gnuplot 6.0

An Interactive Plotting Program

Thomas Williams & Colin Kelley

Version 6.0 organized by: Ethan A Merritt and many others

Major contributors (alphabetic order):
Christoph Bersch, Hans-Bernhard Bröker,
John Campbell, Robert Cunningham,
David Denholm, Gershon Elber,
Roger Fearick, Carsten Grammes,
Lucas Hart, Lars Hecking, Péter Juhász,
Thomas Koenig, David Kotz, Ed Kubaitis,
Russell Lang, Timothée Lecomte,
Alexander Lehmann, Jérôme Lodewyck,
Alexander Mai, Bastian Märkisch, Tatsuro Matsuoka,
Ethan A Merritt, Petr Mikulík, Hiroki Motoyoshi,
Daniel Sebald, Carsten Steger, Shigeharu Takeno,
Tom Tkacik, Jos Van der Woude,
James R. Van Zandt, Alex Woo, Johannes Zellner

Mailing list for comments: `gnuplot-info@lists.sourceforge.net`
Web site and issue trackers: `http://sourceforge.net/projects/gnuplot`

This manual was originally prepared by Dick Crawford
Japanese translation supervised by Shigeharu Takeno (竹野 茂治)

Version 6.0.5 (June 2026))

目 次

第 I 部

Gnuplot

著作権 (Copyright)


```
AUTHORS
        Original Software:
           Thomas Williams,  Colin Kelley.
        Gnuplot 2.0 additions:
           Russell Lang, Dave Kotz, John Campbell.
        Gnuplot 3.0 additions:
           Gershon Elber and many others.
        Gnuplot 4.0 and subsequent releases:
           See list of contributors at head of this document.
```

(以下おおまかな訳; 訳は正しくないかも知れませんので詳しくは上記の原文を当たってください。訳者は責任を持ちません。)

このソフトウェアの修正も認められています。しかし、修正を含む全ソースコードの配布の権利は認められません。修正はリリース版に対するパッチの形で配布しなければなりません。修正されたソースをコンパイルして作られたバイナリの配布は、以下の条件の元で認められます:

1. リリース版からのソースの修正部分を、パッチの形でバイナリと共に配布すること
2. ベースとなるリリース版と区別するために、そのバージョン番号に特別なバージョン指定子を付加すること
3. その修正版のサポート用に、あなたの名前とアクセス可能なアドレスとを提供すること
4. ベースとなるソフトウェアの使用に関しては、我々の連絡情報を保持し続けること

リリース版のソースコードを、パッチの形でのソースの修正と一緒に配布することは、バイナリ配布に関する条項 2 から 4 までの条件の元で許されます。

このソフトウェアは "あるがまま" 提供され、適用可能な法律で許められる範囲の保証を表明あるいは暗示していはいません。

著者

オリジナルソフトウェア:
Thomas Williams, Colin Kelley.
Gnuplot 2.0 追加:
Russell Lang, Dave Kotz, John Campbell.
Gnuplot 3.0 追加:
Gershon Elber とその他の人々。
Gnuplot 4.0 およびそれ以降のリリース:
この文書の先頭の寄与者 (contributors) の一覧参照。

はじめに (Introduction)

gnuplot は、ポータブルなコマンド入力方式のグラフユーティリティで、Linux, OS/2, MS Windows, macOS, VMS, その他多くのプラットホーム上で動作します。ソースコードには著作権がありますが、無料で配布されています (すなわち、それに対価を支払う必要はありません)。元は、科学者や学生が数学関数やデータなどを対話的に表示できるよう作られたのですが、現在までに例えば Web スクリプトなど、多くの非対話型の利用もサポートするように成長しています。これは、例えば Octave のようにサードパーティのアプリケーションの描画エンジンとしても使われています。gnuplot は、1986 よりサポートと活発な開発が行われています。

gnuplot は、2 次元、または 3 次元の多くの種類のグラフを生成できます。います: 折線グラフ、点グラフ、棒グラフ、等高線、ベクトル場描画、画像の取り込み、曲面、そしてそれらに関連するさまざまな文字列等。そしてさらにいくつかの特別なグラフ、例えば温度分布図 (heatmap)、蜘蛛の巣グラフ (レーダーチャート, spiderplot)、極射影 (polar projection)、ヒストグラム (histograms)、箱ひげ図 (boxplot)、ビースウォームグラフ (bee swarm)、非線形軸などもサポートしています。

gnuplot は多くの異なる出力をサポートしています: 対話型スクリーン出力形式 (マウスやホットキー入力も可能)、ペンプロッタや現在のプリンタへの直接出力、または多くのファイル形式への出力 (eps, emf, fig, jpeg, LaTeX, pdf, png, postscript, ...)。gnuplot は、容易に新しい出力形式を追加するよう拡張することができます。最近では例えば webp アニメーションがサポートされました。svg や HTML5 canvas 出力形式を利用すれば、グラフを Web ページ内にマウス利用可能な形で埋め込んだ出力を生成することもできます。

gnuplot のコマンド言語は大文字小文字を区別します。すなわち、小文字で書かれたコマンドや関数名は、それらを大文字で書いたものとは同じではありません。いずれのコマンドも、あいまいさの無い限りにおいて省略することができます。1 行中にはセミコロン (;) で区切って複数のコマンドを書くことができます。文字列は単一引用符、あるいは二重引用符のどちらかで書き始めますが、両者には微妙な違いがあります (詳細は、以下参照: **syntax (p. 84)**)。例:

```
set title "My First Plot";  plot 'data';  print "all done!"
```

コマンドは、複数行にまたがることができます。その場合は、最終行以外の全ての行の行末にバックスラッシュ (\) を書く必要があります。バックスラッシュは必ず各行の最後の空白以外の文字でなくてはなりません。その結果としてバックスラッシュと、それに続く改行文字を存在しなかったかのように扱います。バックスラッシュはコメントを終了しませんので、複数行にまたがる行の先頭をコメントアウトすると、そのコマンド全体をコメントアウトすることになります (以下参照: **comments (p. 41)**)。複数行のコマンドでエラーが起きたときは、エラーメッセージに含まれる行番号は、最初の行に対応することに注意してください。

このドキュメントにおいて、中括弧 ({}) は省略可能な引数を表すものとし、縦棒 (|) は、互いに排他的な引数を区切るものとします。**gnuplot** のキーワードや **help** における項目名は、逆引用符 (`)、または可能な場合には **boldface** (太字) で表します。角括弧 (<>) は、それに対応するものに置き換えられるべきものを表します。多くの場合、オプションの引数には、それが省略されるとデフォルトの値が使用されます。しかし、これらの場合、必ずしも角括弧が中括弧で囲まれて書かれているわけではありません。

大量のグラフサンプルが、以下の Web ページにあります。http://www.gnuplot.info/demo/

コマンドラインから起動するときは、以下の書式が使えます。

```
gnuplot {OPTIONS} file1 file2 ...
```

ここで file1, file2 等は、**local** コマンドで取り込むのと同等の入力ファイル (スクリプトファイル) です。gnuplot に与えるオプションは、コマンド行のどこに置いても構いません。ファイルは指定した順に実行され、同様に -e オプションで任意のコマンドを与えることもできます。例:

```
gnuplot   file1.in   -e "reset"   file2.in
```

特別なファイル名 "-" は、標準入力から読ませるのに使います。**gnuplot** は最後のファイルを処理し終わると終了します。読み込ませるファイルを一つも指定しない場合は、**gnuplot** は標準入力からの対話入力を取ります。詳細は、以下参照: **batch/interactive (p. 38)**。コマンドラインオプションの詳細は以下参照: **command-line-options (p. 38)**。または以下を実行してください。

```
gnuplot --help
```

対話型描画ウィンドウでの作業中は、'h' を打つとホットキー (**hotkeys**) とマウス機能 (**mousing**) に関するヘルプを見ることができます。

ある項目についてのヘルプが必要なときには、**help** に続けてその項目名を入力して下さい。または単に **help ?** でもヘルプの項目のメニューが現われます。

探し出す手助け/バグ (Seeking-assistance / Bugs)

公式の gnuplot ホームページは以下にあります。http://www.gnuplot.info

助けを求める前に、ファイル FAQ.pdf か、または上の Web サイトのFAQ（度々聞かれる質問; Frequently Asked Questions）の一覧

をチェックしてください。

他に、(バグ以外の) 特定のグラフ描画の問題に関する助言は以下でも得られます。

https://stackoverflow.com/questions/tagged/gnuplot

バグの報告と、機能のリクエストは、以下のトラッキングシステムにあげてください。

https://sourceforge.net/p/gnuplot/_list/tickets

ただし、あなたが報告しようとしているバグが、より新しい版で既に修正されていないか、以前の報告をチェックしてください。

バグの報告や質問を投稿するときは、あなたが使用している gnuplot のバージョン、出力形式 (terminal)、オペレーティングシステム、といった情報をすべて入れてください。問題を再現する自己完結型の短いスクリプトを提示するとなお良いでしょう。

gnuplot メーリングリストへの投稿の方法に関しては、gnuplot の開発 Web サイトhttp://sourceforge.net/projects/gnuplot

を参照してください。

gnuplot メーリングリストにメールを書く前に、最初にそのメーリングリストに参加する必要があることに注意してください。これは、スパムの量を減らすために必要です。

メーリングリストメンバーへのメールアドレス:

gnuplot-info@lists.sourceforge.net

開発版に関するメーリングリスト:

gnuplot-beta@lists.sourceforge.net

バージョン 6 での新しい機能 (New features in version 6)

バージョン 6 は、遡ること 1986 年からの gnuplot 開発での最新メジャーリリースです。それは、メジャーバージョン 5 (2015)、そしてその後のマイナーリリース 5.2 (2017), 5.4 (2020) に続くものです。開発は、SouceForge 上の git リポジトリプロジェクト内の、正式版とは別なブランチで続けられています。

このドキュメントに記述されている機能の中には、gnuplot をソースからコンパイルする際にそれを選択して設定している場合だけ使えるものがあります。あなたが実行している特定の gnuplot が、コンパイル時にどのようなオプションを設定して作られたかを知るには、**show version long** とタイプしてください。

関数ブロックと局所変数 (Function blocks and scoped variables)

この版の gnuplot では、標準的な gnuplot コマンドのブロックを呼び出して関数として使える機能を導入しています。関数ブロックは、0 から 9 個の引数を使えて、一つの値を返します。関数ブロックは、計算した新しい値を変数に割り当てたり、異なる関数や演算子を結合したり、与えられたデータに対する繰り返しの作業を行ったりするのに使うことができます。この仕組みには、3 つの要素があります。以下参照: **local (p. 153)**, **scope (p. 80)**, **function blocks (p. 148)**, **return (p. 192)**。

- 修飾子 **local** は、変数や配列のオプション宣言で、そのスコープは、それが書かれているプログラム単位の実行中のみに制限されます。単位とは、現在は、**load** や **call** 文の実行単位、関数ブロック評価、そして **if**, **else**, **do for**, **while** に続く中かっこで囲まれたコードブロック単位です。局所 (local) 変数の名前が大域 (global) 変数の名前とぶつかる場合は、その局所スコープから抜けるまでは大域変数は隠されます。
- コマンド **function** は、gnuplot コマンドからなる名前付き関数ブロック (実際はある文字列の配列) を宣言します。関数ブロックを呼び出すと、そのコマンドが、ブロックの最後になるか、または **return** コマンドに当たるまで、順次実行されます。
- コマンド **return <expression>** は、関数ブロックの実行を終了します。式 <expression> の評価の結果が、その関数の値として返されます。関数ブロックの外では、どこでも **return** は **exit** と同様に動作します。

この仕組みを使って、単純な一行の関数定義 **f(x) = ...** よりもっと複雑で自明ではない関数を定義し、描画する例については、**function_block.dem**

を参照してください。

特殊関数と複素数値関数 (Special and complex-valued functions)

gnuplot バージョン 6 は、膨大な複素数値関数群と、以前のバージョンにあった関数の改良版をいくつか提供します。

- 新規: 複素変数、複素数値のリーマンゼータ (ζ) 関数。以下参照: **zeta (p. 60)**。
- (正規化) 下方不完全ガンマ関数の定義域と精度の改良。複素引数が可能に。以下参照: **igamma (p. 53)**。
- 上方不完全ガンマ関数を新規追加 (実引数のみ)。以下参照: **uigamma (p. 57)**。
- (正規化) 不完全ベータ関数の定義域と精度の改良。以下参照: **ibeta (p. 54)**。
- 逆 (正規化) 不完全ガンマ関数を新しく追加。以下参照: **invigamma (p. 54)**。
- 逆 (正規化) 不完全ベータ関数を新しく追加。以下参照: **invibeta (p. 54)**。
- 多価関数 W_k(z) の第 k 分岐を返す複素関数 LambertW(z,k) を新しく追加。古い lambertw(x) は real(LambertW(real(z), 0)) であることに注意。以下参照: **LambertW (p. 54)**。
- 複素関数 lnGamma(z) を新しく追加。既にある lgamma(x) は real(lnGamma(real(z)) であることに注意。以下参照: **lnGamma (p. 55)**。
- z の複素共役を返す複素関数 conj(z)
- (第 1) シンクロトロン関数 F(x)。以下参照: **SynchrotronF (p. 55)**。
- acosh(z) の定義域を負の実数軸を覆うように拡張。
- asin(z) asinh(z) の複素数引数に対する精度の改良。
- 便利なように I = sqrt(-1) = {0,1} を定義済み変数に。gnuplot は {a,b} を正しい複素数定数と見なしてくれませんが、しかし (a + b*I) なら正しい複素数数式として受けつけてくれるので有用です。

ビルド時に適切な外部ライブラリがあれば、さらにいくつかの特殊関数をサポートします。以下参照: **special_functions (p. 55)**。

- ν 次 (実数) の、引数 z に対する複素ベッセル関数 Iν (z), Jν (z), Kν (z), Yν (z)。以下参照: **BesselK** (p. 53)。
- ν 次 (実数) の、引数 z に対する複素ハンケル関数 H1ν (z), H2ν (z)。以下参照: **BesselH1** (p. 53)。
- 複素エアリー関数 Ai(z), Bi(z)。
- n 次の複素指数積分。以下参照: **expint** (p. 53)。
- フレネル積分 C(x), S(x)。以下参照: **FresnelC** (p. 53)。
- Voigt プロファイルの半値全幅を返す関数 **VP_fwhm(sigma,gamma)**。以下参照: **VP** (p. 50), **VP_fwhm** (p. 50)。

新しい描画スタイル (New plot styles)

- 描画スタイル **with surface** は、2 次元極座標で動作し、平面を色塗りした格子表現のグラフを生成します 色は入力点の任意の集合のからの寄与による重み付きよって色付けされます。これは、3 次元格子曲面を作る **dgrid3d** とスタイル **with pm3d** の類似品です。以下参照: **set polar grid** (p. 275), **polar heatmap** (p. 118)。
- 新しい 2 次元描画スタイル **with sectors** は、完全な極座標格子面を生成する別の仕組みです。これは概念上の極座標格子内に、各入力データ点毎に丸い扇片をひとつ生成します。極座標モードでの **with surface** とは違い、これは極座標グラフでも直交座標グラフでも使用できます。
- 新しい 2 次元描画スタイル **with hsteps** は、存在するスタイル **steps**, **histeps**, **fsteps**, **fillsteps** が提供するもの、およびさらなる表現の多様性を持つ step 系のグラフを生成できます。
- 描画スタイル **with lines** には、現在フィルタオプション **sharpen** があります。このフィルタは関数グラフでスパイクを検出しますが、そのピークは関数から標本として取った 2 つの x 座標の間にあるため、出力では不完全に切り取られてしまいます。このフィルタは、そのようなピーク点の場所に新しい標本点を追加します。以下参照: **filters** (p. 166)。
- これは厳密には新しい描画スタイルではありませんが、フィルタ concave hull と領域塗り潰しのパスに沿った平滑化を組み合わせることで、「膨らんだ領域」グラフの作成が可能になります。これは例えば複数のデータのかたまりが重なっているものの広さを示します。以下参照: **concavehull** (p. 167)。
- 3 次元描画スタイル **with pm3d** でオプション修飾子 **zclip [zmin:zmax]** が使えて、これは曲面全体のひとつの断面のみを選択します。クリッピングの境界値を増加させて連続描画を行うことで、3 次元では断面切り出しアニメーションを見るのに使えますし、間を塗り潰した等高線図を作成するのに使えます。これは、新しい描画スタイル **with contourfill** で自動化でき、それは 2 次元射影で特に有用です。以下参照: **contourfill** (p. 98)。

凸包とマスクと平滑化 (Hulls, masks, and smoothing)

- 2 次元の点の集合を、その境界の多角形で置き換える新しいフィルタ **convexhull**。その境界の曲線を滑らかにするには、"convexhull smooth path with filledcurves" を使って塗り潰し領域として描画することでできます。以下参照: **convexhull** (p. 167)。

- フィルタ **concavehull** は、凸とは限らない、凹の度合を制御する特性長パラメータで決定されるχ -形状の、多角形のデータ境界を生成します。これはデータ点の回りの本質的なしみを描画します。以下参照: **concavehull (p. 167)**。
- pm3d 曲面や image 描画の選択した一部分のみを表示するようマスクするのに、凸包 (convex hull) や他の多角形 (polygon) を使用できます。新しい描画スタイル **with mask** (マスクを定義) や、キーワード **mask** (その後の描画要素にマスクを適用する) を参照してください。
- 閉曲線や x に関して単調ではない 2 次元曲線向きの、パスに沿った 3 次元スプライン平滑化。以下参照: **smooth path (p. 172)**。これは、凸包やマスクの平滑化も可能にします。
- 3 次元曲線の 3 次スプライン平滑化。以下参照: **splot smooth csplines (p. 172)**。
- 平滑化オプションは **with filledcurves {above|below|between}** での描画にも適用します。
- 周期的データの平滑化用の新しいキーワード **period**。以下参照: **smooth kdensity (p. 174)**。

名前付きパレット (Named palettes)

- 現在のパレットをその後の利用のため名前付きカラーマップに保存できます。以下参照: **set colormap (p. 201)**。
- pm3d と image plot で以前保存したパレットを名前で指定できます。これにより、一つの plot コマンドで複数のパレットを使用できます。以下参照: **colorspec palette (p. 73)**。
- 名前付きパレットカラーマップは、32-bit ARGB 色値の配列として操作できます。これにより、アルファチャネル値を追加したり、コマンド **set palette** では簡単に指定できない他の修正を可能にします。
- 定義済みの新しい色機構 **set palette viridis**。
- ファイルやデータブロックから読み込んだパレット (**set palette file**) は、実数の色成分か、24bit 形式の RGB 値のいずれかで指定できます。

新しいデータ形式 (New data formats)

- オプション **sparse matrix=(cols,rows)** は、**plot** と **splot** に、個々のピクセル値を任意の順番で読みだすことができるような一様なピクセル格子を生成させます。これは、不完全なデータから温度分布図 (heatmap) を描画するような場合に有用です。以下参照: **sparse (p. 323)**。
- 非一様な matrix データの入力中、現在は column(0) はその matrix 要素の線形順序を返します。すなわち、MxN matrix の A の要素 A[i,j] に対しては、column(0)/M が行番号 i に、column(0)%M が列番号 j になります。

新しい組み込み関数と配列操作 (New built-in functions and array operations)

- cbrange 内の z に割当てる現在の RGB パレット色を返す関数 **palette(z)**
- 色名からその色の 32bit ARGB 値を返す関数 **rgbcolor("name")**
- 要素 Array[i] が element に等しい最初の添字 **i** を返す関数 **index(Array, element)**。以下参照: **arrays (p. 64)**。

- 配列を引数とするユーザ定義関数の許可。例: dot(A,B) = sum [i=1:|A|] A[i]*B[i]
- 配列名に範囲を指定することで部分配列を実現。Array[n] は単一の要素、Array[n:n+5] は元の配列の 6 要素を持つ部分配列。以下参照: **arrays** (p. **64**), **slice** (p. **65**)。
- **split("string", "separator")** は、文字列 string に含まれるフィールド要素を、文字列の配列に詰めて返します。以下参照: **split** (p. **59**)。
- **join(array, "separator")** は、**split** の逆で、文字列配列の要素を、フィールド区切り文字 separator を間に挟んで結合して一つの文字列にしたものを返します。以下参照: **join** (p. **59**)。
- **stats <non-existent file>** はテスト可能な値を出力。以下参照: **stats test** (p. **329**)。
- **stats $vgrid** で格子内のボクセルの最小/最大/平均/標準偏差を調査
- 新しい数値演算 "prod" (累積的な積)。以下参照: **product** (p. **62**)。例: N! は prod [i=1:N] i で表現できます。

プログラムの流れの制御 (Program control flow)

- 新しい構文 **if ... else if ... else ...**
- gnuplot の設定選択で、XDG ベースのディレクトリ配置をサポートしています。gnuplot は、$XDG_CONFIG_HOME/gnuplot/gnuplotrc から初期コマンドを読み込みます。対話コマンド履歴は、$XDG_STATE_HOME/gnuplot_history に保存します。これらのファイルがない場合は、gnuplot の以前のバージョンと同様、$HOME/.gnuplot と $HOME/.gnuplot_history をそれぞれ使用します。
- **unset warnings** は、stderr への警告メッセージ出力を抑制します。
- **warn "message"** は、ファイル名、行番号と "message" を stderr に出力します。
- コマンド "fit" に対する例外処理。fit エラーが起きた場合でも、制御を常に入力の次の行に返します。エラーが起きた場合は、返ったときに FIT_ERROR がゼロでない値になります。これは、良くないフィッティングから復帰するスクリプトを可能にします。以下参照: **fit error_recovery** (p. **144**)。

多重描画モード (Multiplots)

多重描画グラフ (multiplot) の仕組みは、複数の plot, splot コマンドの出力を一つのページ、あるいは一つの画面に結合するのに使います。この仕組みは、gnuplot 6 で大幅に改修されています。バージョン 6.0.5 からは結合された画面のパネル毎に対話型マウスを使うことが可能になり、また既に存在するコマンド **replot** か新しいコマンド **remultiplot** による multiplot 全体の再描画も可能になっています。多重描画グラフモードは、コマンド **set multiplot** で開始し、**unset multiplot** でそれを閉じるまで続きます。

- コマンド **set multiplot** は現在の描画設定の状態を新しいデータブロック $GPVAL_PRE_MULTIPLOT に保存し、引き続くコマンドの新しいデータブロック $GPVAL_LAST_MULTIPLOT への記録を開始します。
- **plot**, **splot** コマンドそれぞれは、多重描画グラフ内に新しいパネルを作ります。各パネルの位置、サイズ、軸の範囲、その他の属性を、後での利用のために保存します。

- **unset multiplot** 後に、上記の 2 つのコマンドデータブロックとパネル毎のデータが完結します。それらの内容は、その後の gnuplot のコマンドでは影響を受けません。対話型セッションではこの時点で多重描画グラフ全体をスクリーンに表示します。マウス操作では、保存したデータを使い、可能ならばパネル毎に適切な座標と視角を報告します。
- コマンド **replot** は $GPVAL_LAST_MULTIPLOT のコマンドを再実行します。それは、元の描画設定の状態を再構成しようとはせず、よってその結果は元の図を完全に復元したものにはならない可能性があります。これは、単一グラフに対して **replot** を使用するのと同様で、**replot** コマンドを発行する前に、例えば塗りつぶしスタイルや軸範囲の位置などをあえて変更することで、その新しい設定が再描画される図に影響を与えるようにできます。
- コマンド **remultiplot** は違います。これは $GPVAL_PRE_MULTIPLOT に保存したコマンドを読み込むことで、最初に元の描画設定の状態を再構成し、その次に保存した multiplot コマンド列を再実行します。これにより、例えば違う出力形式への出力でも、元の図を再現できるようになります。**replot** とは違い、元の multiplot と remultiplot の間に行われた画面設定はほぼすべて失われます。
- **replot**, **remultiplot** のいずれの仕様もあなたが気にいらない場合は、以下のようなカスタマイズしたコマンド列を使用することも可能です。

```
load $GPVAL_PRE_MULTIPLOT     # 元の描画設定を復帰
set style fill transparent    # 設定を一つ変更
load #GPVAL_LAST_MULTIPLOT    # multiplot コマンド列を再実行
```

[試験段階] 詳細は変更の可能性あり。現時点でいくつかの制限あり。

- 保存コマンドのうち再実行の際に問題になりそうなある種のコマンド、例えば **reset**, **pause**, 新しいデータブロックの定義などは、再実行しません。
- パネル毎に保存するデータは、非線形軸割り当てを解釈するには不十分ですので、そのグラフは正しく表示されますが、マウス操作では座標を正しく表示できません。
- 一つの multiplot に対し最大 16 のパネルでマウス操作できます。multiplot のパネル数が 16 より多い場合、追加パネルは正しく表示されますが、マウス操作は正しく行えません。この制限は、今後改善されるでしょう。
- multiplot のグラフに関しては、コマンドラインからの描画のオン/オフのトグルスイッチは効きません。
- コマンド "set terminal" と "set output" は、multiplot モード内では受け付けません。
- ファイルを '-' として入力する plot コマンドのインラインデータは保存しないので、replot やマウス操作は失敗します。インラインデータを描画、保存するには、名前付きデータブロックとしてください。例は以下。

```
$DATA << EOD
1 2 3
4 5 6
EOD
set multiplot
  plot $DATA using 1:2 with points pt 5
  plot $DATA using 1:3 with points pt 6
unset multiplot
```

新しい出力形式とオプション (New terminals and terminal options)

- 新しい出力形式 **kittygd** と **kittycairo** は、kitty プロトコルをサポートする端末エミュレータで、端末ウィンドウ上でのグラフィックを提供します。kitty は、sixel グラフィックとは別の選択肢で、24 ビット RGB フルカラーを提供します。以下参照: **kittycairo (p. 365)**。
- 新しい出力形式 **block** は、疑似グラフィックのテキストモード用のもので、**dumb** や **caca** 出力形式に対して改良された解像度を提供するために Unicode ブロック、あるいは点字 (Braille 文字) を使用します。
- 新しい出力形式 **webp** は、webp エンコーディングを用いて、単一フレームか、アニメーション列を生成します。各フレームは pngcairo で生成し、その後 libwebp と libwebpmux による WebPAnimEncoder API を通してエンコードを行います。
- **dumb**, **sixel**, **kitty**, **block** 出力形式のように、文字列出力とグラフィック表示を同じウィンドウで行う出力形式では、現在はコマンド **pause mouse** の間はキーボード入力に反応します。その間、それらの出力形式では、マウス操作が可能な出力形式が行うのと同じ方法でキー入力を解釈します。以下参照: **pseudo-mousing (p. 155)**。例えば、左/右/上/下矢印キーは、3 次元グラフでは視方向を変え、2 次元グラフでは視点移動/拡大の増分ステップを実行します。

ウォッチポイント (Watchpoints)

ウォッチポイントは、グラフ内の個々の曲線に関連する対象値です。曲線が描画されるとき、各線分要素が、その両端点の間にウォッチポイント座標 (x, y か z) か関数 f(x,y) の対象値を含んでいるかを確認します。それが見つかった場合、そのマッチする点の座標 [x,y] を後で使用するために保存します。以下参照: **watchpoints (p. 86)**。

- 2 つの曲線の交点を見つけること
- 関数のゼロ点を見つけること
- 従属変数 (y か z) か、関数 f(x,y) が指定値と一致する場所を見つけ表示すること
- マウスを使って複数のグラフに沿った値を同時に追跡すること
- 等高線上のラベル配置の制御

週曜日のサポート (Week-date time support)

2020 年に始まった新型コロナウィルス (Covid-19) の騒動では、疫学的データのグラフ化の関心が高まりましたが、そこでは報告の慣習として、よく「週曜日」を使った表が示されていました。この慣習に対する gnuplot のサポートの不足は改善し、週曜日もサポートするように拡張してあります。

- 日時書式 %W は、ISO 8601 の週曜日規則に従うようになりました。
- 日時書式 %U は、CDC/MMWR の週曜日規則に従うようになりました。
- 新しい関数 **tm_week(time, std)** は、ISO か CDC 規則でのその年の週番号を返します。
- 新しい関数 **weekdate_iso(year, week, day)** は、ISO 規則での週曜日をカレンダー時刻に変換します。
- 新しい関数 **weekdate_cdc(year, week, day)** は、CDC 規則での週曜日をカレンダー時刻に変換します。

その他の新しい機能

- 時間軸の主目盛りと副目盛り用の時間単位時間軸の主目盛りと副目盛りは、minutes/hours/days/weeks/months/years を単位とする目盛り間隔指定を受け付けます。以下参照: **set xtics (p. 307)**, **set mxtics time (p. 250)**。
- **using** 指定内での文字列 $# は、現在の入力データ行にある全列数と評価します。例えば、**plot FOO using 0:(column($# - 1))** は、各行の最後から一つ手前の列を描画します。
- bin (箱) 用の、合計でなく平均を描画するキーワード **binvalue=avg**
- **set colorbox bottom** は、垂直なカラーボックスを右に配置する代わりに水平なカラーボックスをグラフの下に配置します。
- 交差する pm3d 曲面のレンダリングの改良 - 重なる曲面のタイルを、交差曲線に沿って 2 つの部分に分割し、一方の曲面のタイルが他方の曲面を通って不正に突き出てしまうことがないようにします。
- pm3d 光源モデルに、ユーザ制御型のスポットライトを追加。以下参照: **set pm3d spotlight (p. 270)**。
- key の全体の幅と列数を固定する新規オプション。以下参照: **key layout (p. 232)**。
- **set pm3d border retrace** は、各 pm3d 四辺形の周りに、塗り潰し領域と同じ色で境界を描きます。これは、原理的に視覚効果はありませんが、あまりよくない pdf や postscript ビューワのようなディスプレイモードで、エイリアスによる副産物が導入されないようにします。
- **set isotropic** は、2 次元グラフと 3 次元グラフの両方で、x, y, z 軸のすべてのスケールが同じになるようにスケールを合わせます。
- 変更: 文字の回転角が整数という制限はなくなりました。
- 特別な (非数値の) 線種 **lt nodraw**, **lt black**, **lt background** 以下参照: **special_linetypes (p. 74)**。
- データ駆動型の histogram グラフの色割り当て。以下参照: **histograms colors (p. 109)**。
- 凡例 (key) の箱の位置は、gnuplot が他に行う仕組みによるどんな位置決めに対しても、オフセットを与えることで手動で調整できます。以下参照: **set key offset (p. 234)**。
- 描画スタイルなしで keyentry に文字列を指定することで、凡例 (key) の中にその幅全体に渡る第 2 タイトルを生成することができます。例: **plot A, keyentry "left-justified text" left, B** これは、文字列とタイトルの両方を与えることで、2 列の文字列からなる凡例を生成するのにも使えます。以下参照: **keyentry (p. 230)**。
- plot と splot コマンドに対する新しいフィルタ **plot .. if (< 式 >)** は式にマッチする入力行のみを選択します。以下参照: **filters if (p. 169)**。

バージョン 5 で導入された機能の要約 (3 Brief summary of features introduced in version 5)

5.4 で導入された機能 (Features introduced in 5.4)

- 数式と関数は 64 ビット整数演算を使用。以下参照: **integer (p. 51)**。
- 2 次元描画スタイル **polygons**, **spiderplot**, **arrows**
- 3 次元描画スタイル **boxes**, **circles**, **polygons**, **isosurface**, そしてその他ボクセル格子データの表現

- データ前処理フィルタ **zsort**
- カスタマイズした凡例 (key) を作成する **keyentry**
- 新しい LaTeX 系出力形式 pict2e は、古い出力形式 **latex**, **emtex**, **eepic**, **tpic** の代わりです。古い出力形式はもはやデフォルトではビルドしません。
- **set pixmap** は、png/jpeg/gif 画像をピクスマップ画像として取り込み、グラフやページの任意の位置に配置しスケール変換も可能
- 拡張文字列モードで \U+xxxx (xxxx は 16 進値の 4 または 5 文字) で Unicode コードポイントが指定できるように。それは出力時に対応する UTF-8 バイト文字列に変換します。
- **with parallelaxes** の書式の改変により、描画スタイル **histogram** や **spiderplot** と同様な plot コマンド内部での便利な繰り返しが可能に

5.2 で導入された機能 (Features introduced in 5.2)

- 非線形座標系 (以下参照: **set nonlinear (p. 251)**)
- データの階級幅割り当ての自動化 (以下参照: **bins (p. 166)**)
- 2 次元ビースウォームグラフ。以下参照: **set jitter (p. 228)**。
- 3 次元描画スタイル **zerrorfill**
- 3 次元光源モデルで陰影と反射光ハイライトを提供 (以下参照: **lighting (p. 270)**)
- 配列データ型と関連するコマンドや演算子。以下参照: **arrays (p. 64)**。
- 新しい出力形式 **sixelgd**, **domterm**
- 相対時間 (間隔長) を処理する新しい書式指定子 tH tM tS。以下参照: **time_specifiers (p. 221)**。

5.0 で導入された機能 (Features introduced in 5.0)

- 出力形式に依存しない点線/破線型。
- ひとつの plot での引き続く描画要素に使用するデフォルトの色の列は、色弱者により容易に区別できるものに。
- 新しい描画スタイル **with parallelaxes**, **with table**。
- マウスがその上にあるときに有効になるハイパーテキストラベル。
- 2 次元、3 次元関数描画や疑似ファイル '+', '++' での描画における明示的なサンプリング範囲。
- 新しいコマンド **import** によるプラグインのサポート。外部の共有オブジェクトが提供する関数にユーザ定義関数名を割り当てます。

バージョン 5 と 6 との違い (2 Differences between versions 5 and 6)

バージョン 5 で導入したいくつかの変更は、gnuplot の以前のバージョン用のスクリプトを失敗させる、または異なる振舞いをさせることがありました。バージョン 6 で導入した変更では、そういうことはとても少ないです。

描画スタイルの詳細 (Plot style details)

完全に新しい描画スタイルの追加とは別に、バージョン 6 では幾つか既存の描画スタイルへの微調整も行われています。

- いくつかの出力形式 (png/jpeg/sixel/kitty) は、"dot" のサイズを現在の線幅 (line width) で伸縮します。その他のいくつかの出力形式では、既にそうなっています。
- 誤差棒 (errorbar) を描く描画スタイルは、データ点の位置に空白を置きます。その空白サイズは、**set pointintervalbox** で制御できます。
- 疑似ファイル '-' からのインラインデータの読み込みは、多重描画モード (multiplot) 内ではもはやできないので、代わりにデータブロックを利用してください (以下参照: **new multiplots (p. 32)**)。

非推奨な書式 (Deprecated syntax)

バージョン 5.4 では非推奨、6.0 では削除:

```
# 繰り返しを行うために 'reread' を含むファイルを使用
N = 0;  load "file-containing-reread";
file content:
    N = N+1
    plot func(N,x)
    pause -1
    if (N<5) reread
```

現在の同等の機能:

```
do for [N=1:5] {
    plot func(N, x)
    pause -1
}
```

バージョン 5.4 では非推奨、6.0 では削除

```
set dgrid3d ,,foo     # foo が意味する指示のキーワードがない
```

現在は以下と同等

```
set dgrid3d qnorm foo # (例のみ、qnorm は単独のオプションでない)
```

バージョン 5.0 では非推奨、6.0 では configure 時にオプション–enable-backward-compatibility を指定してビルドしたものでのみ有効

```
set style increment user
```

現在は以下と同等

```
必要な範囲の線種を "set linetype" を使用して再定義
明示的に "linestyle N" か "linestyle variable" を使用
```

バージョン 5.0 では非推奨、6.0 では削除

```
set clabel
```

現在は以下と同等

‘set clabel "format"‘ は ‘set cntrlabel format "format"‘ で置き換え、‘unset clabel‘ は ‘set cntrlabel onecolor‘ で置き換え。

バージョン 5.0 では非推奨、6.0 では削除

```
show palette fit2rgbformulae
```

デモ、ネット上のサンプル (Demos and Online Examples)

gnuplot の配布物の **demo** ディレクトリ内には、多くのサンプルが収められています。これらのサンプルの png, svg, canvas 出力形式による出力を、以下のネット上で見ることもできます: http://www.gnuplot.info/demos

そこでは、各デモを作成するコマンドがグラフの隣りに表示されますし、その gnuplot スクリプトをダウンロードすることもできますので、それを保存し同様のグラフを作成することができます。

バッチ/対話型操作 (Batch/Interactive)

gnuplot は、バッチ処理形式、あるいは対話型のどちらの形式でも実行でき、それらを組み合わせることも可能です。

コマンドライン引数は、プログラムへのオプションか、**gnuplot** コマンドを含むファイルの名前であると解釈します。各ファイルとコマンド文字列は、指定した順に実行します。特別なファイル名 "-" は、コマンドを標準入力から読込むことを意味します。最後のファイルを実行した後に **gnuplot** は終了します。読み込ませるファイル、およびコマンド文字列を指定しなかった場合は、**gnuplot** は標準入力からの対話型の入力を受け付けます。

コマンドラインオプション (command line options)

gnuplot は、コマンドラインで以下のオプションを受けつけます:

```
-V, --version
-h, --help
-p, --persist
-d, --default-settings
-s, --slow
-e  "command1; command2; ..."
-c  scriptfile ARG1 ARG2 ...
```

-p は、プログラムの終了時に、残っている対話型グラフウィンドウを一切閉じないよう gnuplot に指示します。

-d は、各ユーザ用、およびシステム用の初期化 (以下参照: **initialization** (**p. 80**)) を一切行わないよう gnuplot に指示します。

-s は、起動時のフォントの初期化をゆっくり待つように指示します。そうでないと、エラーを表示し、不正なフォントサイズ情報で動作を継続します。

-e "command" は、次に進む前に指定した単一のコマンドを実行するよう gnuplot に指示します。

-c は、-e "call scriptfile ARG1 ARG2 ..." と同等です。以下参照: **call (p. 134)**。

例 (Examples)

対話を開始する:

```
gnuplot
```

バッチモードで 2 つのコマンドファイル "input1", "input2" を実行:

```
gnuplot input1 input2
```

初期化ファイル "header" の後、対話型モードを起動し、その後別のコマンドファイル "tailer" を実行する:

```
gnuplot header - trailer
```

コマンドラインから **gnuplot** コマンドを直接与え、終了後にスクリーン上にグラフが残るようにオプション "-persist" を使う:

```
gnuplot -persist -e "set title 'Sine curve'; plot sin(x)"
```

ファイルのコマンドを実行する前に、ユーザ定義変数 a と s をセットする:

```
gnuplot -e "a=2; s='file.png'" input.gpl
```

キャンバスサイズ (Canvas size)

ここの文書で使用する "canvas" という用語は、グラフやそれに関連するラベルやタイトル、凡例などを配置するのに利用可能な描画領域全体を意味します。注意: HTML5 canvas 出力形式に関する情報を知りたい場合は、以下参照: **set term canvas (p. 345)**。

set term <terminal_type> size <XX>, <YY> は、出力ファイルのサイズ、または "キャンバス" のサイズを制御します。デフォルトでは、グラフはそのキャンバス全体に描画されます。

set size <XX>, <YY> は、描画自体をキャンバスのサイズに対して相対的に伸縮させます。1 より小さい伸縮値を指定すると、グラフはキャンバス全体を埋めず、1 より大きい伸縮値を指定すると、グラフの一部分のみがキャンバス全体に合うように描画されます。1 より大きい伸縮値を指定すると、問題が起こるかもしれないことに注意してください。

例:

```
set size 0.5, 0.5
set term png size 600, 400
set output "figure.png"
plot "data" with lines
```

このコマンドは、幅 600 ピクセル、高さ 400 ピクセルの出力ファイル"figure.png" を生成します。グラフはキャンバスの中の左下に置かれます。

注意: gnuplot の以前のバージョンでは、**set size** を出力キャンバスのサイズ自体を制御するのに使用する出力形式がありましたが、それはバージョン 4 で非推奨となっています。

コマンドライン編集 (Command-line-editing)

コマンドラインでの編集機能とコマンドヒストリの機能は、外部の GNU readline ライブラリか外部の BSD libedit ライブラリ、または組み込まれている同等のもののいずれかを使ってサポートしています。この選択は、gnuplot のコンパイル時の configure のオプションで行います。

組み込みの readline 版の場合の編集コマンドは以下の通りですが、DEL キーに関する動作はシステムに依存することに注意してください。GNU readline ライブラリと BSD libedit ライブラリに関しては、それ自身のドキュメントを参照してください。

コマンド行編集コマンド	
文字	機能
	行編集
`^B`	1 文字前へ戻す
`^F`	1 文字先へ進める
`^A`	行の先頭へ移動
`^E`	行の最後へ移動
`^H`	直前の文字を削除
`DEL`	現在の文字を削除
`^D`	現在位置の文字を削除、空行なら EOF
`^K`	現在位置から行末まで削除
`^L`	壊れた表示の行を再表示
`^U`	行全体の削除
`^W`	直前の単語を削除
`^V`	この次のキーを編集コマンドと見なさない
`TAB`	ファイル名補完動作
	履歴
`^P`	前の履歴へ移動
`^N`	次の履歴へ移動
`^R`	後方検索を開始

gnuplot へのコマンドラインを最初に読み込んだ時点、すなわちまだそれを解釈して実行する前の段階で、2 つの形式の単語の置換を行います。置換する文字列は、逆引用符 (バッククォート) 内のもの、または @ が頭についた文字列です。以下参照: **substitution (p. 82)**。

コメント (Comments)

コメント記号 # は、コマンド行中のほとんどどこにでも書くことができます。このとき **gnuplot** は、その行の残りの部分を無視します。ただし、記号 # は引用符内ではこの効果がありません。コメント行が '\' で終わっている場合、次の行もコメントの一部として扱われることに注意してください。

データファイルに対するコメント文字の指定については、以下参照: **set datafile commentschars** (p. 211)。

座標系 (Coordinates)

コマンド **set arrow**, **set key**, **set label**, **set object** はグラフ上の任意の位置が指定できます。その位置は以下の書式で指定します:

```
{<system>} <x>, {<system>} <y> {,{<system>} <z>}
```

各座標系指定 <system> には、**first**, **second**, **polar**, **graph**, **screen**, **character** のいずれかが入ります。

first は左と下の軸で定義される x,y (3D の場合は z も) の座標系を使用します。**second** は x2, y2 軸 (上と右の軸) を使用します。**graph** はグラフ描画領域内の相対的位置を指定し、左下が 0,0 で 右上が 1,1 (splot の場合はグラフ描画領域内の左下が 0,0,0 で、土台の位置は負の z の値を使用します。以下参照: **set xyplane** (p. 312)) となります。**screen** は表示範囲内 (範囲全体であり、**set size** で選択される一部分ではありません) を指定し、左下が 0,0 で 右上が 1,1 となります。**character** 座標は主にずれを指定するのに使用し、絶対的な位置を示すものではありません。**character** の水平、垂直サイズは、現在使用しているフォントに依存します。

polar は、最初の 2 つの値を、x, y ではなく、角 theta と半径 r であると解釈します。これは、例えば 2 次元の極座標、あるいは 3 次元円柱座標でのグラフにラベルを配置するのに役に立つでしょう。

x の座標系が指定されていない場合は **first** が使われます。y の座標系が指定されていない場合は x に対する座標系が使用されます。

与える座標が絶対的な位置ではなくて相対的な値である場合もあります (例えば **set arrow** ... **rto** の 2 番目の数値)。そのほとんどが、与えられた数値を最初の位置に対する差として使います。与えられた座標が対数軸内にある場合は、その相対的な値は倍率として解釈されます。例えば

```
set logscale x
set arrow 100,5 rto 10,2
```

は、x 軸が対数軸で y 軸が線形の軸なので、100,5 の位置から 1000,7 の位置への矢印を書くことになります。

一つ (あるいはそれ以上) の軸が時間軸である場合、**timefmt** の書式文字列に従って、引用符で囲まれた時間文字列で適切な座標を指定する必要があります。以下参照: **set xdata** (p. 302), **set timefmt** (p. 295)。また、**gnuplot** は整数表記も認めていて、その場合その整数は 1970 年 1 月 1 日からの秒数と解釈されます。

文字列データ (Datastrings)

データファイルには、ホワイトスペース (空白やタブ) を含まない任意の印字可能な文字列、あるいは 2 重引用符で囲まれた任意の文字列 (ホワイトスペースが含まれても良い)、のいずれかの形からなる文字列データを持たせることも可能です。データファイルに次のような行が含まれている場合、それは 4 つの列を含み、3 列目がテキスト部分であると見なされます:

```
1.000 2.000 "Third column is all of this text" 4.00
```

テキスト部分は 2 次元や 3 次元描画内で例えば以下のように使用されます:

```
plot 'datafile' using 1:2:4 with labels
splot 'datafile' using 1:2:3:4 with labels
```

テキスト部分の列データは 1 つ、または複数の描画軸の目盛りのラベルとして使用できます。次の例は、入力データの 3 列目と 4 列目を (X,Y) 座標として取り出し、それらの点の列を結ぶ線分を描画します。しかしこの場合 gnuplot は、x 軸に沿って標準的に間の空いた数字ラベルのついた目盛り刻みをつけるのではなく、入力データファイルの 1 行目の X 座標の位置に、目盛り刻みと文字列を x 軸に沿ってつけて行きます。

```
set xtics
plot 'datafile' using 3:4:xticlabels(1) with linespoints
```

入力データの列の最初のエントリ (すなわち列の見出し) をテキスト部分と解釈するもう一つのオプションがあり、それはテキスト部分を、その描画した列のデータの凡例 (key) のタイトル部分として使用します。次の例は、先頭の行の 2 列目の部分を凡例ボックス内のタイトルを生成するのに使用し、その後の列の 2,4 列目は要求された曲線を描画するのに処理されます:

```
plot 'datafile' using 1:(f($2)/$4) with lines title columnhead(2)
```

別の例:

```
plot for [i=2:6] 'datafile' using i title "Results for ".columnhead(i)
```

この列の先頭を使用する方法は、**set datafile columnheaders** か **set key autotitle columnhead** で自動化できます。以下参照: **labels (p. 115)**, **using xticlabels (p. 179)**, **plot title (p. 185)**, **using (p. 176)**, **key autotitle (p. 231)**。

拡張文字列処理モード (Enhanced text mode)

多くの出力形式が、拡張文字列処理モード (enhanced text mode) をサポートしています。これは、文字列に追加の書式情報を埋めこみます。例えば "x^2" は x の自乗を、通常我々が見る上付きの 2 がついた形で書き出します。このモードは、出力形式の設定時にデフォルトとして選択されますが、その後で **set termoption [no]enhanced** を使ってその機能を有効/無効にもできますし、**set label "x_2" noenhanced** のように個別の文字列に対して無効にすることもできます。

注意: TeX ベースの出力形式 (例えば cairolatex, pict2e, pslatex, tikz) の出力では、すべてのテキスト文字列には、これの代わりに TeX/LaTeX の書式を使用すべきです。以下参照: **latex (p. 366)**。

拡張文字列制御記号			
制御記号	例	結果	説明
^	a^x	a^x	上付き文字
_	a_x	a_x	下付き文字
@	a@^b_{cd}	a^b_{cd}	空ボックス (幅がない)
&	d&{space}b	d␣␣␣␣␣b	指定した長さのスペースを挿入
~	~a{.8-}	$\tilde{a}$	'a' の上に '-' を、現在のフォントサイズの .8 倍持ち上げた位置に重ね書き
	{/Times abc}	abc	Times フォント、今のサイズで abc を出力
	{/Times*2 abc}	abc	Times フォント、今の倍のサイズで abc
	{/Times:Italic abc}	*abc*	Times フォント、イタリック体で abc
	{/Arial:Bold=20 abc}	**abc**	Arial フォント、太字、サイズ 20 で abc
\U+	\U+221E	∞	Unicode コードポイント U+221E 無限大

書式制御文字は、それに続く 1 文字、または中カッコで囲まれたものに適用されます。中カッコ内には、例えば 2^{10} のような追加の書式文字列のない文字列か、またはフォントの属性を変更する追加制御文字列を入れることができます。フォント指定は、開き中カッコ '{' の直後に続く '/' のすぐ次に書かなければ「いけません」。フォント名にスペースが含まれる場合、それを単一、または二重引用符で囲まなければいけません。

例: 最初の例はその中カッコの入れ子を示していて、ボールド体の A にイタリック体の下付きの添字 i がついたものが、いずれも現在のフォントで描かれます。この例の :Normal を取ると、下付きの添字はボールド体でかつイタリック体になります。2 つ目の例は同じ書式制御を 20 ポイントサイズの"Times New Roman" フォントに適用したものです。

```
{/:Bold A_{/:Normal{/:Italic i}}}
{/"Times New Roman":Bold=20 A_{/:Normal{/:Italic i}}}
```

空ボックス (phantom box) は a@^b_c の上付き文字と下付き文字を揃えるときに有用ですが、文字にダイアクリティカルマークを重ねる場合にはうまく働きません。その目的のためには、アクセントやその他のダイアクリティカルマークのある文字を持つエンコード (例えば utf8) を使用する方がいいでしょう。以下参照: **set encoding (p. 215)**。そのボックスはスペーシングが行なわれないので、ボックス内 (つまり @ の後ろ) の上付き文字や下付き文字を短く出力するのに適しています。

ある文字列と同じ長さのスペースを文字 '&' を使うことで入れることができます。すなわち、

```
'abc&{def}ghi'
```

は以下を生成します (abc と ghi の間は 3 文字分の空白):

```
'abc   ghi'
```

文字 '~ ' は、次の文字、またはカッコで囲まれた文字列に、それに続く文字またはカッコで囲まれた文字列を重ね書きします。2 番目の文字は最初の文字にあわせて水平方向にセンタリングされます。よって '~ a/' は 'a' を貫くようなスラッシュが得られます。2 番目の文字は、その前に数字を置くことで垂直方向に移動させることができます。その数字は現在のフォントサイズに対する割合を意味し、それに従って文字が上がったり下がったりします。この場合数字と文字列は 1 文字より長くなるのでカッコで囲む必要があります。重ね書きされる文字列が数字から始まっている場合は、垂直にずらす値と文字列との間にスペースを入れてください ('~ {abc}{.5 000}')。それ以外はスペースは不要です ('~ {abc}{.5 — }')。一方、あるいは両方のフォントを変更することもできます ('~ a{.5 /*.2 o}'; 'a' その 1/5 の大きさの 'o'、この場合数字とスラッシュの間のスペー

スは必要です) が、その文字列が始まった後で変更することはできません。それぞれの文字列内で、他の特殊な書式を使うこともできません。制御文字はエスケープしないといけません。例: サーカムフレックス付きの a (\U+00E2) を印字するには '˜ a{.8\^}' とします。

二重引用符内の文字列は単一引用符内の文字列とは異なって解釈されることに注意してください。主な違いは、二重引用符内の文字列ではバックスラッシュは 2 つ重ねる必要があることです。

gnuplot ソース配布物内の /docs/psdoc サブディレクトリにあるファイル"ps_guide.ps" に、拡張された書式に関する例が載っています。同様のものがデモ **enhanced_utf8.dem**

にもあります。

エスケープシーケンス (escape sequences)

バックスラッシュ文字 \ は、単一文字を意味するエスケープシーケンスの開始意味します。

\X の形式 (X は 1 文字) は、単一引用符内と二重引用符内では別の処理をします。以下参照: **quotes (p. 85)**。非拡張文字テキスト (non-enhanced text) での単一引用符内の "\X" は文字通りに使用し、二重引用符内では、\n (改行) \r (復帰) \t (タブ) 等の特別な文字以外の "\X" は "X" になります。

\ooo の形式 (ooo は 3 桁の 8 進数値) は、特定のフォントエンコード内の文字コード番号を指し示すのに使えます。これは PostScript 出力形式で最も有用ですが、それはそれが UTF-8 エンコーディングを簡単に処理できないからです。例えば、PostScript 出力は、歴史的な理由で Adobe Symbol フォントに依っていますが、それは無限大の記号を 8 進 245 番で示すようなカスタムエンコードを使用しています。これは、拡張文字列 (enhanced text) としてフォント名と文字コードを "{/Symbol \245}" のように指定することで埋め込むことができます。この仕組みは、UTF-8 でないフォントやロケールでの文字アクセスについても有用かもしれません。

\U+hhhh の形式の Unicode のコードポイントで文字を指定することができます。ここで hhhh は 16 進値の 4 または 5 文字です。例えば、無限大の記号 ∞ のコードポイントは \U+221E です。これは、必要ならば出力時に UTF-8 のバイト列に変換されます。UTF-8 環境では、印字可能な特殊文字は他の文字と同様に文字列内で処理できるので、この仕組みは必要ありません。しかしこれは結合文字や発音区別符号 (例えばベクトルを意味するための文字の上の矢印など) には有用です。以下参照: **utf8 (p. 215)**, **string encoding (p. 82)**。またはオンラインユニコードデモ

を参照。

Unicode のコードポイントは 4 桁か 5 桁になるので、書式があいまいになる可能性があり、4 桁のエスケープシーケンスの直後に文字がつく場合は、それを 5 桁のエスケープシーケンスと間違って解釈するかもしれません。そのような場合は、エスケープシーケンスとそれに続く文字を明確に分離する必要があります。

```
例:  以下のラベルを作成 "α 1 + α 2"
拡張文字列:    set label "{\U+03B1}1 + {\U+03B1}2"   enhanced
非拡張文字列:  set label "\U+03B1\1 + \U+03B1\2"   noenhanced
```

Unicode エスケープコードは、解釈されずに文字列に保存されることに注意してください。エスケープコードは、その文字列が出力されるか部分文字列処理の評価の際に、それが示す文字に置き換えます。よって、文字列 S = "A + \U+03A3 B" の定義後では、S[5:5] は "\" でなく "Σ " と評価しますが、(S eq S[1:*]) は FALSE と評価します。それは、S には処理されていないエスケープシーケンス自身が含まれるのに対し、S[1:*] の評

価の際にはエスケープシーケンスが Σ に対する UTF-8 のバイト列に置き換わるからです。plot で使用する際は、どちらも "A + Σ B" と表示されます。

環境変数 (Environment)

gnuplot は多くのシェル環境変数を認識します。必須のものはありません。

GNUTERM は、それが定義されていれば、起動時に "set term" に渡されます。これは、システム、または個人的な初期化ファイルによる指定 (以下参照: **startup (p. 80)**)。や、もちろんその後の明示的な **set term** コマンドによる指定で変更できます。terminal オプションを入れることもできます。例:

```
bash$ export GNUTERM="postscript eps color size 5in, 3in"
```

GNUHELP は、それが定義されていれば、ヘルプファイル (gnuplot.gih) のパス名に設定します。

起動時の初期化には、設定ファイル $HOME/.gnuplot と$XDG_CONFIG_HOME/gnuplot/gnuplotrc を探します。MS-DOS, Windows, OS/2 では GNUPLOT か USERPROFILE で指定されたファイルを探します。詳細については以下参照: **startup (p. 80)**。

Unix においては、PAGER がヘルプメッセージの出力用のフィルタとして使われます。

Unix では、SHELL が **shell** コマンドの際に使われます。MS-DOS, OS/2 では COMSPEC が使われます。

FIT_SCRIPT は、当てはめ (fit) が中断されたときに実行する **gnuplot** コマンドの指定に使われます。以下参照: **fit (p. 137)**。FIT_LOG は当てはめによるログファイルのデフォルトのファイル名の指定に使われます。

GNUPLOT_LIB は、データやコマンドファイルの検索ディレクトリを追加定義するのに使われます。その変数は、一つのディレクトリ名かまたは複数のディレクトリ名を書くことができますが、ディレクトリの区切りはプラットホーム毎に違います。例えば Unix では ':' で、MS-DOS, Windows, OS/2 では ';' です。GNUPLOT_LIB の値は変数 **loadpath** に追加されますが、それは **save** や **save set** コマンドでは保存されません。

出力ドライバの中には gd ライブラリ経由で TrueType フォントを扱えるものもいくつかあります (以下参照: **fonts (p. 66)**)。これらの出力形式では、GDFONTPATH や GNUPLOT_DEFAULT_GDFONT がフォントの選択に影響を与えます。

postscript 出力ドライバは自分で持っているフォント検索パスを使いますが、それは、環境変数 GNUPLOT_FONTPATH で制御できます。

PostScript ドライバは、外部 (組み込まれていない) 定義ファイルを探すために環境変数 GNUPLOT_PS_DIR を利用します。インストール時の作業により、gnuplot にはそれらのファイルのコピーが組み込まれているか、またはデフォルトのパスが埋め込まれています。この変数は、postscript 出力形式でデフォルトのファイルの代わりにカスタマイズした prologue ファイルを使用するのに利用できます。以下参照: **postscript prologue (p. 382)**。

式 (Expressions)

基本的には C, FORTRAN, Pascal, BASIC において利用可能な数学表現を使用できます。 演算子の優先順位は C 言語の仕様に従います。数式中の空白文字とタブ文字は無視されます。

gnuplot は "実数" と "整数" 演算を FORTRAN や C のように扱うということに注意してください。"1", "-10" などは整数と見なされ、"1.0", "-10.0", "1e1", 3.5e-1 などは実数と見なされます。 これら 2 つのもっとも重要な違いは割算です。整数の割算は切り捨てられます: $5/2 = 2$。実数はそうではありません: $5.0/2.0 = 2.5$。それらが混在した式の場合、計算の前に整数は実数に "拡張" されます: $5/2e0 = 2.5$。負の整数を正の整数で割る場合、その値はコンパイラによって変わります。"print -5/2" として、あなたのシステムが常に切り捨てる (-5/2 で -3 になる) のか、または 0 の近くに丸める (-5/2 で -2 になる) のかを確認してください。

数式 "1/0" は "未定義値 (undefined)" フラグを生成し、それによりその点を無視します。あるいは、あらかじめ定義されている値 NaN を使っても同じことになります。例については、以下参照: **using (p. 176)**。

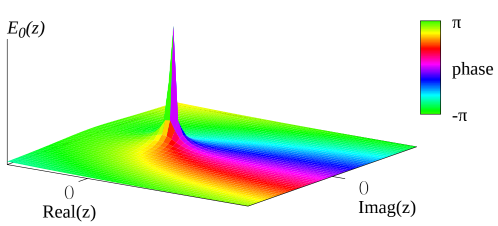
gnuplot は文字列に対する単純な演算、および文字列変数も利用できます。例えば式 ("A" . "B" eq "AB") は真と評価されますが、これは文字列の結合演算子と文字列の等号演算子を意味しています。

数としての値を含む文字列は、それが数式で利用された場合は、対応する整数や実数に変換されます。よって、("3" + "4" == 7) や (6.78 == "6.78") はどちらも真になります。整数は、それが文字列結合演算子で使われた場合は文字列に変換されますが、実数や複素数はダメです。典型的な例は、ファイル名や他の文字列内に整数を使う場合でしょう: 例えば ("file" . 4 eq "file4") は真です。

後置指定する範囲記述子 [beg:end] によって、部分文字列を指定することができます。例えば、"ABCDEF"[3:4] == "CD" で、"ABCDEF"[4:*] == "DEF" です。書式 "string"[beg:end] は、文字列値の組み込み関数 substr("strings",beg,end) を呼ぶこととほぼ同じですが、関数呼び出しでは beg, end は省略することはできません。

複素数値 (Complex values)

計算の演算子とほとんどの組み込み関数は複素数引数の使用をサポートしています。複素定数は、{<real>,<imag>} と表記し、<real> と <imag> は、数定数である必要があります。よって、{0,1} は 'i' を意味します。現在の gnuplot は、あらかじめ I = {0,1} を変数として定義していて、他の変数から複素数値を作るのに使えるようにしています。つまり、**x + y*I** は正しい数式なのですが、**{x,y}** はそうではありません。複素数値 z の実数部分と虚数部分は、real(z), imag(z) として取り出せます。長さは abs(z) で、偏角は arg(z) で得られます。



gnuplot の 2 次元と 3 次元の描画スタイルは、実数値を仮定しています。よって 0 でない虚数部分を持つ複素数値関数 f(x) を描画する場合は、実数部分や虚数部分、あるいは長さや偏角を描画させなければいけません。例えば複素引数に対する関数 f(z) の複素数値の長さと偏角を表示させるには、長さを曲面の高さ、偏角を色で表示するという手があります。その場合、HSV 色空間のカラーパレットを用いて、0 から 1 の範囲の H 成分 (色相) を arg(z) が返す偏角の範囲 [-π : π] に割り当て、偏角が 1 周したら色も巻き戻るようにすると便利でしょう。デフォルトでは、これは H = 0 (赤) から始まりますが、**set palette** の **start** キーワードを使ってその開始位置を変更し、H の他の値を 0 に割り当てることも可能です。以下の例は、H = 0.3 (緑) から開始し巻き戻るようにしています。以下参照: **set palette defined (p. 260)**, **arg (p. 48)**, **set angles (p. 194)**。

```
set palette model HSV start 0.3 defined (0 0 1 1, 1 1 1 1)
```

```
set cbrange [-pi:pi]
set cbtics ("-π " -pi, "π " pi)
set pm3d corners2color c1
E0(z) = exp(-z)/z
I = {0,1}
splot '++' using 1:2:(abs(E0(x+I*y))):(arg(E0(x+I*y))) with pm3d
```

定数 (Constants)

整数定数は、C の strtoll() ライブラリルーチンを使って解釈しますが、これは、"0" で始まる定数は 8 進数と、また "0x" か "0X" で始まる定数は 16 進数とみなすことを意味します。

実数 (浮動小数) 定数は、C の atof() ライブラリルーチンを使って解釈します。

複素数の定数は {<real>,<imag>} と表現します。ここで <real> と <imag> (実部、虚部) は数値定数である必要があります。例えば、{0,1} は 'i' 自身を表し、{3,2} は 3 + 2i を表します。これらには明示的に中カッコを使う必要があります。この gnuplot では、あらかじめ I = {0,1} を変数として定義して、明示的な形式の入力を避ける工夫をしています。例えば、**3 + 2*I** は **{3,2}** と同じですが、こちらは虚数成分に変数係数を使えるところが優位です。すなわち、**x + y*I** は正しい数式ですが、**{x,y}** はそうではありません。

文字列定数は単一引用符か二重引用符のいずれかで囲まれた任意の文字の並びからなるものです。単一引用符と二重引用符の違いは重要です。以下参照: **quotes (p. 85)**。

例:

```
1 -10 0xffaabb        # 整数定数
1.0 -10. 1e1 3.5e-1   # 実数定数
{1.2, -3.4}           # 複素数定数
"Line 1\nLine 2"      # 文字列定数 (\n は改行に展開される)
'123\na456'           # 文字列定数 (\ と n はそのままの文字)
```

関数 (Functions)

特に注意がなければ、**gnuplot** の数学関数の引数は整数、実数、複素数の値を取ることができます。角を引数や戻り値とする関数 (例えば sin(x)) は、その値をラジアンとして扱いますが、これはコマンド **set angles** によって度に変更できます。

数学ライブラリ、組み込み関数			
関数	引数		戻り値 (c は複素数)
abs(x)	整数または実数		x の絶対値, $\lvert x\rvert$
abs(x)	複素数		x の長さ, $\sqrt{\mathrm{real}(x)^2+\mathrm{imag}(x)^2}$
acos(x)		c	$\cos^{-1}x$ (アークコサイン)
acosh(x)		c	$\cosh^{-1}x$ (逆双曲余弦)
airy(x)	実数		実数 x に対するエアリー関数 Ai(x)
arg(x)	複素数		x の偏角, $-\pi \le \mathrm{arg}(x) \le \pi$
asin(x)		c	$\sin^{-1}x$ (アークサイン)
asinh(x)		c	$\sinh^{-1}x$ (逆双曲正弦)
atan(x)		c	$\tan^{-1}x$ (アークタンジェント)
atan2(y,x)	整数または実数		$\tan^{-1}(y/x)$ (アークタンジェント)
atanh(x)		c	$\tanh^{-1}x$ (逆双曲正接)
besj0(x)	実数		x ラジアンの J_0 ベッセル関数 (0 次ベッセル関数)
besj1(x)	実数		x ラジアンの J_1 ベッセル関数 (1 次ベッセル関数)
besjn(n,x)	整数, 実数		x ラジアンの J_n ベッセル関数 (n 次ベッセル関数)
besy0(x)	実数		x ラジアンの Y_0 ベッセル関数 (0 次ノイマン関数)
besy1(x)	実数		x ラジアンの Y_1 ベッセル関数 (1 次ノイマン関数)
besyn(n,x)	整数, 実数		x ラジアンの Y_n ベッセル関数 (n 次ノイマン関数)
besi0(x)	実数		x ラジアンの I_0 (0 次) 変形ベッセル関数
besi1(x)	実数		x ラジアンの I_1 (1 次) 変形ベッセル関数
besin(n,x)	整数, 実数		x ラジアンの I_n (n 次) 変形ベッセル関数
cbrt(x)	実数		x の三乗根 (定義域、値域は共に実数に限定)
ceil(x)			$\lceil x\rceil$, x の実部以上の最小の整数
conj(x)	複素数	c	x の複素共役
cos(x)		c	x のコサイン $\cos x$
cosh(x)		c	$\cosh x$, x ラジアンのハイパボリックコサイン
EllipticK(k)	実数 k ∈ (-1:1)		$K(k)$ 第 1 種完全楕円積分
EllipticE(k)	実数 k ∈ [-1:1]		$E(k)$ 第 2 種完全楕円積分
EllipticPi(n,k)	実数 n<1, 実数 k ∈ (-1:1)		$\Pi(n,k)$ 第 3 種完全楕円積分
erf(x)			erf(real(x)), x の 実部の誤差関数
erfc(x)			erfc(real(x)), 1.0 - (x の実部の誤差関数)
exp(x)		c	e^x, x の指数関数
expint(n,x)	整数 $n \ge 0$, 実数 $x \ge 0$		$E_n(x)=\int_1^\infty t^{-n}e^{-xt}\,dt$, x の指数積分
floor(x)			$\lfloor x\rfloor$, x の実部以下の最大の整数
gamma(x)			$\Gamma(x)$, x の実部のガンマ関数
ibeta(a,b,x)	$a,b>0$, $x\in[0:1]$		$B(a,b,x)=\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\int_0^x t^{a-1}(1-t)^{b-1}dt$, 不完全ベータ関数

数学ライブラリ、組み込み関数			
関数	引数		戻り値 (c は複素数)
inverf(x)			x の実部の逆誤差関数
igamma(a,z)	複素数, $\Re(a) > 0$	c	不完全ガンマ関数 $P(a,z) = \frac{1}{\Gamma(z)} \int_0^z t^{a-1} e^{-t} dt$
imag(x)	複素数		x の虚数部分 (実数)
int(x)	実数		x の 0 に向かって丸めた整数部分
invnorm(x)			x の実部の逆正規分布関数
invibeta(a,b,p)	実数		逆 (正規化) 不完全ベータ関数
invigamma(a,p)	実数		逆 (正規化) 不完全ガンマ関数
LambertW(z,k)	複素数, 整数	c	複素 Lambert W 関数の第 k 分岐
lambertw(x)	実数		Lambert W 関数の主値 (第 0 分岐)
lgamma(x)	実数		実数 x に対する $\ln \Gamma(x)$ (ガンマ対数関数)
lnGamma(x)	複素数	c	複素平面全体で正当な $\ln \Gamma(x)$
log(x)		c	$\log_e x$, x の自然対数 (底 e)
log10(x)		c	$\log_{10} x$, x の対数 (底 10)
norm(x)			x の実部の正規分布 (ガウス分布) 関数
rand(x)	整数		開区間 (0:1) 内の疑似乱数生成器
real(x)			x の実部
round(x)			$\lfloor x \rceil$, x の実部に一番近い整数
sgn(x)			$x > 0$ なら 1, $x < 0$ なら -1, $x = 0$ なら 0 (虚部は無視)
Sign(x)	複素数	c	$x = 0$ なら 0、それ以外は $x/\|x\|$
sin(x)		c	$\sin x$, x のサイン
sinh(x)		c	$\sinh x$, x ラジアンのハイパボリックサイン
sqrt(x)		c	$\sqrt{x}$, x の平方根
SynchrotronF(x)	実数		$F(x) = x \int_x^\infty K_{\frac{5}{3}}(\nu)\, d\nu$
tan(x)		c	$\tan x$, x のタンジェント
tanh(x)		c	$\tanh x$, x ラジアンのハイパボリックタンジェント
uigamma(a,x)	実数, 実数		上方不完全ガンマ関数 $Q(a,x) = \frac{1}{\Gamma(x)} \int_x^\infty t^{a-1} e^{-t} dt$
voigt(x,y)	実数		Voigt/Faddeeva 関数 $\frac{y}{\pi} \int \frac{exp(-t^2)}{(x-t)^2+y^2} dt$
			注意: $\mathrm{voigt}(x,y) = real(\mathrm{faddeeva}(x+iy))$
zeta(s)	複素数	c	リーマンゼータ関数 $\zeta(s) = \Sigma_{k=1}^\infty k^{-s}$

libcerf (利用可能な場合のみ) による特殊関数			
関数	引数		戻り値 (c は複素数)
cerf(z)	複素数	c	複素誤差関数 $cerf(z) = \frac{\sqrt{\pi}}{2} \int_0^z e^{-t^2} dt$
cdawson(z)	複素数	c	Dawson 積分 $D(z) = \frac{\sqrt{\pi}}{2} e^{-z^2} erfi(z)$ の複素拡張
faddeeva(z)	複素数	c	スケール化複素相補誤差関数 $w(z) = e^{-z^2}\ erfc(-iz)$
erfi(x)	実数		虚誤差関数 $erfi(x) = -i * erf(ix)$
FresnelC(x)	実数		フレネル積分 $C(x) = \int_0^x \cos(\frac{\pi}{2} t^2) dt$

libcerf (利用可能な場合のみ) による特殊関数		
関数	引数	戻り値 (c は複素数)
FresnelS(x)	実数	フレネル積分 $S(x) = \int_0^x \sin(\frac{\pi}{2}t^2)dt$
VP(x,σ,γ)	実数	Voigt プロファイル $VP(x,\sigma,\gamma) = \int_{-\infty}^{\infty} G(x';\sigma)L(x-x';\gamma)dx'$
VP_fwhm(σ,γ)	実数	Voigt プロファイルの半値全幅 (FWHM)

Amos ライブラリ (利用可能な場合のみ) による複素特殊関数			
関数	引数		戻り値 (c は複素数)
Ai(z)	複素数	c	複素エアリー関数 $Ai(z)$
Bi(z)	複素数	c	複素エアリー関数 $Bi(z)$
BesselH1(nu,z)	実数, 複素数	c	$H_\nu^{(1)}(z)$ 第 1 種ハンケル関数
BesselH2(nu,z)	実数, 複素数	c	$H_\nu^{(2)}(z)$ 第 2 種ハンケル関数
BesselJ(nu,z)	実数, 複素数	c	$J_\nu(z)$ 第 1 種ベッセル関数
BesselY(nu,z)	実数, 複素数	c	$Y_\nu(z)$ 第 2 種ベッセル関数
BesselI(nu,z)	実数, 複素数	c	$I_\nu(z)$ 第 1 種変形ベッセル関数
BesselK(nu,z)	実数, 複素数	c	$K_\nu(z)$ 第 2 種変形ベッセル関数
expint(n,z)	整数 $n \geq 0$, 複素数 z	c	$E_n(z) = \int_1^\infty t^{-n}e^{-zt}\,dt$, 指数積分

文字列関数		
関数	引数	戻り値
gprintf("format",x)	任意	gnuplot の書式解析器を適用した結果の文字列
sprintf("format",x,...)	複数個	C 言語の sprintf の返す文字列
strlen("string")	文字列	文字列中の文字数
strstrt("string","key")	文字列	部分文字列 "key" が現れる先頭位置
substr("string",beg,end)	複数個	文字列 "string"[beg:end]
split("string","sep")	文字列	部分文字列からなる配列
join(array,"sep")	配列, 文字列	配列要素を一つの文字列に結合
strftime("timeformat",t)	任意	gnuplot による時刻解析結果の文字列
strptime("timeformat",s)	文字列	文字列 s を変換した 1970 年からの秒数
system("command")	文字列	シェルコマンドの出力を持つ文字列
trim(" string ")	文字列	前後につく空白を取り除いた文字列
word("string",n)	文字列, 整数	文字列 "string" の n 番目の単語
words("string")	文字列	文字列 "string" 中の単語数

時刻関数		
関数	引数	戻り値
time(x)	任意	現在のシステム時刻 (秒単位)
timecolumn(N,"timeformat")	整数, 文字列	入力データの N 列目からの書式化日時データ
tm_hour(t)	秒数による時刻	時 (0..23)
tm_mday(t)	秒数による時刻	日 (1..31)
tm_min(t)	秒数による時刻	分 (0..59)
tm_mon(t)	秒数による時刻	月 (0..11)
tm_sec(t)	秒数による時刻	秒 (0..59)
tm_wday(t)	秒数による時刻	曜日 (日から土を 0..6 で)
tm_week(t)	秒数による時刻	ISO 8601 規則での週番号 (1..53)
tm_yday(t)	秒数による時刻	その年の何日目 (0..365)
tm_year(t)	秒数による時刻	西暦
weekdate_iso(year,week,day)	整数	ISO 8601 規則での週曜日に対応する時刻
weekdate_cdc(year,week,day)	整数	CDC による疫学的週曜日に対応する時刻

他の gnuplot の関数		
関数	引数	戻り値
column(x)	整数か文字列	データ入力中の x 列目の数値
columnhead(x)	整数	データファイルの最初の x 列目中の文字列
exists("X")	文字列	変数名 X が定義されていれば 1, そうでなければ 0
hsv2rgb(h,s,v)	h,s,v $\in$ [0:1]	24 ビット RGB 色値
index(A,x)	配列, 任意	A[i] = x となる整数 i。なければ 0。
palette(z)	実数	z に割り当てられた 24 ビット RGB パレット色
rgbcolor("name")	文字列	色名か文字列表現の色の 32 ビット ARGB 値
stringcolumn(x)	整数か文字列	文字列としての x 列目の内容
valid(x)	整数	データ入力中の x 列目の正当性
value("name")	文字列	名前 name の変数の現在の値
voxel(x,y,z)	実数	点 (x,y,z) を含む有効ボクセルの値

整数変換関数 (int floor ceil round) (integer conversion functions)

gnuplot の整数変数値は、使用環境が許せば、64 ビットの精度で保存します。

gnuplot の複素数変数値、実数変数値は、ほとんどの使用環境で IEEE754 の binary64 (double) 浮動小数形式で保存します。その精度は、53 ビットに制限され、有効数字はおよそ 16 桁です。

よって、絶対値が 2^53 よりも大きい整数は、浮動小数変数で一意に表現することはできません。つまり、大きな N に対する int(real(N)) という操作は N に近いけれども N とは異なる整数を返す可能性があります。

さらに、浮動小数値を切り捨てにより整数値に変換する関数は、その値自身が小さくても、16 桁以上の精度に依存する操作では期待する値を得られない可能性があります。例えば、int(log10(0.1)) は、-1 でなく 0 を返しますが、それはその浮動小数表現が -0.9999999999999999... に等しいからです。以下も参照: **overflow (p. 257)**。

int(x) は、引数の 0 の方向に切り捨てた整数部分を返します。|x| > 2^63、すなわち整数値として大きすぎる場合は NaN を返します。|x| > 2^52 の場合は、返り値はある整数の近傍におさまりますが、浮動小数精度の

制限のためにそれらを区別できません。以下参照: **integer conversion** (p. 51)。

floor(x) は、x の実数部分以下の最大の整数を返します。|x| > 2^52 の場合、その値は一意に決定できませんので、その場合は NaN を返します。以下参照: **integer conversion** (p. 51)。

ceil(x) は、x の実数部分以上の最小の整数を返します。|x| > 2^52 の場合、その値は一意に決定できませんので、その場合は NaN を返します。以下参照: **integer conversion** (p. 51)。

round(x) は、x の実数部分に一番近い整数を返します。|x| > 2^52 の場合、その値は一意に決定できませんので、その場合は NaN を返します。以下参照: **integer conversion** (p. 51)。

種々の楕円積分 (elliptic integrals)

関数 **EllipticK(k)** は、第 1 種完全楕円積分、すなわち、関数 **(1 - k^2*sin^2(θ))^(-0.5)** の 0 からπ /2 までの範囲の広義積分の値を返します。**k** の定義域は -1 から 1 です (両端は含まない)。

$$\mathrm{EllipticK}(k) = \int_0^{\pi/2} \sqrt{1 - k^2 \sin^2\theta}^{\;-1}\, d\theta$$

関数 **EllipticE(k)** は、第 2 種完全楕円積分、すなわち、関数 **(1 - k^2*sin^2(θ))^(-0.5)** の 0 からπ /2 までの範囲の広義積分の値を返します。**k** の定義域は -1 から 1 です (両端も含む)。

$$\mathrm{EllipticE}(k) = \int_0^{\pi/2} \sqrt{1 - k^2 \sin^2\theta}\, d\theta$$

関数 **EllipticPi(n,k)** は、第 3 種完全楕円積分、すなわち関数 **(1 - k^2*sin^2(θ))^(-0.5) / (1 - n*sin^2(θ))** の 0 からπ /2 までの範囲の広義積分の値を返します。パラメータ **n** は 1 より小さく、**k** は-1 と 1 の間 (両端は含まない) でなければいけません。定義より、すべての正の **k** に対し EllipticPi(0,k) == EllipticK(k) であることに注意してください。

$$\mathrm{EllipticPi}(n,k) = \int_0^{\pi/2} \left[(1 - n\sin^2\theta)\sqrt{1 - k^2 \sin^2\theta}\,\right]^{-1} d\theta$$

楕円積分のアルゴリズム: B.C.Carlson 1995, Numerical Algorithms 10:13-26.

