれ、描く円弧の始点 (*x0*, *y0*)、終点 (*x1*, *y1*)、中心点 (*xc*, *yc*) が指定される円弧を描く。もしカーソルが始点にあって、パスがまだ描き終わってないときは、描く円弧はそのパスの一部になる。そうでなければ描画途中のパスの描画は終了し、新しいパスとして円弧が描かれる。いずれにしてもカーソルは終点 (*x1*, *y1*) に移動する。

弧の向き (時計回りまたは反時計回りのどちらか) は、(*xc*, *yc*) に中心を持つ円弧の角度が 180 度以下になるような回転方向になる。もし指定された３点が１本の直線上に乗っている場合、円弧の方向は反時計回りになる。またもし、(*xc*, *yc*) からの距離が (*x0*, *y0*) と (*x1*, *y1*) で異なっている場合は、(*x0*, *y0*) と (*x1*, *y1*) を結ぶ線分の垂直二等分線上のもっともに近い場所に (*xc*, *yc*) が移動される。arcrel と farcrel は、カーソルに対する相対座標を使うこと以外は arc と farc と同じである。

int bezier2 (int *x0*, int *y0*, int *x1*, int *y1*, int *x2*, int *y2*);
int fbezier2 (double *x0*, double *y0*, double *x1*, double *y1*, double *x2*, double *y2*);
int bezier2rel (int *x0*, int *y0*, int *x1*, int *y1*, int *x2*, int *y2*);
int fbezier2rel (double *x0*, double *y0*, double *x1*, double *y1*, double *x2*, double *y2*);

bezier2 と fbezier2 は６つの引数で指定される３つの点、つまり始点 p0=(*x0*, *y0*) と終点 p2=(*x2*, *y2*)、その間にある制御点 p1=(*x1*, *y1*) からなる二次のベジェ曲線を描画する。もしカーソルが始点 p0 にあって、パスがまだ描き終わってないときは、そのパスの続きとして描かれる。そうでなければ描画途中のパスの描画は終了し、新しいパスとしてベジェ曲線が描かれる。いずれにしてもカーソルは終点 p2 に移動する。bezier2rel と fbezier2rel は、カーソルに対する相対座標を使うこと以外は bezier2 と fbezier2 と同じである。

二次のベジェ曲線では、点 p0 と p1を結ぶ線分が p0 における接線、点 p1 と p2を結ぶ線分が p2 における接線である。したがってこれは p0、p1、p2 を頂点とする三角形にぴったりとあてはまる。

PCL の Plotter で LaserJet III にベジェ曲線を描く場合は、パラメータ PCL_BEZIERS を "no" にせねばならない。LaserJet III はヒューレット・パッカードの最初の PCL 5 プリンタで、これ以降の PCL 5 と違ってベジェ曲線を描く命令をサポートしていない。Section 9.5 [Plotter Parameters], page 128 を参照のこと。

int bezier3 (int *x0*, int *y0*, int *x1*, int *y1*, int *x2*, int *y2*, int *x3*, int *y3*);
int fbezier3 (double *x0*, double *y0*, double *x1*, double *y1*, double *x2*, double *y2*, double *x3*, double *y3*);
int bezier3rel (int *x0*, int *y0*, int *x1*, int *y1*, int *x2*, int *y2*, int *x3*, int *y3*);
int fbezier3rel (double *x0*, double *y0*, double *x1*, double *y1*, double *x2*, double *y2*, double *x3*, double *y3*);

bezier3 と fbezier3 は８つの引数で指定される４つの点、つまり始点 p0=(*x0*, *y0*) と終点 p3=(*x3*, *y3*)、その間にある２つの制御点 p1=(*x1*, *y1*)、p2=(*x2*, *y2*) からなる三次のベジェ曲線を描画する。もしカーソルが始点 p0 にあって、パスがまだ描き終わってないときは、そのパスの続きとして描かれる。そうでなければ描画途中のパスの描画は終了し、新しいパスとしてベジェ曲線が描かれる。いずれにしてもカーソルは終点 p3 に移動する。bezier3rel と fbezier3rel は、カーソルに対する相対座標を使うこと以外は bezier3 と fbezier3 と同じである。

三次のベジェ曲線では、点 p0 と p1を結ぶ線分が p0 における接線、点 p2 と p3を結ぶ線分が p3 における接線である。したがってこれは p0、p1、p2、p3 を頂点とする四角形にぴったりとあてはまる。

PCL Plotter で LaserJet III にベジェ曲線を描く場合は、パラメータ PCL_BEZIERS を "no" にせねばならない。LaserJet III はヒューレット・パッカードの最初の PCL 5 プリンタで、これ以降の PCL 5 と違ってベジェ曲線を描く命令をサポートしていない。Section 9.5 [Plotter Parameters], page 128 を参照のこと。

int box (int *x1*, int y*1*, int *x2*, int *y2*);
int fbox (double *x1*, double *y1*, double *x2*, double *y2*);
int boxrel (int *x1*, int y*1*, int *x2*, int *y2*);
int fboxrel (double *x1*, double y*1*, double *x2*, double *y2*);

box と fbox は４つの引数で指定される２つの点、つまり左下隅 (*x0*, *y0*) と右上隅 p1=(*x1*, *y1*) で指定される箱形の長方形を描画する。パスがまだ描き終わってないときは、描画途中のパスの描画は終了し、新しいパスとして長方形が描かれる。カーソルは長方形の中心に移動する。boxref と fboxref は、カーソルに対する相対座標を使うこと以外は box と fbox と同じである。

int circle (int *xc*, int *yc*, int *r*);
int fcircle (double *xc*, double *yc*, double *r*);
int circlerel (int *xc*, int *yc*, int *r*);
int fcirclerel (double *xc*, double *yc*, double *r*);

circle と fcircle は３つの引数で指定される中心 (*xc*, *yc*) と半径 (*r*) の円を描く。パスがまだ描き終わってないときは、描画途中のパスの描画は終了し、新しいパスとして長方形が描かれる。カーソルは中心 (*xc*, *yc*) に移動する。circlerel と fcirclerel は、中心の座標にカーソルに対する相対座標を使うこと以外は circle と fcircle と同じである。

int cont (int *x*, int *y*);
int fcont (double *x*, double *y*);
int contrel (int *x*, int *y*);
int fcontrel (double *x*, double *y*);

cont と fcont は2つの引数で指定される点 (*x*, *y*) までの線分をカーソル位置から描く。パスが描画途中のときはそのパスに続けられる。そうでないときは新しいパスになる。どちらの場合もカーソルは (*x*, *y*) に移動する。contrel と fcontrel は、中心の座標にカーソルに対する相対座標を使うこと以外は cont と fcont と同じである。

int ellarc (int *xc*, int *yc*, int *x0*, int *y0*, int *x1*, int *y1*);
int fellarc (double *xc*, double *yc*, double *x0*, double *y0*, double *x1*, double *y1*);
int ellarcrel (int *xc*, int *yc*, int *x0*, int *y0*, int *x1*, int *y1*);
int fellarcrel (double *xc*, double *yc*, double *x0*, double *y0*, double *x1*, double *y1*);

ellarc と fellarc は6つの引数で指定される3つの点 pc=(*xc*,*yc*)、p0=(*x0*,*y0*)、p1=(*x1*,*y1*) で指定される、四半楕円と呼ばれる図形を描く。これは中心が pc で p0 から p1 につながる楕円弧である。もしカーソルが始点 p0 にあって、パスがまだ描き終わってないときは、そのパスの続きとして描かれる。そうでなければ描画途中のパスの描画は終了し、新しいパスとして描かれる。いずれにしてもカーソルは終点 p1 に移動する。

四半楕円は四半円をアフィン変換したものである。つまりこれには p0、p1、p0 + p1 − pc という制御点があることになる。したがって描かれる曲線は点 p0 で p0 と p0 + p1 − pc を結ぶ線分に接し、p1 で p1 と p0 + p1 − pc を結ぶ線分に接することになる。したがって描かれる四半楕円はこれらの制御点を頂点とする三角形に、ぴったりとあてはまる。3つめの制御点は、pc を p0 と p1 を結ぶ線分で対称な位置に移した点である。ellarcrel と fellarcrel は、カーソルに対する相対座標を使うこと以外は ellarc と fellarc と同じである。

int ellipse (int *xc*, int *yc*, int *rx*, int *ry*, int *angle*);
int fellipse (double *xc*, double *yc*, double *rx*, double *ry*, double *angle*);
int ellipserel (int *xc*, int *yc*, int *rx*, int *ry*, int *angle*);
int fellipserel (double *xc*, double *yc*, double *rx*, double *ry*, double *angle*);

ellipse と fellipse は5つの引数、つまり中心 (*xc*, *yc*)、長径と短径の長さ (*rx* および *ry*)、プログラム側の座標系の*x* 軸からの傾き *angle* で表される楕円を描く。描画途中のパスの描画は終了し、新しいパスとして楕円が描かれる。描画終了時にカーソルは (*xc*, *yc*) に移動する。ellipserel と fellipserel は、カーソルに対する相対座標を使うこと以外は ellipse と fellipse と同じである。

int endpath ();

描画途中のパスがあれば、それを終了し、描画する。また同時にそれまでに描画したパスを描画状態から削除し、新しいパスの構築を開始する。

終了するパスは単純パスでも、endsubpath (以下を参照) を使って構築中の複合パスでもよい。単純パスは cont、line、arc、ellarc、bezier2、bezier3 またはこれらの浮動小数点版のどれかを一つあるいは複数、連続して呼ぶことで構築される。単純パスはまた circle、ellipse、box のいずれかを一回呼び出すだけでも構築される。

endpath は自動的に呼ばれることが多いため、明示的に呼ばねばならないことは少ない。パスではないオブジェクトを描画したとき、パスの属性を変更したと

き、カーソル位置を変更するために move か fmove を呼んだときには、endpath を明示的に呼ぶ。restorestate を呼んでスタックから描画状態 (グラフィクス・コンテキスト) を取り出した場合、closepl を呼んでページの描画を終了した場合にも endpath を呼ぶ。したがってときどきは明示的に endpath を呼ぶ必要がある。リアルタイム描画を行う Plotter の場合は、endpath を呼ぶことにより遅延なく完結したパスを描画することになる。

int endsubpath ();
: 構築中の単純パスがあれば、それを完結し次の単純パスの構築を開始する。endsubpath の直後なら構築中の状態のまま move または fmove でカーソル位置を移動することができる (複合パスの構築中などのときは move や fmove を呼ぶと、自動的に endpath が呼ばれパスは構築中であっても完結されられる)。

int label (const char *s);
: 引数で指定された文字列を現在のカーソル位置に描画する。文字列は左揃えになり、カーソルは文字列の右端に置かれる。これは後方互換性を確保するために用意されているものであり、alabel('l','x',s) とするのと同じである。

int labelwidth (const char *s);
double flabelwidth (const char *s);
: labelwidth と flabelwidth は引数で指定された文字列の幅を、プログラム側座標における現在のフォントの場合で計算して返す。なにも描画しない。

int line (int x1, int y1, int x2, int y2);
int fline (double x1, double y1, double x2, double y2);
int linerel (int x1, int y1, int x2, int y2);
int flinerel (double x1, double y1, double x2, double y2);
: line と fline は、4つの引数で指定される2つの点、つまり始点 (x1, y1) と終点 (x2, y2) を結ぶ線分を描く。もしカーソルが (x1, y1) にあって、パスがまだ描き終わってないときは、そのパスの続きとして描かれる。そうでなければ描画途中のパスの描画は終了し、新しいパスとして線分が描かれる。いずれにしてもカーソルは (x2, y2) に移動する。linerel と flinerel は、カーソルに対する相対座標を使うこと以外は line と fline と同じである。

int marker (int x, int y, int type, int size);
int fmarker (double x, double y, int type, double size);
int markerrel (int x, int y, int type, int size);
int fmarkerrel (double x, double y, int type, double size);
: marker と fmarker は4つの引数、つまりプログラム側座標系での位置 (x,y)、種類、サイズの指定にしたがって記号を描画する。描画途中のパスがあればその描画は終了し、記号が描かれる。カーソルの位置は (x,y) になる。markerrel と fmarkerrel は、カーソルに対する相対座標を使うこと以外は marker と fmarker と同じである。

  描かれる記号は描画領域上の点の位置を示すものであり、どの Plotter でも使える。これは point 関数 (次に述べる) とは違うものである。描かれる記号の種類番号 0 から 31 までは標準で用意されている記号、32 以上のものはそのときのフォントのキャラクタ・マップ上の番号とみなされる。Section A.5 [Marker Symbols], page 155 参照。

int point (int x, int y);
int fpoint (double x, double y);
int pointrel (int x, int y);
int fpointrel (double x, double y);

point と fpoint は 2 つの引数で指定される点 (x, y) に点を描く。構築中のパスがあれば endpath を呼ぶのと同様に完結させられ、そのパスが描画され、点が描かれる。カーソルは (x, y) に移動する。pointrel と fpointrel は、カーソルに対する相対座標を使うこと以外は point と fpoint と同じである。

しかし point という関数名はあまり正しいとは言えない。X、X Drawable、PNG、PNM、GIF のビットマップ形式の画像を出力する Plotter では、一つの点は 1 ピクセルとして描かれる。他の Plotter ではほとんどの場合、描画できる最小の塗りつぶした円として描かれるため、見えないこともある。そのため point は場合によっては pixel と呼ぶ方がいいかもしれない。

9.4.3 属性設定関数

以下に libplot の「属性操作」を行う関数を説明する。Plotter に対してこれらの関数を実行することでその関数が対象とする描画属性を指定する値に設定したり、保存、再設定を行ったりできる。たとえばパスに関連する属性には、カーソル位置、ペンの色、塗りつぶしの色、塗りつぶし判定規則、線の太さ、線種、線の端点の形、つなぎ目の形状、角の尖り具合、変換行列がある。文字列に関する属性には、ペンの色、フォント名、フォントサイズ、傾き、変換行列がある。

パスの属性を設定すると、endpath を呼んだときと同様、パスの描画が途中であれば、それは完結させられ、描画される。しかし円、楕円、長方形の 'orientation' 属性 (時計回り／反時計回り) についてはそうではない。それにより複合パス中の円、楕円、長方形のパスの向きをいろいろと変えることができる。

現在の C バインディングでは、これらの各関数の第一引数は plPlotter へのポインタである。また各関数の先頭には "pl_" が、末尾には"_r" が付いている ("_r" は 'revised' または 'reentrant' の意味である)。古い C バインディングについては Section 9.2.2 [Older C APIs], page 89 を参照のこと。C++ バインディングでは、これらは Plotter クラスとその派生クラスのメンバー関数であり、先頭にも末尾にも特になにもついていない。

int capmod (const char *s);

構築中のパスがあれば、endpath を呼んだときと同様その構築を完了して描画し、それ以降に描画領域に描くパスの先端の形状を変更する。指定できる形状は "butt" (これがデフォルト)、"round"、"projecting" のいずれかである。しかし形状の違いは、ある程度パスを描く線が太くないと見分けがつかない。"butt" を指定すると、パスの端はまったく、少しも長くならない。しかし他の形状では少しだけ長くなる。"round" の場合はパスの端に半円が付け加えられ、"projecting" ではパスの端に長方形が付け加えられ、線の太さの半分だけパスが長くなる。

PNG、PNM、GIF、PCL、HP-GL の Plotter では 4 つめの形状として "triangular" も指定できる (しかし PCL または HP-GL 以外の Plotter では、まだ部分的にしかサポートしていない)。"triangular" をサポートしていない Plotter で "triangular" を指定すると、"round" を指定したのと同じことになる。