複素エアリー関数 (Complex Airy functions)

Ai(z) と **Bi(z)** は、複素引数 z のエアリー関数で、変形ベッセル関数 K と I を用いて計算されます。Donald E. Amos, Sandia National Laboratories, SAND85-1018 (1985) によるルーチンを含む外部ライブラリによってサポートしています。

$$\mathrm{Ai}(z) = \frac{1}{\pi}\sqrt{\frac{z}{3}}K_{1/3}(\zeta) \qquad \zeta = \frac{2}{3}z^{3/2}$$

$$\mathrm{Bi}(z) = \sqrt{\frac{z}{3}}\left[I_{-1/3}(\zeta) + I_{1/3}(\zeta)\right]$$

複素ベッセル関数 (Complex Bessel functions)

BesselJ(nu,z) は、実引数の nu と複素引数の z に対する第 1 種ベッセル関数 J_nu です。Donald E. Amos, Sandia National Laboratories, SAND85-1018 (1985) によるルーチンを含む外部ライブラリによってサポートしています。

BesselY(nu,z) は、実引数の nu と複素引数の z に対する第 2 種ベッセル関数 Y_nu です。Donald E. Amos, Sandia National Laboratories, SAND85-1018 (1985) によるルーチンを含む外部ライブラリによってサポートしています。

BesselI(nu,z) は、実引数の nu と複素引数の z に対する第 1 種変形ベッセル関数 I_nu です。Donald E. Amos, Sandia National Laboratories, SAND85-1018 (1985) によるルーチンを含む外部ライブラリによってサポートしています。

BesselK(nu,z) は、実引数の nu と複素引数の z に対する第 2 種変形ベッセル関数 K_nu です。Donald E. Amos, Sandia National Laboratories, SAND85-1018 (1985) によるルーチンを含む外部ライブラリによってサポートしています。

BesselH1(nu,z) と **BesselH2(nu,z)** は、それぞれ実引数の nu と複素引数の z に対する第 1 種、第 2 種のハンケル関数です。

```
H1(nu,z) = J(nu,z) + iY(nu,z)
H2(nu,z) = J(nu,z) - iY(nu,z)
```

Donald E. Amos, Sandia National Laboratories, SAND85-1018 (1985) によるルーチンを含む外部ライブラリによってサポートしています。

Expint

expint(n,z) は、0 以上の整数 n に対して、次数 n の指数積分を返します。これは、t^(-n) e^(-tz) dt の 1 から ∞ までの積分値です。

$$E_n(x) = \int_1^\infty t^{-n} e^{-xt}\, dt$$

あなたが使っている gnuplot が、Amos ライブラリによる複素関数のサポート付きでビルドされたものであれば、n>0 に対して、その評価には Amos ルーチンの cexint を使用します [Amos 1990 Algorithm 683, ACM Trans Math Software 16:178]。その場合、z は -pi < arg(z) <= pi の任意の複素数とできます。expint(0,z) は、exp(-z)/z と計算します。

Amos ライブラリのサポートがなければ、z は 0 以上の実数値に制限されます。

フレネル積分 (Fresnel integrals FresnelC(x) and FresnelS(x))

コサインとサインのフレネル積分は、複素誤差関数 erf(z) との関係を用いて計算します。erf(z) に依存するため、これらの積分は libcerf ライブラリのサポートがある場合にのみ利用できます。

$$C(x) = \int_0^x \cos(\tfrac{\pi}{2}t^2)dt \qquad S(x) = \int_0^x \sin(\tfrac{\pi}{2}t^2)dt$$

$$C(x) + iS(x) = \tfrac{1+i}{2}erf(z)、z = \tfrac{\sqrt{\pi}}{2}(1-i)x$$

Gamma

gamma(x) は、その引数の実数部分のガンマ関数値を返します。整数 n に対しては gamma(n+1) = n! です。引数が複素数の場合、その虚数部分は無視します。複素数引数に対しては、以下参照: **lnGamma (p. 55)**。

Igamma

igamma(a, z) は、下方 (正規化) 不完全ガンマ関数 P(a, z) を返します

[Abramowitz and Stegun (6.5.1); NIST DLMF 8.2.4]。複素関数サポートが

あれば、a と z は real(a) > 0 の複素数値も許されます。逆に、上方不完全ガンマ関数に関しては、以下参照: **uigamma (p. 57)**。

$$\text{igamma}(a,z) = P(a,z) = z^a\gamma^*(a,z) = \frac{1}{\Gamma(z)}\int_0^z t^{a-1}e^{-t}dt$$

a, z の値に依存して、以下の 4 つのアルゴリズムを使用します。
ケース (1) a が大きく (>100) (z-a)/a が小さい (<0.2) 場合、Numerical Recipes 第 3 版 6.2 節 (2007) にある係数での Gauss-Legendre 数値積分公式を利用します。
ケース (2) z > 1 で z > (a+2) の場合、Shea (1988) J. Royal Stat. Soc. Series C (Applied Statistics) 37:466-473 の連分数を利用します。
ケース (3) z < 0 で a < 75 で imag(a) == 0 の場合は、Abramowitz & Stegun (6.5.29) の数列を利用します。
ケース (4) その他の場合は、Peason の級数展開を利用します。

全平面では、ある領域でその収束は良くないことに注意してください。選択したアルゴリズムで 1.E-14 の範囲で収束しなければ、関数は NaN を返し、警告を表示します。

複素関数サポートがない場合は、定義域は実数の引数の a > 0, z >= 0 に制限されます。

Invigamma

逆不完全ガンマ関数 **invigamma(a,p)** は、p = igamma(a,z) となる z の値を返します。p は (0;1] に制限され、a は正の実数でなければいけません。gnuplot での実装は、a<1 に対する 1.e-16 から、a = 1.e10 に対する 5.e-6 までの相対精度を持ちます。

Ibeta

ibeta(a,b,x) は、実数引数 a,b > 0, [0;1] 内の x に対する、正規化下方不完全ベータ積分値を返します。

$$\text{ibeta}(a,b,x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\int_0^x t^{a-1}(1-t)^{b-1}dt$$

引数が複素数の場合、虚数部分は無視します。gnuplot での実装は、Cephes ライブラリ [Moshier 1989, "Methods and Programs for Mathematical Functions", Prentice-Hall] からのコードを利用しています。

Invibeta

逆不完全ベータ関数 **invibeta(a,b,p)** は、p = ibeta(a,b,z) となる z の値を返します。a, b は正の実数に、p は [0,1] 内の値に制限されています。a, b が 0 に近づくとき ($\lessapprox$ 0.05) invibeta() は 1.0 に近づくので、その相対精度は浮動小数精度に制限されることに注意してください。

LambertW

複素定義域と複素値域を持つ Lambert W 関数です。LambertW(z, k) は、方程式 W(z) * exp(W(z)) = z で定義される 関数 W の第 k 分岐を返します。その複素数値は、Corless et al [1996], Adv. Comp. Math 5:329 に記載されている Halley の方法を使って求めます。名目上の精度は 1.E-13 ですが、不連続、すなわち分岐点の近くでは収束は良くありません。

LnGamma

lnGamma(z) は、複素定義域と複素値域を持つ、ガンマ関数の自然対数値を返します。 Lanczos [1964], SIAM JNA 1:86-96 による 14 項近似による実装です。結果の虚数成分は、負の実軸部分を除く全体で連続な曲面を生成するよう位相をずらしています。

乱数の生成 (random)

関数 **rand()** は 0 と 1 の間の疑似乱数列を生成します。これは以下からのアルゴリズムを使用しています: P. L'Ecuyer and S. Cote, "Implementing a random number package with splitting facilities", ACM Transactions on Mathematical Software, 17:98-111 (1991).

```
rand(0)       内部に持つ 2 つの 32bit の種 (seed) の現在の値から生
              成される開区間 (0:1) 内の疑似乱数値を返す
rand(-1)      2 つの種の値を標準値に戻す
rand(x)       0 < x < 2^31-1 の整数なら種の両方を x に設定する
rand({x,y}) 0 < x,y < 2^31-1 の整数なら seed1 を x に seed2 を y
              に設定する
```

複素引数の特殊関数 (Special functions with complex arguments)

複素定義域を持ついくつかの特殊関数を、外部ライブラリを通して提供します。あなたが使っている gnuplot がこれらのライブラリへのリンクなしでビルドされている場合は、定義域が実数の場合のみサポートするか、または関数自体を全く提供しないかとなるでしょう。

libcerf (http://apps.jcns.fz-juelich.de/libcerf) を必要とする関数は、ビルド時の設定オプション **–with-libcerf** に依存し、これがデフォルトです。以下参照: **cerf (p. 49)**, **cdawson (p. 49)**, **faddeeva (p. 49)**, **erfi (p. 49)**, **VP (p. 50)**, **VP_fwhm (p. 50)**。

実数次 nu と複素引数の複素エアリー、ベッセル、ハンケル関数は、Douglas E. Amos, Sandia National Laboratories, SAND85-1018 (1985) によって実装されたルーチンを含むライブラリを必要とします。これらのルーチンは、netlib (http://netlib.sandia.gov) や、libopenspecfun (https://github.com/JuliaLang/openspecfun) にあります。これらに対応するビルド時の設定オプションは **–with-amos=<library directory>** です。以下参照: **Ai (p. 52)**, **Bi (p. 52)**, **BesselJ (p. 52)**, **BesselY (p. 52)**, **BesselI (p. 52)**, **BesselK (p. 53)**, **Hankel (p. 53)**。複素指数積分は、netlib, libamos では提供していますが、libopenspecfun にはありません。以下参照: **expint (p. 53)**。

Synchrotron function

(第 1) シンクロトロン関数 SynchrotronF(x) は、シンクロトロン放射のパワー分布スペクトルを、臨界フォトンエネルギー (臨界周波数 vc) の単位で与える x の関数として記述するものです。

$F(x) = x \int_x^\infty K_{5/3}(\nu)\ d\nu$ ここで、$K_{5/3}$ は第 2 種変形ベッセル関数。

1.E-15 まで正確な近似の Chebyshev 係数は、MacLead (2000) NuclInstMethPhysRes A443:540-545 から採用しています。

時刻関数 (Time functions)

Time 関数 **time(x)** は現在のシステム時刻を返します。この値は **strftime** 関数で日時文字列に変換できますし、**timecolumn** と組み合わせて相対的な日時グラフを作成するのにも使えます。引数の型はそれが返すものを決定します。引数が整数の場合は time() は現在の時刻を 1970 年 1 月 1 日からの整数として返し、引数が実数 (または複素数) ならば同様の値を実数として返します。引数が文字列ならば、それを書式文字列であるとみなし、書式化された日時文字列を提供するようそれを **strftime** に渡します。以下参照: **time_specifiers (p. 221)**, **timefmt (p. 295)**。

Timecolumn **timecolumn(N,"timeformat")** は、N 列目から始まる文字列データを日時データ値として読み、"timeformat" を使用して、それを "Unix エポック (1970 年 1 月 1 日) からの秒数" としてミリ秒精度で解釈します。書式パラメータの指定がない場合、デフォルトの **set timefmt** による文字列を使います。この関数は、plot か stats コマンドでの **using** 指定でのみ有効です。以下参照: **plot datafile using (p. 176)**。

Tm_structure gnuplot は、内部では時刻を、Unix エポック 1970 年 1 月 1 日からの秒数を表す 64 bit の浮動小数値として保持しています。これを時刻や日付として解釈するために、それを POSIX 標準の構造体 **struct_tm** に変換します。1 秒未満の秒数は、どんな場合でも tm_sec() からは取得できないことに注意してください。各要素には、以下の関数を利用して個別にアクセスできます。

- **tm_hour(t)** 整数 0–23 の範囲の時間
- **tm_mday(t)** 整数 1–31 の範囲のその月の日
- **tm_min(t)** 整数 0–59 の範囲の分
- **tm_mon(t)** 整数 0–11 の範囲のその年の月
- **tm_sec(t)** 整数 0–59 の範囲の秒
- **tm_wday(t)** 整数 0(日)– 6(土) の範囲の曜日
- **tm_yday(t)** 整数 0–365 の範囲のその年の日
- **tm_year(t)** 整数 西暦

Tm_week 関数 **tm_week(t, standard)** は、その引数を 1970 年 1 月 1 日からの秒数での時刻とみなします。なお、関数名から POSIX の tm 構造体のメンバと思うかもしれませんが、そうではありません。

standard = 0 の場合、これは ISO 8601 週曜日規則での週番号を返します。これは、gnuplot の時刻書式 %W に対応します。standard = 1 の場合、これは CDC (アメリカ疾病予防管理センター) 疫学的週曜日規則 (「疫学的週」) での週番号を返します。これは、gnuplot の時刻書式 %U に対応します。これらに対応する、週曜日からカレンダー時刻に変換する逆関数については、以下参照: **weekdate_iso (p. 57)**, **weekdate_cdc (p. 57)**。

解説: ISO の YYYY 年の 1 番の週は、YYYY 年 1 月 1 に一番近い月曜日から始まります。これは、前の年になる可能性もあります。例えば、2008 年 12 月 30 日火曜日は、ISO の週曜日では 2009-W01-2 (2009 年の週番号 1 の第 2 日) となります。逆に、1 月の 1 日から 3 日までは、ISO の週番号 1 の月曜より前になることがありえます。この場合、これらの日は、前の年の最後の週番号の週に含まれることになります。例えば、2021 年 1 月 1 日金曜日は、ISO の週曜日では 2020-W53-5 です。

アメリカ疾病予防管理センター (CDC) の定める疫学的週は、同様の週曜日規則ですが、ISO 規則とは、月曜開始ではなく、日曜開始であるところが違います。

Weekdate_iso 書式:

```
time = weekdate_iso( year, week [, day] )
```

この関数は、ISO 8601 週曜日での year(西暦), week(週番号), day(日番号) の要素を、Unix エポックの 1970 年 1 月 1 日からの秒数でのカレンダー時刻に変換します。週曜日系での名目上の年 (year) は、カレンダーでの年と必ずしも一致しないことに注意してください。週番号 week は 1 から 53 の間の整数です。日番号 day はオプションで、それが 0 かまたは省略した場合はその週の開始時刻を返し、そうでなければ day は 1 (月曜) から 7 (日曜) までの整数です。カレンダー日を、ISO 規則での週番号に変換する逆関数に関する情報については、以下参照: **tm_week (p. 56)**。

例:

```
# 1 列目に ISO 週番号を持つファイルからのデータの描画
#     週        感染者   死亡者
#     2020-05    432       1
calendar_date(w) = weekdate_iso( int(w[1:4]), int(w[6:7]) )
set xtics time format "%b\n%Y"
plot FILE using (calendar_date(strcol(1))) : 2   title columnhead
```

Weekdate_cdc 書式:

```
time = weekdate_cdc( year, week [, day] )
```

この関数は、CDC/MMWR (アメリカ疾病予防管理センター/疫学週報) の疫学的週曜日での year(西暦), week(週番号), day(日番号) の要素を、Unix エポックの 1970 年 1 月 1 日からの秒数でのカレンダー時刻に変換します。CDC 週曜日規則は、ISO 規則とは、週が 1 = 日曜から 7 = 土曜まで、と定義されている点に違いがあります。3 番目のパラメータが 0 または省略した場合は、その週の開始時刻が返ります。以下参照: **tm_week (p. 56)**, **weekdate_iso (p. 57)**。

Uigamma

uigamma(a, x) は、上方 (正規化) 不完全ガンマ関数 Q(a, x) を返します [NIST DLMF eq 8.2.4]。逆に、下方不完全ガンマ関数 P(a,x) に対しては、以下参照: **igamma (p. 53)**。
Q(a, x) + P(a, x) = 1 です。

$$\mathrm{uigamma}(a,z) = Q(a,x) = 1 - P(a,x) = \frac{1}{\Gamma(z)}\int_x^\infty t^{a-1}e^{-t}dt$$

現在の実装は、Cephes library (Moshier 2000) によるものです。定義域は、実数の a>0, 実数の x>=0 に制限されています。

Using 指定用関数 (using specifier functions)

以下の関数は、データ入力時のみ有効です。通常それは、**plot**, **splot**, **fit**, **stats** のいずれかのコマンド上の **using** 指定の入力列を使用する数式内で使用します。しかしその関数の適用範囲は、実際の plot コマンド文全体であり、例えばグラフタイトルの作成時の **columnhead** の使用も含まれます。

Column 関数 **column(x)** は、plot, splot, stats コマンドの一部としてのみ使います。これは、x 列目の内容を数値として評価します。文字列を持つと思われる列の場合は、代わりに stringcolumn(x) か、timecolumn(x, "timeformat") を使用してください。以下参照: **plot datafile using (p. 176)**, **stringcolumn (p. 58)**, **timecolumn (p. 56)**。

Columnhead 関数 **columnhead(x)** は、plot, splot, stats コマンドの一部としてのみ使います。これは、データファイルの最初の行の x 列目の内容を文字列として評価します。典型的には、これは先頭行をグラフタイトルとして展開するのに使用します。以下参照: **plot datafile using (p. 176)**。例:

```
set datafile columnheader
plot for [i=2:4] DATA using 1:i title columnhead(i)
```

Stringcolumn 関数 **stringcolumn(x)** は、データ描画か fit における **using** 指定でのみ使います。これは x 列目の内容を文字列として返します。**strcol(x)** は、**stringcolumn(x)** の省略形です。その文字列を、時間や日付と認識させたい場合は、代わりに timecolumn(x, "timeformat") を使用してください。以下参照: **plot datafile using (p. 176)**。

Valid 関数 **valid(x)** は、データ描画か fit における **using** 指定の式の中でしか使いません。これは、明らかな NaN 値や、入力列内の予期せぬゴミを検出したり、多分デフォルト値に置き換えたり、NaN を使ってさらに計算をすることを避けたりするのに使えます。欠損値 ("missing") と NaN (非数値) データ値はどちらも不正と認識されますが、以下は重要ですが、gnuplot がその列を本当に欠損値であると認識するか、または欠損値フラグを含んでいると認識した場合、valid() を使用する数式が呼び出される前に、その入力行は捨てられてしまう、ということに注意してください。以下参照: **plot datafile using (p. 176)**, **missing (p. 210)**。

例:

```
# 箱のある認識できない値を、無視する代わりに、期待値である定数
# prior として、箱全体に寄与するように扱う。
plot DATA using 1 : (valid(2) ? $2 : prior) smooth unique
```

Value

A がユーザー定義変数の名前であれば、B = value("A") は事実上 B = A と全く同じです。これは、変数の名前自身が文字列変数に収められている場合に有用です。以下参照: **user-defined variables (p. 63)**。これは、変数名をデータファイルから読み取ることも可能にします。引数が数式である場合、value() はその数式の値を返します。引数が文字列で、定義されている変数に対応するものがない場合、value() は NaN を返します。

単語の取り出しと単語数 **(word, words)**

word("string",n) は、文字列 (string) の n 番目の単語を返します。例えば **word("one two three",2)** は文字列 "two" を返します。

words("string") は、文字列 (string) の単語数を返します。例えば、**words(" a b c d")** は 4 を返します。

単語は、空白区切りでなければいけません。もし、他の文字で区切られた文字列から個々のフィールドを取り出したいなら、代わりに **split** を使用してください。

関数 **word** と **words** は、単一引用符、二重引用符で囲まれた文字列も、限定的ですがサポートしています:

```
print words("\"double quotes\" or 'single quotes'")   # 3
```

開始引用符の前は、スペースか、または文字列の先頭でなければいけません。これは、単語内、あるいは単語終わりにつくアポストロフィー (') は、それぞれの単語の要素であると見なされることを意味します:

```
print words("Alexis' phone doesn't work") # 4
```

引用符文字のエスケープはサポートしていませんので、ある引用符を維持したい場合は、それぞれを別の種類の引用符で囲まなければいけません:

```
s = "Keep \"'single quotes'\" or '\"double quotes\"'"
print word(s, 2) # 'single quotes'
print word(s, 4) # "double quotes"
```

最後の例では、引用符のエスケープが文字列の定義時のみに必要であることに注意してください。

split("string", "sep") は、**split("string", "sep")** は、"sep" 内の文字をフィールドの区切りとして使用し、文字列 "string" の中身を個々のフィールドに切り分けます。これは、その要素が元の文字列のフィールドにそれぞれ対応する文字列の配列を返します。2 つ目のパラメータ "sep" はオプションで、"sep" を省略した場合、または空白文字一つである場合は、フィールド文字列を任意個のホワイトスペース (スペース、タブ、改ページ、改行、復帰) で切り分けます。それ以外の場合は、区切りは "sep" 内の完全な文字列にマッチする必要があります。

以下の 3 つの例は、いずれも配列 ["A", "B", "C", "D"] を生成します。

```
t1 = split( "A B C D" )
t2 = split( "A B C D", " ")
t3 = split( "A;B;C;D", ";")
```

しかし、以下のコマンド

```
t4 = split( "A;B; C;D", "; " )
```

は、2 つの文字列のみを持つ配列 ["A;B", "C;D"] を生成しますが、それは、2 文字のフィールド区切り文字列 "; " が 1 つしか見つからないからです。

注意: 文字列を、1 文字ずつの配列に保存するために、区切りとして空文字を設定することは、現在は実装されていません。それは、代わりに 1 文字の部分文字列を使うことで実現できます: Array[i] = "string"[i:i]

join(array, "sep") は、配列の文字列要素を、"sep" の文字列で区切られたフィールドの列として一つの文字列に連結します。文字列でない配列要素は、空のフィールドを生成します。この逆に **split** 関数は一つの文字列を複数のフィールドに切り分けて一つの配列を生成します。例:

```
array A = ["A", "B", , 7, "E"]
print join(A,";")
      A;B;;;E
```

trim(" padded string ") は、元の文字列の前後にある空白部分を取り除いた文字列を返します。これは、余計な空白を持ちうる入力データ列の文字列同士を比較する際に有用です。例:

```
plot FOO using 1:( trim(strcol(3)) eq "A" ? $2 : NaN )
```

Zeta

zeta(s) は、複素変数、複素数値のリーマンゼータ関数です。 $\zeta(s) = \Sigma_{k=1}^{\infty} k^{-s}$

この実装は、P. Borwein [2000] Canadian Mathematical Society Conference Proceedings でアルゴリズム 3 として記述されている多項式級数を使用しています。名目上の精度は、複素平面上で 1.e-16 です。しかし、これは、ゼータ関数の自明でない零点が完全に 0 と評価されることを保証はしません。

演算子 (operators)

gnuplot の演算子は、C 言語の演算子とほぼ同じですが、特に注意がなければ全ての演算子が整数、実数、複素数の引数を取ることができます。また、FORTRAN で使える ** (累乗) 演算子もサポートされています。

演算子の優先順位は Fortran や C と同じです。それらの言語同様、演算の評価される順序を変えるためにかっこが使われます。よって -2**2 = -4 で、(-2)**2 = 4 です。

単項演算子 (Unary)

以下は、単項演算子の一覧です:

単項演算子		
記号	例	説明
-	-a	マイナス符号
+	+a	プラス符号 (何もしない)
~	~a	* 1 の補数 (ビット反転)
!	!a	* 論理的否定
!	a!	* 階乗
$	$3	* ‘using‘ 指定上のデータ列
|	|A|	配列 A の要素数

説明に星印 (*) のついた演算子の引数は整数でなければなりません。

階乗演算子は、N! が十分小さければ (64 ビット整数では N <= 20) 整数を返し、大きな N の値に対しては実数での近似値を返します。

基数オペレータ |...| は、配列 A の要素数 |A| を返します。データブロック $DATA に適用した場合は、|$DATA| はデータ行数を返します。

二項演算子 (Binary)

以下は、二項演算子の一覧です:

二項演算子		
記号	例	説明
**	a**b	累乗
*	a*b	積
/	a/b	商
%	a%b	* 余り
+	a+b	和
-	a-b	差
==	a==b	等しい
!=	a!=b	等しくない
<	a<b	より小さい
<=	a<=b	以下
>	a>b	より大きい
>=	a>=b	以上
<<	0xff<<1	符号なし左シフト
>>	0xff>>1	符号なし右シフト
&	a&b	* ビット積 (AND)
^	a^b	* ビット排他的論理和 (XOR)
|	a|b	* ビット和 (OR)
&&	a&&b	* 論理的 AND
||	a||b	* 論理的 OR
=	a = b	代入
,	(a,b)	累次評価
.	A.B	文字列の連結
eq	A eq B	文字列が等しい
ne	A ne B	文字列が等しくない

説明に星印 (*) のついた演算子の引数は整数でなければなりません。大文字の A,B は演算子が文字列引数を要求することを意味します。

論理演算子の AND (&&) と OR (||) は C 言語同様に必要最小限の評価しかしません。すなわち、&& の第 2 引数は、第 1 引数が偽ならば評価されませんし、|| の第 2 引数は、第 1 引数が真ならば評価されません。

累次評価 (,) は、カッコの中でのみ評価され、左から右へ順に実行することが保証され、最も右の式の値が返されます。

三項演算子 (Ternary)

一つだけ三項演算子があります:

三項演算子		
記号	例	説明
?:	a?b:c	三項演算子

三項演算子は C のものと同じ働きをします。最初の引数 (a) は整数でなければいけません。この値が評価され、それが真 (ゼロでない) ならば 2 番目の引数 (b) が評価されその値が返され、そうでなければ 3 番目の引数 (c) が評価され、その値が返されます。

三項演算子は、区分的に定義された関数や、ある条件が満たされた場合にのみ点を描画する、といったことを行なう場合に有用です。

例:

0 <= x < 1 では sin(x) に、1 <= x < 2 では 1/x に等しくて、それ以外の x では定義されない関数を描画:

```
f(x) = 0<=x && x<1 ? sin(x) : 1<=x && x<2 ? 1/x : 1/0
plot f(x)
```

gnuplot は描画時は未定義値に対しては何も表示せずにただ無視するので、最後の場合の関数 (1/0) は点を全く出力しないことに注意してください。また、この関数描画の描画スタイルが lines (線描画) の場合、不連続点 (x=1) の所も連続関数として線が結ばれることにも注意してください。その点を不連続になるようにするには、関数を 2 つの部分それぞれに分けてください。

ファイル 'file' のデータで、4 列目のデータが負でないときだけ、1 列目のデータに関する 2 列目と 3 列目のデータの平均値を描画:

```
plot 'file' using 1:( $4<0 ? 1/0 : ($2+$3)/2 )
```

using の書式の説明に関しては、以下参照: **plot datafile using (p. 176)**。

和と積 (summation and cumulative product)

gnuplot は繰り返しの和 Σ と繰り返しの積 Π 用の演算子を提供します。

#TeX \hspace{2 cm} $\sum_{i=min}^{max} f(i)$ ~ ~ ~ and~ ~ ~ $\prod_{i=min}^{max} f(i)$

和の式は、以下の形式で表します:

```
sum [<var> = <start> : <end>] <expression>
```

ここで <var> は、<start> から <end> までの整数値を順に取る整数変数として扱われます。その各値に対して、式 <expression> の値が合計値に追加され、最終的な合計値がこの和の式の値となります。例:

```
print sum [i=1:10] i
    55.
# 以下は plot 'data' using 1:($2+$3+$4+$5+$6+...) と同等
plot 'data' using 1 : (sum [col=2:MAXCOL] column(col))
```

<expression> は、必ずしも変数 <var> を含む必要はありません。<start> と <end> は変数値や数式で指定もできますが、それらの値は動的に変更することはできません。そうでないと副作用が起こり得ます。<end> が <start> より小さい場合は、和の値は 0 となります。

繰り返しの積に対する同様の書式は以下の形式です。

```
prod [<var> = <start> : <end>] <expression>
```

よって以下のようになります。

```
N! == prod [i=1:N] i
```

<end> が <start> より小さい場合は、繰り返しは行われず、その積の値は 1.0 となります。

定義済み変数 (Gnuplot-defined variables)

gnuplot は、プログラムの現在の内部状態と直前の描画を反映するような読み出し専用の変数をいくつか持っています。これらの変数の名前は、例えば GPVAL_TERM, GPVAL_X_MIN, GPVAL_X_MAX, GPVAL_Y_MIN のように "GPVAL_" で始まります。これらすべての一覧とその値を見るには、**show variables all** と入力してください。ただし、軸のパラメータに関連する値 (範囲、対数軸であるか等) は、現在 set したものではなく、最後に描画されたものが使用されます。

例: 点 [X,Y] のスクリーン比での座標を計算する方法

```
GRAPH_X = (X - GPVAL_X_MIN) / (GPVAL_X_MAX - GPVAL_X_MIN)
GRAPH_Y = (Y - GPVAL_Y_MIN) / (GPVAL_Y_MAX - GPVAL_Y_MIN)
SCREEN_X = GPVAL_TERM_XMIN + GRAPH_X * (GPVAL_TERM_XMAX - GPVAL_TERM_XMIN)
SCREEN_Y = GPVAL_TERM_YMIN + GRAPH_Y * (GPVAL_TERM_YMAX - GPVAL_TERM_YMIN)
FRAC_X = SCREEN_X * GPVAL_TERM_SCALE / GPVAL_TERM_XSIZE
FRAC_Y = SCREEN_Y * GPVAL_TERM_SCALE / GPVAL_TERM_YSIZE
```

読み出し専用変数 GPVAL_ERRNO は、任意の gnuplot コマンドがあるエラーのために早く終わってしまった場合に 0 でない値にセットされ、直前のエラーメッセージは文字列変数 GPVAL_ERRMSG に保存されます。GPVAL_ERRNO と GPVAL_ERRMSG は、コマンド **reset errors** を使ってクリアできます。

mouse 機能が使える対話型入出力形式は、"MOUSE_" で始まる読み出し専用変数をいくつか持っています。詳細は、以下参照: **mouse variables (p. 78)**。

fit 機能は、"FIT_" で始まるいくつかの変数を使用しますので、そのような名前を使うのは避けるべきでしょう。**set fit errorvariables** を使用すると、各当てはめ変数のエラーは、そのパラメータ名に "_err" を追加した変数に保存されます。詳細は、以下参照: **fit (p. 137)**。

以下も参照: **user-defined variables (p. 63)**, **reset errors (p. 192)**, **mouse variables (p. 78)**, **fit (p. 137)**。

ユーザ定義の変数と関数 (User-defined)

新たなユーザ定義変数と 1 個から 12 個までの引数を持つユーザ定義関数を、任意の場所で定義したり使ったりすることができます。それは **plot** コマンド上でも可能です。

ユーザ定義関数書式:

```
<func-name>( <dummy1> {,<dummy2>} ... {,<dummy12>} ) = <expression>
```

ここで <expression> は、仮変数 <dummy1> から <dummy12> で表される数式です。この形式の関数定義は、1 行での使用に制限されています。より複雑で複数行からなる関数は、関数ブロックの仕組みを使えば定義できます (現バージョンでの新機能)。以下参照: **function blocks (p. 148)**。

ユーザ定義変数書式:

```
<variable-name> = <constant-expression>
```

例:

```
w = 2
q = floor(tan(pi/2 - 0.1))
```

```
f(x) = sin(w*x)
sinc(x) = sin(pi*x)/(pi*x)
delta(t) = (t == 0)
ramp(t) = (t > 0) ? t : 0
min(a,b) = (a < b) ? a : b
comb(n,k) = n!/(k!*(n-k)!)
len3d(x,y,z) = sqrt(x*x+y*y+z*z)
plot f(x) = sin(x*a), a = 0.2, f(x), a = 0.4, f(x)

file = "mydata.inp"
file(n) = sprintf("run_%d.dat",n)
```

最後の 2 行の例は、ユーザ定義文字列変数と、ユーザ定義文字列関数を意味しています。

変数 **pi** (3.14159...), **Inf** (無限大) と **NaN** (IEEE 非数 ("Not a Number")) は、プログラムの起動時に定義されます。これらが必要なければ、他のものに再定義することも可能ですし、**reset session** か以下のようにして元の値に復帰することもできます:

```
pi  = GPVAL_pi;  Inf = GPVAL_Inf;  NaN = GPVAL_NaN
```

他にもいくつかの変数が、例えば対話型入出力形式でのマウス操作や当てはめ (fit) などの gnuplot の動作状態に応じて定義されます。詳細は以下参照: **gnuplot-defined variables (p. 63)**。

ある変数 V が既に定義されているかどうかは、式 exists("V") でチェックできます。例:

```
a = 10
if (exists("a")) print "a is defined"
if (!exists("b")) print "b is not defined"
```

変数名や関数名の命名規則は、大抵のプログラミング言語と同じで、先頭はアルファベットで、その後の文字はアルファベット、数字、"_" が使えます。

各関数の定義式は、'GPFUN_' という接頭辞を持つ特別な文字列値変数として利用できます。

例:

```
set label GPFUN_sinc at graph .05,.95
```

以下参照: **show functions (p. 318)**, **functions (p. 180)**, **gnuplot-defined variables (p. 63)**, **macros (p. 83)**, **value (p. 58)**。

配列 (arrays)

配列は、ユーザ変数の添字付きリストとして実装されています。一つの配列の要素は、一つの変数型には限定されていません。配列は、参照する前に明示的に作られていなければいけません。配列を作成した後で、そのサイズを変更することはできません。配列の要素は、宣言時に提供されてない限り、最初は未定義 (undefined) です。ほとんどの場面で、名前付きユーザ変数の代わりに配列要素を利用できます。

配列 A の要素数は、数式 |A| で取得できます。

例:

```
array A[6]
A[1] = 1
A[2] = 2.0
A[3] = {3.0, 3.0}
A[4] = "four"
A[6] = A[2]**3
array B[6] = [ 1, 2.0, A[3], "four", , B[2]**3 ]
array C = split("A B C D E F")

do for [i=1:6] { print A[i], B[i] }
    1 1
    2.0 2.0
    {3.0, 3.0} {3.0, 3.0}
    four four
    <undefined> <undefined>
    8.0 8.0
```

注意: 配列と変数は、同じ名前空間を共有します。例えば、あらかじめ FOO という名前の配列がある場合、FOO という名前の変数に文字列を割り当てると配列を破壊します。

配列の名前を **plot**, **splot**, **fit**, **stats** コマンドに与えることもできます。その場合、配列の添字がファイルの 1 列目の値で (1 から size まで)、その値 A[i] の実数部分 real(A[i]) がファイルの 2 列目、虚数部分 imag(A[i]) がファイルの 3 列目であるようなデータファイルを与えたことと同等になります。

例:

```
array A[200]
do for [i=1:200] { A[i] = sin(i * pi/100.) }
plot A title "sin(x) in centiradians"
```

ただし、plot 時に複素数値配列の虚数部分を描画したい場合、その値は imag(A[$1]) かまたは $3 として参照できます。よって以下の 2 つのコマンドは同等です。

```
plot A using (real(A[$1])) : (imag(A[$1]))
plot A using 2:3
```

配列関数 (array functions)

gnuplot バージョン 6 より、配列を関数に渡すことも、返り値にすることもできるようになりました。例えば、2 つのサイズの同じ数値配列のドット積 (内積) は以下のように定義できます。

```
dot(A,B) = (|A| != |B|) ? NaN : sum [i=1:|A|] A[i] * B[i]
```

配列をやりとりする組み込み関数には、配列の分割操作 array[min:max] や添字取得関数 index(Array,value) があります。

```
T = split("A B C D E F")
U = T[3:4]
```

```
print T
   [ "A", "B", "C", "D", "E", "F" ]
print U
   [ "C", "D" ]
print index( T, "D" )
   4
```

この例の T と U は、それが以前にどのように宣言されていたかどうかに関わらず、この例の時点では配列になることに注意してください。

配列の添字付け (Array indexing)

要素が N 個の配列 (array) の添字は、1 から N までとなります。配列 A の i 番目の要素は、A[i] と参照します。組み込み関数 **index(Array, <value>)** は、A[i] が <value> に等しいような整数 i を返します。ここで、<value> は数値 (整数、実数、または複素数) と評価される任意の数式、あるいは文字列です。配列の要素とは型と値の両方が一致する必要があります。見つからない場合は、0 を返します。

```
array A = [ 4.0, 4, "4" ]
print index( A, 4 )
      2
print index( A, 2.+2. )
      1
print index( A, "D4"[2:2] )
      3
```

フォント

gnuplot それ自身にはどんなフォントも含まれてはおらず、外部フォント処理に頼っているだけで、その細部は悲しいことに出力形式毎に異なります。ここでは、複数の出力形式に適用されるフォント機構について説明します。ここに上げたもの以外の出力形式でのフォントの使用に関しては、その出力形式のドキュメントを参照してください。

一時的に、例えば Adobe Symbol フォントのような特別なフォントに切り替えることでアルファベットではない記号を入れることも可能ですが、現在は、UTF-8 エンコーディングを使用して、他の文字と同様にその記号を扱うのがより良い方法でしょう。その他に、必要な記号の Unicode コードポイントを、拡張文字列モード内でエスケープシーケンスとして指定する手もあります。以下参照: **encoding (p. 215)**, **unicode (p. 44)**, **locale (p. 240)**, **escape sequences (p. 44)**。

Cairo (pdfcairo, pngcairo, epscairo, wxt 出力形式)

出力形式により (例えば cairo 系の出力形式すべて)、fontconfig システムライブラリを使ってフォントにアクセスします。`fontconfig` ユーザマニュアル

を参照してください。これは、gnuplot で一般的な名前やサイズでフォントを要求することを可能にし、必要ならば fontconfig に同等のフォントを代用させることもできるので、通常はこれで十分でしょう。以下は、多分いずれも機能します:

```
set term pdfcairo font "sans,12"
set term pdfcairo font "Times,12"
set term pdfcairo font "Times-New-Roman,12"
```

Gd (png, gif, jpeg, sixel terminals)

png, gif, jpeg, sixelgd 出力形式のフォント処理は、外部ライブラリ libgd が行います。これは、最低でも **tiny**, **small**, **medium**, **large**, **giant** の 5 種類の基本フォントを提供しますが、これは伸縮させたり回転したりはできません。これらの一つを使用する際は、**font** キーワードの代わりに上のキーワードを指定します。例:

```
set term png tiny
```

多くのシステムで、libgd は、fontconfig ツールが提供する一般的なフォント処理も使用できます。以下参照: **fontconfig (p. 66)**。fontconfig のないシステム上では、大抵 Adobe フォント (*.pfa) と TrueType フォント (*.ttf) へのアクセスを提供しています。その場合フォント自身の名前ではなく、フォントファイルの名前を、"<face> {,<size>}" の形式で与えなければいけません。ここで、<face> はフォントファイルのフルパス名か、または環境変数 GDFONTPATH で指示されるディレクトリの一つの中のファイル名の先頭部分、のいずれかです。よって、'set term png font "Face"' は、< あるディレクトリ >/Face.ttf か < あるディレクトリ >/Face.pfa というファイル名のフォントを探そうとします。例えば、GDFONTPATH に **/usr/local/fonts/ttf:/usr/local/fonts/pfa** が含まれている場合は、以下のコマンドの 2 つずつはいずれも同じことになります:

```
set term png font "arial"
set term png font "/usr/local/fonts/ttf/arial.ttf"
set term png font "Helvetica"
set term png font "/usr/local/fonts/pfa/Helvetica.pfa"
```

デフォルトのフォントサイズも同時に指定するには:

```
set term png font "arial,11"
```

"set term" コマンドでフォントを指定しなかった場合、gnuplot は環境変数 GNUPLOT_DEFAULT_GDFONT を参照します。

Postscript (カプセル化 postscript *.eps も)

PostScript フォント処理は、プリンタか表示ソフトが行います。もし、あなたのコンピュータにフォントが一切なくても、gnuplot は正しい PostScript ファイル、またはカプセル化 PostScript (*.eps) ファイルを生成できます。gnuplot は単に出力ファイル中にフォントを名前として入れるだけで、プリンタや表示ソフトがその名前からフォントを見つけるか近似することを仮定しています。

PostScript プリンタや表示ソフトはすべて、標準的な Adobe フォントセット **Times-Roman**, **Helvetica**, **Courier**, **Symbol** は知っているはずです。多分その他にも多くのフォントが使えるようになっていると思い

ますが、それら特定のフォントセットはあなたのシステムやプリンタの設定に依存します。gnuplot は、それは知りませんし気にもしません。gnuplot が作成した *.ps や *.eps 出力は、あなたの要求したフォント名を単に持っているだけです。

よって、

```
set term postscript eps font "Times-Roman,12"
```

は、すべてのプリンタや表示ソフトに適切な出力を作成します。

一方、

```
set term postscript eps font "Garamond-Premier-Pro-Italic"
```

は、正しい PostScript 出力ファイルを作成しますが、それは特別な専用フォントを参照しているので、要求されたその特定のフォントは一部のプリンタや表示ソフトでしか表示できないでしょう。大抵の場合は別なフォントで代用されます。

しかし、指定したフォントを出力ファイル中に埋め込んで、どんなプリンタでもそれを使うようにすることも可能です。これには、あなたのシステムに適切なフォント記述ファイルがあることが必要となります。この方法でフォントを埋め込む場合、特定のライセンスが必要となるフォントファイルもあることに注意してください。より詳細な説明や例については、以下参照: **postscript fontfile (p. 380)**。

ヘルプの用語解説 (Glossary)

gnuplot は 30 年以上かけて開発されているので、コマンドやこの文書で使われている用語の意味は、現在の普通の用法とは違っているかもしれません。この節では、**gnuplot** 内ではそれらの用語のいくつかをどのように使っているかを説明します。

用語 "出力形式 (terminal)" は、出力モードのことを意味し、あなたがキーボード入力するもの (ターミナル) を指しているのではありません。例えば、コマンド **set terminal pdf** は、その後の描画コマンドが PDF 出力を生成することを意味します。通常は、その PDF 出力を書き出す場所を指定するコマンド **set output "filename"** を一緒に使う必要があるでしょう。

"ページ (page)"、"表示画面 (screen)"、"キャンバス (canvas)" は、**gnuplot** がアクセス可能な領域全体を指します。デスクトップではそれはウィンドウ全体を指し、プロッタでは、一枚の紙全体を指します。

データファイルに関する議論では、用語 "行 (record)" は、ファイルの一行の文字列、すなわち、改行文字や行末文字同士に挟まれた文字列を指します。"点 (point)" は、一行から取り出した一つのデータです。データの "ブロック (block)" は、空行で区切られた連続した複数の行からなる集合です。データファイルの議論の中で "line" が参照される場合は、これはブロックの部分集合を指します。"データのブロック (data block)" という言葉は、インラインデータの名前付きブロックを指すのにも使われています。以下参照: **datablocks (p. 68)**。

インラインデータとデータブロック (inline data and datablocks)

gnuplot のコマンド入力の中にデータを埋め込む仕組みは 2 種類用意されています。まず、特殊ファイル名 '-' が plot コマンド中に与えると、その plot コマンド以下に続く行がインラインデータと解釈されます。以下参照: **special-filenames (p. 174)**。この方法で提供されるデータは、その plot コマンドで一度しか使用できません。

もう一つは、ヒアドキュメントとして名前付きのデータブロックを定義する方法です。その名前付きのデータは残るので、複数の plot コマンドで参照できます。例:

```
$Mydata << EOD
11 22 33 first line of data
44 55 66 second line of data
# データファイル同様コメントも機能する
77 88 99
EOD
stats $Mydata using 1:3
plot $Mydata using 1:3 with points, $Mydata using 1:2 with impulses
```

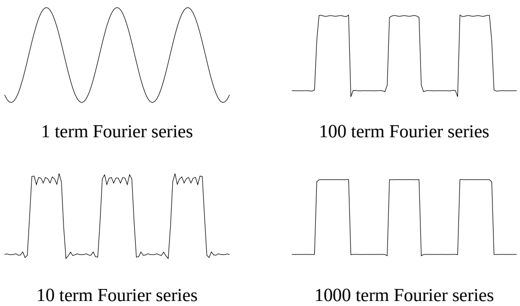
データブロック名は、他の変数と区別するために、最初の文字を $ にする必要があります。データの終わりの区切り (上の例では EOD) は、任意のアルファベット、数字からなる文字列で構いません。

データを名前付きブロックに保存するかわりに、実行可能コマンドを保存する同様の仕組みについては、以下参照: **function blocks (p. 148)**。

コマンド **undefine** を使えば、保存した名前付きデータブロックを削除できます。**undefine $*** は、すべての名前付きデータブロック、関数ブロックを一度に削除します。

繰り返し (iteration)

gnuplot は、繰り返し (iteration) コマンドやブロック構造を扱うための if/else/while/do をサポートしています。以下参照: **if (p. 150)**, **while (p. 334)**, **do (p. 136)**。コマンド **plot**, **set** で単純な繰り返しも可能です。以下参照: **plot for (p. 183)**。複数のコマンドを包含する一般的な繰り返しは、下で紹介するブロック構造を利用することで行えます。関連する新しい機能である数式型の以下も参照: **summation (p. 62)**。以下は、これらの新しい構文機能をいくつか利用した例です:



1 term Fourier series
100 term Fourier series
10 term Fourier series
1000 term Fourier series

```
set multiplot layout 2,2
fourier(k, x) = sin(3./2*k)/k * 2./3*cos(k*x)
do for [power = 0:3] {
    TERMS = 10**power
    set title sprintf("%g term Fourier series",TERMS)
    plot 0.5 + sum [k=1:TERMS] fourier(k,x) notitle
}
unset multiplot
```

繰り返しは、以下のような書式による繰り返し指定で制御します。

```
for [<var> in "string of N elements"]
```

または

```
for [<var> = <start> : <end> { : <increment> }]
```

最初の書式では <var> は文字列変数で、その後ろに指定した文字列に含まれる 1 番目から N 番目までの単語文字列を順に値として取ります。2 番目の書式では、<start>, <end>, <increment> は整数、または整数値を取る数式です。

繰り返し変数の有効範囲 (scope) は、その繰り返しの中だけです。以下参照: **scope (p. 80)**。繰り返し変数の値を、その実行ブロック内で永続的に変更することはできません。繰り返し変数が繰り返しの前に値を持っていたとしたら、その値は繰り返しの終了時に保持、または復帰されます。例えば、以下のコマンドは、1 2 3 4 5 6 7 8 9 10 A を出力します。

```
i = "A"
do for [i=1:10] { print i; i=10; }
print i
```

線種、色、スタイル (linetypes)

とても古い版の gnuplot では、各出力形式は "線種 (linetype)" をある程度用意していて、それらは色、太さ、点線/破線のパターン、または色と点線/破線の組合せで違いを表現していました。それらの色、点線/破線のパターンは、出力形式を越えて同じものになるという保証は何もありませんでしたが、多くは同じ色の列、赤/緑/青/紫/水色/黄色を使用していました。この古い挙動は、今は **set colorsequence classic** で選択できますが、現在の gnuplot のデフォルトは、出力形式に無関係に共通の 8 色列を使用します。

線種 (linetype) の属性の並びは、対話的か初期化ファイルのいずれかによってさらにカスタマイズ可能です。以下参照: **set linetype (p. 238)**。配布パッケージに初期化ファイルのサンプルがいくつか用意されています。

特定の出力形式に対する線種の属性の現在の状態は、その出力形式を設定したあとで **test** コマンドを実行することで確認できます。

一つの描画コマンド内での関数やデータファイルの連続する並びには、現在のデフォルトの線種列から線種が順番に割り当てられます。個々の関数、データファイル、またはその他の描画要素に対する線種は、その描画コマンド上で明示的に線の属性を指定することで上書きできます。

例:

```
plot "foo", "bar"                 # 線種 1, 2 で 2 ファイルを描画
plot sin(x) linetype 4            # 線種色 4 を使用
```

一般に、色の指定は、色の名前か、RGB (赤、緑、青) 成分か、HSV (色相、彩度、明度) 成分か、現在の pm3d パレットに沿った座標で行います。キーワード **linecolor** は、**lc** と省略できます。

例:

```
plot sin(x) lc rgb "violet"       # gnuplot 内の色名の一つを使用
plot sin(x) lc rgb "#FF00FF"      # 明示的な 16 進 RGB 3 つ組
plot sin(x) lc palette cb -45     # 現在のパレットの cbrange の
                                  # -45 に対応する色
plot sin(x) lc palette frac 0.3   # パレットに対応する小数値
```

以下参照: **colorspec** (p. **71**), **show colornames** (p. **318**), **hsv** (p. **51**), **set palette** (p. **258**), **cbrange** (p. **317**)。以下も参照: **monochrome** (p. **243**)。

線種 (linetype) には、点線/破線のパターンも結びつけられていますが、すべての出力形式でそれが使えるわけではありません。線色とは独立に点線/破線パターンを指定できます。以下参照: **dashtype** (p. **73**)。

色指定 (colorspec)

多くのコマンドで、明示的な色の指定をともなった線種を指定することができます。

書式:

```
... {linecolor | lc} {"colorname" | <colorspec> | <n>}
... {textcolor | tc} {<colorspec> | {linetype | lt} <n>}
... {fillcolor | fc} {<colorspec> | linetype <n> | linestyle <n>}
```

<colorspec> は以下の形式のいずれかです:

```
rgbcolor "colorname"    # 例: "blue"
rgbcolor "0xRRGGBB"     # 16 進数値の定数文字列
rgbcolor "0xAARRGGBB"   # 16 進数値の定数文字列
rgbcolor "#RRGGBB"      # x11 形式の 16 進数文字列
rgbcolor "#AARRGGBB"    # x11 形式の 16 進数文字列
rgbcolor <integer val>  # AARRGGBB を表す整数値
rgbcolor variable       # 入力ファイルから整数値を読み込む
palette frac <val>      # <val> は 0 から 1 の値
palette cb <value>      # <val> は cbrange の範囲の値
palette z
palette <colormap>      # 名前付きカラーマップを使用
variable                # 入力ファイルから色番号を読み込む
background または bgnd  # 背景色
black
```

<n> は、その線種 (linetype) 番号が使う色を意味します。以下参照: **test** (p. **330**)。

"colorname" は gnuplot が内部に持っている色の名前のうちの一つを指定します。有効な名前の一覧に関しては、以下参照: **show colornames** (p. **318**)。

16 進定数は、引用符付きで "#RRGGBB" や "0xRRGGBB" の形で与えることができます。RRGGBB は、色の赤、緑、青の成分を意味し、それぞれ 00 から FF までの範囲内でなければいけません。例えば、マゼンタ (紫) は、最も明るい赤 + 最も明るい青、なので "0xFF00FF" と表され、これは 16 進数で $(255 << 16) + (0 << 8) + (255)$ を意味しています。

"#AARRGGBB" は、RGB 色の上位ビットにアルファ値 (透過性) がついていることを意味します。アルファ値 0 は完全に不透明色であることを意味し、よって "#00RRGGBB" は "#RRGGBB" と同じになります。アルファ値の 255 (FF) は完全に透明であることを意味します。

これらの任意の形式から、色の 32 ビット整数表現への変換を行うのに利用できる関数については、以下参照: **expressions functions rgbcolor** (p. **51**)。

カラーパレットとは、色の線型なグラデーションで、単一の数値を特定の色に滑らかに対応づけます。常にそのような 2 つの対応付けが効力を持ちます。**palette frac** は 0 から 1 までの小数値を、カラーパレットの全範囲に対応付けるもので、**palette cb** は、色軸の範囲を同じカラーパレットへ割り当てるものです。以下参照: **set cbrange (p. 317)**, **set colorbox (p. 206)**。これらの対応付けのどちらかを使って、現在のパレットから定数色を選び出すことができます。

"palette z" は、各描画線分や描画要素の z の値を、パレットへ対応づけられている cbrange の範囲に対応づけます。これにより、3 次元の曲線や曲面に沿って色を滑らかに変化させることができます。これは、2 次元描画で、パレット値を追加の列データから読み込ませて色付けするのにも使えます (すべての 2 次元描画スタイルがこの追加列を認識するわけではありません)。特殊な色指定が 2 つあります。背景色の **background** (省略形 **bgnd**) と **black** です。

Background color

多くの出力形式でグラフの背景色を明示的に設定できます。特別な線種 (linetype) **background** (省略名 **bgnd**) はその色で描画しますが、それは色名としても認識されます。例:

```
# 以下はキャンバスの一部分を背景色で上書きすることで消去します。
set term wxt background rgb "gray75"
set object 1 rectangle from x0,y0 to x1,y1 fillstyle solid fillcolor bgnd
# y=0 に「見えない」線を描き、その下のものをすべて消します。
plot 0 lt background
```

Linecolor variable

lc variable は、入力データの一つの列から読んだ値を線種 (linetype) の番号として使い、その線種に属する色を使うようプログラムに指示します。よってこれは、**using** 指定子へ対応する列の指定の追加を必要とします。**ls variable** は、入力データ列から読み込んだ値が線種でなく linestyle 番号として解釈されることを除いて同じことをします。文字の色も同様に、**tc variable** で指定できます。

例:

```
# データの 3 列目を、個々の点に色を割り当てるのに使用
plot 'data' using 1:2:3 with points lc variable

# 一つのデータファイルには複数のデータ集合を入れることが可能で、
# それらは 2 行の空行で分離されています。個々のデータ集合には
# index 値が割り当てられていて (以下参照: `index`)、using 指定の
# column(-2) で取得できます。以下参照: `pseudocolumns`。以下の例
# は -2 の column 値を使って、個々のデータ集合を異なる線色で描画
# します。
plot 'data' using 1:2:(column(-2)) with lines lc variable
```

Palette

書式

```
... {lc|fc|tc} palette {z}
... {lc|fc|tc} palette frac <fraction>
... {lc|fc|tc} palette cb <fixed z-value>
... fc palette <colormap>
```

パレットは、0 から 1 までの灰色値で、色の範囲を定義したものですが、**palette frac <fraction>** は、灰色値 <fraction> でその色を選択します。

palette cb <z> は、灰色値が (z - cbmin) / (cbmax - cbmin) である色を選択します。

palette と **palette z** はどちらも描画要素の z 座標を現在のパレットの色に写像します。z が cbrange の範囲外ならば、それはデフォルトでは palette fraction 0 か palette fraction 1 になります。オプション **set pm3d noclipcb** がセットされている場合、z 座標が範囲外にある pm3d グラフの 四辺形は何も描かれなくなります。

fillcolor palette <colormap> は、描画要素の z 座標を、現在のパレットを使用するのではなく、事前に保存した名前付きカラーマップに写像します。以下参照: **set colormap (p. 201)**。

カラーマップがそれ用の個別の範囲を持っている場合、cbrange を標準パレットに写像するのに使うのと同様に、z の値を写像するのにその範囲を使用します。

Rgbcolor variable

グラフの各データ点、各線分、または各ラベルにそれぞれ異なる色を割り当てることができます。**lc rgbcolor variable** は、データファイルの各行から RGB 色の情報を読み込むようプログラムに指示します。よってこれは、**using** 指定子による対応する列の指定の追加を必要とし、その列は 24-bit 形式の RGB の 3 つ組であるとみなされます。その値をデータファイルから直接与える場合は、これは最も簡単な形式の 16 進値で与えます (以下参照: **rgbcolor (p. 51)**)。一方で、以下の例のように 24-bit RGB 色として評価されるような数式を **using** 指定子に入れることもできます。文字の色も同様に、**tc rgbcolor variable** で指定できます。

例:

```
# 3 次元描画で、各 x,y,z 座標に対応した赤、緑、青の成分を持つ色
# のついた点を配置
rgb(r,g,b) = 65536 * int(r) + 256 * int(g) + int(b)
splot "data" using 1:2:3:(rgb($1,$2,$3)) with points lc rgb variable
```

点線/破線種 (dashtype)

linecolor や **linewidth** と同様に、点線/破線パターン (**dashtype**) が各曲線毎の属性として独立しました。従来のような、使用中の出力形式の点線を書くための特別なモードとして指定する必要はありません。すなわち、**set term <termname> {solid|dashed}** のような古いコマンドは現在は無視されます。

すべての線は、ほかに指定しなければ、**dashtype solid** という属性を持ちますが、このデフォルト値をコマンド **set linetype** で特定の線種に変更しその後のコマンドで使えるようにできます。または、**plot** や他のコマンドの一部分として使用したい点線/破線の型を指定できます。

書式:

```
dashtype N           # 定義済み点線/破線型を番号で呼び出し
dashtype "pattern"   # 点 (.) 横線 (-) 下線 (_) と空白の組み合わ
                     # せ文字列による指定
dashtype (s1,e1,s2,e2,s3,e3,s4,e4)  # 1~4 個の <実線長>,<空白長>
                     # の組による点線/破線パターン指定
```

例:

```
# 2 つの関数が線種 1 を使うが dashtype で区別
plot f1(x) with lines lt 1 dt solid, f2(x) with lines lt 1 dt 3
```

いくつかの出力形式は、それが提供する定義済み点線/破線パターンに、ユーザ定義パターンを追加することをサポートしています。

例:

```
plot f(x) dt 3              # 出力形式の持つパターン 3 を使用
plot f(x) dt ".. "          # 一時的なパターンを作成
plot f(x) dt (2,5,2,15)     # 同じパターンを数値で表現
set dashtype 11 (2,4,4,7) # 新パターンを番号で呼び出せるよう定義
plot f(x) dt 11             # 新パターンを使って描画
```

点線/破線パターンを文字列で指定した場合、gnuplot はそれを < 実線長 >, < 空白長 > の組の列に変換します。ドット "." は (2,5) に、ダッシュ "-" は (10,10) に、下線 "_" は (20,10) に、また空白 " " は直前の < 空白長 > の値に 10 を追加します。その場合、コマンド **show dashtype** は、元の文字列と変換後の数値の列の両方を表示します。

Linestyles と linetypes

linestyle は、属性 linecolor, linewidth, dashtype, pointtype の一時的な組み合わせで、これはコマンド **set style line** で定義します。一度 linestyle を定義すると、1 回の plot コマンド上でそれを使って、1 つ、またはより多くの描画要素の見た目を制御できます。言い換えれば、これは丁度 linetype から永続性を取り除いたもの、と言うことができるでしょう。**linetypes** は永続的 (明示的にそれらを再定義するまでは保持される) ですが、**linestyles** は、次のグラフィックの状態がリセットされるまでの間しか保持されません。

例:

```
# 新しいラインスタイルを、出力形式に依存しない色 cyan、線幅が 3、
# 点種 6 (丸の中に点) と定義
set style line 5 lt rgb "cyan" lw 3 pt 6
plot sin(x) with linespoints ls 5          # 定義スタイル 5 で
```

特別な線種 (special linetypes)

2 つの特別な数値の線種 (linetype) には知る価値があります。これらは、**set linetype** では変更できません。

線種 0 は、出力形式に依存する線種で、gnuplot が軸 (以下参照: **zeroaxis (p. 315)**) と格子線 (以下参照: **grid (p. 223)**) を描くときのデフォルトとして使用します。それは、出力形式がサポートしていれば、常に点線になります。個別の破線属性をサポートしていない出力形式では、これが点線を得るための唯一の方法でしょう。

線種 -1 は、基本の線種です。これは、幅 1.0 の、前景色 (通常黒) の実線に対応します。gnuplot は、多くの描画要素 (描画境界、ラベル、文字列、矢) の描くときのデフォルトとして使用します。

gnuplot は、いくつか数値でない線種も認識します。

lt black は、黒い実線を意味します。

lt background や **lt bgnd** は、現在の出力形式の背景色の実線を意味します。以下参照: **background (p. 72)**。

lt nodraw は、その曲線全体の描画をスキップします。これは、描画スタイル **linespoints** と組み合わせて使うと便利です。すなわち、この描画スタイルのみに有効な点部分の属性を残しつつ、線部分を抑制することが可能になります。例えば、

```
plot f(x) with linespoints lt nodraw pointinterval -3
```

は、3 点置きに描画し、その下に背景色の小さな円を置くことで孤立させせます。以下参照: **linespoints (p. 116)**。**lt nodraw** は、自動的に描かれる線分の特別な集合を抑制するのにも使えます。例えば、等高線描画の中のあるレベルの等高線を、その線種に **nodraw** を設定することで、その描画を抑制できます。

レイヤー (layers)

gnuplot のグラフは、色々な要素を固定された順番で描き上げていくことで構成されています。この順番は、キーワード **behind**, **back**, **front** を使って要素に特定の階層を割り当てることで変更できます。例えば、グラフ領域の背景色を変更するには、色のついた長方形を属性 **behind** で定義すればいいわけです。

```
set object 1 rectangle from graph 0,0 to graph 1,1 fc rgb "gray" behind
```

描画の順番は以下の通りです:

```
behind
back
グラフ自体
グラフの表題 (‘key‘)
front
```

各階層内では、要素の描画は以下の順番です:

```
格子線、軸、境界要素 (grid, axis, border elements)
番号順のピクスマップ画像 (pixmaps)
番号順のオブジェクト (rectangle, circle, ellipse, polygon)
番号順のラベル (label)
番号順の矢印 (arrow)
```

1 ページに複数のグラフがある場合 (multiplot モード)、この順序は、複数グラフを全体として適用するのではなく、各描画要素に別々に適用します。

これに対する例外は、TeX 系の出力形式 (例えば pslatex や cairolatex 等) で、これらは一つの出力にすべての文字列要素を積み重ね、グラフ要素は別な出力に積み重ねます。一般にこの場合、各文字列要素は全部がグラフの前に出てしまうか、逆に全部がグラフの裏に置かれてしまうかのどちらかになります。

マウス入力 (mouse input)

多くの出力形式で、現在の描画にマウスを使って作用をすることが可能になっています。そのうちいくつかはホットキーの定義もサポートしていて、マウスカーソルが有効な描画ウィンドウにあるときに、あるキーを押すことであらかじめ定義した関数を実行させることができます。マウス入力を **batch** コマンドスクリプトと組み合わせることも可能で、例えば **pause mouse** として、その後にマウスクリックによってパラメータとして返って来るマウス変数をその後のスクリプト動作に反映させることができます。以下参照: **bind (p. 76)**, **mouse variables (p. 78)**。また以下も参照: **set mouse (p. 244)**。

Bind

書式:

```
bind {allwindows} [<key-sequence>] ["<gnuplot commands>"]
bind <key-sequence> ""
reset bind
```

bind は、ホットキーの定義、再定義に使用します。ホットキーとは、入力カーソルがドライバのウィンドウ内にあるときに、あるキー、または複数のキーを押すことで、gnuplot のコマンド列を実行させる機能のことを言います。**bind** は、gnuplot が **mouse** をサポートするようにコンパイルされていてかつマウスが有効な出力形式上で使われてる場合にのみ有効であることに注意してください。ユーザ指定のキー割当 (binding) は、組み込み (builtin) キー割当を置き換えますが、<space> と 'q' は通常は再定義はできません。その唯一の例外については、以下参照: **bind space (p. 77)**。

マウスボタン割り当ては、2 次元描画でのみ有効です。

ホットキーの一覧を得るには **show bind**, または **bind** とタイプするか、グラフウィンドウ上でホットキー 'h' を入力してください。

キー定義は、**reset bind** でデフォルトの状態に復帰できます。

修飾キーを含む複数のキーの定義は引用符で囲む必要があることに注意してください。

標準ではホットキーは現在の描画ウィンドウ上に入力カーソルがある場合のみ認識されます。**bind allwindows <key> ...** (**bind all <key> ...** と省略可) は、<key> の割当を、それが現在の有効なものか否かに関わらず、すべての gnuplot の描画ウィンドウ上で可能にします。この場合、gnuplot 変数 MOUSE_KEY_WINDOW にそれが行なわれたウィンドウの ID が保存されるのでそれをキーに割り当てたコマンドで使用することができます。

例:

- キー割当の設定:

```
bind a "replot"
bind "ctrl-a" "plot x*x"
bind "ctrl-alt-a" 'print "great"'
bind Home "set view 60,30; replot"
bind all Home 'print "This is window ",MOUSE_KEY_WINDOW'
```

- キー割当を表示:

```
bind "ctrl-a"           # ctrl-a に対するキー割当を表示
bind                    # 全てのキー定義を表示
show bind               # 全てのキー定義を表示
```

- キー割当を削除:

```
bind "ctrl-alt-a" ""    # ctrl-alt-a のキー割当を削除
                          (組み込みキー定義は削除されません)
reset bind              # デフォルト (組み込み) のキー定義を導入
```

- トグルスイッチ形式にキー割当:

```
v=0
bind "ctrl-r" "v=v+1;if(v%2)set term x11 noraise; else set term x11 raise"
```

修飾キー (ctrl / alt) は大文字小文字の区別はありませんが、キーはそうではありません:

```
ctrl-alt-a == CtRl-alT-a
ctrl-alt-a != ctrl-alt-A
```

修飾キー (alt == meta) の一覧:

```
ctrl, alt, shift (ボタン 1, ボタン 2, ボタン 3 でのみ有効)
```

サポートされている特殊キーの一覧:

```
"BackSpace", "Tab", "Linefeed", "Clear", "Return", "Pause", "Scroll_Lock",
"Sys_Req", "Escape", "Delete", "Home", "Left", "Up", "Right", "Down",
"PageUp", "PageDown", "End", "Begin",

"KP_Space", "KP_Tab", "KP_Enter", "KP_F1", "KP_F2", "KP_F3", "KP_F4",
"KP_Home", "KP_Left", "KP_Up", "KP_Right", "KP_Down", "KP_PageUp",
"KP_PageDown", "KP_End", "KP_Begin", "KP_Insert", "KP_Delete", "KP_Equal",
"KP_Multiply", "KP_Add", "KP_Separator", "KP_Subtract", "KP_Decimal",
"KP_Divide",

"KP_1" - "KP_9", "F1" - "F12"
```

以下は、実際のキーではなく、ウィンドウに関するイベントです:

```
"Button1" "Button2" "Button3" "Close"
```

以下も参照: **mouse (p. 244)**。

Bind space

gnuplot が、configure 時にオプション –enable-rase-console をつけてインストールされた場合は、描画ウィンドウ内で <space> をタイプすると gnuplot のコマンドウィンドウが前面に出ます。多分。実際には、これは強くシステムに依存します。このホットキーは、'gnuplot -ctrlq' のようにして gnuplot を起動するか、または X リソースの 'gnuplot*ctrlq' を設定することで ctrl-space に変更できます。

マウス用の変数 (Mouse variables)

mousing (マウス機能) が有効な場合、現在のウィンドウ上でのマウスクリックによって gnuplot のコマンドライン上で使うことができる色々なユーザ変数が設定されます。クリック時のマウスの座標は変数 MOUSE_X, MOUSE_Y, MOUSE_X2, MOUSE_Y2 に代入されます。クリックされたボタンや、そのときのメタキーの状態は MOUSE_BUTTON, MOUSE_SHIFT, MOUSE_ALT, MOUSE_CTRL に代入されます。これらの変数は任意の描画の開始時には未定義で、有効な描画ウィンドウ中でのマウスクリックイベントによって初めて定義されます。有効な描画ウィンドウ中でマウスが既にクリックされたかどうかをスクリプトから調べるには、これらの変数のうちのどれか一つが定義されているかどうかをチェックすれば十分です。

```
plot 'something'
pause mouse
if (exists("MOUSE_BUTTON")) call 'something_else'; \
else print "No mouse click."
```

描画ウィンドウ上での一連のキー入力を追跡することも、マウスコードを使うことで可能となります。

```
plot 'something'
pause mouse keypress
print "Keystroke ", MOUSE_KEY, " at ", MOUSE_X, " ", MOUSE_Y
```

pause mouse keypress が、キー入力で終了した場合は MOUSE_KEY には押されたキーの ASCII コードが保存されます。MOUSE_CHAR にはその文字自身が文字列値として保存されます。pause コマンドが (例えば ctrl-C や描画ウィンドウが外部から閉じられるなどして) 異常終了した場合は MOUSE_KEY は -1 になります。

マウスによる拡大の後の新しい描画範囲は、GPVAL_X_MIN, GPVAL_X_MAX, GPVAL_Y_MIN, GPVAL_Y_MAX で参照できることに注意してください。以下参照: **gnuplot-defined variables (p. 63)**。

残留 (Persist)

gnuplot の多くの出力形式 (aqua, pm, qt, x11, windows, wxt, ...) が、スクリーン上にグラフをその中に描いた表示用のウィンドウを別に開きます。オプション **persist** は、主たるプログラムが終了したときにも、それらのウィンドウを残すよう gnuplot に指示します。これは、非対話型出力形式出力では何もしません。例えば、以下のコマンドを実行すると

```
gnuplot -persist -e 'plot sinh(x)'
```

gnuplot は、表示ウィンドウを開き、その中にグラフを描き、そして終了し、表示ウィンドウはグラフをその中に持ったままスクリーンに残ります。新しい出力形式を設定するときに **persist** や **nopersist** を指定することもできます。

```
set term qt persist size 700,500
```

出力形式によっては、その残ったウィンドウ上でも多少のマウス操作が可能な場合もあります。しかし、グラフの再描画を要求するズーム (とその逆) のような操作は、既にプログラムが終了しているので無理です。描画ウィンドウを開いたまま残し、その後のマウス機能も完全に可能にするには、例えば gnuplot を対話型ではなく、スクリプトファイルから実行させる方法があります。以下参照: **pause mouse close (p. 154)**。

描画 (Plotting)

gnuplot にはグラフを描画する 4 つのコマンド **plot**, **splot**, **replot**, **refresh** があります。他のコマンドは、最終的にグラフ上に生成される描画要素や、レイアウト、スタイルの制御を行います。**plot** は 2 次元グラフを生成します。**splot** は 3 次元グラフ (もちろん実際にはその 2 次元面への射影) を生成します。**replot** は、直前の **plot** や **splot** コマンドを再実行します。**refresh** は、**replot** と似ていますが、入力データをファイルや入力ストリームから再読み込みする代わりに、前に保存したデータを再使用します。

これら 4 つのコマンドのうちの一つを実行した場合は、現在定義されている軸、ラベル、タイトル、および元の plot コマンドで指定されたさまざまな関数やデータのすべてを含む出力のスクリーンを再描画するか、新しい出力ページを生成します。もし、一つのページに複数のグラフを隣り合うように並べて出力したい場合、例えば複数の図のパネルを作成したり、大きなグラフの中に小さなグラフを挿入したりしたい場合は、コマンド **set multiplot** を使用し、各描画コマンドで新しいページが作られるのを抑制してください。

描画に関する一般的な情報の大半は、**plot** に関する項で見つかります。3 次元描画に固有の情報は **splot** の項にあります。

plot は xy 直交座標系と極座標系が使えます。以下参照: **set polar (p. 274)**。**splot** は xyz 直交座標が使えますが、3 次元極座標、円柱座標データも入力できます。以下参照: **set mapping (p. 241)**。**plot** では、4 つの境界 x (下), x2 (上), y (左), y2 (右) をそれぞれ独立な軸として扱うこともできます。オプション **axes** で、与えられた関数やデータ集合をどの軸のペアで表示させるかを選べます。また、各軸の縮尺や見出しづけを完全に制御するために十分な補佐となる **set** コマンド群が存在します。いくつかのコマンドは、**set xlabel** のように軸の名前をその中に持っていますし、それ以外のものは **set logscale xy** のように、1 つ、または複数の軸の名前をオプションとしてとります。z 軸を制御するオプションやコマンドは 2 次元グラフには効力を持ちません。

splot は、点や線に加えて曲面や等高線を書くことができます。3 次元の関数の格子定義に関する情報については、以下参照: **set isosamples (p. 227)**。3 次元データのファイルに必要な形態については、以下参照: **splot datafile (p. 320)**。等高線に関する情報については、以下参照: **set contour (p. 207)**, **set cntrlabel (p. 203)**, **set cntrparam (p. 204)**。

splot での縮尺や見出し付けの制御は、z 軸にも有効であること、および x2 軸、y2 軸のラベル付けが **set view map** を使って作られる疑似的な 2 次元描画にのみ可能であることを除けば **plot** と全く同じです。

プラグイン (Plugins)

グラフや数式に利用できる関数群は、共有ライブラリから実行可能な関数を取り込むプラグインの仕組みにより拡張できます。例えば、gnuplot のバージョン 5.4 では、上方不完全ガンマ関数 Q(a,x) は組み込み関数としては実装していませんでした。

$Q(a,x) = \frac{1}{\Gamma(x)} \int_x^\infty t^{a-1} e^{-t} dt$. 　これは、gnuplot 内で直接以下のようにして近似的に定義できます。

```
Q(a,x) = 1. - igamma(a,x)
```

しかし、これは、1 の近くの igamma(a,x) の値の精度に実質的な限界があります。より正確な値を返す実装が欲しい場合、プラグインを通して用意するのがいいでしょう (下を参照)。一度取り込めば、関数は、gnuplot 内の他の組み込み関数、ユーザ定義関数と全く同様に利用できます。以下参照: **import (p. 151)**。

gnuplot の配布物のディレクトリ demo/plugin には、プラグインライブラリを生成するための説明とソースコードがあります。簡単なサンプルファイル **demo_plugin.c** の関数を、あなたの興味ある関数の実装に置き換えて修正してください。これには、外部の数学ライブラリの関数の呼び出しも含まれています。

ディレクトリ demo/plugin には、Q(a,x) を実装するプラグインのソースもあります。上で注意したように、このプラグインは、gnuplot バージョン 6 に含まれる **uigamma** と同じ関数を、以前のバージョンに実装することを可能にします。

```
import Q(a,x) from "uigamma_plugin"
uigamma(a,x) = ((x<1 || x<a) ? 1.0-igamma(a,x) : Q(a,x))
```

Scope of variables

gnuplot の変数は、以下に示す特別な場合を除いては大域的 (global) です。有効な変数に対しては、その名前で区別された、永続的な一覧表があります。変数の割り当てとは、その一覧表に項目を一つ作るか、置き換えることを意味します。その一覧表から変数を削除する唯一の方法は、コマンド **undefine** を使用することです。

例外 1: 繰り返し指定の中で使用される変数の有効範囲 (スコープ) は、その繰り返しの中だけです。繰り返し変数の値を、その実行ブロック内で永続的に変更することはできません。繰り返し変数が繰り返しの前に値を持っていたとしたら、その値は繰り返しの終了時に保持、または復帰されます。例えば、以下のコマンドは、1 2 3 4 5 6 7 8 9 10 A を出力します。

```
i = "A"
do for [i=1:10] { print i; i=10; }
print i
```

例外 2: 関数定義で使用されるパラメータの名前は、その関数を呼び出すときに与える実際の値の置き場所に過ぎません。例えば、以下の例では、x と y の現在、あるいは未来の値はここで示される定義には関係しないが、A はこの関数が後で評価されるときには大域変数として存在しなければいけません:

```
func(x,y) = A + (x+y)/2.
```

例外 3: コマンド **local** で宣言された変数。**local** 指定 (バージョン 6 での新機能) は、変数、または配列のオプション的な宣言を可能にし、それによりそれが見つかるコードブロック内部にその有効範囲を制限するものです。このコードブロックとは、**load** や **call** の対象、関数ブロックの評価、そして条件 **if, else, do for, while** の後に続く中括弧内のコードです。局所 (local) 変数の名前が大域 (global) 変数と重なった場合、その局所変数の有効範囲から抜けるまでは、大域変数は隠されます。

初期化 (Startup (initialization))

起動時に、gnuplot はまずシステム用の初期設定ファイル **gnuplotrc** を探します。そのファイルの置き場所は gnuplot のインストール時に決定され、**show loadpath** で知ることができます。次にユーザのホームディレクトリ内に個人用の設定ファイルを探します。そのファイルは Unix 系のシステムでは**.gnuplot** であり、その他の処理系では **GNUPLOT.INI** となっています。(OS/2 では、環境変数 **GNUPLOT** に設定されている名前のディレクトリ内にそれを探します; Windows では、**APPDATA** を使用します)。Unix 系のシステムでは、追加で gnuplot は $XDG_CONFIG_HOME/gnuplot/gnuplotrc もチェックします。

文字列定数、文字列変数、文字列関数 (Strings)

文字列定数に加えて、ほとんどの gnuplot コマンドは文字列変数、文字列式または文字列を返す関数も受け付けます。例えば、以下の 4 つの plot のやり方は結果として全て同じ描画タイトルを生成します:

```
four = "4"
graph4 = "Title for plot #4"
graph(n) = sprintf("Title for plot #%d",n)

plot 'data.4' title "Title for plot #4"
plot 'data.4' title graph4
plot 'data.4' title "Title for plot #".four
plot 'data.4' title graph(4)
```

整数は、それが文字列結合演算子 (文字 '.') によって作用された場合は、文字列に変換されますので、以下の例も上と同様に動作します:

```
N = 4
plot 'data.'.N title "Title for plot #".N
```

一般に、コマンドラインの各要素は、それらが標準的な gnuplot への命令文法の一部分と認識されるもの以外は、有効な文字列変数としての評価のみが行なわれます。よって、以下のコマンド列は、恐らくは混乱を引き起こさないように避けられるべきですが、文法的には間違ってはいません:

```
plot = "my_datafile.dat"
title = "My Title"
plot plot title title
```

部分文字列 (substrings)

任意の文字列、文字列変数、文字列値関数に、範囲指定子をつけることにより部分文字列を指定できます。範囲指定子は [begin:end] の形で、begin は部分文字列の先頭位置、end は最後の位置です。位置指定は、最初の文字を 1 番目と見ます。先頭の位置、最後の位置は空、あるいは '*' でも構いません。その場合、それは元の文字列自体の先頭、あるいは最後を意味します。よって str[:] や str[*:*] はどちらも str の文字列全体を意味します。例:

```
eos = strlen(file)
if (file[eos-3:*] eq ".dat") {
    set output file[1:eos-4] . ".png"
    plot file
}
```

同等の関数 **substr(string, begin, end)** もあります。

文字列演算子 (string operators)

次の 3 つの二項演算子は文字列に作用します: 文字列の結合演算子 ".", 文字列の等号演算子 "eq", および文字列の不等号演算子 "ne" です。以下の例では TRUE が表示されます。

```
if ("A"."B" eq "AB") print "TRUE"
```

文字列関数 (string functions)

gnuplot は、文字列に作用する組み込み関数をいくつか持っています。汎用的な書式関数: 以下参照: **gprintf** (**p. 220**), **sprintf** (**p. 50**)。時刻書式関数: 以下参照: **strftime** (**p. 50**), **strptime** (**p. 50**)。文字列操作: 以下参照: **split** (**p. 59**), **substr** (**p. 50**), **strstrt** (**p. 50**), **trim** (**p. 59**), **word** (**p. 58**), **words** (**p. 58**)。

文字列エンコード (string encoding)

gnuplot の組み込み文字列操作関数は、UTF-8 エンコードを認識します (以下参照: **set encoding** (**p. 215**))。例:

```
set encoding utf8
utf8string = "α β γ "
strlen(utf8string) は 3 を返す (文字数であって、バイト数ではない)
utf8string[2:2] は "β " となる
strstrt(utf8string,"β ") は 2 となる
```

(訳注: いずれも UTF-8 エンドードで与えた場合)

置換とコマンドラインマクロ (Substitution)

gnuplot への命令文字列が最初に読み込まれた時点、すなわちまだそれが解釈され、もしくは実行される前の段階で、2 つの形式の単語の置換が実行されます。それらは逆引用符 (`) (ASCII 番号 96) で囲まれているか、または @ (ASCII 番号 64) が頭についた文字列に対して行なわれます。

逆引用符によるシステムコマンドの置換 (Substitution backquotes)

シェルコマンドを逆引用符 (`) で囲むことによってコマンド置換を行うことができます。このコマンドは子プロセスで実行され、その出力結果でコマンドラインの逆引用符で囲まれた部分を置き換えます。システムコマンドの終了ステータスは、変数 GPVAL_SYSTEM_ERRNO と GPVAL_SYSTEM_ERRMSG に返されます。

注意: 内部の復帰 ('\r') と改行 ('\n') 文字は、置換後の文字入力から取り除きません。

コマンド置換は、単一引用符内の文字列以外は、**gnuplot** のコマンドライン中、どこででも使用可能です。

例えば、以下は現在の日付とユーザー名のラベルを生成します:

```
set label "generated on `date +%Y-%m-%d` by `whoami`" at 1,1
set timestamp "generated on %Y-%m-%d by `whoami`"
```

以下は、カレントディレクトリ内のファイル名からなる配列を生成します:

```
FILES = split( "`ls -1`" )
```

文字列変数のマクロ置換 (Substitution macros)

文字 @ は、コマンドライン上でその文字列変数の値への置換を行なうのに使われます。文字列変数の値としての文は、複数の単語からなることも可能です。これにより文字列変数をコマンドラインマクロとして使うことが可能になります。この機能により展開できるのは文字列定数のみで、文字列を値に取る数式を使うことはできません。例:

```
style1 = "lines lt 4 lw 2"
style2 = "points lt 3 pt 5 ps 2"
range1 = "using 1:3"
range2 = "using 1:5"
plot "foo" @range1 with @style1, "bar" @range2 with @style2
```

この @ 記号を含む行は、その入力時に展開され、それが実際に実行されるときには次のように全部打ち込んだ場合と同じことになります。

```
plot "foo" using 1:3 with lines lt 4 lw 2, \
     "bar" using 1:5 with points lt 3 pt 5 ps 2
```

関数 exists() はマクロの評価に関して有用でしょう。以下の例は、C が安全にユーザ定義変数の名前に展開できるかどうかをチェックします。

```
C = "pi"
if (exists(C)) print C," = ", @C
```

マクロの展開は、単一引用符内、または二重引用符内では行なわれませんが、逆引用符 (`) 内ではマクロ展開されます。

マクロの展開は、gnuplot が新しいコマンド行を見たときに非常に早い段階で gnuplot が処理し、そしてただ一度だけそれを行います。よって、

```
A = "c=1"
@A
```

のようなコードは正しく実行しますが、以下のような行はだめです。それは、マクロの定義が同じ行にあるため展開に間に合わないからです。

```
A = "c=1"; @A   # will not expand to c=1
```

繰り返し用の中括弧内でのマクロの展開は、そのループが実行される前に行います。すなわち、ループ内では A 自体を再定義しても、@A は常に A の元の値に展開されます。

コマンドを完成させて実行するには、コマンド **evaluate** も有用でしょう。

文字列変数、マクロ、コマンドライン置換 (mixing_macros_backquotes)

文字列変数や逆引用符 (`) による置換、マクロによる置換の相互関係は少しややこしいです。逆引用符はマクロ置換を妨げないので、

```
filename = "mydata.inp"
lines = ` wc --lines @filename | sed "s/ .*//" `
```

は、mydata.ipn の行数を整数変数 lines に保存することになります。また、二重引用符は逆引用符の置換を妨げないので、

```
mycomputer = "`uname -n`"
```

は、システムコマンド **uname -n** の返す文字列を文字列変数 mycomputer に保存することになります。

しかし、マクロ置換は二重引用符内では機能しないので、システムコマンドをマクロとして定義してそれをマクロとして利用しかつ逆引用符置換を行なうことはでできません。

```
machine_id = "uname -n"
mycomputer = "`@machine_id`"  # うまくいかない !
```

この失敗は、二重引用符が @machine_id をマクロとして解釈することを妨げているからです。システムコマンドをマクロとして保存し、その後それを実行するには、逆引用符自体もマクロ内に含める必要があります。これは以下のようにマクロを定義することで実現できます。sprintf の書式には 3 種類の引用符全てが入れ子になっていることに注意してください。

```
machine_id = sprintf('"`uname -n`"')
mycomputer = @machine_id
```

区切りやカッコの使い方 (Syntax)

リストや座標がコンマ (,) 区切りであるのに対し、オプションやそれに伴うパラメータはスペース ( ) 区切りです。範囲はコロン (:) で区切ってかぎかっこ ([]) でくくりますし、文字列やファイル名は引用符でくくり、他にいくつかカッコ (()) でくくるものがあります。

コンマは以下の区切りで使用されます。**set** コマンドの **arrow**, **key**, **label** の座標; 当てはめ (fit) られる変数のリスト (コマンド **fit** のキーワード **via** に続くリスト); コマンド **set cntrparam** で指定されるとびとびの等高線の値やそのループパラメータのリスト; **set** コマンドの **dgrid3d dummy**, **isosamples**, **offsets**, **origin**, **samples**, **size**, **time**, **view** の引数; 目盛りの位置やそのループパラメータのリスト; タイトルや軸の見出しの位置; **plot**, **replot**, **splot** コマンドの x,y,z 座標の計算に使われる媒介変数関数のリスト; **plot**, **replot**, **splot** コマンドの複数の描画 (データ、または関数) のそれぞれの一連のキーワードのリスト。

(丸) カッコは、目盛りの見出しを (ループパラメータではなく) 明示的に集合与える場合の区切りとして、または **fit**, **plot**, **replot**, **splot** コマンドの **using** 指定での計算を指示するために使われます。

(カッコやコンマは通常の関数の表記でも使われます。)

かぎかっこは、**set**, **plot**, **splot** コマンドでは範囲を区切るのに使われます。

コロンは **range** (範囲) 指定 (**set**, **plot**, **splot** コマンドで使われる) の両端の値を区切るのに、または **plot**, **replot**, **splot**, **fit** コマンドの **using** 指定の各エントリを区切るのに使われます。

セミコロン (;) は、一行のコマンド行内で与えられる複数のコマンドを区切るのに使われます。

中カッコ ({}) は、拡張文字列処理モード (enhanced text mode) の記述や、if/then/else 文のブロックの区切りとして使われますし、または複素数を記述するのにも使われます: {3,2} = 3 + 2i となります。

引用符 (quote marks)

gnuplot は、文字列を区切るのに、二重引用符 (ASCII コード 34 番)、単一引用符 (ASCII コード 39 番)、および逆引用符 (`) (ASCII コード 96 番) の 3 種類の引用符を使います。

ファイル名は単一引用符、あるいは二重引用符内で囲みます。見出し (label)、タイトル (title)、またはその他の描画要素で使用される文字列定数や複数行文字列は、単一引用符、あるいは二重引用符内で囲みます。引用符で囲まれた文字列のさらなる処理の結果は、どの引用符記号を選ぶかによって変わります。

単一引用符文字列と二重引用符文字列の違いは以下の通り。

- 特殊文字 \n (改行), \r (復帰), \t (タブ) のバックスラッシュ (\) 処理は、2 重引用符内の文字列でのみ処理されます。
- 単一引用符内では、バックスラッシュ自体が通常の文字と見なされます。単一引用符文字列内で単一引用符自体を使うには、重ねて書いてください。つまり、文字列 "d\" s' b\\" と、'd" s'' b\' は全く同じになります。
- Unicode エスケープシーケンス \U+xxxx は、単一引用符内でも二重引用符内でも処理します。
- 拡張文字列 (enhanced text) 処理は、単一引用符文字列でも二重引用符文字列でも利用できます。
- 単一引用符内の 8 進エスケープシーケンス \ooo は、拡張文字列処理 (enhanced text) が有効な場合のみ処理します。二重引用符内の 8 進エスケープシーケンスは、常に処理します。

逆引用符は, コマンドライン中の置換のためにシステムコマンドを囲むのに使います。以下参照: **substitution (p. 82)**。

時間/日付データ (Time/Date)

gnuplot は入力データとして時間/日付情報の使用をサポートしています。この機能は **set xdata time**, **set ydata time** などのコマンドによって有効になります。

内部では全ての時間/日付は 1970 年からの秒数に変換されます。コマンド **set timefmt** は全ての入力に対するデフォルトの書式を定義します。データファイル、範囲、軸の目盛りの見出し、ラベルの位置と、日時データ値を受け入れるすべてのものへの入力の書式が、デフォルトでこれになります。一時には一つのデフォルト入力書式のみが有効です。よって、ファイル内の x と y の両方が時間/日付データである場合は、デフォルトではそれは同じ書式と解釈されます。しかし、このデフォルトは、**using** 指定で関数 **timecolumn** を用いて、それに対応する特定のファイルや列からデータを読みこむことにより、変えることが可能です。

秒数へ (秒数から) の変換は国際標準時 (UT; グリニッジ標準時 (GMT) と同じ) が使われます。各国標準時や夏時間への変換の機能は何も持ち合わせていません。もしデータがすべて同じ標準時間帯に従っているなら (そして全てが夏時間か、そうでないかのどちらか一方にのみ従うなら) これに関して何も心配することはあり

ません。しかし、あなたが使用するアプリケーションで絶対的な時刻を厳密に考察しなければいけない場合は、あなた自身が UT に変換すべきでしょう。

show xrange のようなコマンドは、その整数値を **timefmt** に従って解釈し直します。**timefmt** を変更してもう一度 **show** でその値を表示させると、それは新しい **timefmt** に従って表示されます。このため、(**set xdata** などにより) その軸に対するデータ型をリセットすると、その値は整数値として表示されることになります。

コマンド **set format** または **set tics format** は、指定された軸に対する入力が時間/日付であるなしに関わらず目盛りの見出しに使われる書式を定義します。

時間/日付情報がファイルから描画される場合、**plot**, **splot** コマンドでは **using** オプションを「必ず」使う必要があります。**plot**, **splot** では各行のデータ列の分離にスペースを使いますが、時間/日付データはその中にスペースを含み得るからです。もしタブ区切りを使用しているのなら、あなたのシステムがそれをどう扱うか確かめるために何度もテストする必要があるでしょう。

関数 **time** は、現在のシステム時刻を得るのに使えます。この値は、**strftime** 関数で日時文字列に変換できますし、**timecolumn** と組み合わせて相対的な日時グラフを作成するのにも使えます。引数の型はそれが返すものを決定します。引数が整数の場合は time() は現在の時刻を 1970 年 1 月 1 日からの整数として返し、引数が実数 (または複素数) ならば同様の値を実数として返しますが、小数 (秒以下) 部分の精度は、オペレーティングシステムに依存します。引数が文字列ならば、それを書式文字列であるとみなし、書式化された日時文字列を提供するようそれを **strftime** に渡します。

次の例は時間/日付データの描画の例です。

ファイル "data" は以下のような行からなるとします:

```
03/21/95 10:00  6.02e23
```

このファイルは以下のようにして表示されます:

```
set xdata time
set timefmt "%m/%d/%y"
set xrange ["03/21/95":"03/22/95"]
set format x "%m/%d"
set timefmt "%m/%d/%y %H:%M"
plot "data" using 1:3
```

ここで、x 軸の目盛りの見出しは "03/21" のように表示されます。

現在の gnuplot は、時刻をミリ秒精度で追跡し、時刻のフォーマットもそれに伴って変更されています。例: 現在の時刻をミリ秒精度で表示

```
print strftime("%H:%M:%.3S %d-%b-%Y",time(0.0))
18:15:04.253 16-Apr-2011
```

以下参照: **time_specifiers (p. 221)**, **set xtics time (p. 310)**, **set mxtics time (p. 250)**。

ウォッチポイント (Watchpoints)

ウォッチポイントは、あなたが使用する gnuplot が configure 時にオプション –enable-watchpoints をつけて作られた場合にのみサポートされます。この機能は試験段階です (詳細は、今後の版で変更される可能性があ

ります)。

書式:

```
plot FOO watch {x|y|z|F(x,y)} = <value>
plot FOO watch {x|y|z|F(...)} = <value> {label <文字列値関数>}
plot FOO watch mouse

set style watchpoints nolabels
set style watchpoints label <label-properties>

unset style watchpoints      # デフォルトスタイルに戻す

show watchpoints     # 直前の plot コマンドからのすべてのウォッチ
                     # ポイントを要約表示
```

一つのウォッチポイントは、x, y, z 座標のいずれか、または関数 f(a,...) に対する一つの対象値です。各ウォッチポイントは、コマンド **plot** または **splot** 内の一つのグラフに付随します。その場合、そのグラフのすべての構成線分に対し、そのグラフに付随するすべてのウォッチポイントがチェックされ、一つ以上のウォッチポイントの対象がその線分上の点で満たされるかどうかを調べます。対象条件を満たす (「ヒット」("hits") と呼びます) 点の一覧は、グラフの描画毎に累積されます。

例えば、対象値 y=100 のウォッチポイントがある場合、各線分に対して、その両端点の y 座標がその対象値を挟んでいるかをチェックします。もしそうなら、その線分上のある点 [x,y] が、対象条件 y=100 を完全に満たすことになります。そして線形補間、または 2 分反復法によりその対象点が見つかります。

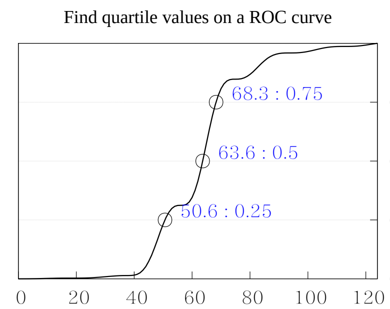
一つの plot コマンド上の複数のウォッチポイントは、順番に番号づけします。各描画要素毎に、1 つ以上のウォッチポイントを指定できます。例:

```
plot DATA using 1:2 smooth cnormal watch y=0.25 watch y=0.5 \
    watch y=0.75
```

ウォッチポイントは、**with lines** と **with linespoints** の描画スタイルでのみ機能しますが、**splot with lines** の 2 次元 xy 射影 (**set view map**) でも使えます。

直前の plot コマンドで対象にヒットしたウォッチポイントは、WATCH_n という名前の配列に保存されます。直前の plot コマンドからヒットしたすべてのウォッチポイントの要約を表示させることもできます。以下参照: **show watchpoints (p. 301)**。

```
gnuplot> show watchpoints
Plot title:     "DATA using 1:2 smooth cnormal"
  Watch 1 target y = 0.25          (1 hits)
          hit 1   x 49.7  y 0.25
  Watch 2 target y = 0.5           (1 hits)
          hit 1   x 63.1  y 0.5
  Watch 3 target y = 0.75          (1 hits)
          hit 1   x 67.8  y 0.75
```



平滑化: 線分は、それらを描画する際にチェックします。平滑化しないデータ描画に対しては、これは、補間によって見つかる対象点は、2 つのデータ点を結ぶ線分の上に完全に乗ることを意味します。データグラフを平滑化する場合は、ヒットする点は平滑化された曲線に対する線分の上にあります。それが平滑化しないデータに対する対象点よりも精度が良いか悪いかは、平滑化の当てはめの質に依存します。

精度: 線分が関数描画で生成されたものであれば、$f(x) = y$ となる x の値は 2 分反復法で見つけます。それ以外の場合は、その線分に沿う線形補間で座標 [x,y] を近似します。

ウォッチラベル (watch labels)

デフォルトではラベルは、対象 "watch mouse" に対して常に生成します。他の対象に対しても、コマンド **set style watchpoint labels** を使うか、または plot コマンドの watchpoint の対象の一部としてラベルを指定することでラベルをオンにできます。

デフォルトのラベル文字列は "x : y" で、x, y は対象点の座標、その書式化は対応する軸に対する現在の設定を使用します。

例:

```
set y2tics format "%.2f°"
set style watchpoint labels point pt 6
plot FOO axes x1y2 watch mouse
```

各 watch の対象に対して、文字列定数か、または文字列を返す関数を通してカスタマイズした文字列を与えることができます。

例:

```
set style data lines
plot FOO watch y=50 label "50" watch y=100 label "100"
f(x,y) = 2*x - y      # f(x,y) = 0 は斜線を定義
plot $CONTOURS using 1:2:3 watch f(x,y) = 0 label sprintf("%.2f",z)
```

各 watchpoint ラベルは、テキストボックス内に置きます。以下参照: **set style textbox (p. 288)**。

関数対象のウォッチポイント (watchpoint function target)

ウォッチポイントは、関数に対象値を設定できます。その関数は、グラフの各線分の始点と終点で評価しますが、終点の値が対象値をまたいだ場合、線形補間ののちにそのヒットした座標 [x,y] を報告します。

```
plot FOO watch func(...)=targetvalue
```

以下の自明な例は、その曲線が直線 $y = ax + b$ と交わる場所にラベルを置くというウォッチポイント対象を設定します。[x,y] を評価したとき、関数 Line(a,b) は、現在の y 座標から値 $ax + b$ までの偏差を返すので、描画グラフと直線 ax+b の交点で評価したときには、その関数は 0 を返すことになります。そのため我々はこれをウォッチの対象として設定します。

```
Line(x,y) = (a*x + b) - y
plot cos(x) watch Line(x,y)=0,   a*x + b
```

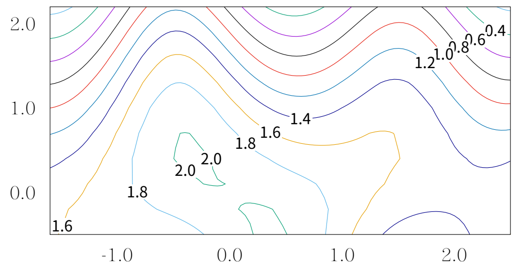
評価の際に、対応する点の x, y, z の値は、それらが名前付きパラメータであるかどうかに係わらず、その関数に対して有効なので、以下の変種も上の例と同じ結果を産みます。

```
Line(dummy1,dummy2) = dummy1*x + dummy2 - y
plot cos(x) watch Line(a,b)=0,   a*x + b
```

さらに、インライン関数の代わりの関数ブロックを使っても同じです。

```
function $Line() << EOF
   return (a*x + b) - y
EOF
plot cos(x) watch $Line()=0,   a*x + b
```

より興味深い使用例は、等高線描画での等高線ラベルの配置の抑制です。gnuplot には、等高線ラベルの自動的な配置を変更するオプションがいくつかあります (以下参照: **set cntrlabel (p. 203)**)。しかしグラフが混んでいる場合はそのラベル付けでは満足できないかもしれません。代わりに、ウォッチポイント関数対象が満たされる場所にのみラベルを配置することができます。



以下の例は、描画領域を横断する直線を選択し、そして等高線ラベルは、等高線がその一つの直線を横切る場所にのみ配置します。完全に機能する例は、デモ **watch_contours.dem** を参照してください。

```
set style watchpoint labels center nopoint font ",9"
set style textbox noborder opaque margins 0.5, 0.5
line(a,b) = a*x+b - y
a = 0.6; b = 0.5

set contours
set cntrparam levels incr 0, .2, 4
set view map

splot f(x,y) with lines nosurface watch line(a,b)=0 label \
  sprintf("%.1f",z)
```

各ラベルを、等高線の z 座標を使って生成します。デフォルトのウォッチポイントラベル " x : y " は、このグラフでは有益ではないので、等高線の z 値を印字する書式関数を使用しています。

ウォッチマウス (watch mouse)

現在のマウス x 座標をウォッチ対象として使用すると、マウスの水平位置を追跡しながら、グラフの線に沿って移動するラベルを生成します。これにより同じグラフ上の複数の曲線の y の値を同時に表示することも可能です。現在の位置を示す点とラベルの見た目は、**set style watchpoint** で変更できます。

例:

```
set style watchpoint labels point pt 6 ps 2 boxstyle 1
set style textbox 1 lw 0.5 opaque
plot for [i=1:N] "file.dat" using 1:(column(i)) watch mouse
```

第 II 部

描画スタイル (Plotting styles)

gnuplot では、たくさんの描画スタイルが利用できます。それらは、アルファベット順に以下に紹介されています。コマンド **set style data** と **set style function** は、それ以降の **plot** や **splot** コマンドに対してデフォルトの描画スタイルを変更します。

描画スタイルは、コマンド **plot** や **splot** の一部分として、明示的にオプション指定することもできます。一つの描画の中で、複数の描画スタイルを組み合わせたい場合は、各要素に対して描画スタイルを指定する必要があります。

例:

```
plot 'data' with boxes, sin(x) with lines
```

各描画スタイルは、それ自体がデータファイルからのいくつかのデータの組を期待します。例えば、デフォルトでは **lines** スタイルは、y の値だけの 1 列のデータ (x の値は暗黙に順番に取られる)、または最初が x, 次が y の 2 つの列を期待しています。ファイルの何列のデータを描画データと解釈させるうまい方法に関する情報については、以下参照: **using (p. 176)**。

Arrows

2 次元のスタイル **arrows** は、各点 (x,y) に長さと方向の角を指定して矢印を描きます。追加の入力列は、各データ点毎の variable color 情報、あるいは arrow style として使用します。これは、矢先の場所の指定方法以外は 2 次元描画スタイルの **with vectors** と同じで、矢先は delta_x + delta_y でなく length (長さ) + angle (角) で与えます。以下参照: **with vectors (p. 124)**。

```
4 列:  x  y  length  angle
```

キーワード **with arrows** の後ろには、直接 arrow style 属性を指定したり定義済みの arrow style 番号を指定、または **arrowstyle variable** による各矢印毎に適用したい arrow style 番号を他の列から読み込ませることの指定、などを追加できます。

正の **length** 値は、x 軸の座標で解釈します。-1 < **length** < 0 の値は、水平グラフ座標、すなわち |length| を全体のグラフ幅に対する割合として解釈します。gnuplot は x と y の拡大率の差、または描画アスペクト比に関して調整して、見た目の長さが方向角とは独立であるようにしようとします。

angle は常に度単位で指定します。

Arrowstyle variable

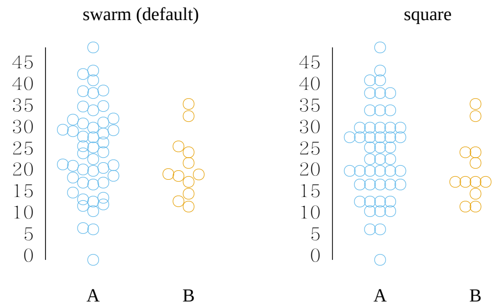
描画スタイル **with arrows** と **with vectors** で、入力データの追加列を与えて、事前に **set style arrow** で定義した arrow スタイルに対応するスタイル番号を指定することができます。

例:

```
set style arrow 1 head nofilled linecolor "blue" linewidth 0.5
set style arrow 2 head filled linecolor "red" linewidth 1.0
# 第 5 列の値は 1 か 2 のいずれかで、それが事前に定義したスタイル
# のうち使用する方を決定する
plot DATA using 1:2:3:4:5 with arrows arrowstyle variable
```

ビースウォーム描画 (Bee swarm plots)

ビースウォーム (bee swarm) グラフは、揺らぎ (jitter) を適用して重複点を分離することにより得られる結果です。その典型的な例は、各点に x 座標を決定する 2 つ以上のカテゴリによって表わされる y の値の分布の比較です。重なりの判定基準や、jitter で使用する移動パターンを制御する方法に関しては、以下参照: **set jitter (p. 228)**。この図のグラフは、異なる jitter の設定に対する同じ plot コマンドによって作られたものです。



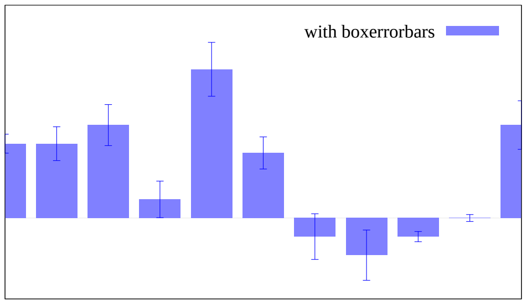

```
set jitter
plot $data using 1:2:1 with points lc variable
```

Boxerrorbars

描画スタイル **boxerrorbars** は 2 次元のデータ描画でのみ利用可能です。これは、3 列、または 4 列、または 5 列のデータが必要です。入力列を追加 (4,5,6 列目) すると、それらは各データ点毎の variable color 情報 (以下参照: **linecolor (p. 71)**, **rgbcolor variable (p. 73)**) として使われます。

```
3 列:  x  y  ydelta
4 列:  x  y  ylow  yhigh          (xdelta <= 0 は boxwidth を使用)
5 列:  x  y  ylow  yhigh  xdelta  (xdelta <= 0 は boxwidth を使用)
```

y の誤差を "ydelta" の形式で与える場合は、箱の横幅は 4 列目の値を使用し、"ylow yhigh" の形式で与える場合は、横幅は 5 列目の値を使用します。xdelta の値が 0 かまたは負の場合は、箱の横幅は、事前に与える boxwidth の値で制御します。以下参照: **set boxwidth (p. 200)**。



誤差線の垂直方向は、**yerrorbars** スタイル同様に y の誤差の値を表現するように描きます。y-ydelta から y+ydelta まで、あるいは ylow から yhigh まで、これらは何列のデータが与えられているかによって決まります。誤差線の描画で使用する線スタイルは、**set bars** を使って制御できますが、使わなければ、箱の境界と同じもので誤差線を描きます。

非推奨: 古いバージョンの gnuplot では、"ylow yhigh" の誤差形式の 4 列のデータに対して **boxwidth = -2.0** を特別に扱っていました。その場合、隣接する箱の間に隙間ができないように箱の幅を調節していました。この処理は、後方互換性のために残されていますが、将来のバージョンでは削除するでしょう。

Boxes

2 次元グラフでは、スタイル **boxes** は与えられた x 座標を中心とし、x 軸から (すなわち y=0 からであって、グラフの境界からではない) 与えられた y 座標まで伸ばした長方形の箱を書きます。箱の横幅は入力の追加列

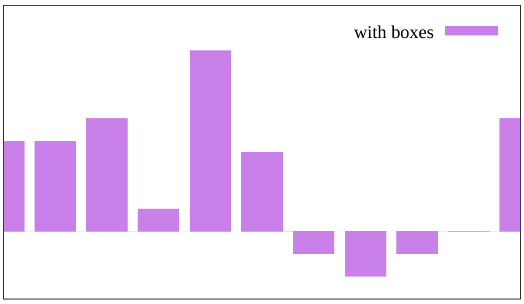
で指定することもできますし、**set boxwidth** で制御することもできます。そうでなければ、各箱は、隣接する箱同士がくっつくように引き伸ばされます。

3 次元グラフでは、スタイル **boxes** は与えられた x, y 座標を中心とし、xy 平面 (z=0) から与えられた z 座標まで伸ばした直方体の箱を書きます。x 方向の箱の幅は、別の入力列か **set boxwidth** で指定でき、y 方向の箱の奥行きは、**set boxdepth** で制御できます。箱は、自動的にはくっつくように引き伸ばされません。

2 次元の boxes (2D boxes)

plot with boxes は、基本的に 2 列、または 3 列のデータを使用します。さらに入力列を追加すると、それは、可変線色 (variable line color) や塗り潰し色の情報として使用します。以下参照: **rgbcolor variable** (p. **73**)。

```
2 列:   x   y
3 列:   x   y   x_width
```



箱の幅は 3 つのうち一つの方法で決定されます。入力データに 3 列目のデータがある場合は、それを箱の幅として使用します。そうでない場合でコマンド **set boxwidth** で箱の幅をセットした場合は、それを使います。そのどちらでもない場合は、隣接する箱がくっつくように箱の幅を自動的に計算します。

箱の内部は現在の塗りつぶしスタイル (fillstyle) で塗りつぶします。それ以外に、塗りつぶしスタイルを plot コマンド上で指定することもできます。以下参照: **set style fill** (p. **283**)。塗り潰し色を plot コマンドで指定しなければ、現在の線色を使用します。

例:

データファイルを単色塗りした箱で描画し、箱同士を少し垂直方向にスペースを空ける (棒グラフ):

```
set boxwidth 0.9 relative
set style fill solid 1.0
plot 'file.dat' with boxes
```

塗り色を明示してパターン塗りスタイルの箱で sin と cos のグラフを描画:

```
set style fill pattern
plot sin(x) with boxes fc 'blue', cos(x) with boxes fc 'gold'
```

sin はパターン 0 で、cos はパターン 1 で描画されます。追加される描画は出力ドライバがサポートするパターンを循環的に使用します。

3 次元の boxes (3D boxes)

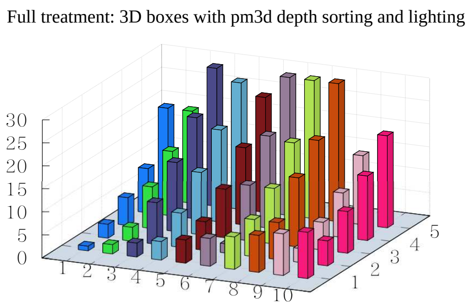
splot with boxes には、少なくとも 3 列の入力列が必要です。さらに入力列を追加すると、それは箱の幅や塗り潰し色の情報として使用します。

```
3 列:  x  y  z
4 列:  x  y  z  [x_width または color]
5 列:  x  y  z  x_width  color
```

最後の列は、splot コマンドで明示的に variable カラーモードを指定している場合のみ色として使用します。例:

```
splot 'blue_boxes.dat' using 1:2:3 fc "blue"
splot 'rgb_boxes.dat' using 1:2:3:4 fc rgb variable
splot 'category_boxes.dat' using 1:2:3:4:5 lc variable
```

最初の例は、すべての箱を青に塗り、幅はあらかじめ **set boxwidth** で設定した幅を使用します。2 つ目の例は、4 列目を 24-bit RGB 色として認識するため、箱の幅は相変わらず **set boxwidth** の幅を使います。3 つ目の例は、4 列目の値を箱の幅として読み、5 列目の整数値を色を提供する線種と解釈します。



デフォルトでは、箱には太さはなく、それらは単に xz 平面に平行な 1 つの長方形で構成されますが、y 方向の幅として 0 でない値を設定すれば 4 面と天井をを持つ本当の箱に変更できます。以下参照: **set boxdepth (p. 200)**。

3 次元の箱は、曲面ではなく pm3d 長方形として処理しています。よって、表裏の描画順は、**set hidden3d** の影響を受けません。以下参照: **set pm3d (p. 267)**。gnuplot バージョン 6 は、箱の端はグラフの fill style の境界色で色付けします。これはバージョン 5 とは異なる変更です。最良の結果を得るには、**set pm3d depthorder base** と **set pm3d lighting** を組み合わせてください。

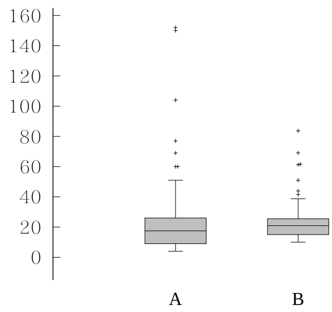
Boxplot

boxplot は、値の統計的な分布を表現する一般的な方法です。gnuplot の boxplot は常に鉛直向きで、値の分布は y 軸に沿って表示します。四分位境界は、1/4 の点が第一四分位境界以下の y の値を持つように、1/2 の点が第二四分位境界 (メジアン) 以下の y の値を持つように、等と決定されます。第一四分位と第三四分位の間の領域を囲むように箱を描画し、メジアン値のところには水平線を描きます。箱ひげは、箱からユーザ指定限界まで延長します。それらの限界の外にある点 (外れ値) は、ひとつひとつ描画します。boxplot の幅は、**set boxwidth** か plot コマンドの **using** 指定の 3 列目を与えることで制御できます。

書式

```
2 列:  x-position        y-value
3 列:  x-position        y-value  boxwidth
4 列:  first-x-position  y-value  boxwidth  category
```

boxplot の水平位置は、通常 plot コマンドの **using** 指定の最初の列に定数値 (x-position) として与えたものになります。その boxplot の位置の下に特定のラベルを置くことができますが、それには、plot コマンドに **xticlabel** 指定を追加するか (2 列、または 3 列書式)、または独立したデータ列内の文字列を与えるか (4 列書式)、の方法があります。以下の例は、両方とも boxplot サンプル図のレイアウトと同様のグラフを生成します。



例

```
#
# 2 つの異なるファイルからの y 値の分布の比較
set border 2                    # 左境界線のみ
set xtics nomirror scale 0      # 目盛り刻み無し、ラベルのみ
set ytics rangelimited nomirror
plot 'dataset_A' using (1.):2:xticlabel('A') with boxplot, \
     'dataset_B' using (2.):2:xticlabel('B') with boxplot

#
# 同じファイル内の 2 つのデータカテゴリの y 値の比較
# 各行の 1 列目にはカテゴリを示す文字列 ("A" か "B")、2 列目にデ
# ータ値が含まれ、ラベルはカテゴリ文字列から自動生成する
start_x = 1.0
boxwidth = 0.5
plot 'mixeddata' using (start_x):2:(boxwidth):1 with boxplot
```

デフォルトでは、using で第 2 フィールドに指定した列のすべての y の値から、ただ 1 つの boxplot を生成します。もし **using** で第 4 フィールドを指定した場合は、その入力列の内容は個別のカテゴリを識別するための文字列として使います。入力列にある個々のカテゴリに対して、別な boxplot を描画します。それらの boxplot 間の水平間隔は、デフォルトでは 1.0 ですが、それは **set style boxplot separation** で変更できます。デフォルトでは、カテゴリ識別子は、各 boxplot の下の目盛りラベルとして書きます。もしカテゴリ列に数値が含まれていても、それはやはりあくまで文字列として扱われるので、boxplot x 座標には通常は対応しません。

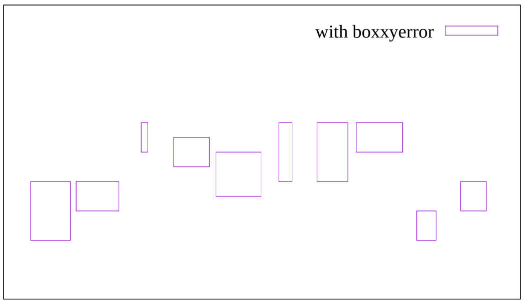
入力ファイル内のデータ点の順序は重要ではありません。データ点が入力ファイル内で 2 行の空行で分離される複数のブロックになっている場合は、個々のブロックはキーワード **index** で選択するか、またはデータブロック番号 (**column(-2)**) を第 4 列のレベル値として使用することができます。以下参照: **pseudocolumns (p. 178)**, **index (p. 170)**。

デフォルトでは箱ひげは、箱の端から、y の値が四分位範囲の 1.5 倍以内で最も離れているような点まで延長します。外れ値は、デフォルトでは円 (point type 7) で描きます。箱ひげの端の棒の幅は、**set bars** か **set errorbars** で制御できます。複数の外れ値が同じ y の値を持つ場合、水平方向に 1 文字幅分ずらしますが、その間隔は、**set jitter** で制御できます。

これらのデフォルトの性質は **set style boxplot** コマンドで変更できます。以下参照: **set style boxplot (p. 281)**, **bars (p. 216)**, **boxwidth (p. 200)**, **fillstyle (p. 283)**, **candlesticks (p. 96)**。

Boxxyerror

boxxyerror 描画スタイルは 2 次元のデータ描画でのみ利用可能です。これは、**xyerrorbars** スタイルが線分の交差で表現するところを長方形で表現することを除けばほぼ同じです。これは入力データの 4 列、または 6 列を使用します。余分な入力列 (5 列目、または 7 列目) は、データ点毎の可変色 (variable color) 情報の提供として扱います (以下参照: **linecolor (p. 71)**, **rgbcolor variable (p. 73)**)。




```
4 列:  x  y  xdelta  ydelta
6 列:  x  y  xlow  xhigh  ylow  yhigh
```

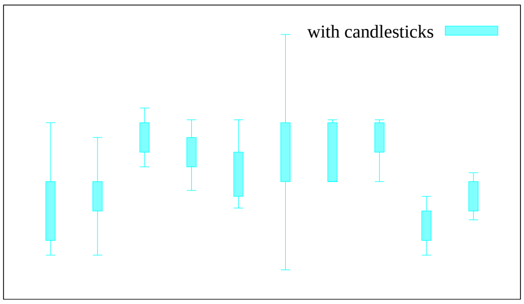
箱の幅と高さは **xyerrorbars** スタイル同様 x, y の誤差から決定されますつまり、xlow から xhigh までと ylow から yhigh まで、または x-xdelta から x+xdelta までと y-ydelta から y+ydelta まで。これらは何列のデータが与えられているかによって決まります。

6 列の形式のコマンドは、任意の x, y の幅の長方形を書く簡単な方法を提供します。

箱の内部は現在の塗りつぶしスタイル (fillstyle) に従って塗られます。詳細は、以下参照: **set style fill (p. 283)**, **boxes (p. 92)**。plot コマンド上で新しい塗りつぶしスタイルを指定することもできます。

Candlesticks

candlesticks スタイルは、金融データの 2 次元のデータ描画、および統計データのひげ付きの棒グラフを生成するのに使えます。記号は、水平方向には x を中心とし、垂直方向には開始値 (open) と終値 (close) を境界とする長方形が使われます。そして、その x 座標のところに長方形のてっぺんから最高値 (high) までと、長方形の底から最安値 (low) までの垂直線が引かれますが、この垂直線は最高値と最安値が入れ替わっても変更されません。



基本的に 5 列のデータが必要です:

```
金融データ:   date  open  low  high  close
箱ひげ描画:   x  box_min  whisker_min  whisker_high  box_high
```

長方形の幅はコマンド **set boxwidth** で制御できますが、以前の gnuplot への後方互換性として、boxwidth パラメータが設定されていない場合は **set errorbars <width>** を長方形の幅として取ります。

これの代わりに、箱ひげ (box-and-whisker) のグループ化に関する明示的な幅の指定を、追加の 6 番目のデータで指定できます。その幅は、x 座標と同じ単位で与えなければいけません。

入力列を追加 (6 列目、または 6 列目がデータの幅として使われる場合は 7 列目) すると、それらは各データ点毎の variable color 情報 (以下参照: **linecolor (p. 71)**, **rgbcolor variable (p. 73)**) として使われます。

デフォルトでは、鉛直線分のてっぺんと底には垂直に交わる水平線は引かれません。それを引きたい場合、例えば典型的な例は箱ひげ図 (box-and-whisker plot) での使用ですが、描画コマンドにキーワード **whiskerbars** を追加してください。デフォルトでは、水平線は箱 (candlestick) の水平幅一杯に引かれますが、それは全体の幅に対する割合を指定することで変更できます。

金融データの通常の慣習では、(開始値) < (終値) の場合は長方形は空で、(終値) < (開始値) の場合は単色塗りします。現在の fillstyle に "empty" をセットしている場合は、実際にこうなります。以下参照: **fillstyle (p. 283)**。fillstyle に solid (単色塗り)、または pattern (パターン) をセットしている場合は、開始値、終値に関係なく、すべての箱にそれが使われます。以下参照: **set errorbars (p. 216)**, **financebars (p. 102)**。また、以下も参照してください。`candlestick`

と `finance`

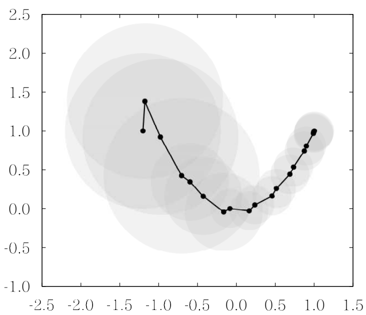
のデモ。

注意: 箱ひげグラフ上に記号や線を追加して置くには、追加の描画要素が必要になります。以下の最初の例は、2 番目の要素で、箱を潰して中央値の場所に置く線分にしています。

```
# データ列:  X '最小値' '1/4 位の値' '中央値' '3/4 位の値' '最大値'
set errorbars 4.0
set style fill empty
plot 'stat.dat' using 1:3:2:6:5 with candlesticks title 'Quartiles', \
     ''         using 1:4:4:4:4 with candlesticks lt -1 notitle

# ひげの上に水平線を伴う描画で、水平線の幅を全体幅の 50% にする
plot 'stat.dat' using 1:3:2:6:5 with candlesticks whiskerbars 0.5
```

以下参照: **set boxwidth (p. 200)**, **set errorbars (p. 216)**, **set style fill (p. 283)**, **boxplot (p. 94)**。

Circles



スタイル **circles** は、各データ点に明示された半径の円を描画します。半径は、常に描画の水平軸 (x または x2) の単位で解釈されます。y 方向の縮尺と描画のアスペクト比は、いずれも無視されます。半径を各点毎の列として指定しない場合、それは **set style circle** から取ります。この場合、半径は graph か screen の座標系を使用できます。

各点毎に、そして事前に、設定する属性の多くの組み合わせ指定が可能です。2 次元描画では、以下が指定できます。

```
using x:y
using x:y:radius
using x:y:color
using x:y:radius:color
using x:y:radius:arc_begin:arc_end
using x:y:radius:arc_begin:arc_end:color
```

デフォルトでは完全な円を描画します。この結果は **points** と pointtype 7 を用いて、可変点サイズのグラフを書くことと同様ですが、その円を x 軸の範囲で伸縮することが違います。4 列目、5 列目に開始角と終了角 (単位は度) を指定することで円弧の一部を描画することもできます。

using 指定の最後の列で、円毎の色も指定できます。この場合、plot コマンドには **lc variable** か **fillcolor rgb variable** のような変動色指定を入れる必要があります。

以下参照: **set style circle (p. 286)**, **set object circle (p. 254)**, **set style fill (p. 283)**。

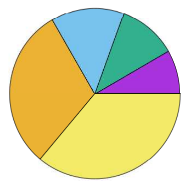
3 次元描画では、using 指定には以下のものが必要です。

```
splot DATA using x:y:z:radius:color
```

変動色の列はオプション (省略可) です。

例:

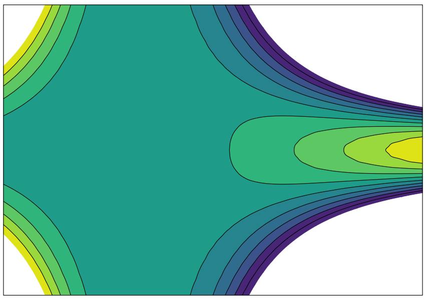
```
# 面積が 3 列目の値に比例するような円を描画
set style fill transparent solid 0.2 noborder
plot 'data' using 1:2:(sqrt($3)) with circles, \
     'data' using 1:2 with linespoints

# 円の代わりにパックマンを描画
plot 'data' using 1:2:(10):(40):(320) with circles

# インランデータで円グラフを描画
set xrange [-15:15]
set style fill transparent solid 0.9 noborder
plot '-' using 1:2:3:4:5:6 with circles lc var
0    0    5    0    30    1
0    0    5   30    70    2
0    0    5   70   120    3
0    0    5  120   230    4
0    0    5  230   360    5
e
```



Contourfill

書式:

```
splot f(x,y) with contourfill {at base} {fillstyle <style>}
```



contourfill は、3 次元描画スタイルで、pm3d 曲面を z の等高線に沿って切った断片の色付けに使います。これは、2 次元射影 (**set view map** か **with contourfill at base**) で、等高線の間を単色で塗り潰した 2 次元の等高線グラフを作成するのに使えます。各断片の境界と色の割り当ては、いずれも **set contourfill** で制御できます。以下も参照: **pm3d (p. 267)**, **zclip (p. 268)**。

デフォルトの塗り潰し属性は **set pm3d** から取り出しますが、splot コマンドで異なる fillstyle を与えればそれを変更できます。

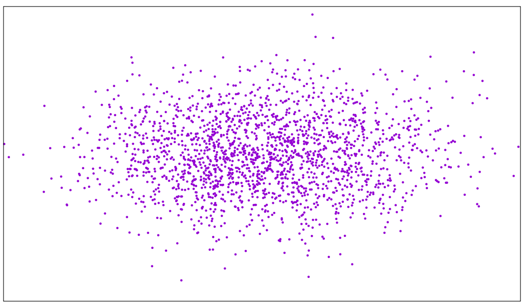
この描画スタイルは、**set contours** と組み合わせて、その断片を切り分ける等高線を重ね描きすることができます。**set contourfill** の断片の境界を **set cntrparam** の等高線境界とを合わせるように注意してください。

```
# ztics で定義される断片境界
# 断片の z の中央値に割り当てられるパレットで色付け
set pm3d border retrace
set contourfill ztics
set ztics -20, 5, 20
set contour
set cntrparam cubic levels increment -20, 5, 20
set cntrlabel onecolor
set view map
splot g(x,y) with contourfill, g(x,y) with lines nosurface
```

デフォルトでは、等高線は元の 3 次元曲面上に描画します。**with contourfill at base** で描画すると、代わりに色付き曲面を底面に投影します。曲面とその色付き等高線の投影の両方を描くには

```
set hidden3d front
splot f(x,y) with lines, f(x,y) with contourfill at base
```

Dots



dots スタイルは各点に小さなドットを描画します。これはたくさんの点からなる散布図の描画に便利でしょう。2 次元描画では 1 列、または 2 列の入力データが、3 次元描画では 3 列のデータが必要です。

出力形式によっては (post, pdf など)、ドットの大きさは linewidth を変更することで制御できることもあります。

```
1 列:  y          # x は行番号
2 列:  x  y
3 列:  x  y  z    # 3D のみ (splot)
```

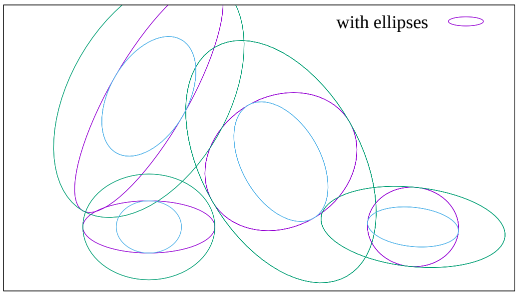
Ellipses

書式:

```
plot <data> with ellipses {units [xx|xy|yy]}
```

スタイル **ellipses** は、各データ点に楕円 (ellipse) を描画します。このスタイルは、2 次元描画にのみ適用されます。各楕円は、中心、主軸直径、副軸直径、x 軸と主軸のなす角、で表現されます。

```
2 列: x y
3 列: x y diam  (主軸、副軸の両方に使用)
4 列: x y major_diam minor_diam
5 列: x y major_diam minor_diam angle
```



2 列のデータのみが与えられた場合は、それらは中心の座標とみなされ、楕円はデフォルトの大きさで描画されます (以下参照: **set style ellipse (p. 287)**)。楕円の向きは、主軸と x 軸のなす角で定義されますが、それもデフォルトの ellipse のスタイルから取られます (以下参照: **set style ellipse (p. 287)**)。

3 列のデータが与えられた場合は、3 列目は主、副両軸の直径 (幅) として使われます。向きはデフォルトで 0 になります。

4 列のデータが与えられた場合は、それらは中心の座標、主軸直径 (幅)、副軸直径として使わわれます。これらは直径であり、半径でないことに注意してください。一方の直径が負ならば、直径は両方とも **set style ellipse** で設定したデフォルト値を使います。

5 列のデータとして、向きの角度 (単位は度) を指定することもできます。楕円は、3,4,5 列の値は負の値として指定することで、それらのデフォルトの値を利用して楕円を書かせることもできます。

上のすべての場合で、variable color データを最後の列 (3,4,5,6 列目) として追加できます。以下参照: **colorspec (p. 71)**。

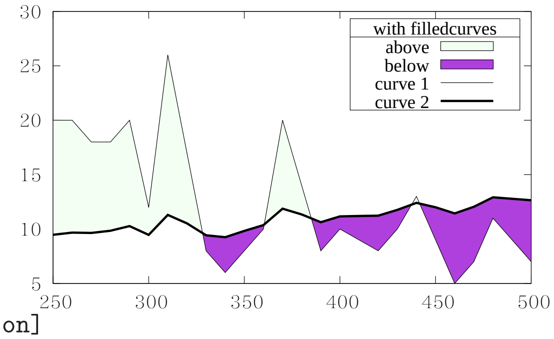
キーワード **units**: **units xy** が描画指定に含まれている場合、主軸直径は水平軸 (x または x2) の単位、副軸直径は垂直軸 (y または y2) の単位であるとみなされます。x 軸と y 軸の縮尺が異なる場合、主軸と副軸の比は回転後には正しくはなりません。**units xx** は、直径は両軸とも x 軸の単位で計算します。**units yy** は、直径は両軸とも y 軸の単位で計算します。後の 2 つは、描画のサイズを変更しても、楕円は正しいアスペクト比を持ちます。plot コマンドで **units** を省略した場合は、**set style ellipse** の設定を使います。

例 (楕円を有効な線種を周期的に使用して描画):

```
plot 'data' using 1:2:3:4:(0):0 with ellipses
```

以下も参照: **set object ellipse (p. 254)**, **set style ellipse (p. 287)**, **fillstyle (p. 283)**。

Filledcurves



スタイル **filledcurves** は、2 次元描画と 3 次元描画 (バージョン 6.1 より) の両方で利用できます。

2 次元描画スタイルは、3 種類の異なる指定が可能です。最初の 2 種類は、1 つの関数描画、あるいは (x,y) 2 列の入力データ用のものです。

2 次元描画用の書式:

```
plot f(x) with filledcurves [option]
plot DATA using 1:2    with filledcurves [option]
plot DATA using 1:2:3  with filledcurves [option]
```

ここで、オプションは以下のうちのいずれかです:

```
closed
{above|below} x1 x2  y r=<a> xy=<x>,<y>
between
```

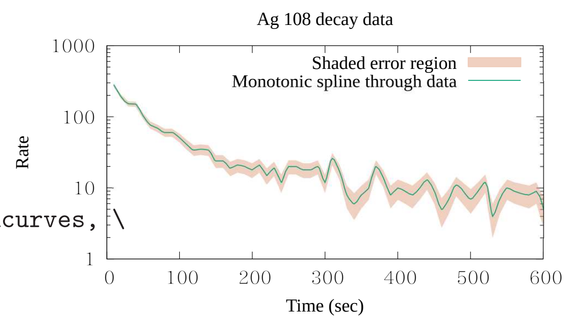
最初のものは **closed** で、これは曲線それ自身を閉多角形と見なします。入力データが 2 列の場合にはこれがデフォルトです。

```
filledcurves closed   ... 丁度閉曲線で囲まれる領域
```

2 種類目は指定された軸、あるいは水平線、垂直線、与えられた点などと、曲線との間に作られる領域を塗りつぶします。この場合、さらに塗り潰し領域を指定した直線の上または下に制限できます。

```
filledcurves x1        ... x1 軸
filledcurves x2        ... x2 軸 (y1, y2 軸も同様)
filledcurves y=42      ... 直線 y=42, すなわち x 軸と平行
filledcurves xy=10,20 ... x1,y1 軸での点 10,20 (扇型のような形状)
filledcurves above r=1.5  極座標での動径軸の 1.5 の外側の領域
```

3 種類目は、x 座標の集合が同じである 2 つの曲線の間の領域を塗り潰します。これは、(x, y1, y2) の 3 列の入力データが必要です。入力データが 3 列以上の場合にはこれがデフォルトです。



2 列目が y の値で、3 列目がその誤差データである場合は、この 3 列データ列指定を、不確定領域をその片側に持つことを示す実線と組み合わせて利用することができます。これに似た 3 次元描画スタイル **zerrorfill** も参照してください。

```
plot $DAT using 1:($2-$3):($2+$3) with filledcurves, \
     $DAT using 1:2 smooth mcs with lines
```

Above/below

above と below オプションは

```
plot f(x) with filledcurves {above|below} {y|r}=<val>
```

および

```
plot DATA with filledcurves using 1:2:3 with filledcurves {above|below}
```

の形のコマンドに適用可能です。どちらの場合でも、これらのオプションは塗りつぶし領域を、境界線、または境界曲線の片側に制限します。

データファイルから描かれた曲線の塗りつぶしを拡大すると、何もなくなったり正しくない領域になることがありますが、それは gnuplot が、領域ではなく点や線をクリッピングしているからです。

<x>, <y>, <a> が描画領域の外にある場合、それらはグラフの境界へ移動されます。よって、オプション xy=<x>,<y> を指定した場合の実際の塗りつぶし領域は、xrange や yrange に依存します。

3 次元滝グラフ (3D waterfall plots)

```
set style fill solid border lc "black"
splot for [scan=N:1:-1] DATA index scan \
      using x:y:z with filledcurves fc background
```

gnuplot 6.1 では、2 次元曲線の集合を直交軸に沿って順番に上書きしていくように表示するように、3 次元の **filledcurves** グラフスタイルを設計しています。通常各曲線に対して x あるいは y の一方は固定値で、それぞれの曲線を順番に増えていく y の場所に z=f(x) として表示する、あるいは順番に増えていく x の場所に z=f(y) として表示するようにします。これは、「滝グラフ」を描くのに便利です。

前面の曲線が遠い曲線を覆い隠すことを保証するためには、後ろから前への曲線の列の順番付けが重要です。

以下も参照: **fenceplots (p. 131)**。

塗り潰しの属性 (fill properties)

with filledcurves での描画は、fillstyle (solid/transparent/pattern) や fillcolor を指定することでさらにカスタマイズできます。plot コマンドで fillstyle (**fs**) を指定しなければ、現在のデフォルトの fill スタイルを使用します。以下参照: **set style fill (p. 283)**。plot コマンドで fillcolor (**fc**) を指定しなければ、現在の線色を使います。

fillstyle の属性の {{no}border} は、filledcurves のモードがデフォルトの **closed** である場合に受け付けます。例:

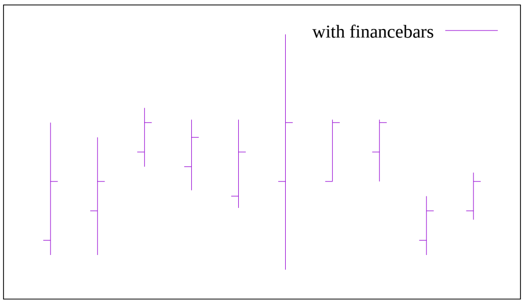
```
plot 'data' with filledcurves fc "cyan" fs solid 0.5 border lc "blue"
```

Financebars

financebars スタイルは金融データの 2 次元のデータ描画でのみ利用可能です。これは、x 座標 1 つ (通常日付) と、4 つの y 座標 (金額) を必要とします。

```
5 列:    date  open  low  high  close
```

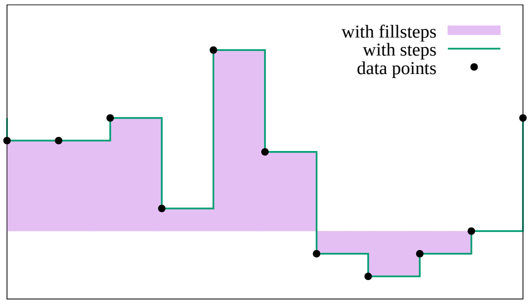
入力列を追加 (6 列目) すると、それらは各行毎の variable color 情報 (以下参照: **linecolor (p. 71)**, **rgbcolor variable (p. 73)**) として使われます。



記号は、水平方向にはその x 座標に置かれ、垂直方向には最高値 (high) と最安値 (low) を端とする線分が使われます。そして、その線分に水平左側の刻みが開始値 (open) の所に、水平右側の刻みが終り値 (close) の所につきます。その刻みの長さは **set errorbars** で変更できます。記号は最高値と最安値が入れ替わっても変わりません。以下参照: **set errorbars (p. 216)**, **candlesticks (p. 96)**。以下も参照してください。金融データデモ。

Fillsteps

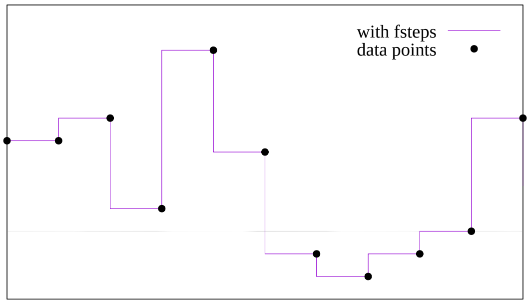
```
plot <data> with fillsteps {above|below} {y=<baseline>}
```



fillsteps スタイルは 2 次元描画でのみ利用可能です。これは、**steps** とほぼ同じですが、曲線とベースライン (デフォルトは y=0) との間の領域を現在の fillstyle で塗り潰します。オプション **above** と **below** は、ベースラインの片方の部分のみ塗り潰します。一つのデータ点から次の点への移動の際に、**steps** と **fillsteps** はいずれもまず x 座標の変化の線を引き、その後 y 座標の変化の線を引くことに注意してください。以下参照: **steps (p. 124)**。

```
1 列:    y      # 行番号 (0 列目) による暗黙の x
2 列:    x y
```

Fsteps



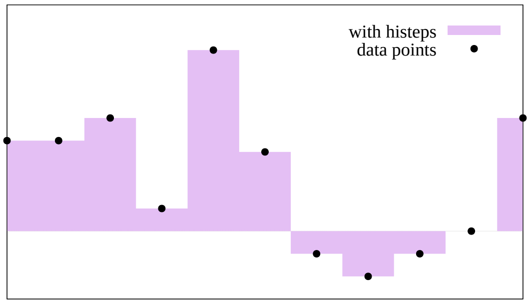
fsteps スタイルは 2 次元描画でのみ利用可能です。これは 2 本の線分で隣り合う点をつなぎます: 1 本目は (x1,y1) から (x1,y2) まで、2 本目は (x1,y2) から (x2,y2) まで。入力列の条件は、**lines** や **points** に対するものと同じです。**fsteps** と **steps** の違いは、**fsteps** は、折れ線を先に y 方向に書いてから次に x 方向に書くのに対し、**steps** は先に x 方向に書いてから次に y 方向に書きます。

以下も参照steps デモ。

```
1 列:    y      # 行番号 (0 列目) による暗黙の x
2 列:    x y
```

Histeps

histeps スタイルは 2 次元描画でのみ利用可能です。これはヒストグラムの描画での利用を意図しています。y の値は、x の値を中心に置くと考え、x1 での点は ((x0+x1)/2,y1) から ((x1+x2)/2,y1) までの水平線として表現されます。端の点では、その線はその x 座標が中心になるように延長されます。隣り合う点同士の水平線の端は、その両者の平均値のところでの鉛直線、すなわち ((x1+x2)/2,y1) から ((x1+x2)/2,y2) の線分で結ばれます。入力列の条件は、**lines** や **points** に対するものと同じです。



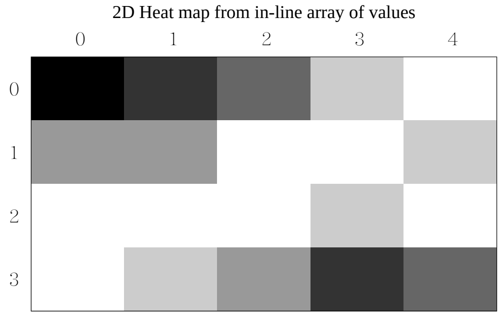
autoscale が有効である場合、x の範囲は、その延長された水平線の範囲ではなく、データ点の範囲が選択されます。よって、端の点に関してはその水平線は半分しか描かれないことになります。以下も参照`steps` デモ。

```
1 列:    y       # 行番号 (0 列目) による暗黙の x
2 列:    x y
```

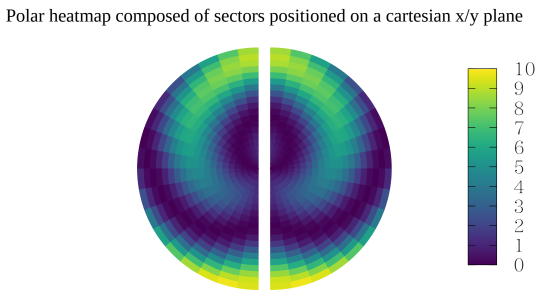
温度分布図 (heatmaps)

gnuplot のいくつかの描画スタイルで温度分布図 (heatmaps) を作ることができます。どのスタイルを使うかは、データの型で決まります。

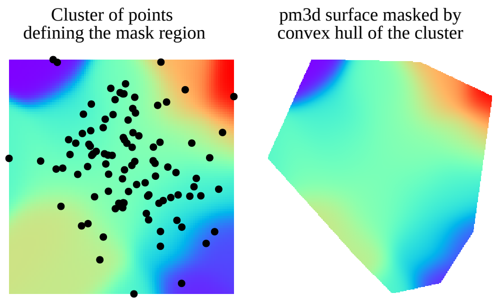
ピクセルベースの温度分布図は、すべて分布図内の各ピクセルが完全に一つの元のデータ値に対応するという特性を持っています。ピクセルベースの画像スタイルは、データ値が規則的な長方形格子であることを要求します。以下参照: **with image (p. 112)**。しかし、格子値の欠けを処理することは可能ですし (以下参照: **sparse (p. 323)**)、格子の一部を表示からマスクして消してしまうことも可能です (以下参照: **masking (p. 116)**)。格子要素がそれほど多くない場合は、各長方形要素を別々に塗って (**with image pixels**)、平滑化や非可逆圧縮が 結果の画像 ("image") に適用されないようにすることは通常はいいでしょう。



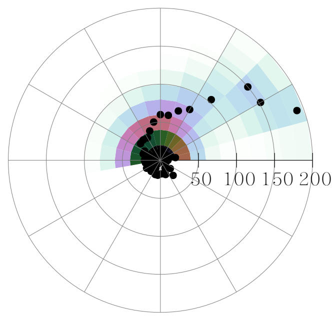
ピクセルベース温度分布図の画像と同等の極形のものは、2 次元描画スタイル **sectors** で生成できます。各入力点は、ピクセルと同等の極座標格子上の一つの環状の扇片に完全に対応します。以下に説明する極座標格子面のオプションとは違い、格子の個々の扇片は任意個提供できます。この描画スタイルを、極座標グラフ、あるいは直交座標グラフの任意の場所に、極扇片を置くように使うことができます。ここの図は、直交座標グラフ上で、温度分布図の極形の 2 つの半分を、原点を挟んで± Δ x 離れた場所に置き直したものを示します。



データ点が規則的な長方形格子を構成しない場合は、補間やスプラインを使って格子曲面に当てはめることがよく行われます。他には、点の密度関数を格子平面や滑らかな曲面に写像することもできます。以下参照: **set dgrid3d (p. 213)**。その格子曲面は、その後 pm3d 曲面として描画できます (例は以下参照: **masking (p. 116)**)。この場合、温度分布図上の点と入力点との 1 対 1 対応は保持されません。すなわち、温度分布表現の妥当性は、格子近似と同程度に過ぎません。デモコレクションに、点集合から 2 次元温度分布図を生成する例があります。`heatmap_points.dem`



あなたが使用する gnuplot が –enable-polar-grid オプション付きでビルドされた場合、極座標データ点を、2 次元極座標温度分布図を生成するのに使え、各 "ピクセル" は事前に決定したθ と r の範囲に対応します。以下参照: **set polar grid (p. 275)**, **with surface (p. 124)**。この作業は、2 次元極座標空間で操作することを除いて、丁度 **set dgird3d** と **with pm3d** を使用することの類似です。



Histograms

スタイル **histograms** は 2 次元描画でのみ有効です。こ
れは、データの各列の並びから平行な棒グラフを作ります。**plot** コマンドの各要素は、それに関する目盛りの値や凡例 (key) のタイトルが付属するかも知れませんが、単一の入力データを指定する必要があります (例えば入力ファイルの 1 つの列)。現在は、4 種類のヒストグラム形式のスタイルをサポートしています。

```
set style histogram clustered {gap <gapsize>}
set style histogram errorbars {gap <gapsize>} {<linewidth>}
set style histogram rowstacked
set style histogram columnstacked
set style histogram {title font "name,size" tc <colorspec>}
```

デフォルトのスタイルは **set style histogram clustered gap 2** に対応しています。このスタイルでは、並列に指定されたデータの値の集合は、選択されたデータ列のそのそれぞれの序列 (行番号) に対応する x 座標の場所に、各々箱のグループとして固められて置かれます。よって、<n> 個のデータ列を並列に指定した場合、最初の固まりは x=1 を中心とする <n> 個の箱の固まりからなり、その各々の高さは、その <n> データ列各々の最初 (1 行目) の値が取られます。その後に少し空白 (gap) が空けられ、次に各データ列の次 (2 行目) の値に対応する箱の固まりが x=2 を中心として置かれます。以下同様です。デフォルトの空白 (gap) 幅の 2 は、箱の固まり同士の間の空白が、箱 2 つの幅に等しいことを意味します。同じ列に対する箱は全て同じ色または同じパターンで与えられます; しかし、以下も参照: **histograms colors (p. 109)**。

箱の固まりそれぞれは、データファイルの 1 つの行から得られます。そのような入力ファイルの各行の最初の項目が見出し (ラベル) でることは良くあることです。その列にある見出し (ラベル) は、**using** に **xticlabels** オプションをつけることで、それに対応する箱の固まりの真下の x 軸に沿ったところに置くことができます。

errorbars スタイル は、各エントリに対して追加の入力列を必要とする以外は **clustered** スタイルにとても良く似ています。最初の列は、clustered スタイルの場合と全く同様に箱の高さ (y の値) として保持されます。

```
2 列:        y yerr          # 線は y-yerr から y+err へ伸びる
3 列:        y ymin ymax     # 線は ymin から ymax へ伸びる
```

誤差線の見た目は、現在の **set errorbars** の値と <linewidth> オプション指定で制御できます。

積み上げ型のヒストグラムも 2 つの形式がサポートされています。それらはコマンド **set style histogram {rowstacked|columnstacked}** で選択できます。これらのスタイルにおいて、選択された列のデータの値は積み上げられた箱として集められます。正の値は、y=0 から上の方へ積み上げられ、負の値は下へ向かって積み上げられます。正の値と負の値が混じってい場合は、上向きと下向きの両方の積み上げが生成されます。デフォルトの積み上げモードは **rowstacked** です。

スタイル **rowstacked** は、まず最初に選択された列の各行の値を x 軸のそれぞれの位置に配置します: 1 行目の値は x=1 の箱、2 行目のは x=2、以下同様となります。2 番目以降に選択された列に対応する箱は、それらの上に積み重ねられて行きます。そして結果として、x=1 にできる箱の積み重ねは、各列の最初の値 (1 行目の値) からなり、x=2 の箱の積み重ねは各列の 2 行目の値、などのようになります。同じ列に対する箱は全て同じ色または同じパターンで与えられます (以下参照: **set style fill (p. 283)**)。

スタイル **columnstacked** も同様ですが、こちらは各箱の積み上げは (各行のデータからではなく) 各列のデータからなります。最初に指定したデータ列の各行のデータが x=1 の箱の積み上げを生成し、2 番目に指定し

たデータ列の各行のデータが x=2 の箱の積み上げ、などのようになります。このスタイルでは、各箱の色は、各データ項目の (列番号ではなく) 行番号から決定されます。

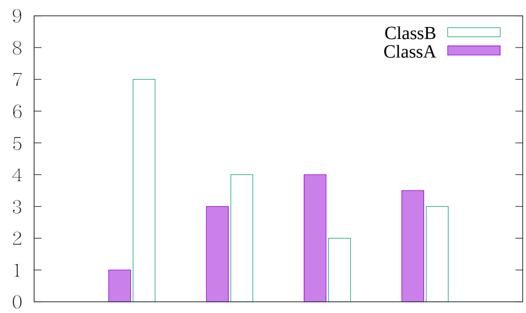
箱の幅はコマンド **set boxwidth** で変更できます。箱の塗りつぶしスタイルはコマンド **set style fill** で設定できます。

histograms は x 軸は常に x1 軸を使いますが、y 軸に関しては y1 軸か y2 軸かを選択できます。plot 命令が、histograms と他のスタイルの描画の両方を含む場合、histogram でない方は、x1 軸を使うか x2 軸を使うかを選択できます。

一つ追加のスタイルオプション **set style histogram nokeyseparators** は、複数のヒストグラムを含むグラフにのみ適切なものです。その場合の追加説明については、以下参照: **newhistogram (p. 108)**。

例:

入力ファイルは、2, 4, 6, ... の列にデータ値を持ち、3, 5, 7, ... の列に誤差評価を持つとします。以下の例は、2 列目、4 列目の値を箱の固まり型 (clustered; デフォルトスタイル) のヒストグラムとして描画します。ここでは、plot コマンドで繰り返し (iteration) を使用していますので、任意の個数のデータ列を一つのコマンドで処理できます。以下参照: **plot for (p. 183)**。

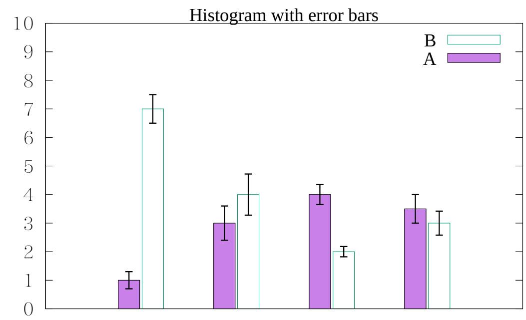



```
set boxwidth 0.9 relative
set style data histograms
set style histogram cluster
set style fill solid 1.0 border lt -1
plot for [COL=2:4:2] 'file.dat' using COL
```

これは、x 軸上の各整数値を中心とするそれぞれ 2 つの箱 (鉛直な棒) 毎の固まりによる描画を生成します。入力ファイルの最初の列にラベルが含まれているならそれを、以下の少し変更したコマンドで x 軸に沿って配置できます。

```
plot for [COL=2:4:2] 'file.dat' using COL:xticlabels(1)
```

ファイルが、各データの測定値と範囲の情報の両方を含んでいる場合、描画に誤差線を追加することができます。以下のコマンドは誤差線を (y-<error>) から (y+<error>) に引き、その頭に箱と同じ幅の水平線をつけます。誤差線と誤差線の端の線は、黒で線幅 2 で描画されます。

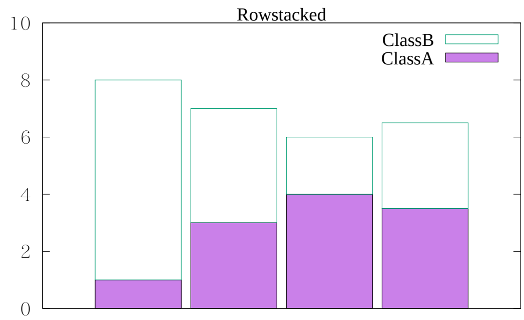



```
set errorbars fullwidth
set style fill solid 1 border lt -1
set style histogram errorbars gap 2 lw 2
plot for [COL=2:4:2] 'file.dat' using COL:COL+1
```

以下は、同じデータを行毎の積み上げ型 (rowstacked) のヒストグラムで描画する方法を示しています。これまでとは違い、以下の例の描画コマンドでは、繰り返しの使用ではなく、個別に別々の列を指定しています。

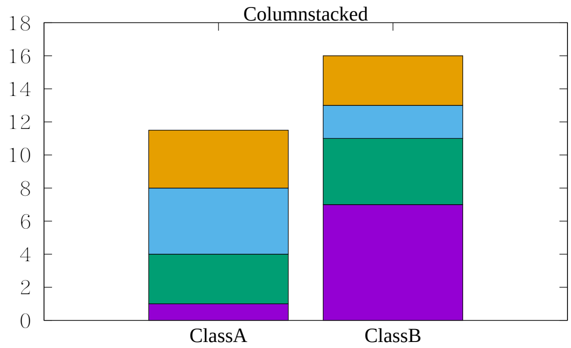
```
set style histogram rowstacked
plot 'file.dat' using 2, '' using 4:xtic(1)
```

これは、一つ一つの鉛直な棒が、データの一つの列に対応する描画を生成します。各棒は、2 つの部分の積み上げの形であり、それぞれの部分の高さが、データファイルの 2 列目と 4 列目の値に対応します。最後に以下のコマンド




```
set style histogram columnstacked
plot 'file.dat' using 2, '' using 4
```

は、一つ一つがそれぞれデータ列に対応する、2 つの鉛直な積み重ねの棒を生成します。x=1 にある棒は、データファイルの 2 列目の各行の値に対応する箱からなります。x=2 にある棒は、データファイルの 4 列目の各行の値に対応する箱からなります。



これは、gnuplot の通常の入力の縦、横の解釈を入れ換えることになりますので、凡例のタイトルや x 軸の目盛りの見出しの指定も変更する必要があります。以下のコメント部分を参照してください。

```
set style histogram columnstacked
plot '' u 5:key(1)            # 1 列目を凡例タイトルに使用
plot '' u 5 title columnhead  #
```

この 2 つの例は、全く同じデータ値を与えているのですが、異なる書式であることに注意してください。

Newhistogram

書式:

```
newhistogram {"<title>" {font "name,size"} {tc <colorspec>}}
             {lt <linetype>} {fs <fillstyle>} {at <x-coord>}
```

一回の描画に 2 つ以上のヒストグラムの組を作ることもできます。この場合コマンド **newhistogram** を使うことで、それらを強制的に分離し、またそれぞれのラベルを分離することができます。例:

```
set style histogram  cluster
plot newhistogram "Set A", 'a' using 1, '' using 2, '' using 3, \
     newhistogram "Set B", 'b' using 1, '' using 2, '' using 3
```

ラベル "Set A" と "Set B" は、それぞれのヒストグラムの組の下、x 軸の全てのラベルの下の位置に現われます。

コマンド newhistogram は、ヒストグラムの色付けを強制的に指定した色 (linetype) で始めるのにも使えます。デフォルトでは、色の番号はヒストグラムの境界をまたいでさえも連続的に増加し続けます。次の例は、複数のヒストグラムに同じ色付けを施します。

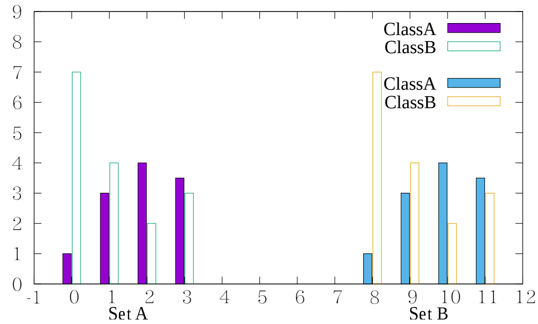
```
plot newhistogram "Set A" lt 4, 'a' using 1, '' using 2, '' using 3, \
     newhistogram "Set B" lt 4, 'b' using 1, '' using 2, '' using 3
```

同様に、次のヒストグラムを指定した fillstyle で始めさせることが可能です。その fillstyle を **pattern** にセットした場合、塗り潰しに使用されるパターン番号は自動的に増加されていきます。

新しいヒストラムの開始は、通常は凡例 (key) に空のエントリを追加し、それによりそのヒストグラム要素の集合のタイトルは、それ以前のヒストグラムのものと分離されることになります。しかしこの空行は、その要素が個々のタイトルを持たない場合は望ましいものではありません。これは、そのスタイルを **set style histogram nokeyseparators** と変更することで、その空行の出力を抑制できます。

オプション **at <x-coord>** は、その後のヒストグラムの x 座標の位置を <x-coord> に設定します。例:

```
set style histogram cluster
set style data histogram
set style fill solid 1.0 border -1
set xtic 1 offset character 0,0.3
plot newhistogram "Set A", \
     'file.dat' u 1 t 1, '' u 2 t 2, \
     newhistogram "Set B" at 8, \
     'file.dat' u 2 t 2, '' u 2 t 2
```



この場合、2 つ目のヒストグラムの位置は x=8 から始まります。

複数の列に渡る自動的な繰り返し (automated)

一つのデータファイルのたくさんの列から、一つのヒストグラムを生成したい場合、plot の繰り返し (iteration) 機能を使うと便利でしょう。以下参照: **plot for (p. 183)**。例えば、3 列目から 8 列目までのデータを積み上げた形のヒストグラムを生成する例:

```
set style histogram columnstacked
plot for [i=3:8] "datafile" using i title columnhead
```

ヒストグラムの色の割り当て (histogram color assignments)

gnuplot は、ヒストグラム内の各要素の箱に自動的に色を割り当てますが、同値なデータには、それらがヒストグラムの行、または列にどこに現れたとしても首尾一貫した色を保持するようにします。色は、連続する線種 (linetype) の、まだ使用されていない次の線種、または **newhistogram** 命令の提示によって初期化される最初の線種のいずれかから始まるものが使われます。

この仕組みは、データソースが真に並列ではない (すなわち、いくつかのファイルのデータが不完全) ために失敗することがあります。または、基本色の光度や彩度を変えることにより視覚化するためのデータの追加属性を与えたいかもしれません。自動的な色の割り当ての代わりに、各データ用の明示的な色の値を、2 番目の **using** 列で、**linecolor variable** か **rgb variable** の仕組みによって与えることができます。以下参照: **colorspec (p. 71)**。あなたのデータのレイアウトによって、色のカテゴリは、行のヘッダか、列のヘッダか一つのデータ列でありえます。多分、凡例 (key) のサンプルの色は、それに合うようにカスタマイズする必要があることに注意してください (以下参照: **keyentry (p. 230)**)。

例: file_001.dat から file_008.dat は、1 列目はカテゴリ識別子の A, B, C, ... で、2 列目がデータ値になっています。すべてのファイルがすべてのカテゴリに対する行を持っているとは限らず、よって完全な並列データ群ではありません。この場合、gnuplot は、間違って各ファイルの N 行目の値に同じ色を割り当てようとしてしまいます。その代わりに、1 列目のカテゴリに基づいた色を割り当てる例です。

```
file(i) = sprintf("file_%03d.dat",i)
array Category = ["A", "B", "C", "D", "E", "F"]
color(c) = index(Category, strcol(c))
set style data histogram
plot for [i=1:8] file(i) using 2:(color(1)) linecolor variable
```

さらに key のカスタマイズの生成も含む完全な例が、デモの中のhistogram_colors.dem

にあります。

Hsteps

2 次元描画スタイル **with hsteps** は、各データ点に対し一つの水平線分 ("step" (段)) を描画します。この段は、データ点の x 座標の左側、あるいは右側、あるいは両側へ延ばすことができます。追加キーワードで、隣接する段と接続する線や、段とベースラインとなる y の値との間を塗り潰すためのオプションを制御します。

書式:

```
plot <data> with hsteps
                    {forward|backward}
                    {baseline|pillars|link|nolink}
                    {{above|below} y=<baseline>}
                    {offset <y-offset>}

2 列:  x  y
3 列:  x  y  width
```

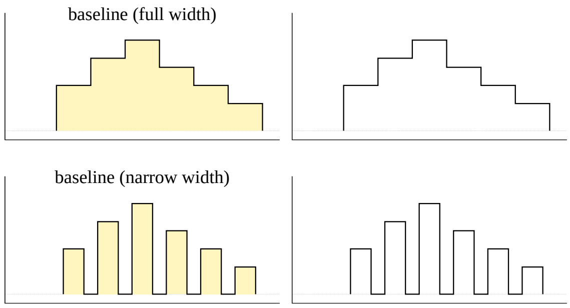
この描画スタイルには 2 列か 3 列のデータが必要です。さらに入力列を追加すると、それは可変線色や塗り潰し色の情報として使用します (以下参照: **rgbcolor variable (p. 73)**)。入力データの x の値は、単調であると仮定します。

各段の幅を 3 列目の入力列として明示的には与えなかった場合、各線分の幅は、隣接する水平線分と接触するように計算します。3 列目が負の値の場合は全幅の段の要求として処理します。

キーワード **forward**, **backward** は、指定した x 座標から水平線分を延長する方向を指定するのに使用します。どちらも指定しない場合は水平線分は与えられた x の値から両側へ、隣接する次の点の x の値との中点まで延長します。しかし、始点と終点では、対応する外側の隣接点がないので、水平線分は内側の隣接点との距離を利用して外側への延長幅を外挿します (以下参照: **histeps (p. 104)**, **boxes (p. 92)**)。

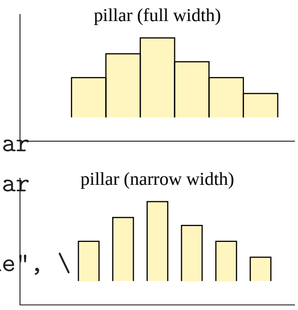
デフォルト (**baseline**) と、その変種の **pillar** は、ベースラインとなる y の値を使用します。plot コマンドで何も指定しなければ、ベースラインは y=0 となります。plot コマンドが塗り潰しスタイルを使用する場合、ベースラインは塗り潰し領域の一つの境界を定めることにもなります。4 つの変種が利用可能です。

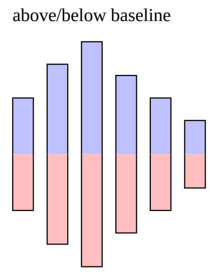
baseline (デフォルト): x 方向に隣接する段との間に隙間がない場合、それらを垂直線分で接続します。これは、スタイル **steps**, **histeps**, **fsteps** のような曲線を生成します。段の間に隙間がある場合、通常は点の隙間よりも段の幅が小さいので、それらを接続する線は一旦ベースラインまで落とし、そしてその上を進んでから再び上に上がります。これは、長方形パルスの列を生成します。




```
set xzeroaxis
plot $data using 1:2 with hsteps
plot $data using 1:2:(0.5) with hsteps
```

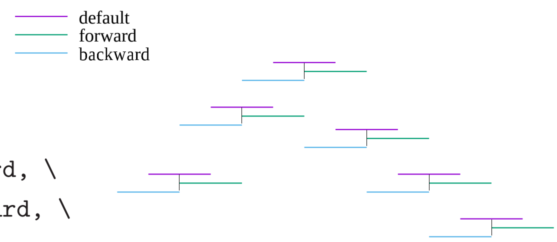
pillar: 各段の端で、ベースラインへの垂直線を描きます。ベースライン上には水平線分を一切描かないことに注意してください。






```
plot $data using 1:2        with hsteps pillar
plot $data using 1:2:(0.5) with hsteps pillar
plot $data using 1:2:(0.5) \
          with hsteps pillar above fc "blue", \
     $data using 1:2:(0.5) \
          with hsteps pillar below fc "red"
```

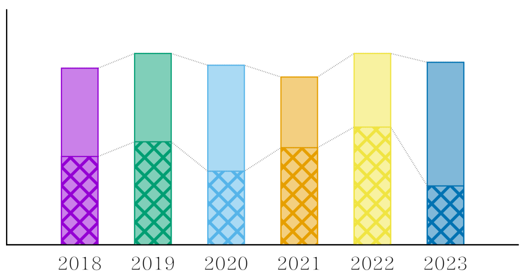
nolink: 隣接する段の間の線は何も描きません。この変種に対しては、ベースラインと塗り潰しの属性とは何の関連もありません。




```
plot $data using 1:2 with hsteps nolink, \
     $data using 1:2 with hsteps nolink forward, \
     $data using 1:2 with hsteps nolink backward, \
     $data using 1:2 with points pt "|"
```

link: 隣接する段を、一本の真っ直ぐな線分で接続します。段の幅によっては、その線は垂直でなくむしろ斜めになります。

例: 変種 **link** を変種 **pillar** に重ねることで、カテゴリの境界同士を線で接続するような積み上げ型ヒストグラムグラフを生成できます。

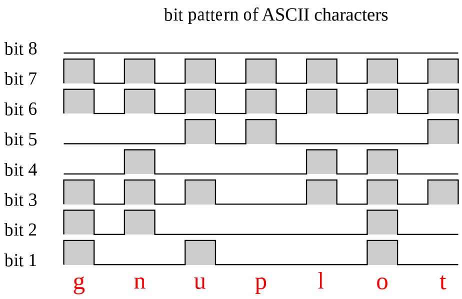



```
set style line 11 lw 2 lc "gray" dt "."
set style line 12 lw 2 lc variable
plot $data using 1:3:(0.5)   ls 11 with hsteps link, \
```

```
$data using 1:3:(0.5):1 ls 12 with hsteps pillar \
  fs solid 0.7 border, \
$data using 1:4:(0.5)   ls 11 with hsteps link, \
$data using 1:4:(0.5):1 ls 12 with hsteps pillar \
  fs transparent pattern 1 border
```

Offset

オフセット値は、データ点それ自身 (2 列目) とグラフに現れるベースラインの両方に y の値に増分を追加することで、**with hsteps** の任意の変種を修正します。例えば、論理回路のタイミングチャートを描くときに、パルス波形を垂直方向に位置合わせするのに使えます。一般には、y の値の範囲を共有する複数のデータ集合の積み上げグラフにオフセットが利用できます。




```
# bit(k,char) は、アスキー文字の k bit の状態に応じて
# 0 か 1 を返す関数
set style fill solid 0.2 border
plot for [k=1:8] STR using 1:(bit(k,STR[$1])):(0.5) \
     with hsteps fillcolor "black" offset k
```

Missing data

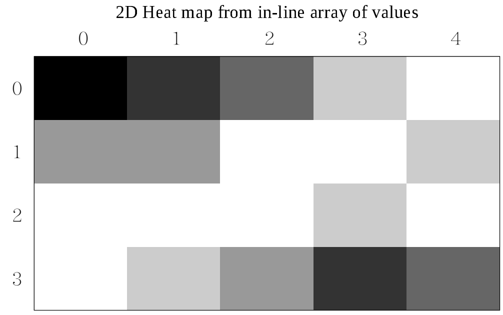
描画スタイル hstep では、空行、NaN 値、欠損データは異なる意味を持ちます。データに空行がある場合は、データの連続列をその点でリセットします。これは、**with lines** の場合に空行が新しい曲線の開始を中断するのと同様です。x の値が NaN の場合、それは空行の場合と同じ処理を行います。x の値が有効で、y の値が NaN の場合は、そのデータ点には水平線は描きませんが、x の値は、段の幅の評価で必要な場合に使用します。

Image

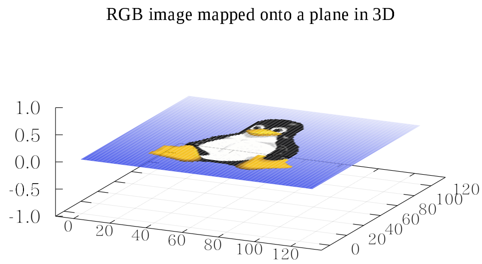
描画スタイル **image**, **rgbimage**, **rgbalpha** は、いずれも一様に標本点を取った格子状データ値を、2 次元、または 3 次元中のある平面上に射影します。入力データは、既にあるビッマップ画像ファイル (PNG のような標準的なフォーマットから変換したものでよい) か、単純な数値配列です。これらの描画スタイルは、温度分布図 (heatmap) を作るのによく使われます。極座標での 2 次元温度分布図については、以下参照: **set polar grid (p. 275)**。

この図は、スカラー値の配列から温度分布を生成した例です (訳注: 図が表示されている場合)。現在のパレットを、各スカラー値から対応するピクセルの色への割り当てに使用します。以下も参照: **sparse** (p. 323)。

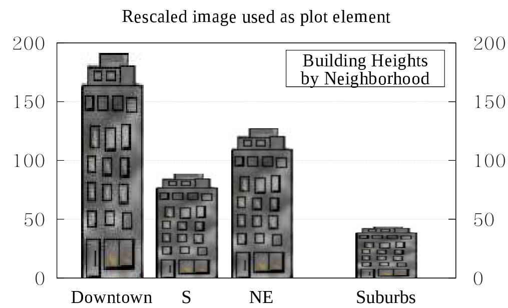
```
plot '-' matrix with image
5 4 3 1 0
2 2 0 0 1
0 0 0 1 0
0 1 2 4 3
e
e
```



入力 2 次元画像の各ピクセル (データ点) は、描画グラフ中では長方形、または平行六面体となります。画像の各データ点の座標は、平行六面体の中心を決定します。すなわち、M x N 個のデータ集合は M x N ピクセルの画像を生成します。これは、M x N 個のデータ集合が (M-1) x (N-1) 要素を作成する pm3d の構造とは異なります。バイナリ画像データの格子の走査方向は、追加キーワードでさらに制御可能です。以下参照: **binary keywords flipx** (p. 161), **keywords center** (p. 161), **keywords rotate** (p. 161)。



各ピクセルの x と y の大きさを示すことで、画像データを 2 次元描画座標系内の特定の長方形に収まるように伸縮することができます。以下参照: **binary keywords dx** (p. 160), **dy** (p. 160)。右の画像を生成するのには、同じ入力画像を、それぞれ dx, dy, origin を指定して複数回配置しました。入力 PNG 画像であるビルの絵は 50x128 ピクセルです。高いビルは、**dx=0.5 dy=1.5** で割り当てて描画し、低いビルは、**dx=0.5 dy=0.35** としています (訳注: 図が表示されている場合)。



スタイル **image** は、グレイスケール (灰色階調)、またはカラーパレット値を含んでいるピクセルの入力を処理します。よって 2 次元描画 (**plot** コマンド) では 3 列のデータ (x,y,value) を、3 次元描画 (**splot** コマンド) では 4 列のデータ (x,y,z,value) が必要になります。

スタイル **rgbimage** は、赤、緑、青の 3 つの色成分 (RGB) で記述されたピクセルの入力を処理します。よって **plot** では 5 次元データ (x,y,r,g,b) が、**splot** では 6 次元データ (x,y,z,r,g,b) が必要になります。赤、緑、青の各成分は [0:255] の範囲内にあると仮定されます。これは、PNG や JPEG ファイルで使用されている仕組みに合っています (以下参照: **binary filetype** (p. 159))。しかし、RGB 成分として [0:1] の範囲の実数値を取る仕組みを使用するようなデータファイルも中にはあります。そのようなデータで **rgbimage** スタイルを使用するには、まず **set rgbmax 1.0** としてください。

スタイル **rgbalpha** は、赤、緑、青の RGB 成分に加えて、アルファ値 (透過パラメータ) の情報も含んだピクセルの入力を処理します。よって、**plot** では 6 次元データ (x,y,r,g,b,a) が、**splot** では 7 次元データ

(x,y,z,r,g,b,a) が必要になります。赤、緑、青、およびアルファの各成分は [0:255] の範囲内にあると仮定されます。RGBA 成分が [0:1] の範囲の実数値であるデータを描画するには、まず **set rgbmax 1.0** としてください。

rgbimage か rgbalpha のいずれかでの描画で色要素用にデータ 1 列しか指定しなかった場合は、それは、alpha=0 は不透明、alpha=255 が完全な透明を意味する、32 bit パックの ARGB データであるとみなします。このアルファ値の見方は、アルファ値が別の列で与えられる場合は古臭い慣習ですが、しかし、色を設定する必要のある個々のコマンド用の ARGB パックの仕組みには合っています。

透明化 (transparency)

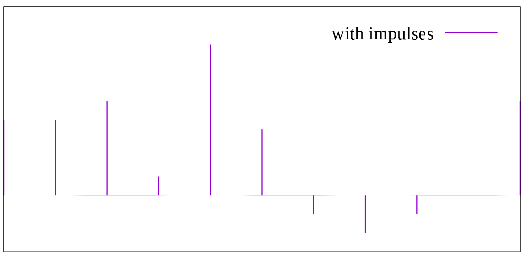
描画スタイル **rgbalpha** は、入力データの各ピクセルが [0:255] の範囲内のアルファ値を持っている必要があります。alpha = 0 のピクセルは完全な透明で、その下 (奥) の描画要素を全く変えません。alpha = 255 のピクセルは完全な不透明です。すべての出力形式は、これら 2 つの両極端な値をサポートします。$0 <$ alpha $<$ 255 のピクセルは半透明で、半透明をサポートしていない出力形式では、その値を 0 か 255 のいずれかに丸めます。

Image pixels

出力形式によっては、2 次元の長方形領域内での画像データ描画の、デバイスやライブラリに依存した最適化ルーチンを使用しますが、これは、中間ピクセル平滑化を行ったり、クリッピングがうまくなかったり、縁が欠けるなど、望ましくない出力を生成することがあります。例として、SVG 画像レンダリング時のウェブブラウザによる平滑化があります。キーワード **pixels** は、画像を 1 ピクセルずつ描画するような一般的なコードを使用するよう gnuplot に指示します。この描画モードでは、描画は遅く、大きな出力ファイルを生成しますが、どの出力形式でも共通的な見た目を作成してくれるます。これは、特にピクセル数の少ない温度分布図 (heatmap) で好まれるでしょう。例:

```
plot 'data' with image pixels
```

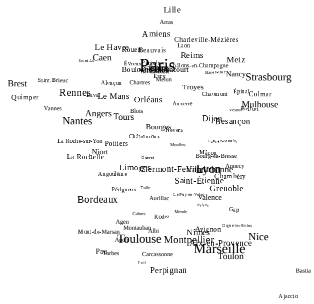
Impulses



impulses スタイルは、2 次元描画では y=0 から各点の y の値への、3 次元描画では z=0 から各点の z の値への、垂直な線分を表示します。y や z の値は負の値でもよいことに注意してください。データの追加列を各垂直線分の色の制御に利用できます。このスタイルを 3 次元描画で使用する場合、太い線 (linewidth $>$ 1) を利用するとより効果的でしょう。それは 3 次元の棒グラフに似たものになります。

```
1 列:  y
2 列:  x  y     # [x,0] から [x,y] への線 (2D)
3 列:  x  y  z  # [x,y,0] から [x,y,z] への線 (3D)
```

Labels



スタイル labels は、データファイルから座標と文字列を読み込み、その文字列をその 2 次元、または 3 次元座標に置きます。これは基本的に 3 列、または 4 列の入力を必要とします。さらに余分な入力列は、文字列の回転角 (キーワード **rotate variable**) や文字色 (以下参照: **textcolor variable** (p. 72)) のような、点毎に変動する属性値が指定されたとみなされます。

```
3 列:  x  y  string    # 2 次元版
4 列:  x  y  z  string # 3 次元版
```

フォント、色、回転角やその他の描画テキストの属性は追加オプションとして指定可能です (以下参照: **set label** (p. 235))。次の例は、入力ファイルの 1 列目から取った市の名前から作られる文字列を、4, 5 列目から取った地図座標に描画します。フォントサイズは、3 列目の値から計算していて、この場合はそれは人口を意味しています。

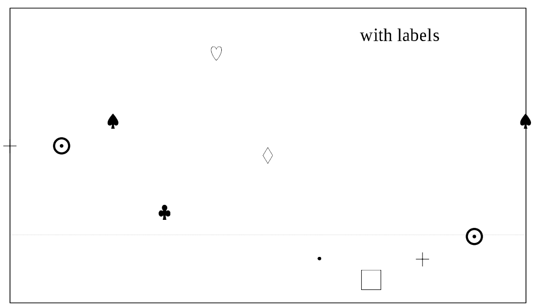
```
CityName(String,Size) = sprintf("{/=%d %s}", Scale(Size), String)
plot 'cities.dat' using 5:4:(CityName(stringcolumn(1),$3)) with labels
```