ReGIS または Tektronix の Plotter では capmod は無効である。また HPGL_VERSION パラメータの値が 2 より小さいときは、HP-GL でも無効である。同様

に、CGM_MAX_VERSION の値が 3 より小さいときは、CGM の Plotter でも無効である。Section 9.5 [Plotter Parameters], page 128 参照。

int color (int *red*, int *green*, int *blue*);
この関数を使うと、pencolor と fillcolor の両方を呼んでこれ以降に描かれるすべての図形の線の色と塗りつぶしの色をそれぞれ同じ色に設定するのと同じことができる。塗りつぶしのときに実際に使われる色は filltype を呼んで指定される「塗りつぶし率」によって変わる。

int colorname (const char **name*);
この関数を使うと、pencolorname と fillcolorname の両方を呼んでこれ以降に描かれるすべての図形の線の色と塗りつぶしの色をそれぞれ同じ色に設定するのと同じことができる。塗りつぶしのときに実際に使われる色は filltype を呼んで指定される「塗りつぶし率」によって変わる。

int fillcolor (int *red*, int *green*, int *blue*);
構築中のパスがあれば、endpath を呼んだときと同様その構築を完了して描画し、これ以降に描画領域に描かれるすべての図形の塗りつぶし色を 48-bit RGB で指定する。引数 *red*、*green*、*blue* で塗りつぶし色の赤、緑、青の濃さを指定する。それぞれ 0x0000...0xffff、つまり 0...65535 の範囲の整数である。(0, 0, 0) を指定すると黒に、(65535, 65535, 65535) を指定すると白になる。塗りつぶしのときに実際に使われる色は filltype を呼んで指定される「塗りつぶし率」によって変わる。

int fillcolorname (const char **name*);
構築中のパスがあれば、endpath を呼んだときと同様その構築を完了して描画し、これ以降に描画領域に描かれるすべての図形の塗りつぶし色を色名 *name* で指定される色に設定する。無効な色名を指定したときは "black" に置き換えられる。設定できる色名については Appendix B [Color Names], page 158 を参照のこと。"#c0c0c0" にように 6 桁の 16 進数で 24 ビットカラーとしても指定できる。

塗りつぶしのときに実際に使われる色は filltype を呼んで指定される「塗りつぶし率」によって変わる。

int fillmod (const char **s*);
構築中のパスがあれば、endpath を呼んだときと同様その構築を完了して描画し、これ以降に描画領域に描かれるすべての図形について、塗りつぶし判定の仕方を指定する。これは、描画領域上のある点がパスの「内側」かどうかを判定する方法である。"even-odd" (どの Plotter でもこれがデフォルト) または "nonzero-winding" が指定できる。詳しくは *PostScript Language Reference Manual* を参照のこと。"even-odd" の代わりに "alternate"、 "nonzero-winding" の代わりに "winding" と指定しても良い。

CGM、Fig、ReGIS の Plotter では "nonzero-winding" は無効である。これは、それぞれの画像形式がその機能をサポートしていないからである。HPGL_VERSION が 2 より小さいときは HP-GL の Plotter でも無効である (2 がデフォルト)。Section 9.5 [Plotter Parameters], page 128 参照。

LaserJet III はヒューレット・パッカードの最初の PCL 5 プリンタで、これ以降の PCL 5 プリンタと違って nonzero-winding をサポートしていない。

int filltype (int *level*);

構築中のパスがあれば、endpath を呼んだときと同様その構築を完了して描画し、これ以降に描画領域に描かれる図形の塗りつぶしの濃さを指定する。*level* の値を 0 にすると、図形は塗りつぶされない。これがデフォルトである。*level* の値を 1 にすると 100% の濃さ (fillcolor や fillcolorname で指定されるままの色) で塗りつぶされる。

level は 0x0000...0xffff、つまり 1...65535 の範囲の整数でも指定できる。非零であれば塗りつぶしが行われる。*level*=0xffff とすると白色で塗りつぶされる。0x0001 から 0xffff の間の値は彩度 (の逆) またはグレイスケールと見なされる。たとえば 0x8000 は彩度を 50% にして塗りつぶす (実際に塗りつぶされる色は fillcolor か fillcolorname で指定された色と白色の中間の色になる)。

パスに囲まれた領域を塗りつぶし、その外枠を描きたくない場合、filltype で塗りつぶしを指定し pentype でパスの描画を抑制すればよい。

Tektronix の Plotter は塗りつぶしができない。また HP-GL の Plotter はパラメータ HPGL_VERSION が "1.5" か "2" (2 がデフォルト) のときにだけ有効である (バージョンが "1" のときは、座標軸上に置かれた円と長方形だけが塗りつぶし可能である)。白色を含めた不透明塗りつぶしは、HPGL_VERSION が "2" でパラメータ HPGL_OPAQUE_MODE が "yes" のとき (それがデフォルト) だけ可能である。Section 9.5 [Plotter Parameters], page 128 参照。

int fmiterlimit (double *limit*);

構築中のパスがあれば、endpath を呼んだときと同様その構築を完了して描画し、これ以降に描画領域に描かれるパスの、角の尖り具合を指定する。パスの連結のモードが "miter" に設定されている時 (これがデフォルト) のときに有効である。パスの連結点における角の内側から外側までの長さが「角幅」である。この角幅の線の太さに対する割合の最大限度をこの関数で設定できる。角幅が指定の値を超える場合、パスの連結モードが "bevel" に変更され、角が「切り落とされる」。

limit の値は 10.43 (デフォルト値。パスの連結する角度が 11 度以下になるとこの値を超える)、または 2.0 (同じく 60 度)、または 1.414 (同じく 90 度) といった値にすることが多い。この値は、パスの連結する角度の 1/2 の余割 (コセカント) である。*limit* の値を 1.0 にすると、連結する角度に関係なくすべて "bebel" モードになる。1.0 より小さな値は無効で、そういう値を指定するとデフォルト値が使われる。

X Drawable または X の Plotter ではこの設定は無効である。X Window システムの設定は 10.43 のまま変更できないからである。また Tektronix、ReGIS、Fig の各 Plotter でも無効であり、HPGL_VERSION が 2 より小さいときは HP-GL の Plotter でも無効である (2 がデフォルト)。Section 9.5 [Plotter Parameters], page 128 参照。また HP-GL または PCL の Plotter のときは、実数値が指定されていた場合、小数点以下は切り捨てられて整数値が使われる。

int fontname (const char **font_name*);
double ffontname (const char **font_name*);

これ以降に描画領域に描かれる文字列のフォントを引数 *font_name* で示されるフォントに設定する。フォントの指定では大文字、小文字を区別せず、一つのフォントだけが指定できる (実際に描画される文字は fontname、fontsize、textangle で決定される)。プログラム側座標でのフォントのサイズが返り値になる。

デフォルトのフォントは Plotter の種類によって異なっている。PCL の Plotter では "Univers"、PNG、PNM、GIF、HP-GL、ReGIS、Tektronix、Metafile の Plotter では"HersheySerif"、それ以外の Plotter では "Helvetica" がデフォルトである。*font_name* が NULL または空文字列のとき、あるいは無効なフォントを指定したときは、デフォルトのフォントが使われる。使えるフォントの種類も Plotter の種類によって異なっている。そのリストについては Section A.1 [Text Fonts], page 136 を参照のこと。

int `fontsize` (int *size*);
double `ffontsize` (double *size*);

これ以降に描画領域に描かれる文字列のフォントのサイズを引数 *size* に設定する (実際に描画される文字は `fontname`、`fontsize`、`textangle` で決定される)。プログラム側座標でのフォントのサイズが返り値になる。

負の値を *size* に指定するとデフォルトのサイズに設定されるが、具体的な値は Plotter の種類によって異なる。普通はディスプレイ (の縦と横の短い方) のサイズの 1/50 がデフォルトである。フォントのサイズが 0 の場合の動作も Plotter の種類によって異なるが、ほとんどの Plotter ではその場合文字列を描画しない。

int `joinmod` (const char *s);

構築中のパスがあれば、`endpath` を呼んだときと同様その構築を完了して描画し、これ以降に描画領域に描かれるパスのつなぎ方を設定する。"miter" (これがデフォルト)、"round"、"bevel" の 3 種類がある。線の太さがある程度なければ、3 つの種類はそれぞれ見分けがつかないだろう。線のつなぎ目が "miter" モードのときは、つなぎ目は尖った角になる。"round" モードでは丸く、"bevel" モードでは切り落としたような感じになる。あまり尖りすぎている角では、"miter" が指定されていても角が鋭く尖るのではなく、切り落とされる。どの程度尖ったら切り落とされるのかは、`fmiterlimit` を呼んで設定することができる。

PNG、PNM、GIF、PCL、HP-GL の Plotter では "triangular" も指定できる。他の Plotter で これを指定すると、"round" と同じ扱いになる。

この機能は Tektronix と ReGIS の Plotter では無効である。またパラメータ `HPGL_VERSION` が 2 より小さいときは HP-GL の Plotter でも無効である (2 がデフォルト)。同様に、`CGM_MAX_VERSION` の値が 3 より小さいときは、CGM の Plotter でも無効である。Section 9.5 [Plotter Parameters], page 128 参照。

int `linedash` (int *n*, const int *dashes, int *offset*);
int `flinedash` (int *n*, const double *dashes, double *offset*);

構築中のパスがあれば、`endpath` を呼んだときと同様その構築を完了して描画し、これ以降に描画領域に描かれるパス、円、楕円の線種を指定する。`linemod` (後述) よりもこの関数の方がより詳細に描き方を指定できる。引数 *dashes* はサイズが *n* の配列で、その各要素は正の値でなければならない。その値はプログラム側の座標系における距離とみなされる。パス、円、楕円を描く線は、この各要素の値 *dashes*[0]...*dashes*[*n*-1] で交互に線の長さと線の間の空白の長さを示される破線として描かれる。配列の終わりまで破線を描いたら、また配列の先頭から繰り返される。この配列が空の場合、つまり *n* が 0 のときは、破線にはならず、実線として描かれる

offset は、パスの先端からの破線の「位相」を指定する。つまり破線パターンのいくらか進んだところをパスの先端に合わせて、そこから破線を描く。*offset* が 0 ならパスの先端で破線パターンが最初から始まり、プログラム側座標で長さ

dashes[0] のあとには破線の切れ目の空白になる。もし *offset* が *dashes*[0] と同じ値だったら、パスの先頭は長さ *dashes*[1] の切れ目から始まり、以下同様である。*offset* は負の値でも良い。

`linedash` および `flinedash` の機能には Plotter によっては制限がある。HP-GL と PCL の Plotter では *offset* は 0 でなければならない。また X と X Drawable の Plotter では破線パターン中の線と空白はそれぞれ 255 ピクセル以下でなければならない。Tektronix、ReGIS、Fig の各 Plotter では `linedash` と `flinedash` は無効である。またパラメータ `HPGL_VERSION` が 2 より小さいときは HP-GL の Plotter でも無効である (2 がデフォルト)。同様に、`CGM_MAX_VERSION` の値が 3 より小さいときは、CGM の Plotter でも無効である。Section 9.5 [Plotter Parameters], page 128 参照。

注意: プログラム側からデバイス上の座標系への変換が一様 (均一) ではない場合、パスを描く破線上の線の部分の長さは、デバイス上では線の向きによって変化する。しかし現在これが可能なのは PostScript の Plotter だけである。現在の他の Plotter では、破線の線と空白の長さはパスの向きがどうであっても変わらない。そのときに使われる線の部分の長さは、プログラム側座標系で指定された長さから変換されうる長さのうち、もっとも短い長さになる。

int `linemod` (const char *s);

構築中のパスがあれば、`endpath` を呼んだときと同様その構築を完了して描画し、これ以降に描画領域に描かれるパス、円、楕円の線種を指定する。指定できる線種は"solid"、"dotted"、"dotdashed"、"shortdashed"、"longdashed"、"dotdotdashed"、"dotdotdotdashed"、"disconnected" のいずれかである。最後の一つを除くと、それぞれ以下のような破線を意味する。

```
"solid"             ---------------------------------
"dotted"            -   -   -   -   -   -   -   -
"dotdashed"         ----   -   ----   -   ----   -
"shortdashed"       ----    ----    ----    ----
"longdashed"        -------    -------    -------
"dotdotdashed"      ----   -   -   ----   -   -
"dotdotdotdashed"   ----   -   -   -   ----   -   -   -
```

上の図中で、ハイフン一つが描かれている線の太さと同じ長さを示す。ただしこれは線がある程度太い場合のことであり、その場合は、線の太さに比例して線分の間隔も広くなる。

"disconnected" は上記とは違って、直径が線の太さの指示値と同じであるような、塗りつぶされた円でパスを描くことを指示する。パスの連結点 (パスを構成する線分や円弧のつなぎ目) に中心が来るように円が描かれる。円や楕円の線種に "disconnected" を指定すると、連結点がないので、見えなくなる。この線種を指定されたパス、円、楕円には塗りつぶしは指示できない。

上述の線種はすべてどの Plotter にも使えるが、例外がある。パラメータ `HPGL_VERSION` が "2" でないときは、HP-GL の Plotter では "dotdotdotdashed" は使えない (2 がデフォルト)。Tektronix の Plotter では "dotdotdotdashed" は無効である。Section 9.5 [Plotter Parameters], page 128 参照。

int linewidth (int *size*);
int flinewidth (double *size*);

構築中のパスがあれば、endpath を呼んだときと同様その構築を完了して描画し、これ以降に描画領域に描かれるパス、円、楕円の線の太さを、プログラム側の座標系における値で指定する。負の値を指定すると、その値は無視され、デフォルト値に設定される。デフォルトの太さは Plotter の種類によって異なる。ほとんどの Plotter では描画領域の (縦か横の短い方の) 幅の長さの 1/850 である。しかし X、X Drawable、PNG、PNM、GIF のビットマップ画像の Plotter では、デフォルト値は 0 である。

太さ 0 の線はその Plotter で描画できるもっとも細い線になるが、しかし idraw や xfig では、太さ 0 の線は表示されない。そのため PostScript や Fig の Plotter の出力画像を編集したいようなときは、太さ 0 の線は避けた方が良い。

Tektronix、ReGIS の Plotter では線の太さはデフォルト以外は使えない。またそれは、パラメータ HPGL_VERSION が 2 より小さいときは HP-GL の Plotter でも同じである (2 がデフォルト)。Section 9.5 [Plotter Parameters], page 128 を参照のこと。

注意: プログラム側からデバイス上の座標系への変換が一様 (均一) ではない場合、パスを描く線の太さは、デバイス上では線の向きによって変化する。しかし現在これが可能なのは PostScript の Plotter だけである。現在の他の Plotter では、線の太さはパスの向きがどうであっても変わらない。そのときに使われる線の太さは、プログラム側座標系で指定された太さから変換されうる太さのうち、もっとも短い太さになる。

int move (int *x*, int *y*);
int fmove (double *x*, double *y*);
int moverel (int *x*, int *y*);
int fmoverel (double *x*, double *y*);

二つの引数で指定される一つの点 (*x*, *y*) にカーソルを移動する。描画途中のパスがあった場合には、endpath を呼んだときと同様にそのパスはそこで完結させられ、カーソルが (*x*, *y*) に移動する。これはプロッタがペンを上げて、線を描かずに指定された点まで移動するのと同じ操作になる。moverel と fmoverel は、カーソルに対する相対座標を使うこと以外は move と fmove と同じである。

openpl 操作で新しくページを開始したとき、カーソルの位置はプログラム側座標での (0,0) にセットされる。描画を行う操作ではほとんどの場合カーソルの位置が移動する。Section 9.4.2 [Drawing Functions], page 115 参照。

int orientation (int *direction*);