フォントサイズを、個々の市の名前に対して異なるサイズに合わせなくていいならば、コマンドはもっと簡単です:

```
plot 'cities.dat' using 5:4:1 with labels font "Times,8"
```

labels に **hypertext** がついている場合、その文字列はマウスがそれに対応する点の上に来たときにだけ現われます。以下参照: **hypertext** (p. 237)。この場合ハイパーテキストの置き場所として機能する点を作るためにそのラベルの **point** 属性を有効にする必要があります:

```
plot 'cities.dat' using 5:4:1 with labels hypertext point pt 7
```



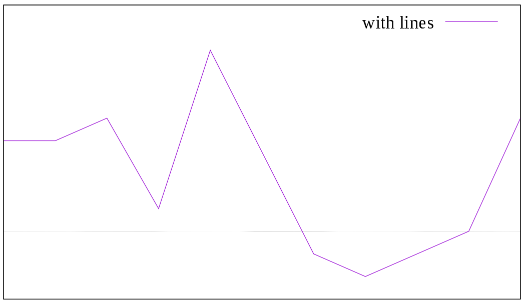
スタイル **points** であらかじめ定義されている点の記号が適切でない、あるいは十分でない場合、その代わりとしてスタイル **labels** を使うこともできます。例えば、以下は単一文字として選択される組を定義し、3 列目のデータ値に対応するその一つをグラフの各点に割り当てる例です (訳注: 以下のサンプルの <UTF-8 文字列> の部分には、元々、丸に中点記号や□ 、+、トランプ記号などの UTF-8 文字列が並んでいますが、この日本語訳とは両立しないため取り除いています):

```
set encoding utf8
symbol(z) = "<UTF-8 文字列>"[int(z):int(z)]
splot 'file' using 1:2:(symbol($3)) with labels
```

以下は、4 列目の値を可変値の回転角とし、5 列目の値を文字色 ("tc") とするラベルの使用例です。可変値の色指定は、常に **using** 指定の最後の列から取ることに注意してください。

```
plot $Data using 1:2:3:4:5 with labels tc variable rotate variable
```

Lines



lines スタイルは隣接する点を真直な線分で結びます。これは、2 次元描画でも、3 次元描画でも使用でき、基本的には、1 列か 2 列か 3 列かの入力データを必要とします。余分な入力列は、線の色の変更などの情報が提供されたものとして使用されます (以下参照: **rgbcolor variable (p. 73)**)。

2 次元 ("using" 指定なし) の場合

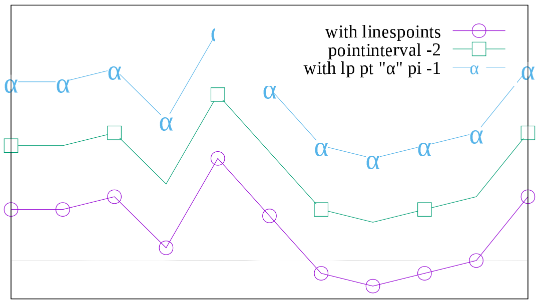
```
1 列:  y      # 行番号による暗黙の x
2 列:  x y
```

3 次元 ("using" 指定なし) の場合

```
1 列:  z      # x は暗黙の行番号、y は index から
3 列:  x y z
```

以下も参照: **linetypes (p. 70)**, **linewidth (p. 285)**, **linestyle (p. 285)**。

Linespoints



linespoints スタイル (省略形 **lp**) は、隣接する点を真っ直ぐな線分で結び、その後で最初に戻って各点に小さな記号を描きます。点記号は、**set pointsize** で決まるデフォルトの大きさで描きますが、plot コマンド上で点のサイズを指定したり、あるいは入力データの追加列で個別の点サイズを指定することもできます。追加の入力列は、個別の線の色などの情報を提供するのにも使われます。以下参照: **lines (p. 116)**, **points (p. 118)**。

グラフのすべての点に記号の印をつけるか否かを制御する 2 つのキーワード **pointinterval** (省略形 **pi**), **pointnumber** (省略形 **pn**) があります。

pi N あるいは **pi -N** は、N 個毎に 1 つだけ記号を置くよう gnuplot に指示します。負の値を指定すると、記号の下の線分の部分を消します。その消す部分のサイズは **set pointintervalbox** で制御できます。

pn N あるいは **pn -N** は、データ点のうち N 個だけラベル付けするよう gnuplot に指示します。点はそのデータ全体に渡って均等な間隔に取ります。**pi** 同様、負の値を指定すると、記号の下の線分の部分を消します。

マスキング (masking)

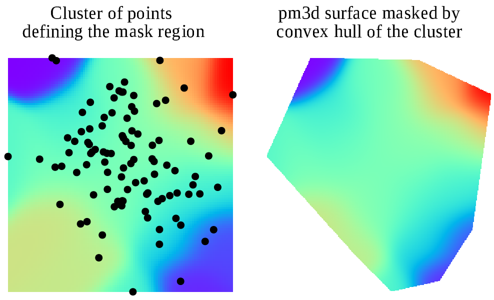
描画スタイル **with mask** は、マスキング領域を定義するのに使います。これは、同じ **plot**, **splot** コマンド上で後で指定した pm3d 曲面や画像に適用できます。入力データは、1 つ、または複数の多角形の頂点を定義する [x,y] 座標列、または [x,y,z] 座標列として解釈します。描画スタイル **with polygons** と同様に、多角形

は空行で分離します。マスクが 3 次元描画コマンド (splot) の一部である場合、入力には z の値が必要ですが、しかしそれは現在のところ全く使いません。

plot コマンド上にマスク定義がある場合、その後の同じコマンド上の image 描画、pm3d 曲面が、キーワード **mask** を追加することでマスクされます。マスクを定義していなければ、そのキーワードは無視します。

以下の例は、点の集合を囲む凸包を使い、ある pm3d 曲面に対応する領域をマスクする方法を示します。

```
set table $HULL
plot $POINTS using 1:2 convexhull
unset table

set view map
set multiplot layout 1,2
splot $POINTS using 1:2:3 with pm3d, \
      $POINTS using 1:2:(0) nogrid with points
splot $HULL using 1:2:(0) with mask, \
      $POINTS using 1:2:3 mask with pm3d
unset multiplot
```



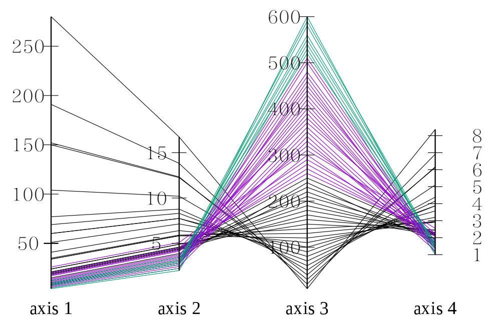
最初のパネルに対するコマンド **splot** は、元の点から dgrid3d で生成したマスクされていない曲面を描画し、その後で点自身を順に描画します。2 つ目のパネルに対するコマンド **splot** はマスクした曲面を描画します。マスクの定義 (**with mask** での plot) は最初に行わなければならず、それを適用する pm3d 曲面はその後であることに注意してください (描画スタイル **with pm3d** にキーワード **mask** をつけて変更)。この例のより完全な版は、デモ集合`mask_pm3d.dem`

内にあります。

ここには示しませんが、一つのマスクが複数の多角形領域を含み得ます。

Parallelaxes

平行座標描画 (parallel axis plot, parallel coordinates plot とも呼ばれる) は、多次元データの相関を視覚化します。入力データの個々の列は、それぞれ別々のスケールの縦軸に割り当てられます。一つのファイルのすべての列を描画する場合は、グラフの折れ線 1 本が、そのファイルのデータの 1 行分の値を表しています。それらを数種類に分類して色を割り当てることはよく行われますが、それは、その分類と軸との間の関係を視覚的に調査することを可能にします。



書式:

```
set style data parallelaxes
plot $DATA using col1{:varcol1} {at <xpos>} {<line properties}, \
      $DATA using col2, ...
```

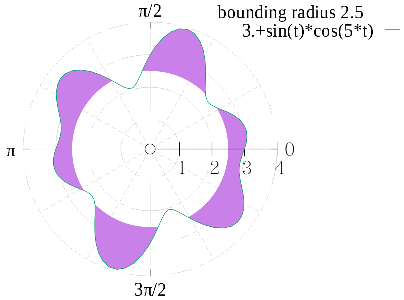
キーワード **at** で、以下の例に見られるように、平行座標軸の x 軸位置での明示的な配置も可能になっています。明示的な x 座標指定がなければ、N 番目の軸は x=N の場所に置かれます。

```
array xpos[5] = [1, 5, 6, 7, 11, 12]
plot for [col=1:5] $DATA using col with parallelaxes at xpos[col]
```

デフォルトでは、gnuplot は自動的に個々の軸の範囲、スケールを入力データから決定しますが、通常の **set axis range** コマンドによってそれをカスタマイズすることも可能です。以下参照: **set paxis (p. 265)**。

極座標描画 (Polar plots)

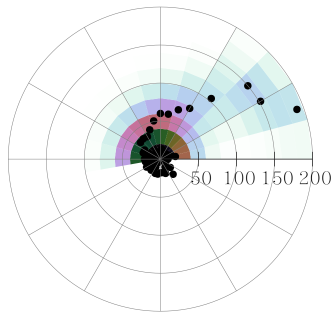
Polar plots



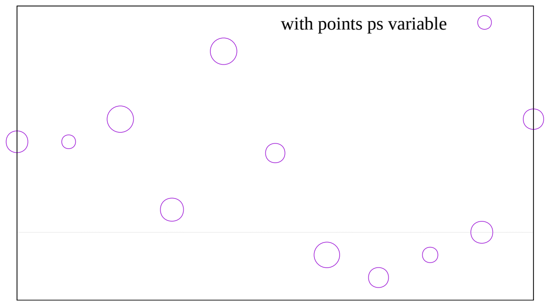
極座標描画 (polar) は、plot コマンドを入力する前に現在の座標系を極座標に変更することによって生成します。オプション **set polar** は、入力する 2 次元座標を <x>,<y> の代わりに < 角 >,< 半径 > と解釈することを gnuplot に指示します。すべてではないですが、多くの 2 次元描画スタイルが極座標モードでも機能します。図は、描画スタイル **lines** と **filledcurves** の組み合わせを示しています。(訳注: 図が表示されている場合) 以下参照: **set polar (p. 274)**, **set rrange (p. 277)**, **set size square (p. 278)**, **set theta (p. 292)**, **set ttics (p. 297)**。

極座標での温度分布_図は、スタイル **with surface** と **set polar grid** を合わせて使用することで生成できます。

```
set size square
set angle degrees
set rtics
set grid polar
set palette cubehelix negative gamma 0.8
set polar grid gauss kdensity scale 35
set polar grid theta [0:190]
plot DATA with surface, DATA with points pt 7
```



Points



points スタイルは各点に小さな記号を表示します。すべての記号のデフォルトの大きさを変更するにはコマンド **set pointsize** を使います。デフォルトの点種は、線種と同じものになります。以下参照: **linetypes (p. 70)**。plot コマンドに **using** 指定がない場合は、入力データ列を暗黙のうちに、以下で説明するように **x y pointsize pointtype color** の順で解釈します。

最初の 8 つの点種は、すべての出力形式で共通ですが、より多くの点種を個別にサポートする出力形式もあります。現在の出力形式の設定でどのような点種が用意されているかを見るには、コマンド **test** を使用してください。

また、下の例のようにして、任意の印字可能文字を点種番号の代わりに使用することもできます。点種として、任意の UTF-8 文字を使用できます (utf8 をサポートしている必要あり)。以下参照: **escape sequences (p. 44)**。より長い文字列は、描画スタイル **points** ではなく **labels** を使えば出力できます。

```
plot f(x) with points pt "#"
plot d(x) with points pt "\U+2299"
```

可変点属性 (variable point properties)

点 (point) 記号を含む描画スタイルでは、オプションとして追加データ列を **using** 指定で受け取ることでその点の見た目を制御することができます。これは、plot コマンド上でキーワード **pointtype**, **pointsize**, **linecolor** を使う際に、番号の代わりにキーワード **variable** を追加することで指示します。描画スタイル **with labels** でも、可変文字列回転角を受け取ることができます。例:

```
# 入力データは、列 1:2 で [x,y] を与え
# 点のサイズは 5 列目で与え
# RGB 色は 4 列目で 16 進数値として与え
# すべての点は pointtype 7 を使用
  plot DATA using 1:2:5:4 with points lc rgb variable ps variable pt 7
```

2 つ以上の可変 (variable) 属性を指定すると、plot コマンド内のキーワードの順とは無関係に、以下の順で列を解釈します。

```
textrotation : pointsize : pointtype : color
(文字列の回転角) : (点のサイズ) : (点の種類) : (色)
```

よって、上の例では、"lc rgb variable" は plot コマンドの最初に現れていますが、色は **using** 指定の最後の列 (4) から取ります。可変色 (variable color) 指定は、常に最後の追加列から取ります。可変色を指定するには、いくつかの方法があります。以下参照: **colorspec (p. 71)**。

注意: 「ユーザ定義変数」の方の variable に関する情報については、以下参照: **variables (p. 63)**。

Polygons

2 次元グラフ:

```
plot DATA {using 1:2} with polygons
```

plot with polygons は、**plot with filledcurves closed** として処理しますが、各多角形の境界を閉曲線として描画する場合を除きます。それは、そのその最初と最後の点が同じでなくてもです。境界線の線種は、塗り潰しスタイル (fill style) から取ります。入力データファイルには、単一の空行で区切って複数の多角形を入れることも可能です。各多角形には、3 列目の値を与え、キーワード **fc variable** を指定する (その値を linetype と解釈) か、**fc rgb variable** を指定する (その値を 24 ビット RGB 色と解釈) ことで、個別の塗り潰し色を割り当てることができます。多角形の最初の頂点の色の値のみを使用します。

3 次元グラフ:

```
splot DATA {using x:y:z} with polygons
      {fillstyle <fillstyle spec>}
      {fillcolor <colorspec>}
```

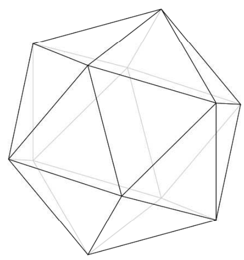
splot with polygons は、3 次元空間の個々の三角形、四角形、およびそれ以上の多角形を描画するのに pm3d を使用します。これらは 3 次元曲面の一面や単独の形状となり得ます。このルーチンは、頂点が一つの平面に乗っていなければいけません。個々の多角形を定義する頂点は、入力ファイルの連続する行から読み込みます。空行は多角形同士を分離します。

塗り潰しスタイルと色は、splot コマンド上で指定できますが、そうでなければ大域的な塗り潰しスタイルの **set style fill** の設定を使用します。pm3d ルーチンの制限のため、**set pm3d border** による一つの境界線スタイルが、**splot with polygons** のすべての多角形に適用されますが、この制限は将来の版の gnuplot では緩和されるでしょう。

各多角形には、4 列目の値を与え、キーワード **fc variable** を指定する (その値を linetype と解釈) か、**fc rgb variable** を指定する (その値を 24 ビットの塗り潰し色と解釈) ことで、個別の色を割り当てることができます。多角形の最初の頂点の色の値のみが使用されます。

面には、pm3d のソート順と光源モデルを適用しますので、常に、**set pm3d depthorder** を使う方がいいかもしれせん。

```
set xyplane at 0
set view equal xyz
unset border
unset tics
set pm3d depth
set pm3d border lc "black" lw 1.5
splot 'icosahedron.dat' with polygons \
      fs transparent solid 0.8 fc bgnd
```



Rgbalpha

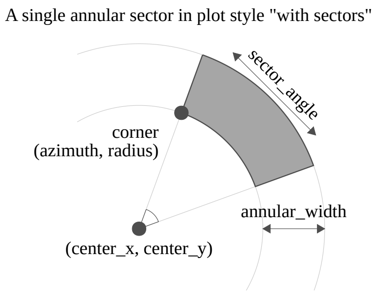
以下参照: **image (p. 112)**。

Rgbimage

以下参照: **image (p. 112)**。

扇片 (sectors)

2 次元描画スタイル **with sectors** は、入力データの 1 行毎に一つの環状の扇片 ("sector") を描画します。各扇片の形は、データ値として要求する 4 つの値で決定します。この扇片の環の部分の原点を、データ値に追加指定することもできます。扇片毎の色を追加列として指定することもできます。



この描画スタイルは、直交座標で、または極座標モード (**set polar**) の元で使用できます。方位角と扇の中心角の単位、解釈は、**set angles** と **set theta** で制御します。

データの 1,2 列目は、その扇片の一つの角の方位 (theta) と半径 (r) です。
データの 3,4 列目は、その扇片の方位の変位 (中心角; sector_angle) と半径の変位 (動径方向の幅; annular_width) です。
データの 5,6 列目を指定した場合は、それは扇形の中心の座標を意味し (デフォルトは [0,0])、直交座標では [x,y]、極座標モードでは [theta,r] と解釈します。

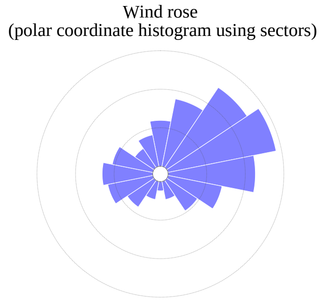
書式:

```
plot DATA {using specifier} {units xy | units xx | units yy}
```

using 指定

```
4 列: azimuth radius sector_angle annular_width
5 列: azimuth radius sector_angle annular_width color
6 列: azimuth radius sector_angle annular_width center_x center_y
7 列: azimuth radius sector_angle annular_width center_x center_y
        color
```

x 軸と y 軸スケールが等しくない場合、x,y 座標の環状の全体の概形は円ではなく楕円となって現れることに注意してください。等しくない軸のスケールに関して楕円 (ellipse) と同じ仕組みを使用して、環状の概形、および扇片の幅の外見を正しく合わせることができます。コマンドラインに **units xx** を追加すると、現在の x 軸のスケールを x, y 両方に等しく適用したかのように扇片を描きます。同様に、**units yy** は、現在の y 軸のスケールを x,y 両方に等しく適用したかのように扇片を描きます。以下参照: **set isotropic (p. 228)**, **set style ellipse (p. 287)**。



with sectors での描画は、直交座標での描画スタイルである **boxes** (風配図 (wind rose) の図を参照)、**boxxyerror** と **image pixels** (**heatmaps** の例を参照) に対する極座標での同等品を提供します。sector での描画は直交座標モードでのグラフレイアウトと両立するため、一つのグラフ上の異なる場所に複数のグラフを配置することが可能ですが、それは他の極座標モードグラフスタイルでは不可能です。

ここに示しているのは、sectors を使って風配図 (wind rose) を生成する例です。極座標温度分布図を含む他の応用例としては、ダイヤルチャート、円グラフ/円環グラフ、そして 極座標でのデータ点に対する環状のエラーボックスです。これらすべての加工された例は、sector のデモ

に置かれています。

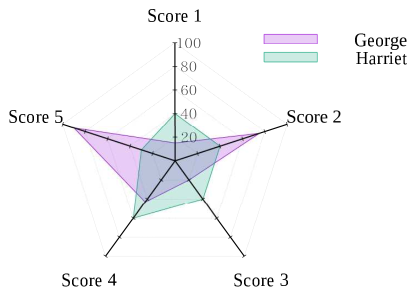
Spiderplot

クモの巣グラフ (spiderplot) は、本質的に、平行座標描画 (parallel axis) で、その軸を垂直方向ではなく放射状に配置したものです。これはよく「レーダーチャート」(**radar chart**) とも呼ばれます。gnuplot 内部では、これはコマンド **set spiderplot** で確立される座標系での作業が必要となりますが、これは偏角座標が平行軸番号によって暗黙に決まることを除けば **set polar** に似ています。見た目や、ラベル付け、軸の目盛りの配置は、**set paxis** で制御できます。このスタイルのさらなる設定は、**set style spiderplot**、**set grid**、および plot コマンドの個々の指定で行えます。

各クモの巣グラフは、列データではなく行データに対応するため、通常の方法での凡例 (key) タイトルの生成は意味がありません。代わりに、描画要素がタイトル文字列を持っていれば、その文字列を対応する軸のラベルに使用します。これは、事前の **set paxis n label "Foo"** をすべて上書きします。凡例にタイトルを配置するには、個別の **keyentry** コマンドを使用するか、または using 指定で **key(column)** を使って入力ファイル列から文字列を展開する方法が使えます。

以下の図は、軸が 5 つのクモの巣グラフで、それぞれ 5 つの点数で特徴付けられる複数のものを比較するのに使います。$DATA の各行は、グラフ上に新しい多角形を生成します。

```
$DATA << EOD
         A       B       C       D       E       F
George   15      75      20      43      90      50
Harriet 40       40      40      60      30      50
EOD
set datafile columnheader
set spiderplot
set style spiderplot fs transparent solid 0.2 border
set for [p=1:5] paxis p range [0:100]
set for [p=2:5] paxis p tics format ""
set             paxis 1 tics font ",9"
set for [p=1:5] paxis p label sprintf("Score %d",p)
set grid spiderplot
```



```
plot for [i=2:6] $DATA using i:key(1)
```

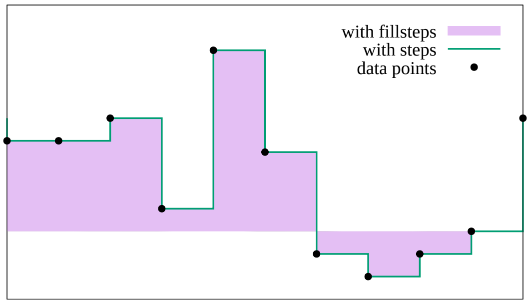
Newspiderplot

通常、**with spiderplot** による plot コマンドに与えられるデータ列の個々の値は、1 個の多角形に対するそれぞれの頂点に対応します。1 つのグラフ上に、複数の多角形を描くには、それらを **newspiderplot** で分離して指定します。例:

```
# 以下は 10 個の頂点を持つ 1 つの多角形
plot for [i=1:5] 'A' using i, for [j=1:5] 'B' using j
# 以下は 5 個の頂点を持つ 2 つの多角形
plot for [i=1:5] 'A' using i, newspiderplot, for [j=1:5] 'B' using j
```

Steps

steps スタイルは 2 次元描画でのみ利用可能です。これは 2 本の線分で隣り合う点をつなぎます: 1 本目は (x1,y1) から (x2,y1) まで、2 本目は (x2,y1) から (x2,y2) まで。入力列の条件は、**lines** や **points** に対するものと同じです。**fsteps** と **steps** の違いは、**fsteps** は、折れ線を先に y 方向に書いてから次に x 方向に書くのに対し、**steps** は先に x 方向に書いてから次に y 方向に書きます。曲線とベースラインである y=0 との間の領域を塗り潰すには、**fillsteps** を使用してください。以下も参照steps デモ。




```
1 列:    y     # 行番号 (0 列目) による暗黙の x
2 列:    x y
```

Surface

描画スタイル **with surface** には、2 種類の利用法があります。

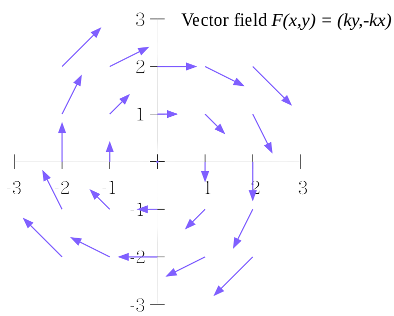
3 次元グラフでは、**with surface** は常に曲面を生成します。3 次元データ集合が格子集合 (grid) と認識できれば、gnuplot はデフォルトでは、**with lines** を **with surface** に対する同義語として、格子曲面を要求したかのように暗黙に描画スタイル **with lines** で処理します。しかしコマンド **set surface explicit** はこの処理を抑制し、その場合 **with surface** と **with lines** は異なるスタイルとなり、同じグラフ内で利用することもできます。

格子点とは認識されない 3 次元の点がある場合、最初に適切な格子に合わせることが可能です。以下参照: **set dgrid3d (p. 213)**。

2 次元極座標グラフでは、**with surface** は、データの色塗り格子表現を生成するのに使います。その曲面の生成は、コマンド **set polar grid** で制御します。

Vectors

2 次元の **vectors** スタイルは (x,y) から (x+xdelta,y+ydelta) までのベクトルを書きます。3 次元の **vectors** スタイルも同様ですが、データは基本的に 6 列必要です。いずれの場合も、入力列を追加 (2D では 5 列目、3D では 7 列目) すると、それらは各データ点毎の variable color 情報 (以下参照: **linecolor (p. 71)**, **rgbcolor variable (p. 73)**) として使われます。各ベクトルの先端には小さな矢先も書かれます。




```
4 列:    x  y  xdelta  ydelta
6 列:    x  y  z  xdelta  ydelta  zdelta
```

キーワード "with vectors" は、その後ろに、arrow スタイル属性を直接書いたり、事前に定義した arrow スタイルを参照したり、あるいは別の入力列から各ベクトルに対する適用したい arrow スタイルのインデックスを読むよう指定したりすることができます。下の最初の 3 つの例を見てください。

例:

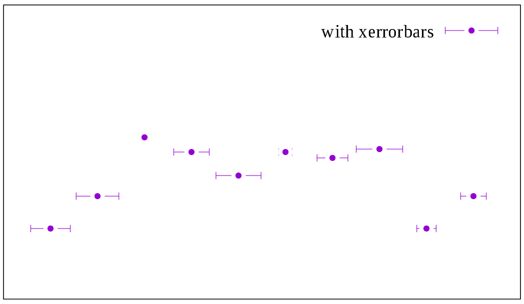
```
plot ... using 1:2:3:4 with vectors filled heads
plot ... using 1:2:3:4 with vectors arrowstyle 3
plot ... using 1:2:3:4:5 with vectors arrowstyle variable
splot 'file.dat' using 1:2:3:(1):(1):(1) with vectors filled head lw 2
```

注意: plot コマンド内にキーワード **arrowstyle** と他の線属性を混在させることはできません。arrow スタイルに **lc variable** か **lc rgb variable** がある場合、色の値用の追加列が必要です。

vectors スタイルを使っての splot は **set mapping cartesian** のみでサポートされています。**set clip one** と **set clip two** は 2 次元のベクトルの描画に影響を与えます。以下参照: **set clip (p. 202)**, **arrowstyle (p. 280)**。

2 次元描画スタイルの以下も参照: **with arrows (p. 90)**。これは、各矢印を x:y:length:angle の形式で指定する以外は **with vectors** と同じです。

Xerrorbars

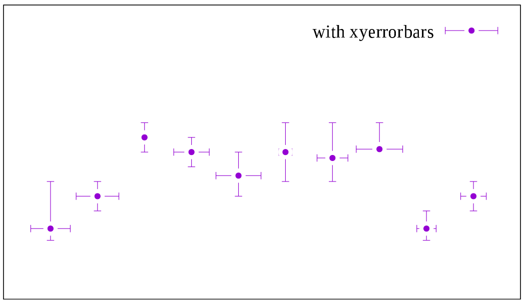


xerrorbars スタイルは 2 次元のデータ描画のみで利用可能です。**xerrorbars** は、水平の誤差指示線 (error bar) が表示される以外は **points** と同じです。各点 (x,y) において (xlow,y) から (xhigh,y) まで、または (x-xdelta,y) から (x+xdelta,y) までの線分が引かれますが、これらはいくつのデータ列が与えられるかによって変わります。誤差指示線の端の刻みの印の見た目は、**set errorbars** で制御できます。点と誤差指示線の間の隙間は、**set pointintervalbox** で制御します。誤差線を断線せずに直接点記号を貫通させるには、**unset pointintervalbox** を使用してください。このスタイルは基本的に、3 列か 4 列のデータが必要です:

```
3 列:  x  y  xdelta
4 列:  x  y  xlow  xhigh
```

入力列を追加 (4,5 列目) するとそれらを variable color として使います。このスタイルでは、variable point 属性は使えません。

Xyerrorbars



xyerrorbars スタイルは 2 次元のデータ描画のみで利用可能です。**xyerrorbars** は、水平、垂直の誤差指示線 (error bar) も表示される以外は **points** と同じです。各点 (x,y) において (x,y-ydelta) から (x,y+ydelta) までと (x-xdelta,y) から (x+xdelta,y) まで、または (x,ylow) から (x,yhigh) までと (xlow,y) から (xhigh,y) までの線分が引かれますが、これらはいくつのデータ列が与えられるかによって変わります。誤差指示線の端の刻みの印の見た目は、**set errorbars** で制御できます。点と誤差指示線の間の隙間は、**set pointintervalbox** で制御します。誤差線を断線せずに直接点記号を貫通させるには、**unset pointintervalbox** を使用してください。これは 4 列か、6 列のデータが必要です。

```
4 列:  x  y  xdelta  ydelta
6 列:  x  y  xlow  xhigh  ylow  yhigh
```

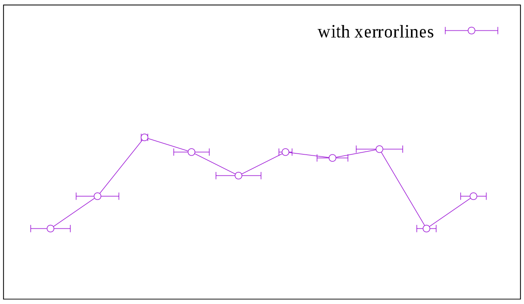
データが、サポートされていない混合型の形式で与えられた場合、**plot** コマンドの **using** 指定を使って適切な形に直さないといけません。例えばデータが (x,y,xdelta,ylow,yhigh) という形式である場合、以下のようにします:

```
plot 'data' using 1:2:($1-$3):($1+$3):4:5 with xyerrorbars
```

入力列を追加 (5,7 列目) するとそれらを variable color として使います。このスタイルでは、variable point 属性は使えません。

Xerrorlines

xerrorlines スタイルは 2 次元のデータ描画のみで利用可能です。**xerrorlines** は **linespoints** に似ていますが、水平の誤差線が描かれることが違います。各点 (x,y) で、データ列の個数に応じて (xlow,y) から (xhigh,y) まで、または (x-xdelta,y) から (x+xdelta,y) までの線分が描かれます。誤差線の端の刻みの印の見た目は、**set errorbars** で制御できます。基本的には、3 列か 4 列のデータが必要です:

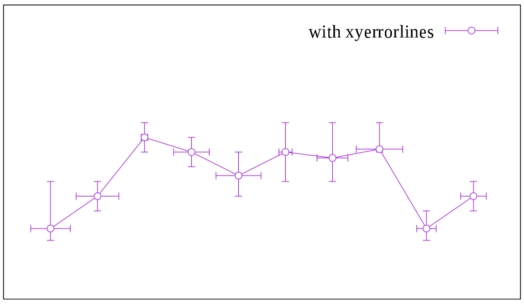



```
3 列:   x   y   xdelta
4 列:   x   y   xlow   xhigh
```

入力列を追加 (4,5 列目) するとそれらを variable color として使います。このスタイルでは、variable point 属性は使えません。

Xyerrorlines

xyerrorlines スタイルは 2 次元のデータ描画のみで利用可能です。**xyerrorlines** は **linespoints** に似ていますが、水平と垂直の誤差線も描かれることが違います。各点 (x,y) で、データ列の個数に応じて、(x,y-ydelta) から (x,y+ydelta) までと (x-xdelta,y) から (x+xdelta,y) まで、あるいは (x,ylow) から (x,yhigh) までと (xlow,y) から (xhigh,y) までの線分が描かれます。誤差線の端の刻みの印の見た目は、**set errorbars** で制御できます。これは、4 列か 6 列の入力データが必要です。



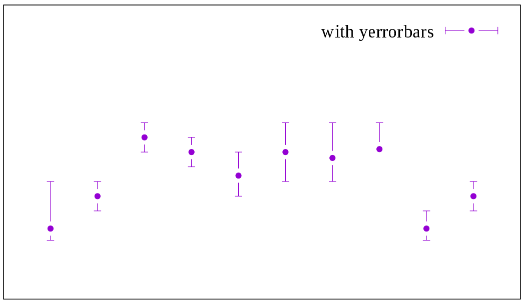

```
4 列:   x   y   xdelta   ydelta
6 列:   x   y   xlow   xhigh   ylow   yhigh
```

データが、サポートされていない混合型の形式で与えられた場合、**plot** コマンドの **using** 指定を使って適切な形に直さないといけません。例えばデータが (x,y,xdelta,ylow,yhigh) という形式である場合、以下のようにします:

```
plot 'data' using 1:2:($1-$3):($1+$3):4:5 with xyerrorlines
```

入力列を追加 (5,7 列目) するとそれらを variable color として使います。このスタイルでは、variable point 属性は使えません。

Yerrorbars



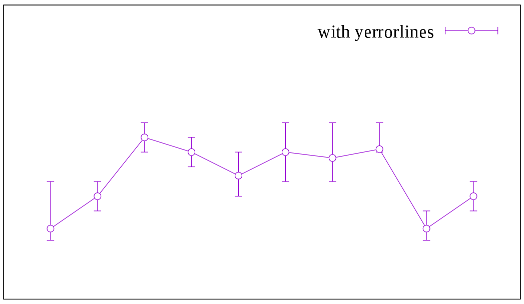
yerrorbars (または **errorbars**) スタイルは 2 次元のデータ描画のみで利用可能です。**yerrorbars** は、垂直の誤差指示線 (error bar) が表示される以外は **points** に似ています。各点 (x,y) において (x,y-ydelta) から (x,y+ydelta) まで、または (x,ylow) から (x,yhigh) までの線分が引かれますが、これらはいくつのデータ列が与えられるかによって変わります。誤差指示線の端の刻みの印の見た目は、**set errorbars** で制御できます。点と誤差線との隙間は **set pointintervalbox** で制御します。誤差線を断線せずに直接点記号を貫通させるには、**unset pointintervalbox** を使用してください。

```
2 列:  [暗黙の x]  y  ydelta
3 列:  x  y  ydelta
4 列:  x  y  ylow  yhigh
```

入力列を追加すると、それらは可変 (variable) pointsize, 可変 pointtype, 可変色 (variable color) の情報として使われます。

以下も参照`errorbar` デモ。

Yerrorlines



yerrorlines (または **errorlines**) スタイルは 2 次元のデータ描画のみで利用可能です。**yerrorlines** は **linespoints** に似ていますが、垂直の誤差線が描かれることが違います。各点 (x,y) で、データ列の個数に応じて (x,y-ydelta) から (x,y+ydelta) まで、または (x,ylow) から (x,yhigh) までの線分が描かれます。誤差線の端の刻みの印の見た目は、**set errorbars** で制御できます。これは、3 列か 4 列の入力が必要です。

```
3 列:  x  y  ydelta
4 列:  x  y  ylow  yhigh
```

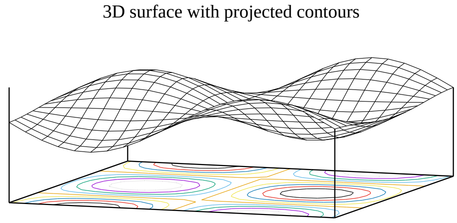
入力列を追加 (4,5 列目) すると、それらは点の variable color 情報として使われます。さらなる入力列は、variable point size, variable point type, variable color の情報として使われます。

以下も参照エラーバーのデモ

3 次元描画 (3D plots)

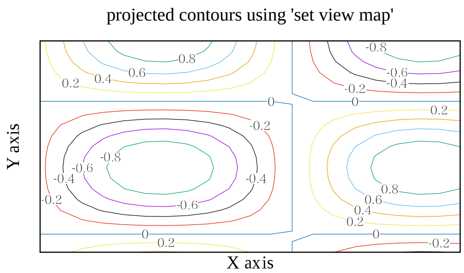
3 次元グラフは、コマンド **plot** ではなくコマンド **splot** を使って生成します。多くの 2 次元描画スタイル (points, images, impulse, labels, vectors) は、z 座標データ列を追加指定すれば 3 次元でも使えます。2 次元射影グラフのみが欲しい場合でも、**splot** コマンドを使って生成しなければいけない描画型 (pm3d coloring, surfaces, contours) も中にはあります。

曲面描画 (surface plots)



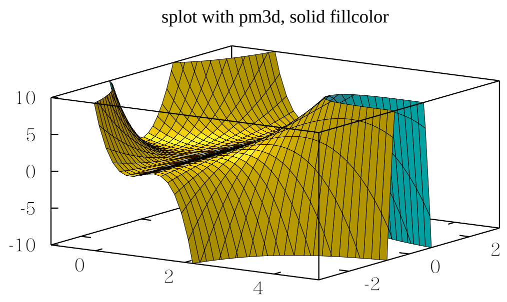
描画スタイル **splot with lines** や **splot with surface** はいずれも曲面を格子線で生成します。曲面の塗り潰しは、スタイル **splot with pm3d** で行うことができます。曲面は通常、それが 3 次元の曲面であるとはっきりわかるような適切な視方向から表示されます。以下参照: **set view (p. 298)**。その場合、X, Y, Z 軸はすべて描画内に表示されます。3 次元的な錯覚は、隠線処理により、より増幅されます。以下参照: **hidden3d (p. 224)**。コマンド **splot** は、定数の Z 値に対する等高線を計算し描画することもできます。これらの等高線は、曲面それ自体の上に書くこともできますし、XY 平面へ射影することもできます。以下参照: **set contour (p. 207)**。

2 次元射影 (set view map)



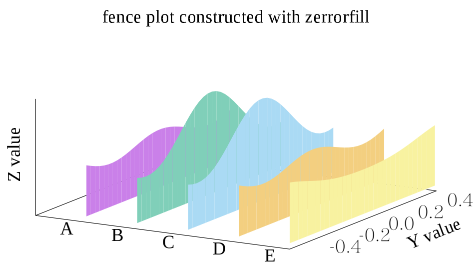
コマンド **splot** の特別な場合として、グラフの Z 軸に沿った xy 平面への射影による、Z 座標の 2 次元曲面への地図作成 (map) があります。以下参照: **set view map (p. 298)**。この描画モードは、等高線の描画や温度分布 (heatmap) の生成に便利でしょう。以下の図は描画スタイル **lines** を一度、**labels** を一度描画した等高線を示しています (訳注: 図が表示されている場合)。

PM3D 描画 (PM3D plots)



3 次元曲面は、線分でなく、単色の pm3d 四辺形を使って描画することもできます。その場合、隠面処理はありませんが、各面要素を背景から前面に向かって描くことで同様の効果が得られます。以下参照: **set pm3d depthorder (p. 271)**。pm3d 曲面は、デフォルトでは滑らかな連続的なカラーパレットを使って色付けします (以下参照: **set palette (p. 258)**) が、単色の曲面を指定することもできますし、この図にあるように (訳注: 図が表示されている場合)、上の面と下の面に異なる単色を指定することもできます。以下参照: **pm3d fillcolor (p. 273)**。hidden3d モードでの線分の切り取りとは違い、pm3d 曲面は現在の zrange の範囲に滑らかにクリッピングできます。以下参照: **set pm3d clipping (p. 271)**。

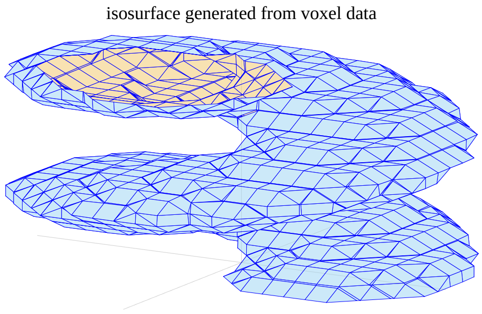
柵グラフ (Fence plots)



柵グラフ (fence plot) は、複数の 2 次元グラフを、その Y 座標は揃え、それぞれを区別するために X に沿ってはずらした形で結合したものです。土台の値から個々のグラフの Z 座標までの間の領域を塗り潰すことで、Y 方向の整列と Z 座標の高さの違いの見た目を強調します。gnuplot でこの形式のグラフを書くにはいくつかの方法があります。最も単純なのは、5 列の形式の **zerrorfill** スタイルを利用する方法です。i で添字化された複数の曲線 z = Fi(y) があるとき、**splot with zerrorfill** で以下のような入力列を使えば柵グラフを書くことができます:

```
i y z_base z_base Fi(y)
```

ボクセルデータのモザイク型曲面 (isosurface)



この 3 次元描画スタイルには、値の入っているボクセル格子データが必要です (以下参照: **set vgrid** (p. **298**), **vfill** (p. **333**))。要求する値レベルに対応する非整数格子座標を評価する際は、ボクセル格子値の線形補間を使用します。それらの点は、モザイク型曲面を生成するのに使用します。その曲面を構成する切片は、**set pm3d** で色、透過、境界属性などが制御できるよう pm3d 多角形として描画します。一般に、その曲面は、その切片が細い境界で与えられ、その境界が塗り潰しの色よりも暗い色であれば、見た目がわかりやすくなります。デフォルトでは、モザイク型曲面は四角形と三角形を混ぜて使いますが、三角形のみを使いたい場合は、以下参照: **set isosurface** (p. **227**)。例:

```
set style fill solid 0.3
set pm3d depthorder border lc "blue" lw 0.2
splot $helix with isosurface level 10 fc "cyan"
```

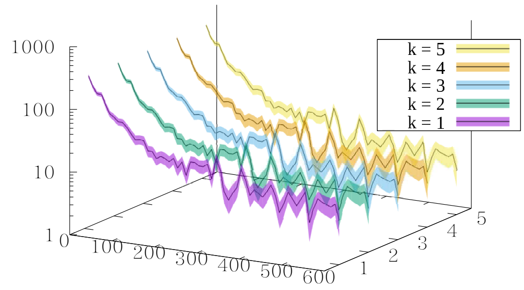
3 次元での曲線間の塗り潰し (zerrorfill)

書式:

```
splot DATA using 1:2:3:4[:5] with zerrorfill {fc|fillcolor <colorspec>}
              {lt|linetype <n>} {<line properties>}
```

描画スタイル **zerrorfill** は、2 次元の描画スタイルの一つの変種のようなものです。これは、2 つの関数の間、または同じ x, y に対して 2 つの z の値を点を与えて得られるデータの折れ線の間の領域を塗り潰します。これは、4 列か 5 列の入力が必要です。

```
4 列:  x  y  z  zdelta
5 列:  x  y  z  zlow  zhigh
```



zlow と zhigh の間の領域を塗り潰し、その後 z の値のところに線を描きます。デフォルトでは、その線と塗り潰しには同じ色を使いますが、その色は splot コマンド上で変更できます。塗り潰しの設定は、大域的な fill style の影響も受けます。以下参照: **set style fill (p. 283)**。

splot コマンドに複数の曲線を指定した場合は、後に描いた曲線が、前のすべての曲線を隠してしまう可能性があります。見る人側の手前の曲線のみが隠すように適切な深さ順の並べかえを行うには、後ろから前への順番付けが最良です。他には、**set pm3d depthorder base** を用いて自動的に並べかえることができますが、残念ながら、これは z の値に対応するすべての折れ線を描いた後に、すべての領域の塗り潰しを行います。よって、折れ線を見えるようにして、かつ領域の塗り潰しの深さ順の並べかえを行うには、領域の塗り潰しは部分的に透過 (transparent) させるといいかもしれません。

以下の最初の 2 つの例の塗り潰し領域は、同じものになります。

```
splot 'data' using 1:2:3:4 with zerrorfill fillcolor "grey" lt black
splot 'data' using 1:2:3:($3-$4):($3+$4) with zerrorfill
splot '+' using 1:(const):(func1($1)):(func2($1)) with zerrorfill
splot for [k=1:5] datafile[k] with zerrorfill lt black fc lt (k+1)
```

この描画スタイルは、柵グラフ (fence plot) を作成するのにも使えます。以下参照: **fenceplots (p. 131)**。以下も参照: **waterfallplots (p. 102)**。

アニメーション (Animation)

gnuplot の対話型出力形式 (qt, win, wxt, x11, aqua) ではいずれも、コマンドラインやスクリプトから連続するフレームの描画を行うことでアニメーションを表示することができます。

マウスが使えない出力形式でも、ある種のアニメーションをサポートするものがあります。以下参照: **term sixelgd (p. 387)**, **term kittycairo (p. 365)**。

アニメーションをファイルとして保存して、後で手元で再生したり Web ページに埋め込んだりできる出力形式が 2 つあります。以下参照: **term gif animate (p. 361)**, **term webp (p. 395)**。

例:

```
unset border; unset tics; unset key; set view equal xyz
set pm3d border linecolor "black"

set term webp animate delay 50
set output 'spinning_d20.webp'
do for [ang=1:360:2] {
    set view 60, ang
    splot 'icosahedron.dat' with polygons fc "gold"
}
unset output
```

第 III 部

コマンド (Commands)

このセクションでは **gnuplot** が受け付けるコマンドをアルファベット順に並べています。このドキュメントを紙に印刷したものは全てのコマンドを含んでいますが、対話型で参照できるドキュメントの方は完全ではない可能性があります。実際、この見出しの下に何のコマンドも表示されないシステムがあります。

ほとんどの場合、コマンド名とそのオプションは、紛らわしくない範囲で省略することが可能です。すなわち、**"plot f(x) with lines"** の代わりに**"p f(x) w li"** とすることができます。

書式の記述において、中カッコ ({}) は追加指定できる引数を意味し、縦棒 (|) は互いに排他的な引数を区切るものとします。

Break

コマンド **break** は、**do, while** 文の繰り返し実行部分のカッコ内でのみ意味を持ちます。このコマンドは、その中カッコ内の残りの命令をスキップし、繰り返しを中断し、その閉じカッコの次の文から実行を再開します。以下も参照: **continue (p. 135)**。

Cd

cd コマンドはカレントディレクトリを変更します。

書式:

```
cd '<ディレクトリ名>'
```

ディレクトリ名は引用符に囲まれていなければなりません。

例:

```
cd 'subdir'
cd '..'
```

バックスラッシュ (\) は二重引用符内 (") では特別な意味を持ってしまうためにエスケープする必要がありますので、Windows ユーザには単一引用符を使うことを勧めます。例えば、

```
cd "c:\newdata"
```

では失敗しますが、

```
cd 'c:\newdata'
cd "c:\\newdata"
```

なら期待通りに動くでしょう。

Call

call コマンドは、読み込むファイル名の後ろに、9 つまでのパラメータを与えることができることを除けば **load** コマンドと等価です。

```
call "inputfile" <param-1> <param-2> <param-3> ... <param-9>
```

現在の gnuplot は、文字列変数 ARG0, ARG1, ..., ARG9 と、整数変数 ARGC を提供します。**call** コマンドを実行すると、ARG0 には入力ファイル名が、ARGC にはパラメータ数が設定され、ARG1 から ARG9 にはコマンドラインに並べられたパラメータの値が読み込まれます。

通常パラメータ ARG1 ... ARG9 は文字列値として保存されるので、それをマクロ展開して参照することもできます。しかし、多くの場合、それらは他の変数と同様に利用する方がより自然でしょう。

パラメータ ARG1 ... ARG9 の文字列表現と平行して、そのパラメータ自身を配列 ARGV[9] にも保存します。以下参照: **ARGV (p. 134)**。

非推奨: 5.0 以前のバージョンでは、<param-1> ... 等の内容を、特別な記号$0, $1, ..., $9 をマクロのように置換することで表現していました。この古い仕組みは、もはやサポートしていません。

関数ブロック (このバージョンからの新機能) は、**call** に代わるより柔軟な機能を提供します。以下参照: **function blocks (p. 148)**。

ARGV[ ]

call コマンドにより gnuplot スクリプトに入った場合、呼び出し側からのパラメータは、2 つの仕組みで利用できます。各パラメータは、まず文字列として変数 ARG1, ARG2, ... ARG9 に保存します。さらにそれらは配列 ARGV[9] の各要素としても保存します。こちらは、数値は複素変数値として保存しますが、それ以外はすべて文字列として保存します。ARGC はパラメータの個数を保持します。よって、以下の call 後には

```
call 'routine_1.gp'  1 pi "title"
```

以下の 3 つの引数を routine_1.gp 内で以下の値として利用できます:

```
ARGC = 3
ARG1 = "1"         ARGV[1] = 1.0
ARG2 = "3.14159"   ARGV[2] = 3.14159265358979...
ARG3 = "title"     ARGV[3] = "title"
```

この例では、ARGV[1] と ARGV[2] は可能な限りの精度を持つ浮動小数値となりますが、ARG2 は書式 "%g" による文字列として保存されるため精度が落ちています。

ARGC とそれに対応する配列 ARGV[ARGC] は関数ブロック呼び出しの内部でも利用可能です。しかし、関数ブロック呼び出しでは、文字列変数 ARG1,... は作られません。

例 (Example)

```
以下を call すると:
    MYFILE = "script1.gp"
```

```
FUNC = "sin(x)"
call MYFILE FUNC 1.23 "This is a plot title"
```

呼び出されたスクリプト内では以下のようになり:

```
ARG0 は "script1.gp"
ARG1 は文字列値 "sin(x)"
ARG2 は文字列値 "1.23"
ARG3 は文字列値 "This is a plot title"
ARGC は 3
```

そのスクリプト内では以下のようなものを実行できる:

```
plot @ARG1 with lines title ARG3
print ARG2 * 4.56, @ARG2 * 4.56
print "This plot produced by script ", ARG0
```

この例の ARG1 は文字列なので、それはマクロとして参照しなければいけませんが、ARG2 はマクロ参照でも (数値定数になる)、変数のままでも (文字列"1.23" が実数値に自動的に変換された後の同じ数値になる) 構わないことに注意してください。

シェルスクリプトで gnuplot をコマンドラインオプション **-c** つきで実行することで、これと同じことを直接行うこともできます:

```
gnuplot -persist -c "script1.gp" "sin(x)" 1.23 "This is a plot title"
```

Clear

clear コマンドは、**set terminal** や **set output** で選択した画面、出力装置をクリアします。ハードコピー装置に対しては通常改ページを行います。

いくつかの出力装置は **clear** コマンドでは **set size** で定義された描画領域のみを消去します。そのため、**set multiplot** とともに使用することで挿入図を一つ作ることができます。

例:

```
set multiplot
plot sin(x)
set origin 0.5,0.5
set size 0.4,0.4
clear
plot cos(x)
unset multiplot
```

詳細については、以下参照: **set multiplot** (p. **247**), **set size** (p. **278**), **set origin** (p. **256**)。

Continue

コマンド **continue** は、**do**, **while** 文の繰り返し実行部分のカッコ内でのみ意味を持ちます。このコマンドは、その中カッコ内の残りの命令をスキップし、次の繰り返しに進みます (もしループの残りがあれば)。以下も参照: **break** (p. **133**)。

Do

書式:

```
do for <iteration-spec> {
     <commands>
     <commands>
}
```

これは、コマンド列を複数回実行します。コマンドは中カッコ {} で囲み、かつ開始カッコ "{" は、キーワード **do** と同じ行に置く必要があります。このコマンドは、古い形式 (かっこなし) の if/else 構文と一緒に使うことはできません。繰り返し指定 <iteration-spec> の例については、以下参照: **iteration (p. 69)**。例:

```
set multiplot layout 2,2
do for [name in "A B C D"] {
    filename = name . ".dat"
    set title sprintf("Condition %s",name)
    plot filename title name
}
unset multiplot
```

以下も参照: **while (p. 334)**, **continue (p. 135)**, **break (p. 133)**。

Evaluate

コマンド **evaluate** は、文字列、または関数ブロックに含まれる gnuplot コマンドを実行します。その文字列中に改行文字を入れてはいけません。

```
evaluate "commands in a string constant"
evaluate string_valued_function( ... arguments ... )
evaluate $functionblock( ... arguments ... )
```

これは、特に同様のコマンドの繰り返しに有用です。

例:

```
set_label(x, y, text) \
  = sprintf("set label '%s' at %f, %f point pt 5", text, x, y)
eval set_label(1., 1., 'one/one')
eval set_label(2., 1., 'two/one')
eval set_label(1., 2., 'one/two')
```

gnuplot コマンドを持つ文字列を構成したり実行する他の仕組みについては、以下参照: **function blocks (p. 148)**, **substitution macros (p. 83)**。

Exit

```
exit
```

```
exit message "エラーメッセージ文字列"
exit status <整数のエラーコード>
```

exit と **quit** の両コマンドは END-OF-FILE 文字 (通常 Ctrl-D) 同様、現在の入力ストリーム、すなわち端末の対話やパイプ入力、ファイル入力 (パイプ) からの入力を終了させせます。入力ストリームが入れ子 (階層的な **load** のスクリプトで) になっている場合、読み込みは親のストリームで継続されます。トップレベルのストリームが閉じられると、プログラムはそれ自身終了します。

コマンド **exit gnuplot** は、直ちに、無条件に、そして例え入力ストリームが多段階にネストされていても、gnuplot を終了させせます。その場合、開かれていた全ての出力ファイルはきれいに完全な形では閉じられない可能性があります。使用例:

```
bind "ctrl-x" "unset output; exit gnuplot"
```

コマンド **exit error "error message"** は、疑似プログラムエラーを行います。対話型モードでは、そのエラーメッセージを表示し、すべてのネストされたループや call を中断してコマンドラインに帰ります。非対話型モードでは、プログラムを終了します。

gnuplot が終了しシェルの制御に戻る場合、その返り値は意味のないものになることがありますが、このコマンドを以下のように実行すれば、シェルに特定の値を返すことが可能です。

```
exit status <value>
```

詳細は、以下参照: **batch/interactive (p. 38)**。

Fit

コマンド **fit** は、Marquardt-Levenberg 法による非線形最小自乗法 (NLLS) を用いて、データ点の集合にユーザが与える式を当てはめます。独立変数は 12 まで許されていて、従属変数は常に 1 つで、任意個数のパラメータを当てはめることができます。さらに追加で、データ点の重み付け用に誤差評価を入力することも可能です。

fit の最も基本的な使用法は、以下の単純な例が示しています。ここで、ファイルから読み込む x と y の計測値 (measured) の集合は、関数 y = f(x) のモデル化に使います。

```
f(x) = a + b*x + c*x**2
fit f(x) 'measured.dat' using 1:2 via a,b,c
plot 'measured.dat' u 1:2, f(x)
```

書式:

```
fit {<ranges>} <expression>
    '<datafile>' {datafile-modifiers}
    {{unitweights} | {y|xy|z}error | errors <var1>{,<var2>,...}}
    via '<parameter file>' | <var1>{,<var2>,...}
```

範囲 (xrange, yrange 等) は、当てはめに使用するデータを制限する目的で使うことができ、その範囲を超えたデータは無視します。その書式は **plot** コマンド同様

```
[{dummy_variable=}{<min>}{:<max>}],
```

です。以下参照: **plot ranges** (p. 181)。

<expression> は、通常はあらかじめユーザ定義された f(x) または f(x,y) の形の関数ですが、**gnuplot** で有効などんな数式でも指定できます。ただし実数値関数でなければいけません。独立変数の名前は、コマンド **set dummy** で設定するか、fit の範囲指定部分 (<rangse>) で設定します (以下を参照)。デフォルトでは、最初の 2 つは x, y となります。さらに、その数式は、当てはめの作業により決定する値を持つ 1 つ以上の変数 (パラメータ) に依存すべきです。

<datafile> は **plot** コマンドと同様に扱われます。**plot datafile** の修飾子 (**using**, **every**,...) は、**smooth** を除いて、全て **fit** に使うことができます。以下参照: **plot datafile** (p. 162)。

データファイルの内容は、plot コマンドに使用するのと同じ **using** 指定を使うことで柔軟に解釈させることができます。例えば、独立変数 x を 2 列目と 3 列目の和として生成し、z の値を 6 列目から取り、重みを 1 としたい場合は以下のようにします:

```
fit ... using ($2+$3):6
```

using 指定がない場合、fit は暗黙に独立変数は 1 つだけと仮定します。ファイル自身、または using 指定が 1 列だけのデータを持つ場合、その行番号を独立変数値として使用します。**using** 指定を与えた場合、最大 12 個 (指定してコンパイルしていればさらにそれ以上) の独立変数を利用できます。

オプション **unitweights** (これがデフォルト) は、すべてのデータ点が等しい重みを持つとみなします。これは、キーワード **error** を使用することで変更でき、これはデータファイルから 1 つ以上の変数の誤差評価を読み込み、その誤差評価を対応する変数値の標準偏差 s とみなし、各データに 1/s**2 の重みを計算するのに使用します。

独立変数の誤差評価において、その重みには、"有効分散法" (effective variance method; Jay Orear, Am. J. Phys., Vol. 50, 1982) に従って、さらに当てはめ関数の微分係数をかけます。

キーワード **errors** には、その後ろに、入力がどの変数の誤差であるのかを示すコンマ区切りの 1 つ以上の変数名のリストが付きます。従属変数 z は常にその中になければいけませんが、独立変数は必須ではありません。そのリストの各変数に対し、ファイルからその分の、各変数の誤差評価を持つ追加の列を読み込みます。繰り返しになりますが、**using** 指定により柔軟な解釈が可能になります。よって、独立変数の数は暗黙に、**using** 指定内の列の数から 1 を引いて (従属変数分)、さらに **errors** 指定内の変数の個数を引いた数になることに注意してください。

例として、2 つの独立変数があり、そして 1 つ目の独立変数と従属変数の誤差データがある場合は、**errors x,z** 指定と 5 列の **using** 指定を使うことになりますが、それは x:y:z:sx:sz のように解釈されます (x, y は独立変数、z が従属変数、sx, sz は x, z の標準偏差)。

errors 指定のちょっとした略記法も 2,3 用意されています: **yerrors** (独立変数が 1 列ある当てはめ用)、**zerrors** (より一般の場合) は、いずれも **errors z** と同値で、1 列だけ追加の従属変数用の誤差列があることを意味しています。

xyerrors は、独立変数は 1 列で、その独立変数と従属変数の両方の 2 列の誤差列が追加されることを意味します。この場合、x と y の誤差は Orear の有効分散法 (effective variance method) で処理されます。

yerror と **xyerror** の形式および解釈は、それぞれ 2 次元描画スタイルの **yerrorlines** と **xyerrorlines** に同等であることに注意してください。

コマンド **set fit v4** を使用すると、fit のコマンド書式は **gnuplot** バージョン 4 と互換の書式になります。その場合、**using** には、独立変数が 2 つ以上ならば、独立変数の数より 2 つ (z と s) 多い指定が必要で、**gnuplot** は、**using** 指定で与えられた列の数に応じて、以下の書式に従います:

```
z                         # 独立変数は 1 つ (行番号)
x:z                       # 独立変数は 1 つ (第 1 列)
x:z:s                     # 独立変数は 1 つ (全部で 3 列)
x:y:z:s                   # 独立変数は 2 つ (全部で 4 列)
x1:x2:x3:z:s              # 独立変数は 3 つ (全部で 5 列)
x1:x2:x3:...:xN:z:s       # 独立変数は N 個 (全部で N+2 列)
```

これは、2 つ以上の独立変数で fit をする場合、z-誤差 s を与える必要があることを意味することに注意してください。重みを 1 にしたい場合は、それを、例えば x:y:z:(1) のような書式を using に指定することで明示的に与える必要があります。

仮変数名は、下で紹介するように範囲指定で指定することで変更できます。最初の範囲は **using** 指定の最初のものに対応し、以下同様です。従属変数である z の範囲指定もできますが、それは、f(x,...) の値をその範囲外にしてしまうようなデータ点が、残差を最小化することには寄与しない場合に有効です。

複数のデータ集合も複数の 1 変数関数に同時に当てはめることも、y を '仮変数' とすれば可能です。例えばデータ行番号を使い、2 変数関数への当てはめ、とすればいいでしょう。以下参照: **fit multi-branch (p. 144)**。

via 指定子は、パラメータの最適化を、直接行うか、またはパラメータファイルを参照することによって行うかを指定します。

例:

```
f(x) = a*x**2 + b*x + c
g(x,y) = a*x**2 + b*y**2 + c*x*y
set fit limit 1e-6
fit f(x) 'measured.dat' via 'start.par'
fit f(x) 'measured.dat' using 3:($7-5) via 'start.par'
fit f(x) './data/trash.dat' using 1:2:3 yerror via a, b, c
fit g(x,y) 'surface.dat' using 1:2:3 via a, b, c
fit a0 + a1*x/(1 + a2*x/(1 + a3*x)) 'measured.dat' via a0,a1,a2,a3
fit a*x + b*y 'surface.dat' using 1:2:3 via a,b
fit [*:*][yaks=*:*] a*x+b*yaks 'surface.dat' u 1:2:3 via a,b

fit [][][t=*:*] a*x + b*y + c*t 'foo.dat' using 1:2:3:4 via a,b,c

set dummy x1, x2, x3, x4, x5
h(x1,x2,x3,x4,s5) = a*x1 + b*x2 + c*x3 + d*x4 + e*x5
fit h(x1,x2,x3,x4,x5) 'foo.dat' using 1:2:3:4:5:6 via a,b,c,d,e
```

反復の個々のステップの後で、当てはめの現在の状態についての詳細な情報が画面に表示されます。そし最初と最後の状態に関する同じ情報が "fit.log" というログファイルにも書き出されます。このファイルは前の当てはめの履歴を消さないように常に追加されていきます。これは望むなら削除、あるいは別な名前にできます。コマンド **set fit logfile** を使ってログファイルの名前を変更することもできます。

set fit errorvariables を使用した場合、各当てはめパラメータの誤差はそのパラメータと似た名前 ("_err" が追加された名前) の変数に保存されますので、その誤差をその後の計算の入力として使用することができます。

set fit prescale とした場合、当てはめパラメータを、それらの初期値からスケール変換します。これにより、個々のパラメータの大きさにかなり違いがあるような場合でも、Marquardt-Levenberg ルーチンがより早く、より信頼性のある値に収束させられるようになります。

当てはめの反復は Ctrl-C (wgnuplot では Ctrl-Break) を押すことで中断できます。現在の反復が正常に終了した後、以下のいずれかを選ぶことができます: (1) 当てはめを止めて現在のパラメータの値を採用する (2) 当てはめを続行する (3) **set fit script** か、環境変数 **FIT_SCRIPT** で指定した **gnuplot** コマンドを実行する。そのデフォルトは **replot** で、もしデータと当てはめ関数を一つのグラフにあらかじめ描画してあれば、現在の当てはめの状態を表示することができます。

fit が終了した後は、最後のパラメータの値を保存するのに **save fit** コマンドを使います。その値は再びパラメータの値として使うことができます。詳細は、以下参照: **save fit (p. 192)**。

パラメータの調整 (adjustable parameters)

via はパラメータを調節するための 2 つの方法を指定できます。一つはコマンドラインから直接指示するもので、もう一つはパラメータファイルを参照して間接的に行うものです。この 2 つは初期値の設定で違った方法を取ります。

調整するパラメータは、**via** キーワードの後ろにコンマで区切られた変数名のリストを書くことで指定できます。定義されていない変数は初期値 1.0 として作られます。しかし当てはめは、変数の初期値があらかじめ適切な値に設定されている方が多分速く収束するでしょう。

パラメータファイルは個々のパラメータを、個別に 1 行に一つずつ、初期値を次のような形で指定して書きます。

```
変数名 = 初期値
```

'#' で始まるコメント行や空行も許されます。特別な形式として

```
変数名 = 初期値          # FIXED
```

は、この変数が固定されたパラメータであることを意味し、それはこのファイルで初期化されますが、調節はされません。これは、**fit** でレポートされる変数の中で、どれが固定された変数であるかを明示するのに有用でしょう。なお、**# FIXED** と言うキーワードは厳密にこの形でなくてはなりません。

Fit の概略 (fit beginners_guide)

fit は、与えられたデータ点を与えられたユーザ定義関数にもっとも良く当てはめるようなパラメータを見つけるのに使われます。その当てはめは、同じ場所での入力データ点と関数値との自乗誤差、あるいは残差 (SSR:Sum of the Squared Residuals) の和を基に判定されます。この量は通常χ（カイ）自乗と呼ばれます。このアルゴリズムは SSR を 最小化することをしようとします。もう少し詳しく言うと、データ誤差の重みつき残差の自乗和 (WSSR) の最小化を行っています。ここで、残差は、二乗する前に入力データ誤差で重みづけします。詳細は、以下参照: **fit error_estimates (p. 141)**。

これが、(非線形) 最小自乗当てはめ法と呼ばれるゆえんです。非線形 が何を意味しているのかを見るための例を紹介しますが、その前にいくつかの仮定について述べておきます。ここでは簡単のため、1 変数のユーザー定義関数は z=f(x), 2 変数の関数は z=f(x,y) のようにし、いずれも従属変数として z を用いることにします。パラメータとは **fit** が調整して適切な値を決定するユーザ定義変数で、関数の定義式中の未知数です。ここで言う、線形性/非線形性とは、従属変数 z と **fit** が調整するパラメータとの関係に対するものであり、z と独立変数 x (または x と y) との関係のことではありません (数学的に述べると、線形最小自乗問題では、当てはめ関数のパラメータによる 2 階 (そして更に高階の) 導関数は 0、ということになります)。

線形最小自乗法では、ユーザ定義関数は単純な関数の和であり、それぞれは一つのパラメータの定数倍で他のパラメータを含まない項になります。非線形最小自乗法では、より複雑な関数を扱い、パラメータを色々な使い方をします。フーリエ級数は線形と非線形の最小自乗法の違いを表す一つの例です。フーリエ級数では一つの項は

```
z=a*sin(c*x) + b*cos(c*x).
```

のように表されます。もし、a と b が未知なパラメータで c は定数だとすればパラメータの評価は線形最小自乗問題になります。しかし、c が未知なパラメータならばそれは非線形問題になります。

線形の場合、パラメータの値は比較的簡単な線形代数の直接法によって決定できます。しかし、'gnuplot' が使用する反復法は、その特別な線形の場も含めて、より一般的な非線形の問題を解くことができます。**fit** は検索を行うことで最小値を探そうとします。反復の各ステップは、パラメータの新しい値の組に対して WSSR を計算します。Marquardt-Levenberg のアルゴリズムは次のステップのパラメータの値を選択します。そしてそれはあらかじめ与えた基準、すなわち、(1) 当てはめが "収束した" (WSSR の相対誤差がある限界値より小さくなった場合。以下参照: **set fit limit (p. 217)**)、または (2) あらかじめ設定された反復数の限界に達した場合 (以下参照: **set fit maxiter (p. 217)**)、のいずれかを満たすまで続けられます。キーボードからその当てはめの反復は中断できますし、それに続いて中止することもできます (以下参照: **fit (p. 137)**)。ユーザ変数 FIT_CONVERGED は、 直前の fit コマンドが収束により終了した場合は 1 を持ち、それ以外の理由で中断した場合は 0 を持ちます。FIT_NITER は、直前の当てはめで行われた繰り返しの回数を持ちます。

当てはめに使われる関数はしばしばあるモデル (またはある理論) を元にしていて、それはデータの振舞を記述したり、あるいは予測しようとします。よって **fit** は、データがそのモデルにどれくらいうまく当てはまっているのかを決定するため、そして個々のパラメータの誤差の範囲を評価するために、モデルの自由なパラメータの値を求めるのに使われます。以下参照: **fit error_estimates (p. 141)**。

そうでなければ、曲線による当てはめにおける関数は、モデルとは無関係に選ばれています (それは十分な表現力と最も少ない数のパラメータを持ち、データの傾向を記述しそうな関数として経験に基づいて選ばれるでしょう)。

しかし、もしあなたが全てのデータ点を通るような滑らかな曲線を欲しいなら **fit** ではなく、むしろ **plot** の **smooth** オプションでそれを行うべきでしょう。

誤差評価 (error estimates)

fit において "誤差" という用語は 2 つの異なった文脈で用いられます。一つはデータ誤差、もう一つはパラメータ誤差です。

データ誤差は、平方残差の重み付きの和 WSSR、すなわちχ 自乗を決定する際個々のデータ点の相対的な重みを計算するのに用いられます。それらはパラメータの評価に影響を与えます。それは、それらが、当てはめられた関数からの個々のデータ点の偏差が最終的な値に与える影響の大きさを決定することによります。正確なデータ誤差評価が与えられている場合には、パラメータの誤差評価等の **fit** が出力する情報はより役に立つでしょう。

statistical overview では **fit** の出力のいくつかを説明し、'practical guidelines' に対する背景を述べています。

統計的な概要 (statistical overview)

非線形最小自乗法 (Non-Linear Least-Squares) の理論は、誤差の正規分布の点から一般的に記述されています。すなわち、入力データは与えられた平均とその平均に対する与えられた標準偏差を持つガウス (正規) 分布に従う母集団からの標本と仮定されます。十分大きい標本、そして母集団の標準偏差を知ることに対しては、χ 自乗分布統計を用いて、通常「χ 自乗」と呼ばれる値を調べることにより「当てはめの良さ」を述べることができます。減らされた自由度のχ 自乗 (χ 自乗の自由度は、データ点の数から当てはめられるパラメータの個数だけ引いた数) が 1.0 である場合は、データ点と当てはめられた関数との偏差の重みつき自乗和が、現在のパラメータ値に対する関数と与えられた標準偏差によって特徴付けられた母集団の、ランダムなサンプルに対する自乗和とが全く同じであることを意味します。

分散 = 総計である数え上げ統計学同様、母集団の標準偏差が定数でない場合、各点は観測される偏差の和と期待される偏差の和を比較するときに個別に重みづけされるべきです。

最終段階で **fit** は 'stdfit'、すなわち残差の RMS (自乗平均平方根) で求められる当てはめの標準偏差と、データ点が重みづけられている場合に '減らされたχ 自乗' とも呼ばれる残差の分散をレポートします。自由度 (データ点の数から当てはめパラメータの数を引いたもの) はこれらの評価で使用されます。なぜなら、データ点の残差の計算で使われるパラメータは同じデータから得られるものだからです。データ点が重みを持つ場合、**gnuplot** はいわゆる p-値を計算します。それはその自由度と結果のχ 自乗値に対するχ 自乗分布の累積分布関数値を 1 から引いた値です。以下参照: **fit practical_guidelines (p. 142)**。これらの値は以下の変数に代入されます:

```
FIT_NDF = 自由度の数
FIT_WSSR = 重みつき残差の自乗和
FIT_STDFIT = sqrt(WSSR/NDF)
FIT_P = p-値
```

パラメータに関する信頼レベルを評価することで、当てはめから得られる最小のχ 自乗と、要求する信頼レベルのχ 自乗の値を決定するためのχ 自乗の統計を用いることが出来ます。しかし、そのような値を生成するパラメータの組を決定するには、相当のさらなる計算が必要となるでしょう。

fit は信頼区間の決定よりむしろ、最後の反復後の分散-共分散行列から直ちに得られるパラメータの誤差評価を報告します。これらの評価は、標準偏差として計算される量の指定に関する統計上の条件が、一般には非線形最小自乗問題では保証されないのですが、線形最小自乗問題での標準誤差 (各パラメータの標準偏差) と同じ方法で計算されます。そしてそのため慣例により、これらは "標準誤差" とか "漸近標準誤差" と呼ばれています。漸近標準誤差は一般に楽観過ぎ、信頼レベルの決定には使うべきではありませんが、定性的な指標としては役に立つでしょう。

最終的な解は、解の範囲におけるパラメータの相関を示す相関行列も生成します: その主対角要素、すなわち自己相関は常に 1 で、全てのパラメータが独立ならば非対角要素はすべて 0 に近い値になります。完全に他を補いあう 2 つの変数は、大きさが 1 で、関係が正の相関か負の相関かによって正か負になる符号を持つ非対角要素を持ちます。非対角要素の大きさが小さいほど、各パラメータの標準偏差の評価は、漸近標準誤差に近くなります。

実用的なガイドライン (practical guidelines)

個々のデータ点への重みづけの割り当ての基礎を知っているなら、それが測定結果に対するより詳しい情報を使用させようとするでしょう。例えば、幾つかの点は他の点より当てになるということを考慮に入れることが可能です。そして、それらは最終的なパラメータの値に影響します。

データの重み付けは、最後の反復後の **fit** の追加出力に対する解釈の基礎を与えます。各点に同等に重み付けを行なうにしても、重み 1 を使うことよりもむしろ平均標準偏差を評価することが、χ 自乗が定義によりそうであるように、WSSR を 無次元変数とすることになります。

当てはめ反復の各段階で、当てはめの進行の評価に使うことが出来る情報が表示されます ('*' はより小さい WSSR を見つけられなかったこと、そして再試行していることを意味します)。'sum of squares of residuals' (残差の自乗和) は、'chisquare' (χ 自乗) とも呼ばれますが、これはデータと当てはめ関数との間の WSSR を意味していて、**fit** はこれを最小化しようとします。この段階で、重み付けされたデータによって、χ 自乗の値は自由度 (= データ点の数 - パラメータの数) に近付くことが期待されます。WSSR は補正されたχ 自乗値 (WSSR/ndf; ndf = 自由度)、または当てはめ標準偏差 (stdfit = sqrt(WSSR/ndf)) を計算するのに使われます。それらは最終的な WSSR に対してレポートされます。

データが重み付けされていなければ、stdfit は、ユーザの単位での、データと当てはめ関数の偏差の RMS (自乗平均平方根) になります。

もし妥当なデータ誤差を与え、データ点が十分多く、モデルが正しければ、補正χ 自乗値はほぼ 1 になります (詳細は、適当な統計学の本の 'χ 自乗分布' の項を参照してください)。この場合、この概要に書かれていること以外に、モデルがデータにどれくらい良く当てまっているかを決定するための追加の試験方法がいくつかあります。

補正χ 自乗が 1 よりはるかに大きくなったら、それは不正なデータ誤差評価、正規分布しないデータ誤差、システム上の測定誤差、孤立した標本値 (outliers)、または良くないモデル関数などのためでしょう。例えば **plot 'datafile' using 1:($2-f($1))** などとして残差を描画することは、それらのシステム的な傾向を知るための手がかりとなります。データ点と関数の両者を描画することは、他のモデルを考えための手がかりとなるでしょう。

同様に、1.0 より小さい補正χ 自乗は、WSSR が、正規分布する誤差を持つランダムなサンプルと関数に対して期待されるものよりも小さいことを意味します。データ誤差評価が大きすぎるのか、統計的な仮定が正しくないのか、またはモデル関数が一般的すぎて、内在的傾向に加えて特殊なサンプルによる変動の当てはめになっているのでしょう。最後の場合は、よりシンプルな関数にすればうまく行くでしょう。

当てはめの p-値は、自由度と結果のχ 自乗値に対するχ 自乗分布の累積分布関数値を 1 から引いた値です。これは、当てはめの良さのものさしを提供します。p-値の範囲は 0 から 1 までで、p-値がとても小さい、あるいはとても大きい場合は、モデルがデータとその誤差をちゃんと記述していないことを意味します。上で述べたように、これはデータに問題があるか、誤差かモデルに問題がある、またはそれらの組み合わせなのだろうと思います。p-値が小さいことは、誤差が過小評価されているので、よって最終的なパラメータ誤差をスケール変換すべきだろうということを意味するでしょう。以下も参照: **set fit errorscaling (p. 217)**。

標準的なエラーを、パラメータの不確定性に関する、あるより現実的な評価に関係付けること、および相関行列の重要性を評価することができるようになる前に、あなたは **fit** と、それを適用しようとするある種の問題に慣れておく必要があるでしょう。

fit は、大抵の非線形最小自乗法の実装では共通して、距離の自乗 (y-f(x))**2 の重み付きの和を最小化しようとすることに注意してください。それは、x の値の "誤差" を計算に関してはどんな方法も与えてはおらず、単に y に関する評価のみです。また、"孤立点" (正規分布のモデルのから外れているデータ点) は常に解を悪化させる可能性があります。

制御 (control)

fit に影響を与えるように定義できる環境変数が 2 つあります。これらの環境変数は **gnuplot** が立ち上がる前に定義しなければなりません。その設定方法はオペレーティングシステムに依存します。

```
FIT_LOG
```

は、当てはめのログが書かれるファイル名 (およびパス) を変更します。デフォルトでは、作業ディレクトリ上の "fit.log" となっています。これは、実行時にコマンド **set fit logfile** を使って上書きできます。

```
FIT_SCRIPT
```

は、ユーザが中断した後に実行するコマンドを指定します。デフォルトでは **replot** ですが、**plot** や **load** コマンドとすれば、当てはめの進行状況の表示をカスタマイズするのに便利でしょう。これは、実行時にコマンド **set fit script** を使って上書きできます。

fit の動作におけるその他多くの実行時の調整については、以下参照: **set fit (p. 217)**。

エラー処理 (error recovery)

gnuplot バージョン 6 より、コマンド **fit** は、フィッティング処理の成功失敗に関わらず、常にコマンド入力行の次に戻るようになりました。これは、fit のエラーから復帰するスクリプトを可能にします。変数 FIT_ERROR は、成功すれば 0 に、エラーの場合は 0 以外の値になります。以下の例は、5 つのデータ集合のうちどれだけ多くのものが正常に fit できても描画します。例えば 2 番目のデータ集合で失敗しても、それが 3 番目から 5 番目のデータ集合に対する fit を妨げることはありません。

```
do for [i=1:5] {
    DATA = sprintf("Data_%05d.dat", i)
    fit f(x) DATA via a,b,c
    if (FIT_ERROR || !FIT_CONVERGED) {
        print "Fit failed for ", DATA
        continue
    }
    set output sprintf("dataset_%05.png", i)
    plot DATA, f(x)
    unset output
}
```

複数の当てはめ (multi-branch)

複数当てはめ法 (multi-branch fitting) では、複数のデータ集合を、共通のパラメータを持つ複数の 1 変数の関数に、WSSR の総和を最小化することによって同時に当てはめることが出来ます。各データ集合に対する関数とパラメータ (枝) は '疑似変数' を使うことで選択できます。例えば、データ行番号 (-1; 'データ列' の番号) またはデータファイル番号 (-2) を 2 つ目の独立変数とします。

例: 2 つの指数減衰形 z=f(x) が与えられていて、それぞれ異なるデータ集合を記述しているが、共通した減衰時間を持ち、そのパラメータの値を評価する。データファイルが x:z:s の形式であったとすると、この場合以下のようにすればよい。

```
f(x,y) = (y==0) ? a*exp(-x/tau) : b*exp(-x/tau)
fit f(x,y) 'datafile' using  1:-2:2:3  via a, b, tau
```

より複雑な例については、デモファイル "fit.dem" で使われる "hexa.fnc" を参照してください。

もし従属変数のスケールに差がある場合、単位の重み付けでは 1 つの枝が支配してしまう可能性があるので、適当な重み付けが必要になります。各枝をバラバラに当てはめるのに複数当てはめ法の解を初期値として用いるのは、全体を合わせた解の各枝に対する相対的な影響に関する表示を与えることになるでしょう。

初期値 (starting values)

非線形当てはめは、大域的な最適値 (残差の自乗和 (SSR) の最小値を持つ解) への収束は保証はしませんが、局所的な極小値を与えることはできます。そのサブルーチンはそれを決定する方法を何も持ち合わせていないので、これが起こったかどうかを判断するのはあなたの責任となります。

fit は、解から遠くから始めると失敗するかも知れませんし、しばしばそれは起こり得ます。遠くというのは、SSR が大きく、パラメータの変化に対してその変化が小さい、あるいは数値的に不安定な領域 (例えば数値が大きすぎて浮動小数の桁あふれを起こす) に到達してしまって、その結果 "未定義値 (undefined value)" のメッセージか **gnuplot** の停止を引き起こしてしまうような場合を意味します。

大域的な最適値を見つける可能性を改善するには、最初の値をその解に少なくともほぼ近くに取るべきでしょう。例えば、もし可能ならば一桁分の大きさの範囲内で。最初の値が解に近いほど不正な解で終了してしまう可能性は低くなります。最初の値を見つける一つの方法は、データと当てはめ関数を同じグラフの上に描画して適当な近さに達するまで、パラメータの値を変更して **replot** することを繰り返すことです。その描画は、不正な極小値を見つけたことで当てはめが終了したかどうかをチェックするのにも有用です。

もちろん、見た目が良い当てはめが見つかっても、それは "それよりよい" 当てはめ (ある改良された当てはめの良さの基準によって特徴付けられた統計学的な意味で、あるいはそのモデルのより適切な解である、という物理的な意味で) が存在しないことの証明にはなりません。問題によっては、各パラメータの意味のある範囲をカバーするような様々な初期値の集合に対して **fit** することが望ましいかも知れません。

時刻データ (time data)

時刻データの当てはめでは、gnuplot は 1970 1 月 1 日からの秒数として時刻を表現していることを思い出すことが重要です。例えば、2023 年のある 1 日の間に計測したなんらかの時間に依存したデータに対して、2 次関数モデルでの当てはめをしたい場合、以下のようにそれができると期待するでしょう:

```
T(x) = a + b*x + c*x*x
set xdata time
fit T(x) 'hits.dat' using 1:3 via a,b,c
```

しかしそれは多分失敗します。なぜならそのある日に対応する内部での x のの値は [1.67746e+09 : 1.67754e+09] のような範囲になってしまうからです。計測データの x の小さな変更はわずかに 1.e-05 程度なので、収束を保証するためには、初期パラメータ評価で多分ずっと先の小数第何位の精度が必要になってしまうでしょう。

一つの解決策は、時間を測定の開始時間からの時間に変更して問題を作り変えてしまうことです。

```
set xdata time        # データ書式は "27-02-2023 12:00:00 計測値"
```

```
timefmt  = "%d-%m-%Y %H:%M:%S"
set timefmt timefmt
t0 = strptime( timefmt, "27-02-2023 00:00:00" )
fit T(x) 'temperature.dat' using ($1-t0):3 via a,b,c
```

これはデータの範囲を $[0 : 86400]$ に変え、より扱いやすくなります。この場合の他の方法としては、1 列目の日付を無視し、2 列目に相対時刻形式 (tH/tM/tS) を使用することです。

```
set timefmt "%tH:%tM:%tS"
fit T(x) 'temperature.dat' using 2:3 via a,b,c
```

ヒント (tips)

ここでは、**fit** を最大限に利用するためにいくつか覚えておくべきヒントを紹介します。それらは組織的ではないので、その本質がしみ込むまで何回もよく読んでください。

fit の引数の **via** には、2 つの大きく異なる目的のための 2 つの形式があります。**via "file"** の形式は、バッチ処理 (非対話型での実行が可能) で最も良く使われ、そのファイルの中で初期値を与えることができます。

via var1, var2, ... の形式は対話型の実行で良く使われ、コマンドヒストリの機構が使ってパラメータリストの編集を行い、当てはめを実行したり、あるいは新しい初期値を与えて次の実行を行なったりします。これは難しい問題に対しては特に有用で、全てのパラメータに対して 1 度だけ当てはめを直接実行しても、良い初期値でなければうまくいかないことが起こり得るからです。それを見つけるには、いくつかのパラメータのみに対して何回か反復を行ない、最終的には全てのパラメータに対する 1 度の当てはめがうまくいくところに十分近くなるまでそれを繰り返すことです。

当てはめを行なう関数のパラメータ間に共通の依存関係がないことは確認しておいてください。例えば、a*exp(x+b) を当てはめに使ってはいけません。それは a*exp(x+b)=a*exp(b)*exp(x) だからです。よってこの場合は a*exp(x) または exp(x+b) を使ってください。

技術的なお話: 絶対値が最も大きいパラメータと最も小さいパラメータの比が大きい程当てはめの収束は遅くなります。その比が、マシンの浮動小数の精度の逆数に近いか、またはそれ以上ならば、ほぼずっと収束しないか収束する前に実行が中断するでしょう。よってそのような場合は、その関数の定義で例えば 'parameter' を '1e9*parameter' にするとか、最初の値を 1e9 で割るとかしてこれを避けるように改良するか、または **set fit prescale** でパラメータの初期値に従ってそのスケール変換を内部でやらせる機能を用いるか、のいずれかが必要でしょう。

もし、関数を、当てはめるパラメータを係数とする、単純な関数の線形結合で書けるなら、それはとてもいいので是非そうしてください。何故なら、問題がもはや非線形ではないので、反復は少ない回数で収束するでしょう。もしかしたらたった一回ですむかもしれません。

実際の実験の講義ではデータ解析に対するいくつかの指示が与えられ、それでデータへの最初の関数の当てはめが行なわれます。もしかすると、基礎理論の複数の側面にひとつずつ対応する複数回のプロセスが必要かも知れませんが、そしてそれらの関数の当てはめのパラメータから本当に欲しかった情報を取り出すでしょう。しかし、**fit** を使えば、求めるパラメータの視点から直接モデル関数を書くことにより、それはしばしば 1 回で済むのです。時々はより難しい当てはめ問題の計算コストがかかりますが、データ変換もかなりの割合で避けることが出来ます。もしこれが、当てはめ関数の単純化に関して、前の段落と矛盾してると思うなら、それは正解です。

"singular matrix" のメッセージは、この Marquardt-Levenberg アルゴリズムのルーチンが、次の反復に対するパラメータの値の計算が出来ないことを意味します。この場合、別な初期値から始めるか、関数を別な形で書き直すか、より簡単な関数にしてみてください。

最後に、他の当てはめパッケージ (fudgit) のマニュアルから、これらの文書を要約するようないい引用を上げます: "Nonlinear fitting is an art! (非線形当てはめ法は芸術だ !)"

関数ブロック (function blocks)

コマンド **function** は、gnuplot コードからなる名前付きブロックのヒアドキュメント形式による定義を意味する符号で、それは関数として呼び出すことが可能です。データブロックと同様、関数 (function) ブロックの名前は '$' で始まる必要があります。その定義には、最大 9 つの名前付きの引数を指定できます。それらの名前は、その関数ブロック内で局所変数として扱われます。以下参照: **local (p. 153)**, **scope (p. 80)**。

一度関数ブロックを定義すると、それを通常の関数を呼び出すのと同じようにどこでもその名前で呼び出すことができます。返り値が適切でない場合、関数ブロックを、数式の一部としてでなく、コマンド "evaluate" で呼び出すことができます。

例:

```
function $sinc(arg) << EOF
    if (arg == 0) { return 1.0 }
    return sin(arg) / arg
EOF

gnuplot> plot $sinc(x) with lines title "sinc(x) as a function block"
```

名前付き引数の一覧は、関数ブロックの宣言時に指定する必要はありません。コマンドラインで指定した関数の引数の個数とその値は、関数ブロックの内部から、整数変数 ARGC とそれに対応する配列 ARGV[ARGC] でアクセスできます。以下参照: **ARGV (p. 134)**。これにより、可変な個数の引数を操作できる関数ブロックを定義することができます。**call** 文によるファイルの読み込みとは違い、引数は文字列変数化 (例えば ARG1) はされません。

例:

```
function $max << EOF
    local max = real("-Inf")
    if (ARGC == 0) { return NaN }
    do for [i=1:ARGC] {
        if (max < ARGV[i]) {
            max = ARGV[i]
        }
    }
    return max
EOF
gnuplot> foo = $max( f(A), 2.0, C, Array[3] )
gnuplot> baz = $max( foo, 100. )
```

関数ブロックをサポートする一番の目的は、複雑な関数を gnuplot 内部で直接定義できるようにすることです。もちろん、同じ関数を C や Fortran でコードした場合よりも実行速度は遅くなりますが、これは色々な目的に答えることを可能にします。実行速度が重要な場合は、代わりにその関数を別にプラグインとして実装すればいいでしょう (以下参照: **plugins (p. 79)**)。

関数ブロックを使う 2 つ目の目的は、gnuplot コマンドが、これを使う以外には存在できないような状況でその実行を可能にすることです。例えばあなたが 2 つの CSV ファイルからのデータを描画したいが、一つのファイルはフィールドがコンマ区切りで、もう一方はセミコロン区切りであるとします。通常この属性に対しては、

事前にコマンド **set datafile** をセットしますが、それは plot コマンドが使用するすべてのファイルに適用されてしまいます。しかし我々は、各ファイルが plot コマンドで参照される直前にそれを設定するように呼びだすような関数ブロックを定義できます。

```
function $set_csv(char) << EOF
    set datafile separator char
EOF
plot tmp=$set_csv(",") FILE1, tmp=$set_csv(";") FILE2
```

制限:

- 関数ブロック内で、データブロックや関数ブロックを定義することはできません。
- 疑似ファイル '-' は、関数ブロック内でのデータの読み込みには使用できません。
- 以下のコマンドは関数ブロック内で実行できません。**reset**, **shell**, **!<shell command>**。
- コマンド **plot**, **replot**, **splot**, **refresh**, **stats**, **vfill**, **fit** は、そのいずれもが実行中でない場合のみ、関数ブロック内でも使用できます。例えば、コマンド **plot** が呼び出す関数ブロック内で **stats** を使うことはできませんし、コマンド **fit** の中から **plot** を呼び出すことはできない等々。

関数ブロックを使う自明でない例として、複素対数ガンマ関数 lngamma に対する 15 項 Lanczos 近似の実装とグラフがデモコレクション内にあります。function_block.dem

この関数ブロックによる実装は、同じアルゴリズムで C で直接コードされている組み込み関数の lnGamma と比べて、だいたい 25 倍位遅いですが、それでも対話型での 3 次元グラフの回転には十分な位の速さです。デモにある関数定義は以下の通り。

15 項 Lanczos 近似を用いた $log\Gamma(z)$ の関数ブロックによる実装

```
array coef[15] = [ ... ]
 function $Lanczos(z) << EOD
    local Sum = coef[1] + sum [k=2:15] coef[k] / (z + k - 1)
    local temp = z + 671./128.
    temp = (z + 0.5) * log(temp) - temp
    temp = temp + log( sqrt(2*pi) * Sum/z )
    return temp
EOD
 function $Reflect(z) << EOD
    local w = $Lanczos(1.0 - z)
    local temp = log( sin(pi * z) )
    return log(pi) - (w + temp)
EOD
 my_lngamma(z) = (z == 0) ? NaN : (real(z) < 0.5) ? $Reflect(z) : $Lanczos(z)
```

Help

help コマンドは、組み込みヘルプを表示します。ある項についての説明を指定したいときには、次の書式を使って下さい:

```
help {<項目名>}
```

もし < 項目名 > が指定されなかった場合は、**gnuplot** についての簡単な説明が表示されます。指定した項目についての説明が表示された後、それに対する細目のメニューが表示され、その細目名を入力することで細目に対するヘルプを続けることができます。そして、その細目の説明が表示された後に、さらなる細目名の入力を要求されるか、または 1 つ前の項目のレベルへ戻ります。これを繰り返すとやがて、**gnuplot** のコマンドラインへと戻ります。

また、疑問符 (?) を項目として指定すると、現在のレベルの項目のリストが表示されます。

History

コマンド **history** は、コマンド履歴の一覧を表示したり、保存したり、一覧の中のコマンドを再実行したりします。このコマンドの挙動、および履歴ファイルの保存場所を変えるには、以下参照: **set history** (**p. 226**)。

history コマンドで始まる入力行は、コマンド履歴には保存しません。

例:

```
history                # 履歴全体を表示
history 5              # 履歴内の直前の 5 つを表示
history quiet 5        # エントリ番号なしで直前の 5 つを表示
history "hist.gp"      # 履歴全体をファイル hist.gp に書き出す
history "hist.gp" append # 履歴全体をファイル hist.gp に追加する
history 10 "hist.gp"   # 直前の 10 個をファイル hist.gp に出力
history 10 "|head -5 >>diary.gp" # パイプで履歴を 5 つ書き出す
history ?load          # 履歴内の "load" で始まるものすべてを表示
history ?"set c"       # 上と同様 (複数の語は引用符で囲む)
hist !"set xr"         # 上と同様 (複数の語は引用符で囲む)
hist !55               # 55 番目の履歴項目のコマンドを再実行
```

If

書式:

```
if (<condition>) { <commands>;
       <commands>
       <commands>
} else if (<condition>) {
       <commands>
} else {
```

```
        <commands>
    }
```

このバージョンの gnuplot は、if/else のブロック形式をサポートしています。キーワード **if**, **else** の後ろに開始カッコ "{" が続く場合、"}" で終了するブロックまでのすべての文 (複数の入力行も可) に条件的な実行が適用されます。if コマンドは入れ子にすることもできます。

バージョン 5 より前の gnuplot では、if/else コマンドの通用範囲は 1 行内に留まっていましたが、現在は複数行を中カッコ { } で囲んで書くことができます。古い形式も一応残されていますが、それは中カッコのブロック内で使うことはできません。

以前の書式:

```
if (<条件>) <コマンド行> [; else if (<条件>) ...; else ...]
```

キーワード **if** が "{" をともなわない場合は、< 条件 > が真 (ゼロでない) ならば < コマンド行 > のコマンド (複数も可) が実行され、偽 (ゼロ) ならばスキップされます。いずれの場合も入力行の最後になるか、**else** が現れるところまでそれが行われます。; を使うと同じ行に複数のコマンド置くことが可能ですが、条件付きのコマンド (**if** の構文自体) はそこでは終らないことに注意してください。

For

plot, **splot**, **set**, **unset** コマンドでは、繰り返しの節を使うこともできます。これは、基本的なコマンドを複数回実行する効果を持ち、そのおのおのの実行では繰り返し制御変数によって数式は再評価されます。**do** コマンドでは、どんなコマンド列でも繰り返し実行させることができます。繰り返し節は現在は以下の 2 つの形式をサポートしています:

```
for [intvar = start:end{:increment}]
for [stringvar in "A B C D"]
```

例:

```
plot for [filename in "A.dat B.dat C.dat"] filename using 1:2 with lines
plot for [basename in "A B C"] basename.".dat" using 1:2 with lines
set for [i = 1:10] style line i lc rgb "blue"
unset for [tag = 100:200] label tag
```

繰り返しの入れ子もサポートしています:

```
set for [i=1:9] for [j=1:9] label i*10+j sprintf("%d",i*10+j) at i,j
```

さらなる説明については、以下参照: **iteration (p. 69)**, **do (p. 136)**。

Import

コマンド **import** は、ユーザ定期関数名を外部共有オブジェクトから取り込まれる関数に結びつけます。これは、gnuplot で利用可能な関数の設定を拡張するプラグイン機構を構成します。

書式:

```
import func(x[,y,z,...]) from "sharedobj[:symbol]"
```

例:

```
# 関数 myfun を "mylib.so" か "mylib.dll" から取り込んで作成する
# gnuplot では描画、または数値計算で利用可能
import myfun(x) from "mylib"
import myfun(x) from "mylib:myfun"    # 上と同様

# "theirlib.so" か "theirlib.dll" で定義済の関数 theirfun を作成
# 異なる名前で利用可能
import myfun(x,y,z) from "theirlib:theirfun"
```

プログラムは共有オブジェクトとして与えられた名前に、オペレーティングシステムに従って ".so" か ".dll" を追加し、まずそれをフルパス名として検索し、次にカレントディレクトリからの相対パス名として検索します。オペレーティングシステム自体も LD_LIBRARY_PATH か DYLD_LIBRARY_PATH の任意のディレクトリを検索します。以下参照: **plugins (p. 79)**。

Load

load コマンドは、指定された入力ファイルの各行を、それが対話的に入力されたかのように実行します。**save** コマンドでつくられたファイルは、**load** することができます。有効な gnuplot コマンドの書かれたテキストファイルは、**load** コマンドによって、実行することができます。**load** 中のファイルの中にさらに **load** または **call** コマンドがあっても構いません。**load** するファイルに引数を与えるには、以下参照: **call (p. 134)**。

書式:

```
load "<入力ファイル名>"
load $datablock
```

入力ファイル名は引用符で囲まなければなりません。

load コマンドは、標準入力からのコマンドの入力のために、特別なファイル名 "-" を用意しています。これは、**gnuplot** のコマンドファイルが、いくつかのコマンドを標準入力から受け付けることを意味します。詳細については、以下参照: **batch/interactive (p. 38)**。

popen 関数をサポートするようなシステムでは、'<' で始まるファイル名にすることで、入力ファイルをパイプから読み込むことができます。

例:

```
load 'work.gnu'
load "func.dat"
load "< loadfile_generator.sh"
```

gnuplot への引数として与えられたファイル名は、暗黙のうちに **load** コマンドによって実行されます。これらは、指定された順にロードされ、その後 **gnuplot** は終了します。

内部で保存したテキストの行からコマンドを実行することも可能です。以下参照: **function blocks (p. 148)**。関数ブロックは、コマンドラインで定義するか、ファイルから読み込めます。関数ブロック内のコマンドは、**evaluate** を使うことで、あらためてファイルから読み込まなくても内部のコピーを繰り返し実行できます。

Local

書式:

```
local foo = <expression>
local array foo[size]
```

キーワード **local** は、変数の宣言を導入し、その変数の有効範囲 (scope) を、その宣言が含まれるコードブロック内の実行のみに制限します。変数宣言は必須ではありませんが、それがなければすべての変数は大域 (global) 変数となります。局所 (local) 変数の名前が大域変数と重なった場合は、局所変数の有効範囲から抜けるまでは、大域変数は隠します。以下参照: **scope (p. 80)**。

local の宣言は、**call** や **load** 文によって大域変数の値が意識せずに上書きされてしまうことを避けるために使います。これは特に関数ブロック内で有用です。コマンド **local** は、**if**, **else**, **do for**, **while** に続く中括弧のコードブロック内でも有効です。

例: データ集合の一群を描画するコマンドからなる以下のようなスクリプト"plot_all_data.gp" を書くとします。この便利なスクリプトは、"file" や"files" や "dataset" や "outfile" の名前の任意の変数が破壊される心配をせずにコマンドライン、あるいは他のスクリプトから呼び出せます。変数 "file" は本質的に局所的で、それはそれが繰り返し変数だからです (以下参照: **scope (p. 80)**) が、他の 3 つは、それらを保護するにはキーワード **local** が必要です。

plot_all_data.gp:

```
local files = system("ls -1 *.dat")
do for [file in files] {
   local dataset = file[1:strstrt(file,".dat")-1]
   local outfile = dataset . ".png"
   set output outfile
   plot file with lines title dataset
}
unset output
```

Lower

以下参照: **raise (p. 189)**。

Pause

pause コマンドは、コマンドに続く任意の文字列を表示した後、指定された時間または、改行キーが押されるまで待ちます。**pause** コマンドは、**load** 用のファイルと共に使用すると、便利になるでしょう。

書式:

```
pause <time> {"<string>"}
pause mouse {<endcondition>}{, <endcondition>} {"<string>"}
pause mouse close
```

<time> は、任意の定数または実数値の式です。**pause -1** は改行キーが押さされるまで待ち、0 を指定すると一切待たず、正の数を指定するとその秒数だけ待ちます。

使用している出力形式が **mousing** (マウス機能) をサポートしている場合、**pause mouse** は、マウスクリックがあるか ctrl-C が押されるまで待つようになります。そうでない出力形式、またはマウス機能が有効になってない場合 **pause mouse** は **pause -1** と同じです。

一つ、あるいは複数の終了条件 (endcondition) が **pause mouse** の後に与えられた場合、そのうちのどの一つでも pause は終了します。指定できる終了条件は、**keypress**, **button1**, **button2**, **button3**, **close**, **any** のいずれかです。pause がキー入力によって終了した場合、押されたキーの ASCII コードは MOUSE_KEY に保存され、文字それ自身は、1 文字の文字列値として MOUSE_CHAR に返されます。**keypress** が終了条件の一つであれば、ホットキー (キー割り当てコマンド) は無効になります。**buttons3** が終了条件の一つであれば、拡大機能は無効になります。

どの場合でもマウスの座標は変数 MOUSE_X, MOUSE_Y, MOUSE_X2, MOUSE_Y2 に保存されます。以下参照: **mouse variables (p. 78)**。

注意: **pause** コマンドは OS へのコマンドであり描画の一部ではないので、異なる出力装置では異なる動作をする可能性があります。(これは、テキストとグラフィックスが、どのように混在するかによります。)

例:

```
pause -1     # 改行キーが押されるまで待つ
pause 3      # 3 秒待つ
pause -1  "続けるには return を打ってください"
pause 10  "これは美しくないですか ? 3 次の spline です"
pause mouse "選択したデータ点上で任意のボタンをクリックしてください"
pause mouse keypress "有効なウィンドウ内で A-F の文字を入力してください"
pause mouse button1,keypress
pause mouse any "任意のキー、ボタンで終了します"
```

亜種である "pause mouse key" は、有効な描画ウィンドウ内での任意のキー入力によって再開されます。特別なキー入力まで待つようにしたい場合は、以下のようなループを使うことができます:

```
print "描画ウィンドウ内で Tab キーを打つと復帰します。"
plot <something>
pause mouse key
while (MOUSE_KEY != 9) {
    pause mouse key
}
```

Pause mouse close

コマンド **pause mouse close** は、外部イベントを待つために中断する特徴的な例の一つです。この場合、gnuplot は描画ウィンドウから "close" イベントが来るのを待ちます。デスクトップ環境、構成毎に正確にそのイベントを生成する方法は異なりますが、通常は、ウィンドウの境界にあるなんらかの部品をマウスクリックするか、<alt><F4> や <ctrl>q のようなホットキー列をタイプすることで描画ウィンドウを close できま

す。それに適切な枠の部品や、ホットキーを利用可能であるかわからない場合、gnuplot 自身の仕組みでホットキー列を定義することもできます。以下参照: **bind (p. 76)**。

以下のコマンド列は、gnuplot をコマンドラインからでなくスクリプトから実行している場合に有用です。

```
plot <...whatever...>
bind all "alt-End" "exit gnuplot"
pause mouse close
```

Pause 中の疑似マウス操作 (pseudo-mousing during pause)

dumb, **sixel**, **kitty**, **domterm** 出力形式のように、文字列出力とグラフィック表示を同じウィンドウで行う出力形式があります。これらの出力形式では、今のところ実質的にマウス動作をサポートしていませんが、コマンド **pause mouse** の間は、マウス動作が可能な出力形式の場合と同じ仕組みでキー操作を解釈します。例えば、左/右/上/下の矢印キーは、3 次元グラフでは視方向を変化させ、2 次元グラフでは視点移動/拡大の増分ステップを実行します。**l** は対数軸のトグル、**a** は現在のグラフを自動縮尺に、**h** はキー割り当ての一覧を表示します。改行キーは **pause** を終了させ、通常のコマンドライン操作に復帰します。

Plot

plot と **splot** は **gnuplot** で図を描くための基本的なコマンドです。それらは関数やデータの、多くの種類のグラフ表現を提供します。**plot** は 2 次元の関数やデータを描き、**splot** は 3 次元の曲面やデータの 2 次元投影を描きます。

書式:

```
plot {<axis-ranges>} <plot-element> {, <plot-element>, <plot-element>}
```

各描画要素 (plot-element) は、定義 (definition) か関数 (function) かデータ (data source) のいずれか 1 つに、オプションの属性、修正子などがついたものです:

```
描画要素 (plot-element):
     {<iteration>}
     <definition> | {sampling-range} <function> | <data source>
                  | keyentry
     {axes <axes>} {<title-spec>}
     {with <style>}
```

各描画要素のグラフ表現形式は、例えば **with lines** や **with boxplot** などのようにキーワード **with** で決定します。以下参照: **plotting styles (p. 90)**。

描画するデータは、1 つの関数から生成されるもの (媒介変数モード (parametric) では 2 つの関数から)、または一つのデータファイルから読み込まれるもの、または事前に定義された名前付きデータブロックから読み込まれるもの、または配列から抜き出したもの、のいずれかです。コンマで区切ることで、複数のデータファイル、データブロック、配列、関数などを 1 つの plot コマンドで描画できます。

データ描画に特有の多くの追加キーワードがあります。以下参照: **plot datafie (p. ??)**。以下も参照: **data (p. 162)**, **inline data (p. 68)**, **functions (p. 180)**。

関数、変数の定義の描画要素は、画像出力を生成しません。下の 3 つ目の例を参照してください。

例:

```
plot sin(x)
plot sin(x), cos(x)
plot f(x) = sin(x*a), a = .2, f(x), a = .4, f(x)
plot "datafile.1" with lines, "datafile.2" with points
plot [t=1:10] [-pi:pi*2] tan(t), \
     "data.1" using (tan($2)):($3/$4) smooth csplines \
              axes x1y2 notitle with lines 5
plot for [datafile in "spinach.dat broccoli.dat"] datafile
```

以下参照: **show plot (p. 319)**。

軸 (axes)

軸 (axes) は、4 種類の組が利用できます; キーワード <axes> は、特定の直線をどの軸に尺度を合わせるか、ということを選択するのに使われます。**x1y1** は下の軸と左の軸を指定; **x2y2** は上と右の軸の指定; **x1y2** は下と右の軸の指定; **x2y1** は上と左の軸の指定です。**plot** コマンドで指定された範囲は、この最初の軸の組 (下と左) にのみ適用されます。

Binary

バイナリデータファイル:

ファイル名の後ろに **binary** のキーワードを与えなければいけません。ファイル形式に関する十分詳細な情報は、ユーザがコマンドラインから与えるか、またはサポートしている **filetype** のバイナリ形式のファイルそれ自身から抜き出されるかする必要があります。バイナリファイルには、大きく 2 つの形式、binary matrix 形式と binary general 形式があります。

binary matrix 形式は、32 ビット IEEE 規格の浮動小数値 (float) が 2 次元配列の形で並び、それらの座標値を表す行と列が追加されています。plot コマンドの **using** 指定において、1 番目 (column(1)) は行列の行の座標を参照し、2 番目 (column(2)) は列の座標を参照し、3 番目 (column(3)) は、配列のそれらの座標の場所に保存されている値を参照します。

binary general 形式は、任意個の列のデータを含み、それらの情報はコマンドラインで指定する必要があります。例えば **array**, **record**, **format**, **using** などでサイズや形式、データの次元を指定できます。他にも、ファイルヘッダ読み飛ばしたり、エンディアン (endian) を変更するための有用なコマンドがありますし、配置、データの変換を行なうコマンドの組があります。それは、一様に標本化されたデータの場合、その座標がファイルには含まれないことが良くあるからです。matrix バイナリファイルやテキストデータからの入力と違うところですが、general バイナリは 1,2,3 といった **using** リストで生成される列番号を使わず、代わりに 1 列目はファイルの 1 列目、あるいは **format** リストで指定されたもの、になります。

さまざまな binary オプションに対する大域的なデフォルトの設定も可能で、それは **(s)plot <filename> binary ...** コマンドに与えるオプションと全く同じ書式で指定できます。その書式は **set datafile binary ...**

です。一般的な規則として、デフォルトのパラメータはファイルから抜き出されたパラメータで上書きされ、それはコマンドラインで指定された共通なパラメータで上書きされます。

例えば **array**, **record**, **format**, **filetype** の **binary general** 形式を特定するようなキーワードが何もついていなければ、デフォルトのバイナリ形式は **binary matrix** です。

general バイナリデータは、特別なファイル名 '-' を使ってコマンドラインから入力することもできます。しかし、これはキーボードからの入力を意図したものではなく、パイプを使ってプログラムにバイナリ形式を変換させるためのものです。バイナリデータには最後を表す記号がありませんので、gnuplot はパイプからデータを読み込む場合、**array** 指定子で指定した数の点数になるまでデータを読み込み続けます。詳細に関しては、以下参照: **binary matrix (p. 321)**, **binary general (p. 157)**。

index キーワードは、ファイルフォーマットが 1 つのファイルにつき 1 つの曲面しか許さないため、サポートされません。**every** や **using** 指定はサポートされます。**using** は、データがあたかも上の 3 つ組の形で読まれたかのように働きます。バイナリファイルの `splot` のデモ。

General

キーワード **binary** を単独で指定した場合は、非一様な格子を形成する座標情報と、各格子点での値の両方を持つバイナリデータであることを意味し (以下参照: **binary matrix (p. 321)**)、他の形式のバイナリデータの場合は、そのデータの形式を意味する追加キーワードを指定する必要があります。残念ながら、これらの追加キーワードの書式は単純ではありませんが、それでも general バイナリモードは、特に多量のデータを gnuplot に送るようなアプリケーションに取っては有用です。

書式:

```
plot '<file_name>' {binary <binary list>} ...
splot '<file_name>' {binary <binary list>} ...
```

general バイナリ形式は、ファイル構造に関する情報に関連するキーワード、すなわち **array**, **record**, **format**, **filetype** などを <binary list> 内に与えることで有効になります。それ以外の場合は、非一様な matrix バイナリ形式と見なします。(詳細に関しては、以下参照: **binary matrix (p. 321)**)

gnuplot は、例えば PNG 画像のように完全に自己記述される標準的なバイナリファイル形式の読み込み方法をいくつか知っています。その一覧は、対話画面で **show datafile binary** と入力することで参照できます。それら以外のものについては、概念上はバイナリデータはテキストデータと同様に考えることができます。各点には、**using** 指定で選択される情報の列があります。**format** 文字列を指定しなかった場合、gnuplot はバイナリ数値の数を <**using list**> で与えられる最大の列番号に等しく取ります。例えば、**using 1:3** とすると 3 列ずつデータが読み取られ、2 番目のものは無視します。各描画スタイルにはデフォルトの using 指定があります。例えば **with image** はデフォルトで **using 1** を、**with rgbimage** はデフォルトで **using 1:2:3** を使います。

Array

バイナリファイルの標本の配列の大きさを設定します。座標は gnuplot が生成してくれます。各方向の次元を表す数を指定しなければいけません。例えば **array=(10,20)** は、2 次元で最初の次元方向 (x) には 10 点、2 番目の次元方向 (y) には 20 点の標本化データがあることを意味します。ファイルの終了までデータが続くこ

とを示すのに負の値を使えます。データ次元が 1 の場合は、カッコは省略できます。複数のデータのサイズ指定を分離するのに、コロンを使うことができます。例えば **array=25:35** は 2 つの 1 次元データがファイルの中にあることを意味します。

Record

キーワード **binary record** は、バイナリファイル内のデータをどのように配列化すればよいかを示すための配列の次元を与えます。配列の各次元を指定しなければいけません。例えば、**record=(10,20)** は、データの内部構造が 2 次元で、最初 (x) の次元に 10 点並び、2 つ目 (y) の次元に 20 点並ぶことを意味します。負の数の指定は、データをファイルの最後まで読まなければいけないことを指示します。

1 次元のみの場合、かっこは省略できます。複数のレコードの次元を分離するために、コロンを使用できます。例えば、**record=25:35** は、2 つの 1 次元レコードが含まれることを示します。

このキーワードは **array** と同じ書式で、同じ機能を提供します。しかし **array** は gnuplot に座標情報を自動生成させますが、**record** はそうしません。バイナリデータファイルレコードのある列から座標を読み込むべき場合は、**record** を使用してください。

Skip

このキーワードは、バイナリファイルのある区画のスキップを可能にします。例えば、そのファイルがデータ領域の開始位置の前に 1024 バイトのヘッダを持つような場合には、以下のようにしたいと思うでしょう:

```
plot '<file_name>' binary skip=1024 ...
```

ファイルに複数のレコードがある場合、そのそれぞれに対する先頭のずらし位置を指定することができます。例えば、最初のレコードの前の 512 バイトをスキップし、2 番目、3 番目のレコードの前の 256 バイトをスキップするには以下のようにします:

```
plot '<file_name> binary record=356:356:356 skip=512:256:256 ...
```

Format

デフォルトのバイナリ形式は、単精度浮動小数 (float) が一つ、です。それをより柔軟に設定するために、この format で変数のサイズに関する詳細な情報を指定できます。例えば **format="%uchar%int%float"** は、最初の using 列として符号なし文字型変数 (unsigned char) を、2 番目の列は符号つき整数 (int) を、3 番目の列は単精度浮動小数 (float) を指定しています。もしサイズ指定子の数が最大列数より小さい場合は、残りの列の変数サイズは暗黙のうちに最後に与えた変数サイズに等しく取られます。

さらに **using** 指定同様、* 文字がついた読み捨てる列を書式に指定することもできますし、繰り返しフィールドへの回数指定によって暗黙の繰り返しを指定することもできます。例えば、**format="%*2int%3float"** は、3 つの実数データを読む前に、2 つの整数データを読み捨てます。使用できる変数サイズの一覧は、**show datafile binary datasizes** で見ることができます。それらは、それぞれのコンパイルによってそのバイトサイズとともにマシンに依存する変数名のグループと、マシンに依存しない変数名のグループに分かれています。

Blank

テキスト入力ファイルから読み込むデータ点群を分離するのに、空行を期待する描画スタイルがあります。例えば、一つの入力データ列内の一つの多角形 (polygon) の頂点群は、次の多角形の頂点群とは空行で分離します。バイナリファイルでは、実際の空行は存在しないので、このオプションにより一般のバイナリファイル内の特別なレコードを、空行であるかのように解釈できるようになります。現在サポートするオプションは **blank=NaN** のみで、これは一つのレコードの最初のフィールドが NaN の値であるときに、そのレコード全体を空行であるかのように扱います。

例:

```
plot DATA binary format="%2float" blank=NAN using 1:2 with polygons
```

Endian

ファイルのバイナリデータのエンディアンは、gnuplot が動作するプラットホームのエンディアンとは異なる場合も良くあります。いくつかの指定で gnuplot がバイトをどのように扱うかを制御できます。例えば **endian=little** は、バイナリファイルを、そのバイトの並びが小さい桁から大きい桁へ並んでいると見なされます。オプションは以下のものが使えます。

`little:`	小さい桁から大きな桁へ並ぶ
`big:`	大きな桁から小さな桁へ並ぶ
`default:`	`compiler` と同じエンディアンと見なす
`swap (swab):`	エンディアンを変更する（おかしいようならこれを使ってみてください）

gnuplot は、コンパイル時にオプションが指定されていれば、"middle" (や"pdp") エンディアンもサポートできます。

Filetype

gnuplot は、いくつか標準的なバイナリファイル形式については必要な情報をそのファイルから抜き出すことができます。例えば "format=edf" は ESRF ヘッダーファイル形式のファイルとして読み込みます。現在サポートしているファイル形式については、**show datafile binary filetypes** で見てください。

特別なファイル形式として **auto** があり、この場合 gnuplot はバイナリファイルの拡張子が、サポートされている形式の標準的な拡張子であるかをチェックします。

コマンドラインキーワードはファイルから読み取る設定を上書きするのに使われ、ファイルから読み取る設定はデフォルトの設定を上書きします。以下参照: **set datafile binary** (p. **212**)。

Avs **avs** は、自動的に認識される画像イメージに対するバイナリファイルの型の一つです。AVS は非常単純なフォーマットで、アプリケーション間でやりとりするのに最も適しています。これは、2 つの long (xwidth と ywidth) と、その後続くピクセルの列から成り、その各ピクセルは alpha/red/green/blue の 4 バイトから成ります。

Edf **edf** は、自動的に認識される画像イメージに対するバイナリファイルの型の一つです。EDF は ESRF データフォーマット (ESRF Data Format) を意味していて、それは edf と ehf の両方の形式をサポートしています (後者は ESRF Header Format)。画像の使用に関する詳しい情報は以下で見つかるでしょう:

```
http://www.edfplus.info/specs
```

Png gnuplot が png/gif/jpeg 出力用に libgd ライブラリを使うようにインストールされている場合、それらの画像形式をバイナリファイルとして読み込むこともできます。以下のような明示的なコマンド

```
plot 'file.png' binary filetype=png
```

を使うこともできますし、あらかじめ以下のように設定して、拡張子から自動的に画像形式を自動的に認識させることもできます。

```
set datafile binary filetype=auto
```

Keywords

以下のキーワード (keyword) は、バイナリファイルから座標を生成するときにのみ適用されます。つまり、binary array, matrix, image の個々の要素を特定の x,y,z の位置への配置の制御のためのものです。

Scan gnuplot がバイナリファイルをどのように走査するか、ということと実際の描画で見られる軸の方向との間の関係については多くの混乱が起こり得ます。その混乱を減らすには、gnuplot はバイナリファイルを " 常に" 点/線/面、または速い/普通/遅い、と走査すると考えるといいでしょう。このキーワードは gnuplot に、その走査の方向を描画内のどの座標方向 (x/y/z) に割り当てるかを指定します。指定は 2 つ、または 3 つの文字の並びで表現し、最初の文字が点に、次の文字が線に、3 つ目の文字が面に対応します。例えば、**scan=yx** は、最も速い走査 (点の選択) は y 方向に対応し、普通の速さの走査 (線の選択) が x 方向に対応することを意味します。

描画モードが **plot** の場合、指定には x と y の 2 つの文字を使うことができ、**splot** に対しては x, y, z の 3 つの文字を使うことができます。

割り当てに関しては、点/線/面から直交座標方向へのみに制限する内部事情は別にありません。この理由で、円柱座標への割り当てのための指定子も用意されていて、それらは直交座標の x, y, z に類似した形で t (角度), r, z となっています。

Transpose **scan=yx**、または **scan=yxz** と同じです。すなわち、これは入力時の走査行のピクセルへの割り当てに影響を与えます。表示する際に画像を転置するには、以下のようにしてみてください:

```
plot 'imagefile' binary filetype=auto flipx rotate=90deg with rgbimage
```

Dx, dy, dz gnuplot が座標を生成する場合、その間隔はこれらのキーワードで指定されたものが使用されます。例えば **dx=10 dy=20** は x 方向に 10、y 方向に 20 の間隔で標本化されたことを意味します。**dy** は **dx** がなければ使えません。同様に **dz** は **dy** がなければ使えません。もしデータの次元が指定したキーワードの次元よりも大きい場合、残りの次元方向の間隔は、指定された最も高い次元のものと同じ値が使用されます。

例えば画像がファイルから読み込まれ、**dx=3.5** のみ指定された場合、gnuplot は x 方向の間隔も y 方向の間隔も 3.5 を使用します。

以下のキーワードも座標の生成時にのみ適用されます。しかし、以下のものは matrix バイナリファイルにも使われます。

Flipx, flipy, flipz バイナリデータファイルの走査方向が gnuplot の走査方向と一致しないことがたまにあります。これらのキーワードは、それぞれ x, y, z 方向のデータの走査方向を逆向きにします。

Origin gnuplot は転置 (transpose) や反転 (flip) において座標を生成する場合、常に配列の左下の点が原点になるようにします。すなわち、データが、転置や反転の行なわれた後の直交座標系の第 1 象限に来るようにします。

配列をグラフのその他の場所に配置したい場合、**origin** キーワードで指定した場所に gnuplot は配列の左下の点を合わせます。その指定は、**plot** では 2 つの座標の組、**splot** では 3 つの座標の組を指定してください。例えば **origin=(100,100):(100,200)** は、一つのファイルに含まれる 2 つのデータに対する指定で、2 次元の描画に対する指定です。2 つ目の例として **origin=(0,0,3.5)** をあげると、これは 3 次元描画用の指定です。

Center **origin** と似ていますが、このキーワードは、配列の中心がこのキーワードで指定した点になるように配置します。例えば **center=(0,0)** のようにします。配列のサイズが **Inf** のときは center は適用されません。

Rotate 転置 (transpose) と反転 (flip) コマンドは座標の生成と座標軸の方向にある種の柔軟性を与えてくれます。しかし、角度に関する完全な制御は、2 次元の回転角を記述した回転角ベクトルを与えることにより行なうことが可能になります。

キーワード **rotate** は, **plot**, **splot** の両方で、2 次元面に対して適用されます。回転は座標平面の正の角度に関して行なわれます。

角度は、ラジアン単位ですが、pi や degrees の倍数としてのラジアンでも表現できます。例えば、**rotate=1.5708**, **rotate=0.5pi**, **rotate=90deg** はすべて同じ意味です。

origin が指定された場合、回転は平行移動の前に左下の点を中心にして行なわれます。それ以外では回転は配列の中心 (**center**) に関して行なわれます。

Perpendicular **splot** に関して回転ベクトルの設定が、ベクトルを表現する 3 つの数字の組を指定することで実装されていて、このベクトルは 2 次元の xy 平面に対して向き付けられた法線ベクトル (perpendicular) を表しています。もちろんそのデフォルトは (0,0,1) です。rotate と perpendicular の両方を指定することにより、3 次元空間内で無数の方向へデータを向き付けられることになります。

まず最初に 2 次元の回転が行なわれ、その次に 3 次元の回転が行なわれます。つまり、R' をある角による 2 x 2 の回転行列とし、P を (0,0,1) を (xp,yp,zp) へ子午線方向に回転させる 3 x 3 の行列とし、R' を左上の部分行列として持ち 3,3 成分が 1 でその他の成分が 0 であるような行列 (つまり z 軸周りの回転行列) とすれば、この変換を表す行列による関係式は v' = P R v となります。ここで、v はデータファイルから読み込まれた 3 x 1 の位置ベクトルです。ファイルのデータが 3 次元的なものでない場合は、論理的なルールが適用されて 3 次元空間内のデータと見なされます (例えば、通常は z 座標は 0 とされ、xy 平面内の 2 次元データと見なされます)。

データ (data)

ファイル内にあるデータは、**plot** コマンドライン上で、そのデータファイル名を単一引用符または二重引用符で囲んで指定することで表示できます。データはファイルではない、入力ストリームから受け取ることもできます。以下参照: **special-filenames** (p. **174**), **piped-data** (p. **176**), **datablocks** (p. **68**)。

書式:

```
plot '<file_name>' {binary <binary list>}
                     {{nonuniform|sparse} matrix}
                   {index <index list> | index "<name>"}
                   {every <every list>}
                   {skip <number-of-lines>}
                   {using <using list>}
                   {convexhull} {concavehull}
                   {smooth <option>}
                   {bins <options>}
                   {mask}
                   {volatile} {zsort} {noautoscale}
                   {if (<expression>)}
```

修正子の **binary**, **index**, **every**, **skip**, **using**, **smooth**, **bins**, **mask**, **convexhull**, **concavehull**, **zsort**, **if** をそれぞれに分けて説明します。簡単に言うと以下の通り:

- **skip N** は入力ファイルの先頭 N 行を無視するよう gnuplot に指示
- **binary** はファイルがテキストでなくバイナリデータを持つと指示
- **bins** は個々の入力点を x 軸に沿う等幅の区間に仕分け、各区間毎の単一の累積値として描画
- **every** は一つのデータ集合からどの点を描画するかを選択
- **if (<expression>)** は入力データから指定条件を満たす行のみを選択
- **index** は複数のデータ集合からどのデータ集合を描画するかを選択
- **smooth** は描画の前にデータに単純なフィルタリング、補間、曲線補間を実行
- **convexhull** と **concavehull** は単独、または with **smooth** との組で指定し、入力データ点集合をその境界多角形の頂点を構成する新しい点集合に置き換えます。
- **mask** は、事前に定義されたマスクを通して、画像内のピクセルの選択された部分集合、または pm3d 曲面の選択された領域のみを描画するようフィルタします。
- **using** はファイルのどの列をどの順で使うかを指定
- **volatile** はファイルの内容が後で再読み込みすることはできず、よって再使用のためには内部に保持しておかなければいけないことを指示
- **zsort** は入力データの各ブロックを z に関してソート

splot もよく似た書式を使いますが、いくつかのフィルタ、平滑化オプションしかサポートしていません。

キーワード **noautoscale** は、自動的に軸の範囲が決定される機能が有効である場合に、この描画を構成するデータ点については、それを無視させる (自動縮尺機能の計算対象から外す) ようにします。

テキストデータファイル:

データファイルの空でない各行は、一つのデータ点を記述しますが、# で始まる行は例外で、これはコメントとして扱われ無視されます。

描画スタイルと指定したオプションに依存し、各行から 1 つ以上 8 個以下の値を読み込み、単一のデータ点に関連付けます。以下参照: **using (p. 176)**。

データファイルの単一行中の個々の値は、ホワイトスペース (一つまたは複数の空白かタブ) か、またはコマンド **set datafile** で指定した特別なフィールド区切り文字で区切られている必要があります。フィールド全体が二重引用符で囲まれている場合、またはフィールド区切り文字りがホワイトスペース以外になっている場合は、そのフィールドがホワイトスペースを含むことが可能です。二重引用符内のホワイトスペースは列数の勘定の際には無視されるので、次のデータ行は 3 列と見なされます:

```
1.0 "second column" 3.0
```

データは、指数部に e, E の文字をつけた指数表記で書かれていても構いません。コマンド **set datafile fortran** が有効な場合は、fortran の指数指定子 d, D, q, Q も使えます。

データファイルの空行は重要です。1 行のブランク行は、**plot** に不連続を指示します; ブランク行によって区切られた点は線で結ばれることはありません (line style で書かれている場合には)。2 行のブランク行は、別々のデータ集合間の区切りを示します。以下参照: **index (p. 170)**。

もし autoscale の状態であれば (以下参照: **set autoscale (p. 196)**)、軸は全てのデータポイントを含むように自動的に引き伸ばされて、目盛りが書かれる状態ならば全ての目盛りがマークされます。これは、2 つの結果を引き起こします: i) **splot** では、曲面の角は底面の角に一致していないことがあります。この場合、縦の線は書かれることはありません。ii) 2 種類の軸での、同じ x の範囲のデータの表示の際、もし x2 の軸に対する目盛りが書かれていない場合は、x 座標があっていないことがあります。これは x 軸 (x1) は全ての目盛りにまで自動的に引き延ばされるのに対し、x2 軸はそうではないからです。次の例でその問題を見ることができます:

```
reset; plot ’-’, ’-’ axes x2y1
1 1
19 19
e
1 1
19 19
e
```

これを避けるには、**set autoscale** コマンドか **set [axis]range** コマンドの **noextend** オプションを使うことができます。これは、次の目盛りの刻みを入れるような軸の範囲の拡張機能を無効にします。

ラベルの座標と文字列もデータファイルから読み込むことができます (以下参照: **labels (p. 115)**)。

Columnheaders

コマンド plot のキーワード **skip** を使うことで、データファイルの先頭の追加行 (複数行も可) を、明示的に無視させることができます。データファイルには、文字列の列ヘッダを含む追加の単一行がある場合もあります。コマンド plot が列ヘッダ (column header) を、例えばタイトルとして使用するなどのように明示的に参照する場合は、その行を自動的にスキップします。そうでなければ、スキップ数に 1 を追加するか、**set datafile**

columnheaders の属性を設定することで明示的にその行をスキップする必要があるでしょう。以下参照: **skip** (p. **170**), **columnhead** (p. **58**), **autotitle columnheader** (p. **231**)。

コンマ区切りファイル (csv files)

書式:

```
set datafile separator {whitespace | tab | comma | "chars"}
```

"csv" は、本来はコンマ区切り ("comma-separated values") の略語ですが、ここでの「csv ファイル」 ("csv file") という言葉は、その中のデータフィールドが、必ずしもコンマである必要はない特定の文字で区切られているようなファイルとします。csv ファイルからデータを読むためには、gnuplot にフィールド区切り文字がなんであるかを示す必要があります。例えば、フィールド区切り文字としてセミコロンを使うファイルから読む場合は:

```
set datafile separator ";"
```

以下参照: **set datafile separator** (p. **211**)。これは、入力用のファイルにのみ適用されます。出力として CSV ファイルを生成するには、**set table** にオプション **separator** を使用してください。

Every

キーワード **every** は、描画するデータをデータ集合から周期的にサンプリングすることを可能にします。

通常のファイルに対しては、「ポイント」は 1 つの行、データの「ブロック」は、前後のブロックと空行で区切られる連続した行のまとまりを意味することとします。

matrix データに対しては、「ブロック」と「ポイント」は、それぞれ「行」と「列」に対応します。以下参照: **matrix every** (p. **324**)。

書式:

```
 plot 'file' every {<ポイント増分>}
                     {:{<ブロック増分>}
                       {:{<開始ポイント>}
                         {:{<開始ブロック>}
                           {:{<終了ポイント>}
                             {:<終了ブロック>}}}}}
```

描画するデータポイントは、< 開始ポイント > から < 終了ポイント > まで < ポイント増分 > の増加で選び、ブロックは < 開始ブロック > から < 終了ブロック > まで < ブロック増分 > の増加で選びます。

各ブロックの最初のデータは、ファイル中の最初のブロックと同じように、「0 番」と数えます。

プロットできない情報を含む行もカウントすることに注意して下さい。

いくつかの数字は省略できます; 増分のデフォルトは 1 、開始の値は最初のポイントか最初のブロック、そして終了の値は最後のポイントか最後のブロックに設定します。**every** のオプションが ':' で終わるのは許されていません。**every** を指定しなければ、全ての行の全てのポイントをプロットします。

例:

```
every :::3::3      # 4 番目のブロックだけ選びます (0 番が最初)
every :::::9       # 最初の 10 ブロックを選びます
every 2:2          # 1 つおきのブロックで 1 つおきのポイントを選び
                   # ます
every ::5::15      # それぞれのブロックでポイント 5 から 15 までを
                   # 選びます
```

以下も参照単純な plot デモ (simple.dem)

, 非媒介変数モードでの splot デモ

, 媒介変数モードでの splot デモ

。

データファイルの例 (example)

次の例は、ファイル "population.dat" 中のデータと理論曲線を図にするものです。

```
pop(x) = 103*exp((1965-x)/10)
set xrange [1960:1990]
plot 'population.dat', pop(x)
```

ファイル "population.dat" は次のようなファイルです。

```
# Gnu population in Antarctica since 1965
   1965   103
   1970   55
   1975   34
   1980   24
   1985   10
```

binary の例:

```
# 2 つの float の値を選択し (2 つ目の値は無意味)、一方を読み捨て、
# 一つおきの float 値を無限に長く続く 1 次元データとして使用
plot '<file_name>' binary format="%float%*float" using 1:2 with lines

# データファイルから座標を生成するのに必要な情報をすべてそのヘッ
# ダに含んでいる EDF ファイルの場合
plot '<file_name>' binary filetype=edf with image
plot '<file_name>.edf' binary filetype=auto with image

# 3 つの符号なし文字型整数値 (unsigned char) を生の RGB 画像の色
# 成分として選択し、y 方向は反転させ画像の方向を座標平面上で変更
# する (左上が原点になるように)。ピクセルの間隔も指定し、ファイ
# ルには 2 つの画像が含まれていて、そのうち一つは origin で平行
# 移動する。
plot '<file_name>' binary array=(512,1024):(1024,512) format='%uchar' \
     dx=2:1 dy=1:2 origin=(0,0):(1024,1024) flipy u 1:2:3 w rgbimage
```

```
# 4 つの別のデータからなり、座標情報もデータファイルに含まれてい
# る。ファイルは gnuplot が実行されているシステムとは異なるエン
# ディアンで生成されている。
splot '<file_name>' binary record=30:30:29:26 endian=swap u 1:2:3

# 同じ入力ファイルで、今回は 1 番目と 3 番目のレコードをスキップ
splot '<file_name>' binary record=30:26 skip=360:348 endian=swap u 1:2:3
```

以下参照: **binary matrix (p. 321)**。

フィルター (filters)

入力データを読み込んだ直後、他の smooth 処理やスタイル固有の処理オプションを適用する前に、フィルター (filters) 操作を直ちに適用します。一般にフィルターの目的は、元々の入力点集合全体を、多分変形や再グループ化、再整列化を行い、そこから抜き出した部分集合に置き換えます。現在サポートしているフィルターは **bins**, **convexhull**, **concavehull**, **if**, **mask**, **sharpen**, **zsort** です。

度数分布 (bins)

書式:

```
plot 'DATA' using <XCOL> {:<YCOL>} bins{=<NBINS>}
     {binrange [<LOW>:<HIGH>]} {binwidth=<width>}
     {binvalue={sum|avg}}
```

plot コマンドに対するオプション **bins** は、最初に元のデータを、x 軸上で等しい幅を持ついくつかの階級 (ビン) に割り当て、そして階級毎に一つの値のみを描画します。階級の数のデフォルト値は、**set samples** で決定しますが、これは plot コマンドでビンの数を明示的に指定することで変更できます。

binrange を指定しないと、範囲は 'DATA' 内の値の両端を取ります。

階級幅は、指定した範囲と階級の数から自動的に計算し、各点を 0 から NBINS-1 までの階級に割り当てます:

```
BINWIDTH = (HIGH - LOW) / (NBINS-1)
xmin = LOW - BINWIDTH/2
xmax = HIGH + BINWIDTH/2
first bin holds points with (xmin <= x < xmin + BINWIDTH)
最初の階級は (xmin <= x < xmin + BINWIDTH) の範囲の点を保持
最後の階級は (xmax-BINWIDTH <= x < xman) の範囲の点を保持
各点は i = floor(NBINS * (x-xmin)/(xmax-xmin)) 番の階級に割り当て
```

それとは別に、固定幅の階級を指定することも可能です。その場合、階級の数 (nbins) は、点の範囲全体にわたる最小の階級の数となります。

階級の出力は、その中点で描画、または表にされます。例えば gnuplot が上のように階級の幅を計算する場合、最初の階級の x 座標の出力は x=LOW であり、x=xmin ではありません。

using 命令で一つの列のみを指定した場合、各データ点は、その x 座標値に対する階級の合計値に 1 だけ寄与します。2 列目を指定すると、その階級の合計値には 2 列目の値が追加されます。よって、以下の 2 つの plot コマンドは同じになります:

```
plot 'DATA" using N bins=20
set samples 20
plot 'DATA' using (column(N)):(1)
```

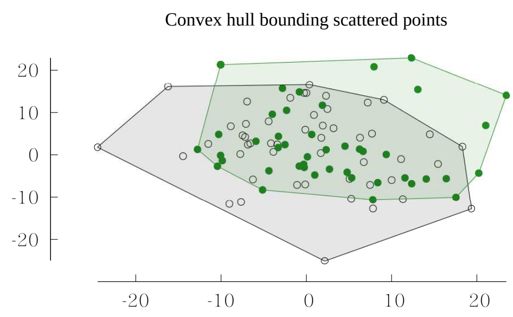
デフォルトでは、各階級に対して描画される y の値は、その階級内のすべての点に関する y の値の和になります。これは、オプション **binvalue=sum** に対応します。これに対して、**binvalue=avg** ではその階級内のすべての点に対する y の値の平均を描画します。

関連する処理オプションに関しては、以下参照: **smooth frequency (p. 173)**, **smooth kdensity (p. 174)**。

凸包 (convexhull) convexhull は描画スタイルではありません。これは、フィルターキーワードとして単独で、または **smooth path** や **expand <increment>** との組み合わせとして使います。

```
plot FOO using x:y convexhull
plot FOO using x:y convexhull smooth path
plot FOO using x:y convexhull expand <increment> {smooth path}
```

これは、FOO 内の点を、それを包含する凸多角形を構成する一意的な部分集合、すなわち凸包 (convex hull) で置き換えます。この多角形の頂点集合は、時計回りの閉曲線となるように出力します。よってその曲線の始点と終点は一致し、**lines** や **polygons**, **filledcurves** の描画スタイルで適切に描画できるようにしています。凸包は、画像や元々のデータ点すべてを含む pm3d 曲面のある領域を選択的に描画するマスクとしても便利に使えます。以下参照: **masking (p. 116)**。

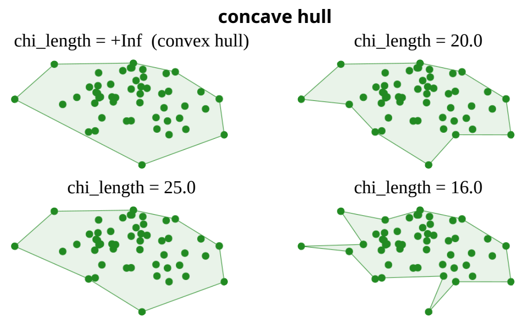


キーワード **smooth** をつけると、その頂点は滑らかな曲線を生成するためのガイド点として使います (以下参照: **smooth path (p. 172)**)。デフォルトでは、その平滑化曲線はガイド点を通ります。

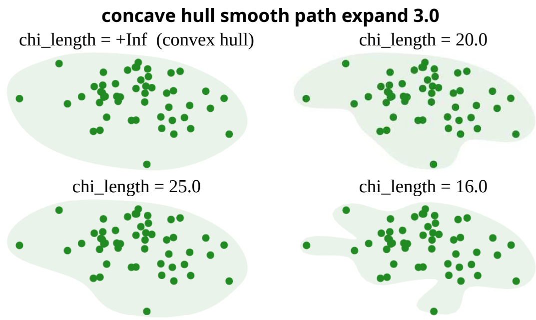
オプションのキーワード **expand** と増分値 (<increment>) は、凸包の辺の線分を増分値の距離だけ内部から遠ざけます。そして移動した線分を留め継ぎ (mitered) でつなぎます。これは、元々の凸包の各頂点を 2 つの頂点で置き変えることを意味しますが、それは隣接する辺との間に隙間ができてしまうからです。

凹包 (concavehull) あなたの gnuplot が –enable-chi-shapes 付きでビルドされた場合のみ利用可能。

凹包 (concavehull) は、描画スタイルではありません。これは、入力データ点の境界多角形「閉包」(hull) を見つけるフィルタの一つで、これは、元の点集合を、この多角形の周上の順序づけられた部分集合で置き換えます。凸包は任意の点集合に対して一意に決定しますが、これはそれとは違い、複数の凹包がありえます。そこから一つの凹包を選択する仕組みはさまざまありますが、gnuplot は Duckham ら (2008; Patttern Recognition 41:3224-3236) によって定義されたχ -形状の閉包を生成します。



与えられた点集合に対し、ドロネー三角形分割から三角形を反復削除することでχ -形状を生成します。各反復では、以下の基準に従って三角形を一つ削除します: (1) 一点で接触する境界形状の連結度を減らさない場合は削除に望ましい、(2) 三角形の一つの辺が現在の周囲の最も長い線分である場合、(3) この辺が事前に選択した、χ -形状を完全に決定する特性長パラメータより長い場合。gnuplot では、この特性長パラメータは、ユーザー変数 **chi_length** から取ります。反復は、取り除ける三角形がなくなったら停止します。**chi_length** が大きい場合、三角形は一つも取り除けず、χ -形状は元の周である凸包になります。**chi_length** を小さくするとその分多くの三角形が削除され、最終的な形状は凸性がより低くなります。小さすぎる **chi_length** は望ましくありません。



chi_length の適切な選択は、入力データ点の分布と密度に強く依存します。ユーザが **chi_length** を設定しなければ、gnuplot はそれを自動的に選択しますが、それがあなたのデータに対し適切である保証はありません。この図に示されてるデータに対しては、gnuplot は chi_length=22.6 をデフォルトで選択し、これは凸包の最長辺の 0.6 倍の長さです。デフォルトで使用されるこの最長辺に対する比率は、コマンド **set chi_shape fraction <value>** で変更できます。

現在のグラフで使った、ユーザ定義か gnuplot が選択した **chi_length** の値は、変数 GPVAL_CHI_LENGTH に保存します。

オプションキーワード **expand** と増分値 (<increment>) は、閉包の各辺を固定した距離だけ内部から遠ざけます。これは、元のすべての点の外側にある閉曲線を構成する、新たな点集合を生成します。それは、**smooth path** と組み合わせることが可能です。

マスキング (mask)

```
plot FOO using 1:2:3 mask with {pm3d|image}
```

マスクを一度定義すると、image 描画、または pm3d 描画から点の部分集合を選択するのにそれを利用できます。以下参照: **masking (p. 116)**。

先鋭化 (shapen)　フィルタ **sharpen** は、関数描画にのみ適用します。それは描画する関数の極を探しますが、それは、そのグラフを構成する線分要素を作る標本点のいずれかの x 値の上に正しく乗っているとは限りません。これは、本当の極を二分法によって、見つけそしてそれを標本点の集合に追加します。これは、ピークの鋭い先端の切り捨てを減らしますが、粗い標本化ではそれを完全には取り除けません。

例:

```
set samples 150
set xrange [-8:8]
plot abs(sqrt(sin(x))) sharpen
```

キーワード "sharpen" なしでは、結果のグラフは連続曲線で、π 毎に極小となり、その極小値は 0 に達するべきものですが、人為的に切り捨てられ、その y の極小値は見た目には 0.02 から 0.20 の間になります。キーワード "sharpen" をつけることで、その関数の正しい見た目である、周期的で鋭く y=0 に達する極小値を持つグラフを生成します。

sharpen フィルタは、階段関数にある垂直なへりのような垂直な不連続も探し、そして直ちにへりのどちらかの側に追加点を配置することで、それが対角線だと誤って伝わらないようにしようとします。

条件フィルタ (if)

```
plot ... if (<expression>)
splot ... if (<expression>)
```

入力データの各行に対し、filter **if** 内の式 (expression) が評価され、その値が真 (0 以外) の場合にのみ、その行を受け付けます。**using** 指定子内で有効な任意の関数、変数はこのフィルタの式内でも有効で、描画には使用しない入力データ列も利用できます。

式 (expression) が偽 (0) と評価したデータ列は、それがファイル内に存在しなかったものとして扱います (以下参照: **missing (p. 210)**)。これは、以前サポートしていた、**using** 指定の中で行う論理テストによる手法より、可読性の高い同等の方法を提供します。

古い書式 (まだ使えます):

```
set datafile missing NaN
plot FOO using (strcol(1) eq "ABC" ? $2 : NaN):3 with linespoints
plot $DATA using 1:($2 < 999. ? $2 : NaN)
```

新しい書式:

```
plot FOO using 2:3 with linespoints if (strcol(1) eq "ABC")
plot $DATA using 1:2 if ($2 < 999.)
```

古い例と新しい例の両方とも、入力ファイル FOO の行で最初の列が **ABC** である行、$DATA の 2 列目が 999 より小さい行のみを選択します。

このフィルタは splot コマンドでも利用可能ですが、そちらはフィルタの式が 0 (偽) を返す場合、その点を欠損データ (missing) ではなく未定義 (undefined) として扱う、というわずかな違いがあります。

Z ソート (zsort)

```
plot FOO using x:y:z:color zsort with points lc palette
```

入力データを、他の smooth オプションを適用する前に入力後直ちにソートします。他の smooth オプションがデータを再ソートして、**zsort** の効果がグラフには全く現れない場合もあることに注意してください。z 軸が自動縮尺でない場合、範囲外の z の値の点はフラグ付けされますが、削除はされません。

この機能は、とても膨大な点数データの 2 次元の散布図の表示を、高得点の分布がわかる形であり続けるようにフィルタするような使い方を意図しています。z 保証値によるソートにより、高い z 値を持つような点が、低い z 値の点で覆い隠されることはなくなります。

Index

キーワード **index** は、描画用に複数のデータ集合を持つファイルから、特定のデータ集合を選択することを可能にします。配列の添字としての index については、以下参照: **arrays (p. 64)**。

書式:

```
plot 'file' index { <m>{:<n>{:<p>}} | "<name>" }
```

データ集合は 2 行の空白で分離されています。**index <m>** は <m> 番目の集合だけを選択します; **index <m>:<n>** は <m> から <n> までのデータ集合の選択; **index <m>:<n>:<p>** は、<m>, <m>+<p>, <m>+2<p>, など、<p> おきの集合を選択し、集合 <n> で終了します。C 言語の添字 (index) の付け方に従い、index 0 はそのファイルの最初のデータ集合を意味します。大きすぎる index の指定にはエラーメッセージが返されます。<p> を指定し、<n> を空欄にした場合、<p> 毎のデータをファイルの最後まで読み込みます。**index** を指定しない場合は、ファイルのデータ全体を単一のデータ集合として描画します。

例:

```
plot 'file' index 4:5
```

ファイルの各点に対して、それが含まれるデータ集合の index 値は、疑似列 **column(-2)** で利用できます。これは、以下に見るように、そのファイル内の個々のデータ集合を区別する別の方法を提供します。これは、描画用に 1 つのデータ集合の選択しかしない場合は **index** コマンドよりも不恰好ですが、個々のデータ集合に異なる属性を割り当てたい場合にはとても便利です。以下参照: **pseudocolumns (p. 178)**, **lc variable (p. 72)**。

例:

```
plot 'file' using 1:(column(-2)==4 ? $2 : NaN)        # とても不恰好
plot 'file' using 1:2:(column(-2)) linecolor variable # とても便利 !
```

index '<name>' は、データ集合を名前 '<name>' で選択します。名前はコメント行に書いてデータ集合に割り当てます。コメント文字とそれに続く空白をそのコメント行から取り除いて、その結果が <name> から始まっていれば、それに続くデータ集合に <name> という名前がつけられて、それを指定できます。

例:

```
plot 'file' index 'Population'
```

<name> で始まるすべてのコメントがそれに続くデータ集合の名前になることに注意してください。問題を避けるために、例えば '== Popolation ==' や'[Population]' などの命名法を選択すると便利でしょう。

Skip

キーワード **skip** は、プログラムにテキストデータファイル (バイナリデータは不可) の先頭の数行をスキップさせます。スキップする行は、**every** キーワード処理での行数にはカウントしません。**every ::N** はそのファイル内のデータのすべてのブロックの先頭をスキップしますが、**skip N** はそのファイルの先頭部分の行のみをスキップすることに注意してください。バイナリデータファイルに適用される同様のオプションについては、以下参照: **binary skip (p. 158)**。

Smooth

gnuplot は、元々の入力データがそうであったかのように補間や他の操作をデータに適用するルーチンをいくつか持っています。これらは **smooth** オプション内にグループ化されています。さらに洗練されたデータ処理をしたければ、gnuplot の外でデータの前処理をするか、または適切なモデルでの **fit** を使うのがいいでしょう。以下も参照: **plot filters (p. 166)**。

書式:

```
smooth {unique | frequency | fnormal | cumulative | cnormal
        | csplines | acsplines | mcsplines  bezier | sbezier
        | path
        | kdensity {bandwidth} {period}
        | unwrap}
```

オプション **unique**, **frequency**, **fnormal**, **cumulatie**, **cnormal** は、x 座標に関してデータをソートし、そしてその x の値に対する分布のある種の様子を描画します。

spline 系、Bezeir 系のオプションは、データの端と端を結ぶ連続曲線の係数を決定します。この曲線は関数グラフと同じ方法、すなわちその値を x 座標に沿う同じ幅の区間ごとに選び (以下参照: **set samples (p. 278)**)、それらの点を線分でつなぐことで描画します。データ集合が空行や未定義値で切られている場合、切られていないそれぞれの部分を別々の連続曲線としてつなぎます。これらの別々につないだ部分同士は、曲線として切れたり、不連続になったりするかもしれません。

unwrap は、データがπ より大きなジャンプをしないように、2 π の整数倍を加える操作をします。

もし **autoscale** の状態であれば、軸の範囲は元のデータからではなく、そこから作られる最終的な曲線に対して計算されます。

もし **autoscale** の状態でなく、かつスプライン曲線を生成する場合、そのスプライン曲線の標本化は、入力データを含むような x の範囲と、**set xrange** で定義される固定した横座標の範囲の共通部分の上で行なわれます。

要求する平滑化オプションを適用するにはデータの点数が少なすぎる場合は、エラーメッセージが表示されます。

smooth オプションは、関数の描画のときには無視されます。極座標モードでは、**smooth path** のみ有効です。

3 次元 plot (splot) での平滑化は、現在は 3 次元の点の集合を通る自然 3 次スプライン曲線の生成に限定されています。一般的には、軌道 (**smooth path**) にスプラインを沿って生成します。3 次元データの 2 次元射影に対しては、**smooth csplines** はそれが 2 次元データであるかのように作用します。一つの **splot** コマンドでは、いずれか一つのキーワードのみが許されています。

```
splot $DATA using 1:2:3 smooth path with lines
```

Acsplines　オプション **smooth acsplines** は自然な滑らかなスプラインでデータを近似します。データが x に関して単調にされた後 (以下参照: **smooth unique (p. 173)**)、1 つの曲線が、いくつかの 3 次多項式の一部分により区分的に構成されます。それらの 3 次式の係数は、個々のデータ点に合うように求められますが、using 指定によって 3 列目の値が与えられた場合は、その値で個々の点に重みをつけます。デフォルトは、以下と同じです:

```
plot 'data-file' using 1:2:(1.0) smooth acsplines
```

性質上、重みの絶対的な大きさは、曲線を構成するのに使われる区分の数を決定します。もし重みが大きければ、個々のデータの影響は大きくなり、そしてその曲線は、隣り合う点同志を自然 3 次スプラインでつないで得られるものに近づきます。もし重みが小さければ、その曲線はより少ない区分で構成され、それによってより平滑的になります。その最も極端な場合はただ 1 つの区分からなる場合であり、それは全てのデータに重みの付き線形最小 2 乗近似によって作られます。誤差の立場から言えば、平滑さの重みは、その曲線に対する「平滑化因子」によって分割された各点への、統計的な重みと見ることができます。それにより、そのファイル中の (標準的な) 誤差は平滑さの重みとして使うことができます。

例:

```
sw(x,S)=1/(x*x*S)
plot 'data_file' using 1:2:(sw($3,100)) smooth acsplines
splot 'data_file' using 1:2:3:(sw($4,100)) smooth acsplines
```

splot ... smooth acsplines with lines は、連続するデータ点の x, y, z 座標にスプライン曲線を当てはめます。2 次元の場合とは違い、点を最初にソートしませんので、当てはめるスプラインの軌道はループを持つ可能性があります。警告: 一般の 3 次元の場合、たくさんのスプライン曲線が当てはまるので、似たような効果を得るには、重み値をかなり大きくしなければいけません。また、実数の経路長が暗黙の制御値として使われるので、重みづけされるその区間は、一つの軸への射影とは一致しないことに注意してください。

Bezier オプション **smooth bezier** は、n 次 (データ点の個数) のベジェ曲線でデータを近似します。この曲線は両端の点をつなぎます。

Bins **smooth bins** は **bins** と同じです。以下参照: **bins (p. 166)**。

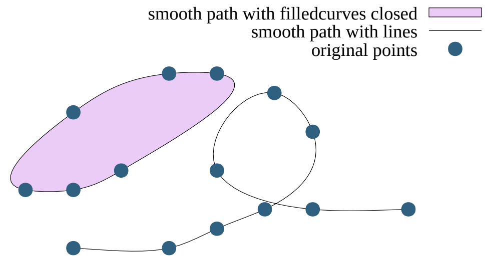
Csplines オプション **smooth csplines** はデータを x 上で単調に揃えた後で (以下参照: **smooth unique (p. 173)**) 自然 3 次スプライン曲線で引き続く点をつなぎます。その平滑化曲線は常にデータ点を通るので、よって点同士の間隔が近い場合、その滑らかな曲線に所々がたつきや遠回りができてしまうかもしれません。

splot ... smooth csplines with lines は、連続するデータ点の x, y, z 座標にスプライン曲線を当てはめます。2 次元の cspline とは違い、点を最初にソートしませんので、当てはめるスプラインの軌道はループを持つ可能性があります。一般に、スプライン係数の別々な 3 つの集合を生成し、それぞれが一つの座標 x, y, z を、共通の明示的でない軌道パラメータの関数値として扱います。これは、2 次元の **plot ... smooth path** オプションと同等です。

その曲線が xy, yz, xy のいずれかの平面上にあるような特別な場合は、単一の係数集合のみを生成します。これにより、3 次元空間内の平滑化曲線を、座標を射影した 2 次元グラフのスプライン曲線の当てはめのコピーの積み重ねで生成できるようになります。

Mcsplines オプション **smooth mcsplines** は、平滑化された関数が元の点の単調性と凸性を保存するような 3 次スプライン曲線で引き続く点をつなぎます。これは、外れ値の影響を低減します。FN Fritsch & RE Carlson (1980) "Monotone Piecewise Cubic Interpolation", SIAM Journal on Numerical Analysis 17: 238-246.

Path



オプション **smooth path** は、入力データ内で現れる順、すなわち最初に x でソートしたりせずに、3 次元スプライン曲線で引き続く点をつなぎます。これは、閉曲線や、ループを含む軌道向きの滑らかなスプラインを生成します。この平滑モードは、2 次元, 3 次元の両方の描画コマンドでサポートしています。入力ファイルの空行で区切られた各点集合に対して別々の曲線を作ります。**smooth path with filledcurves closed** による描画は、各点集合から閉曲線を描くことを保証し、**smooth path with lines** による描画は、始点と終点が同じであれば閉曲線を、そうでなければ開いた曲線を描くことを保証します。以下も参照してください。smooth_path.dem

Sbezier　オプション **smooth sbezier** は、最初にデータを単調に揃え (以下参照: **unique** (p. **173**)) そして **bezier** アルゴリズムを適用します。

Unique　オプション **smooth unique** は、データを x 方向に単調にします。同じ x を持つデータ点は y の値を平均して一つの点で置き換えます。そしてその結果として得られる点を線分で結びます。

Unwrap　オプション **smooth unwrap** は、2 つの続く点がπ を越える違いが出ないようにデータを修正します: y の値がその範囲を越えるような点に対しては、前の点との差がπ の範囲に収まるように 2 π の整数倍を加えます。この操作は、巻き戻しを持つ系の値を時間的に連続にさせるのに有用です。

Frequency　オプション **smooth frequency** は、データを x に関して単調にします。x 座標が同じ点は、それらの y の値の合計を y の値として持つ一つの点に置き換えます。多くの値のデータのヒストグラムを一定の階級幅 (bin) で描くには、それらの y の値を 1.0 にして、それでその和が同じ階級幅内の点の個数を表すようにします。これは、データ 1 列だけを指定した場合は、暗黙のうちに行なわれます。例:

```
binwidth = <適当な値>   # x の値の各階級幅
bin(val) = binwidth * floor(val/binwidth)
plot "datafile" using (bin(column(1))):(1.0) smooth frequency
plot "datafile" using (bin(column(1))) smooth frequency  # 同上
```

以下も参照 smooth.dem

Fnormal　オプション **smooth fnormal** は、オプション **frequency** と同様の動作をしますが、正規化したヒストグラムを生成します。すなわち、データを x に関して単調にして、y の値はそのすべての和が 1 になるように正規化します。x 座標が同じ点は、それらの y の値の合計を y の値として持つ一つの点に置き換えます。多くの値のデータのヒストグラムを一定の階級幅 (bin) で描くには、それらの y の値を 1.0 にして、それでその和が同じ階級幅内の点の個数を表すようにします。これは、データ 1 列だけを指定した場合は、暗黙のうちに行なわれます。以下も参照 smooth.dem

Cumulative　オプション **smooth cumulative** は、データを x に関して単調にします。x 座標が同じ点は、それ以下の x の値を持つすべての点 (すなわち現在のデータ点の左側の点) に対する y の値の累積的な合計を

y の値として持つ一つの点に置き換えられます。これは、データから累積分布関数を得るのに利用できます。以下も参照 smooth.dem

Cnormal オプション **smooth cnormal** は、x に関して単調で、y の値は [0:1] に正規化されたデータを生成します。同じ x の値を持つ点が複数ある場合は、それより小さい x の値を持つすべてのデータ点 (すなわち現在のデータ点よりも左にある点) の累積和を、すべての y の値の和で割った値を y の値として持つような一点のデータに置き変えられます。これは、データから正規化された累積分布関数を得るのに使えます (特に標本点数の異なるデータ集合を比較するのに有用です)。以下も参照 smooth.dem

Kdensity オプション **smooth kdensity** は、描画された値集合の分布に対するガウス核による核密度評価を生成し描画します。値は最初のデータ列から取り、オプションとして重みを第 2 列から取ります。ガウス核は、各点の位置に置かれ、これらのガウス核すべての和が関数として描画されます。正規化されたヒストグラムを得るには、各重みを 1/(点の個数) とすべきです。

バンド幅: デフォルトでは、gnuplot は正規分布のデータ値に対して最適となるようなバンド幅を計算し使用します。

```
default_bandwidth = sigma * (4/3N) ** (0.2)
```

これは通常はとても保守的で、すなわち幅広いバンド幅です。バンド幅は、明示的に指定することもできます。

```
plot $DATA smooth kdensity bandwidth <value> with boxes
```

前の描画で使用したバンド幅は、GPVAL_KDENSITY_BANDWIDTH に保存します。

周期: 周期的なデータに対しては、個々のガウス核成分は、1 周期の区間を繰り返すように扱う必要があります。その一例は、角の関数として測定されたデータで、これは周期が 2π です。他の例は、複数年に渡って、各年の日付毎に取られたデータで、この周期は 365 です。このような場合、その周期を描画コマンドで渡す必要があります。

```
plot $ANGULAR_DAT smooth kdensity period 2*pi with lines
```

特別なファイル名 (special-filenames)

特別な意味を持つファイル名として、次のものがあります: '', '-', '+', '++'

空のファイル名 '' は、同じ plot コマンド上で、直前の入力ファイルを再び使用することを gnuplot に指示します。よって、同じ入力ファイルの 2 つのデータ列を描画するには以下のようにします:

```
plot 'filename' using 1:2, '' using 1:3
```

この filename は、この後の plot コマンドでも '' で再利用できますが、その場合に save すると、コメントとしてその名前を記録するのみです。

'+' と '++' という特別なファイル名は、**using** 指定の全体と描画スタイルにインライン関数を使えるようにするための仕組みです。通常、関数描画はサンプル点毎に単一の y (または z) の値しか持てません。しかし疑似ファイル'+' はそれがあたかも実際の入力ファイルであるように、**using** 指定による 1 列目の値を標本点として扱い、さらに追加の列の値を指定することも可能です。標本点数は **set samples** かまたは明示的に標本間隔を範囲指定部分に指定することで制御できます。標本点は、**set trange** で設定されていればそれで設定した範囲全体に渡りますが、そうでなければ **set xrange** の範囲全体に渡ります。

注意: trange の使用は、以前のある版の gnuplot の仕様とは異なりますが、これにより x 軸の範囲とは異なる標本範囲が使用できることになります。

```
plot '+' using ($1):(sin($1)):(sin($1)**2) with filledcurves
```

'+' の直前に、独立な標本範囲を指定することもできます。通常の関数描画のと同様、独立変数に名前を割り当てることもできます。以下は、標本間隔 (1.5) を標本範囲内に指定した例です。標本点は、-3, 1.5, 0, 1.5, ... 24 の位置で取ります。

```
plot $MYDATA, [t=-3:25:1.5] '+' using (t):(f(t))
```

さらに、+ の範囲指定には、標本増分を与えることもできます。**plot** か **splot** コマンドのすぐ後に標本範囲を書くと, 間違って x 軸の範囲との解釈されてしまうかもしれません。そのあいまいさを避けるために、キーワード **sample** を前置するか、または標本範囲の 3 番目のフィールドとして標本間隔を与えるかすればいいでしょう。その標本間隔は空でもよく、その場合はデフォルトを使用します。そのようなあいまいさの例については、以下参照: **plot sampling (p. 182)**。

```
plot sample [beta=0:2*pi] '+' using (sin(beta)):(cos(beta)) with lines
plot        [beta=0:2*pi:] '+' using (sin(beta)):(cos(beta)) with lines
```

疑似ファイル '++' は、u 方向は **set samples** で制御される点の数、v 方向は **set isosamples** で制御される点の数の、標準的な [u,v] 座標の格子を生成する 2 列のデータを返します。よって、'++' の描画の前に、urange と vrange を設定する必要がありますが、x と y の範囲は自動的に設定されるか、または明示的に urange, vrange とは違う値に設定できます。例:

```
splot '++' using 1:2:(sin($1)*sin($2)) with pm3d
plot '++' using 1:2:(sin($1)*sin($2)) with image
```

'-' という特別なファイル名は、データがインラインであることを指示します。すなわち、データをコマンドの後に続けて指定します。このときはデータのみがコマンドに続き得ます。よって、**plot** コマンドに対するフィルター、タイトル、ラインスタイルといったオプションは、**plot** のコマンドラインの方に書かないといけません。これは、unix シェルスクリプトにおける << (ヒアドキュメント) と同様です。そのデータは、それらがファイルから読み込まれたかのように、1 行につき 1 つのデータ点を入力します。そしてデータの終りは、1 列目の始めに文字 "e" を置くことで指示します。

'-' は、データとコマンドを一緒に持つことが有用である場合のためにあります。例えば、別々のアプリケーションから **gnuplot** にその両方がパイプ入力される場合です。例えば、デモファイルの中にはこの機能を使うものがあるでしょう。**index** や **every** のような **plot** のオプションが与えられていると、それらは使われることのないデータの入力を要求してきます。ごく単純な場合を除くすべての場合で、**'-'** からデータを読み込むよりも、最初にデータブロックを定義してそれを読み込む方が多分簡単です。以下参照: **datablocks (p. 68)**。

もし、**replot** コマンドで **'-'** を使うなら、あなたは 1 度以上データを入力する必要があるでしょう。以下参照: **replot (p. 190)**, **refresh (p. 190)**。繰り返しますが、データブロックを使う方がいいです。

空のファイル名 ('') は、直前のファイル名が再び使われることを指示します。これは、

```
plot 'ある/とても/長い/ファイル名' using 1:2, '' using 1:3, '' using 1:4
```

のようなときに便利です。もし同じ plot コマンド上で、**'-'** と '' の両方を使用すると、インラインデータの 2 つの集合を与える必要があり、一つ目のものを再利用することはできません。

パイプによる入力データ (piped-data)

popen 関数を持っているシステム上では、データファイルは、'<' で始まるファイル名によって、シェルコマンドからパイプ入力することができます。例えば

```
pop(x) = 103*exp(-x/10)
plot "< awk '{print $1-1965, $2}' population.dat", pop(x)
```

は、最初の人口の例と同じ情報を描画します。ただし、x 座標は 1965 年からの経過年を表すようになります。この例を実行するときは、上のデータファイルのコメント行をすべて削除しなければなりませんが、または上のコマンドの最初の部分を次のように変えることもできます (コンマに続く部分):

```
plot "< awk '$0 !~ /^#/ {print $1-1965, $2}' population.dat"
```

このアプローチは最も柔軟性がありますが、**using** キーワードを用いた単純なフィルタリングで行うことも可能です。

fdopen() 関数を持つシステムでは、データを、ファイルかパイプに結びつけられた任意のファイルデスクリプタから読み込むことができます。**n** 番のファイルデスクリプタから読み込むには、**'<&n'** としてください。これにより、1 回の POSIX shell からの呼び出しの中で、複数のデータファイルからのパイプ入力が容易に行えるようになります:

```
$ gnuplot -p -e "plot '<&3', '<&4'" 3<data-3 4<data-4
$ ./gnuplot 5< <(myprogram -with -options)
gnuplot> plot '<&5'
```

Using

最もよく使われるデータファイルの修飾子は **using** で、これは入力ファイルのどの行を描画するのかを指示します。

書式:

```
plot 'file' using <entry> {:<entry> {:<entry> ...}} {'format'}
```

各 <entry> は、入力ファイルの一つのフィールドを選択するための単なる列の番号か、一つのデータ集合の最初の行の列のラベルに一致する文字列、カッコで囲まれた数式、xticlabels(2) のようにカッコで囲まない特別な関数、のいずれかです。

そのエントリがカッコで囲まれた数式の場合、N 列目の値を指定するのに関数 column(N) を使用できます。つまり、column(1) は読み込まれた最初の項目を参照し、column(2) は次の項目、といった具合です。column(1), column(2), ... の略記として、特別な記号 $1, $2, ... を使用できます。

特別な記号 $# は、現在の入力行の全列数と評価しますので、column($#) や stringcolumn($#) は、列数が不明な場合、あるいは行によって異なる個数の列を含むファイルからの入力の場合でも、必ず最終列の内容を返します。

関数 **valid(N)** で、N 番目の列が有効な数字であるかどうかテストできます。その列の値が欠けていたり、解釈できなかったり、NaN の場合は 0 を返します。入力ファイルの最初の行の各列に、データの値ではなくラベルを持っている場合、このラベルを入力列の特定や plot タイトルに使用できます。関数 column() は、列番号以外にラベルで入力列を選択できます。例えば、データファイルが以下のような場合:

```
Height     Weight     Age
val1       val1       val1
...        ...        ...
```

以下の plot コマンドは同じ意味になります:

```
plot 'datafile' using 3:1, '' using 3:2
plot 'datafile' using (column("Age")):(column(1)), \
                '' using (column("Age")):(column(2))
plot 'datafile' using "Age":"Height", '' using "Age":"Weight"
```

指定文字列が完全に一致する必要がありますし、大文字小文字も区別します。列のラベルを plot タイトルに使うには、**set key autotitle columnhead** とするか、または個別にタイトルを指定する場合は関数 **columnhead(N)** を使ってください。

入力データファイルの 1...N という実際の列に加えて、gnuplot は管理情報を持ついくつかの "疑似列" を提供します。例えば、$0 または column(0) は、データ集合内のそのデータ行の行番号を返します。以下参照: **pseudocolumns (p. 178)**。

<entry> に何も書かなければ、そのエントリのリストの順にデフォルトの値が使われます。例えば **using ::4** は、**using 1:2:4** と解釈されます。

using にただ一つのエントリを指定した場合は、その <entry> は y の値として使われ、データ点の番号 (疑似列 $0) が x として使われます。例えば"**plot 'file' using 1**" は "**plot 'file' using 0:1**" と同じ意味です。**using** に 2 つのエントリを与えた場合、それらは x, y として使われます。さらにエントリを追加して、入力からのデータを利用するような描画スタイルの詳細については、以下参照: **set style (p. 279)**, **fit (p. 137)**。

Format format を指定すると、それを C ライブラリ関数 'scanf' に適用してデータファイルの各行を読みます。そうでなければ、各行はホワイトスペース (スペースやタブ) で区切られたデータの列 (フィールド) からなるとみなしますが以下も参照: **datafile separator (p. 211)**。

'scanf' 関数では色々なデータ形式の数値入力が使えますが、**gnuplot** は全ての入力データを倍精度浮動小数とみなしますから、**gnuplot** では **%lf** が本質的に唯一の数値入力指定、ということになります。その書式文字列には少なくとも一つ、そして 7 つ以下の、そのような入力指定子を入れる必要があります。'scanf' は数と数の間にホワイトスペース、すなわち空白、タブ ("\t")、改行 ("\n")、または改ページ ("\f") があると期待します。それ以外の入力は明らかにスキップしなければいけません。

"\t", "\n", "\f" を使うときは単一引用符よりむしろ二重引用符を使うべきであることに注意してください。

Using の例 (using_examples) 次の例は、1 番目のデータに対する 2 番目と 3 番目の和の値を plot します。書式文字列は、各列データがスペース区切りでなく、コンマ区切りであることを指示していますが、同じことが **set datafile separator comma** を指定することでも可能です。

```
plot 'file' using 1:($2+$3) '%lf,%lf,%lf'
```

次の例は、より複雑な書式指定でデータをファイル "MyData" から読み込みます。

```
plot 'MyData' using "%*lf%lf%*20[^\n]%lf"
```

この書式指定の意味は以下の通りです:

%*lf	数値を無視

`%lf`	倍精度浮動小数を読み込む（デフォルトでは `x` の値）
`%*20[^\n]`	20 個の改行以外の文字を無視
`%lf`	倍精度浮動小数を読み込む（デフォルトでは `y` の値）

3 項演算子 **?:** を使ってデータをフィルタする一つの芸当を紹介します。

```
plot 'file' using 1:($3>10 ? $2 : 1/0)
```

これは、1 列目のデータに対して、3 列目のデータが 10 以上であるような 2 列目のデータを plot します。**1/0** は未定義値であり、**gnuplot** は未定義の点を無視するので、よって適切でない点は隠されることになります。または、あらかじめ定義されている値 NaN を使っても同じことになります。