描画領域に円、楕円、長方形を描くときの線の向きを指定する。反時計回りは *direction* を 1 に、時計回りは −1 にする。デフォルトは 1 である。

円、楕円、長方形の輪郭を破線などで描く場合や、それらが複合パスの一部である場合に、線の向きが意味を持つ。塗りつぶし則に "nonzero-winding" が指定されているときは、複合パス中の単純パスの向きによって、図形の「内側」、「外側」が決まるため、線の向きの設定によって視覚的な効果が得られることもある。

int pencolor (int *red*, int *green*, int *blue*);

構築中のパスがあれば、endpath を呼んだときと同様その構築を完了して描画し、これ以降の描画に使われるペンの色を、48-bit RGB で指定する。3 つの引数 *red*、*green*、*blue* で赤、緑、青の濃さを指定する。それぞれ 0x0000...0xffff、

つまり 0...65535 の範囲の整数である。(0, 0, 0) を指定すると黒に、(65535, 65535, 65535) を指定すると白になる。

HP-GL の Plotter はパラメータ HPGL_VERSION が "2" で、パラメータ HPGL_OPAQUE_MODE が "yes" のときだけペンの色に白色が使える。Section 9.5 [Plotter Parameters], page 128 参照。

int pencolorname (const char *name);

構築中のパスがあれば、endpath を呼んだときと同様その構築を完了して描画し、これ以降の描画に使われるペンの色を、*name* で指定される色に設定する。無効な色名を指定したときは "black" に置き換えられる。"#c0c0c0" にように 6 桁の 16 進数で 24 ビットカラーでも指定できる。

HP-GL の Plotter はパラメータ HPGL_VERSION が "2" で、パラメータ HPGL_OPAQUE_MODE が "yes" のときだけペンの色に白色が使える。Section 9.5 [Plotter Parameters], page 128 参照。

int pentype (int *level*);

構築中のパスがあれば、endpath を呼んだときと同様その構築を完了して描画し、これ以降の描画に使われるペンのレベルを、*level* で指定される値に設定する。*level* の値を 1 に設定すると、これ以前の最後に実行された pencolor または pencolorname で指定された色で図形の輪郭線を描く。これがデフォルトである。*level* の値を 0 にすると輪郭は描かれない。

輪郭のない図形を描きたいときは、pentype で輪郭を描かないように指定し、filltype で内部を塗りつぶすように指定するとよい。

pentype による指定は記号や点 (ピクセル) の描画にも適用される。*level* の値を 0 にするとこれらも描かれなくなる。

注意: 将来的には pentype の指定は文字列に対しても有効になる予定 (値を 0 にすると文字列も描かれなくなる予定) だが、現在のバージョンでも Hershey フォントの文字に対しては有効である。Hershey フォントの文字は線分から構築されるパスで描かれるからである。

int restorestate ();

現在の描画状態をスタックから取り出して復元する。描画状態は、この節で説明されている libplot の属性からなる。スタックからそれらを取り出すことにより、各種の描画属性の設定が取り出した値 (スタックに積んだときの値) に再設定される。構築途中のパスも、描画状態の一部として扱われる。したがって restorestate を呼ぶと自動的に endpath が呼ばれ、そのときに構築途中のパスがあったときはその構築はそこで終了する。また closepl を呼ぶとそれまでに restorestate が呼ばれたときと同様に、スタックに積まれている描画状態が復元される。

int savestate ();

現在の描画状態をスタックに積んで保存する。描画状態は、この節で説明されている libplot の属性からなる。構築途中のパスがあれば、そのことも描画状態の一部として扱われる。パスは線分や円弧を一つずつつなげて描かれるものだからである。savestate によって (保存された描画状態の代わりに) 新しく作られる描画状態には、なんのパスも含まれない。それを含む描画状態が restorestate を呼ぶことでスタックから取り出され復元されるときには、そのパスは構築途中の状態で復元され、続きを描画することができる。

int `textangle` (int *angle*);
double `ftextangle` (double *angle*);

これ以降に描画領域に描画される文字列に対して、それを傾ける角度を、引数 *angle* で度 (degree) で x (横) 軸からの角度として指定される値に設定する。デフォルト値は 0 である (実際に描画される文字は `fontname`、`fontsize`、`textangle` で決定される)。プログラム側座標でのフォントのサイズが返り値になる。

注意: X Window では、システムによっては回転角が指定した通りにならないことがある。そういったシステムでは事実上、回転角には 0 しか指定できない。

9.4.4 座標変換関数

以下に `libplot` の「座標変換関数」について説明する。これらの関数を Plotter に対して実行すると、プログラム側の座標系からデバイス上の座標系への変換を操作することができる。この変換がどう行われるのかは、プログラム側の座標系上で考えるようになっている。以下の操作は、その操作を行った後に描画領域に描かれる図形に対して有効になる。

関数名は、PostScript 言語で定義されている同様の機能の関数と同様になっている。これらの関数をどう使うと便利か、については PostScript プログラミングの本か、*PostScript Language Reference Manual* を参照されたい。

これらの操作を行うときに構築中のパスがあれば、`endpath` を呼んだときと同様その構築を完了する。

現在の C バインディングでは、これらの各関数の第一引数は `plPlotter` へのポインタである。また各関数の先頭には `"pl_"` が、末尾には`"_r"` が付いている (`"_r"` は 'revised' または 'reentrant' の意味である)。古い C バインディングについては Section 9.2.2 [Older C APIs], page 89 を参照のこと。C++ バインディングでは、これらは `Plotter` クラスとその派生クラスのメンバー関数であり、先頭にも末尾にも特になにもついていない。

int `fsetmatrix` (double *m0*, double *m1*, double *m2*, double *m3*, double *tx*, double *ty*);

プログラム側の座標から NDC (normalized device coordinate) 上での値への変換を、変換行列 [*m0 m1 m2 m3 tx ty*] を使って行うように設定する。この変換行列によりプログラム側の座標系から、正規化された出力でバイス上の座標系への変換が設定され、この操作以降に描画領域に描かれる図形に対して、この変換が適用される。

NDC 座標系では、描画領域 (または viewport) の四隅はそれぞれ `(0,0)`、`(1,0)`、`(1,1)`、`(0,1)` である。描画領域の大きさや物理的な長さについては Appendix C [Page and Viewport Sizes], page 159 を参照のこと。

プログラム側で使う座標系から NDC への変換行列は、デフォルトでは [1 0 0 1 0 0] であり、つまりプログラム側の座標系と NDC は一致している。変換行列は以下の関数や `space`、`fspace`、`space2`、and `fspace2` でも指定できる。

int `fconcat` (double *m0*, double *m1*, double *m2*, double *m3*, double *tx*, double *ty*);

プログラム側の座標系から NDC への変換の際に、その変換行列よりも先にプログラム側座標に行列 [*m0 m1 m2 m3 tx ty*] をかけることで、変換を修正するように設定する。つまり、プログラム側の座標値に 2 x 2 の行列 [*m0 m1 m2 m3*] をかけて、それから x 方向に *tx*、y 方向に *ty* だけ平行移動する線形変換と同じである。

`fconcat` は内部で `fsetmatrix` を呼び出して使っている。`fconcat` で指定できる変換の一部を簡便に指定する関数として、以下の 3 つの関数 (`frotate`、`fscale`、`ftranslate`) が用意されている。

int frotate (double *theta*);
: プログラム側の座標系から NDC への変換の際に、その変換行列よりも先にプログラム側座標に行列 [cos(*theta*) sin(*theta*) −sin(*theta*) cos(*theta*) 0 0] をかけることで、変換を修正する。プログラム側座標系の座標軸を *theta* で指定される角度だけ、プログラム側座標系の原点を中心に回転させる。原点の位置と x と y の長さは変わらない。

int fscale (double *sx*, double *sy*);
: プログラム側の座標系から NDC への変換の際に、その変換行列よりも先にプログラム側座標に行列 [*sx* 0 0 *sy* 0 0] をかけることで、変換を修正する。プログラム側の座標系での x と y の単位長さを、元の座標系における *sx* と *sy* の長さにする。原点の位置と座標軸の向きは変わらない。

int ftranslate (double *tx*, double *ty*);
: プログラム側の座標系から NDC への変換の際に、その変換行列よりも先にプログラム側座標に行列 [0 0 0 0 *tx* *ty*] をかけることで、変換を修正する。プログラム側の座標系の原点を、プログラム側座標における x 方向に *tx*、y 方向に *ty* だけ移動する。x と y の長さと座標軸の向きは変わらない。

9.5 Plotter パラメータ

libplot ライブラリの設計に当たっては、各描画デバイスの各機能とは独立して描画が行えるようにしているが、各 Plotter でそれぞれ独立に異なった描画ができるように、Plotter パラメータでさまざまな動作を個別に指定できるようにしている。パラメータは型のないポインタ (void *) でその値を指定できるが、ほとんどのパラメータは文字列型 (char *) である。

Plotter に対するパラメータの値は、どのパラメータも、その Plotter が生成されてから破棄されるまで変更できない。その値は Plotter が生成されるときに決定される。C バインディングでは pl_setplparam 関数でその値を指定する。pl_setplparam 関数は Plotter のパラメータを保持する plPlotterParams 型を持つオブジェクトに対して操作を行う。pl_newpl_r 関数が呼ばれて Plotter を生成するときに、その plPlotterParams 型のオブジェクトを最後の引数として渡す。

文字列をとるパラメータの場合、Plotter の生成時にそれが設定されておらず、パラメータと同名の環境変数に値がセットされているときは、それが使われる。また同名の環境変数も設定されていないときは、デフォルト値が使われる。これにより、実行時にプログラムを制御することができる。さほど重要でないようなパラメータを、プログラムの実行時に環境変数で指定するようにすることができる。

C++ バインディングでは PlotterParams クラスとそのメンバー関数 PlotterParams::setplparam がそれぞれ、plPlotterParams 型と関数 pl_setplparam に相当する。

以下に、現在有効なパラメータを列挙する (認識されないパラメータは省いてある)。重要なパラメータは X の Plotter の DISPLAY、X、PNG、PNM、GIF の Plotter の BITMAPSIZE、Illustrator、PostScript、CGM、Fig、HP-GL の Plotter の PAGESIZE、Metafile 以外のすべての Plotter の ROTATION である。最初にこの４つのパラメータの説明を挙げ、他のものはその後にアルファベット順に述べる。その中で、たとえば名前が "HPGL" で始まるようなパラメータは、その Plotter に対してだけ有効である。

DISPLAY (デフォルト値は NULL) グラフィクス・ディスプレイのウィンドウが開かれる X Window システムのディスプレイを指定する。これは X の Plotter でだけ有効である。

BITMAPSIZE
(デフォルト値は "570x570") ピクセル単位での描画領域のサイズ。これは X、PNG、PNM、GIF の Plotter でだけ有効である。X の Plotter では、これが設定されていなければ X resource の Xplot.geometry から自動的に設定されるが、これは後方互換性を保つための仕様である。

X の Plotter ではこのパラメータによりウィンドウの位置も指定できる。たとえば BITMAPSIZE を "570x570+0+0" と指定すると、X Window システムの画面の左上隅にウィンドウが開かれる。

PAGESIZE (デフォルト値は "letter") 描画領域 (viewport) が置かれる用紙のサイズ。これは SVG、Illustrator、PostScript、CGM、Fig、PCL、HP-GL の Plotter でだけ有効である。"letter" サイズは 8.5 in x 11 in である。ISO の"a0"...."a4" や ANSI の "a"...."e" の各サイズも指定できる ("letter" は "a" を、"tabloid" は "b" を指定するのと同じである)。"legal"、"ledger"、"b5" も指定できる。

Illustrator、PostScript、PCL、FIG の各 Plotter では、描画領域は用紙の中央に置かれ、用紙の幅いっぱいの大きさの正方形の領域になる (たとえば PAGESIZE が "letter" のときは、用紙の中央に置かれた一辺が 8 in の正方形になる)。HP-GL の Plotter では、描画領域の大きさは同じだが、デフォルトでは中央には置かれない。SVG と WebCGM では、画像がウェブページ上でどこに置かれるかは画像自体では指定されない。画像の大きさは指定できるが、それもウェブページ上に表示されるときには変更されるのが普通である。SVG と CMG の Plotter の描画領域のデフォルトのサイズは、ほかの Plotter と同じである。

各出力形式での、描画領域のデフォルトの大きさ (と位置) については Appendix C [Page and Viewport Sizes], page 159 を参照のこと。描画領域の縦、横の幅はそれぞれ指定できるので、必ずしもデフォルト値を使う必要はない。たとえば PAGESIZE には"letter,xsize=4in" や "a4,xsize=10cm,ysize=15cm" といった指定ができる。長さには負の値も指定できる (そのときは鏡像が描かれる)。

SVG と CGM 以外の Plotter では用紙上の描画領域の相対的な位置は、ベクトルとして指定できる。たとえば PAGESIZE には"letter,yoffset=1.2in" や "a4,xoffset=−5mm,yoffset=2.0cm" といった指定ができる。長さの単位にはインチ、cm、mm が使える。指定には"xsize"、"ysize" と "xoffset"、"yoffset" を合わせて使うことができる。

また用紙の左下隅から見た描画領域の左下隅の位置を指定することで、描画領域の位置を指定することもできる。たとえば PAGESIZE には "letter,xorigin=2in,yorigin=3in" や "a4,xorigin=0.5cm,yorigin=0.5cm" といった指定ができる。"xorigin"、"yorigin" は "xsize"、"ysize" と合わせて使うことができる。SVG と WebCGM では、画像がウェブページ上でどこに置かれるかは画像自体では指定されないため、"xoffset"、"yoffset"、"xorigin"、"yorigin" は無視される。

ROTATION (デフォルト値は "0.0") このパラメータは Metafile 形式では具体的な出力デバイスが想定されていないので無効だが、他のすべての Plotter に対して有効である。用紙上での描画領域 (viewport) の傾きを、デフォルトの角度に対する反時計回り方向での相対的な角度 (度数) で指定する。

描画領域は傾けられても、その四隅の用紙上の位置は変わらない。傾けられるのはそこに描かれる図形である。描画領域が正方形ではなく長方形の場合、単に傾くだけではなく、縦と横で違うスケーリングが行われることになる。

このパラメータは、用紙の横置きと縦置き (ランドスケープとポートレート) を切り替えるときに使う。ライブラリの内部で NDC (normalized device coordinate) からプログラム側座標系へのアフィン変換が設定される。

BG_COLOR (デフォルト値は "white") 各ページが開始されるときの描画領域の背景色を設定する。これは X、PNG、PNM、GIF、CGM、ReGIS、Metaifle の各 Plotter で有効である。また X Drawable では背景色は erase が呼ばれたときにだけ見えるようになる。指定できる色名については Appendix B [Color Names], page 158 を参照のこと。背景色は Plotter が生成された後でも bgcolor (または bgcolorname) と erase を行うことでいつでも変更できる。

SVG と CGM の Plotter では背景色に "none" を指摘できる。これにより背景色が無効になり、描かれる画像には背景がなくなる。ウェブページに載せる画像を生成するときに指定するとよい。

CGM_ENCODING
(デフォルト値は "binary") CGM の Plotter でだけ有効である。"binary" を指定すると、CGM 出力がバイナリ・モードで行われる。"clear_text" を指定すると、人間に読める形式で出力される。webCGM プロファイルではバイナリ形式とすることになっているが、CGM を表示する多くの実装ではどちらの形式でも表示できる。 3つめの標準形式 "character" は現在はサポートしていない。