カッコで始まっていない限りは定数式を列番号として使うことができます。例えば **using 0+(複雑な式)** の様なことができます。そして、その数式は、カッコでスタートしていなければ数式の値が一度評価され、カッコでスタートしていれば個々のデータ点を読み込むためにその値が一度評価される、という点が重要です。

時系列フォーマットデータを使っている場合、その時間のデータは複数の列に渡らせることができます。その場合、他のデータの開始位置を計算するとき、時間のデータに空白が含まれていることに注意してください。例えば、データ行の最初の要素がスペースが埋め込まれた時間データであるならば、y の値は 3 列目の値として指定されるべきです。

(a) **plot 'file'** と、(b) **plot 'file' using 1:2**、そして (c) **plot 'file' using ($1):($2)** には微妙な違いがあることに注意してください。以下参照: **missing (p. 210)**。

最初に単に

```
plot 'file' using 1:2
```

と指定することで、大抵の場合どんなにゴミのデータを含む行を持つファイルをも plot することが可能になります。しかし、どうしてもデータファイルに文字列を残しておきたいならば、そのテキスト行の第一列にコメント文字 (#) を置く方がより安全でしょう。

疑似列 (pseudocolumns)　plot 文の **using** 項目内の式では、入力ファイルに含まれる実際のデータ値に加えて管理情報も参照でき、これらは "疑似列" (pseudocolumns) に含まれています。

`column(0)`	データ集合内での各点の順番。順番は 0 から始まり、空行やコメント行でない行で増え、2 行の連続する空行でリセットされます。非一様な `matrix` データ形式では、`column(0)` は各 `matrix` 要素の線形順序になります。略記 `$0` も使用可。
`column(-1)`	この番号は 0 から始まり、1 行の空行で増え、2 行の連続する空行でリセットされます。これは、行列、または格子状データ内のデータ行に対応します。また、データ集合内の別々の線分や多角形を区別するのにも使えます。
`column(-2)`	0 から始まり、2 行の連続する空行で増えます。これは、複数のデータ集合を持つファイル内の、現在のデータ集合の `index` 番号です。以下参照: `'index'`。
`column($#)`	特別な記号 `$#` は、存在する全列数と評価されますので、よって `column($#)` は、現在の入力行の最終列（最右列）

を参照します。同様に `column($# - 1)` は、最終列の一つ
手前の列、などとなります。

配列 (arrays) 描画するデータが配列か、または配列値関数である場合、**using** 指定の中の"列 (column)" は以下のように解釈します。詳細については以下参照: **arrays** (p. **64**)。

1 列目	配列の添字
2 列目	配列の数値要素の実数部分、または文字列配列の文字列値
3 列目	配列の数値要素の虚数部分

Key ある描画スタイル (列積み上げ型ヒストグラムやクモの巣グラフ) では、データ列の先頭から描画タイトルを生成することが無意味なレイアウトになっています。(**using 2:3:xticlabels(1)** のように) データ列の内容から軸目盛りラベルを生成することも無意味になっています。それらの描画スタイルでは、代わりに **using 2:3:key(1)** の形式を使ってデータ列に含まれる文字列から凡例 (key) の描画タイトルを生成します。通常は、行の先頭の第 1 列目を取ります。**spiderplot** で提示されている例を参照してください。

Xticlabels 軸の刻みの見出し (ticlabel) は文字列関数によって作ることもでき、それは通常は引数としてデータ列から取得します。最も単純な形式は、データ列自身の文字列としての利用で、xticlabels(N) は xticlabels(stringcolumn(N)) の省略形として使えます。以下の例は 3 列目の要素を x 軸の刻みの見出しとして使用します。

```
plot 'datafile' using <xcol>:<ycol>:xticlabels(3) with <plotstyle>
```

軸の目盛りの見出しは、任意の描画軸 x,x2,y,y2,z 用に生成できます。**ticlabels(<labelcol>)** 指定は、**using** 指定の中で、そのデータの座標指定が全て済んだ後に行う必要があります。有効な X,Y[,Z] 座標の組を持つ各データ点に対して、xticlabels() に与える文字列値は、それに対応する点の x 座標と同じ場所の x 軸の見出しのリストに追加されます。**xticlabels()** は **xtic()** と省略することもでき、他の軸に関しても同様です。

例:

```
splot "data" using 2:4:6:xtic(1):ytic(3):ztic(6)
```

この例では、x 軸、y 軸の見出しは x,y 座標値とは別の列から取り出されますが、z 軸の見出しは、対応する点の z 座標値から生成されます。

例:

```
plot "data" using 1:2:xtic( $3 > 10. ? "A" : "B" )
```

この例は、x 軸の見出しの生成に文字列値関数を使用したもので、データファイルの各点の x 軸の刻みの見出しは、3 列目の値によって "A" か "B" かのいずれかとなります。

X2ticlabels 以下参照: **plot using xticlabels** (p. **179**)。

Yticlabels 以下参照: **plot using xticlabels** (p. **179**)。

Y2ticlabels 以下参照: **plot using xticlabels** (p. **179**)。

Zticlabels 以下参照: **plot using xticlabels** (p. **179**)。

Volatile

plot コマンドのキーワード **volatile** は、入力ストリームかファイルから以前に読み込んだデータが、再読み込み時には有効ではないことを意味します。これは、**replot** コマンドの代わりに、可能な限り **refresh** コマンドを使うよう gnuplot に指示します。以下参照: **refresh** (p. **190**)。

関数描画 (functions)

コマンド **plot**, **splot** では、ファイルから読み込んだデータの描画だけでなく、組み込み関数やユーザ定義関数を描画することもできます。関数の値は、独立な軸の通常の範囲に渡ってデータサンプルを取ることで評価します。以下参照: **set samples** (p. **278**), **set isosamples** (p. **227**)。例:

```
approx(ang) = ang - ang**3 / (3*2)
plot sin(x) title "sin(x)", approx(x) title "approximation"
```

関数のデフォルトの描画スタイルを設定する方法については、以下参照: **set style function** (p. **284**)。組み込み関数の情報については、以下参照: **expressions functions** (p. **48**)。自前で関数を定義する方法については、以下参照: **user-defined** (p. **63**)。

媒介変数モード描画 (parametric)

媒介変数モード (**set parametric**) では、**plot** では 2 つの数式の組を、**splot** では 3 つの数式の組を与える必要があります。

例:

```
plot sin(t),t**2
splot cos(u)*cos(v),cos(u)*sin(v),sin(u)
```

データファイルは前と同じように描画されます。ただし、データファイルが描画のために与えられる前に、任意の媒介変数関数が先に完全に指定された場合を除いてです。言い換えると、x の媒介変数関数 (上の例では **sin(t)**) と y の媒介変数関数 (上の例では **t**2**) との間に、他の修飾子やデータ関数をはさみこんではいけません。そのようなことをすると、構文エラーになり、媒介変数関数が完全には指定されていない、と表示されます。

with や **title** のような他の修飾子は、媒介変数関数の指定が完了した後に指定しなければいけません。

```
plot sin(t),t**2 title 'Parametric example' with linespoints
```

以下も参照媒介変数モードのデモ。

範囲 (ranges)

このセクションでは、コマンド **plot**, **splot** の一番最初の項目として書く、軸の範囲のオプションについてのみ説明します。これを指定すると、その範囲は、それ以前のどの **set range** による範囲の制限よりも優先して扱われます。コマンド **plot** の別な場所に指定する、個々の描画要素の範囲の制限ためのオプションについては以下参照: **sampling (p. 182)**。

書式:

```
[{<dummy-var>=}{{<min>}:{<max>}}]
[{{<min>}:{<max>}}]
```

1 つ目の形式の範囲指定は独立変数の範囲 (**xrange**、または媒介変数モードでの **trange**) 用で、2 つ目の形式は従属変数の範囲用です。オプションの <dummy-var> で独立変数の新しい名前を利用できます (デフォルトの変数名は **set dummy** で変更できます)。

媒介変数モード (parametric) でなければ、範囲指定は以下の順に与えなければいけません:

```
plot [<xrange>][<yrange>][<x2range>][<y2range>] ...
```

媒介変数モード (parametric) では、範囲指定は以下の順に与えなければいけません:

```
plot [<trange>][<xrange>][<yrange>][<x2range>][<y2range>] ...
```

以下の **plot** コマンドは、**trange** を [-pi:pi], **xrange** を [-1.3:1.3], **yrange** を [-1:1] に設定する例です:

```
plot [-pi:pi] [-1.3:1.3] [-1:1] sin(t),t**2
```

* は、min (最小値) や max (最大値) に自動範囲指定 (autoscale) の機能を使うことを可能にします。指定順番のためだけに必要な範囲指定には、空の範囲 [] を使ってください。

plot や **splot** のコマンド行で指定された範囲はそのグラフ一つにのみ影響を及ぼします。よって、その後のグラフのデフォルトの範囲を変更するには **set xrange** や **set yrange** を使用してください。

リンクされた軸に対しては、plot コマンドでの一時的な範囲指定の使用は、期待する結果を生まないかもしれません (以下参照: **set link (p. 239)**)。

時間データに対しては、範囲は、データファイルから読み込むのに使用するのと同じ書式で、引用符で囲んで指定する必要があります。以下参照: **set timefmt (p. 295)**。

例:

以下は現在の範囲を使用します:

```
plot cos(x)
```

以下は x の範囲のみの指定です:

```
plot [-10:30] sin(pi*x)/(pi*x)
```

以下は上と同じですが、仮変数として t を使います:

```
plot [t = -10 :30]  sin(pi*t)/(pi*t)
```

以下は x と y の両方の範囲の指定です:

```
plot [-pi:pi] [-3:3]  tan(x), 1/x
```

以下は、y の範囲のみの指定です:

```
plot [ ] [-2:sin(5)*-8] sin(x)**besj0(x)
```

以下は x の最大値と y の最小値のみの指定です。

```
plot [:200] [-pi:]  $mydata using 1:2
```

以下は x の範囲を時系列データとして指定しています:

```
set timefmt "%d/%m/%y %H:%M"
plot ["1/6/93 12:00":"5/6/93 12:00"] 'timedata.dat'
```

サンプリング (sampling)

1 次元のサンプリング (x または t 軸) (1D sampling)

デフォルトでは、計算対象の関数に対して、描画範囲全体にわたって標本 (サンプル) を取ります。この範囲は、事前にコマンド **set xrange** で設定するか、plot コマンドの一番最初の場所で x 軸の範囲指定をするか、またはそのグラフのすべての要素を含む範囲内にデータ全体が入るように自動縮尺 (autoscaling) された x の範囲です。疑似ファイル "+" が生成する標本点は、t 軸の現在の範囲全体に渡りますが、それは x 軸の範囲と同じでも同じでなくても構いません。

個々の描画要素毎に対し、標本範囲をさらに制限して割り当てることもできます。

例:

以下は、x 全体の範囲を 0 から 1000 としてファイルのデータを描画し、2 つの関数を全体の範囲の一部分だけそれぞれ描画します:

```
set xrange [0:1000]
plot 'datafile', [0:200] func1(x), [200:500] func2(x)
```

以下は、上とほぼ同様ですが、全体の範囲はデータファイルの内容によって決定します。この場合、標本化される関数は、全体がグラフ内に収まるかもしれませんし、収まらないかもしれません:

```
set autoscale x
plot 'datafile', [0:200] func1(x), [200:500] func2(x)
```

以下のコマンドはあいまいです。先頭の範囲 [0:10] は、多分最初の関数の標本化のみに向けたのだと思いますが、実際には、グラフ全体に適用され、事前の xrange の定義も無視されてしまいます:

```
set xrange [0:50]
plot [0:10] f(x), [10:20] g(x), [20:30] h(x)
```

上の例のあいまいさを除くためには、キーワード **sample** を挿入することで [0:10] が軸の範囲ではなく標本範囲であることを指示するか、または範囲指定内に 2 番目のコロンに続く標本増分フィールドを追加することです。後者は、軸範囲としては完全な範囲指定 [min:max:increment] は解釈されないために機能します。増分フィールドが空の場合は、増分はデフォルトの (max-min) を標本点数で割った値となり、よって以下の 3 つの変種は同じ結果を生じます。

```
set samples 100
plot sample [0:10]     f(x), [10:20] g(x), [20:30] h(x)
plot        [0:10:0.1] f(x), [10:20] g(x), [20:30] h(x)
plot        [0:10:]    f(x), [10:20] g(x), [20:30] h(x)
```

以下の例は、3 次元グラフにらせんの曲線を描く一つの方法を提示します:

```
set xrange [-2:2]; set yrange [-2:2]
set angle degrees
splot [phi=1:720:2] '+' using (cos(phi)):(sin(phi)):(phi)
```

2 次元のサンプリング (u と v 軸) (2D sampling)

計算する関数や疑似ファイル '++' に対して生成するデータは、u, v 軸に沿って標本化 (サンプリング) を行います。以下参照: **special-filenames ++ (p. 174)**。2 次元のサンプリングは **plot**, **splot** コマンドで使用できます。

以下は、2 次元の **plot** コマンドに対する 2 次元サンプリングの例です。これは、描画スタイル **with vectors** で表示されるグラフを生成します。以下参照: **vectors (p. 124)**。

```
set urange [ -2.0 : 2.0 ]
set vrange [ -2.0 : 2.0 ]
plot '++' using ($1):($2):($2*0.4):(-$1*0.4) with vectors
```

以下は 3 次元の **splot** コマンドに対する 2 次元サンプリングの例です。これは、**sampling.dem** で使用されているものに似たコマンド列です。この 2 つの曲面は、生成されるグラフの範囲全体よりも狭い u, v の範囲で標本化を行うことに注意してください。

```
set title "3D sampling range distinct from plot x/y range"
set xrange [1:100]
set yrange [1:100]
splot sample [u=30:70][v=0:50] '++' using 1:2:(u*v) lt 3, \
      [u=40:80][v=30:60] '++' using (u):(v):(u*sqrt(v)) lt 4
```

u, v のサンプリングの範囲指定には、サンプリングデータの数とスペースを制御する、明示的なサンプリング間隔を入れることもできます:

```
splot [u=30:70:1][v=0:50:5] '++' using 1:2:(func($1,$2))
```

Plot コマンドの for ループ (for loops in plot command)

多くの同等のファイルや関数を同時に描画する場合は、それぞれの plot コマンドの繰り返し (iteration) でそれを行うのが便利です。

書式:

```
plot for [<variable> = <start> : <end> {:<increment>}]
plot for [<variable> in "string of words"]
```

繰り返しの適用範囲 (scope) は、次のコンマ (,) かコマンドの終わり、のいずれか先に現れたところまでです。ただし、描画する項目の前に定義式 (複数も可) が並んでいる場合は、コンマが間に入っていてもその例外となります。繰り返しは媒介変数モード (parametric) では機能しないことに注意してください。

例:

```
plot for [j=1:3] sin(j*x)
```

例:

```
plot for [dataset in "apples bananas"] dataset."dat" title dataset
```

この例では、繰り返しはファイル名と対応するタイトルの生成の両方で使われています。

例:

```
file(n) = sprintf("dataset_%d.dat",n)
splot for [i=1:10] file(i) title sprintf("dataset %d",i)
```

この例は、ファイル名で生成される文字列値関数を定義し、そのような 10 個のファイルを同時に描画します。繰り返しの変数 (この例では 'i') は一つの整数として扱われ、それを 2 度以上使用できます。

例:

```
set key left
plot for [n=1:4] x**n sprintf("%d",n)
```

この例は、関数の組を描画します。

例:

```
list = "apple banana cabbage daikon eggplant"
item(n) = word(list,n)
plot for [i=1:words(list)] item(i).".dat" title item(i)
list = "new stuff"
replot
```

この例では、リストに従って各ステップが進行し、その各項目に対して一つの描画が行われます。この各項目は動的に取得されますので、そのリストを変更し、そのまま replot することができます。

例:

```
list = "apple banana cabbage daikon eggplant"
plot for [i in list] i.".dat" title i
list = "new stuff"
replot
```

この例は、整数の繰り返し変数ではなく、文字列の繰り返し変数形式を用いていること以外は前の例と全く同じです。

<end> の整数の代わりに記号 * を使用すれば、繰り返しはすべての有効なデータがなくなるまでの繰り返しとなります。これは、各行に含まれるすべての列の処理、あるいはファイル内のすべてのデータセット (2 行の空行で区切られる) の処理、指定に当てはまるすべてのファイルなどを一度に処理するのに便利です。

例:

```
plot for [file in "A.dat B.dat"] for [column=2:*] file using 1:column
splot for [i=0:*] 'datafile' index i using 1:2:3 with lines
plot for [i=1:*] file=sprintf("File_%03d.dat",i) file using 2 title file
```

警告: この最初の例のように、繰り返しはかっこなしの形でも入れ子にできます。しかしかっこのない繰り返しを他のかっこのない繰り返しの中に入れ子にするの多分有益ではありません。それは、データが見つからなかった場合に両者が同時に終了してしまうからです。gnuplot はこれが起きると警告を発します。

Title

デフォルトでは各曲線は、対応する関数やファイル名でキーの中に一覧表示されますが、plot のオプション **title** を使うことで、明示的なタイトルを与えることもできます。

書式:

```
title <text> | notitle [<ignored text>]
title columnheader | title columnheader(N)
      {at {beginning|end}} {{no}enhanced}
```

ここで <text> は、引用符で囲まれた文字列か、文字列と評価される式のいずれかです。引用符はキーには表示されません。

入力データの列の最初の項目 (すなわち列の先頭) を文字列フィールドと解釈し、それをキータイトルとして利用するオプションもあります。以下参照: **datastrings (p. 41)**。これは、**set key autotitle columnhead** を指定すればデフォルトの挙動となります。

曲線タイトルとサンプルは予約語 **notitle** を使うことでキーから削除できます。何もないタイトル (**title ''**) は **notitle** と同じ意味を持ちます。サンプルだけが欲しいときは、一つ以上の空白をタイトルの後ろに入れてください (**tilte ' '**)。**notilte** の後ろに文字列をつけた場合、その文字列は無視されます。

key autotitles が設定されて (デフォルト)、かつ **title** も **notitle** も指定されなかった場合、曲線のタイトルは **plot** コマンド上にある関数名かデータファイル名になります。ファイル名の場合は、指定される任意のデータファイル修飾子もそのデフォルトタイトルに含まれます。

位置やタイトルの位置揃えなどの凡例のレイアウトは、**set key** で制御できます。

キーワード **at** により、曲線のタイトルを、自動的に作られる key の箱の外にでも置くことができるようになります。**at {beginning|end}** を使用した場合は、曲線のタイトルをグラフの曲線自身の直前、あるいは直後に置きます。このオプションは、**with lines** で描画する場合は有用ですが、他の描画スタイルでは無意味です。

at <x-position>,<y-position> の形式を使用すれば、曲線のタイトルをページ内の任意の位置に置くことができます。デフォルトでは、その位置指定はスクリーン座標と解釈します。例えば **at 0.5, 0.5** は、グラフの軸に縮尺や境界には関係なく、常にスクリーンのど真ん中を意味します。この方法で配置するタイトルの書式は、key のオプション指定の影響を受けます。以下参照: **set key (p. 229)**。

例:

以下は y=x をタイトル 'x' で表示します:

```
plot x
```

以下は、x の 2 乗をタイトル "x^2" で、ファイル "data.1" をタイトル"measured data" で表示します:

```
plot x**2 title "x^2", 'data.1' t "measured data"
```

以下は、ファイルの先頭行の各列にタイトルを含む複数列のデータを描画します。各タイトルは、独立した凡例ではなく、対応する曲線の後ろに置きます:

```
unset key
set offset 0, graph 0.1
plot for [i=1:4] 'data' using i with lines title columnhead at end
```

以下は、2 つの別々のグラフの key の場所を 1 箇所にします:

```
set key Left reverse
set multiplot layout 2,2
plot sin(x) with points pt 6 title "Left plot is sin(x)" at 0.5, 0.30
plot cos(x) with points pt 7 title "Right plot is cos(x)" at 0.5, 0.27
unset multiplot
```

With

関数やデータの表示にはたくさんのスタイルのうちの一つを使うことができます。キーワード **with** がその選択のために用意されています。

書式:

```
with <style> { {linestyle | ls <line_style>}
               | {{linetype  | lt <line_type>}
                  {linewidth | lw <line_width>}
                  {linecolor | lc <colorspec>}
                  {pointtype | pt <point_type>}
                  {pointsize | ps <point_size>}
                  {arrowstyle | as <arrowstyle_index>}
                  {fill | fs <fillstyle>} {fillcolor | fc <colorspec>}
                  {nohidden3d} {nocontours} {nosurface}
                  {palette}}
             }
```

ここで、<style> は以下のいずれか:

```
lines        dots      steps     vectors      yerrorlines
points       impulses  fsteps    xerrorbar    xyerrorbars
linespoints  labels    histeps   xerrorlines  xyerrorlines
financebars  surface   arrows    yerrorbar    parallelaxes
```

または、

```
boxes         boxplot       ellipses      histograms  rgbalpha
boxerrorbars  candlesticks  filledcurves  image       rgbimage
boxxyerror    circles       fillsteps     pm3d        polygons
isosurface    zerrorfill
```

または

```
table         mask
```

最初のグループのスタイルは、線、点、文字の属性を持ち、第 2 のグループのスタイルは、さらに塗り潰し属性も持っています。以下参照: **fillstyle (p. 283)**。さらにサブスタイルを持つスタイルもあります。個々のスタイルの詳細については、以下参照: **plotting styles (p. 90)**。最後の特別なスタイル 2 つは、すぐに描画するものではありません。以下参照: **set table (p. 290)**, **with mask (p. 116)**。スタイル **table** は、表形式の出力をテキストファイルかデータブロックの形で生成します。スタイルが **with mask** である plot コマン

ドの要素は、多角形領域の集合を定義し、それはその plot コマンドのその後に続く要素をマスクするのに使えます。

デフォルトのスタイルは、**set style function** と **set style data** で選択できます。

デフォルトでは、それぞれの関数やデータファイルは、使うことができる型の最大数に達するまで異なる線種、点種を使います。すべての端末用ドライバは最低 6 つの異なる点種をサポートしていて、もしたくさん要求された場合、それらを順に再利用していきます。使用中の出力形式での線種、点種の集合全体を見たければ、**test** としてください。

一つの描画で線種や点種を選びたいならば、<line_type> や <point_type> を指定してください。これらの値は、その描画で使われる線種や点種を指定する正の整定数 (または数式) です。使用する端末で使える線種、点種を表示するには **test** コマンドを使ってください。

描画の線の幅や点の大きさは <line_width> や <point_size> で変更できます。これらはその各々の端末のデフォルトの値に対する相対的な値として指定します。点の大きさは全体に通用するように変更できます。詳細は、以下参照: **set pointsize (p. 274)**。しかし、ここでセットされる <point_size> と、**set pointsize** でセットされる大きさは、いずれもデフォルトのポイントサイズに掛けられることに注意してください。すなわち、それらの効果は累積はしません。例えば、**set pointsize 2; plot x with points ps 3** は、デフォルトのサイズの 3 倍であって、6 倍ではありません。

ラインスタイルの一部分、あるいは各 plot において **pointsize variable** という指定も可能です。この場合、入力には追加の 1 列が要求されます。例えば 2D 描画では 3 列、3D 描画では 4 列のデータが必要になります。個々の点のサイズは、全体を通しての pointsize に、データファイルからの入力による値をかけたものとして決定されます。

set style line を使って線種/線幅、点種/点幅の組を定義すれば、そのスタイルの番号を <line_style> にセットすることでそれらを使うことができます。

2 次元、3 次元両方の描画で (**plot** と **splot** コマンド)、事前にコマンド **set palette** で設定した滑らかなパレットからの色を使えます。色の値は、点の z 座標の値か、または **using** によるオプションの追加列で与える個別の色座標に対応します。色の値は、小数値 (**palette frac**) か、またはカラーボックスの範囲へ対応づけられた座標値 (**palette** か **palette z**) のいずれかで指定できます。以下参照: **colorspec (p. 71)**, **set palette (p. 258)**, **linetypes (p. 70)**。

キーワード **nohidden3d** は、**splot** コマンドで生成される描画にのみ適用されます。通常、グローバルなオプション **set hidden3d** はグラフ上の全ての描画に適用されますが、各々の描画に **nohidden3d** オプションをつけることで、それを hidden3d の処理から除外することができます。**nohidden3d** がマークされた曲面以外の個々の描画要素 (線分、点、ラベル等) は、通常は他の何らかの描画要素で隠されてしまう場合も全て描画されます。

同様に、キーワード **nocontours** は、グローバルに **set contour** 指定が有効な場合でも、個別の plot に対する等高線描画機能をオフにします。

同様に、キーワード **nosurface** は、グローバルに **set surface** 指定が有効な場合でも、個別の plot に対する 3 次元曲面描画をオフにします。

キーワードは暗示するような形で省略可能です。

linewidth, **pointsize**, **palette** オプションは全ての端末装置でサポートされているわけではないことに注意してください。

例:

以下は、sin(x) を鉛直線で描画します:

```
plot sin(x) with impulses
```

以下は、x を点で描画し、x**2 をデフォルトの方式で描画します:

```
plot x w points, x**2
```

以下は、tan(x) を関数のデフォルトの方式で、"data.1" を折れ線で描画します:

```
plot tan(x), 'data.1' with l
```

以下は、"leastsq.dat" を鉛直線で描画します:

```
plot 'leastsq.dat' w i
```

以下は、データファイル "population" を矩形で描画します:

```
plot 'population' with boxes
```

以下は、"exper.dat" をエラーバー付きの折れ線で描画します (エラーバーは 3 列、あるいは 4 列のデータを必要とします):

```
plot 'exper.dat' w lines, 'exper.dat' notitle w errorbars
```

もう一つの "exper.dat" のエラーバー付きの折れ線 (errorlines) での描画方法 (エラーバーは 3 列、あるいは 4 列のデータが必要):

```
plot 'exper.dat' w errorlines
```

以下は、sin(x) と cos(x) をマーカー付きの折れ線で描画します。折れ線は同じ線種ですが、マーカーは異なったものを使います:

```
plot sin(x) with linesp lt 1 pt 3, cos(x) with linesp lt 1 pt 4
```

以下は、"data" を点種 3 で、点の大きさを通常の 2 倍で描画します:

```
plot 'data' with points pointtype 3 pointsize 2
```

以下は、"data" を描画しますが、4 列目から読んだデータを pointsize の値として使用します:

```
plot 'data' using 1:2:4 with points pt 5 pointsize variable
```

以下は、2 つのデータ集合に対して、幅のみ異なる線を用いて描画します:

```
plot 'd1' t "good" w l lt 2 lw 3, 'd2' t "bad" w l lt 2 lw 1
```

以下は、x*x の曲線の内部の塗りつぶしと色の帯を描画します:

```
plot x*x with filledcurve closed, 40 with filledcurve y=10
```

以下は、x*x の曲線と色の箱を描画します:

```
plot x*x, (x>=-5 && x<=5 ? 40 : 1/0) with filledcurve y=10 lt 8
```

以下は、滑らかに変化する色の線で曲面を描画します:

```
splot x*x-y*y with line palette
```

以下は、2 つの色のついた曲面を、異なる高さで表示します:

```
splot x*x-y*y with pm3d, x*x+y*y with pm3d at t
```

Print

書式:

```
print <式> {, <式>, ...}
```

print コマンドは、1 つ、または複数の式の値を出力します。出力は、コマンド **set print** でリダイレクトされていない限り、画面へ行われます。以下参照: **expressions (p. 45)**。以下も参照: **printerr (p. 189)**。

< 式 > は、gnuplot で有効な任意の式で、数値でも、文字列定数でも、数字や文字列を返す関数でも、配列でも、または変数名でも入れることができます。データブロックを出力することも可能です。**print** に sprintf や gprintf 関数を組み合わせて、さらに柔軟な書式の出力を行うことも可能です。

print コマンド内で繰り返しを使うことで、単一行に複数の値を入れることも可能です。

例:

```
print 123 + 456
print sinh(pi/2)
print "rms of residuals (FIT_STDFIT) is ", FIT_STDFIT
print sprintf("rms of residuals is %.3f after fit", FIT_STDFIT)
print "Array A: ", A
print "Individual elements of array A: ", for [i=1:|A|] A[i]
print $DATA
```

Printerr

printerr は **print** コマンドとほぼ同じですが、その前の **set print** コマンドの効果が続いている状態でも出力を常に stderr に送るところだけが違います。その出力に現在のファイル名 (または関数ブロック名) と行番号を入れたい場合は、代わりにコマンド **warn** を使用してください。

Pwd

pwd コマンドはカレントディレクトリの名前を画面に表示します。

カレントディレクトリを文字列変数に保存したり、文字式の中で使いたい場合は、変数 GPVAL_PWD を使うことができることに注意してください。以下参照: **show variables all (p. 319)**。

Quit

quit は、コマンド **exit** と同義です。以下参照: **exit (p. 136)**。

Raise

書式:

```
raise {plot_window_id}
lower {plot_window_id}
```

コマンド **raise** と **lower** は、出力形式のいくつかにしか機能せず、そしてあなたが使用するウィンドウマネージャや表示優先機能の設定にも依存する可能性があります。

```
set term wxt 123     # 最初の描画ウィンドウを生成
plot $FOO
lower                # 存在する描画ウィンドウのみを下に
set term wxt 456     # 2 つ目を生成 (1 つ目の上にかぶる)
plot $BAZ
raise 123            # 1 つ目の描画ウィンドウを上に
```

これらのコマンドは、あまり当てにならないと思ってください。

Refresh

コマンド **refresh** は、**replot** に似ていますが、主に 2 つの点で違いがあります。**refresh** は、既に読み込んだデータを用いて、現在の描画を再整形し再描画します。これは、**refresh** を (疑似デバイス '-' からの) インラインデータの描画、および内容が変化しうるデータファイルからの描画に使えるということを意味します。ただし、コマンド **refresh** は、既に存在する描画に新しいデータを追加するのには使えません。

マウス操作、特にズームインとズームアウトでは、適切な場合は **replot** の代わりにむしろ **refresh** を使用します。例:

```
plot 'datafile' volatile with lines, '-' with labels
100 200 "Special point"
e
# 色んなマウス操作をここで実行
set title "Zoomed in view"
set term post
set output 'zoom.ps'
refresh
```

Remultiplot

remultiplot は、直前の multiplot の開始時に $GPAVL_PRE_MULTIPLOT に保存した描画設定を最初に復帰し、次に $GPVAL_LAST_MULTIPLOT という名前付きデータブロックのコマンドを再実行します。その結果、元の multiplot が生成した図を再生成します。それに対し、**replot** は **remultiplot** とは違い、最初の描画設定の復帰を行いません。以下参照: **new multiplots** (p. 32)。

Replot

replot コマンドを引数なしで実行すると、最後に実行した **plot** または **splot** コマンドを再実行します。これは、あるプロットを異なる **set** オプションでみたり、同じプロットを異なる装置に出力したりするときに便利

でしょう。

replot コマンドに対する引数は最後に実行した **plot** または **splot** コマンドの引数に (暗黙の ',' と共に) 追加され、それから再実行されます。**replot** は、範囲 (range) を除いては、**plot** や **splot** と同じ引数をとることができます。よって、直前のコマンドが **splot** ではなく **plot** の場合は、関数をもう一つの軸刻みでプロットするのに **replot** を使うことができます。

注意:

```
plot '-' ; ... ; replot
```

は推奨されません。それは、これがあなたに再び同じデータすべての入力を要求することになるからです。たいていの場合、代わりにコマンド **refresh** を使えます。これは、以前に読み込んだデータを使ってグラフを再描画します。

最後に実行した **plot** (splot) コマンドの内容を修正する方法については以下も参照: **command-line-editing (p. 40)**。

直前の描画コマンドの全体を表示させることや、それを **history** の中にコピーする方法については、以下も参照: **show plot (p. 319)**。

以前の gnuplot の版では、multiplot 全体は再描画できず、**replot** コマンドはそのうち最後の plot 要素しか再生成しませんでした。gnuplot バージョン 6 では、multiplot を生成するのに使用したコマンドをデータブロック$GPVAL_LAST_MULTIPLOT に保存します。そして **replot** か、新しいコマンド **remultiplot** を使って再実行し multiplot 全体を再生できます。注意がいくつかあります。以下参照: **new multiplots (p. 32)**, **remultiplot (p. 190)**。

Reread

[バージョン 5.4 では非推奨]

明示的な繰り返し (iteration) を支持し、このコマンドは非推奨とします。以下参照: **iterate (p. 69)**。**reread** コマンドは、**load** コマンドで指定した **gnuplot** の入力ファイルからの実行を、直ちにそのファイルの先頭から再開します。これは、ファイルの最初から **reread** コマンドまでのコマンドの無限ループを本質的に実装していることになります。標準入力からの入力の際は、**reread** コマンドは何の効力も持ちません。

Reset

```
reset {bind | errors | session}
```

コマンド **reset** は、**set** コマンドで定義できる、グラフに関する全てのオプションをデフォルトの値に戻します。このコマンドは、load したコマンドファイルを実行した後でデフォルトの設定を復帰したり、設定をたくさん変更した後で元の状態に戻したいときなどに便利です。

以下のものは、**reset** の影響を受けません:

```
`set term` `set output` `set loadpath` `set linetype` `set fit`
`set encoding` `set decimalsign` `set locale` `set psdir`
`set overflow` `set multiplot` `set minussign` `set micro`
```

reset は、必ずしもプログラム立ち上がった初期状態には戻さないことに注意してください。それは、初期設定ファイル gnuplotrc や $HOME/.gnuplot、$XDG_CONFIG_HOME/gnuplot/gnuplotrc 内のコマンドでデフォルトの値を変更した場合は、それもリセットされてしまうからです。しかし **reset session** とすれば、それらのコマンドも再実行します。

reset session は、ユーザ定義変数、ユーザ定義関数すべてを削除し、デフォルトの設定を復帰し、システム全体の初期設定ファイル gnuplotrc と個人用の初期設ファイル $HOME/.gnuplot、$XDG_CONFIG_HOME/gnuplot/gnuplotrc を再実行します。以下参照: **initialization (p. 80)**。

reset errors は、エラー状態変数 GPVAL_ERRNO と GPVAL_ERRMSG のみをクリアします。

reset bind は、キー定義をデフォルトの状態に復帰します。

Return

書式:

```
return <expression>
```

コマンド **return** は、 コマンド **exit** や **quit** が、現在のコードブロックの実行や入力ストリームを終了するのと同じ方法で作用します。返り値は関数ブロック内の実行コードの状況でのみ意味があります。以下参照: **function blocks (p. 148)**。

例:

```
function $myfun << EOF
local result = 0
if (error-condition) { return -1 }
... body of function ...
return result
EOF
```

Save

書式:

```
save  {functions | variables | terminal | set | fit | datablocks}
      '<filename>' {append}

save changes '<filename>' {append}
```

どれも指定しなかった場合は、**gnuplot** は、ユーザ定義関数、ユーザ変数、set で設定するオプション、一番最後に実行した **plot** か **splot** コマンドの全てを保存します。**set term** と **set output** の現在の状態は、コメントとして書き出します。保存ファイルには、gnuplot コマンドの列がテキスト形式で保存され、それはコマンド **load** の入力として使用できます。

save changes "savefile.gp" は、gnuplot セッションの開始時のプログラムの状態から変更された関数、変数、設定のみを書き出します。これは、**reset session; load "savefile.gp"** で現在のグラフとプログラムの状

態を再生するのに十分な、とても短い出力を作成します。このコマンドはすべてのシステムでサポートしているわけではなく、あなたの gnuplot では利用できないかもしれません。

save terminal は、**terminal** の状態を、コメント記号をつけずに書き出します。これは主に、ちょっとの間だけ **terminal** の設定を入れ替え、その後保存しておいた **terminal** の状態を読み込むことで以前の terminal の設定に戻す場合などに役立ちます。ただ、単一の gnuplot セッションでは、現在の terminal を保存/復元する他の方法であるコマンド **set term push** と **set term pop** を使う方がむしろいいかもしれません。以下参照: **set term (p. 291)**。

save variables は、すべてのユーザ変数を書き出しますが、データブロックと内部変数 GPVAL_* GPFUN_* MOUSE_* ARG* は書き出しません。

save fit は、直近の **fit** コマンドで使用した変数のみを保存します。その保存ファイルは、後で **via** キーワードを使うことで fit コマンドの初期化用のパラメータファイルとして利用できます。

ファイル名は引用符に囲われていなければなりません。

特別なファイル名 "-" により **save** コマンドに標準出力に出力させることができます。popen 関数をサポートするようなシステム (Unix など) では、save の出力をパイプ経由で他の外部プログラムに渡すことができます。その場合、ファイル名としてコマンド名の先頭に '|' をつけたものを使います。これは、**gnuplot** とパイプを通して通信するプログラムに、**gnuplot** の内部設定に関する首尾一貫したインターフェースを提供します。詳細は、以下参照: **batch/interactive (p. 38)**。

例:

```
save 'work.gnu'
save functions 'func.dat'
save var 'state.dat'; save datablocks 'state.dat' append
save set 'options.dat'
save term 'myterm.gnu'
save '-'
save '|grep title >t.gp'
```

Set-show

set コマンドは実に多くのオプションを設定するのに使われます。しかし、**plot**, **splot**, **replot** コマンドが与えられるまで何のグラフも描きません。

そのほとんどのオプションに対して、コマンド **show** がそれに対応する現在の設定を表示します。**show palette** や **show colornames** などのごく少数のコマンドのみ、個別に説明しています。

set コマンドで変更されたオプションは、それに対応する **unset** コマンドを実行することでデフォルトの状態に戻すことができます。以下も参照: **reset (p. 191)**。これは全てのパラメータの設定をデフォルトの値に戻します。

set と **unset** コマンドには繰り返し節も利用できます。以下参照: **plot for (p. 183)**。

角の単位 (angles)

デフォルトでは gnuplot は極座標グラフの独立変数の単位はラジアンを仮定します。**set polar** の前に **set angles degrees** を指定すると、その単位は度になり、デフォルトの範囲は [0:360] となります。これはデータファイルの描画で特に便利でしょう。角度の設定は、**set mapping** コマンドを設定することにより 3 次元でも有効です。

書式:

```
set angles {degrees | radians}
show angles
```

set grid polar で指定される角度も、**set angles** で指定した単位で読まれ表示されます。

set angles は組み込み関数 sin(x), cos(x), tan(x) の引数や asin(x), acos(x), atan8x), atan2(x), arg(x) の出力にも影響を与えます。双曲線関数や、ベッセル関数の引数には影響を与えません。しかし、複素数を引数とする逆双曲線関数の出力には影響が出ます。それらの関数が使われるときは、**set angles radians** は入出力の引数の間に一貫性を持った管理を実現していなければなりません。

```
x={1.0,0.1}
set angles radians
y=sinh(x)
print y          #{1.16933, 0.154051} と表示
print asinh(y)   #{1.0, 0.1} と表示
```

しかし、

```
set angles degrees
y=sinh(x)
print y          #{1.16933, 0.154051} と表示
print asinh(y)   #{57.29578, 5.729578} と表示
```

以下も参照poldat.dem: **set angles** を用いた極座標描画のデモ

矢印 (arrow)

set arrow コマンドを使うことにより、グラフ上の任意の位置に矢印を表示することができます。

書式:

```
set arrow {<tag>} from <position> to <position>
set arrow {<tag>} from <position> rto <position>
set arrow {<tag>} from <position> length <coord> angle <ang>
set arrow <tag> arrowstyle | as <arrow_style>
set arrow <tag> {nohead | head | backhead | heads}
                {size <headlength>,<headangle>{,<backangle>}} {fixed}
                {filled | empty | nofilled | noborder}
                {front | back}
                {linestyle | ls <line_style>}
```

```
                    {linetype | lt <line_type>}
                    {linewidth | lw <line_width>}
                    {linecolor | lc <colorspec>}
                    {dashtype | dt <dashtype>}

unset arrow {<tag>}
show arrow {<tag>}
```

タグ <tag> は各矢印を識別する整数です。タグを指定しない場合は、その時点で未使用の最も小さい数が自動的に割り当てられます。タグを使うことで、特定の矢印を変更したり、削除したりできます。既に存在する矢印の属性を変更する場合は、タグを明示した **set arrow** コマンドで変更したい属性を指定してください。

矢印の最初の端点の位置は、常に "from" で指定しますが、もう一つの端点は以下で説明する 3 つの異なる仕組みのいずれかで指定できます。<position> は x,y あるいは x,y,z で指定します。そしてその前に座標系を選択するために **first**, **second**, **graph**, **screen**, **character** を置くことができます。座標を指定しなければデフォルトでは 0 と見なされます。詳細は以下参照: **coordinates (p. 41)**。最初の端点に対する座標指定子は、2 番目の端点には影響しません。

1) "to <position>" は、もう一つの端点の絶対座標を指定します。

2) "rto <position>" は、"from" の位置からのずれを指定します。この場合、線形軸 (非対数軸)、および **graph**, **screen** 座標に対しては、始点と終点の距離が与えられた相対的な値に対応します。一方、対数軸に対しては、与えられた相対的な値は、始点から終点への倍数に対応します。よって、対数軸の場合、相対的な値として 0 や負の値を与えることは許されません。

3) "length <coordinate> angle <angle>" は、グラフ平面内での矢印の方向を指定します。length には任意の座標系を適用できます。angle の単位は常に度になっています。

矢印の他の属性も、あらかじめ定義した矢のスタイルで、またはコマンド **set arrow** でそれぞれ与えることが可能です。矢印の他の属性の詳細については以下参照: **arrowstyle (p. 280)**。

例:

原点から (1,2) への矢印をユーザ定義済のラインスタイル 5 で描くには:

```
set arrow to 1,2 ls 5
```

描画領域の左下角から (-5,5,3) へタグ番号 3 の矢印を描くには:

```
set arrow 3 from graph 0,0 to -5,5,3
```

矢印の端を 1,1,1 に変更し、矢先を外して幅を 2 にするには:

```
set arrow 3 to 1,1,1 nohead lw 2
```

x=3 の所へグラフの下から上まで鉛直線を描くには:

```
set arrow from 3, graph 0 to 3, graph 1 nohead
```

T 字型の矢先を両端に持つ鉛直方向の矢を描くには:

```
set arrow 3 from 0,-5 to 0,5 heads size screen 0.1,90
```

始点からの相対的な距離をグラフ座標で与えて矢を描くには:

```
set arrow from 0,-5 rto graph 0.1,0.1
```

x の対数軸に相対的な終点を指定して矢を描く場合:

```
set logscale x
set arrow from 100,-5 rto 10,10
```

これは 100,-5 から 1000,5 までの矢を描きます。線形軸 (y) に対しては相対的な座標 10 が "差 10" を意味するのに対し、対数軸 (x) に対しては相対的な座標 10 は "倍数 10" として働きます。

2 番の矢印を消すには:

```
unset arrow 2
```

全ての矢印を消すには:

```
unset arrow
```

全ての矢印の情報を (タグの順に) 見るには:

```
show arrow
```

矢印のデモ

自動縮尺 (autoscale)

自動縮尺機能 (autoscale) は x, y, z の各軸に対して独立に、または一括して指定できます。デフォルトでは全ての軸に対して自動縮尺設定を行います。図の中の一部の描画 (**plot**) の組のみを autoscale したい場合は、除外するものの plot コマンドにフラグ **noautoscale** をつければいいでしょう。以下参照: **datafile (p. 162)**。

書式:

```
set autoscale {<axis>{|min|max|fixmin|fixmax|fix} | fix | keepfix}
set autoscale noextend
unset autoscale {<axis>}
show autoscale
```

ここで、<axis> (軸) は **x**, **y**, **z**, **cb**, **x2**, **y2**, **xy**, **paxis** <p> のいずれかです。軸名の後ろに **min** または **max** を追加すると、それは **gnuplot** にその軸の最小値、または最大値のみを自動縮尺させることになります。

軸名を指定しない場合は、全ての軸が自動縮尺の対象となります。

独立変数軸 (**plot** のときは x 軸、**splot** のときは x,y 軸) の自動縮尺機能は、描画されるデータに合うようにそれらの軸の範囲を調整します。描画が関数のみ (入力データなし) の場合、これらの軸の自動縮尺機能は、なんの効果も持ちません。

従属変数軸 (**plot** のときは y 軸、**splot** のときは z 軸) の自動縮尺機能は、描画されるデータや関数に合うようにそれらの軸の範囲を調整します。

軸の範囲の調整は、次の目盛り刻みへの延長を行うことがあります。例えば、端のデータの座標が目盛り刻みに丁度一致する場合、データと描画境界との間には何もないスペースができることになります。**noextend** を使うことで、この余計なスペースの作成を抑制できます。コマンド **set offset** を使うことで、それをより増やすこともできます。さらなる情報については、以下参照: **set xrange (p. 305)**, **set offsets (p. 255)**。

媒介変数モード (parametric) でも自動縮尺機能は有効です (以下参照: **set parametric (p. 265)**)。この場合、より多くの従属変数があるので、x, y, z 各軸に関して、より多くの制御が行われます。媒介変数モードで

の独立変数 (仮変数) は **plot** では t で **splot** では u, v です。そして媒介変数モードでは、自動縮尺機能は (t, u, v, x, y, z) の全ての描画範囲を制御し、x, y, z の範囲の自動設定を完全に行います。

目盛りが第 2 の軸に表示され、しかもこれらの軸に対する描画が行われなかった場合には、x2range と y2range は xrange と yrange の値を受け継ぎます。これは、範囲のずらしの実行や、範囲を整数個の目盛り幅に自動伸縮する「前」に行いますので、場合によって予期しない結果をもたらす可能性があります。これを避けるのに、第 2 軸の範囲を第 1 軸の範囲に明示的にリンク (link) する方法があります。以下参照: **set link (p. 239)**。

Noextend

```
set autoscale noextend
```

デフォルトでは、自動縮尺機能は軸の範囲の限界を、描画データ全体を含む、最も近い目盛りラベル位置に設定します。キーワード **fixmin**, **fixmax**, **fix**, **noextend** は、次の目盛り位置までの範囲の自動拡大を gnuplot に行わせないようにします。その場合軸の範囲の限界は、一番端にあるデータ点の座標値に完全に一致します。**set autoscale noextend** は、**set autscale fix** と同じです。軸の範囲指定コマンドの後ろにキーワード **noextend** を追加すれば、一つの軸の範囲の延長機能だけ無効にすることもできます。例:

```
set yrange [0:*] noextend
```

set autoscale keepfix は、fix の設定を変更せずに残したまま、すべての軸を自動縮尺にします。

例 (examples)

例:

以下は y 軸の自動縮尺機能を指定します (他の軸には影響を与えません):

```
set autoscale y
```

以下は y 軸の最小値に対してのみ自動縮尺機能を指定します (y 軸の最大値、および他の軸には影響を与えません):

```
set autoscale ymin
```

以下は x2 軸の隣の目盛りへの自動範囲拡大機能を無効にし、よって描画データ内、または関数に対する丁度の描画範囲を維持します:

```
set autoscale x2fixmin
set autoscale x2fixmax
```

以下は x, y 両軸の自動縮尺機能を指定します:

```
set autoscale xy
```

以下は x, y, z, x2, y2 全軸の自動縮尺機能を指定します:

```
set autoscale
```

以下は x, y, z, x2, y2 全軸の自動縮尺機能を禁止します:

```
unset autoscale
```

以下は z 軸のみについて自動縮尺機能を禁止します:

```
unset autoscale z
```

極座標モード (polar)

極座標モード (**set polar**) では、xrange と yrange は自動縮尺モードではなくなります。動径軸の範囲制限用に **set rrange** を使用した場合、xrange と yrange はそれに合うように自動的に調整されます。しかし、さらにそれを調整したければ、その後に明示的に xrange や yrange コマンドを使うことができます。以下参照: **set rrange (p. 277)**。

以下も参照極座標のデモ。

Bind

show bind は、現在のホットキーの割り当て (binding) を表示します。以下参照: **bind (p. 76)**。

Bmargin

コマンド **set bmargin** は、下部の余白のサイズを設定します。詳細は以下参照: **set margin (p. 242)**。

グラフの枠線 (border)

set border と **unset border** は **plot** や **splot** でのグラフの枠の表示を制御します。枠は必ずしも軸とは一致しないことに注意してください。**plot** では大抵一致しますが、**splot** では大抵一致していません。

書式:

```
set border {<integer>}
           {front | back | behind}
           {linestyle | ls <line_style>}
           {linetype | lt <line_type>} {linewidth | lw <line_width>}
           {linecolor | lc <colorspec>} {dashtype | dt <dashtype>}
           {polar}
unset border
show border
```

set view 56,103 のように任意の方向で表示されうる **splot** では、 x-y 平面上の 4 つの角は 手前 (**front**), 後ろ (**back**) , 左 (**left**), 右 (**right**) のように呼ばれます。もちろんこの同じ 4 つの角は天井の面にもあります。よって、例えば x-y 平面上の後ろと右の角をつなぐ境界を"底の右後ろ (bottom right back)" と言い、底と天井の手前の角をつなぐ境界を "鉛直手前 (front vertical)" と呼ぶことにします (この命名法は、読者が下の表を理解するためだけに使われます)。

枠は、12 ビットの整数に符号化されています: 下位 4 ビットは **plot** に対する外枠、**splot** に対しては底面の外枠、次の 4 ビットは **splot** の鉛直な外枠、そして上位 4 ビットは **splot** の天井面の外枠を制御します。よって外枠の設定は、次の表の対応する項目の数字の和になります:

グラフ境界の符号化		
ビット	plot	splot
1	下	底の左手前
2	左	底の左後ろ
4	上	底の右手前
8	右	底の右後ろ
16	効果なし	鉛直左
32	効果なし	鉛直後ろ
64	効果なし	鉛直右
128	効果なし	鉛直の手前
256	効果なし	天井の左後ろ
512	効果なし	天井の右後ろ
1024	効果なし	天井の左手前
2048	効果なし	天井の右手前
4096	極座標系	効果なし

デフォルトの設定値は 31 で、これは **plot** では 4 方向の外枠全て、**splot** では底面の枠線全部と z 軸を描くことを意味します。

3 次元の 4 本の鉛直な境界線とは別に、コマンド **splot** はデフォルトで、曲面のそれぞれの角からグラフの床面への鉛直線を描画します。**set border** はこの鉛直線を制御しません。代わりに、**set/unset cornerpoles** を使ってください。

2 次元描画では境界はすべての描画要素の一番上に描かれます (**front**)。もし境界を描画要素の下に描かせたい場合は、**set border back** としてください。

3 次元隠線処理 (hidden3d) 描画では、通常は境界を構成する線も描画要素と同様に隠線処理の対象になります。**set border behind** とするとこのデフォルトの挙動が変わります。

<linestyle>, <linetype>, <linewidth>, <linecolor>, <dashtype> を指定して、枠線の描画にそれらを反映させることができます (現在の出力装置がサポートするものに限定されます)。さらに、軸の刻み (tics) を描画する際も、それらを境界線上で描画するか、軸上で描画するかに関わらず、このラインスタイルを使用します。

plot では、第 2 軸を有効にすることで、下と左以外の境界に目盛りを描くことができます。詳細は、以下参照: **xtics (p. 307)**。

"unset surface; set contour base" などによって **splot** で底面にのみ描画する場合、鉛直線や天井はそれらが指定されていても描画されません。

set grid のオプション 'back', 'front', 'layerdefault' でも、描画出力の境界線を書く順番を制御できます。

キーワード **polar** は、極座標グラフに円形の境界をつけます。

例:

以下は、デフォルトの枠線を描きます:

```
set border
```

以下は、**plot** では左と下、**splot** では底面の左手前と左後ろの枠線を描きます:

```
set border 3
```

以下は、**splot** で周りに完全な箱を描きます:

```
set border 4095
```

以下は、手前の鉛直面と天井のない箱を描きます:

```
set border 127+256+512 # または set border 1023-128
```

以下は、**plot** に対して上と右枠線のみを描き、それらを軸として目盛りづけします:

```
unset xtics; unset ytics; set x2tics; set y2tics; set border 12
```

棒グラフ幅 (boxwidth)

コマンド **set boxwidth** は **boxes**, **boxerrorbars**, **boxplot**, **candlesticks**, **histograms** スタイルにおける棒のデフォルトの幅を設定するために使います。

書式:

```
set boxwidth {<width>} {absolute|relative}
show boxwidth
```

デフォルトでは、隣り合う棒が接するように各々の棒の幅が広げられます。それとは異なるデフォルトの幅を設定するには **set boxwidth** コマンドを使用します。**relative** の場合の幅は、デフォルトの幅に対する比であると解釈されます。

修飾子 **relative** を指定しなかった場合、棒の幅 (boxwidth) として指定された明示的な値は、現在の x 軸の単位での数字 (**absolute**) であると解釈されます。x 軸が対数軸 (以下参照: **set log (p. 240)**) である場合、boxwidth の値は実際には x=1 でのみ "絶対的" となり、その物理的な長さが軸全体を通じて保持されます (すなわち、棒は x 座標の増加にともなって狭くなったりはしません)。対数軸の x 軸の範囲が x=1 から離れている場合は、適切な幅を見出すには何度か試してみる必要があるかも知れません。

デフォルトの値は、**boxes** や **boxerrorbars** スタイルの幅指定用の追加のデータ列の明示的な値があればそれによって置き換えられます。詳細は、以下参照: **style boxes (p. 92)**, **style boxerrorbars (p. 92)**。

棒の幅を自動的にセットするには

```
set boxwidth
```

棒の幅を自動的な値の半分にするには

```
set boxwidth 0.5 relative
```

棒の幅を絶対的な値 2 にするには

```
set boxwidth 2 absolute
```

3 次元箱の奥行き (boxdepth)

```
set boxdepth {<y extent>} | square
```

コマンド **set boxdepth** は、**splot with boxes** で作った 3 次元グラフにのみ影響します。これは、y 軸方向の各箱の奥行き (箱の太さ) を設定します。**set boxdepth square** は、y 軸方向の奥行きを、x と y の軸の縮尺とは無関係に、見た目が正方形の断面になるように選択しようとします。

χ -形状 (chi_shapes)

```
set chi_shapes fraction <value>
unset chi_shapes
```

凹包 (concave hull) フィルタは、特性長 chi_length で定義されるχ -形状 (chi-shapes) を生成します。chi_length 変数が設定されていなければ、それは境界多角形 (凸包) における最長の辺の比率に等しい値を選択します。この比率のデフォルトは 0.6 ですが、それはこのコマンドで変更できます。その値を 1.0 にすると結果の閉包は凸包に縮まります。より小さくすると、凹部分がより増える閉包になります。以下参照: **concavehull (p. 167)**。コマンド **unset chi_shapes** は比率 0.6 を復帰し、chi_length 変数を未定義にします。

カラーモード (color)

gnuplot は、**plot** や **splot** コマンドの各要素に、事前に定義した列から取り出した新しい線属性集合を割り当てます。デフォルトでは、色を変更することで引き続く曲線同士を区別できるようにしますが、**set monochrome** によるもう一つの方法は、線幅か点線/破線パターンで区別できる黒い曲線の列を使用します。コマンド **set color** は、この白黒モードを中断し、デフォルトのカラー曲線の集合に戻ります。以下参照: **set monochrome (p. 243)**, **set linetype (p. 238)**, **set colorsequence (p. 202)**。

カラーマップ (colormap)

書式:

```
set colormap new <colormap-name>
set colormap <colormap-name> range [<min>:<max>]
show colormaps
```

set colormap new <name> は、カラーマップ配列 <name> を作成し、現在のパレット設定をそれに書き出します。この保存したカラーマップは、32bit の ARGB カラー値の配列としてのさらなる操作も可能ですし、その後の plot で名前を指定して使用することもできます。

以下は、暗い赤から白へ連続するパレットを生成し、それを 'Reds' という名前のカラーパレット配列に保存し、カラーマップのすべてのエントリを少しだけ透明化する例です。この名前付きカラーマップは、後で pm3d 曲面に色付けするのに使っています。名前付きカラーマップのアルファチャンネル値は、ARGB 線属性に従う、すなわち 0 が不透明、0xff が完全な透明であることに注意してください。

```
set palette defined (0 "dark-red", 1 "white")
set colormap new Reds
do for [i=1:|Reds|] { Reds[i] = Reds[i] | 0x3F000000 }
splot func(x,y) with pm3d fillcolor palette Reds
```

z の値からこのカラーマップへの写像は、端の値に対応する z の最小値と最大値を指定することで調整できます。例:

```
set colormap Reds range [0:10]
```

範囲を設定しない場合、あるいは最小値と最大値が同じ値の場合は、現在の cbrange の限界値を使用します。以下参照: **set cbrange** (p. **317**)。

カラーマップは、長方形領域をグラデーション塗りするのにも使えます。以下参照: **pixmap colormap** (p. **267**)。

色巡回列 (colorsequence)

書式:

```
set colorsequence {default|classic|podo}
```

set colorsequence default は、出力形式に依存しない 8 色の巡回列を選択します。以下参照: **set linetype** (p. **238**), **colors** (p. **70**)。

set colorsequence classic は、出力形式別にそのドライバが用意する線色の列を選択します。色の種類は、4 色から 100 色まで幅がありますが、その多くは、赤、緑、青、紫、水色、黄色、で始まります。これがバージョン 5 以前のデフォルトの挙動です。

set colorsequence podo は、Wong (2011) [Nature Methods 8:441] で推奨されている、P 型、D 型 (Protanopia, Deuteranopia) の色弱者が容易に区別できる 8 色の組を選択します。

いずれの場合でも、色列の長さとその色についてはさらにカスタマイズできます。以下参照: **set linetype** (p. **238**), **colors** (p. **70**)。

Clabel

このコマンドは非推奨です。代わりに **set cntrlabel** を使用してください。**set clabel "format"** は **set cntrlabel format "format"** に、**unset clabel** は **set cntrlabel onecolor** に置き換わっています。

クリッピング (clip)

書式:

```
set clip {points|one|two|radial}
unset clip {points|one|two|radial}
show clip
```

デフォルトの状態:

```
unset clip points
set clip one
unset clip two
unset clip radial
```

グラフ領域の境界内に中心があるデータ点は、その点を表す記号のサイズがその記号を境界線の外にはみだしてしまうような場合でも、通常は描画します。**set clip points** は、2 次元描画でそのような点の中心がグラ

フ領域内にある場合でも、そのような点をクリッピングします (つまり描画しません)。点の中心がグラフ領域外にあるようなデータ点は、決して描画しません。

unset clip の場合は、線分の一方の端点が描画範囲 (xrange と yrange) の外にあれば、その線分は描画しないようにします。

set clip one の場合は、一方の端点が描画範囲内にあって、かつもう一方の端点が範囲外にあるような線分の、範囲内に含まれる部分を描画するように **gnuplot** に指示します。**set clip two** は、両方の端点が描画範囲外にある線分の、範囲内の部分を描画するように **gnuplot** に指示します。線分全体が描画範囲外であるような線分は決して描画しません。

set clip radial は、極座標モードでのみ効力を持ちます。これは、線のクリッピングを、**set rrange [0:MAX]** で確定する円状境界に対して行います。この仕組みは、**set clip {one|two}** と組み合わせて使用します。すなわち、R > RMAX である 2 つの点の間の線が R = RMAX の円によって切り取られる部分は、**clip two** と **clip radial** の両方が設定されている場合のみ描画されます。

注意:

* **set clip** は、描画スタイル **lines**, **linespoints**, **points**, **arrows**, **vectors** で生成される点、線分のみに影響します。

* **pm3d** 曲面や他の中身の詰まったオブジェクトの描画に使用する色長方形のクリッピングの制御は、**set pm3d clipping** で行います。デフォルトは、現在の zrange に対する滑らかなクリッピングです。

* オブジェクトのクリッピングは、個々のオブジェクト (object) の **clip** か **noclip** 属性で制御します。

* 現在の gnuplot の版では、極座標モードの "plot with vectors" は、テストしていませんし、最大半径に対するクリッピングもしません。

等高線ラベル (cntrlabel)

書式:

```
set cntrlabel {format "format"} {font "font"}
set cntrlabel {start <int>} {interval <int>}
set cntrlabel onecolor
```

set cntrlabel は、凡例内 (デフォルト) か、**splot ... with labels** の際のグラフ上の等高線のラベルを制御します。後者の場合、ラベルはラベル記述属性の **pointinterval** か **pointnumber** に従って各等高線に沿って配置されます。デフォルトではラベルは等高線を構成する 5 番目の線分の上に置かれ、20 個の線分毎に繰り返されます。このデフォルトは、以下と同じです:

```
set cntrlabel start 5 interval 20
```

これらの値はコマンド **set cntrlabel** で、あるいは **splot** コマンドに間隔を指定することで変更できます:

```
set contours; splot $FOO with labels point pointinterval -1
```

間隔を負の値に設定すると、ラベルは各等高線に 1 つだけつきます。しかし **set samples** か **set isosamples** が大きな値の場合は多くの等高線をラベル一つだけで描きます。

凡例 (key) には、等高線ラベルをそれぞれの線種 (linetype) を使用して書きます。デフォルトでは、線種自身が各等高線のレベルを与えるので、それぞれに対する別々のラベルが現れます。コマンド **set cntrlabel**

onecolor はすべての等高線を同じ線種で描画するので、凡例には一つのラベルのみを書きます。このコマンドは、古いコマンド **unset clabel** を置き換えるものです。

等高線制御 (cntrparam)

set cntrparam は等高線の生成方法、およびそれを滑らかに描画する方法を制御します。**show contour** は現在の **contour** の設定だけでなく **cntrparam** の設定をも表示します。

書式:

```
set cntrparam { { linear
                | cubicspline
                | bspline
                | points <n>
                | order <n>
                | levels { <n>
                           | auto {<n>}
                           | discrete <z1> {,<z2>{,<z3>...}}
                           | incremental <start>, <incr> {,<end>}
                         }
                  {{un}sorted}
                  {firstlinetype N}
                }
              }
show cntrparam
```

このコマンドは 2 つの機能を持っています。一つは等高線上の点を決めるための z の値の設定です。等高線のレベルの数 <n> は整数型の定数式でなければいけません。<z1>, <z2> ... は実数値の数式です。もう一つは、個々の等高線の見た目の制御です。

等高線の平滑化を制御するキーワード:

linear, **cubicspline**, **bspline** — 近似 (補間) 方法を指定します。**linear** ならば、等高線は曲面から得られた値を区分的に直線で結びます。**cubicspline** (3 次スプライン) ならば、区分的な直線はいくぶんなめらかな等高線が得られるように補間されますが、多少波打つ可能性があります。**bspline** (B-spline) は、より滑らかな曲線を描くことが保証されますが、これは z の等しい点の位置を近似しているだけです。

points — 最終的には、全ての描画は、区分的な直線で行われます。ここで指定する数は、**bspline** または **cubicspline** での近似に使われる線分の数を制御します。実際には cubicspline と bspline の区間 (曲線線分) の数は **points** と線分の数の積に等しくなります。

order — bspline 近似の次数です。この次数が大きくなるにつれて、等高線はなめらかになります (もちろん、高次の bspline 曲線になるほど、元の区分的直線からは離れていきます)。このオプションは **bspline** モードでのみ有効です。指定できる値は、2 (直線) から 10 までの整数です。

等高線レベルの選択を制御するキーワード:

levels auto — これがデフォルトです。<n> は仮のレベルの数であり、実際のレベルの数は、簡単なラベルを生成するように調節されます。曲面の z 座標が zmin から zman の範囲にあるとき、等高線はその間の dz

の整数倍になるように生成されます。ここで、dz は 10 のあるべき乗の 1, 2, 5 倍、のいずれかです (2 つの目盛りの間を丁度割り切るように)。

levels discrete — 等高線は指定された z = <z1>, <z2> ... に対して生成されます。指定した個数が等高線のレベルの個数となります。**discrete** モードでは、**set cntrparams levels <n>** という指定は常に無視されます。

levels incremental — 等高線は z = <start> から始まり、<increment> ずつ増えて行き限界の個数に達するまで書かれます。<end> はその等高線の数を決定するのに使いますが、これは後の **set cntrparam levels <n>** によって常に変更されます。z 軸が対数軸の場合、**set ztics** の場合と同様に、<increment> は倍数として解釈し、<end> は使用しません。

等高線の線種の割り当てを制御するキーワード:

デフォルトでは、等高線は指定の逆順に生成します (**unsorted**)。すなわち、**set cntrparam levels increment 0, 10, 100** は、100 から始まって、0 で終わる 11 本の等高線を作ります。キーワード **sorted** を追加すると、数値の増加方向の順の生成に変更し、例えば今の例では、最初に 0 の等高線を書くようになります。

デフォルトでは、等高線は、対応する曲面に使用した線種の、次からの線種列で描きます。すなわち、**splot x*y lt 5** の最初の等高線は線種 6 です。**hidden3d** モードが有効な場合、各曲面には 2 つの線種を使うので、デフォルトの設定では、最初の等高線と曲面の裏面の描画に同じ線種を使ってしまいますが、これは望ましくありません。これを避けるには、以下の 2 つの方法があります。(1) **set hidden3d offset N** により、曲面の裏面の線種を変更すること。**offset -1** とするのがいいですが、これならすべての等高線の線種とぶつかりません。(2) オプション **set cntrparam firstlinetype N** により、曲面で使用する線種とは独立な、等高線で使用する線種群を指定すること。これは、特に等高線の線種をカスタマイズしたい場合には有用でしょう。N <= 0 の場合はデフォルトに戻ります。

コマンド **set cntrparam** を引数無しで使用すると、指定したすべてのオプション値をデフォルトにリセットします。

```
set cntrparam order 4 points 5
set cntrparam levels auto 5 unsorted
set cntrparam firstlinetype 0
```

Cntrparam の例 (cntrparam examples)

例:

```
set cntrparam bspline
set cntrparam points 7
set cntrparam order 10
```

以下はレベルの基準が合えば 5 個のレベルがに自動的に選択されます:

```
set cntrparam levels auto 5
```

以下は .1, .37, .9 にレベルを設定します:

```
set cntrparam levels discrete .1,1/exp(1),.9
```

以下は 0 から 4 まで、1 ずつ増やすレベルを設定します:

```
set cntrparam levels incremental  0,1,4
```

以下はレベルの数を 10 に設定します (増加の最後の値 (end) または自動で設定されるレベルの数は変更されます):

```
set cntrparam levels 10
```

以下はレベルの数は保持したままレベルの開始値と増分値を設定します:

```
set cntrparam levels incremental 100,50
```

以下はカスタマイズした等高線の線種群を定義し、使用します:

```
set linetype 100 lc "red" dt '....'
do for [L=101:199] {
    if (L%10 == 0) {
        set linetype L lc "black" dt solid lw 2
    } else {
        set linetype L lc "gray" dt solid lw 1
    }
}
set cntrparam firstlinetype 100
set cntrparam sorted levels incremental 0, 1, 100
```

等高線を描く場所の制御に関しては、以下参照: **set contour (p. 207)**。等高線のラベルの書式と線種の制御に関しては、以下参照: **set cntrlabel (p. 203)**。

以下も参照してください。等高線のデモ (contours.dem)

およびユーザ定義レベルの等高線のデモ (discrete.dem).

カラーボックス (colorbox)

パレットでの色付けを使用するグラフにおいて、特に pm3d でのグラフでは、パレットのグラデーションを、**unset colorbox** でスイッチがオフになっていない限り、グラフの横のカラーボックス (colorbox) 内に描きます。

```
set colorbox
set colorbox {
           { vertical | horizontal } {{no}invert}
           { default | bottom | user }
           { origin x, y }
           { size x, y }
           { front | back }
           { noborder | bdefault | border [line style] }
         }
show colorbox
unset colorbox
```

グラデーションの方向は、オプション **vertical** (縦) と **horizontal** (横) で設定します。

カラーボックスの位置は、**default**, **bottom**, **user** のいずれかを指定できます。キーワード **bottom** は、以下と同等の便利なショートカットです:

```
set colorbox horizontal user origin screen 0.1, 0.07 size 0.8, 0.03.
```

bottom で指定した場合のようにカラーボックスがグラフの下にあるときは、それ用に追加スペースを取ると便利でしょう: **set bmargin screen 0.2**。

origin x, y と **size x, y** は、**user** か **bottom** の配置での正確な位置合わせのために使います。x, y の値は、デフォルトではスクリーン座標と解釈しますが、これは 3 次元グラフに関しては形式的なオプションに過ぎません。**set view map** による splot を含む 2 次元描画では、任意の座標系が使えます。

back/front は、カラーボックスをグラフより前に書くか、後で書くかを制御します。

border は境界描画を ON にします (デフォルト) し、**noborder** は境界描画を OFF にします。**border** の後ろに正の整数を与えると、それを境界を描画する時の line style のタグとして使います。例えば:

```
set style line 2604 linetype -1 linewidth .4
set colorbox border 2604
```

は line style **2604**、すなわち細い線のデフォルトの境界色 (-1) で境界を描画します。**bdefault** (デフォルト) は、カラーボックスの境界の描画にデフォルトの境界の line style を使います。

カラーボックスの軸は **cb** と呼ばれ、通常の軸のコマンドで制御されます。すなわち対象となる属性 **cbrange**, **[m]cbtics**, **format cb**, **grid [m]cb**, **cblabel** に対する **set/unset/show**、そして多分 **cbdata**, **[no]cbdtics**, **[no]cbmtics** なども使えるでしょう。

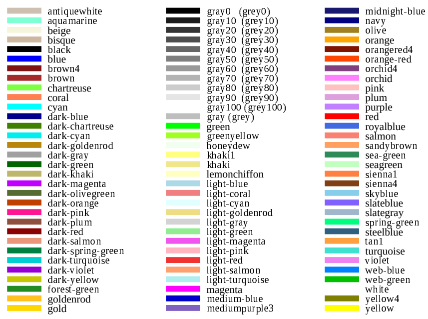
パラメータ無しの **set colorbox** はデフォルトの位置へ切替えます。**unset colorbox** はカラーボックスのパラメータをデフォルト値にリセットし、その上でカラーボックスを OFF にします。

以下も参照: **set pm3d (p. 267)**, **set palette (p. 258)**, **set style line (p. 285)**。

色名 (colornames)

gnuplot は限定された個数の色の名前を持っています。これらは、pm3d パレットでつながれる色の範囲を定義するのに、あるいは個々の線種やラインスタイルの色を定義しするのに、または現在のカラーパレットに対するグラデーションを定義するのに使えます。コマンド **show colornames** を使用することで、持っている色名の一覧とその RGB 成分の定義を見ることができます。例:

```
set style line 1 linecolor "sea-green"
set palette defined (0 "dark-red", 1 "white")
print sprintf("0x%06x", rgbcolor("dark-green"))
      0x006400
```



等高線 (contour)

コマンド **set contour** で 3 次元曲面に等高線を置くことができます。このオプションは **splot** でのみ有効です。これは、格子状データ (grid data) を必要とします。すなわち、一つの y-孤立線に対するすべての点が並べ

られ、その次に次の y-孤立線に対するすべての点が並び、以下同様に続くデータのファイルです。y-孤立線同士を分離するのは単一の空行 (空白以外の文字を一切含まない行) です。詳細は以下参照: **grid_data (p. 325)**。

データが格子状になっていない場合、**set dgrid3d** を使って最初に適切な格子を生成し、その格子にデータを当てはめることができます。

書式:

```
set contour {base | surface | both}
unset contour
show contour
```

これらの 3 つのオプションは等高線をどこに引くかを指定します。**base** では等高線を x/y 軸の刻みのある底面に描かれ、**surface** では等高線はその曲面自体の上に描かれ、**both** では底面と曲面上の両方に描かれます。オプションが指定されていない場合は **base** であると仮定されます。

等高線の描画に影響を与えるパラメータについては、以下参照: **set cntrparam (p. 204)**。等高線のラベルの制御に関しては、以下参照: **set cntrlabel (p. 203)**。

このオプションは、線やラベルを配置するもので、曲面自身のその他の見た目は変えないことに注意してください。曲面の色付けを、等高線で区切られた領域が他と区別できるような色に割り当てたい場合は、代わりに描画スタイル contourfill を使用してください。以下参照: **contourfill (p. 98)**。

set contour が有効な場合、**splot with <style>** で points, lines, impulses, labels 等の描画要素を等高線に沿って配置できます。**with pm3d** は、pm3d 曲面を生成し、さらに等高線も書きます。**set contour** が有効な際に生成する等高線に、ファイルから読み込んだラベルなどのその他の描画要素も混ぜたい場合は、splot コマンド内のその命令の後ろにキーワード **nocontours** を追加しないといけません。

曲面をスイッチオフ (以下参照: **unset surface (p. 289)**) にして、等高線のみのグラフにすることも可能です。等高線の 2 次元射影とオプションのラベルは以下のようにして生成できます:

```
set view map
splot DATA with lines nosurface, DATA with labels
```

以前のバージョンの gnuplot では、以下のように、代わりとなる多段階の方法を使って 3 次元の等高線をファイルやデータブロックに保存し、その後でそれを 2 次元の plot コマンドを使用してそれを描画していました。

```
set contour
set table $datablock
splot DATA with lines nosurface
unset table
# 等高線は $datablock に、1 つの等高線が 1 つの index で
plot for [level=0:*] $datablock index level with lines
```

以下も参照: **splot datafile (p. 320)**。また、^ 等高線のデモ (contours.dem) ^ と ^ 等高線レベルのユーザ定義のデモ (discrete.dem) ^ も参照。

グラフ角の支柱 (cornerpoles)

デフォルトで、splot は 3 次元曲面のそれぞれの角から床面への鉛直線を描画します。これらの鉛直線は、**unset cornerpoles** で消すことができます。

等高線間の塗り潰し (contourfill)

3 次元描画スタイル **with contourfill** は、1 つの pm3d 曲面を、z 軸に垂直な平面群で区切られる断片に切り分けます。コマンド **set contourfill** は、区切る平面群の配置と個々の断片に割り当てる色を制御します。

書式:

```
set contourfill auto N            # zrange を N 等分した断片に
set contourfill ztics             # z 軸の各主目盛りで切り分け
set contourfill cbtics            # cb 軸の各主目盛りで切り分け
set contourfill {palette | firstlinetype N}
set contourfill defined [zmin1:zmax1] color1, [zmin2:zmax2] color2, ...
```

デフォルトは **set contourfill auto 5 palette** で、これは現在の z 軸の範囲を 5 等分し (6 つの平面で区切る)、その曲面の各断片にその z の中点に対応するパレットの色で色付けます。

オプション **ztics**, **cbtics** は、zrange の分割を、その軸の主目盛りの位置で行います。例えば、z=2.5, z=7, z=10 に指定して切り分けるには、以下のコマンドを使用します。

```
set ztics add ("floor" 2.5, "boundary X" 7, "ceiling" 10)
set contourfill ztics
```

断片の色付けにパレットを使いたくない場合は、任意連続する線種の範囲を選択し、それらに望む色の列を割り当てることができます。

```
set for [i=101:110] linetype i lc mycolor[i]
set contourfill firstlinetype 101
```

set contourfill palette でパレットによる色付けに復帰できます。

他に、z の範囲のリストを、それぞれの範囲に色名や 16 進の RGB 色値をつけて与えることもできます:

```
set contourfill defined [-10:-1] "blue", [-1:1] 0Xeeffee, [1:10] "red"
```

点線/破線設定 (dashtype)

コマンド **set dashtype** は、点線/破線パターンを番号で参照できるように登録します。これはとても便利で、その点線/破線パターンをその番号で受けつけてくれる場所ならば、どこでも明示的な点線/破線パターンも受けつけてくれます。例:

```
set dashtype 5 (2,4,2,6)    # 5 番の dashtype を定義または再定義
plot f1(x) dt 5             # その dashtype を使って plot
plot f1(x) dt (2,4,2,6)     # 上と全く同じグラフ
set linetype 5 dt 5         # このパターンを linetype 5 で常に使う
set dashtype 66 "..-"       # 文字列で新しい dashtype を定義
```

以下参照: **dashtype (p. 73)**。

Datafile

コマンド **set datafile** は、**plot**, **splot**, **fit** コマンドで入力データを読む場合に、その列 (field) の解釈の仕方を制御するオプションを持ちます。現在は、そのようなオプションがいくつか実装されています。その設定は、その後のコマンドで読み出されるすべてのデータファイルに一様に適用しますが、それとは矛盾する書式でファイルを同時に操作しなければいけない場合、これを回避する方法に関しては、以下参照: **functionblocks (p. 148)**。

Set datafile columnheaders

コマンド **set datafile columnheaders** は、入力の最初の行を、データ値としてではなく、columnheader として解釈することを保証します。これは、plot, splot, fit, stats の各コマンドの入力データ源すべてに影響します。この設定を **unset datafile columnheaders** で無効にすると、明示的な columnheader() 関数が using 指定にあるか、plot タイトルがファイルに関連づけされている場合、同じ効果がファイル毎にオンにされます。以下参照: **set key autotitle (p. 231)**, **columnheader (p. 185)**。

Set datafile fortran

コマンド **set datafile fortran** は、入力ファイルの Fortran D 型、Q 型の定数値の特別なチェックを可能にします。この特別なチェックは入力処理を遅くしますので、実際にそのデータファイルが Fortran D 型、Q 型の定数を持っている場合にのみこれを選択すべきです。このオプションは、その後で **unset datafile fortran** を行えば無効にできます。

Set datafile nofpe_trap

コマンド **set datafile nofpe_trap** は、入力ファイルからデータの読み込みの際に、すべての数式の評価の前に浮動小数点例外ハンドラの再初期化をしないように gnuplot に命令します。これにより、とても大きなファイルからのデータの入力がかなり速くなりますが、浮動小数点例外が起きた場合にプログラムが異常終了してしまう危険はあります。

Set datafile missing

書式:

```
set datafile missing "<string>"
set datafile missing NaN
show datafile missing
unset datafile
```

コマンド **set datafile missing** は、入力データファイル中で欠損データを記述する特別な文字列があることを **gnuplot** に指示します。**missing** に関するデフォルト値 (文字) はありません。gnuplot は「欠損データ」と「無効な値」(例えば "NaN" や 1/0) を区別します。例えば、連続するデータ点に対するグラフの折れ線描画は、無効な値によってそこで切れますが、欠損データの場合はそうではありません。

数値が期待される場面で数値ではない文字が現れた場合は、それが **missing** で指定する文字列にマッチする場合を除いて、通常欠損データではなく、無効な値として解釈します。

逆に、**set datafile missing NaN** とすると、数式やデータ中の数値ではない値 (NaN) はすべて欠損データとして扱います。imageNaN デモ

を参照してください。

gnuplot は plot コマンドで using 指定が直接列の値を **using N**, **using ($N)**, **using (function($N))** のように参照した場合は列 N に欠損値フラグを通知します。これらの場合は、数式、例えば func($N) は一切評価されません。

現在のバージョンの gnuplot は、(column(N)) の形式の直接参照も通知しますが、式が "missing" (欠損値) のフラグがついた列の値に間接的に依存する場合でも、評価の最中に通知します。

これらすべての場合で、gnuplot は入力データ行全体をそれが全くなかったものとして扱います。しかし、式が真に欠けているデータ値に (例えば csv ファイルの空フィールドのように) 依存している場合、それはこれらのチェックをすりぬけるかもしれません。それを NaN 値と評価すれば、それは欠損データ点ではなく、不正なデータとして扱います。もしそのような不正値をすべて欠損値として等しく扱いたい場合は、コマンド **set datafile missing NaN** を使用してください。

Set datafile separator

コマンド **set datafile separator** は、この後の入力ファイルのデータ列の分離文字が、空白 (whitespace) でなくて、ここで指定する文字であると **gnuplot** に指示します。このコマンドの最も一般的な使用例は、表計算ソフトやデータベースソフトが作る csv (コンマ区切り) ファイルを読む場合でしょう。デフォルトのデータ列の分離文字は空白 (whitespace) です。

書式:

```
set datafile separator {whitespace | tab | comma | "<chars>"}
```

例:

```
# タブ区切りのファイルを入力
set datafile separator "\t"

# コンマ区切りのファイルを入力
set datafile separator comma

# 入力ファイルが * か | のいずれかで区切られた列を持つ場合
set datafile separator "*|"
```

Set datafile commentschars

コマンド **set datafile commentschars** は、データファイル中のコメント行の開始文字としてどの文字を使うかを指定します。指定した文字の中の一つがデータ行の最初の非空白文字として現われた場合、そのデータ行のそれ以降の部分を無視します。デフォルト文字列は、VMS では "#!"、それ以外では "#" です。

書式:

```
set datafile commentschars {"<string>"}
show datafile commentschars
unset commentschars
```

よって、データファイルの以下の行は完全に無視されます:

```
# 1 2 3 4
```

が、以下の行

```
1 # 3 4
```

は、2 列目にゴミがあり、その後に有効なデータが 3 列目と 4 列目にあると認識されます。

例:

```
set datafile commentschars "#!%"
```

Set datafile binary

コマンド **set datafile binary** は、データファイルの読み込み時にバイナリファイルをデフォルトと設定するのに使われます。書式は、それが **plot** または **splot** コマンドで使われるのと正確に同じです。<binary list> に書けるキーワードに関しては、詳しくは、以下参照: **binary matrix (p. 321)**, **binary general (p. 157)**。

書式:

```
set datafile binary <binary list>
show datafile binary
show datafile
unset datafile
```

例:

```
set datafile binary filetype=auto
set datafile binary array=(512,512) format="%uchar"

show datafile binary   # 現在の設定の一覧表示
```

小数点設定 (desimalsign)

コマンド **set decimalsign** は、目盛りの見出し、あるいは **set label** 文字列に書かれる数の小数点記号を選択します。

書式:

```
set decimalsign {<value> | locale {"<locale>"}}
unset decimalsign
show decimalsign
```

引数 <value> は、通常の小数点記号に置き換えて使う文字列です。典型的なものはピリオド '.' やコンマ ',' ですが他にも有用なものがあるでしょう。引数 <value> を省略すると、小数点の区切りはデフォルト (ピリオド) から変更されません。unset decimalsign も <value> を省略するのと同じ効果を持ちます。

例:

多くのヨーロッパ諸国での正しい出力形式を得るには:

```
set decimalsign ','
```

次のことに注意してください: 明示的な文字列を設定した場合、これは軸の目盛りなどの gnuplot の gprintf() 書式関数で出力される数値のみに影響し、入力データの書式指定や sprintf() 書式関数で出力される数値には影響しません。それらの入力や出力の形式の挙動も変更したい場合は、代わりに以下を使用してください:

```
set decimalsign locale
```

これは、gnuplot に、入力と出力の書式を、環境変数 LC_ALL, LC_NUMERIC, LANG の現在の設定に従ったものを使わせるようにします。

```
set decimalsign locale "foo"
```

これは、gnuplot に、入力と出力の書式を、ロケール "foo" に従ったものにしますが、そのロケールがインストールされている必要があります。もしロケール "foo" が見つからなかった場合、エラーメッセージが出力され、小数点の設定は変更されません。linux システム上では、そこにインストールされているロケールの一覧は "locale -a" で見ることができます。linux のロケール文字列はだいたい "sl_SI.UTF-8" のような形式をしていますが、Windows のロケール文字列は "Slovenian_Slovenia.1250"、または "slovenian" のような形式です。ロケール文字列の解釈は、C のランタイムライブラリが行うことに注意してください。古い C ライブラリでは、ロケール設定のサポート (例えば数字の 3 桁毎の区切り文字など) を部分的にしか提供していないかもしれません。

```
set decimalsign locale; set decimalsign "."
```

これは、現在のロケールに合ったどんな小数点でも、全ての入出力に対して使用するように設定しますが、gnuplot の内部関数 gprintf() を使って書式化する数値は明示的に指定された '.' になります (上書き)。

格子状データ処理 (dgrid3d)

コマンド **set dgrid3d** は、非格子状データから格子状データへの写像機能を有効にし、そのためのパラメータを設定します。格子状データの構造についての詳細は、以下参照: **splot grid_data (p. 325)**。この処理は、3 次元曲面への当てはめで利用できること以外に、2 次元温度分布図を生成するのにも使えます。その場合、各点の 'z' の値は局所的な重み付けに寄与します。

書式:

```
set dgrid3d {<rows>} {,{<cols>}} splines
set dgrid3d {<rows>} {,{<cols>}} qnorm {<norm>}
set dgrid3d {<rows>} {,{<cols>}} {gauss | cauchy | exp | box
                | hann} {kdensity} {<dx>} {,<dy>}
unset dgrid3d
show dgrid3d
```

デフォルトでは **dgrid3d** は無効になっています。有効な場合は、ファイルから読み込まれる 3 次元のデータ点は、格子曲面に当てはめるのに使用する「散在した」データ (非格子状データ) であると見なされます。格子の寸法は、**set dgrid3d** 文で与えるパラメータ row_size/col_size の行数、列数で再分割する散在データを囲む矩形 (bounding box) から求めます。格子は x 方向 (行) と y 方向 (列) に等間隔です。z の値は散在するデータの z の値の重み付きの平均、またはスプライン補間として計算します。言い変えれば、規則的な間隔の格子を生成し、全ての格子点で元データの滑かな近似値を評価します。そして元データの代わりにこの曲面を描画します。

dgrid3d モードが有効な間は、これを使用せずに格子曲面を生成する個々の点や線分を描きたい場合は、対応する splot コマンドにキーワード **nogrid** を追加する必要があります。

デフォルトの列の数は行の数に等しく、そのデフォルトの値は 10 です。

元のデータから近似値を計算するためのいくつかのアルゴリズムが用意されていて、追加のパラメータを指定できるものもあります。これらの補間は、格子点に近いデータ点ほど、その格子点に対してより強い影響を与えます。

splines アルゴリズムは、薄いつぎ板を元にした補間計算を行います。これは追加パラメータを取りません。

qnorm アルゴリズムは各格子点で入力データの重み付き平均を計算します。各点には、格子点からの距離の逆数のあるベキ乗で重み付けします。そのベキは追加パラメータの整数値として指定できますが、デフォルトは 1 です。このアルゴリズムがデフォルトになっています。

最後に、重み付き平均の計算用に、いくつかの平滑化重み付け関数 (kernel) が用意されています: z = Sum_i w(d_i) * z_i / Sum_i w(d_i), ここで z_i は i 番目のデータの値で、d_i は現在の格子点と i 番目のデータ点の位置との距離です。すべての重み付け関数が、現在の格子点に近い方のデータ点には大きな重み、遠い方のデータ点には小さい重みを付けます。

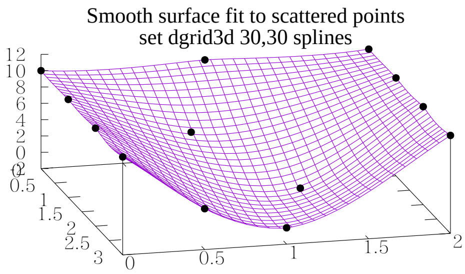
以下の重み付け関数が使用できます:

```
gauss :     w(d) = exp(-d*d)
cauchy :    w(d) = 1/(1 + d*d)
exp :       w(d) = exp(-d)
box :       w(d) = 1                   d<1 の場合
                 = 0                   その他
hann :      w(d) = 0.5*(1+cos(pi*d))   d<1 の場合
            w(d) = 0                   その他
```

これら 5 つの平滑化重み付け関数のうち一つを使用する場合、2 つまでの追加パラメータ dx と dy を指定できます。これらは、距離の計算時に座標の違いをスケール変換するのに使えます: d_i = sqrt(((x-x_i)/dx)**2 + ((y-y_i)/dy)**2), ここで、x,y は現在の格子点の座標で、x_i,y_i は i 番目のデータ点の座標です。dy のデフォルトの値は dx で、そのデフォルトの値は 1 になっています。パラメータ dx と dy は、データ点が格子点へ「データそれ自身の単位で」の寄与を行う範囲の制御を可能にします。

オプションキーワード **kdensity** は、重み付け関数名の後ろでオプションのスケール変換のパラメータの前に置くもので、これはアルゴリズムを変更して、格子点用に計算する値を重みの和 (z = Sum_i w(d_i) * z_i) では割らないようにします。z_i がすべて定数の場合、これは事実上 2 変数の重み付け評価を描画します: (上の 5 つのうちの一つの) 重み付け関数が各データ点に置かれ、それらの重みの和がすべての格子点で評価され、そして元のデータの代わりにこの滑らかな曲面が描画されます。これは、1 次元のデータ集合に対する **smooth kdensity** オプションが行うこととおおまかには同じです。使用例に関しては、kdensity2d.dem, heatmap_points.dem を参照してください。

オプション **dgrid3d** は、散在するデータを重み付き平均で規則的な格子に置き変える単純な仕組みに過ぎません。この問題に対するより洗練された手法が存在しますので、この単純な方法が不十分であれば、**gnuplot** の外でそのような方法でデータを前処理するべきでしょう。



ネット上の以下のデモも参照^ dgrid3d ^ ^ scatter ^ heatmap_points

仮変数 (dummy)

コマンド **set dummy** はデフォルトの仮変数名を変更します。

書式:

```
set dummy {<dummy-var>} {,<dummy-var>}
show dummy
```

デフォルトでは、**gnuplot** は **plot** では、媒介変数モード、あるいは極座標モードでは "t", そうでなければ "x" を独立変数 (仮変数) とし、同様に **splot** では、媒介変数モードでは (**splot** は極座標モードでは使えません) "u" と "v", そうでなければ "x" と "y" を独立変数とします。

仮変数は、物理的に意味のある名前、あるいはより便利な名前として使う方が便利でしょう。例えば、時間の関数を描画する場合:

```
set dummy t
plot sin(t), cos(t)
```

例:

```
set dummy u,v
set dummy ,s
```

第二の例は、2 番目の変数を s とします。仮変数名をデフォルトの値に戻すには以下のようにしてください。

```
unset dummy
```

文字エンコード (encoding)

コマンド **set encoding** は文字のエンコード (encoding) を選択します。

書式:

```
set encoding {<value>}
set encoding locale
show encoding
```

有効な値 (value) は以下の通りです。

```
default       - 出力形式にデフォルトのエンコードの使用を命令
iso_8859_1    - UTF-8 より最も一般的な西ヨーロッパエンコード。このエ
                ンコードは PostScript の世界での 'ISO-Latin1' です。
iso_8859_15 - ユーロ記号を含む iso_8859_1 の亜種
iso_8859_2    - 中央/東ヨーロッパで使用されるエンコード
iso_8859_9    - (Latin5 として知られる) トルコで使用されるエンコード
koi8r         - 良く使われる Unix のキリル文字エンコード
koi8u         - Unix のウクライナ地方のキリル文字エンコード
cp437         - MS-DOS のコードページ
cp850         - 西ヨーロッパの OS/2 のコードページ
cp852         - 中央/東ヨーロッパの OS/2 のコードページ
cp950         - MS 版の Big5 (emf terminal のみ)
cp1250        - 中央/東ヨーロッパの MS Windows のコードページ
cp1251        - ロシア、セルビア、ブルガリア、マケドニア語 (8 ビット)
cp1252        - 西ヨーロッパの MS Windows のコードページ
cp1254        - トルコの MS Windows のコードページ (Latin5 の拡張)
sjis          - Shift_JIS 日本語エンコード
utf8          - 各文字の Unicode エントリポイントの、可変長 (マルチバ
                イト) 表現
```

コマンド **set encoding locale** は、他のオプションとは違い、これは現在のロカールを実行時の環境から決定しようとします。たいていのシステムではこれは環境変数 LC_ALL, LC_CTYPE, LANG のいずれかによって制御されます。この仕組みは、例えば wxt, pdf 出力形式で、UTF-8 や EUC-JP のようなマルチバイト文字エンコードを通すために必要です。このコマンドは日付や数字などのロカール特有の表現には影響を与えません。以下も参照: **set locale (p. 240)**, **set decimalsign (p. 212)**。

一般にエンコードの設定は、それがフォントの選択に影響を与えるように、出力形式の設定の前に行なう必要があります。

誤差線の端 (errorbars)

コマンド **set errorbars** は、誤差グラフ (errorbar) の両端、および boxplot につく箱ひげの両端のマークを制御します。

書式:

```
set errorbars {small | large | fullwidth | <size>} {front | back}
              {line-properties}
unset errorbars
show errorbars
```

small は 0.0 (交差線なし)、**large** は 1.0 と同じです。サイズを指定しなければデフォルトの値は 1.0 です。

キーワード **fullwidth** は、errorbar を伴う boxplot と histograms にのみ関連します。これは errorbar の両端の幅を、対応する箱の幅と同じに設定しますが、箱の幅自体を変更することはありません。

キーワード **front**, **back** は、塗り潰し長方形のついた errorbar のみに関連します (boxes, candlesticks, histograms)。

誤差線 (errorbar) は、デフォルトでは関連する箱の境界線と同じ線属性で描画しますが、これを誤差線用に用意した線属性に変更できます。

```
set errorbars linecolor black linewidth 0.5 dashtype '.'
```

非線形関数回帰 (fit)

コマンド **set fit** は、**fit** コマンド用のオプションを制御します。

書式:

```
set fit {nolog | logfile {"<filename>"|default}}
        {{no}quiet|results|brief|verbose}
        {{no}errorvariables}
        {{no}covariancevariables}
        {{no}errorscaling}
        {{no}prescale}
        {maxiter <value>|default}
        {limit <epsilon>|default}
        {limit_abs <epsilon_abs>}
        {start-lambda <value>|default}
        {lambda-factor <value>|default}
        {script {"<command>"|default}}
        {v4 | v5}
unset fit
show fit
```

オプション **logfile** は、**fit** コマンドがその出力を書き出す場所を定義します。 引数 <filename> は、単一引用符か二重引用符で囲む必要があります。ファイル名を指定しなかった場合、または **unset fit** を使用した場合は、ログファイルはデフォルトの値である "fit.log"、または環境変数 **FIT_LOG** の値にリセットされます。与えられたログファイル名が / か \ で終っている場合、それはディレクトリ名と解釈され、ログファイルはそのディレクトリの "fit.log" となります。

デフォルトでは、そのログファイルに書かれる情報は、対話型出力にも出力します。**set fit quiet** はその対話型出力をオフにし、**results** は最終結果のみを出力します。**brief** は、追加で fit のすべての繰り返しに関して 1 行の要約を提供します。**verbose** は、バージョン 4 のような詳細な繰り返しの報告を行います。

オプション **errorvariables** を ON にすると、**fit** コマンドで計算された個々の当てはめパラメータの誤差が、そのパラメータの名前に "_err" をつけた名前のユーザ定義変数にコピーされます。これは主に、当てはめ関数とデータの描画グラフの上にパラメータとその誤差を参照用に出力するのに使われます。例:

```
set fit errorvariables
fit f(x) 'datafile' using 1:2 via a, b
print "error of a is:", a_err
set label 1 sprintf("a=%6.2f +/- %6.2f", a, a_err)
plot 'datafile' using 1:2, f(x)
```

オプション **errorscaling** を指定すると (デフォルト)、パラメータの計算誤差を補正χ 自乗 (reduced χ -square) で伸縮します。これは、結果として補正χ 自乗値になる、当てはめ計算の標準偏差 (FIT_STDFIT) に等しいデータ誤差を提供することと同等になります。オプション **noerrorscaling** では、評価誤差は、伸縮されない当てはめパラメータの標準偏差になります。データの重みを指定しなければ、パラメータの誤差は常に伸縮されます。

オプション **prescale** をオンにすると、Marquardt-Levenberg ルーチンに渡す前に、各パラメータの値をそれらの初期値に従って事前にスケール変換します。これは、各パラメータの大きさにかなり大きな違いがある場合に、大変有効です。ただし、初期値が完全に 0 の当てはめパラメータには、決してこのスケール変換は行いません。

反復数の限界値は、オプション **maxiter** で制限できます。それを 0 か **default** とすると、それは限界がないことを意味します。

オプション **limit** は、収束を検出するためのもっとも小さい数字の限界 (1e-5) のデフォルトの値を変更するのに使えます。自乗残差の和がこの数値未満の比率の変化しかしない場合は、当てはめは「収束した」と判断されます。オプション **limit_abs** は、自乗残差の和の変化の限界 (絶対値) を追加します。デフォルトは 0 です。

アルゴリズムに関する別の制御をしたい場合、そして Marquardt-Levenberg アルゴリズムを良く知っている場合、それに影響を与える以下のオプションが利用できます: **lambda** の初期値は、通常自動的に ML-行列から計算されますが、必要ならばオプション **start_lambda** を使ってそれを与えることができます。それを **default** とすると、再び自動設定が有効になります。オプション **lambda_factor** は、対象とする関数のχ 自乗値が意味ありげに増加する/減少するときは常に **lambda** を増加させる/減少させる因子を設定します。それを **default** とすると、デフォルトの因子である 10.0 にします。

オプション **script** は、fit を中断したときに実行する **gnuplot** コマンドを指定するものです。以下参照: **fit (p. 137)**。この設定はデフォルトの **replot** や環境変数 **FIT_SCRIPT** よりも優先順位は上です。

オプション **covariancevariables** をオンにすると、最終的なパラメータ間の共分散をユーザ定義変数に保存します。各パラメータの組に対してその共分散を保存する変数名は、"FIT_COV_" に最初のパラメータ名と "_" と 2 つ目のパラメータをつなげた名前になります。例えばパラメータ "a" と "b" に対しては、その共分散変数名は "FIT_COV_a_b" となります。

バージョン 5 では、コマンド fit の書式は変更され、キーワード **error** が指定されていない場合は単位重み (**unitweights**) がデフォルトになりました。オプション **v4** で gnuplot バージョン 4 のデフォルトの挙動に戻ります。以下も参照: **fit (p. 137)**。

フォントパス (fontpath)

書式:

```
set fontpath "/directory/where/my/fonts/live"
set term postscript fontfile <filename>
```

[version 5.4 では非推奨]

fontpath のディレクトリは、postscript 出力形式が作る PostScript 出力内に埋め込むフォントにのみ関係します。他の gnuplot 出力形式には何の影響も与えません。あなたがフォントを埋めこまなければ、このコマンドはあなたには必要ありませんし、埋めこむ場合でも、以下に示す他のパスにフォントが見つからない場合にのみ必要なだけです。

以前の版の gnuplot は、フォントを含む複数のディレクトリツリーを探索することでフォント管理ソフトをまねていました。しかし現在は、以下の場所を検索する方法に置き換わっています。(1) **set term postscript fontfile** コマンドで与えた絶対パス (2) 現在のディレクトリ (カレントディレクトリ) (3) **set loadpath** で指定したディレクトリのすべて (4) **set fontpath** で指定したディレクトリ (5) 環境変数 GNUPLOT_FONTPATH に指定されているディレクトリ

注意: libgd の出力形式 (png gif jpeg sixel) 用にファイル名で指定するフォントの検索パスは、環境変数 GDFONTPATH で制御できます。

軸の刻み書式 (format)

座標軸の刻みの見出しは、コマンド **set format** または **set tics format** または個別にコマンド **set {** 軸 **}tics format** で書式を設定できます。入力データに対する明示的な書式の使用法については、以下参照: **using format (p. 177)**。

書式:

```
set format {<axes>} {"<format-string>"} {numeric|timedate|geographic}
show format
```

ここで、<axes> (軸) は **x**, **y**, **xy**, **x2**, **y2**, **z**, **cb**、または何も指定しないか (その場合その書式はすべての軸に適用されます) のいずれかです。以下の 2 つのコマンドは全く同等です:

```
set format y "%.2f"
set ytics format "%.2f"
```

書式の型 (numeric (数値)、timedate (日時)、geographic (地理座標)) は、書式文字列の解釈を変更します。数値書式は gnuplot の **gprintf** ルーチンで処理します。以下参照: **format specifiers (p. 220)**。これは、軸が timedate でも geographic でもない場合のデフォルトです。以下参照: **set xdata time (p. 302)**, **set xdata geographic (p. 311)**。

書式文字列の長さは 100 文字まで、と制限されています。デフォルトの書式文字列は "% h" で、LaTeX 系の出力形式では "$%h$" です。これは、例えば'set xtics format "%.2f MHz"' のように単位や、異なる数値書式を与えることで変更することも可能です。"set format" の後ろに何もつけずに実行すると、デフォルトに戻します。

空文字列 "" を指定した場合、刻み自身は表示しますが見出しはつけません。刻み自身を消すには、 **unset xtics** または **set tics scale 0** を使用してください。

書式文字列では、改行文字 (\n) や拡張文字列処理 (enhanced text) 用のマークアップも使えます。 この場合は、単一引用符 (') でなく (") を使ってください。以下も参照: **syntax (p. 84)**。"%" が頭につかない文字はそのまま表示されます。よって、書式文字列内にスペースや文字列などを入れることができます。例えば "%g m" とすれば、数値の後に " m" が表示されます。"%" 自身を表示する場合には "%g %%" のように 2 つ重ねます。

刻みに関するより詳しい情報については、以下も参照: **set xtics (p. 307)**。また、この方法で出力される数字にデフォルト以外の小数分離文字を使うやり方については、以下参照: **set decimalsign (p. 212)**。以下も参照エレクトロン (電子) デモ (electron.dem).

Gprintf

文字列関数 gprintf("format",x) は、gnuplot コマンドの **set format**, **set timestamp** などと同様の、gnuplot 独自の書式指定子を使います。これらの書式指定子は、標準的な C 言語の関数である sprintf() のものと全く同じではありません。gprintf() は、整形化される引数は一つしか受けつけません。そのために、gnuplot には sprintf("format",x1,x2,...) 関数も用意されています。gnuplot の書式オプションの一覧については、以下参照: **format specifiers (p. 220)**。

書式指定子 (format specifiers)

使用可能な書式 (時間/日付モードでない場合) は以下の通りです:

目盛りラベルの数値書式指定子	
書式	説明
`%f`	固定小数点表記
`%e`, `%E`	指数表記; 指数の前に "e", "E" をつける
`%g`, `%G`	`%e` (または `%E`) と `%f` の略記
`%h`, `%H`	`%g` に `"e%S"` でなく `"x10^{%S}"` か `"*10^{%S}"` をつける
`%x`, `%X`	16 進表記
`%o`, `%O`	8 進表記
`%C`, `%Ci`	複素数値
`%t`	10 進の仮数部
`%l`	現在の対数尺の底を基数とする仮数部
`%s`	現在の対数尺の底を基数とする仮数部; 補助単位 (scientific power)
`%T`	10 進の指数部
`%L`	現在の対数尺の底を基数とする指数部
`%S`	補助単位の指数部 (scientific power)
`%c`	補助単位文字
`%b`	ISO/IEC 80000 記法 (ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi) の仮数部
`%B`	ISO/IEC 80000 記法 (ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi) の接頭辞
`%P`	π の倍数

補助単位 ('scientific' power) は、指数が 3 の倍数であるようなものです。補助単位指数 ("%c") の文字への変換は -18 から +18 までの指数に対してサポートされています。この範囲外の指数の場合、書式は通常の指数形式に戻ります。

ほかに使うことのできる修飾詞 ("%" と書式指定子の間に書くもの) には、次のいくつかがあります: "-" は数字を左詰めにし、"+" は正の数にも符号をつけ、" " (空白一つ) は負の数に "-" をつけるべき場所に正の数の場合に空白を一つつけ、"#" は小数点以下の数字が 0 だけであっても小数点をつけ、正の整数は出力幅を定め、出力幅指定の直前の "0" (文字でなく数字) は先頭に空いた部分を空白で埋める代わりに 0 で埋め、小数点の後に非負の整数を書いたものは精度を意味します (整数の場合は最小桁、小数の場合は小数点以下の桁数)。

これらの全ての修飾詞をサポートしていない OS もあるでしょうし、逆にこれ以外のものをもサポートする OS もあるでしょう。疑わしい場合は、適切な資料を調べ、そして実験してみてください。

例:

```
set format y "%t"; set ytics (5,10)          # "5.0" と "1.0"
set format y "%s"; set ytics (500,1000)      # "500" と "1.0"
set format y "%+-12.3f"; set ytics(12345)    # "+12345.000  "
set format y "%.2t*10^%+03T"; set ytic(12345)# "1.23*10^+04"
set format y "%s*10^{%S}"; set ytic(12345)   # "12.345*10^{3}"
set format y "%s %cg"; set ytic(12345)       # "12.345 kg"
set format y "%.0P pi"; set ytic(6.283185)   # "2 pi"
set format y "%.0f%%"; set ytic(50)          # "50%"

set log y 2; set format y '%l'; set ytics (1,2,3)
#"1.0", "1.0", "1.5" と表示される (3 は 1.5 * 2^1 なので)
```

丸めと指数が必要となるような書式で 9.999 の様な数字が書かれる場合は問題が起こることがあります。

軸のデータ型が日時データ (time/date) の場合、書式文字列は 'strftime' 関数 ('gnuplot' 外。"man strftime" としてみてください) に関する有効な指定を行う必要があります。使える入力書式指定の一覧に関しては、以下参照: **set timefmt (p. 295)**。

複素数値書式指定子 (complex format specifiers)

gnuplot は、複素数値を出力する %C と %Ci の 2 つの書式指定子を提供します。いずれも、実数成分と虚数成分に別々に書式 %g を適用し、オプションのフィールド修飾子が与えられた場合はそれも使用します。

例えば複素数 Z = {1.2, -3.4} に対し、

```
gprintf("%C", Z) は文字列 "{1.2, -3.4}" を生成
gprintf("%Ci", Z) は文字列 "1.2 - 3.4i" を生成
```

"%Ci" の出力は、コマンド **set imaginary_i** でさらにカスタマイズでき、虚数成分につく文字 "i" を、違う文字、または文字列で置き換えることができます。以下参照: **imaginary_i (p. 226)**。

例:

```
Z = {1.2222, 0.3333}
gprintf("%.2C", Z)            # {1.2, 0.33}
gprintf("%.3Ci", Z)           # 1.22 + 0.333i
set imaginary_i "{/Times*0.5 im}"
gprintf("[%Ci]", Z)           # [1.2222 + 0.3333im]
gprintf("[%.1Ci]", -Z)        # [-1 - 0.3im]
```

日時データ指定子 (time/date specifiers)

日時書式指定には、日時指定と相対時刻の 2 つのグループがあります。これらは、軸の刻みのラベルを生成したり、時刻を文字列にエンコードするのに使われます。以下参照: **set xtics time (p. 310)**, **strftime (p. 50)**, **strptime (p. 50)**。

日時書式は以下の通りです。

日付指定子	
書式	説明
%a	曜日名の省略形 (Sun,Mon,...)
%A	曜日名 (Sunday,Monday,...)
%b, %h	月名の省略形 (Jan,Feb,...)
%B	月名 (January,February,...)
%d	日 (01–31)
%D	"%m/%d/%y" の簡略形 (出力のみ)
%F	"%Y-%m-%d" の簡略形 (出力のみ)
%k	時 (0–23; 1 桁または 2 桁)
%H	時 (00–23; 常に 2 桁)
%l	時 (1–12; 1 桁または 2 桁)
%I	時 (01–12; 常に 2 桁)
%j	その年の通算日 (001–366)
%m	月 (01–12)
%M	分 (00–60)
%p	"am" または "pm"
%r	"%I:%M:%S %p" の簡略形 (出力のみ)
%R	"%H:%M" の簡略形 (出力のみ)
%S	秒 (出力では 00–60 の整数、入力では実数)
%s	1970 年最初からの秒数
%T	"%H:%M:%S" の簡略形 (出力のみ)
%U	その年の通算週 (CDC/MMWR 疫学的週) (入力では無視)
%w	曜日番号 (0–6, 日曜 = 0)
%W	その年の通算週 (ISO 8601 の週番号) (入力では無視)
%y	西暦 (0-99、1969-2068 年の下 2 桁)
%Y	西暦 (4 桁)
%z	タイムゾーン、[+-]hh:mm
%Z	タイムゾーン名、文字列は無視

書式 %W (ISO の週番号) に関する詳細は、以下参照: **tm_week (p. 56)**。書式 %U (CDC/MMWR: アメリカ疾病予防管理センター疫学週報の疫学的週番号) は、週が月曜開始でなく日曜開始であることを除けば %W と同様です。警告: バージョン 5.4.2 より前の gnuplot では、書式 %W と %U はいずれも信頼できません。"week_date.dem" の単位テストを参照してください。

相対時刻書式は、時刻 0 地点のいずれかの側の時間間隔の長さを表現します。相対時刻書式は以下の通りです。

時刻指定子	
書式	説明
%tD	時刻 0 への相対的な正負の日付
%tH	時刻 0 への相対的な正負の時 (24 での巻戻しなし)
%tM	時刻 0 への相対的な正負の分
%tS	直前の tH, tM 項目に対応する正負の秒数

数字を表す書式には、先頭に 0 を埋めるために "0" (ゼロ) を前につけることができ、また最小の出力幅を指定

するために正の整数を前につけることもできます。書式 %S と %t は精度指定も受けつけるので、小数の時/分/秒を書くこともできます。

例 (Examples)　日付書式の例:

x の値が、1976 年 12 月 25 日の深夜少し前の時刻に対応する秒数であると仮定します。この位置の軸の刻みラベル文字列は、以下のようになります:

```
set format x                  # デフォルトでは "12/25/76 \n 23:11"
set format x "%A, %d %b %Y"  # "Saturday, 25 Dec 1976"
set format x "%r %D"          # "11:11:11 pm 12/25/76"
set xtics time format "%B"    # "December"
```

時刻書式の例:

日付書式指定は、秒数での時間の値を、ある特定の日の時計の時刻にエンコードします。よって、時は 0 から 23 まで、分は 0 から 59 までのみを動きますが、それらの負の値は、エポック (1970 年 1 月 1 日) より前の日付に対応します。秒数での時間の値を、時間 0 に対する相対的な時/分/秒の数値として出力させるには、時間書式 %tH %tM %tS を使用します。-3672.50 秒の値は以下のように出力されます。

```
set format x                  # デフォルトでは "12/31/69 \n 22:58"
set format x "%tH:%tM:%tS"    # "-01:01:12"
set format x "%.2tH hours"    # "-1.02 hours"
set format x "%tM:%.2tS"      # "-61:12.50"
```

格子線 (grid)

コマンド **set grid** は格子線を描きます。

書式:

```
set grid {{no}{m}xtics} {{no}{m}ytics} {{no}{m}ztics}
         {{no}{m}x2tics} {{no}{m}y2tics} {{no}{m}rtics}
         {{no}{m}cbtics}
         {polar {<angle>}}
         {layerdefault | front | back}
         {{no}vertical}
         {<line-properties-major> {, <line-properties-minor>}}
unset grid
show grid
```

格子線は任意の軸の任意の大目盛り/小目盛りに対して有効/無効にでき、その大目盛りと小目盛りに対する線種、線幅も指定でき、現在の出力装置がサポートする範囲で、あらかじめ定義したラインスタイルを使用することもできます (以下参照: **set style line (p. 285)**)。

2 次元描画では極座標格子も選択できます。これは、gnuplot が極座標モード (polar) のときの **set grid** のデフォルトの挙動ですが、明示的に **set grid polar <angle> rtics** とすれば、極座標モードであるなしに関わらず実行できます。同心円は r 軸の主目盛/副目盛で交差するように描き、動径は <angle> の角を空けて描き

ます。同心円の周囲の目盛りの刻みは、**set ttics** で制御しますが、これは動径の格子線を新たには生成しません。

set grid が描く前に、必要な目盛りは有効になっていなければなりません。**gnuplot** は、存在しない目盛りに対する格子の描画の命令は単に無視します。しかし、後でその目盛りが有効になればそれに対する格子も描きます。

小格子線に対する線種を何も指定しなければ、大格子線と同じ線種が使われます。デフォルトの極座標の角度は 30 度です。

front を指定すると、格子線はグラフのデータの上に描かれます。**back** が指定された場合は格子線はグラフのデータの下に描かれます。**front** を使えば、密集したデータで格子線が見えなくなることを防ぐことができます。デフォルトでは **layerdefault** で、これは 2D 描画では **back** と同じです。3D 描画のデフォルトは、格子とグラフの枠を 2 つの描画単位に分離し、格子は後ろに、枠は描画データまたは関数の前に書きます。ただし、**hidden3d** モードでは、それがそれ自身の並び換えをしていますので、格子線の順番のオプションは全て無視され、格子線も隠線処理にかけられます。これらのオプションは、実際には格子線だけでなく、**set border** による境界線とその目盛りの刻み (以下参照: **set xtics (p. 307)**) にも影響を及ぼします。

3 次元描画では、x 軸と y 軸の刻み位置に対する格子線は、デフォルトでは z=0 に平行な底面上にしか描きませんが、キーワード **vertical** は、格子線を xz 面と yz 面にも zmin から zmax まで描くようにします。

z の格子線は描画の底面に描かれます。これは描画の周りに部分的な箱が描画されている場合にはいいでしょう。以下参照: **set border (p. 198)**。

隠線処理 (hidden3d)

set hidden3d コマンドは曲面描画 (以下参照: **splot (p. 320)**) で隠線処理を行なうように指示します。その処理の内部アルゴリズムに関する追加機能もこのコマンドで制御できます。

書式:

```
set hidden3d {defaults} |
             { {front|back}
               {{offset <offset>} | {nooffset}}
               {trianglepattern <bitpattern>}
               {{undefined <level>} | {noundefined}}
               {{no}altdiagonal}
               {{no}bentover} }
unset hidden3d
show hidden3d
```

gnuplot の通常の表示とは異なり、隠線処理では与えられた関数、またはデータの格子線を、実際の曲面がその曲面の背後にあって隠されている描画要素は見せないのと同じように処理します。これが機能するためには、その曲面が' 格子状' (以下参照: **splot datafile (p. 320)**) である必要があり、またそれらは **with lines** か **with linespoints** で描かれていなければいけません。

hidden3d が有効なときは、格子線だけでなく、面部分や土台の上の等高線 (以下参照: **set contour (p. 207)**) も隠されます。複数の面を描画している場合は、各曲面は自分自身と他の曲面で隠される部分も持ちます。曲面上への等高線の表示 (**set contour surface**) は機能しません。

グラフ上に曲面が一つもない状態でも、hidden3d は **points**, **labels**, **vectors**, **impulses** の 3 次元の描画スタイルに影響を与えます。**vectors** は、隠されない部分は線分 (矢先なし) として表示されます。グラフ内の各々の描画をこの処理から明示的に除外したいときは、**with** 指定に特別のオプション **nohidden3d** を追加してください。

hidden3d は、pm3d モードで描画された、単色塗りの曲面には影響を与えません。pm3d の曲面に対して同様の効果を純粋に得たいならば、これの代わりに **set pm3d depthorder** を使ってください。複数の pm3d 曲面に通常の **hidden3d** 処理を組み合わせるには、オプション **set hidden3d front** を使用してください。これは、hidden3d 処理の全ての要素を、pm3d 曲面を含む残りの他の描画要素の後に強制的に描画するものです。

関数値は格子孤立線の交点で評価されます。見ることの出来る線分を求めるときは個々の関数値、あるいはデータ点の間はそのアルゴリズムによって線形補間されます。これは、**hidden3d** で描画する場合と **nohidden3d** で描画する場合で関数の見かけが異なることを意味します。なぜならば、後者の場合関数値は各標本点で評価されるからです。この違いに関する議論については、以下参照: **set samples (p. 278)**, **set isosamples (p. 227)**。

曲面の隠される部分を消去するのに使われるアルゴリズムは、このコマンドで制御されるいくつかの追加オプションを持っています。**defaults** を指定すればそれらはすべて、以下で述べるようなデフォルトの値に設定されます。**defaults** が指定されなかった場合には、明示的に指定されたオプションのみが影響を受け、それ以外のものは以前の値が引き継がれます。よって、それらのオプションの値をいちいち修正することなく、単に **set {no}hidden3d** のみで隠線処理をオン/オフできることになります。

最初のオプション **offset** は ' 裏側' の線を描画する線の線種に影響を与えます。通常は曲面の表裏を区別するために、裏側の線種は、表側の線種より一つ大きい番号の線種が使われます。**offset <offset>** によって、その追加する値を、デフォルトの 1 とは異なる増分値に変更できます。**nooffset** オプションは **offset 0** を意味し、これは表裏で同じ線種を使うことになります。

次のオプションは **trianglepattern <bitpattern>** です。<bitpattern> は 0 から 7 までの数字で、ビットパターンと解釈されます。各曲面は三角形に分割されますが、このビットパターンの各ビットはそれらの三角形の各辺の表示を決定します。ビット 0 は格子の水平辺、ビット 1 は格子の垂直辺、ビット 2 は、元々の格子が 2 つの三角形に分割されるときの対角辺です。デフォルトのビットパターンは 3 で、これは全ての水平辺と垂直辺を表示し、対角辺は表示しないことを意味します。対角辺も表示する場合は 7 を指定します。

オプション **undefined <level>** は、定義されていない (欠けているデータまたは未定義の関数値) か、または与えられた x,y,z の範囲を超えているデータ点に適用させるアルゴリズムを指示します。そのような点は、それでも表示されてしまうか、または入力データから取り除かれます。取り除かれてしまう点に接する全ての曲面要素は同様に取り除かれ、よって曲面に穴が生じます。<level> = 3 の場合、これは **noundefined** と同じで、どんな点も捨てられません。これは他の場所であらゆる種類の問題を引き起こし得るので使わないべきです。<level> = 2 では未定義の点は捨てられますが、範囲を超えた点は捨てられません。<level> = 1 では、これがデフォルトですが、範囲を超えた点も捨てられます。

noaltdiagonal を指定すると、**undefined** が有効のとき (すなわち <level> が 3 でない場合) に起こる以下の場合のデフォルトでの取扱いを変更できます。入力曲面の各格子状の部分は一方の対角線によって 2 つの三角形に分割されます。通常はそれらの対角線の全てが格子に対して同じ方向を向いています。もし、ある格子の 4 つの角のうち一つが **undefined** 処理によりとり除かれていて、その角が通常の方向の対角線に乗っている場合は、その両方の三角形が取り除かれてしまいます。しかし、もしデフォルトの設定である **altdiagonal** が有効になっている場合、その格子については他方向の対角線が代わりに選択され、曲面の穴の大きさが最小になるようにします。

bentover オプションは今度は **trianglepattern** とともに起こる別のことを制御します。かなりしわくちゃの曲面では、下の ASCII 文字絵に書いたように、曲面の 1 つの格子が 2 つに分けられた三角形の表と裏の反対側が見えてしまう場合 (すなわち、元の四角形が折り曲げられている ('bent over') 場合) があります:

```
                                                          C----B
  元の 4 角形:      A--B      表示される 4 角形:         |\   |
 ("set view 0,0")    | /|   ("set view 75,75" perhaps)  | \  |
                     |/ |                                |  \ |
                     C--D                                |   \|
                                                         A    D
```

曲面の格子の対角辺が <bitpattern> の 2 bit によって見えるようにはなってはいない場合、上の対角辺 CB はどこにも書かれないことになり、それが結果の表示を理解しにくいものにします。デフォルトで定義される **bentover** オプションは、このような場合それを表示するようにします。もしそうしたくないなら、**nobentover** を選択してください。以下も参照隠線処理のデモ (`hidden.dem`)

および複雑な隠線のデモ (`singulr.dem`).

コマンド履歴 (history)

書式:

```
set history {size <N>} {quiet|numbers} {full|trim} {default}
```

最近の gnuplot コマンド履歴を、デフォルトでは $HOME/.gnuplot_history に保存します。このファイルが見つからず、かつ XDG デスクトップサポートが有効な場合、gnuplot は代わりに $XDG_STATE_HOME/gnuplot_history を使用します。

gnuplot の終了時にヒストリファイルに保存する行数を、history size の値に制限します。**set history size -1** とすると、ヒストリファイルに書き出す行数の制限がなくなります。

デフォルトでは、コマンド **history** は各コマンドの前に行番号を出力します。**history quiet** は、今回の実行に対してのみ番号を省略しますが、**set history quiet** は、今後のすべての **history** の番号を省略します。

オプション **trim** は、現在のコマンドに対する前の同じものを削除することで、コマンド履歴内の重複する行の数を減らします。

デフォルトの設定: **set history size 500 numbers trim**

Imaginary_i

書式:

```
set imaginary_i {"<character string>"}
```

gprintf の書式指定子 %Ci は、複素数値 {A,B} を A + Bi のように出力します (以下参照: **gprintf complex** (**p. 221**))。コマンド **set imaginary_i** "< 文字列 >" は、%Ci の挙動の文字 "i" を指定した文字列で置き換えるよう変更します。その文字列の一般的な慣習は以下の表の通りです。文字列を与えなかった場合は、デフォルトに復帰します。(訳注: 以下の表の U+2148, U+1D456 に対する結果の部分、元々はそれらの UTF-8 の文字が書かれていますが、ここでは全角の i で代用しています。)

文字列	結果	慣習
	A+Bi	デフォルト
"\imath"	$A + B\imath$	LaTeX 出力では必要かも
"{/:Italic i}"	A + B*i*	gnuplot の拡張文字列処理
"\U+2148"	A + B ⅈ	Unicode DOUBLE-STRUCK ITALIC SMALL I (U+2148)
"\U+1D456"	A + B*i*	Unicode MATHEMATICAL ITALIC SMALL I (U+1D456)
"im"	A + Bim	Julia
"j"	A + Bj	Numpy

孤立線サンプル数 (isosamples)

関数を面として描画する場合の孤立線 (格子) の密度はコマンド **set isosamples** で変更できます。

書式:

```
set isosamples <iso_1> {,<iso_2>}
show isosamples
```

各曲面グラフは <iso_1> 個の u-孤立線と <iso_2> 個の v-孤立線を持ちます。<iso_1> のみ指定すれば、<iso_2> は <iso_1> と同じ値に設定されます。デフォルトでは、u, v それぞれ 10 本の標本化が行われます。標本数をもっと多くすればより正確なグラフが作られますが、時間がかかります。これらのパラメータは、データファイルの描画には何も影響を与えません。

孤立線とは、曲面の一つの媒介変数を固定して、もう一つの媒介変数によって描かれる曲線のことです。孤立線は、曲面を表示する単純な方法を与えます。曲面 s(u,v) の媒介変数 u を固定することで u-孤立線 c(v) = s(u0,v) が作られ、媒介変数 v を固定することで v-孤立線 c(u) = s(u,v0) ができます。

関数の曲面グラフが隠線処理なしで描かれている場合、**set samples** は各孤立線上で標本化される点の数を制御します。以下参照: **set samples** (p. 278), **set hidden3d** (p. 224)。等高線描画ルーチンは、関数の点の標本化は各孤立線の交点で行われると仮定しているので、関数の曲面と等高線の解像度を変更するときは、**isosamples** と同じように **samples** を変更するのが望ましいでしょう。

等値曲面 (isosurface)

書式:

```
set isosurface {mixed|triangles}
set isosurface {no}insidecolor <n>
```

コマンド **splot $voxelgrid with isosurface** で描かれる曲面は、デフォルトでは四角形と三角形の混合で構成されます。四角形を使用することで、見た目の複雑さの印象を減らす効果があります。このコマンドには、三角形のみでモザイク型曲面を描画するオプションも用意されています。

デフォルトでは、等値曲面の内部は、個別の色で塗ります。色の選択法は、hidden3d 曲面の場合と同じで、基本となる線種にオフセットの <n> を追加した値を使用します。曲面の内側と外側の両方を同じ色で塗るには、**set isosurface noinsidecolor** を使用してください。

Isotropic

書式:

```
set isotropic
unset isotropic
```

set isotropic は、グラフのサイズとアスペクト比を、x, y, z 軸に沿う単位長さが全く同じになるように合わせます。これは次のコマンドと同値で、これの代わりに使えます: **set size ratio -1; set view equal xyz**。これは、2 次元、3 次元グラフの両方に影響します。

unset isotropic は、2 次元、3 次元グラフの両方でその束縛を開放します。これは以下の古いコマンドと同値ですが、覚えるにはこれの方が簡単でしょう: **set size noratio; set view noequal_axes**。

揺らぎ (jitter)

書式:

```
set jitter {overlap <yposition>} {spread <factor>} {wrap <limit>}
           {swarm|square|vertical}
```

例:

```
set jitter                      # 1 文字幅内の点の jitter
set jitter overlap 1.5          # 1.5 文字幅内の点の jitter
set jitter over 1.5 spread 0.5  # 同上、しかし x の移動は半分幅
```

データの 1 つ、または両方の座標が離散値に制限されている場合、多くの点が完全に互いの真上に乗っかる場合があります。jitter (揺らぎ) は、これら重なる点を、それらの座標にずらし値を与えることでその点を房状に広げます。点が重なっていると見なすための閾値は、文字幅単位、あるいは任意の座標オプションを使って指定できます。以下参照: **coordinates (p. 41)**。jitter は、2 次元グラフの **with points**, **with impulses**, **with boxplot** に影響を与えます。これは、3 次元のボクセルデータの描画にも影響を与えます。

デフォルトの jitter 動作は、点を x 方向のみ移動します。これは、「ビースォームグラフ」(bee swarm plot) と呼ばれる独特のパターンを作ります。オプションのキーワード **square** は、移動する点の x 座標に加えて、**overlap** で指定した距離だけは少なくとも離れた別々の階層にいるように y 座標も揃えます。

jitter を x ではなく y (のみ) に沿わせるには、キーワード **vertical** を使用してください。

文字単位での最大の移動距離は、キーワード **wrap** で制限をかけられます。

重なりの判定基準と、揺らぎの大きさは、いずれもデフォルトでは 1 文字単位であることに注意してください。よってグラフの見た目は、出力形式のフォントサイズ、キャンバスサイズ、拡大率によって変更してしまいます。これを避けるには、重なりの判定基準を y 座標系の単位 (キーワード **first**) で指定し、点のサイズと拡大係数を適切な値に調整してください。以下参照: **coordinates (p. 41)**, **pointsize (p. 274)**。

警告: jitter は、"pointsize variable" と両立しません。

set jitter は、3 次元のボクセルデータでも有用です。ボクセル格子データは、均等に配置された点の規則正しい格子であるため、多くの視方向で、点が重なったり、モアレパターンを生成したりします。それらの副作用は、各格子点からランダムに移動させた場所に描画する記号を置くことで避けることが可能です。

凡例 (key)

コマンド **set key** は、描画領域内の各グラフに対するタイトルとサンプル (線分、点、箱) を持つ凡例 (または表題) を有効にします。凡例の機能は、**set key off** か **unset key** とすることで無効にできます。凡例の個々の項目については、対応する plot コマンドでキーワード **notitle** を使用することで無効にできます。凡例のタイトル文字列は、オプション **set key autotitle** や、個々の **plot** や **splot** コマンド上の **title** キーワードで制御できます。

凡例の配置に影響するオプションの書式については以下参照: **key placement (p. 232)**。
凡例の内容に影響するオプションの書式については以下参照:**key layout (p. 232)**。

書式 (大域的オプション):

```
set key {on|off} {default}
      {font "<face>,<size>"} {{no}enhanced}
      {{no}title "<text>" {<font or other text options>}}
      {{no}autotitle {columnheader}}
      {{no}box {<line properties>}} {{no}opaque {fc <colorspec>}}
      {width <width_increment>} {height <height_increment>}
unset key
```

デフォルトでは、凡例はグラフ領域の内側の右上の角に置きます。追加の **font** 指定は、凡例のすべての要素に対するデフォルトになります。凡例の頂上に、凡例の横幅全体に渡る、凡例全体用のオプションのタイトルを与えるこもできます。このタイトルには、個々の描画タイトルのフォント、色、位置合わせ、拡張文字列処理とは別のものを使うことができます。

凡例内には、plot コマンドの各要素に対して、そのタイトル文字列と、そのグラフの描画スタイルを示す線分か記号か形が 1 行で表現されます。タイトル文字列は、自動的に生成しますが、plot コマンド中に **title "text"** を入れることで明示的に与えることもできます。plot コマンドでキーワード **notitle** を使うとそのグラフの凡例行を生成しませんが、タイトルだけを消したい場合は plot コマンドで **title ""** としてください。

等高線描画では、凡例内に対応する行を追加します (以下参照: **cntrlabel (p. 203)**)。ファイル名や関数名の代わりにキーワード **keyentry** を与えたダミーの plot コマンドを使うことで、凡例内に余分に行を追加することができます。以下参照: **keyentry (p. 230)**。

凡例の回りに枠を、ユーザ指定線属性で描くこともできます (**box {...}**)。**height** と **width** の増分は、文字幅単位で指定し、それはその分だけ凡例の箱のサイズを大きくしたり小さくしたりします。これは、主に凡例のエントリを囲む境界を大きくするのに有用です。

デフォルトでは、凡例は一つのグラフと同時に作られます。すなわち、凡例の記号とタイトルは、それに対応するグラフと同時に描かれます。それは、新しいグラフが時には凡例の上に要素をかぶせて配置しうることを意味します。**set key opaque** は、凡例をすべてのグラフの描画が終った後に生成させます。この場合、凡例の領域を背景色か指定した色で塗りつぶし、その後で凡例の記号とタイトルを書きます。**set key noopaque** でデフォルトに復帰できます。

凡例の文字列は、デフォルトでは拡張文字列モード (**enhanced**) を使用します。これはオプション **noenhanced** で変更でき、凡例全体、あるいは凡例タイトルのみ、あるいはグラフタイトル毎に変更することも可能です。

set key default は、以下のデフォルトの key の設定を復帰します。

```
set key notitle
set key nobox noopaque
```

```
set key fixed right top vertical Right noreverse enhanced autotitle
set key noinvert samplen 4 spacing 1 width 0 height 0
set key maxcolumns 0 maxrows 0
```

3 次元グラフの凡例 (3D key)

3 次元グラフ (splot) の凡例の配置は、デフォルトでは **fixed** オプションを使用します。これは、**inside** による配置に似ていますが、重要な違いが一つあります。3 次元グラフの視点が回転したりスケールすると、それに伴なって描画の境界も変化します。凡例の配置が **inside** の場合は、視点が変更するときにこれらの境界が移動するため、凡例も移動します。**fixed** の場合は、視角やスケールの変更を無視して凡例を配置するので、グラフが回転しても、凡例の位置はキャンバスの一つの場所に固定されたままになります。

なお、2 次元グラフでは、**fixed** オプションは完全に **inside** と同じです。

splot で等高線を書く場合、デフォルトでは異なる線種の個々の等高線レベルに対して、別々の凡例の項目を生成します。これを変更するには、以下参照: **set cntrlabel (p. 203)**。

凡例のサンプル (key examples)

以下はデフォルトの位置に凡例を表示します:

```
set key default
```

以下は、凡例をスクリーン座標での指定位置 (右上) に配置します。

```
set key at screen 0.85, 0.85
```

以下は、凡例をグラフより下 (奥) に置き、垂直方向の幅を最小化します。

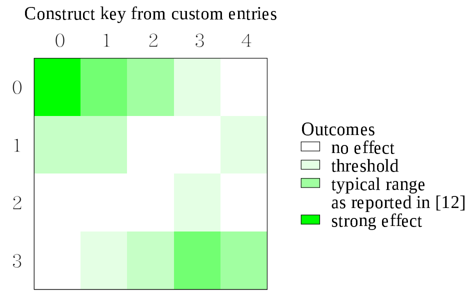
```
set key below horizontal
```

以下は、凡例をグラフ領域の左下隅に置き、文字列は左揃えとし、上に凡例のタイトルを与え、回りを太い境界線の枠で囲みます。

```
set key left bottom Left title 'Legend' box lw 3
```

凡例行の追加 (extra key entries)

通常、各グラフに対して凡例内に 1 行のエントリが自動生成されます。凡例内に見た目をより細かく制御したい場合は、コマンド **plot**, **splot** にキーワード **keyentry** をつけることで、余分に凡例行を追加できます。plot にファイル名や関数名を与える代わりに、**keyentry** をその場所に指定し、スタイル情報 (凡例の記号を生成するのに使用される) とタイトルをその後ろに指定します。通常のタイトルフォント、文字色、**at** 座標、拡張文字列処理に対するオプションは、すべて適用します。例:




```
set key outside right center
plot $HEATMAP matrix with image notitle, \
     keyentry "Outcomes" left, \
     keyentry with boxes fc palette cb 0 title "no effect", \
     keyentry with boxes fc palette cb 1 title "threshold", \
     keyentry with boxes fc palette cb 3 title "typical range", \
     keyentry                            title "as reported in [12]", \
     keyentry with boxes fc palette cb 5 title "strong effect"
```

keyentry "Outcomes" left の行は、通常の凡例を保持するスペースに、文字列を左揃えで配置します。これは、凡例の幅全体に渡ってタイトルを埋め込ませることを可能にします。同じ keyentry に title も与えた場合は、両方の文字列が同じ行に現れ、それにより 2 列の凡例のエントリの生成が可能になります。位置合わせ用のキーワード **left/right/center** や, **boxed** などが利用できます。例:

```
plot ..., keyentry "West Linn" boxed title "locations"
```

凡例の自動タイトル (key autotitle)

set key autotitle は、凡例の各グラフを、plot コマンドで使用したデータファイルや関数の名前によって特定するようにしますが、これがデフォルトの挙動です。**set key noautotitle** は、その自動的なグラフのタイトル付けを無効にします。コマンド **set key autotitle columnheader** は、各入力データの先頭行の各列のエントリをテキスト文字列と解釈し、対応する描画グラフのタイトルとして使用します。描画される量が、複数の列データの関数である場合は、gnuplot はどの列をタイトルの描画に使えばいいのかわかりませんので、その場合、plot コマンド上で、例えば以下のように明示的にタイトルの列を指定する必要があります。

```
plot "datafile" using (($2+$3)/$4) title columnhead(3) with lines
```

注意: **set key autotitle columnheader** とすると、たとえ凡例 (key) が **unset key** で無効になっている場合でも、1 列目をデータとしてではなく、列のヘッダとして処理します。これは、**stats** や **fit** のように凡例を作らないコマンドに対しても同様です。データの先頭行をグラフのタイトルではなく columnheader として使用したい場合は、**set datafile columnheaders** としてください。

また、いずれの場合でも、plot コマンドに明示的な **title** や **notitle** キーワードを指定すれば、それは **set key autotitle** による設定より優先されます。

凡例のレイアウト (key layout)

凡例のレイアウト用オプション:

```
set key {vertical | horizontal}
        {maxcols {<max no. of columns> | auto}}
        {maxrows {<max no. of rows> | auto}}
        {columns <exact no. of columns>}
        {keywidth [screen|graph] <fraction>}
        {Left | Right}
        {{no}reverse} {{no}invert}
        {samplen <sample_length>} {spacing <line_spacing>}
        {width <width_increment>} {height <height_increment>}
        {title {"<text>"} {{no}enhanced} {center | left | right}}
        {font "<face>,<size>"} {textcolor <colorspec>}
```

凡例の要素を自動的に行、または列に並べるやり方は、上のキーワードの影響を受けます。デフォルトは、**vertical** で、これは可能な限り列数を少なくしようとします。各要素は、垂直方向に余裕があるうちは縦に揃えて並べますが、足りなくなれば新しい列に並べます。垂直方向の幅は、'maxrows' で上限を設定できます。**horizontal** の場合は、可能な限り行数を少なくしようとし、水平方向の幅は、'maxcols' で上限を設定できます。

自動で選択される行数、列数には満足できないかもしれません。その場合、**set key columns <N>** で列数を明確に指定できます。この場合、サンプル幅 (**samplen**) と全体の凡例幅 (**keywidth**) も調整する必要があるかもしれません。

デフォルトでは、最初の描画のラベルが凡例の一番上に現われ、それに続くラベルがその下に並んで行きます。オプション **invert** は、最初のラベルを凡例の一番下に置き、それに続くラベルをその上に並べて行きます。このオプションは、凡例のラベルの縦の並びの順番を、積み上げ形式のヒストグラム (**histograms**) の箱の順番に合わせるときに便利でしょう。

set key title "text" は、凡例の上に、全体に渡るタイトルを置きます。そのタイトルのフォント、文字列の行揃え、およびその他の文字属性は、このコマンドの **"text"** の直後に必要なキーワードを置くことで指定できます。他の場所で指定したフォントや文字列の属性は、凡例内のすべての文字列に適用します

デフォルトのレイアウトは、スタイルサンプル (色、線、点、形状等) を凡例の各行の左に置き、タイトル文字列を右に置きます。このサンプルと文字列の位置は、**reverse** キーワードで逆転できます。凡例内のグラフタイトルの行揃えは、**Left**、**Right** (デフォルト) で指示します。スタイルサンプルの水平方向の幅は、ほぼ文字幅単位の数値で設定できます (**samplen**)。

`TeX, LaTeX` 系の出力形式や、整形情報が文字列に埋め込まれる出力形式を使

う場合は、**gnuplot** は必要な幅の見積りがうまくはできませんので、自動的な凡例のレイアウトは、見すぼらしいものになり得ます。凡例を左に置く場合は、**set key left Left reverse** という組合せがいいかもしれませんし、適切な列数や全体の凡例幅を強制的に設定するといいかもしれません。

凡例の配置 (key placement)

凡例の配置用オプション:

```
set key {inside | outside | fixed}
        {lmargin | rmargin | tmargin | bmargin}
        {at <position>}}
        {left | right | center} {top | bottom | center}
        {offset <dx>,<dy>}
```

この節では、自動的に生成される通常の凡例の配置の説明をします。二次的な凡例の構成や、他の場所への描画タイトルの配置については、以下参照: **multiple keys (p. 234)**。

配置の仕組みを理解ための最も重要な概念は、グラフ領域、すなわち内か外かということと、グラフ領域の境界との間の余白 (margin) を考えることです。グラフ領域に沿って、キーワード **left/center/right** (l/c/r) と **top/center/bottom** (t/c/b) は、凡例 (key) をその領域の内側のどこに置くかを制御します。モード **inside** では、凡例はキーワード **left** (l), **right** (r), **top** (t), **bottom** (b), **center** (c) によって以下の図のように描画領域の境界に向かって出力されます:

t/l	t/c	t/r
c/l	c	c/r
b/l	b/c	b/r

モード **outside** でも上と同様に自動的に配置されますが、グラフ領域の境界に対して、というよりもむしろ見た目に対して、というべきでしょう。すなわち、グラフの境界は、グラフ領域の外の凡例の場所を作るために、内側に移動することになります。しかし、これは他のラベルの邪魔をしますし、もしかしたら出力デバイスによってはエラーを引き起こすかもしれません。凡例の出力に合わせてどの描画境界が移動するかは、上に述べた凡例の位置、および重ね上げの方向に依存します。4 方向の中心揃えのオプション (center) に関しては、どの境界が動くのかに関するあいまいさはありませんが、角への出力のオプションについては、重ね上げ方向が **vertical** の場合は左または右の境界が、**horizontal** の場合は上または下の境界が、それぞれ内側に適切に移動します。

余白 (margin) の書き方は、重ね上げの方向にかかわらない自動的な配置を可能にしています。**lmargin** (lm), **rmargin** (rm), **tmargin** (tm), **bmargin** (bm) のうちの一つを、矛盾しない 1 方向のキーワードと組み合わせて使用した場合、凡例の位置は、以下の図に示すようにページの外側に沿って配置されます。キーワード **above** と **over** は **tmargin** と同じ意味で、キーワード **below** と **under** は **bmargin** と同じ意味です。

	l/tm	c/tm	r/tm	
t/lm				t/rm
c/lm				c/rm
b/lm				b/rm
	l/bm	c/bm	r/bm	

以前のバージョンとの互換性のために、**above**, **over**, **below**, **under** に l/c/r や重ね上げ方向のキーワードがないと、**center** と **horizontal** を使用します。キーワード **outside** に t/b/c や重ね上げ方向のキーワードがないと、**top**, **right**, **vertical** (つまり上の t/rm と同じ) を使用します。

凡例の位置 (<position>) は、以前のバージョンと同様単に x,y,z を指定してもいいですが、その最初のサンプル行の座標の座表系を選択するための 5 つのキーワード (**first**, **second**, **graph**, **screen**, **character**) を頭につけることもできます。詳細は、以下参照: **coordinates (p. 41)**。<position> が与えられた場合の **left**, **right**, **top**, **bottom**, **center** の効果は、label コマンドで配置される文字列の場合と同じように基準位置の位置揃えに使用されます。すなわち、**left** は凡例が <position> の右に置かれて左合わせで出力されます。他の場合も同様です。

凡例の位置の微調整 (key offset)

凡例 (key) の配置オプションとは無関係に、凡例の最終的な場所を、位置のずれ (offset) を指定することでも手動で位置合わせせすることができます。いつものように、ずれの x, y 成分は character, graph, screen のいずれの座標でも与えることができます。

凡例のサンプル (key samples)

デフォルトでは、グラフ上の各描画は凡例 (key) 内にそれぞれに対応するエントリを生成します。このエントリには、描画タイトルと、その描画で使われるのと同じ色、同じ塗りつぶし属性による線/点/箱 のサンプルが入ります。font と textcolor 属性は、凡例内に現われる個々の描画タイトルの見た目を制御します。textcolor を "variable" にセットすると、凡例の各エントリの文字列は、描画グラフの線や塗りつぶし色と同じ色になります。これは、以前のある時期の gnuplot のデフォルトの挙動でした。

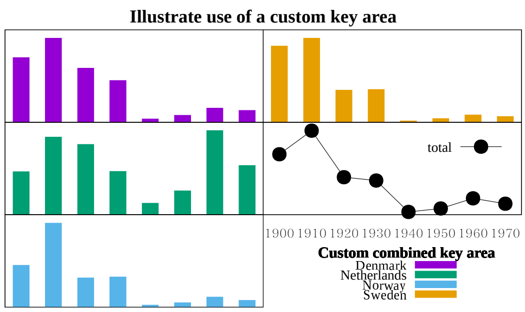
グラフ曲線のサンプルの線分の長さは **samplen** で指定できます。その長さは、目盛りの長さと <sample_length>*(文字幅) の和として計算します。点の記号は、サンプル線分の中央に書かれるため、これは凡例内の点のサンプル位置にも (線分は描かなくても) 影響を与えます。

凡例のベースライン間隔は、現在のフォントサイズに対する「1 行空き」(single space) になっています。これは **set key spacing <line-spacing>** で変更できます。

<width_increment> は、文字列の長さに加えたり減らしたりする幅 (何文字分か) を表す数値です。これは、凡例に外枠を書き、文字列に制御文字を使う場合にだけ有用でしょう。**gnuplot** は外枠の幅を計算するときは、ラベル文字列の文字数を単純に数えるだけなので、それを修正するのに使えます。

複数の凡例の集約 (multiple keys)

各グラフのタイトルを、すべて自動的に生成される凡例 (key) 内に表示させる代わりに、表題や凡例を手動で配置することができます。これにより、例えば多重描画モード (multiplot) での各グラフ要素に対する表題を 1 箇所に集約して作ることができるようになります。




```
set multiplot layout 3,2 columnsfirst
set style data boxes
plot $D using 0:6  lt 1 title at 0.75, 0.20
plot $D using 0:12 lt 2 title at 0.75, 0.17
plot $D using 0:13 lt 3 title at 0.75, 0.14
plot $D using 0:14 lt 4 title at 0.75, 0.11
set label 1 at screen 0.75, screen 0.22 "Custom combined key area"
plot $D using 0:($6+$12+$13+$14) with linespoints title "total"
unset multiplot
```

ラベル (label)

set label コマンドを使うことによって任意の見出し (label) をグラフ中に表示することができます。

書式:

```
set label {<tag>} {"<label text>"} {at <position>}
          {left | center | right}
          {norotate | rotate {by <degrees>}}
          {font "<name>{,<size>}"}
          {noenhanced}
          {front | back}
          {textcolor <colorspec>}
          {point <pointstyle> | nopoint}
          {offset <offset>}
          {nobox} {boxed {bs <boxstyle>}}
          {hypertext}
unset label {<tag>}
show label
```

位置 (<position>) は x,y か x,y,z のどちらかで指定し、座標系を指定するにはその座標の前に **first**, **second**, **polar**, **graph**, **screen**, **character** をつけます。詳細は、以下参照: **coordinates (p. 41)**。

タグ (<tag>) は見出しを識別するための整数値です。タグを指定しなかった場合未使用のもので最も小さい値が自動的に割り当てられます。現在の見出しを変更するときはそのタグと変更したい項目を指定して **set label** コマンドを使います。

<label text> は文字列定数でも構いませんし、文字列変数、または文字列の値を持つ式でも構いません。以下参照: **strings (p. 81)**, **sprintf (p. 50)**, **gprintf (p. 220)**。

デフォルトでは、指定した点 x,y,z に見出しの文章の左端が来るように配置されます。x,y,z を見出しのどこに揃えるかを変更するには変数 <justification> を指定します。これには、**left**, **right**, **center** のいずれかが指定でき、それぞれ文章の左、右、真中が指定した点に来るように配置さるようになります。描画範囲の外にはみ出るような指定も許されますが、座標軸の見出しや他の文字列と重なる場合があります。

箱枠付きのラベルをサポートする出力形式もあります。以下参照: **set style textbox (p. 288)**。回転させた文字列の箱付けは、すべての出力形式が可能なわけではありません。

rotate を指定するとラベルは縦書きになります。**rotate by <degrees>** を指定すると、文字列のベースラインを指定した角に設定します。ただし、文字列の回転をサポートしていない出力形式もあります。

フォントとそのサイズは、出力形式がフォントの設定をサポートしていれば **font "<name>{,<size>}"** で明示的に選択できます。そうでない出力形式では、デフォルトのフォントが使われます。

通常は、現在の出力形式がサポートしていれば、ラベル文字列の全ての文字列に拡張文字列処理モード (enhanced text mode) が使用されます。**noenhanced** を使用することで、特定のラベルを拡張文字列処理から外すことができます。これは、ラベルが例えばアンダースコア (_) を含んでいる場合などに有用です。以下参照: **enhanced text (p. 42)**。

front が与えられると、見出しはデータのグラフの上に書かれます。**back** が与えられると (デフォルト)、見出しはグラフの下に書かれます。**front** を使うことで、密なデータによって見出しが隠されてしまうことを避けることが出来ます。

textcolor <colorspec> は見出し文字列の色を変更します。<colorspec> は線種、rgb 色、またはパレットへの割当のいずれかが指定できます。以下参照: **colorspec (p. 71)**, **palette (p. 51)**。**textcolor** は、**tc** と省略可能です。

```
`tc default` は、文字色をデフォルトにします。
`tc lt <n>` は、文字色を線種 <n> (line type) と同じものにします。
`tc ls <n>` は、文字色を line style <n> と同じものにします。
`tc palette z` は、見出しの z の位置に対応したパレット色になります。
`tc palette cb <val>` は、色見本 (colorbox) の <val> の色になります。
`tc palette fraction <val>` (0<=val<=1) は、[0:1] から `palette` の
    灰色階調/カラーへの写像に対応した色になります。
`tc rgb "#RRGGBB"`, `tc rgb "0xRRGGBB"` は、任意の 24-bit RGB 色を
    設定します。
`tc rgb 0xRRGGBB` も同じです (16 進定数値には引用符は不要)。
```

<pointstyle> がキーワード **lt**, **pt**, **ps** とともに与えられると (以下参照: **style (p. 186)**)、与えられたスタイルと、与えられた線種の色で見出し位置に点 (point) が描画され、見出し文字列は少し移動されます。このオプションは **mouse** 拡張された出力形式でのラベルの配置に、デフォルトで使用されています。見出し文字列近くの点の描画機能を off (これがデフォルト) にするには、**nopoint** を使用してください。

その移動は、デフォルトでは、<pointstyle> が与えられれば **pointsize** の単位で 1,1 で、<pointstyle> が与えられていなければ 0,0 です。移動は、追加の **offset <offset>** でも制御できます。ここで、<offset> は x,y かまたは x,y,z の形式ですが、それに座標系を選択して、その前に **first**, **second**, **graph**, **screen**, **character** のいずれかをつけることもできます。詳細は、以下参照: **coordinates (p. 41)**。

もし一つ (あるいはそれ以上の) 軸が時間軸である場合、座標は **timefmt** の書式にしたがって引用符で囲まれた文字列で与える必要があります。以下参照: **set xdata (p. 302)**, **set timefmt (p. 295)**。

set label に関して有効なオプションは、描画スタイル **labels** でも有効です。以下参照: **labels (p. 115)**。この場合、**textcolor**, **rotate**, **pointsize** の属性の後ろにキーワード **variable** をつけて、それらを固定値でないようにすることが可能です。その場合個々のラベルの対応する属性値は、**using** 指定の追加列により決定します。

Examples

例:

(1,2) の位置に "y=x" と書く場合:

```
set label "y=x" at 1,2
```

Symbol フォントのサイズ 24 の "シグマ" (Σ) をグラフの真中に書く場合:

```
set label "S" at graph 0.5,0.5 center font "Symbol,24"
```

見出し "y=x^2" の右端が (2,3,4) に来るようにし、タグ番号として 3 を使う場合:

```
set label 3 "y=x^2" at 2,3,4 right
```

その見出しを中央揃えにする場合:

```
set label 3 center
```

タグ番号 2 の見出しを削除する場合:

```
unset label 2
```

全ての見出しを削除する場合:

```
unset label
```

全ての見出しをタグ番号順に表示する場合:

```
show label
```

x 軸が時間軸であるグラフに見出しを設定する例:

```
set timefmt "%d/%m/%y,%H:%M"
set label "Harvest" at "25/8/93",1
```

データと、新たに当てはめられたパラメータによる当てはめ関数を描画したい場合、**fit** の後でかつ **plot** の前に以下を実行します:

```
set label sprintf("a = %3.5g",par_a) at 30,15
bfit = gprintf("b = %s*10^%S",par_b)
set label bfit at 30,20
```

当てはめられるパラメータのついた関数の定義式を表示したい場合:

```
f(x)=a+b*x
fit f(x) 'datafile' via a,b
set label GPFUN_f at graph .05,.95
set label sprintf("a = %g", a) at graph .05,.90
set label sprintf("b = %g", b) at graph .05,.85
```

見出し文字列を小さい点から少しだけ移動する場合:

```
set label 'origin' at 0,0 point lt 1 pt 2 ps 3 offset 1,-1
```

pm3d を使った 3 次元のカラー曲面上のある点の位置に、その z の値 (この場合 5.5) に対応した色を見出し文字列につける場合:

```
set label 'text' at 0,0,5.5 tc palette z
```

ハイパーテキスト (hypertext)

出力形式の中には (wxt, qt, svg, canvas, win) グラフ上の特定の位置やキャンバス内のその他の部分にハイパーテキストを貼り付けることができるものがあります。マウスをその場所に持っていくと、文字列を含む箱がポップアップされますが、ハイパーテキストをサポートしない出力形式では、それは何も表示しません。ハイパーテキストを貼り付けるには、そのラベルの **point** 属性を有効にする必要があります。拡張文字列制御書式は、ハイパーテキストラベルには適用されません。例:

```
set label at 0,0 "Plot origin" hypertext point pt 1
plot 'data' using 1:2:0 with labels hypertext point pt 7 \
     title 'mouse over point to see its order in data set'
```

```
# この pm3d 曲面上の任意の場所にマウスを置くとその Z 座標をハイ
# パーテキストとして表示
splot '++' using 1:2:(F($1,$2)) with pm3d, \
      '++' using 1:2:(F($1,$2)):(sprintf("%.3f", F($1,$2))) \
            with labels \
            hypertext point lc rgb "0xff000000" notitle
```

wxt と qt 出力形式では、文字列が表示されたあとにハイパーテキスト部分を左クリックするとそのハイパーテキストがクリップボードにコピーされます。

試験段階の機能 (仕様の細かい部分は変更の可能性あり) - "image{<xsize>,<ysize>}:<filename>{\n<caption text>}" の形式の文字列はポップアップボックス内で画像ファイルを表示させるようにします。サイズ指定によりデフォルトのサイズ 300x200 を変更できます。認識する画像ファイルの型は出力形式によって違いますが、*.png は常に OK です。画像ファイル名の後ろに書いた文字列は、通常のハイパーテキストと同様に表示します。例:

```
set label 7 "image:../figures/Fig7_inset.png\nFigure 7 caption..."
set label 7 at 10,100 hypertext point pt 7
```

線種 (linetype)

コマンド **set linetype** は各種描画に使用される基本的な線種 (linetype) を再定義することを可能にします。このコマンドのオプションは、"set style line" のものと全く同じです。ラインスタイルと違うところは、**set linetype** による再定義は永続的なことで、これは **reset** の影響を受けません。しかし、**reset session** で初期線種の属性を復帰します。

例えば、線種 1 と 2 を以下のように再定義してみます:

```
set linetype 1 lw 2 lc rgb "blue" pointtype 6
set linetype 2 lw 2 lc rgb "forest-green" pointtype 8
```

すると、それらの線種の最初の見た目がどうであったかに関わらず、lt 1 を使用しているすべてのものが、その後は太い青線になります。この性質は、lt 1 によって作られた一時的なラインスタイルの定義のようなものにも適用されます。同様に、線種 2 は、その後は太い緑線になります。

この仕組みは、gnuplot で使用する線種列に対する個人的な好みを設定するのにも使えます。それを行うには、実行時初期化ファイル ~ /.gnuplot に、例えば以下のようなそれ用のコマンド列を追加することをお勧めします:

```
set linetype 1 lc rgb "dark-violet" lw 2 pt 1
set linetype 2 lc rgb "sea-green"   lw 2 pt 7
set linetype 3 lc rgb "cyan"        lw 2 pt 6 pi -1
set linetype 4 lc rgb "dark-red"    lw 2 pt 5 pi -1
set linetype 5 lc rgb "blue"        lw 2 pt 8
set linetype 6 lc rgb "dark-orange" lw 2 pt 3
set linetype 7 lc rgb "black"       lw 2 pt 11
set linetype 8 lc rgb "goldenrod"   lw 2
set linetype cycle 8
```

こうすると、あなたが gnuplot を実行する度に線種はこれらの値に初期化されます。線種はあなたが好む数だけ初期化できます。再定義しない場合は、それはデフォルトの属性を持ち続けます。例えば線種 3 を再定義から外せば、それは青で pt 3, lw 1 となります。

同様のスクリプトファイルで、テーマベースの色選択の定義を行ったり、特定の描画タイプ、あるいは特定の出力形式用に色をカスタマイズしたりすることも可能です。

コマンド **set linetype cycle 8** は、大きな番号の線種に対しては色や線幅に関するこれらの定義を再利用することを gnuplot に伝えます。すなわち、線種 (linetype) 9-16, 17-24 等に対しては、これと同じ色、幅の列を使用します。ただし、点の属性 (pointtype, pointsize, pointinterval) は、このコマンドの影響は受けません。**unset linetype cycle** はこの機能を無効にします。大きな線種番号の線の属性を明示的に定義した場合は、それは小さい番号の線種の属性の再利用よりも優先されます。

第 2 軸との対応 (link)

書式:

```
set link {x2 | y2} {via <expression1> inverse <expression2>}
unset link
```

コマンド **set link** は、x 軸と x2 軸、または y 軸と y2 軸の間の対応を設定します。<expression1> は、第 1 軸の座標を第 2 軸に写像する数式ですが、<expression2> は第 2 軸の座標を第 1 軸に写像する数式です。

例:

```
set link x2
```

これは、このコマンドの最も単純な形式で、x2 軸を範囲 (range) も伸縮 (scale) も方向も x 軸と全く同じにします。**set xrange**, **set x2range** や **set auto x** などのコマンドは、この場合 x 軸にも x2 軸にも作用します。

```
set link x2 via x**2 inverse sqrt(x)
plot "sqrt_data" using 1:2 axes x2y1, "linear_data" using 1:2 axes x1y1
```

このコマンドは、x 軸と x2 軸の、順方向と逆方向の対応を設定しています。順方向の対応は、x2 軸の刻みラベルと、マウスの x2 座標を生成するのに使い、逆方向の対応は、x2 軸系で指定された座標を描画するのに使います。この対応は、非負の x 座標にのみ有効であることに注意してください。y2 軸に対応させた場合、<expression1> と <expression2> には仮変数として y を使う必要があります。

Lmargin

コマンド **set lmargin** は左の余白のサイズをセットします。詳細は、以下参照: **set margin** (p. 242)。

読み込み検索パス (loadpath)

loadpath の設定は、**call**, **load**, **plot**, **splot** コマンドのデータファイル、コマンドファイルの検索パスを追加定義します。ファイルが現在のディレクトリに見つからなかった場合、**loadpath** のディレクトリが検索されます。

書式:

```
set loadpath {"pathlist1" {"pathlist2"...}}
show loadpath
```

パス名は単一のディレクトリ名、または複数のパス名のリストとして入力します。複数のパスからなるパスリストは OS 固有のパス区切り、例えば Unix ではコロン (':'), MS-DOS, Windows, OS/2 ではセミコロン (';') 等で区切ります。**show loadpath**, **save**, **save set** コマンドは、OS 固有のパス区切りをスペース (' ') で置き換えます。

環境変数 GNUPLOT_LIB が設定されている場合、その内容は **loadpath** に追加されますが、**show loadpath** は、**set loadpath** と GNUPLOT_LIB の値を別々に表示しますし、**save**, **save set** コマンドは、GNUPLOT_LIB の値の方は無視します。

ロケール (locale)

locale の設定は **{x,y,z}{d,m}tics** が書く日付の言語を決定します。

書式:

```
set locale {"<locale>"}
```

<locale> にはインストールされたシステムで使うことの出来る任意の言語を指定できます。可能なオプションについてはシステムのドキュメントを参照してください。コマンド **set locale ""** は、環境変数 LC_TIME, LC_ALL, または LANG からロカールの値を決定しようとします。

小数点に関する locale を変更したい場合は、以下参照: **set decimalsign (p. 212)**。文字エンコードを現在のロカールのものに変更したい場合は、以下参照: **set encoding (p. 215)**。

対数軸 (logscale)

書式:

```
set logscale <axes> {<base>}
unset logscale <axes>
show logscale
```

ここで、<axes> (軸) は、**x**, **x2**, **y**, **y2**, **z**, **cb**, **r** の任意の順序の組み合せが可能です。<base> は、対数目盛りの底です (デフォルトの底は 10)。軸を指定しなかった場合は、**r** 以外のすべての軸が対象となります。コマンド **unset logscale** は、すべての軸の対数目盛りを解除します。対数軸に対してつけられる目盛りの刻みは、等間隔ではないことに注意してください。以下参照: **set xtics (p. 307)**。

例:

x, z 両軸について対数目盛りを設定する:

```
set logscale xz
```

y 軸について底 2 とする対数目盛りを設定する:

```
set logscale y 2
```

pm3d plot 用に z と色の軸に対数目盛りを設定する:

```
set logscale zcb
```

z 軸の対数目盛りを解除する:

```
unset logscale z
```

マクロ (macros)

現在のバージョンの gnuplot では、マクロ置換は常に有効です。コマンドライン内の @<stringvariablename> の形式の部分文字列は、文字列変数 <stringvariablename> に含まれるテキスト文字列に置き換えられます。以下参照: **substitution (p. 82)**。

3 次元座標系 (mapping)

データが **splot** に球面座標や円柱座標で与えられた場合、**set mapping** コマンドは **gnuplot** にそれをどのように扱うかを指定するのに使われます。

書式:

```
set mapping {cartesian | spherical | cylindrical}
```

デフォルトではカーテシアン座標 (通常の x,y,z 座標) が使われます。

球面座標では、データは 2 つか 3 つの列 (またはその個数の **using** エントリ) として与えられます。最初の 2 つは、**set angles** で設定された単位での方位角 (theta) と仰角 (phi) (すなわち "経度" と "緯度") とみなされます。半径 r は、もし 3 列目のデータがあればそれが使われ、もしなければ 1 に設定されます。各変数の x,y,z との対応は以下の通りです:

```
x = r * cos(theta) * cos(phi)
y = r * sin(theta) * cos(phi)
z = r * sin(phi)
```

これは、"極座標系" というより、むしろ "地学上の座標系" (緯度、経度) に相当することに注意してください (すなわち、phi は z 軸となす角、というより赤道から計った仰角、になります)。

円柱座標では、データはやはり 2 つか 3 つの列で与えられ、最初の 2 つは theta (**set angle** で指定された単位の) と z と見なされます。半径 r は球面座標の場合と同様、3 列目のデータがあればそれが、なければ 1 と設定されます。各変数の x,y,z との対応は以下の通りです:

```
x = r * cos(theta)
y = r * sin(theta)
z = z
```

mapping の効果は、**splot** コマンド上の **using** 指定で実現することも可能ですが、多くのデータファイルが処理される場合は **mapping** の方が便利でしょう。しかし、**mapping** を使っていても、もしファイルのデータの順番が適切でなかったら結局 **using** が必要になってしまいます。

mapping は **plot** では何もしません。`world.dem: mapping` のデモ。

周囲の余白 (margin)

margin (周囲の余白) とは、描画領域の境界からキャンバスの一番外側までの間隔のことです。この余白の大きさは自動的にとられますが、コマンド **set margin** で変更することもできます。**show margin** は現在の設定を表示します。描画領域の境界から内側の描画要素までの間隔を変更したい場合は以下参照: **set offsets (p. 255)**。

書式:

```
set lmargin {{at screen} <margin>}
set rmargin {{at screen} <margin>}
set tmargin {{at screen} <margin>}
set bmargin {{at screen} <margin>}
set margins <left>, <right>, <bottom>, <top>
show margin
```

<margin> のデフォルトの単位には、適切と思われる、文字の高さと幅が使われます。正の値は余白の絶対的な大きさを定義し、負の値 (または無指定) は **gnuplot** によって自動計算される値を使うことになります。3 次元描画では左の余白 (lmargin) のみが文字の大きさを単位として設定できます。

キーワード **at screen** は、その余白の指定が全体の描画領域に対する割合であることを意味します。これは、多重描画 (multiplot) モードでの 2D, 3D グラフの角を正確に揃えるのに使えます。この配置は現在の **set origin** やや **set size** の値を無視するようになっていて、それは多重描画内のグラフの配置の別の方法として使われることを意図しています。

描画の余白は通常目盛り、目盛りの見出し、軸の見出し、描画のタイトル、日付、そして境界の外にある場合の凡例 (key) のサイズ等を元に計算されます。しかし、目盛りの刻みが境界でなく軸の方についている場合 (例えば **set xtics axis** によって)、目盛りの刻み自身とその見出しは余白の計算には含まれませんし、余白に書かれる他の文字列の位置の計算にも含まれません。これは、軸と境界が非常に近い場合、軸の見出しが他の文字列を上書きする可能性を示唆します。

Micro

デフォルトでは、軸の刻みラベルの生成に使用する科学系出力のための書式指定子 "%c" は、「マイクロ」(10^-6) を示す接頭辞として小文字の u を使います。コマンド **set micro** は、それとは異なる印字用文字 (unicode U+00B5) を使用するよう gnuplot に指示します。その文字を表現するのに使用されるバイト列は、現在の encoding に依存します。以下参照: **format specifiers (p. 220)**, **encoding (p. 215)**。

現在のエンコーディングによるデフォルトが満足いかない場合、欲しい表現を生成する文字列をオプションとして与えることもできます。これは、latex 系の出力形式では有用で、例えば以下のようにできます。

```
set micro "{\textmu}"
```

Minussign

gnuplot はほとんどの書式付き入力は C 言語のライブラリルーチンである sprintf() で処理します。しかし、gnuplot には独自の書式化ルーチン **gprintf()** もあり、それは軸の刻み文字列の生成に使われています。C の

ライブラリルーチンは、-7 のような負の数の表示には常にハイフン文字 (ASCII \055) を使用しますが、むしろその目的では、それとは異なる印字用のマイナス符号文字 (Unicode U+2212) を使い − 7 のように表示したいと多くの人々は思うでしょう (訳注: 原文では '7' の前は Unicode の「マイナス記号」が使われているが、ここでは JIS の全角マイナス記号 245D を用いた)。コマンド

```
set minussign
```

は、gprintf() の数字の出力に、ハイフンの代わりにマイナス符号文字を使用するよう指示します。UTF-8 ロケールではそれは Unicode U+2212 に対応するマルチバイト文字列になり、Window コードページ 1252 ロケールでは、これは 8 ビット文字の ALT+150 ("en dash") になります。コマンド **set minussign** は、軸の刻みのラベルと、gprintf を明示的に呼び出して生成された文字列に影響を与えますが、その他の場合のハイフンを含む文字列には何ら影響を与えません。以下参照: **gprintf (p. 220)**。

LaTeX は、それ自身が負の符号を自前で処理する仕組みを持っているため、このコマンドは、LaTeX 系の出力形式を使用している場合は無視されることに注意してください。postscript 出力形式を使用する場合も、gnuplot の postscript 用前処理ルーチンが ascii のハイフンコード \055 を **minus** という名前の異なる文字に変換するので、このコマンドは必要はありません。

例 (utf8 ロケールを仮定):

```
set minus
A = -5
print "A = ",A                  # ハイフンを含む文字列を出力
print gprintf("A = %g",A)       # U+2212 文字を含む文字列を出力
set label "V = -5"              # ハイフンを含むラベル
set label sprintf("V = %g",-5)  # ハイフンを含むラベル
set label gprintf("V = %g",-5)  # U+2212 を含むラベル
```

白黒モード (monochrome)

書式:

```
set monochrome {linetype N <linetype properties>}
```

コマンド **set monochrome** は、線種群の別の設定法を選択しますが、それは色の違いではなく、点線/破線パターンや線幅の違いによるものです。このコマンドは、gnuplot の以前のバージョンのある出力形式で monochrome オプションとして提供していたものに置き換わるもので、後方互換性のため、それらの出力形式で "mono" オプションを指定すると、暗黙で **set monochrome** を呼び出します。例えば、

```
set terminal pdf mono
```

は、以下と同等です。

```
set terminal pdf
set mono
```

白黒モード (monochrome) の選択は、明示的な RGB 色、パレット色を使用してのカラーの線の描画を妨げるものではありませんが、以下も参照: **set palette gray (p. 262)**。デフォルトでは 6 つの白黒線種が定義されていますが、それらの属性を変更したり、白黒線種を追加することは、フル形式でそのコマンドを使用することでできます。白黒線種にほどこされた変更は、カラーの線種には影響を与えませんし、その逆も同様です。カラー線種に復帰するには、**unset monochrome** か、**set color** としてください。

マウス (mouse)

コマンド **set mouse** は、現在の対話型出力形式に対してマウス機能を有効にします。デフォルトでこれが有効になっています。

マウスモードは 2 種類用意されています。2 次元モードは、**plot** コマンドと **splot** の 2 次元射影 (すなわち、z の回転角が 0, 90, 180, 270, 360 度の **set view**、および **set view map**) で動作します。このモードでは、マウス位置が追跡され、マウスボタンや矢印キーを使って拡大したり視点移動したりできます。グラフに対応する凡例のタイトルや別なウィジェットアイコンなどをクリックすることで、個々のグラフの描画をオン/オフに切り替えることをサポートする出力形式もあります。

splot による 3 次元グラフに対しては、グラフの視方向 (view) と縮尺の変更が、それぞれマウスボタン 1 と 2 (によるドラッグ) で行えます。ボタン 2 の垂直方向のドラッグを shift キーと同時に行うと、z 軸の一番下の位置 (**xyplane**) を上下します。これらのボタンにさらに <ctrl> キーを押すと、座標軸は表示されますが、データの表示は消えます。これは大きなデータに対して有用でしょう。マウスボタン 3 は、z 軸の向き (azimuth) を制御します (以下参照: **set view azimuth (p. 299)**)。

gnuplot バージョン 6.0.5 より、多重描画 (multiplot) のすべてのパネルのマウス座標が読み出せるようになっています。以下参照: **new multiplots (p. 32)**。