CGM_MAX_VERSION
(デフォルト値は "4") CGM の Plotter でだけ有効である。生成される CGM 出力形式のバージョンを最大でいくつまでにするかを指定する。CGM を表示するソフトウェアの古いものはバージョン 1 しかサポートしていない。フォントや線種を十分にサポートするのはバージョン 3 以降である。デフォルトでは、現在の libplot はバージョン 3 形式で出力し、バージョン 4 の機能は使わない。

EMULATE_COLOR
(デフォルト値は "no") すべての Plotter で有効である。これを "yes" に設定すると、描画に使われる色が、適切なグレイスケールに置き換えられる。CIE 輝度の計算法としてよく知られている、$0.212671R + 0.715160G + 0.072169B$ という式で変換する。

このパラメータは、PCL の Plotter で白黒の PCL 5 機器に出力するときはあまり意味がない。白黒の LaserJet などの白黒の PCL 5 機器では、もともと色のエミュレーションの機能が乏しく、HP-GL/2 でのペン色を、黄色を含めて 7 色とも黒に置き換えてしまう。

GIF_ANIMATION
(デフォルト値は "yes") このパラメータは GIF の Plotter でだけ有効である。"yes" に設定すると erase で行われる処理が変わる。erase の実行は、その最初のときを除いて、一つの疑似 GIF ファイルに出力される複数の一連の画像の区切りとして扱われる。"no" に設定すると、他のリアルタイム描画を行わない Plotter における場合と同様に、描画途中の図形を背景色で塗りつぶすことで消去する。"no" に設定されている場合、一つの画像につき一つの疑似 GIF ファイルが出力される。

GIF_DELAY
: (デフォルト値は "0") このパラメータは GIF の Plotter でだけ有効である。アニメーション疑似 GIF における各コマの表示時間を 1/100 秒単位で指定する。指定できる範囲は"0"...."65535" である。

GIF_ITERATIONS
: (デフォルト値は "0") このパラメータは GIF の Plotter でだけ有効である。アニメーション疑似 GIF において、画像が最後まで表示された後に冒頭からまた繰り返す回数を指定する。指定できる範囲は "0"...."65535" である。

HPGL_ASSIGN_COLORS
: (デフォルト値は "no") このパラメータは HP-GL の Plotter で HPGL_VERSION が "2" のときにだけ有効である。"no" を指定するとパラメータ HPGL_PENS であらかじめ設定されている色数のペンだけが使えるようになる。"yes" にすると、使えるペンの色は HPGL_PENS による制約を受けず、必要に応じて設定される 1...31 番の範囲の番号で指定される「論理ペン」に色を設定できる。カラーの LaserJet プリンタおよび DesignJet プロッタ以外では、HP-GL/2 をサポートするデバイスでもこの「論理ペン」機能が使える機種は多くないため、この機能の利用には注意を払う必要がある。

HPGL_OPAQUE_MODE
: (デフォルト値は "yes") このパラメータは HP-GL の Plotter で HPGL_VERSION が "2" のときにだけ有効である。"yes" を設定すると HP-GL/2 出力は透明モードから不透明モードに切り替わる。これにより、白色あるいは他の不透明な色による図形の塗りつぶしができるようになる。同様に、0 番のペン を使うことで白色の線を描けるようになる。HP-GL/2 をサポートしている機器でも必ずしもこの不透明モードや 0 番のペンによる白線が使える訳ではない。特に HP-GL/2 でも、ペン・プロッタでは無効である。また古い HP-GL/2 の機器には不透明モードに切り替えるときに不都合があるものがある。HP-GL の Plotter の出力をそういった機器に使うときは、このパラメータを "no" にしておくとよい。

HPGL_PENS
: (デフォルト値は HPGL_VERSION が "1.5" または "2" のとき "1=black:2=red:3=green:4=yellow:5=blue:6=magenta:7=cyan"、HPGL_VERSION が "1" のとき"1=black") このパラメータは HP-GL の Plotter でだけ有効である。利用できるペンを設定する。設定の仕方は、デフォルト値を見て分かる通りである。1...31 番の各ペンに色を指定できる。指定できる色名については Appendix B [Color Names], page 158 を参照のこと。ペン 1 の指定は、省略してはならない。しかしかならずしも黒でなくてもよい。2...31 番の他のペンの指定は省略してもよい。

HPGL_ROTATE
: (デフォルト値は "0") このパラメータは HP-GL の Plotter でだけ有効である。用紙上の描画領域 (viewport) を、指定された角度 (度数) だけ回転させて置く。指定できる角度は "0"、"90"、"180"、"270" であり、"0" と "90" の代わりに "no" と "yes" でも指定できる。"180" と "270" は HPGL_VERSION が "2" のときにだけ有効である。

  このパラメータは ROTATION による指定とは異なる。ROTATION パラメータでは、描画領域そのものはそこにあるままで回転しないが、HPGL_ROTATE では描画領

域に対して回転と、用紙と描画領域の左下隅同士の間の距離を指定できる。これは HP-GL 言語の機能である。

このパラメータを使うことで、用紙の向き (ポートレート／ランドスケープ) を簡単に指定できることになる。デフォルトの用紙の向きは、HP-GL のプロッタでは横向き (ランドスケープ)、プリンタでは縦 (ポートレート) ことがあるので、出力機器によってこのパラメータの設定を変えればよい。

`HPGL_VERSION`

(デフォルト値は "2") このパラメータは HP-GL の Plotter でだけ有効である。"1" は元々の HP-GL 形式、"1.5" は HP7550A グラフィクス・プロッタや HP758x、HP7595A、HP7596A といった製図用プロッタ (いくつかの HP-GL/2 拡張機能を備えた HP-GL プロッタ)、"2" はもっと新しい HP-GL/2 機器に対応した形式で出力を行うことを指定する。この値が "2" よりも小さい場合は、使えるフォントはベクトル・フォントのみ、描ける線の太さはデフォルトの太さのみになるため、`linewidth`、`capmod`、`joinmod`、`fmiterlimit` による操作は無効になる。また塗りつぶし判定の方法である 'nonzero winding number rule' がサポートされないため、`fillmod` も無効である。さらに `HPGL_VERSION` が "1" の場合は、任意のパスの内側の塗りつぶしができなくなる (座標軸上の円と長方形の内部の塗りつぶしだけができる)。

`INTERLACE`

(デフォルト値は "no") このパラメータは PNG と GIF の各 Plotter でだけ有効である。このパラメータを "yes" に設定すると、以降に生成されるファイルはインターレースになる。そうやって生成されたファイルは、多くのアプリケーションでインターレース・モード (ノンリニア・モード) で表示される。

`MAX_LINE_LENGTH`

(デフォルト値は "500") 描画されるまでに一本のパスを構成できる連結点の数の最大数を設定する。描画されるときにはパスは自動的に (特に何の通知もなく) 複数のパスに分割される。パスに塗りつぶしが行われるときには、分割は行われない。

このパラメータは Tektronix と metafile 以外のどの Plotter でも有効である。このパラメータは、出力デバイスによっては描画用のバッファのサイズが限られている (たとえば PostScript プリンタや HP-GL のペン・プロッタなど) ために用意されている。Tektronix と metafile の Plotter はリアルタイムで描画を行い、したがってバッファを持たないため、このパラメータは無効である。

`META_PORTABLE`

(デフォルト値は "no") このパラメータは metafile の Plotter でだけ有効である。"yes" に設定しておくと、人間に読める形式で画像データが出力される。デフォルトではバイナリ形式である。Appendix D [Metafiles], page 161 参照。

`PCL_ASSIGN_COLORS`

(デフォルト値は "no") このパラメータは PCL の Plotter でだけ有効である。"no" に設定しておくと、あらかじめ設定されている限られた色数のペンだけが使える。"yes" にしておくとこの制限はなくなり、必要なら様々な色を「論理ペン」に設定できる。カラーの LaserJet プリンタ以外では、PCL 5 をサポートしていてもこの機能が使えるものは多くない。したがってこの機能を使うときには注意が必要である。

PCL_BEZIERS
: (デフォルト値は "yes") このパラメータは PCL の Plotter でだけ有効である。"yes" にしておくとベジェ曲線を描くときにそれに特化した命令を使う。"no" のときにはその命令は使われず、ベジェ曲線は多角形で近似して描かれる。ヒューレット・パッカードの PCL 5 プリンタは、初代の LaserJet III を除いてすべての機種でベジェ描画命令をサポートしている。

PNM_PORTABLE
: (デフォルト値は "no") このパラメータは PNM の Plotter でだけ有効である。"yes" に設定しておくと、人間に読める形式で画像データが出力される。デフォルトではバイナリ形式である。PBM/PGM/PPM の画像形式は、バイナリ形式でも機種依存性がないように設計されていて十分に portable なので、このパラメータ名はあまり適切ではない。

TERM
: (デフォルト値は NULL) このパラメータは Tektronix の Plotter でだけ有効である。この値を "xterm"、"nxterm"、"kterm" のいずれかで始まる文字列にしておくと、プログラムが現在、X Window システムの VT100 端末エミュレータの `xterm`、`nxterm`、`kterm` 上で実行されている、と解釈される。描画をはじめる前に Tektronix の Plotter はエスケープ・シーケンスを送って、普段は表に出ていない、Tektronix 描画のためのウィンドウを前面にポップアップする。描画が終了すると、制御を Tektronix のウィンドウから実行中の VT100 に戻すためのエスケープ・シーケンスが送られる。Tektronix のウィンドウは消去されず、そのまま画面上に残る。

  このパラメータを "kermit"、"ansi.sys"、"nansi.ssys" のいずれかにしておくと、プログラムが現在、MS-DOS 版 `kermit` の VT100 端末エミュレータ上で実行されている、と解釈される。描画をはじめる前に Tektronix の Plotter はエスケープ・シーケンスを送って、端末エミュレータを Tektronix モードに切り替える。また `kermit` 特有の Tektronix 制御コードも送られる。普通は色は使えないが、この Tektronix モードでは制限はあるが色が使える (`ansi.sys` の 16 色が使える)。同様に、普通は使えない線種 "dotdotdashed" も使える。描画が終了すると、Tektronix モードから VT100 モードにに戻すためのエスケープ・シーケンスが送られる。キーボードから 'ALT' - '-' を入力すると、VT100 モードと Tektronix モードを切り替えることができる。

TRANSPARENT_COLOR
: (デフォルト値は "none") このパラメータは PNG と GIF の各 Plotter でだけ有効である。有効な色名を指定しておくと、出力ファイル内でその色の部分が透明色として設定され、多くのソフトウェアでその部分が透明として表示される。指定できる色名については、Appendix B [Color Names], page 158 を参照のこと。

  `TRANSPARENT_COLOR` を設定してアニメーション疑似 GIF を生成すると、次の静止画を表示するときに 'restore to background' が適用され、前の静止画が背景色で塗りつぶされる。そうでなければ 'unspecified' が適用され、前の静止画の上に直接次の静止画が描かれる。

USE_DOUBLE_BUFFERING
: (デフォルト値は "no") このパラメータは X と X Drawable の Plotter でだけ有効である。これを "yes" に設定しておくと、描画の際に二重バッファリングが有効になる。画像ページ内の各フレームは `openpl...closepl` の間に描かれたものになるが、これがディスプレイに直接ではなく、一旦バッファ内に描かれる。

そして erase 操作でフレームの描画が終了したとき、または closepl が行われたときにバッファ内の画像の各ピクセルがその Plotter のディスプレイにすべてコピーされる。連続する画像が似ていれば、これによりアニメーションがより滑らかに表示できる。

X のディスプレイには二重バッファリングのためのハードウェアを備えているものがある。それが使えるときには X の Plotter はそれを検知して X11 プロトコルの拡張機能 (DBE または MBX) を利用して描画を行う。この場合、高価なグラフィック・カードなどを使っていると、アニメーションの再生が非常に速くなってしまうことがある。

VANISH_ON_DELETE
: (デフォルト値は "no") このパラメータは X の Plotter でだけ有効である。"yes" にしておくと、Plotter が破棄されるときに、画像が描画されていたウィンドウが閉じられる。"yes" でなければ、ウィンドウ内でキーボードから 'q' を入力するか、マウスクリックをしない限り、ウィンドウは消えない。

XDRAWABLE_COLORMAP
: (デフォルト値は NULL) このパラメータは X Drawable の Plotter でだけ有効である。このパラメータが NULL でなければ、それは定義されている色が保持されているカラーマップへのポインタ Colormap * であると見なされる。NULL の場合は X のディスプレイのデフォルトのカラーマップが使われる。

XDRAWABLE_DISPLAY
: (デフォルト値は NULL) このパラメータは X Drawable の Plotter でだけ有効である。描画対象となる X のディスプレイ へのポインタ Display * とみなされる。

XDRAWABLE_DRAWABLE1
XDRAWABLE_DRAWABLE2
: (デフォルト値は NULL) このパラメータは X Drawable の Plotter でだけ有効である。設定されていれば、出力先へのポインタ Drawable * と見なされる。出力先 'drawable' は、ウィンドウかまたはビットマップ画像かのどちらかである。X Drawable の Plotter が生成されるときに、この二つのパラメータのうち、少なくともどちらか一方は設定されていなければならない。

  XDrawable の Plotter は二つの出力先に同時に描画できるようになっている。X のウィンドウとビットマップ画像ファイルの両方で同時に同じ画像が生成できると、便利なこともあるからである。出力先を二つ指定するときは、その描画領域の大きさ、深度が同じでなければならない。また X のディスプレイは同じディスプレイでなければならない。

XDRAWABLE_VISUAL
: (デフォルト値は NULL) このパラメータは X Drawable の Plotter でだけ有効である。設定されていれば、カラーマップ (上述) の「visual」へのポインタ Visual * であると解釈される。このパラメータはかならずしも設定する必要はないが、XDRAWABLE_COLORMAP を指定するときにはこちらも指定するとよい。環境によっては、X drawable に対する色の設定 (color cell allocation) が高速になることがある。

X_AUTO_FLUSH

(デフォルト値は "yes") このパラメータは X の Plotter でだけ有効である。この値を "yes" に設定すると各描画操作が行われるたびに XFlush が行われるようになる。これにより各描画操作後すぐに X Window システムの画面に実際に描画が行われ、画面上に見えるようになる。しかしこれにより描画の速度はかなり遅くなる。このパラメータを "no" に設定すると、リアルタイムで画面に描画されなくなる分、描画は速くなる。

Appendix A フォント、文字列、記号

ベクトル画像を操作するためのライブラリである `libplot` と `graph`、`plot`、`pic2plot`、`tek2plot`、`plotfont` などのこれを利用するソフトウェアでは、文字列を描画するのに様々なフォントを利用できる。一つのタイプフェイス中で複数のフォントや、上付き、下付き文字や、ルート記号などを混在させることができる。また多種の記号も描画できる。以下にその使い方を述べる。

A.1 利用できるフォント

ベクトル画像を操作するためのライブラリである `libplot` と `graph`、`plot`、`pic2plot`、`tek2plot`、`plotfont` などのこれを利用するソフトウェアでは、文字列を描画するのに様々なフォントを利用できる。利用できるのは、22 種類の Hershey ベクトル・フォント、35 種類の PostScript フォント、45 種類の PCL 5 フォント、18 種類のヒューレット・パッカードのベクトル・フォントである。この合計 120 種類のフォントをここでは「組み込みフォント」と呼ぶ。Hershey フォントは、Dahlgren, VA にある米 Naval Surface Weapons Center で Dr. Allen V. Hershey が 1967 年に電子化した、線分を組み合わせて作られたフォントである。35 種類の PostScript フォントは現在どの PostScript プリンタにも搭載されているアウトライン・フォントである。45 種類の PCL 5 フォントは現在どの LaserJet プリンタ、プロッタにも搭載されている、ヒューレット・パッカードのアウトライン・フォントである (ただし初代の PCL 5 プリンタである LaserJet III では、この 45 種類のうち 8 種類、Univers と CGTimes しか使えない)。18 種類のヒューレット・パッカードのベクトル・フォントは、それより新しいヒューレット・パッカードのプリンタとプロッタに搭載されている。

Hershey フォントは `libplot` に実装されているすべての Plotter で利用できる。PostScript フォントは X、SVG、Illustrator、PostScript、Fig、CGM の各 Plotter で利用できる。つまりたとえば `graph` ではどの出力形式のときにも Hershey フォントが使え、`graph -T X`、`graph -T svg`、`graph -T ai`、`graph -T ps`、`graph -T cgm`、`graph -T fig` のときに PostScript フォントが使える。PCL 5 フォントは SVG、Illustratos、PCL、HP-GL の各 Plotter で、つまり `graph -T svg`、`graph -T ai`、`graph -T pcl`、`graph -T hpgl` で使える。ヒューレット・パッカードのベクトル・フォントは PCL と HP-GL の Plotter で、つまり `graph -T pcl` と `graph -T hpgl` で使える。X の Plotter と `graph -T X` では組み込みの Hershey と PostScript の両フォントに加えて、X Window システムのフォントが利用できる。

`plotfont` コマンドを使えば、'`-T`' で指定する出力形式において利用できるフォントのキャラクタ・マップが表示できる。Chapter 6 [plotfont], page 51 参照。

文字列を描画する時 (Section A.4 [Text String Format], page 144 参照)、120 種類の組み込みフォントはそれぞれ書体 (タイプフェイス) で分けられる。以下の表に示すように、各書体は複数のフォントからなり、各フォントに番号をつけてある。フォント 1 は通常のフォント、フォント 2 は斜体 (イタリック)、フォント 3 は太字 (ボールド)、フォント 4 は太斜体 (ボールド・イタリック) である。他の書体があればそれには 5 番以降の番号を付けてある。

22 種類の Hershey ベクトル・フォントには以下の書体がある。

- HersheySerif
  1. HersheySerif
  2. HersheySerif-Italic
  3. HersheySerif-Bold
  4. HersheySerif-BoldItalic

5. HersheyCyrillic
6. HersheyCyrillic-Oblique
7. HersheyEUC

- HersheySans
  1. HersheySans
  2. HersheySans-Oblique
  3. HersheySans-Bold
  4. HersheySans-BoldOblique
- HersheyScript
  1. HersheyScript
  2. HersheyScript
  3. HersheyScript-Bold
  4. HersheyScript-Bold
- HersheyGothicEnglish
- HersheyGothicGerman
- HersheyGothicItalian
- HersheySerifSymbol
  1. HersheySerifSymbol
  2. HersheySerifSymbol-Oblique
  3. HersheySerifSymbol-Bold
  4. HersheySerifSymbol-BoldOblique
- HersheySansSymbol
  1. HersheySansSymbol
  2. HersheySansSymbol-Oblique

Symbol フォントを除くほとんどの Hershey フォントのエンコーディングは ISO-Latin-1 である (ASCII コードの上位互換のエンコーディング)。Symbol フォントはギリシャ文字と数学の記号からなる。そのエンコーディングや使い方の詳細は *PostScript Language Reference Manual* にある。`libplot` では、Hershey の各書体には 0 番のフォントとして symbol フォントが含めてある (HersheySerifSymbol および HersheySansSymbol)。

Symbol 以外の Hershey 書体で、HersheyCyrillic、HersheyCyrillic-Oblique、HersheyEUC (最後のは日本語のフォント) の各書体はエンコーディングが ISO-Latin-1 encoding ではない。これらについては Section A.2 [Cyrillic and Japanese], page 141 を参照のこと。

35 種類の PostScript フォントには以下の書体がある。

- Helvetica
  1. Helvetica
  2. Helvetica-Oblique
  3. Helvetica-Bold
  4. Helvetica-BoldOblique
- Helvetica-Narrow

    1. Helvetica-Narrow
    2. Helvetica-Narrow-Oblique
    3. Helvetica-Narrow-Bold
    4. Helvetica-Narrow-BoldOblique
- Times
    1. Times-Roman
    2. Times-Italic
    3. Times-Bold
    4. Times-BoldItalic
- AvantGarde
    1. AvantGarde-Book
    2. AvantGarde-BookOblique
    3. AvantGarde-Demi
    4. AvantGarde-DemiOblique
- Bookman
    1. Bookman-Light
    2. Bookman-LightItalic
    3. Bookman-Demi
    4. Bookman-DemiItalic
- Courier
    1. Courier
    2. Courier-Oblique
    3. Courier-Bold
    4. Courier-BoldOblique
- NewCenturySchlbk
    1. NewCenturySchlbk-Roman
    2. NewCenturySchlbk-Italic
    3. NewCenturySchlbk-Bold
    4. NewCenturySchlbk-BoldItalic
- Palatino
    1. Palatino-Roman
    2. Palatino-Italic
    3. Palatino-Bold
    4. Palatino-BoldItalic
- ZapfChancery-MediumItalic
- ZapfDingbats
- Symbol

ZapfDingbats と Symbol フォントを除く PostScript フォントのエンコーディングは ISO-Latin-1 である。ZapfDingbats と Symbol フォントのエンコーディングは *PostScript Language Reference Manual* にある。`libplot` では PostScript の各書体には番号 0 番のフォントとして symbol フォントが含めてある。

45 種類の PCL 5 フォントには以下の書体がある。

- Univers
  1. Univers
  2. Univers-Oblique
  3. Univers-Bold
  4. Univers-BoldOblique
- UniversCondensed
  1. UniversCondensed
  2. UniversCondensed-Oblique
  3. UniversCondensed-Bold
  4. UniversCondensed-BoldOblique
- CGTimes
  1. CGTimes-Roman
  2. CGTimes-Italic
  3. CGTimes-Bold
  4. CGTimes-BoldItalic
- Albertus
  1. AlbertusMedium
  2. AlbertusMedium
  3. AlbertusExtraBold
  4. AlbertusExtraBold
- AntiqueOlive
  1. AntiqueOlive
  2. AntiqueOlive-Italic
  3. AntiqueOlive-Bold
- Arial
  1. Arial-Roman
  2. Arial-Italic
  3. Arial-Bold
  4. Arial-BoldItalic
- ClarendonCondensed
- Coronet
- Courier
  1. Courier
  2. Courier-Italic

3. Courier-Bold
4. Courier-BoldItalic

- Garamond
  1. Garamond
  2. Garamond-Italic
  3. Garamond-Bold
  4. Garamond-BoldItalic
- LetterGothic
  1. LetterGothic-Roman
  2. LetterGothic-Italic
  3. LetterGothic-Bold
  4. LetterGothic-BoldItalic
- Marigold
- CGOmega
  1. CGOmega-Roman
  2. CGOmega-Italic
  3. CGOmega-Bold
  4. CGOmega-BoldItalic
- TimesNewRoman
  1. TimesNewRoman
  2. TimesNewRoman-Italic
  3. TimesNewRoman-Bold
  4. TimesNewRoman-BoldItalic
- Wingdings
- Symbol

Wingdings と Symbol フォントを除く PCL 5 フォントのエンコーディングは ISO-Latin-1 である。Symbol フォントのエンコーディングは 'symbol font encoding' であり、これは *PostScript Language Reference Manual* に詳しく説明されている。`libplot` に PCL の各書体には番号 0 番のフォントとして symbol フォントが含めてある。

18 種類のヒューレット・パッカードのベクトル・フォントには以下の書体がある。

- Arc
  1. Arc
  2. Arc-Oblique
  3. Arc-Bold
  4. Arc-BoldOblique
- Stick
  1. Stick
  2. Stick-Oblique
  3. Stick-Bold

4. Stick-BoldOblique

- ArcANK
  1. ArcANK*
  2. ArcANK-Oblique*
  3. ArcANK-Bold*
  4. ArcANK-BoldOblique*
- StickANK
  1. StickANK*
  2. StickANK-Oblique*
  3. StickANK-Bold*
  4. StickANK-BoldOblique*
- ArcSymbol*
- StickSymbol*

名前にアスタリスクが付いているヒューレット・パッカードのベクトル・フォント (ANK と Symbol フォント) は、HP-GL/2 形式の出力を行うときか、または HP7550A グラフィクス・プロッタ、HP758x、HP7595A、HP7596A 製図プロッタのいずれかに HP-GL 出力を行うときにだけ有効である。そしてこれらは `HPGL_VERSION` が `"2"` または `"1.5"` のときに有効である。ANK フォントは日本語フォントである (Section A.2 [Cyrillic and Japanese], page 141 参照)。Symbol フォントには様々な数学記号が含まれている。

ANK と symbol 以外のすべてのヒューレット・パッカードのベクトル・フォントのエンコーディングは ISO-Latin-1 である。Arc フォントはプロポーショナル・フォント (可変幅)、Stick フォントは固定幅である。出力形式が HP-GL/2 か HP-GL のとき、Arc フォントでは出力機器に備わっている機能によりカーニングが行われる。しかし PCL 5 形式のときは、出力機器ではカーニングは行われないと `libplot` では想定している。ヒューレット・パッカードは PCL 5 機器での HP-GL/2 エミュレーション機能におけるカーニング処理機能を削除している。ヒューレット・パッカードのベクトル・フォントと、そのカーニング処理 (少なくともペンプロッタの場合) については、*Hewlett–Packard Journal* の Nov. 1981 発行の号に載っている L. W. Hennessee et al. の論文を参照されたい。

`libplot` がサポートするフォントでは、少なくともサポートする範囲では、PostScript、PCL 5、ヒューレット・パッカードのベクトル・フォントのどれにも、リガチャーは入っていない。しかし 22 種類の Hershey フォントのうち、6 種類にはリガチャーがある。フォントが HersheySerif または HersheySerif-Italic のときに `"fi"`、`"ff"`、`"fl"`、`"ffi"`、`"ffl"` のどれかが文字列中に出てきたら、それは自動的にリガチャーに置き換えられる (HersheyCyrillic と HersheyEUC も、印字可能 ASCII 文字については (ほとんど) HersheySerif と同じと考えられる)。さらに `"tz"` と `"ch"` も HersheyGothicGerman ではリガチャーになる。ドイツ語のエスツェット character 'ß' はどのフォントでもリガチャーにはならない。エスツェットを描画するには、エスケープ・シーケンスで `"\ss"` とすればよい (Section A.4 [Text String Format], page 144 参照)。またはドイツ語キーボードがあるなら直接エスツェットを入力してもよい。

A.2 キリル文字と日本語

前節の組み込みフォントには、キリル文字と日本語の文字のベクトル・フォントも含まれている。それらのエンコード、つまりキャラクタ・マップ上でどの文字がどこにあるのかを、以下に説明する。キリル文字と日本語の文字のベクトル・フォントを含むすべてのフォントに

ついて、そのキャラクタ・マップを `plotfont` コマンドで表示することができる。Chapter 6 [plotfont], page 51 参照。

HersheyCyrillic と HersheyCyrillic-Oblique フォントのエンコーディングは KOI8-R である。これは ASCII コードの上位互換であり、ソ連時代から Unix 上のアプリケーションのデファクト・スタンダードである。これは、印字可能な ASCII 文字に限っては HersheySerif ベクトル・フォントと同じであるが、フォントの後半は異なっている。`0xc0...0xdf` の範囲の文字はキリル文字の小文字、`0xe0...0xff` は大文字である。その他のキリル文字が `0xa3` および `0xb3` にある。詳細は the official KOI8-R Web page (`http://koi8.pp.ru/main.html`) および Information Sciences Institute (`http://www.isi.edu`) にある RFC 1489 を参照されたい。

HersheyEUC は日本語を表示するためのベクトル・フォントである。これは 8 ビットの EUC-JP でエンコードされている。EUC とは 'extended Unix code' のことで、他の多バイト文字 (ギリシャ文字やキリル文字) でも使われている。EUC 文字列の詳細に付いては Ken Lunde の *Understanding Japanese Information Processing* (O'Reilly, 1993) を参照のこと。この本には日本語のテキスト処理についての多くの情報が載っている。この本の追加情報 (`http://www.praxagora.com/lunde/cjk_inf.html`) と同じ著者によるより新しい本 *CJKV Information Processing* (O'Reilly, 1999) も参照されたい。

HersheyEUC では印刷可能な ASCII 文字は `0x20...0x7e` の範囲であり、HersheySerif と同様である (HersheyEUC のエンコーディングは 'JIS Roman' と呼ばれ、これは日本工業規格 JIS に定められた標準規格である)。また JIS X0208 では `0xa1...0xfe` の範囲内の連続した 2 バイトで一つの文字を表している。JIS X0208 には平仮名、カタカナ、漢字、ローマ字、ギリシャ文字、キリル文字、句読点類、その他の記号が定義されている。例えば 83 個の平仮名 (濁点、半濁点、小文字なども含めて) が `0x2421...0x2473` にマッピングされている。JIS X0208 に定義されている文字の EUC コードを得るには、各バイトに `0x80` を加えればよい (つまり各バイトの最上位ビットを立てればよい)。したがって 83 個の平仮名のうち最初の文字 (`0x2421`) は `0xa4` と `0xa1` の連続する 2 つのバイトで表される。

HersheyEUC フォントでの JIS X0208 の実装は Dr. Hershey の電子化によるもので、実用上まったく問題なく使える。83 個の平仮名と 86 個のカタカナのすべてが使えるが、あまり使う機会のない「半角カナ」はサポートしていない。使える漢字は 603 個であり、そのうちの 596 個は JIS 第一水準に定められている (つまりよく使われる) 2965 個から選ばれている。平仮名、カタカナ、漢字の幅はすべて同じである。'`kanji.doc`' ファイルにこの 603 個の漢字を列挙してある。このファイルは多くの場合、'`/usr/share/libplot`' または '`/usr/local/share/libplot`' にインストールされている。JIS X0208 のうち利用できない漢字は 'undefined character' グリフ (複数の横線) で描画される。

ArcANK 書体と StickANK 書体の 8 種類のヒューレット・パッカードのベクトル・フォントでも日本語が使える。HP-GL/2 出力、または HP7550A グラフィクス・プロッタ、HP758x、HP7595A、HP7596A 製図プロッタのいずれかに HP-GL 出力を行うときに有効である。これは `HPGL_VERSION` が `"2"` か `"1.5"` のときに有効である。

ANK とは Alphabet、Numerals、Katakana のことであり、ANK フォントでは複数のエンコーディングが使われている。各フォントの前半は JIS Roman エンコーディングで、後半が半角カナである。この「半角カナ」には、簡略化された形のカタカナと、いくつかの分音記号 (句読点、濁点、半濁点、カッコなど) が含まれる。分音記号はそれぞれ独立した文字として含まれている。

A.3 X Window フォント

`libplot` を組み込んでいる `graph -T X`、`plot -T X`、`pic2plot -T X`、`tek2plot -T X`、`plotfont -T X` の各コマンドは、X Window で用意されている多種多様なフォントを利用することができる。22 種類の Hershey ベクトル・フォントも使える。35 種類の PostScript フォントも、X で使えるように設定されていれば使える。プロットを行うプログラムの多くが、フリーに配布されている 35 種類の Type 1 形式の PostScript フォントを組み込んでいる。これは X display にインストールするのも容易である。

実際には、利用中の X display で使えるフォントはほとんど使うことができる。XLFD (X Logical Font Description) 名が付いているスケーラブル・フォントも使える。システム内部の情報を表示する `xlsfonts` コマンドを使えばフォント名を調べることができる。フォント名の末尾が"-0-0-0-0-p-0-iso8859-1" または "-0-0-0-0-m-0-iso8859-1" のフォントは ISO-Latin-1 エンコーディングのスケーラブル・フォントで、`libplot` とこれを組み込んだプログラムから利用できる。たとえば "CharterBT-Roman" フォントは多くの X display で使えるが、このフォントの XLFD 名は正確には"-bitstream-charter-medium-r-normal–0-0-0-0-p-0-iso8859-1" である。`libplot` を組み込んでいるプログラムではこれをハイフンが 3 つしかない短縮形の "charter-medium-r-normal" で認識できる。以下のコマンド

```
echo 0 0 1 1 2 0 | graph -T X -F charter-medium-r-normal
```

は X window の画面にウィンドウを開き、両座標軸と目盛りのラベルをこのフォントで描画する。

フォント名の末尾が "iso8859-2" や "adobe-fontspecific" のものも使えるが、これらのエンコーディングは標準の ISO-Latin-1 ではない。`libplot` はデフォルトでは "iso8859-1"、つまり指定されるフォントのうち ISO-Latin-1 エンコーディングのものがあれば、それを使う。また、正式なフォント名をフルネームで指定することで、ISO-Latin-1 エンコーディングを明示的に指定してもよい。"fixed" や "9x15" などの画面描画に使われる X のフォント (ビットマップフォントなど) を使いたいときも、同様に正確な名前でフォントを指定すればよい。XFLD 名ではなくフルネームで指定しておくことで、フォントのグリフのビットマップ・パターンを一度 X のディスプレイから取得すれば、`libplot` 側で適切なサイズに変更されるようになる。

`graph` などの GNU plotting utilities のコマンドでは '`--bitmap-size`' が使える。'`-T X`' が指定されているときは、このオプションにより表示されるウィンドウのサイズが指定される。また他の視覚効果を得ることもできる。ploting utilities の各コマンドでは正方形の描画領域が想定されている。そこでたとえば '`--bitmap-size 800x400`' を指定すると、縦と横で違う割合でスケーリングが行われ、描画される画像は横に引き延ばされる。画像中に X のフォントが使われていれば、同様に縦と横で違った割合で引き延ばされる。同じような考え方で、'`--rotation`' オプションを指定すると画像が回転させられ、文字列も同様に回転させられる。たとえば '`--rotation 45`' を指定すると反時計回りに 45 度回転する。'`--bitmap-size`' と '`--rotation`' は同時に指定してもよい。

ISO-Latin-1 の X Window システムのフォントを使うとき、エスケープ・シーケンスを使えば組み込み ISO-Latin-1 フォントに含まれる ASCII でない 8 ビット文字を表示させることができる。使えるエスケープ・シーケンスについては Section A.4 [Text String Format], page 144 を参照のこと。例えば "\Po" と文字列に入れるとイギリスの通貨のポンド記号 symbol '£' が表示される。以下のコマンド

```
echo 0 0 1 1 | graph -T X -F times-medium-r-normal -L "A \Po1 Plot"
```

を実行してみると、この記号がプロット内のラベルに使えるのが分かる。同様にして、使っているフォントが ISO-Latin-1 でもそうでなくても、どの X Window システムのフォントで

も、エスケープ・シーケンスを使えば数学記号やギリシャ文字が表示できる。ほとんどすべての X ディスレプイの Symbol フォント中の記号が使える。

A.4 文字列とエスケープ・シーケンス

`libplot` やこれを組み込んでいる `graph`、`plot`、`pic2plot`、`tek2plot`、`plotfont` が描画する文字列は、印字可能な文字でなければならない。改行文字 (NL および CR) などの制御コードは文字列に使ってはならない。印刷可能な文字というのは `0x20...0x7e` および `0xa0...0xff` のいずれかの範囲内にある文字のことである。前者はいわゆる印字可能な ASCII 文字、後者はいわゆる印字可能な 8 ビット文字である。

しかし文字列には「エスケープ・シーケンス」を入れることができる。これによりフォント切り替えたり、文字を上付きや下付きにしたり、ASCII 文字でない数学記号などを描画することができる。ということはつまり、`graph` の描画における座標軸のラベルにもそういった文字が使えるということである。`pic2plot` がラベル・オブジェクト内に置く文字列も同様である。

エスケープ・シーケンスの指定法は、TEX、`troff`、`groff` などの文書整形システムをを使ったことのある人なら違和感なく理解できるだろう。エスケープ・シーケンスはそれぞれ、３文字からなる。１文字目はバックスラッシュである。以下に、よく使われるものを挙げる。

`"\sp"` 以降の文字を上付き文字にする (上付き文字モード開始)。

`"\ep"` 上付き文字モード終了。

`"\sb"` 以降の文字を下付き文字にする (下付き文字モード開始)。

`"\eb"` 下付き文字モード終了。

`"\mk"` その位置を覚えておく。

`"\rt"` 覚えておいた位置に戻る。

たとえば `"x\sp2\ep"` 文字列があったとすると、'x の二乗' として描画される。下付き文字の下付き、なども指定できる。下付き文字と上付き文字は `"\mk"` と`"\rt"` を使うとうまくそろえることができる。たとえば `"a\mk\sbi\eb\rt\sp2\ep"` とすると、a に下付きの i と上付きの 2 が付く。ベキ乗数の '2' は下付きの i の真上に描かれる。

同じ書体でフォントを切り替えるエスケープ・シーケンスもある。何に切り替えられるかについては Section A.1 [Text Fonts], page 136 を参照のこと。たとえば現在のフォントが Times-Roman、つまり 'Times' 書体のフォント 1 だとする。`"A \f2very\f1 well labeled axis"` とすると、この中の 'very' は、'Times' 書体のフォント 2 が Times-Italic であるため、斜体 Times-Italic になる。フォントを切り替えるエスケープ・シーケンスは `"\fn"` という形式で、*n* でどのフォントに切り替えるかを指定する。`groff` との互換性を持たせるため、`"\f1"`、`"\f2"`、`"\f3"` の代わりにそれぞれ `"\fR"`、`"\fI"`、`"\fB"` と指定してもいいようになっている。`"\fP"` を使うと、直前の切り替えの前のフォントに戻る (しかしその前のフォントにまでは戻れない)。異なる書体間の切り替えは現在のところ、できない。

横方向に移動するためのエスケープ・シーケンスが他にもいくつかあり、斜体とそうでないフォントの間をきれいにそろえることなどに使うことができる。`"\r1"`、`"\r2"`、`"\r4"`、`"\r6"`、`"\r8"`、`"\r^"` でそれぞれ右方向に 1 em、1/2 em、1/4 em、1/6 em、1/8 em、1/12 em だけ移動する。`"\l1"`、`"\l2"`、`"\l4"`、`"\l6"`、`"\l8"`、`"\l^"` も同様だが、これらは左に移動する。`"A \fIvery\r^\fP well labeled axis"` とすると、`"A \fIvery\fP well labeled axis"` よりは見栄えがいいはずである。

平方根の記号は、上述した "\mk" と "\rt" を組み合わせることで描画できる。平方根記号の最初の部分は "\sr" であり、上線 ('run') を描く"\rn" で好きなだけ記号を長くできる。たとえば "\sr\mk\rn\rn\rtab" とすると 'ab の平方根' が描画できる。上線の長さを調整したいときは、"\rn" の数をいくつか変えてみて、試行錯誤する必要があるだろう。

下線を引くには、"\ul" を好きな数だけ並べればよい。上で示した "\mk"...."\rt" の技が下線でも使える。したがって "\mk\ul\ul\ul\rtabc" とすると、"abc" に下線が引かれる。下線の長さを調整したいときは、"\ul" の数をいくつか変えてみて、試行錯誤する必要があるだろう。横方向の移動量も同様に、試行錯誤して決めねばならないだろう。たとえば"HersheySerif" を使っているときは、"\mk\ul\ul\l8\ul\rtabc" よりも"\mk\ul\ul\ul\rtabc" の方が見栄えがいいはずである。

上で説明したものの他に、ISO-Latin-1 フォント (Symbol フォント、HersheyCyrillic、HersheyEUC、ZapfDingbats 以外の組み込みフォント) の印刷可能な非 ASCII 文字を操作するためのエスケープ・シーケンスもある。アクセント記号付きの文字などは非 ASCII 文字である。それら `0xa0...0xff` の範囲にある 8 ビット文字は、文字列中に直接指定してよい。しかし利用中の端末で表示できないなどの場合には、直接入力する代わりにエスケープ・シーケンスで指定することもできる。

エスケープ・シーケンスは数学記号やギリシャ文字も同様に指定できる。Symbol フォントの文字は通常、エスケープ・シーケンスを使って指定する。使っているフォントが Hershey フォントかそうでないかによって、記号を描画するために使われるフォントが異なる。Hershey フォントを使っているときには HersheySerifSymbol か HersheySansSymbol が、そうでないときには Symbol フォントが使われる。

以下に、ISO-Latin-1 フォントを使っているときの非 ASCII 文字を指定するエスケープ・シーケンスを列挙する。各エスケープ・シーケンスについて ISO-Latin-1 エンコーディングでの対応する文字の番号 (十進数で) と正式な PostScript 名を示している。ほとんどはその名前を見れば何の記号か分かる。たとえば 'eacute' は小文字の 'e' に鋭アクセント符号がついたものである。

"\r!" [161] exclamdown

"\ct" [162] cent

"\Po" [163] sterling

"\Cs" [164] currency

"\Ye" [165] yen

"\bb" [166] brokenbar

"\sc" [167] section

"\ad" [168] dieresis

"\co" [169] copyright

"\Of" [170] ordfeminine

"\Fo" [171] guillemotleft

"\no" [172] logicalnot

"\hy" [173] hyphen

"\rg" [174] registered

"\a-" [175] macron
"\de" [176] degree
"\+-" [177] plusminus
"\S2" [178] twosuperior
"\S3" [179] threesuperior
"\aa" [180] acute
"*m" [181] mu
"\ps" [182] paragraph
"\md" [183] periodcentered
"\ac" [184] cedilla
"\S1" [185] onesuperior
"\Om" [186] ordmasculine
"\Fc" [187] guillemotright
"\14" [188] onequarter
"\12" [189] onehalf
"\34" [190] threequarters
"\r?" [191] questiondown
"\‘A" [192] Agrave
"\’A" [193] Aacute
"\^A" [194] Acircumflex
"\~A" [195] Atilde
"\:A" [196] Adieresis
"\oA" [197] Aring
"\AE" [198] AE
"\,C" [199] Ccedilla
"\‘E" [200] Egrave
"\’E" [201] Eacute
"\^E" [202] Ecircumflex
"\:E" [203] Edieresis
"\‘I" [204] Igrave
"\’I" [205] Iacute
"\^I" [206] Icircumflex
"\:I" [207] Idieresis

"\-D" [208] Eth
"\~N" [209] Ntilde
"\'O" [210] Ograve
"\'O" [211] Oacute
"\^O" [212] Ocircumflex
"\~O" [213] Otilde
"\:O" [214] Odieresis
"\mu" [215] multiply
"\/O" [216] Oslash
"\`U" [217] Ugrave
"\'U" [218] Uacute
"\^U" [219] Ucircumflex
"\:U" [220] Udieresis
"\'Y" [221] Yacute
"\TP" [222] Thorn
"\ss" [223] germandbls
"\`a" [224] agrave
"\'a" [225] aacute
"\^a" [226] acircumflex
"\~a" [227] atilde
"\:a" [228] adieresis
"\oa" [229] aring
"\ae" [230] ae
"\,c" [231] ccedilla
"\`e" [232] egrave
"\'e" [233] eacute
"\^e" [234] ecircumflex
"\:e" [235] edieresis
"\`i" [236] igrave
"\'i" [237] iacute
"\^i" [238] icircumflex
"\:i" [239] idieresis
"\Sd" [240] eth

"\~n" [241] ntilde

"\‘o" [242] ograve

"\’o" [243] oacute

"\^o" [244] ocircumflex

"\~o" [245] otilde

"\:o" [246] odieresis

"\di" [247] divide

"\/o" [248] oslash

"\‘u" [249] ugrave

"\’u" [250] uacute

"\^u" [251] ucircumflex

"\:u" [252] udieresis

"\’y" [253] yacute

"iTp" [254] thorn

"\:y" [255] ydieresis

フォントが HersheySerifSymbol か HersheySansSymbol (Hershey フォントの場合)、または Symbol (PostScript フォントの場合) のときの数学記号とギリシャ文字のエスケープ・シーケンスを以下に列挙する。各エスケープ・シーケンスについて symbol エンコーディングでの対応する文字の番号 (八進数で) と正式な PostScript 名を示している。ほとんどはその名前を見れば何の記号か分かる。たとえば "*a" は小文字のギリシャ文字のアルファである。*PostScript Language Reference Manual* に以下に挙げている文字の表がある。

"\fa" [0042] universal

"\te" [0044] existential

"\st" [0047] suchthat

"**" [0052] asteriskmath

"\=~" [0100] congruent

"*A" [0101] Alpha

"*B" [0102] Beta

"*X" [0103] Chi

"*D" [0104] Delta

"*E" [0105] Epsilon

"*F" [0106] Phi

"*G" [0107] Gamma

"*Y" [0110] Eta

"*I" [0111] Iota
"\+h" [0112] theta1
"*K" [0113] Kappa
"*L" [0114] Lambda
"*M" [0115] Mu
"*N" [0116] Nu
"*O" [0117] Omicron
"*P" [0120] Pi
"*H" [0121] Theta
"*R" [0122] Rho
"*S" [0123] Sigma
"*T" [0124] Tau
"*U" [0125] Upsilon
"\ts" [0126] sigma1
"*W" [0127] Omega
"*C" [0130] Xi
"*Q" [0131] Psi
"*Z" [0132] Zeta
"\tf" [0134] therefore
"\pp" [0136] perpendicular
"\ul" [0137] underline
"\rx" [0140] radicalex
"*a" [0141] alpha
"*b" [0142] beta
"*x" [0143] chi
"*d" [0144] delta
"*e" [0145] epsilon
"*f" [0146] phi
"*g" [0147] gamma
"*y" [0150] eta
"*i" [0151] iota
"\+f" [0152] phi1
"*k" [0153] kappa

"*l" [0154] lambda
"*m" [0155] mu
"*n" [0156] nu
"*o" [0157] omicron
"*p" [0160] pi
"*h" [0161] theta
"*r" [0162] rho
"*s" [0163] sigma
"*t" [0164] tau
"*u" [0165] upsilon
"\+p" [0166] omega1
"*w" [0167] omega
"*c" [0170] xi
"*q" [0171] psi
"*z" [0172] zeta
"\ap" [0176] similar
"\+U" [0241] Upsilon1
"\fm" [0242] minute
"\<=" [0243] lessequal
"\f/" [0244] fraction
"\if" [0245] infinity
"\Fn" [0246] florin
"\CL" [0247] club
"\DI" [0250] diamond
"\HE" [0251] heart
"\SP" [0252] spade
"\<>" [0253] arrowboth
"\<-" [0254] arrowleft
"\ua" [0255] arrowup
"\->" [0256] arrowright
"\da" [0257] arrowdown
"\de" [0260] degree
"\+-" [0261] plusminus

"\sd" [0262] second
"\>=" [0263] greaterequal
"\mu" [0264] multiply
"\pt" [0265] proportional
"\pd" [0266] partialdiff
"\bu" [0267] bullet
"\di" [0270] divide
"\!=" [0271] notequal
"\==" [0272] equivalence
"\~~" [0273] approxequal
"\.." [0274] ellipsis
NONE [0275] arrowvertex
"\an" [0276] arrowhorizex
"\CR" [0277] carriagereturn
"\Ah" [0300] aleph
"\Im" [0301] Ifraktur
"\Re" [0302] Rfraktur
"\wp" [0303] weierstrass
"\c*" [0304] circlemultiply
"\c+" [0305] circleplus
"\es" [0306] emptyset
"\ca" [0307] cap
"\cu" [0310] cup
"\SS" [0311] superset
"\ip" [0312] reflexsuperset
"\n<" [0313] notsubset
"\SB" [0314] subset
"\ib" [0315] reflexsubset
"\mo" [0316] element
"\nm" [0317] notelement
"\/_" [0320] angle
"\gr" [0321] nabla
"\rg" [0322] registerserif

"\co"	[0323] copyrightserif
"\tm"	[0324] trademarkserif
"\PR"	[0325] product
"\sr"	[0326] radical
"\md"	[0327] dotmath
"\no"	[0330] logicalnot
"\AN"	[0331] logicaland
"\OR"	[0332] logicalor
"\hA"	[0333] arrowdblboth
"\lA"	[0334] arrowdblleft
"\uA"	[0335] arrowdblup
"\rA"	[0336] arrowdblright
"\dA"	[0337] arrowdbldown
"\lz"	[0340] lozenge
"\la"	[0341] angleleft
"\RG"	[0342] registersans
"\CO"	[0343] copyrightsans
"\TM"	[0344] trademarksans
"\SU"	[0345] summation
NONE	[0346] parenlefttp
NONE	[0347] parenleftex
NONE	[0350] parenleftbt
"\lc"	[0351] bracketlefttp
NONE	[0352] bracketleftex
"\lf"	[0353] bracketleftbt
"\lt"	[0354] bracelefttp
"\lk"	[0355] braceleftmid
"\lb"	[0356] braceleftbt
"\bv"	[0357] braceex
"\eu"	[0360] euro
"\ra"	[0361] angleright
"\is"	[0362] integral
NONE	[0363] integraltp

NONE [0364] integralex

NONE [0365] integralbt

NONE [0366] parenrighttp

NONE [0367] parenrightex

NONE [0370] parenrightbt

"\rc" [0371] bracketrighttp

NONE [0372] bracketrightex

"\rf" [0373] bracketrightbt

"\RT" [0374] bracerighttp

"\rk" [0375] bracerightmid

"\rb" [0376] bracerightbt

最後に Hershey フォントを使っているときにだけ有効なエスケープ・シーケンスを挙げる。これを使うと、どのフォントにも入っていない特殊な記号や、他に指定方法がない記号を使うことができる。これには大きく分けて 2 種類ある。一つは雑多な記号類、もう一つは天文学や十二星座の記号である雑多な記号類を指定するためのエスケープ・シーケンスには、以下のものがある。

"\dd" daggerdbl

"\dg" dagger

"\hb" hbar

"\li" lineintegral

"\IB" interbang

"\Lb" lambdabar

"\~-" modifiedcongruent

"\-+" minusplus

"\||" parallel

"\s-" [variant form of s]

上で最後に挙げているエスケープ・シーケンス "\s-" は、記号ではなく文字を指定する。HersheyGothicGerman フォントでは、単語の末尾の小文字の 's' の形が若干変わってしまうために用意されている。たとえばドイツ語の "besonders" を正しく描画するためには、"besonder\s-" と指定せねばならない。Hershey の 2 種類の symbol フォントでギリシャ文字を使うときも同様である (ギリシャ語では、小文字の末尾の 's' は末尾でない 's' と形が違う)。末尾と末尾でない 's' の形が変わらない Hershey フォントでは、"s" と "\s-" は同じである。

以下に 12 種の星座の記号を含む天文学記号を表すためのエスケープ・シーケンスを以下に示す。これらも上の雑多な記号類と同様、使っているフォントが Hershey フォントのときにだけ有効である。

"\SO" sun

"\ME"	mercury
"\VE"	venus
"\EA"	earth
"\MA"	mars
"\JU"	jupiter
"\SA"	saturn
"\UR"	uranus
"\NE"	neptune
"\PL"	pluto
"\LU"	moon
"\CT"	comet
"\ST"	star
"\AS"	ascendingnode
"\DE"	descendingnode
"\AR"	aries
"\TA"	taurus
"\GE"	gemini
"\CA"	cancer
"\LE"	leo
"\VI"	virgo
"\LI"	libra
"\SC"	scorpio
"\SG"	sagittarius
"\CP"	capricornus
"\AQ"	aquarius
"\PI"	pisces

上に上げた雑多な記号類および天文学記号は、からなずしも、Hershey フォントを使っているときにだけ指定できる特殊なフォント外記号である、というわけではないが、Allen Hershey が電子化したすべてのグリフのライブラリを GNU `libplot` は搭載している。したがって文字列にはそれらを指定できる。指定できる Hershey グリフはそれぞれ、４桁の番号で示される。Hershey グリフ 1 は "\#H0001" と表される。標準の Hershey グリフを表す数値は "\#H0001" から"\#H3999" だが、その中には欠番もたくさんある。他の人による他のグリフが"\#H4000"..."\#H4194" の範囲で指定できる。日本語の仮名は "\#H4195"..."\#H4399" の範囲である。

"\#H0001"..."\#H3999" の範囲内の、ほとんどすべての Hershey グリフを記載した表を、米 National Technical Information Service (電話番号 +1 703 487 4650) に連絡すれば注文することができる。商品番号は PB251845 で、現在の価格は US$40 である。この４桁の番号による指定の例を挙げると、以下の文字列

```
"\#H0744\#H0745\#H0001\#H0002\#H0003\#H0869\#H0907\#H2330\#H2331"
```

を描画すると、三つ葉のクローバー、フラ・ダ・リ紋章、地図マークの A, B, C、ベル、大きな円、ト音記号、ヘ音記号が描かれる。繰り返すが、これは Hershey フォントを使っている場合である。

また Hershey フォントならどれを使っていても日本語の平仮名、カタカナ、漢字が使える。603 個の漢字はどれも使える。使えるのは HersheyEUC フォントと同じものである。日本語の文字は JIS X0208 に定めるタイポグラフィにしたがって番号が付いており、2 バイトで 1 文字を表す。GNU plotting utilities に付属している '`kanji.doc`' ファイルに利用できるすべての漢字を列挙してある。このファイルは多くの場合、'`/usr/share/libplot`' または '`/usr/local/share/libplot`' にインストールされている。

JIS X0208 の文字はどれも、たとえば "\#J357e" のような具合に、エスケープ・シーケンスで 2 バイトを 4 つの十六進数で表すことで指定できる。IS X0208 の文字を表すためには、2 バイトとも `0x21...0x7e` の範囲内でなければならない。漢字は "\#J3021" より上である。JIS X0208 に定められている文字はどれも同様に指定できる。たとえば平仮名とカタカナは"\#J2421"..."\#J257e" の範囲内であり、ローマ字は "\#J2321"..."\#J237e" である。'`kanji.doc`' ファイルと同じディレクトリにインストールされる '`kana.doc`' に平仮名とカタカナのエンコーディングを述べている。JIS X0208 の詳細に付いては Ken Lunde の *Understanding Japanese Information Processing* (O'Reilly, 1993) とこの本の追加情報 (`http://www.praxagora.com/lunde/cjk_inf.html`) を参照のこと。

昔から参照されている A. N. Nelson の *Modern Reader's Japanese-English Character Dictionary* による漢字の番号も利用できる (この辞典は C. E. Tuttle and Co. が出版しており、ISBN 0-8048-0408-7 である。1997 年に改訂版 (ISBN 0-8048-2036-8) が出ているが、番号付けが変わってしまっている)。この Nelson による番号で感じを指定するやり方も JIS X0208 の場合と同様だが、4 桁の十六進数ではなく、4 桁の十進数を使う。'`kanji.doc`' に JIS X0208 とネルソン番号の対応を述べている。たとえばネルソン番号 "\#N0001" の文字は JIS X0208 の "\#J306c" である。これは Unicode で利用できる漢字の番号でもある。

利用できる漢字の幅はすべて同じであり、平仮名、カタカナも同じ幅である。利用できない漢字は 'undefined character' グリフ (複数の横線) で描画される。JIS X0208 の漢字以外の利用できない文字でも同様である。

A.5 プロットに使える記号

GNU `libplot` ライブラリではプロットに使う記号に 1 から 31 まで番号を付け、標準で利用できるようにしている。ここで言う記号とは、プロット画像内でデータの点を表すために用いる、プロット用の記号のことである。`graph` プログラムに '`-S`' オプションを指定したときに、データの各点を描画するのに使われる。

文字の場合と同様に、記号にもフォントサイズがある。どの出力形式でも、フォントサイズが十分に大きければ目に見える大きさで表示される。しかし 0 番の記号は例外で、`libplot` では記号 0 番はどの記号も表さないものとしている。1...31 番として定義されている記号を以下にしめす。

1. dot (·)
2. plus (+)
3. asterisk (∗)
4. circle (○)
5. cross (×)

6. square
7. triangle
8. diamond
9. star
10. inverted triangle
11. starburst
12. fancy plus
13. fancy cross
14. fancy square
15. fancy diamond
16. filled circle
17. filled square
18. filled triangle
19. filled diamond
20. filled inverted triangle
21. filled fancy square
22. filled fancy diamond
23. half filled circle
24. half filled square
25. half filled triangle
26. half filled diamond
27. half filled inverted triangle
28. half filled fancy square
29. half filled fancy diamond
30. octagon
31. filled octagon

1 番から 5 番までの記号はよく知られた GKS (Graphical Kernel System) と同じである。

32 番以降の記号は、フォントマップ内でその番号が示す文字になる。`libplot` ではそれはそのときに使っているフォントである。`graph` ではそれは '`--symbol-font-name`' オプションで指定されるフォントである。デフォルトでは ZapfDingbats フォントだが、`graph -T png`、`graph -T pnm`、`graph -T gif`、`graph -T pcl`、`graph -T hpgl`、`graph -T tek` では PostScript フォントが使えないため、代わりに HersheySerif フォントが使われる。

ZapfDingbats フォントの文字にはプロット記号に適したものが多い。たとえば 74 番はテキサス・スター (Texas Star) である。以下のようにすると、

```
echo 0 0 1 2 2 1 3 2 4 0 | graph -T ps -m 0 -S 74 0.1 > plot.ps
```

5 点からなるデータが、線では結ばれずにプロットされ、PostScript 形式で出力される。各データ点は大きな (描画領域の 1/10 の大きさの) テキサス・スターで示される。

`graph -T pcl` または `graph -T hpgl` でそれらの記号を使いたいときは、PCL 5 や HP-GL/2 で使える Wingdings を試すとよい。以下のようにすると、

```
echo 0 0 1 2 2 1 3 2 4 0 |
    graph -T pcl -m 0 --symbol-font Wingdings -S 181 0.1 > plot.pcl
```

上の PostScript の例と同様の描画が PCL 5 形式で行われる。Wingdings フォントではテキサス・スターは 181 番である。

Appendix B 色名

GNU plotting utilities の多くのアプリケーションでは、色を名前で指定できる。それは `bgcolorname`, `pencolorname`, `fillcolorname` の各関数を、引数に色を渡して呼ぶことで可能である。

たとえば `graph` では '`--frame-color`' オプションで色名が使える。他のプログラムとでは '`--pen-color`' オプションが使えるものがあり (`graph` でも使える)、それが使えるものでは '`--bg-color`' オプションが使える。これらのプログラムが使っている `libplot` ライブラリには `pencolorname`、`fillcolorname`、`bgcolorname` の各機能がある。

`libplot` やこれを組み込んでいる `graph`、`plot`、`pic2plot`、`tek2plot`、`plotfont` では、コマンドラインでのオプション指定に色名を使うことができる。各コマンドではコマンドライン・オプションに '`--bg-color`' が使え、`graph` 以外では '`--pen-color`' も使える (`graph` では、より複雑な '`--pen-colors`' オプションに加え、'`--frame-color`' が使える)。

いずれの場合でも、16 進数で表現される 48 ビットカラーで正確に色を指定することができる。たとえば "#c0c0c0" は明るい灰色、"#ffffff" は白である。また色は色名でも指定できる。指定できる色名には "red"、"green"、"blue" などのはっきりしたものだけでなく"dark magenta"、"forest green"、"olive drab" などの曖昧な感じのものもあり、合計 665 種類の名前が使える。色名では大文字、小文字は区別されず、また空白文字も無視される。したがってたとえば "RosyBrown" と "rosy brown"、"DarkGoldenrod3" と"dark goldenrod 3" はそれぞれ同じと見なされる。

GNU plotting utilities に付属している '`colors.txt`' ファイルに利用できるすべての色名を列挙してある。このファイルは多くの場合、'`/usr/share/libplot`' または '`/usr/local/share/libplot`' にインストールされている。この色名はほとんど X Window システムでも有効である。X Window システムで有効な色名は '`/usr/lib/X11/rgb.txt`' に列挙されている。しかし、色名に "gray" が入っているものについては、X Window では "grey" である。たとえば "dark slate gray 4" と "dark slate grey 4" はどちらも色名として有効である。

Appendix C ページと描画領域のサイズ

Illustrator、PostScript、PCL 5、HP-GL、Fig の各 Plotter でベクトル形式の画像出力を行う場合、出力先におけるページの大きさの指定が重要である。ページの大きさとして "letter" や "a4" などが指定できる。指定可能なもののリストを後に載せる。GNU `libplot` におけるページサイズの指定は `PAGESIZE` パラメータによって行う。`libplot` を組み込んでいる `graph`、`plot`、`pic2plot`、`tek2plot`、`plotfont` の各コマンドでは、環境変数 `PAGESIZE` およびコマンドライン・オプション '`--page-size`' でページサイズが指定できる。

`libplot` による描画は通常、「描画領域」または 'viewport' に対して行われる。PNG、PNM、疑似 GIF のビットマップ形式での出力を行う場合は viewport は単に正方形あるいは長方形のビットマップである。しかし Illustrator、PostScript、PCL 5、HP-GL、Fig でベクトル形式の出力を行う場合は、出力先の用紙上の正方形あるいは長方形の領域である (SVG および WebCGM の場合の viewport については後述)。HP-GL の場合を除いて、viewport は用紙の中央に置かれるのがデフォルトである。たとえば用紙サイズが "letter" の場合、viewport は一辺が 8 in の正方形で、8.5 in x 11.0 in の用紙の中央に置かれる。viewport の外側に描いた図形がクリッピングされることはないので、原理的には用紙全体に描画できることになる。

viewport の縦の長さと横の長さはそれぞれ別に指定できる。たとえば"letter,xsize=4in" や "a4,xsize=10cm,ysize=15cm" と言った方法で指定すれば、デフォルト値の代わりにその指定が使われる。viewport の縦と横の長さは負の値でもよい (負の値を指定すると鏡像になる)。長さの単位にはインチ、センチメートル、ミリメートルが使える。

ベクトル形式の出力では、viewport の位置を用紙の左下隅に対する相対座標値で指定できる。たとえば、"letter" や "a4" などの用紙サイズだけでなく、"letter,xorigin=2in,yorigin=3in" や "a4,xorigin=0.5cm,yorigin=0.5cm" のように指定できる ('xorigin' と 'yorigin' は 'xsize' や 'ysize' と同時に指定できる)。または、'xorigin' と 'yorigin' の代わりに、のデフォルトの位置に対する相対的な viewport の位置をベクトルを使って指定することもできる。たとえば用紙サイズの指定で"letter,yoffset=1.2in" や "a4,xoffset=−5mm,yoffset=2.0cm" とすることができる。また、`ROTATION` パラメータやその環境変数、または (`libplot` を組み込んでいるプログラムの場合は) '`--rotation`' オプションを使うと viewport の回転角を指定できる。しかし viewport は回転させられても、その四隅の用紙上の位置は変化しない。代わりにそこに描画される図形が、回転して描かれることになる。もし viewport が正方形でなく長方形だった場合は、回転の際に必要に応じて縦、横それぞれに拡大・縮小が行われる。

用紙サイズには ISO の "a0"..."a4" と ANSI の "a"..."e" が指定できる ("letter" は "a" と、"tabloid" は "b" と と同じである。デフォルト値は"letter")。また"legal"、"ledger"、JIS (Japanese Industrial Standard) の "b5" も指定できる。以下が、指定できる用紙サイズと、その用紙サイズを指定したときの正方形の viewport の大きさである。

"a" (or "letter"; 8.5 in by 11.0 in)
8.0 in

"b" (or "tabloid"; 11.0 in by 17.0 in)
10.0 in

"c" (17.0 in by 22.0 in)
16.0 in

"d" (22.0 in by 34.0 in)
20.0 in

"e" (34.0 in by 44.0 in)
32.0 in

"legal" (8.5 in by 14.0 in)
8.0 in

"ledger" (17.0 in by 11.0 in)
10.0 in

"a4" (21.0 cm by 29.7 cm)
19.81 cm

"a3" (29.7 cm by 42.0 cm)
27.18 cm

"a2" (42.0 cm by 59.4 cm)
39.62 cm

"a1" (59.4 cm by 84.1 cm)
56.90 cm

"a0" (84.1 cm by 118.9 cm)
81.79 cm

"b5" (18.2 cm by 25.7 cm)
16.94 cm

SVG と WebCGM の場合はこれとは別である。ウェブページ上で最終的に表示される画像の大きさを、これらの画像ファイル中に指定しておくことはできない。画像ファイル中で viewport の大きさを指定はできるが、多くの場合ウェブページを作るときにその大きさには別の値が指定される。SVG または WebCGM 形式の出力を行うときは、viewport の大きさは普通に `PAGESIZE` または (`libplot` を組み込んだプログラムによる場合は) '`--page-size`' で指定する。たとえば用紙サイズに "letter" を指定したときは SVG または WebCGM での viewport の大きさは一辺が 8 in の正方形である。"letter,xsize=6in,ysize=7in" と指定すると viewport は 6 in x 7 in の長方形になる。"xorigin"、"yorigin"、"xoffset"、"yoffset" は無視される。

同じ理由で HP-GL や HP-GL/2 出力のときにも "xorigin" と "yorigin" は無視される。HP-GL のデフォルトでは viewport の左下隅は出力機器によって異なる、'scaling point' P1 と呼ばれる位置である。これは実際の用紙の左下隅に近いが、そことはずれた場所である。"xoffset" や"yoffset" の指定も有効ではあるが、これは viewport の位置をデフォルトの位置からずらすために使う。

Appendix D Metafile 形式

GNU graphics metafile 形式の画像は、GNU `libplot` でサポートしている metafile の Plotter を使うソフトウェアで生成できる。`graph`、`plot`、`pic2plot`、`tek2plot`、`plotfont` の生データ出力がそれにあたる。Metafile 形式のファイルは、一種の「監査証跡 (audit trail)」とも言える。つまり、プロット命令とプロットされるデータが連続したものである。各プロット命令は 1 文字の ASCII 文字である 'op code' であり、Plotter に対する操作を表す。その操作に必要なデータがあれば、プロット命令の引数として指定される。

Metafile にはバイナリ形式 (デフォルト) とテキスト形式 (人が読める形式) の二種類がある。バイナリ形式では画像データは `"#PLOT 1\n"` で始まり、テキスト形式では `"#PLOT 2\n"` で始まる。Metafile 形式のデータを、生成した計算機とは違う環境に移動する場合などは、バイナリ形式よりもテキスト形式のほうがいいだろう。テキスト形式のデータは GNU `graph` や他の plotting utilities のプログラムで '`-O`' を指定するか、Metafile の Plotter のパラメータ `META_PORTABLE` に `"yes"` を指定すればよい。`plot` を使えば、バイナリ形式でもテキスト形式でも、metafile 形式のデータを他の形式に変換できる。Chapter 3 [plot], page 27 参照。

テキスト形式では、各操作の引数 (整数、浮動小数点実数、文字列) は、それぞれ空白で区切られ、改行が引数列の終端を示す。バイナリ形式では、引数は整数、単精度浮動小数点実数、改行で終端が示される文字列である。`libplot` で実装している描画操作では、文字列引数は多くても 1 つで、その中には改行文字は含められないので、改行文字が引数列の終端として扱われる。したがって、文字列は引数列中の最後の引数でなければならない。

全部で 97 種の Plotter に対する操作が実装されている。もっとも重要なのは `openpl` および `closepl` 操作である。これらによって Plotter がオープン、クローズされ、それによって画像のページが開始、終了する。`openpl` および `closepl` 操作を表す op code はそれぞれ '`o`' と '`x`' である。`erase` 操作は、もし行われれば、ページ中のフレームの区切りになる。リアルタイム描画を行うデバイスでは、それによりスクリーン上に描画された画像が消える。この操作の op code は '`e`' である。

その他の 94 種の Plotter 操作には、12 の例外を除いて op code が割り当てられている。その 12 の例外は、(1) `flushpl` による準備操作、(2) `havecap`、`labelwidth`、`flabelwidth` の情報を返すだけの操作、(3) `color`、`colorname`、`pencolorname`、`fillcolorname`、`bgcolorname` の、内部で `pencolor`、`fillcolor`、`bgcolor` に割り当てられている操作、(4) `frotate`、`fscale`、`ftranslate` の内部で `fconcat` に割り当てられている操作、(5) `ffontname` (metafile 中では `fontname` となにも違うところのない操作) である。したがって '`o`' および '`x`' の他に 83 種の op code が利用でき、つまり合計で op code は 85 種類である。以下に '`o`' and '`x`' 以外の 10 個の op code を挙げる。各 op code には `libplot` での操作の名称を付けてある。

Op Code	Operation
'`a`'	`arc`
'`c`'	`circle`
'`e`'	`erase`
'`f`'	`linemod`
'`l`'	`line`
'`m`'	`move`

‘n’ cont

‘p’ point

‘s’ space

‘t’ label

85 種類のすべての op code のリストは libplot のヘッダファイル ‘plot.h’ または libplotter のヘッダファイル ‘plotter.h’ にある。このファイルは多くの場合、‘/usr/include’ または ‘/usr/local/include’ にインストールされている。

上に挙げた 10 種類の op code は GNU 版より前の graph と libplot で使われていた ‘plot(5)’ 形式のものと同じで、その使い方も ‘plot(5)’ と同じである。plot(5) 形式のファイル中には op code の ‘o’ および ‘x’ や冒頭に GNU metafile 特有の文字列 (magic string) がないので、GNU 版と簡単に区別することができる。GNU plot を使えば、plot(5) 形式の metafile を GNU metafile 形式に変換できる。Chapter 3 [plot], page 27 参照。

Appendix E 関連するソフトウェア

E.1 idraw の入手先

idraw についてはこの文書内でも何回か触れているが、これはフリーの X Window アプリケーションで、画像を編集するソフトウェアである。これを使って libplot を組み込んだソフトウェア (graph -T ps、plot -T ps、pic2plot -T ps、tek2plot -T ps、plotfont -T ps) の PostScript の Plotter で生成された画像を編集することができる。

idraw は現在 Vectaport 社がメンテナンスしており、Vectaport 社のウェブサイト (http://www.vectaport.com/) からダウンロードできる。idraw はドローソフトを作るためのフレームワーク ivtools の一部である。idraw は元々、スタンフォード大学とシリコン・グラフィクス社で開発されていた InterViews パッケージの一部であった。その開発は現在すでに行われていないが、いまでもウェブ (ftp://interviews.stanford.edu/) で公開されている。ダウンロードするなら InterViews よりも ivtools にしたほうがいいだろう。

Vectaport 社のウェブページ (http://www.vectaport.com) には、idraw の高機能版である drawtool もある。idraw が X11 のビットマップ形式画像を取り込めるのと同様に、drawtool は TIFF および PBM/PGM/PPM 形式のビットマップ画像を取り込むことができる。

E.2 xfig の入手先

xfig についてもこの文書内でも何回か触れているが、これはフリーの X Window アプリケーションで、画像を編集するソフトウェアである。これを使って libplot を組み込んだソフトウェア (graph -T fig、plot -T fig、pic2plot -T fig、tek2plot -T fig、plotfont -T fig) の Fig の Plotter で生成された画像を編集することができる。

最新版は ftp://ftp.x.org/contrib/applications/drawing_tools/ からダウンロードできる。GIF、X11 ビットマップ、PostScript の各形式の画像を取り込むことができる。transfig というパッケージが付属していて、これを使うと xfig の画像を他の様々な形式の画像に変換することができる。現在 GIF、X11 ビットマップ、LaTeX、PostScript の各形式がサポートされている。

Fig 形式についてのウェブページ (http://duke.usask.ca/~macphed/soft/fig) に xfig を使うときにあると便利なアプリケーションなどが紹介されている。

Appendix F 開発の経過と関わった人たち

GNU plotting utilities の一部は Unix plotting utilities を元にしている。`graph` コマンドを含むさまざまなプロット・フィルタは、Unix が最初にベル研からリリースされたときにすでに含まれていて、少なくとも Version 4 (1973 年) まではあった。最初は Tektronix 611 storage scope が出力端末であった。様々なプロット・フィルタをまとめて、`libplot` の機種依存性をなくしたのは Lorinda Cherry (`llc@research.att.com`) である。Unix の Version 7 (1979 年) とそれに続くバークレー版から、`graph`、`plot`、`spline` をまとめたパッケージと機種依存性の残っている `libplot` が標準で搭載されるようになった。1980 年代前半では、各種の Tektronix storage scope、初期のグラフィック端末、Versatec と Varian の 200 dpi 静電プリンタ／プロッタ、ヒューレット・パッカードのペン・プロッタなどがサポートされていた。

最初の GNU 版の `graph`、`plot`、`spline` とそれらの付属文書は Rich Murphey (`rich@freebsd.org`) が 1989 年に書いた。Richard Stallman は開発続行を指示し、付属文書の編集をサポートした。スタンフォード大学で InterViews の開発チームの John Interrante (`interran@uluru.stanford.edu`) は `idraw` で使っていた PostScript prologue ファイルを惜しげもなく提供し、様々な助言も行ってくれた。それは最新の `libplot` でも使われている。1991 年には、パッケージは 'GNU graphics' という名前で公開されていた。

1995 年には Robert S. Maier (`rsm@math.arizona.edu`) が開発を引き受け、現在の機種依存性がほとんどなくなったスタンドアロンの `libplot` の設計、実装を行った。彼は `graph` をスクラッチから書き直し、新しくなった `libplot` を使ってリアルタイム・フィルタとして使えるようにした。彼は `spline` も書き直して、テンション、周期スプライン、三次のベッセル関数による補間の各機能を使えるようにした。

`libplot` には現在、多角形の塗りつぶしや多角形や円弧を描く X Window のコードが取り入れられている。ビットマップ画像 (PNG、PNM と疑似 GIF) を生成するときにこのコードが使われている。このコードは Brian Kelleher、Joel McCormack、Todd Newman、Keith Packard、Robert Scheifler、Ken Whaley (以上 Digital Equipment Corp.) と MIT、X Consortium によるものである (© 1985–89 by the X Consortium)。文字列のアフィン変換は Alan Richardson による `xvertext` パッケージで行われているのと同様の方法を使っていて、これにより文字列の回転を行っている。

現在の `libplot` の 疑似 GIF 機能は、der Maus (`mouse@rodents.montreal.qc.ca`) と ivo によるランレングス圧縮を採用した 'miGIF' を使っている (© 1998 by Hutchison Avenue Software Corporation)。miGIF の著作権や利用許諾条件は GNU plotting utilities パッケージに同梱されている。

`ode` はほとんど Nick Tufillaro (`nbt@reed.edu`) が 1978–1994 年にかけて、PDP-11 上の Unix Version 4 用に作ったものである。Nick の 1994 年版を元に Robert Maier が 1997 年に GNU 用にコーディングとコマンドライン・オプションを修正し、また `gnuplot` の特殊関数をすべてサポートし、例外処理を改良した。

Plotting utilities の開発に貢献した人は他にもたくさんいる。現在 `libplot` に組み込まれている Hershey ベクトル・フォントはもちろん、主に 1960 年代後半に Allen V. Hershey が電子化したものが元になっている。彼には感謝を捧げる。他にも Robert C. Beach が 1970 年代中頃に開発した SLAC Unified Graphics System や Thomas Wolff (`wolff@inf.fu-berlin.de`) が Ghostscript 用に作ったものから文字や記号が採用されている。`spline` コマンドの補間アルゴリズムは Alan K. Cline (`cline@cs.utexas.edu`) のものに基づいており、それは Apr. 1974 issue of *Communications of the ACM* に採録されている彼の論文によるものである。`tek2plot` の表指向の構文解析ルーチンは 1980 年中頃に Edward Moy (`moy@parc.xerox.com`) がバー

クレーで書いたものである。`libplot` で円弧や楕円弧を描くときに使われる 'sagitta' アルゴリズムは、URW の Peter Karnow と Apple の Ken Turkowski (`turk@apple.com`) によるものである。Raymond Toy (`toy@rtp.ericsson.se`) は `graph` でも目盛りの配置を行うコードに貢献し、また GNU `getopt` を最初に導入した。以前 Cornell 大学 LASSP にいた Arthur Smith は、彼が書いた `xplot` プログラムのコードを寄贈してくれた。Nelson Beebe (`beebe@math.utah.edu`) はパッケージのインストール法を徹底的にテストしてくれた。

Robert Maier がこの文書を書いた。この文書には現在 Nick Tufillaro の `ode` のマニュアルも含まれている。Julie Sussmann が文書全体の校正を行った。

以上のすべての部分の英語から日本語への翻訳は、(独) 産業技術総合研究所 (生命情報工学研究センター) のとみながだいすけ (`tominaga@cbrc.jp`) (Daisuke TOMINAGA) が 2009 年 8 月に行った。

Appendix G バグ・レポート

GNU plotting utilities に関するバグは、`libplot` ライブラリのものも含めてすべて、`bug-plotutils@gnu.org` に送ってほしい。その際、GNU plotting utilities のどのバージョンを使ったときのバグかを明記するようにお願いする。GNU plotting utilities に含まれるコマンドライン・プログラムのバージョンは、‘`--version`’ オプションを付けてコマンドを実行すると表示される。

ソースファイルからコンパイルしてインストールしている場合は、そのときに使ったコンパイラの種類 (とバージョン) も教えてほしい。もし生じた問題が利用環境に特有なものである場合は、それに関する情報もすべて教えてほしい。

Appendix H GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

H.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Table of Contents