書式:

```
set mouse {doubleclick <ms>} {nodoubleclick}
          {{no}zoomcoordinates}
          {zoomfactors <xmultiplier>, <ymultiplier>}
          {noruler | ruler {at x,y}}
          {polardistance{deg|tan} | nopolardistance}
          {format <string>}
          {mouseformat <int> | <string> | function <f(x,y)>}
          {{no}labels {"labeloptions"}}
          {{no}zoomjump} {{no}verbose}
unset mouse
```

オプション **noruler** と **ruler** は、定規 (ruler) 機能を off, on にします。**ruler** には座標を与えて原点を設定することもできます。**ruler** が on の間、ruler の原点からマウスまでのユーザ単位での距離が連続的に表示されます。デフォルトでは、ruler のトグルスイッチは 'r' にキー割り当てされています。

オプション **polardistance** は、マウスカーソルから定規 (ruler) までの距離を極座標でも表示 (距離、および角度または傾き) するかどうかを決定します。これはデフォルトのキー割り当て '5' に対応します。

ボタン 2 の gnuplot の永続的なラベルを定義するには、オプション **labels** を使用します。デフォルトは **nolabels** で、ボタン 2 は単に一時的なラベルをマウス位置に描画します。ラベルは現在の **mouseformat** の設定に従って書かれます。**labeloptions** 文字列は、コマンド **set label** コマンドに渡されます。そのデフォルトは "point pointtype 1" で、これはラベル位置に小さいプラス (+) を描画します。一時的なラベルは、その次の **replot**、またはマウスズーム操作では現れません。永続的なラベルは、ラベルの点の上で Ctrl キーを押してボタン 2 をクリックすることで消すことができます。実際のラベルの位置にどれ位近くでクリックしなければいけないかの閾値も **pointsize** で決定されます。

オプション **verbose** が ON の場合、実行時の報告コマンドが表示されます。このオプションはドライバウィンドウ上で **6** を打つことで ON/OFF がスイッチできます。デフォルトでは **verbose** は OFF になっています。

ドライバウィンドウ上で 'h' を打つと、マウスとキー割り当ての説明を表示します。これは、ユーザ定義のキー割り当て、すなわち **bind** コマンドによる **hotkeys** も表示します。ユーザ定義のキー割り当てはデフォルトのキー割り当てを無効にすることに注意してください。以下も参照: **bind (p. 76)**。

Doubleclick

ダブルクリックの解像度はミリ秒 (ms) 単位で与えます。これは、ボタン 1 用のもので、現在のマウス位置をクリップボード (**clipboard**) にコピーするのに使う出力形式があります。デフォルトの値は 300 ms です。これを 0 ms に設定するとシングルクリックでそのコピーを行うようになります。

Format

コマンド **set mouse format** は、sprintf() に対する書式文字列の指定で、マウスカーソルの [x,y] 座標を描画ウィンドウとクリップボードにどのように表示するかを決定します。デフォルトは "% #g" です。

この設定は、"set mouse mouseformat" を置き換えるものです。

Mouseformat

書式:

```
set mouse mouseformat i
set mouse mouseformat "custom format"
set mouse mouseformat function string_valued_function(x, y)
```

このコマンドは、現在のマウス位置を報告するのに使用する書式を制御します。整数を指定することで、下の表の書式オプションの一つを選択できます。文字列を指定すると、それを書式オプション 7 の sprintf() に対する書式として使用しますが、x, y に対応する 2 つの実数指定子を持つ必要があります。

最後の文字列を返すカスタム関数の指定は「試験段階」です。それは、スクリーン座標からグラフ座標への逆の対応が x, y の両方を組で考慮する必要があるような座標系の読み出しを可能にします。例については、map_projection デモを参照してください。

例:

```
set mouse mouseformat "mouse x,y = %5.2g, %10.3f"
```

この文字列をまたオフにするには、**set mouse mouseformat ""** とします。

以下の書式が利用可能です:

```
0   デフォルト (1 と同じ)
1   軸の座標                          1.23, 2.45
2   グラフ座標 (0 から 1 まで)       /0.00, 1.00/
3   x = timefmt      y = 軸座標       [(‘set timefmt‘ の設定), 2.45]
4   x = 日付         y = 軸座標       [31. 12. 1999, 2.45]
5   x = 時刻         y = 軸座標       [23:59, 2.45]
6   x = 日付/時刻    y = 軸座標       [31. 12. 1999 23:59, 2.45]
```

7 `‘set mouse mouseformat <format-string>‘` による書式
8 `‘set mouse mouseformat function <func>‘` による書式

マウススクロール (scrolling)

マウスホイールは、2 次元、3 次元の両方のグラフで、x 軸、y 軸の範囲を調節します。この調節は、デフォルトでは現在の範囲の 10% の増加ですが、これは、**set mouse zoomfactor <x-multiplier>,<y-multiplier>** で変更できます。

- <wheel-up> は y と y2 軸範囲を現在の範囲をわずかに上スクロール
- <wheel-down> は y と y2 軸範囲を現在の範囲をわずかに下スクロール
- <shift+wheel-up> は左スクロール (x と x2 軸範囲を減少)
- <shift+wheel-down> は右スクロール (x と x2 軸範囲を増加)
- <control+wheel-up> は現在のマウス位置中心にズームイン
- <control+wheel-down> は現在のマウス位置中心にズームアウト
- <shift+control+wheel-up> は x, x2 軸のみをズームイン
- <shift+control+wheel-down> は x, x2 軸のみをズームアウト

Zoom

現在のマウス位置中心のズームイン/ズームアウトの比率は、マウスホイールで制御します (以下参照: **scrolling (p. 246)**)。

2 次元グラフの選択した領域を広げるには、マウスのドラッグで拡大範囲の線引きを行い、そして左マウスボタンを押すことで行います。グラフウィンドウ上でホットキー ’u’ をタイプすることで元のグラフに復元できます。ホットキー ’p’ と ’n’ は、拡大操作の履歴を後方と前方にたどります。

オプション **zoomcoordinates** は、拡大の際に、拡大の枠の端にその座標を書くかどうかを決定し、デフォルトでは ON になっています。

オプション **zoomjump** が ON の場合、ボタン 3 による拡大範囲の選択を開始すると、マウスポインタは自動的に少しだけずれた位置に移動します。これは、ごく小さい (または空でさえある) 拡大範囲を選択してしまうことを避けるのに便利でしょう。デフォルトでは **zoomjump** は OFF です。

Mttics

極座標グラフの周囲の小目盛り刻みの印は **set mttics** で制御されます。以下参照: **set mxtics (p. 249)**。

多重描画モード (multiplot)

コマンド **set multiplot** は **gnuplot** を多重描画モードにします。これは、複数のグラフを同じページや同じスクリーンウィンドウに隣り同士に並べて表示します。

書式:

```
set multiplot
    { title <page title> {font <fontspec>} {enhanced|noenhanced} }
    { layout <rows>,<cols>
      {rowsfirst|columnsfirst} {downwards|upwards}
      {scale <xscale>{,<yscale>}} {offset <xoff>{,<yoff>}}
      {margins <left>,<right>,<bottom>,<top>}
      {spacing <xspacing>{,<yspacing>}}
    }
set multiplot {next|previous}
unset multiplot
```

出力形式 (terminal) によっては、コマンド **unset multiplot** が与えられるまで何の描画も表示されないことがあります。この場合このコマンドによりページ全体の描画が行なわれ、gnuplot は標準の単一描画モードになります。それ以外の出力形式では、各 **plot** コマンドがそれぞれ表示を更新します。

コマンド **clear** は、次の描画に使う長方形領域を消すのに使えます。典型的には、大きいグラフの内部に小さなグラフを挿入するような場合に必要です。

定義済の見出しやベクトルは、各描画において、毎回現在のサイズと原点に従って書かれます (それらが **screen** 座表系で定義されていない場合)。それ以外の全ての **set** で定義されるものも各描画すべてに適用されます。もし 1 度の描画にだけ現われて欲しいものを作りたいなら、それが例えば日付 (timestamp) だとしたら、**set multiplot** と **unset multiplot** で囲まれたブロック内の **plot** (または **splot**, **replot**) 命令の一つを **set time** と **unset time** ではさんでください。

multiplot のタイトルは、個々の描画タイトルがあったとしても、それとは別のもので、ページの上部にそのためのキャンバス全体の幅にわたるスペースが確保されます。

layout が指定されていない場合、あるいはより良い位置決めをしたい場合は、コマンド **set origin** と **set size** 各描画で正しい位置に設定する必要があります。詳細は、以下参照: **set origin (p. 256)**, **set size (p. 278)**。

例:

```
set multiplot
set size 0.4,0.4
set origin 0.1,0.1
plot sin(x)
set size 0.2,0.2
set origin 0.5,0.5
plot cos(x)
unset multiplot
```

これは、cos(x) のグラフを、sin(x) の上に積み重ねて表示します。

set size と **set origin** は全体の描画領域を参照し、それは各描画で利用されます。以下も参照: **set term size (p. 39)**。描画境界を一列に揃えたいならば、**set margin** コマンドで、境界の外の余白サイズを同じサ

イズに揃えることが出来ます。その使用に関しては、以下参照: **set margin** (p. 242)。余白サイズは文字サイズ単位の絶対的な数値単位を使用することに注意してください。よって残ったスペースに描かれるグラフは表示するデバイスの表示サイズに依存します。例えば、プリンタとディスプレイの表示は多分違ったものになるでしょう。

オプション **layout** により、各描画の前にそれぞれ与えていた **set size** や **set origin** コマンドなしに、単純な複数グラフの描画を作成できます。それらの設定は自動的に行なわれ、いつでもその設定を変更できます。**layout** では表示は <rows> 行と <cols> 列の格子に分割され、各格子は、その後に続く対応する名前のオプションによって行 (rowsfirst)、あるいは列 (columnsfirst) が先に埋められて行きます。描画グラフの積み上げは下方向 (**downwards**) に、または上方向 (**upwards**) に伸びるようにできます。デフォルトは **rowsfirst** で **downwards** です。コマンド **set multiplot next** と **set multiplot previous** は、レイアウトオプションを使用している場合のみに関係します。**next** は、格子内の次の位置をスキップし、空白を残します。**prev** は、直前に描画した位置の直前の格子位置に戻ります。

各描画は **scale** で伸縮を、**offset** で位置の平行移動を行なうことができます。scale や offset の y の値が省略された場合は、x の値がそれに使用されます。**unset multiplot** により自動配置機能はオフになり、そして **set size** と **set origin** の値は **set multiplot layout** の前の状態に復帰されます。

例:

```
set size 1,1
set origin 0,0
set multiplot layout 3,2 columnsfirst scale 1.1,0.9
[ ここには 6 つまでの描画コマンド ]
unset multiplot
```

上の例では 6 つの描画が 2 列の中に上から下へ、左から右へと埋められて行きます。各描画は水平サイズが 1.1/2、垂直サイズが 0.9/3 となります。

他にも、そのレイアウト内のすべてのグラフに一様なマージンをオプション **layout margins** と **spacing** で設定することができますが、これは一緒に使う必要があります。**margins** は、格子配置の複数グラフ全体の外側に対するマージンを設定します。

spacing は、隣接する部分グラフ間の隙間を与えますが、**character** か **screen** 単位で指定することもできます。単一の値を指定すると、それは x, y の両方の方向に使用されますが、2 つの異なる値を指定することもできます。

一つの値に単位がなければ、直前のマージン設定のものを使用します。

例:

```
set multiplot layout 2,2 margins 0.1, 0.9, 0.1, 0.9 spacing 0.0
```

この場合、左にあるグラフの左の境界は、スクリーン座標の 0.1 に置かれ、右にあるグラフの右の境界はスクリーン座標 0.9 の場所に置かれる、等となります。各グラフの隙間は 0 に指定しているので、内側の境界線は重なります。

例:

```
set multiplot layout 2,2 margins char 5,1,1,2 spacing screen 0, char 2
```

これは、左のグラフの境界は、キャンバスの左端から 5 文字幅の場所に、右のグラフの右の境界は、キャンバスの端から 1 文字幅の場所にあるようなレイアウトを生成します。下のマージンは 1 文字分の高さ、上のマー

ジンは 2 文字分の高さになります。グラフ間の水平方向の隙間はありませんが、縦方向には 2 文字分の高さに等しい隙間があります。

例:

```
set multiplot layout 2,2 columnsfirst margins 0.1,0.9,0.1,0.9 spacing 0.1
set ylabel 'ylabel'
plot sin(x)
set xlabel 'xlabel'
plot cos(x)
unset ylabel
unset xlabel
plot sin(2*x)
set xlabel 'xlabel'
plot cos(2*x)
unset multiplot
```

以下参照: **remultiplot (p. 190)**, **new multiplots (p. 32)**。また以下も参照multiplot のデモ (multiplt.dem)

Mx2tics

x2 (上) 軸の小目盛り刻みの印は **set mx2tics** で制御されます。以下参照: **set mxtics (p. 249)**。

小目盛り刻み (mxtics)

x 軸の小目盛り刻みの印は **set mxtics** で制御されます。**unset mxtics** によってそれを表示させなくすることが出来ます。同様の制御コマンドが各軸毎に用意されています。

書式:

```
set mxtics <freq>
set mxtics default
set mxtics time <N> <units>
unset mxtics
show mxtics
```

これらの書式は **mytics**, **mztics**, **mx2tics**, **my2tics**, **mrtics**, **mttics**, **mcbtics** に対しても同じです。

<freq> は、大目盛り間の、小目盛りによって分割される区間の数であり、小目盛りの数ではありません。通常の線形軸に対するデフォルトの値は、2 (目盛り 1 つ) か 5 (目盛り 4 つ) で、これは大目盛りの間隔によって決まります。

default を指定すると小目盛りの数はデフォルトの値に戻ります。

set mxtics time <N> <units> は、大目盛りが時刻モードの場合にのみ適用します。以下参照: **set mxtics time (p. 250)**。

軸が対数軸である場合、分割区間の数はデフォルトでは有意な数にセットされます (10 個の長さを元にして)。<freq> が与えられていればそちらが優先されます。しかし、対数軸では通常の小目盛り (例えば 1 から 10 までの 2, 3, ..., 8, 9 の刻み) は、9 つの部分区間しかありませんが、<freq> の設定は 10 とすることでそうなります。

小目盛りを任意の位置に設定するには、("<label>" <pos> <level>, ...) の形式を **set {x|x2|y|y2|z}tics** で使用してください。ただし、<label> は空 ("") で、<level> を 1 にします。

コマンド **set m{x|x2|y|y2|z}tics** は、大目盛りが一様の間隔の場合にのみ働きます。もし全ての大目盛りが **set {x|x2|y|y2|z}tics** によって手動で配置された場合は、この小目盛りのコマンドは無視されます。自動的な大目盛りの配置と手動の小目盛りの配置は、**set {x|x2|y|y2|z}tics** と **set {x|x2|y|y2|z}tics add** とを使うことで共存できます。

例:

```
set xtics 0, 5, 10
set xtics add (7.5)
set mxtics 5
```

この場合、大目盛りは 0,5,7.5,10、小目盛りは 1,2,3,4,6,7,8,9 の場所

```
set logscale y
set ytics format ""
set ytics 1e-6, 10, 1
set ytics add ("1" 1, ".1" 0.1, ".01" 0.01, "10^-3" 0.001, \
               "10^-4" 0.0001)
set mytics 10
```

この場合、大目盛りは指定された書式で、小目盛りは対数的に配置

デフォルトでは小目盛りの表示は、線形軸ではオフで、対数軸ではオンになっています。その設定は、大目盛りに対する **axis|border** と **{no}mirror** の指定を継承します。これらに関する情報については、以下参照: **set xtics** (p. 307)。

Mxtics time

書式:

```
set mxtics time <N> {seconds|minutes|hours|days|weeks|months|years}
```

これは、gnuplot バージョン 6 で導入された新しいコマンドオプションです。これは、小目盛りの刻みを、大目盛り区間に対する分割位置ではなく、時刻単位のある整数番号の場所に正確に配置します。

大目盛りが時刻モード (**set xdata time** か **set xtics time**) の場合は、小目盛りを生成しないのが新しいデフォルトです。

set mxtics や **set mxtics <freq>** で、6 より前のバージョンの挙動に戻せますが、これには常に問題を抱えていました。例えば、72 年間隔に対して自動分割機能は大目盛りを 12 年間隔に、小目盛りを 5 年間隔にしてしまっていました。

しかし **set mxtics time 2 years** を使えば、正確に 1 年置きの年始めのところに小目盛り刻みを置きます。**set mxtics time 1 month** は、各月の日数が等しくないですが、1 月 1 日、2 月 1 日、3 月 1 日、... の各月 1 日の場所に正しく刻みを置きます。

My2tics

y2 (右) 軸の小目盛り刻みの印は **set my2tics** で制御されます。以下参照: **set mxtics (p. 249)**。

Mytics

y 軸の小目盛り刻みの印は **set mytics** で制御されます。以下参照: **set mxtics (p. 249)**。

Mztics

z 軸の小目盛り刻みの印は **set mztics** で制御されます。以下参照: **set mxtics (p. 249)**。

Nonlinear

書式:

```
set nonlinear <axis> via f(axis) inverse g(axis)
unset nonlinear <axis>
```

このコマンドはコマンド **set link** に似ていますが、2 つのリンクされた軸の一方のみを表示する点が違います。隠される軸は線形軸のままです。表示する軸に沿う座標は、g(x) を適用して隠れている軸の座標から割り当てられ、f(x) は表示する軸の座標を隠れている線形軸に対応させます。変換式と逆変換式の両方を指定する必要があります。

これがどのように機能するかを理解するには、x2 軸が対数軸の場合を考えてみてください。

```
set x2ange [1:1000]
set nonlinear x2 via log10(x) inverse 10**x
```

この例は、**set log x2** と同じ効果を生みます。この場合隠れている軸は、[log10(xmin):log10(xmax)] を計算することで [0:3] の範囲になります。

変換関数 f(), g() は、非線形軸毎に適切なダミー変数を使って定義する必要があります。

```
axis: x x2   dummy variable x
axis: y y2   dummy variable y
axis: z cb   dummy variable z
axis: r      dummy variable r
```

例:

```
set xrange [-3:3]
set nonlinear x via norm(x) inverse invnorm(x)
```

この例は確率スケール ("プロビット") の x 軸を作成し、累積正規分布関数 Phi(x) のグラフが線形の y 軸に対して直線となります。

例:

```
logit(p) = log(p/(1-p))
logistic(a) = 1. / (1. + exp(-a))
set xrange [.001 : .999]
set nonlinear y via logit(y) inverse logistic(y)
plot logit(x)
```

この例はロジットスケールの y 軸を作成し、線形の x 軸に対する logit(x) のグラフが直線になります。

例:

```
f(x) = (x <= 100) ? x : (x < 500) ? NaN : x-390
g(x) = (x <= 100) ? x : x+390
set xrange [0:1000] noextend
set nonlinear x via f(x) inverse g(x)
set xtics add (100,500)
plot sample [x=1:100] x, [x=500:1000] x
```

この例は "切断軸" を作成します。x 座標は左に 0 から 100、右に 500 から 1000 が並び、その間に 10 幅の小さな隙間ができます。$100 < x < 500$ の間のデータは描画されず、これは期待通りの動作をします。

図形オブジェクト (object)

コマンド **set object** は、その後のすべてのグラフに現れる単一のオブジェクトを定義します。オブジェクトはいくつでも定義できます。オブジェクトの型は、現在は **rectangle** (長方形)、**circle** (円)、**ellipse** (楕円) をサポートしています。長方形は、コマンド **set style rectangle** によって設定されたスタイルの属性の組 (塗り潰し、色、境界) をデフォルトとして受け継ぎます。個々のオジェクトは、定義時、または後からのコマンドで別々のスタイル属性を与えることが可能です。

2 次元グラフのオブジェクトは、軸座標、グラフ座標 (**graph**)、極座標、スクリーン座標 (**screen**) のいずれの組み合わせでも定義できます。3 次元グラフのオブジェクト指定では、グラフ座標は使えません。3 次元の長方形と楕円は、スクリーン座標だけに制限されています。

書式:

```
set object <index>
    <object-type> <object-properties>
    {front|back|behind|depthorder}
    {clip|noclip}
    {fc|fillcolor <colorspec>} {fs <fillstyle>}
    {default} {lw|linewidth <width>} {dt|dashtype <dashtype>}
unset object <index>
```

<object-type> は、**rectangle**, **ellipse**, **circle**, **polygon** のいずれかです。個々のオブジェクトの型は、その型に特有の性質もいくつか持っています。

オプション **front**, **back**, **behind** は、グラフ自身の描画の前、あるいは後のどちらに描くかを制御します。以下参照: layers (p. 75)。**front** を指定すると、オブジェクトはすべての描画要素の前 (上) に描画されますが、**front** と指定されたラベルよりは後ろ (下) になります。**back** を指定すると、すべての描画要素、すべて

のラベルの後ろに配置されます。**behind** は、軸や **back** の長方形を含むすべてのものの後ろに配置されます。よって、

```
set object rectangle from screen 0,0 to screen 1,1 behind
```

は、グラフやページ全体の背景に色をつけるのに利用できます。

デフォルトでは、オブジェクトは、少なくとも 1 つの頂点がスクリーン座標で与えられていない限り、グラフ境界でクリッピングされます。**noclip** と設定すると、グラフ境界でのクリッピングは無効になりますが、スクリーンサイズに対するクリッピングは行われます。

オブジェクトの塗り潰しの色は <colorspec> で指定します。**fillcolor** は **fc** と省略できます。塗り潰しスタイルは <fillstyle> で指定します。詳細は、以下参照: **colorspec (p. 71)**, **fillstyle (p. 283)**。キーワード **default** を指定すると、これらの属性は描画が実際に行われるときのデフォルトの設定を受け継ぎます。以下参照: **set style rectangle (p. 287)**。

長方形 (rectangle)

書式:

```
set object <index> rectangle
    {from <position> {to|rto} <position> |
     center <position> size <w>,<h> |
     at <position> size <w>,<h>}
```

長方形の位置は、対角に向かい合う 2 つの頂点 (左下と右上) の位置、あるいは中心点の位置と横幅 (<w>) と縦幅 (<h>) で指定できます。いずれの場合も点の位置は、軸の座標 (**first**, **second**)、グラフ領域内の相対座標 (**graph**)、スクリーン座標 (**screen**) のいずれかを使用できます (以下参照: **coordinates (p. 41)**)。オプション **at** と **center** は同じ意味です。

例:

```
# 座標軸で囲まれた領域全体の背景を水色に
set object 1 rect from graph 0, graph 0 to graph 1, graph 1 back
set object 1 rect fc rgb "cyan" fillstyle solid 1.0

# 左下角が 0,0, 右上角が 2,3 の赤い四角を一つ置く
set object 2 rect from 0,0 to 2,3 fc lt 1

# 青い境界の空（塗り潰さない）長方形を置く
set object 3 rect from 0,0 to 2,3 fs empty border rgb "blue"

# 頂点は移動しないまま、塗り潰しと色をデフォルトに変更
set object 2 rect default
```

スクリーン座標で長方形の角を指定すると、それは現在のグラフ領域の端を越えることも可能ですが、その他の場合は長方形はグラフ内に収まるようにクリッピングされます。

楕円 (ellipse)

書式:

```
set object <index> ellipse {at|center} <position> size <w>,<h>
    {angle <orientation>} {units xy|xx|yy}
    {<other-object-properties>}
```

楕円の位置は、中心を指定し、その後ろに幅と高さ (主軸と副軸) を指定します。キーワード **at** と **center** は同じ意味です。中心の位置の指定には、軸の座標 (**first**, **second**)、グラフ領域内の相対座標 (**graph**)、スクリーン座標 (**screen**) のいずれかを使用できます (以下参照: **coordinates (p. 41)**)。主軸と副軸の長さは、軸の座標で与えなければいけません。楕円の向き (orientation) は、水平軸と楕円の主軸との間の角度で指定します。角度を与えなければ、デフォルトの楕円の向きが代わりに使われます (以下参照: **set style ellipse (p. 287)**)。キーワード **units** は、楕円の軸の縮尺の制御に使用します。**units xy** は、主軸は x 軸の単位で、副軸は y 軸の単位で計算しますが、**units xx** は両軸とも x 軸の単位で縮尺し、**units yy** は両軸とも y 軸の単位になります。デフォルトは **xy** ですが、**set style ellipse units** の設定でいつでも変更できます。

注意: x 軸と y 軸の縮尺が等しくない場合 (そして **units xy** の場合)、回転後の主軸と副軸の比は正しくはなりません。

set object ellipse size <**2r**>**,**<**2r**> と **set object circle** <**r**> とは、一般には同じことにはならないことに注意してください。circle の半径は常に x 軸の単位で計られ、よって x 軸と y 軸の縮尺が違ったり、描画のアスペクト比が 1 でなくても、常に円が生成されます。**units** が **xy** に設定されていれば、'set object ellipse' では、最初の <2r> は x 軸の単位で、後ろの <2r> は y 軸の単位で計られますが、これは x 軸と y 軸の縮尺が同じで、かつ描画のアスペクト比が 1 である場合のみ円を生成することを意味します。しかし、**units** を **xx** や **yy** にセットすれば、コマンド **set object** で指定した直径は同じ単位で計算されるので、楕円は正しいアスペクト比を持ち、描画をリサイズしてもそのアスペクト比は保持されます。

円 (circle)

書式:

```
set object <index> circle {at|center} <position> size <radius>
    {arc [<begin>:<end>]} {no{wedge}}
    {<other-object-properties>}
```

円の位置は、中心を指定し、その後ろに半径を指定します。キーワード **at** と **center** は同じ意味です。2 次元グラフでは、位置と半径は任意の座標系で指定できます。以下参照: **coordinates (p. 41)**。3 次元グラフの円にはグラフ座標は使えません。そのどの場合でも、半径は軸、グラフ、スクリーンの水平方向の縮尺に対して計られ、水平方向と垂直方向の縮尺にずれがあっても、結果が常に正しく円になるように直されます。円をグラフの座標で描きたい (つまり水平軸と垂直軸のスケールが違う場合にはそれが楕円として表示されるようにしたい) 場合は、代わりに **set object ellipse** を使ってください。

デフォルトでは、完全な円が描画されます。オプションの **arc** に開始角と終了角を度を単位として指定すると円弧を描画します。円弧は、常に反時計回りに描かれます。

以下も参照: **set style circle (p. 286)**, **set object ellipse (p. 254)**。

多角形 (polygon)

書式:

```
set object <index> polygon
    from <position> to <position> ... {to <position>}
```

または

```
    from <position> rto <position> ... {rto <position>}
```

多角形の位置は、頂点の位置の列を与えることで指定できます。それらには、任意の座標系が使えます。相対的な座標 (rto) を指定する場合は、その座標系は前の頂点と同じ座標系でなければいけません。以下参照: **coordinates (p. 41)**。

例:

```
set object 1 polygon from 0,0 to 1,1 to 2,0
set object 1 fc rgb "cyan" fillstyle solid 1.0 border lt -1
```

Depthorder オプション **set object N depthorder** は、3 次元多角形オブジェクトのみに適用されます。オブジェクトを front/back/behind のレイヤではなく、ソートされた pm3d 四辺形のリストの中に入れ、**set pm3d depthorder** の深さの順に描画します。pm3d 曲面で使う場合、両面の色付けは object fillcolor を linestyle で指定することで生成できます。この場合、多角形の最初の 3 つの頂点の順序が「表裏」を決定します。

3 次元多角形ではないオブジェクトにこの機能を設定すると、それは多分全く描画されません。

グラフ位置の調整 (offsets)

自動縮尺は、x 軸と y 軸の範囲を描画されるデータの座標に合わせます。オフセットは、この範囲を広げる仕組みを提供し、それによりデータと描画範囲の境界の間に隙間を作るようにします。そうすると、自動縮尺機能は、それが **set autoscale noextend** や **set xrange noextend** によって抑えられていない場合は、軸の次の目盛りに達っするまでさらにそれぞれの範囲を拡張します。以下参照: **noextend (p. 197)**。オフセットは、x1, y1 軸の縮尺にのみ影響を与えます。

書式:

```
set offsets <left>, <right>, <top>, <bottom>
unset offsets
show offsets
```

各オフセットは定数、または数式が使え、それらのデフォルトの値は 0 です。デフォルトでは、左右のオフセットは x1 軸と同じ単位で指定し、上下のオフセットは y1 軸と同じ単位で指定しますが、キーワード "graph" を用いることで全グラフサイズに対する割合としてオフセットを指定することもできます。非線形軸 (nonlinear axes) に対しては、"graph" によるオフセットのみ可能です。

正のオフセットの値は、軸の範囲を指定された方向へ伸ばします。例えば正の下方向のオフセットは y の最小値をより小さな値にします。負のオフセット値は、自動縮尺とクリッピングに対して悪く影響します。

例:

```
set autoscale noextend
set offsets graph 0.05, 0, 2, 2
plot sin(x)
```

この sin(x) のグラフの y の範囲は [-3:3] になります。それは、関数の y の範囲は [-1:1] に自動縮尺されますが、垂直方向のオフセットが端にそれぞれ 2 を追加するためです。x の範囲は [-11:10] になりますが、これはデフォルトが [-10:10] で、左に全範囲の 0.05 の割合分広げられるためです。

グラフ位置の指定 (origin)

コマンド **set origin** はスクリーン上で曲面描画の原点を指定 (すなわち、グラフとその余白) するのに使用します。その座標系はスクリーン座標系 (screen) で与えます。この座標系に関する情報については、以下参照: **coordinates (p. 41)**。

書式:

```
set origin <x-origin>,<y-origin>
```

出力先指定 (output)

書式:

```
set output {"<filename>"}
unset output
show output
```

非対話型出力形式では、デフォルトでは生成したグラフは **stdout** に送ります。コマンド **set output** は、その出力を指定したファイルやデバイスにリダイレクトします。このコマンドで開いたファイルは、次の set/unset output コマンド、または次の出力形式の変更、または gnuplot を終了するまで開いたままになっています。

対話型出力形式では、**set output** は無視します。

ファイル名は引用符で囲まなければなりません。ファイル名を省略した場合、

そのコマンドは **unset output** と同じになり、それは直前の **set output** で開いた任意の出力ファイルを閉じ、新しい出力は **stdout** に送ります。

set terminal と **set output** の両方を指定する場合、**set terminal** を先に指定する方が安全です。それは、ある種の terminal では、OS が必要とするフラグをセットすることがあるからです。例えば、バイナリファイルに対して別々の open コマンドを必要とするような OS などがそれに該当します。

パイプをサポートする環境では、パイプ出力も有用です。例えば以下の通りです:

```
set output "|lpr -Plaser filename"
set term png; set output "|display png:-"
```

MS-DOS では、**set output "PRN"** とすると標準のプリンタに出力されます。

Overflow

書式:

```
set overflow {float | NaN | undefined}
unset overflow
```

gnuplot のこのバージョンは、64 ビット整数演算をサポートします。これは、2^53 から 2^63 (おおまかには 10^16 から 10^19) の整数評価の方が、IEEE 754 の浮動小数演算を使用する評価よりも精度の高い値を保持することを意味します。しかし IEEE 浮動小数表現は、精度は犠牲にする代わりに、おおまかには [-10^307 : 10^307] の全範囲をカバーしますが、整数演算は、その結果が [-2^63 : 2^63] の範囲外になる場合はオーバーフローします。そのオーバーフローが起きた場合に何をさせるかはコマンド **set overflow** により制御できます。そのオプションは、以下を参照してください。

set overflow は、**set overflow float** と同じで、結果を整数として返す代わりに実数値として返します。これがデフォルトです。

コマンド **unset overflow** は、整数演算のオーバーフローを無視するようにします。エラーは出ません。32 ビット整数演算しかできない環境で、5.4 より前のバージョンの gnuplot の挙動に近づけたい場合は、これを使用するといいでしょう。

コマンド **reset** は、オーバーフロー処理の状態に影響を与えません。

前のバージョンの gnuplot は、32 ビット演算に制限され、整数オーバーフローは無視していました。しかし、組み込み演算の一部は、整数引数を与えても完全に整数演算を行うとは限らないことに注意してください。これには、指数演算 N**M や、和の演算 (以下参照: **summation (p. 62)**) などが含まれます。これらの演算は、現在は整数引数を与えた場合は整数値を返し、それらを内在的にオーバーフローの影響を受けやすくすることで **set overflow** の状態に支配されるようにしています。

Float

整数演算式が制限範囲 (64 ビット整数では [-2^63 : 2^63]) をオーバーフローした場合、その結果は代わりに浮動小数値として返します。これは、エラーとしては扱われません。例:

```
gnuplot> set overflow float
gnuplot> A = 2**62 - 1;  print A, A+A, A+A+A
4611686018427387903 9223372036854775806 1.38350580552822e+19
```

NaN

整数演算式が制限範囲 (64 ビット整数では [-2^63 : 2^63]) をオーバーフローした場合、その結果は NaN (非数) を返します。これは、エラーとしては扱われません。例:

```
gnuplot> set overflow NaN
gnuplot> print 10**18, 10**19
1000000000000000000 NaN
```

Undefined

整数演算式が制限範囲 (64 ビット整数では [-2^63 : 2^63]) をオーバーフローした場合、その結果は未定義値となります。これは、エラーとして扱われます。例:

```
gnuplot> set overflow undefined
gnuplot> A = 10**19
                ^
         undefined value
```

Affected operations

set overflow の状態は、以下の整数演算

```
+ - * / **
```

と、組み込みの和演算 **sum** に影響します。

これらの演算はすべて、引数がすべて整数ならば、その評価でオーバーフローが起きない限り、整数値の結果を返します。

set overflow は、以下の論理演算、ビット演算には影響しません。

```
<< >>  | ^ &
```

和の評価の過程のどこかでオーバーフローが起きた場合は、**set overflow float** とすると、最終的な和が整数の制限範囲内におさまる場合であっても、実数値を返すようになります。

パレット (palette)

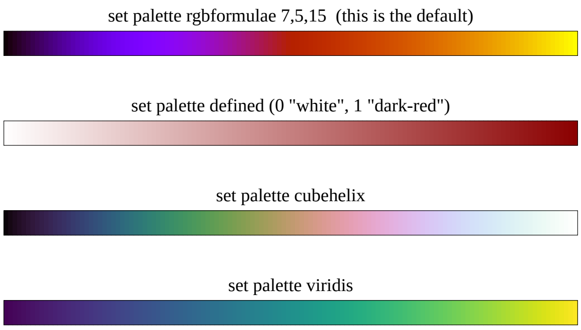
パレットは、色の集合で、通常は一つ以上の段階的なグラデーションの形式で順序づけられ、**pm3d** 曲面や温度分布図 (heatmap)、その他の描画要素を色付けするのに使われます。plot の z 座標か、追加のデータ列の灰色階調値が現在のパレットの色に自動的に写像されます。現在のパレットは、デフォルトでは描画スタイル **pm3d** を使用するグラフの隣に別のカラーボックス (colorbox) として表示されます。カラーボックスは、カスタマイズしたり無効にしたりできます。以下参照: **set colorbox (p. 206)**。以下も参照: **show palette (p. 318), test palette (p. 330)**。

書式:

```
set palette
set palette {
           { gray | color }
           { gamma <gamma> }
           {   rgbformulae <r>,<g>,<b>
             | defined { ( <gray1> <color1> {, <grayN> <colorN>}... ) }
             | file '<filename>' {datafile-modifiers}
             | colormap <colormap-name>
             | functions <R>,<G>,<B>
```

```
            }
            { cubehelix {start <val>} {cycles <val>} {saturation <val>} }
            { viridis }
            { model { RGB | CMY | HSV {start <radians>} } }
            { positive | negative }
            { nops_allcF | ps_allcF }
            { maxcolors <maxcolors> }
          }
```

パレットは、いくつかの方法で定義できます。



- 赤、緑、青の要素に対する公式を、0 から 1 の灰色階調変数の関数として与えること。**set palette rgbformulae** により、あらかじめ定義されている 36 個の公式から選ぶことができます。**set palette functions** により、あなた自身の関数を定義することもできます。
- z の範囲全体を、部分的に 1 つ、または複数の滑らかなグラデーションで覆うよう指定するために **set palette defined** を使うこと。
- 事前に保存したパレットを、現在のパレットに読み込む (load) こと。**set palette file** は、保存したパレットをファイルから読み込みます。**set palette colormap** は、保存したカラーマップから RGB の成分を展開します。
- 必要ならカスタマイズ用の追加パラメータ付きで名前付きパレットを指定。現在提供している名前付きパレットは、**cubehelix** (パターンの族でカスタマイズ可能) と **viridis** です。

オプションのない **set palette** は、デフォルト値に戻します。

set palette negative は、パレットの方向を逆転します。例えば、**set palette viridis negative** は、青から黄色ではなく、黄色から青へのグラデーションを作成します。

set palette gray は、灰色階調のパレットに切り替えます。**set palette color** は最も最近のカラーパレットに戻します。

pm3d カラー曲面では、各微小四辺形の 4 つの角の z 座標の平均値を、範囲 [min_z, max_z] から灰色階調値の範囲 (常に [0:1]) への写像で変換することにより、その四辺形の灰色階調値が得られます。パレットは、その灰色階調値から RGB 色への写像です。

パレット色を、明示的な色指定でも言及できます (以下参照: **colorspec (p. 71)**)。これは、オブジェクトかラベルにパレット色を割り当てるときに便利です。

パレットは、3 種類の色空間 RGB, CMY, HSV のいずれでも定義できます。以下参照: **set palette model (p. 264)**。いずれの色空間でも、その色成分はすべて [0,1] の範囲に制限されています。

Rgbformulae

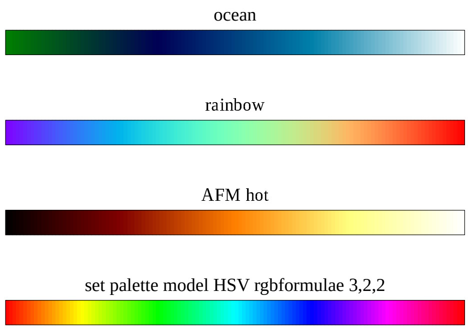
```
set palette rgbformulae <function 1>, <function 2>, <function 3>
```

この名前とは関係なく、このオプションは全ての色空間に適用されます。あなたは、各色成分に対して、あらかじめ登録されている 36 個の割り当て関数のうちの一つを指定しなければいけません。有効な割り当て関数

の一覧は、**show palette rgbformulae** で見ることができます。デフォルトは、**set palette rgbformulae 7,5,15** です。RGB の色空間では、これは 7 番の関数を赤の成分の写像に使用し、5 番の関数を緑成分の写像、15 番の関数を青成分の写像に使用します。関数番号として負の値を使用すると、それは割り当てを逆、すなわち f(gray) でなく、f(1-gray) の写像による成分にします。

RGB の色空間では、いくつかの良い割り当て公式があります:

```
7,5,15   ... デフォルト (黒-青-赤-黄)
3,11,6   ... 緑-赤-紫
23,28,3  ... 海 (緑-青-白)
21,22,23 ... 温度色 (黒-赤-黄-白)
30,31,32 ... 黒-青-紫-黄-白
33,13,10 ... 虹 (青-緑-黄-赤)
34,35,36 ... AFM 温度色 (黒-赤-黄-白)
```



HSV 色空間でのフルカラーパレット:

```
3,2,2    ... 赤-黄-緑-水色-青-紫-赤
```

Defined

灰色から RGB への対応は **palette defined** を使うことで手動で設定できます: グラデーションは RGB の値を与えるために定義され使用されます。グラデーションは、[0,1] の灰色値から [0,1]x[0,1]x[0,1] の RGB 空間への区分的に線形な写像です。その線形補間に使われる灰色値と RGB 値の組を指定する必要があります:

書式:

```
set palette  defined { ( <gray1> <color1> {, <grayN> <colorN>}... ) }
```

ここで N ≧ 2 で、<grayN> は [0,1] に割り当てる灰色値です。それに対応する RGB 色 <colorN> は、3 種類の方法で指定できます:

```
<color> :=  { <r> <g> <b> | '<color-name>' | '#rrggbb' }
```

赤、緑、青に対応する空白で区切られた 3 つの値 (それぞれ [0,1] 内)、引用符でくくられた色名、または引用符でくくられた X 形式の指定方式、のいずれかです。グラデーションの定義では、これらの 3 種の型を自由に組み合わせることができますが、色空間として RGB でないものが選択された場合色名 "red" は少し違ったものになるでしょう。使用できる色名は **show colornames** でその一覧を見ることができます。

<gray> の値は実数の昇順に並べる必要があります。その列の値は自動的に [0,1] に変換されます。

カッコつきのグラデーションの定義なしで **set palette defined** とした場合、RGB 色空間にし、あらかじめ設定されたフルスペクトルグラデーションを使用します。グラデーションを表示するには **show palette gradient** を使用してください。

例:

灰色のパレット (役に立たないが教訓的な) を生成するには:

```
set palette model RGB
set palette defined ( 0 "black", 1 "white" )
```

青-黄-赤のパレット (全てが等価の) を生成するには:

```
set palette defined ( 0 "blue", 1 "yellow", 2 "red" )
set palette defined ( 0 0 0 1, 1 1 1 0, 2 1 0 0 )
set palette defined ( 0 "#0000ff", 1 "#ffff00", 2 "#ff0000" )
```

HSV 色空間でのフルカラースペクトル:

```
set palette model HSV
set palette defined ( 0 0 1 1, 1 1 1 1 )
set palette defined ( 0 0 1 0, 1 0 1 1, 6 0.8333 1 1, 7 0.8333 0 1)
```

赤以外のある色相で巻き進めた HSV フルカラースペクトル

```
set palette model HSV start 0.15
set palette defined ( 0 0 1 1, 1 1 1 1 )
```

等間隔な少しの色だけのパレットを生成するには:

```
set palette model RGB maxcolors 4
set palette defined ( 0 "yellow", 1 "red" )
```

' 交通信号' (滑らかではなく $\mathrm{gray} = 1/3, 2/3$ で跳びを持つ):

```
set palette model RGB
set palette defined (0 "dark-green", 1 "green", \
                     1 "yellow",     2 "dark-yellow", \
                     2 "red",        3 "dark-red" )
```

Functions

```
set palette functions <f1(gray)>, <f2(gray)>, <f3(gray)>
```

このオプションは、**set palette rgbformulae** に似ていますが、各色成分に対して、定義済みの関数番号を指定する代わりに実際の関数を与えることが違います。各関数のダミー変数が必要なら、それは "gray" としなければいけません。関数は、[0,1] の範囲の gray の値を、[0,1] 内の値に写像しなければなりません。

例:

フルカラーパレットを生成するには:

```
set palette model HSV functions gray, 1, 1
```

黒から金色への良いパレット:

```
set palette model RGB functions 1.1*gray**0.25, gray**0.75, 0
```

ガンマ補正の白黒のパレット:

```
gamma = 2.2
map(gray) = gray**(1./gamma)
set palette model RGB functions map(gray), map(gray), map(gray)
```

Gray

set palette gray は、0.0 = 黒から 1.0 = 白への灰色階調 (グレイスケール) パレットに切り替えます。灰色階調パレットから、直前のカラーパレットにまた戻すには、**set palette color** とするのが簡単です。

Cubehelix

オプション "cubehelix" はあるパレット族を定義しますが、これは、灰色階調値が 0 から 1 に増加すると正味の感覚光度が単調に増加する一方、そのパレットの色相 (hue) は標準色相環を回って変化します。

```
D A Green (2011) http://arxiv.org/abs/1108.5083
```

start は、色相環に沿った開始点をラジアン単位で決定します。**cycles** は、パレットの範囲を渡って色相環を何回回るかを決定します。**saturation** (彩度) が大きいと、よりあざやかな色になります。1 より大きい彩度は、個々の RGB 成分をクリッピングすることになり、光度は単調ではなくなってしまいます。**set palette gamma** もパレットに影響を与えます。デフォルト値は以下の通りです。

```
set palette cubehelix start 0.5 cycles -1.5 saturation 1
set palette gamma 1.5
```

Viridis

```
set palette viridis
```

パレット "viridis" は、色覚に障害のあるユーザ向けの青から黄色へのグラデーションです。viridis は、Ste'fan van der Walt と Nathaniel Smith により開発されました。それは、感覚的な明るさ (輝度) の近似的に線形なグラデーションとなります。gnuplot が使用するカラーマップ版は、以下に基づいています。

```
"Viridis - Colorblind-Friendly Color Maps for R", Garnier et al (2021)
https://CRAN.R-project.org/package=viridis
```

Colormap

set palette colormap <name> は、事前に colormap として保存した定義済みのグラデーションを読み込みます。そのカラーマップのアルファチャンネル (透過) 情報 は、それがあったとしても、色の値がパレット定義にコピーされる際に失われます。以下参照: **colormap (p. 201)**。

File

set palette file は基本的に **set palette defined (<gradient>)** と同じで、この <gradient> をデータファイル、またはデータブロックから読み込みます。色の値は、単一の RGB 3 つ組の 24 ビット整数 (**using** 列が 1 つか 2 つの場合) か、または 3 つの別々な R, G, B 成分の実数値 (**using** 列が 3 つか 4 つの場合) のいずれかで与えられます。最初の入力列に明示的な灰色値が与えられない場合は、行番号をそれとして使います。これは、色軸に沿って等間隔なパレットを生成します。

ファイルは通常のデータファイルとして読むので、全てのデータファイル修飾子が使えます。HSV 色空間が選択されている場合には、**R** は実際には **H** を指すことに注意してください。

グラデーションを表示するには **show palette gradient** を使用してください。

例:

RGB のパレットを [0,255] の範囲で読み込む:

```
set palette file 'some-palette' using ($1/255):($2/255):($3/255)
```

等間隔の虹色 (青-緑-黄-赤) パレット:

```
set palette model RGB file "-" using 1:2:3
0 0 1
0 1 0
1 1 0
1 0 0
e
```

明示的な灰色値指定と RGB 値指定で同じことを:

```
set palette model RGB file "-" using 1:2
1  0x0000ff
2  0x00ff00
3  0xffff00
4  0xff0000
e
```

バイナリパレットファイルも同様にサポートされています。以下参照: **binary general (p. 157)**。R,G,B の double のデータの 64 個の 3 つ組をファイル palette.bin に出力し、それを読み込む例:

```
set palette file "palette.bin" binary record=64 using 1:2:3
```

ガンマ補正 (gamma correction)

set palette gamma <gamma> は灰色階調写像 (**set palette gray**) と **cubehelix** のカラーパレット形式に対して自動的なガンマ補正を行います。gamma = 1 は、線形の光度グラデーションを生成します。以下参照: **test palette (p. 330)**。

灰色階調写像に対して <gamma> のデフォルトは 1.5 で、通常は適切な値です。

ガンマ補正は、cubehelix カラーパレット形式には適用されますが、他の色形式には適用されません。しかし、明示的な色関数にガンマ補正を実装するのは難しくありません。

例:

```
set palette model RGB
set palette functions gray**0.64, gray**0.67, gray**0.70
```

補間されたグラデーションを使ってガンマ補正を行うには、適当な色に中間の値を指定します。

```
set palette defined ( 0 0 0 0, 1 1 1 1 )
```

の代わりに例えば以下を指定してください:

```
set palette defined ( 0 0 0 0, 0.5 .73 .73 .73, 1 1 1 1 )
```

または、線形補間が "ガンマ補正" の補間に十分良く適合するまでより良い中間の点を探してください。

最大色数 (maxcolors)

set palette maxcolors <N> はパレットを、連続なパレットの等間隔な区間からサンプリングした N 個の離散的な色に制限します。離散的な N 個の色を不等間隔にしたい場合は、単一の連続パレットの代わりに **set palette defined** を使ってください。

これの第一の使用目的は、離散的な色それぞれが値の範囲を代表するもので、それによって温度分布図を生成することです。

2 つ目の使用目的は、限定された色数 (例えば gif や sixel では 256 色) しかサポートしない出力形式での処理です。gnuplot のデフォルトの線種色はそのうちのいくつかまでを使用するので、パレットが利用できる色数はさらに制限されます。よって、複数のパレットを使用する multiplot では、最初のパレットが利用可能な色位置を使い切ってしまって失敗します。これは、それぞれのパレットで使用できる色数を制限することで、回避できます。

色空間モデル (color model)

```
set palette model { RGB | CMY | HSV {start <radians>} }
```

色空間は、**model** を **RGB**, **HSV**, **CMY** とすることで変更できます。RGB は、標準的な赤 (Red)、緑 (Green)、青 (Blue) に、CMY は水色 (Cyan)、紫 (Magenta)、黄色 (Yellow) に、HSV は 色相 (Hue)、彩度 (Saturation)、明度 (Value) に基づく色空間です。HSV 色空間では、H が 0 から 1 に変化するのに応じて色相環全体を一回りし、よって H=0 と H=1 は同じ色を示すことになります。デフォルトでは、その回転の開始と終了位置は赤です。オプションパラメータ **start** はそれをずらすので、よって **set palette model HSV start 0.3** の後では、H=0 と H=1 はいずれも緑に対応します。

カラーモデルに関するさらなる情報については、以下を参照してください: http://en.wikipedia.org/wiki/Color_space

(訳注: 日本語では "http://ja.wikipedia.org/wiki/色空間" があります。)

パレットオプションに関するマニュアルは、RGB 色空間について書かれていましたが、例えばそれの **R** は「最初の色成分」を意味し、よって実際に使用している色空間によっては **H**、**C** であることに注意してください。

Postscript

このセクションの内容は、**set term postscript color** からの出力のみに関連しています。パレットを **set palette rgbforumulae** で定義している場合、gnuplot は要求した解析的な成分関数の postscript での実装を、pm3d の描画の直前にヘッダとして書きます。/g や /cF の定義を参照してください。通常は、palette が使用する 3 つの公式の定義のみを書き出せばそれでよくて、それがデフォルトのオプション **nops_allcF** です。オプション **ps_allcF** は、その代わりに 36 個の公式全ての定義を書き出します。これにより、一つのグラフ内で曲面毎に違うパレットを使うために postscript ファイルを編集することが可能になります。

postscript ファイルに pm3d 曲面を書き出す場合、その後で gnuplot に付属する awk スクリプト **pm3dCompress.awk** を実行するとそのファイルサイズを小さくできるかもしれません。データが四角形の格子状になっている場合は、awk スクリプト **pm3dConvertToImage.awk** を使うことでより大きな圧縮率が得られる可能性があります。いずれもスクリプトも gnuplot とともに配布されています。使用法:

```
awk -f pm3dCompress.awk thefile.ps >smallerfile.ps
awk -f pm3dConvertToImage.awk thefile.ps >smallerfile.ps
```

媒介変数モード (parametric)

set parametric コマンドは **plot** および **splot** の意味を通常の関数描画から媒介変数表示 (parametric) 関数描画に変更します。**unset parametric** を使えば元の描画モードに戻ります。

書式:

```
set parametric
unset parametric
show parametric
```

2 次元グラフにおいては、媒介変数表示関数はひとつの媒介変数に対する 2 つの関数で定められます。例としては plot sin(t),cos(t) とすることによって円が描けます (アスペクト比が正しく設定されていれば。以下参照: **set size (p. 278)**)。**gnuplot** は、両方の関数が媒介変数による **plot** のために与えられていなければエラーメッセージを出します。

3 次元グラフにおいては面は x = f(u,v), y = g(u,v), z = h(u,v) で定められます。よって 3 つの関数を組で指定する必要があります。例としては、**cos(u)*cos(v),cos(u)*sin(v),sin(u)** とすることによって球面が描けます。**gnuplot** は、3 つ全部の関数が媒介変数による **splot** のために与えられていなければエラーメッセージを出します。

これによって表現できる関数群は、単純な f(x) 型の関数群の内包することになります。なぜならば、2 つ (3 つ) の関数は x, y (, z) の値を独立に計算する記述ができるからです。実際、t,f(t) のグラフは、一番目の関数のような恒等関数を用いて x の値が計算される場合に f(x) によって生成されるグラフと等価です。同様に、3 次元での u,v,f(u,v) の描画は、f(x,y) と等価です。

媒介変数表示関数は、x の関数、y の関数 (、z の関数) の順に指定し、それらは共通の媒介変数およびその変域で定義されることに留意して下さい。

さらに、**set parametric** の指定は、新しい変数変域を使用することを暗に宣言します。通常の f(x) や f(x,y) が xrange、yrange (、zrange) を使用するのに対して、媒介変数モードではそれに加えて、trange, urange, vrange を使用します。これらの変域は **set trange**, **set urange**, **set vrange** によって直接指定することも、**plot** や **splot** で指定することもできます。現時点では、これらの媒介変数のデフォルトの変域は [-5:5] となっています。将来的にはこれらのデフォルト値をもっと有意なものに変更する予定です。

平行描画軸設定 (paxis)

書式:

```
set paxis <axisno> {range <range-options> | tics <tic-options>}
```

```
set paxis <axisno> label <label-options> { offset <radial-offset> }
show paxis <axisno> {range | tics}
```

コマンド **set paxis** は、平行座標描画 (parallel axis) とクモの巣グラフ (spiderplot) の p1, p2, ... 軸の一つに作用すること以外は、**set xrange** や **set xtics** と同じです。以下参照: **parallelaxes (p. 117)**, **set xrange (p. 305)**, **set xtics (p. 307)**。range と tics コマンドへの通常のオプションは、平行座標描画スタイルには意味のないものもありますが、一応すべてを受けつけます。

set paxis <axisno> label <label-options> は、spiderplot 用で、その他では無視されます。平行座標描画の軸は、plot コマンドの **title** オプションでラベル付けできます。これは xtic ラベルを生成するので、**set xtics** も必要となることに注意してください。

軸の線種属性は、**set style parallelaxis** で制御します。

Pixmap

書式:

```
set pixmap <index> {"filename" | colormap <name>}
           at <position>
           {width <w> | height <h> | size <w>,<h>}
           {front|back|behind} {center}
show pixmaps
unset pixmaps
unset pixmap <index>
```

コマンド **set pixmap** は、その後に続く plot で表示されるオブジェクトを定義するコマンド **set object** と似ています。そのピクスマップを構成する R/G/B/alpha の値の長方形配列が png, jpeg, gif ファイルのいずれかから読み込まれます。gnuplot 出力上の位置とピクスマップが占有する範囲は、任意の座標系 (以下参照: **coordinates (p. 41)**) で指定できます。**at <position>** で与えた座標は、キーワード **center** が指定されていない限り、ピクスマップの左下角を意味します。

width <x-extent> を使用して描画するピクスマップの x の範囲を 指定した場合、元の画像のアスペクト比が保持され、軸のスケールや回転ではアスペクト比もピクスマップの向きも変更しません。**height <y-extent>** を使用して y の範囲を指定した場合も同様です。**size <x-extent> <y-extent>** を使用して x と y の範囲の両方を指定した場合、元のアスペクト比を変更します。サイズを指定しなければピクセル単位での元のサイズを使用します (よってその実際のサイズは、出力形式に依存します)。

ピクスマップは、グラフの境界でクリッピングしません。オブジェクトやレイヤの一般的な挙動に対する例外として、**behind** レイヤに割り当てたピクスマップは、multiplot では最初の plot でのみ描画します。これは、一つの背景用のピクスマップを、multiplot のすべてのパネルで共有することを可能にします。

例:

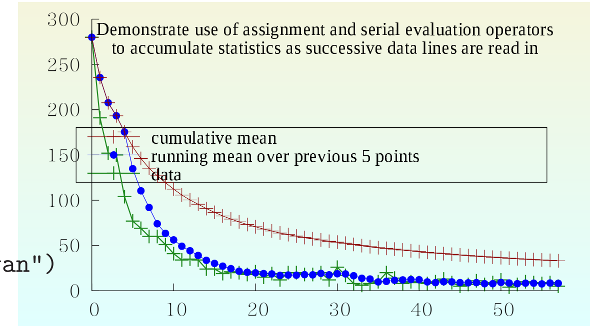
```
# すべてのグラフの背景としてグラデーションを使用
# キャンバス全体を埋めるよう x, y の両方をリサイズする
set pixmap 1 "gradient.png"
set pixmap 1 at screen 0, 0 size screen 1, 1 behind
```

```
# グラフの各ページの右下にロゴをを配置
set pixmap 2 "logo.jpg"
set pixmap 2 at screen 0.95, 0 width screen 0.05 behind

# ある 3 次元座標に小さい画像を配置
# それは描画されている曲面に張り付いているように移動するが、
# 常に前を向き直立し続ける
set pixmap 3 "image.png" at my_x, my_y, f(my_x,my_y) width screen .05
splot f(x,y)
```

カラーマップから作る pixmap (pixmap from colormap)

書式:

```
set pixmap <index> colormap <name>
```



pixmaps の他の利用法は、名前付きパレットで記述したグラデーションを保存することです。これは、長方形領域に対するグラデーション塗りを指定するという簡単な方法です。それは、その名前付きパレットに対する別のカラーボックスや、さらにグラフ全体の背景、あるいはキャンバス全体を塗るのに使うこともできます。

```
set palette defined (0 "beige", 1 "light-cyan")
set colormap new Gradient
set pixmap 1 colormap Gradient behind
set pixmap 1 at screen 0,0 size screen 1,1
plot <something>
```

Pm3d

pm3d は **splot** の一つのスタイルで、パレットに割り付けられた 3 次元、4 次元データを、カラー/灰色の色地図や曲面として描画します。これは、格子状のデータや非格子状のデータを前処理なしに描画できます。pm3d のスタイルオプションは、他の 3 次元描画要素を構築するのに使われる単色多角形にも影響を与えます。

書式 (オプションは任意の順で与えることができます):

```
set pm3d {
           { at <position> }
           { interpolate <steps/points in scan, between scans> }
           { scansautomatic | scansforward | scansbackward
                            | depthorder {base} }
           { flush { begin | center | end } }
           { ftriangles | noftriangles }
           { clip | clip1in | clip4in }
```

```
                { {no}clipcb }
                { corners2color
                  { mean|geomean|harmean|rms|median|min|max|c1|c2|c3|c4 }
                }
                { {no}lighting
                  {primary <fraction>} {specular <fraction>}
                  {spec2 <fraction>}
                }
                { {no}border {retrace} {<linestyle-options>}}
                { implicit | explicit }
                { map }
              }
     show pm3d
     unset pm3d
```

pm3d の曲面は、splot コマンドに与えた順に連続して描画することに注意してください。先に描かれたグラフは、後のグラフで隠される可能性があります。それを避けるために、scan オプションの **depthorder** を使用することができます。

pm3d 曲面は、表示枠の天井 (**top**) や底面 (**bottom**) に射影できます。以下参照: **pm3d position (p. 271)**。以下のコマンドは、異なった高さで 3 つの色付きの曲面を描きます:

```
     set border 4095
     set pm3d at s
     splot 10*x with pm3d at b, x*x-y*y, x*x+y*y with pm3d at t
```

以下も参照: **set palette (p. 258)**, **set cbrange (p. 317)**, **set colorbox (p. 206)**。そしてデモファイル **demo/pm3d.dem** も参考になるでしょう。

With pm3d (明示的な pm3d; pm3d explicit)

書式

```
    splot DATA using (x):(y):(z){:(color)} with pm3d
               {fs|fillstyle <fillstyle>} {fc|fillcolor <colorspec>}
               {zclip [zmin:zmax]}
```

すべての pm3d 曲面の描画属性は、**set pm3d** を使って制御できます。デフォルトでは、曲面全体を四辺形の格子として描画し、各格子は z 座標に割り当てられるパレット色で色付けします。4 番目の入力列を与えると、パレットの割り当ては z の値でなくその値を使用します。以下参照: **pm3d fillcolor (p. 273)**, **pm3d color_assignment (p. 272)**。

set pm3d implicit が有効な状態で **with pm3d** 以外の描画スタイルを使用するのではなく、plot コマンドで明示的に **with pm3d** を使用する場合、描画オプションを追加することが可能です。これにより、同じグラフ上で、違う曲面に別々の色付けの仕組みを使うことが可能になります。

このバージョンの gnuplot は、オプション **zclip** を導入していて、それは z の値の境界で曲面を滑らかな曲面を生成するようにクリッピングします。以下の例は、2 色の 3 次元曲面の頂上の部分を徐々に消していくアニメーションを表示します。

```
set style line 101 lc "gray"
set style line 102 lc "blue"
set pm3d depthorder
do for [i=0:N] {
    splot f(x,y) with pm3d fillcolor ls 101 zclip [* : zmax-(i*delta)]
    pause 0.2  # アニメーションフレーム間隔は 1/5 秒
}
```

暗黙的な pm3d (pm3d implicit)

splot コマンドで明示的に **with pm3d** を指定した場合、またはデータや関数描画スタイル (**style**) が大域的に pm3d にセットされている場合、あるいは、pm3d モードが **set pm3d implicit** となっている場合は、pm3d のカラー曲面を描画します。後の 2 つの場合は、plot コマンドで指定したスタイルで生成する網目に pm3d 曲面を追加する形で描画します。例えば、

```
splot 'fred.dat' with lines, 'lola.dat' with lines
```

は、各データ集合毎に折れ線による網目と pm3d 曲面の両方を描画します。オプション **explicit** (明示的) が ON (または **implicit** が OFF) の場合は、属性 **with pm3d** を指定したグラフのみが pm3d 曲面として描画されます。例えば

```
splot 'fred.dat' with lines, 'lola.dat' with pm3d
```

は、'freq.dat' は折れ線で (線のみで)、'lola.dat' は pm3d 曲面で描画します。

gnuplot の起動時はそのモードは **explicit** (明示的) になっています。歴史的、そして互換性のために、コマンド **set pm3d;** (すなわちオプションがない場合) と **set pm3d at X ...** (すなわち **at** が最初のオプションの場合) はモードを **implicit** (暗黙的) に変更します。コマンド **set pm3d;** は、その他のオプションをそれらのデフォルトの状態に設定します。

デフォルトのデータ/関数の描画スタイルを **pm3d** にしたい場合は、例えば

```
set style data pm3d
```

とします。この場合、オプション **implicit** と **explicit** は効力を持ちません。

Pm3d のアルゴリズム (algorithm)

まず、地図/曲面がどのように描かれるのかについて記述します。入力データは、関数を評価して得られるかまたは **splot data file** から得られます。曲面は、走査 (孤立線) の繰り返しで構成されます。pm3d アルゴリズムでは、最初の走査で検出された隣り合う 2 点と、次の走査で検出された他の 2 点の間の領域が、これら 4 点の z の値 (または追加された 'color' 用の列の値、以下参照: **using** (p. **176**)) に従って灰色で (または カラーで) 塗られます。デフォルトでは 4 つの角の値の平均値が使われますが、それはオプション **corners2color** で変更できます。それなりの曲面を描くためには、隣り合う 2 点の走査が交差してはいけなくて、近接点走査毎の点の数が違いすぎてはいけません。もちろん、最も良いのは走査の点の数が同じことです。他には何も必要ではありません (例えばデータは格子状である必要もない)。他にもこの pm3d アルゴリズムは、入力された (計測された、あるいは計算された) 領域の外には何も描かない、という長所があります。

曲面の色づけは、以下のような入力データに関して行われます:

1. 関数、または 1 つか 3 つのデータ列からなるデータの splot: 上に述べた四角形の 4 つの角の z 座標の平均値 (または **corners2color**) から、灰色の範囲 [0:1] を与える **zrange** または **cbrange** の範囲 [min_color_z,max_color_z] への対応により、灰色/カラーの値が得られます。この値は、直接灰色の色地図用の灰色の値として使うことができます。正規化された灰色の値をカラーに対応させることもできます。完全な説明は、以下参照: **set palette (p. 258)**。

2. 2 つか 4 つのデータ列からなるデータの splot: 灰色/カラーの値は、z の値の代わりに最後の列の座標を使って得られますので、色と z 座標が独立なものになります。これは 4 次元データの描画に使うことができます。

他の注意:

1. 物理学者の間では、gnuplot の文書やソースに現われる 'iso_curve' (孤立線) という言葉よりも、上で言及した ' 走査 (scan)' という言葉の方が使われています。1 度の走査と他の走査の記録により色地図を評価する、というのはそういう意味です。

2. 'gray' や 'color' の値 (scale) は、滑らかに変化するカラーパレットへの、連続な変数の線形写像です。その写像の様子は描画グラフの隣に長方形で表示されます。この文書ではそれを "カラーボックス (colorbox)" と呼び、その変数をカラーボックス軸の変数と呼びます。以下参照: **set colorbox (p. 206)**, **set cbrange (p. 317)**。

光源モデル (lighting)

書式:

```
set pm3d lighting {primary <frac>} {specular <frac>} {spec2 <frac>}
set pm3d spotlight {rgb <color>} {rot_x <angle>} {rot_z <angle>}
                   {Phong <value>} {default}
```

デフォルトでは、pm3d の色の割り当ては、向きや視方位には依存しません。その状態は **set pm3d nolighting** に対応します。一方、コマンド **set pm3d lighting** は、一点の点光源からの照明からの 50% の光による単純な光源モデルを選択します。周囲の明るさに対するその光源の強度は **set pm3d lighting primary <fraction>** で調整できます。反射光 (specular) を含ませる度合いは、その比率 (fraction) の設定ができます:

```
set pm3d lighting primary 0.50 specular 0.0   # ハイライトなし
set pm3d lighting primary 0.50 specular 0.6   # 強いハイライト
```

ベタ塗り (solid color) の pm3d 曲面は、反射光のハイライトがないととても平らに見える傾向があります。

主光源のハイライトは曲面の片側にしか影響を与えないので、他の方向から光る 2 番目のスポットライトの照明を追加するといい場合があります。この 2 番目のスポットライトの強さは、"spec2 <fraction>" (比率) で設定します。2 番目のスポットライトは、spec2 が生の場合にのみ光源モデルに含まれます。その方向、色、反射モデルは、"set pm3d spotlight" で制御します。このスポットライトの利用法と位置取りは、対話形式のデモ **spotlight.dem** で説明しています。hidden_compare.dem も参照してください。(単色塗り曲面の hidden3d と pm3d の処理の比較)

例:

```
set pm3d lighting primary 0.8 spec 0.4 spec2 0.4
set pm3d spot rgb "blue"
```

Pm3d の位置 (position)

pm3d の色付け曲面は、その曲面の本当の z の位置、または底面か天井の平面に射影を描くことができます。これは、オプション **at** に、**b**, **t**, **s** の 6 つまでの組合せの文字列をつけて指定することで制御できます。例えば **at b** は底面のみに描画しますし、**at st** は最初に曲面に描いてそれから天井面に描きますし、**at bstbst** は ... 役には立たないでしょう。

塗られた四角形は、次から次へと描画されて行きます。これは、後に描く四角形が、以前に描いた四角形を覆ったり、重なったりします。最初に走査されるデータを最初に描くか最後に描くかを切り替えるスイッチオプション **scansforward** と **scansbackward** を試してみてください。デフォルトは **scansautomatic** で、これは gnuplot 自身に走査の順を推測させます。オプション **depthorder** は四角形の順序を視点からの距離でソートすることで完全に再構成します。これによりかなり複雑な曲面でも視覚的なものにすることができます。詳細は、以下参照: **pm3d depthorder (p. 271)**。

走査の順番 (scanorder)

```
set pm3d {scansautomatic | scansforward | scansbackward | depthorder}
```

デフォルトでは、pm3d の単色塗り曲面を構成する四角形は、それらが曲面の格子点に沿って出会う順番に塗り潰されます。この順番は、オプション **scansautomatic|scansforward|scansbackward** で制御できます。これらの走査 (scan) オプションは、一般には隠面処理とは両立しません。

2 回の連続する走査で点の数が同じでなかった場合、四角形の点の取り始めを、両方の走査の最初から (**flush begin**) にするか、最後から (**flush end**) にするか、真中から (**flush center**) にするかを決定しなければいけません。**flush (center|end)** は **scansautomatic** とは両立せず、よって **flush center** または **flush end** を指定して **scansautomatic** が設定された場合、それは無言で **scansforward** に変更されます。

2 回の連続する走査で点の数が同じでなかった場合、個々の走査で点が足りない場合に、走査の最後に色三角形を描くかどうかをオプション **ftriangles** は指示します。これは滑らかな色地図の境界を描くのに使われます。

gnuplot は、曲面の単色塗りにおいては、本当の隠面処理は行いませんが、たいていは遠い方から近い方へ順に四角形要素を塗り潰すことで十分なできあがりになります。このモードは、以下のオプションを使うことで選択できます:

```
set pm3d depthorder
```

大域的なオプションである **set hidden3d** は、pm3d 曲面には影響しないことに注意してください。

オプション **depthorder** は、**splot with boxes** で作った細長い長方形に適用すると、良くない結果を生む傾向があります。その場合、キーワード **base** を追加することで、z=0 の平面と箱の共通部分で深さのソートを行うため少しましになります。その形式のグラフは、光源モデル (lighting) を追加すれば更に改善できます。例:

```
set pm3d depthorder base
set pm3d lighting
set boxdepth 0.4
splot $DATA using 1:2:3 with boxes
```

クリッピング (clipping)

書式:

```
set pm3d {clip | clip1in | clip4in}
set pm3d {no}clipcb
```

pm3d 曲面や他の 3 次元オブジェクトを構成する四辺形は、デフォルトでは現在の zrange に関して滑らかにクリッピングします。これは、gnuplot 5.0 以前とは異なる挙動です。2 次元射影 (**set view map**) では、このモードは xrange と yrange に対してもクリッピングします。

それとは別に、4 つの角全部が x, y, z の範囲内である四辺形全体 (**set pm3d clip4in**)、または少なくとも 1 つの角が x, y, z の範囲内である四辺形全体 (**set pm3d clip1in**) を描画することにより、クリッピングすることもできます。オプション **clip**, **clip1in**, **clip4in** は相互に排他的です。

空間座標 x, y, z ベースのクリッピングとは別に、四辺形を描画するかどうかを最終的なパレットカラー値で決定することもできます。**clipcb**: (デフォルト) cbmin 未満のパレットカラー値は cbmin として、cbmax より大きいパレットカラー値は cbmax として扱います。**noclipcb**: cbrange 外のパレットカラー値の四辺形は何も描画しません。

色の割り当て

デフォルトでは、pm3d の色は、その曲面の格子の各 4 辺形毎に個別に割り当てられます。曲面全体に一様な色を割り当てるような他の彩色の仕組みについては、以下参照: **pm3d fillcolor (p. 273)**。

各四辺形には一つの灰色/カラー値 (グラデーションではない) を割り当てます。その値は、**corners2color <option>** に従って四辺形の 4 つの角の z 座標から計算します。そして、その値を現在のパレットから色を選択するのに使用します。以下参照: **set palette (p. 258)**。一つの **splot** コマンド内でパレットを変更することはできません。

4 列目にデータを与えた場合、個々の四角形の彩色は上と同様に行いますが、色の値は z の値とは別とみなされます。別の彩色オプションにより、4 列目のデータに RGB 色を与えることもできます。以下参照: **rgbcolor variable (p. 73)**。この場合、描画コマンドは以下のようにする必要があります:

```
splot ... using 1:2:3:4 with pm3d lc rgb variable
```

z の値の範囲と曲面の色の値の範囲は、**set zrange**, **set cbrange**, **set log z**, **set log cb** 等によって独立に調整し得ることに注意してください。

Corners2color

pm3d 曲面の各四角形の色は、その 4 つの頂点の色の値に基づいて割り当てられます。<option> は 'mean' (デフォルト)、'geomean', 'harmean', 'rms', 'median' で、曲面のカラーの平滑化に幾つかの種類を与え、'min','max' はそれぞれ最小値、最大値を選択します。これらは鋭敏な、あるいは急激なピーク値を持つようなピクセルイメージや色地図を作るときには必要ありません。そのような場合には、むしろオプション 'c1', 'c2', 'c3', 'c4' を使って、四角形の色の割当にただ一つの角の z 座標を使うようにすればいいでしょう。どの角が 'c1' に対応するのかを知るためには何回か実験してみる必要があるでしょう。その向きは描画の方向に依存しています。pm3d アルゴリズムは、カラー曲面を入力データ点の範囲の外には描かないので、オプション 'c<j>' は、格子の 2 つのへりに沿ったピクセルが、どの四角形の色にも寄与しない、という結果をもたらします。例えば、pm3d アルゴリズムを 4x4 のデータ点の格子に適用するスクリプト **demo/pm3d.dem** (是非見てください) では、(4-1)x(4-1)=9 色しかない長方形が生成されます。

Border

```
set pm3d border {retrace} {line-properties}
set pm3d noborder
```

このオプションは、各四角形の境界線を、四角形が描かれているように描画します。追加の線属性 (線種、色、線幅) は任意で、デフォルトでは、幅 1 の黒の実線で境界を書きます。

set pm3d border retrace は、四辺形の塗るのと同じ色で境界を塗らせます。これは、結果として **noborder** と同じことになりますが、出力モードによっては、隣接した塗り潰し四辺形の間のアンチエイリアスによる副産物に苦しむかもしれません。境界を再描画 (retrace) することで、出力ファイルは大きくなってしまいますが、これらの副産物を隠すことができます。

Fillcolor

```
splot FOO with pm3d fillcolor <colorspec>
```

描画スタイル **with pm3d** は、splot コマンド上で fillcolor 追加オプションを受けつけます。その指定は、pm3d 曲面全体に適用されます。以下参照: **colorspec (p. 71)**。たいていの fillcolor 指定では単一色の単色塗りになりますが曲面の表裏の要素を区別するための光源モデルが存在しない場合は見た目を解釈するのが難しいでしょう。以下参照: **pm3d lighting (p. 270)**。

2,3 特別な例を紹介します。**with pm3d fillcolor palette** は、デフォルトの pm3d のパレットベースの配色と全く同じ結果を生成しますので、役に立つオプションではありません。**with pm3d fillcolor linestyle N** は多少意味があります。これは、gnuplot の **hidden3d** モードを使った場合の配色の仕組みと同様、pm3d 曲面の上側と下側に異なる色を割り当てる変種です。線種 N を上側の曲面に、線種 N+1 を下側の曲面に使用します。「上側」と「下側」は、走査 (scan) 順序に依存しますので、**pm3d scansbackward** と **pm3d scansforward** では配色が逆になることに注意してください。この配色オプションは、**pm3d depthorder** とでは最適に機能しますが、それは残念ながら走査順序の制御を許しませんので、代わりに線種 N と N+1 で定義される色を入れ替えないといけないかもしれません。

Interpolate

オプション **interpolate m,n** は、より細かな網目を作るために格子点間を補間します。データ描画に対しては、これは色の曲面を滑らかにし、その曲面の尖りを補正します。関数描画に対しては、この補間はほとんど意味はありませんから、関数描画の場合は普通 **samples** や **isosamples** を使って標本数を増加させるのがいいでしょう。

正の m, n に対しては各四角形、または三角形は、それぞれの方向に m 回、n 回補間されます。負の m, n では補間の頻度は、少なくとも |m|, |n| 点が描画されるように選択されます。これは特別な格子関数と見なすことができます。

注意: **interpolate 0,0** は、自動的に最適な補間曲面点数を選択します。

注意: corners2color で幾何平均 (geomean) のような非線形評価が設定されていたとしても、現在の色の補間は常に線形補間で行われます。

非推奨なオプション

非推奨なオプション **set pm3d map** は、以下と同等です。**set pm3d at b; set view map; set style data pm3d; set style func pm3d;**

非推奨なオプション **set pm3d hidden3d N** は、以下と同等です。**set pm3d border ls N**

Pointinterval の箱サイズ (pointintervalbox)

線属性 **pointinterval**, **pointnumber** は、描画スタイル **linespoints** でのみ使われます。pointinterval や pointnumber の値を負、例えば -N とすると、点の記号を描く前に、各点の記号の後ろの箱 (実際には円) の部分を背景色で塗りつぶすことで消します。コマンド **set pointintervalbox** はその消す領域の大きさ (半径) を制御します。指定する値はデフォルトの半径 (= pointsize) に対する倍率です。**unset pointintervalbox** は、点記号の背景の削除をやめます。

点サイズ (pointsize)

コマンド **set pointsize** は描画で使われる点の大きさを変更します。

書式:

```
set pointsize <multiplier>
show pointsize
```

デフォルトは 1.0 倍です。画像データ出力では、大きいポイントサイズの方が見やすいでしょう。

一つの描画に対するポイントサイズは plot コマンドの上でも変更できます。詳細は、以下参照: **plot with (p. 186)**。

ポイントサイズの設定は、必ずしも全ての出力形式でサポートされているわけではないことに注意してください。

極座標モード (polar)

コマンド **set polar** はグラフの描画方法を xy 直交座標系から極座標系に変更します。

書式:

```
set polar
set polar grid <grid options>
unset polar
show polar
```

極座標モードでは、仮変数 (t) は角度θ を意味します。t のデフォルトの範囲は [0:2*pi] ですが、単位として度が選択されていれば [0:360] となります (以下参照: **set angles (p. 194)**)。

コマンド **unset polar** は描画方法をデフォルトの xy 直交座標系に戻します。

set polar コマンドは 2 次元描画にのみ効力を持ちます。同様の 3 次元機能のコマンドについては、以下参照: **set mapping (p. 241)**。

極座標モードでは t の数式の意味は r=f(t) となり、t は回転角となります。trange は関数の定義域 (角度) を制御し、rrange, xrange, yrange はそれぞれグラフの x,y 方向の範囲を制御することになります。これらの範囲と rrange は自動的に設定されるか、または明示的に設定できます。詳細に関しては、以下参照: **set rrange (p. 277)**, **set xrange (p. 305)**。

例:

```
set polar
plot t*sin(t)
set trange [-2*pi:2*pi]
set rrange [0:3]
plot t*sin(t)
```

最初の **plot** はデフォルトの角度の範囲の 0 から 2*pi を使います。動径方向とグラフのサイズは自動的に伸縮されます。2 番目の **plot** は角度の定義域を拡張し、グラフのサイズを原点から 3 の幅に制限します。これは x,y のそれぞれの方向を [-3:3] に制限することになります。

デフォルトでは極座標グラフは角度 0 (θ =0) が右向きで、増加は反時計回りとなるように向きづけられています。その 0 の向きと増加方向の両方を明示的に変更可能です。以下参照: **set theta (p. 292)**。

set size square とすると **gnuplot** はアスペクト比 (縦横の比) を 1 にするので円が (楕円でなく) 円に見えるようになります。同心円の周囲の目盛りの刻みは、**set ttics** で指定できます。以下も参照極座標のデモ (polar.dem)

および極座標データの描画 (poldat.dem)。

極座標格子 (polar grid)

書式:

```
set polar grid {<theta_segments>, <radial_segments>}
               { qnorm {<power>} | gauss | cauchy | exp | box | hann }
               { kdensity } { scale <scale> }
               {theta [min:max]} {r [min:max]}
```

極座標格子の設定は、描画スタイル **with surface** と組み合わせて極座標の点集合から温度分布図 (heat map) を生成するのに使います。面は、円を埋め尽くす格子からなり、それによりθ と r の離散的な範囲で形作る部分で円を分割します。

各部分には、個々の散在点 [x,y,z] の入力集合からフィルタ操作の適用により導かれる値を割り当てます。デフォルトのフィルタは **qnorm 1** で、これは各点の、格子部分の中心からその点までの距離の逆数で重みづけされた z 値の平均を取ることを意味します。

他のフィルタ操作 gauss, cauchy, exp, box, hann に関しては、ほかの場所で詳しく説明しています。以下参照: **dgrid3d (p. 213)**。

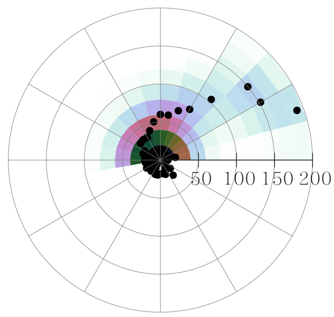
kdensity: このキーワードは、gnuplot に、重みつきの平均の代わりに、すべての点からの寄与の重み付きの和を使用することを指示します。

scale: このスケール因子 (デフォルトは 1.0) は、重み計算で値を使う前に、すべての距離に対してスケーリングを適用します。

マスキング: すべての入力点を、格子値を計算するのに使用します。完全に格子の張られた面は、常にθ の範囲 [0:360] 、および自動縮尺、または前に実行されたコマンド **set rrange** で定義される動径範囲に渡ります。しかし面の実際にグラフに現れる部分は、θ や r の下限、上限によって切り落したくさび型状に制限できます。θ の範囲は度単位で与える必要があります。

例えば、以下のコマンドは、サイズにおいて自動縮尺されたすべての入力点を示すグラフを生成します。すべての入力点の寄与は平均ではなく和であって (**kdensity**)、結果としての格子面のくさび部分のみを表示します。

```
set rrange [0:*]
set polar grid qnorm kdensity theta [0:190]
plot DATA with surface, DATA with points
```



Print コマンドの出力先 (print)

コマンド **set print** は **print** コマンドの出力をリダイレクトします。

書式:

```
set print
set print "-"
set print "<filename>" [append]
set print "|<shell_command>"
set print $datablock [append]
```

パラメータなしの **set print** は、出力を <STDERR> に復帰させます。"-" という <filename> は <STDOUT> を意味します。**append** フラグはファイルを追加 (append) モードで開くことを意味します。パイプをサポートするプラットホーム上では、<filename> が "|" で始まっていたら、<shell_command> へのパイプが開かれます。

コマンド **print** の対象は名前付きデータブロックでも構いません。データブロック名は '$' で始まります。以下参照: **inline data (p. 68)**。データブロックへ文字列を出力する場合、改行文字が入っていると、それは複数のデータブロック行を生成するように展開されます。

PostScript 定義ファイルパス (psdir)

コマンド **set psdir <directory>** は、postscript 出力形式が prologue.ps や文字エンコード用のファイルを探すのに使用する検索パスを制御します。この仕組みは、別にローカルにカスタマイズした prolog ファイル群と切り替えるのに使えます。検索の順番は以下のようになっています。

1) `set psdir` を指定した場合はそのディレクトリ
2) 環境変数 GNUPLOT_PS_DIR で指定したディレクトリ
3) 組み込まれたヘッダー、またはデフォルトのシステムディレクトリ
4) `set loadpath` で指定したディレクトリ

極座標の動径軸 (raxis)

コマンド **set raxis** と **unset raxis** は、動径軸を格子線と x 軸から分離して描画するかどうかを切り替えます。現在の rrange の最小値が 0 でない (そして自動縮尺でない) 場合、グラフと軸が原点に達しないことを示す白丸が極座標グラフの中心に描かれます。軸の線は、グラフの境界と同じ線種で描画されます。以下参照: **polar (p. 274)**, **rrange (p. 277)**, **rtics (p. 277)**, **rlabel (p. 277)**, **set grid (p. 223)**。

Rgbmax

書式:

```
set rgbmax {1.0 | 255}
unset rgbmax
```

rgbimage グラフの RGB (赤、緑、青) 色成分は、デフォルトでは [0:255] の範囲の整数値であるとみなします。**set rgbmax 1.0** とすると、gnuplot は **rgbimage** や **rgbalpha** でのグラフの色成分の生成に使うデータ値を、[0:1] の範囲の実数値であるとみなします。**unset rgbmax** は、それをデフォルトの整数値の範囲 [0:255] に戻します。

Rlabel

このコマンドは、r 軸の上にラベルを配置します。そのラベルは、グラフが極座標モード (polar) であるか否かに関わらず表示されます。追加の指定キーワードについては以下参照: **set xlabel (p. 303)**。

Rmargin

コマンド **set rmargin** は右の余白のサイズをセットします。詳細は、以下参照: **set margin (p. 242)**。

Rrange

コマンド **set rrange** は極座標モードのグラフの動径方向の範囲を設定します。これは xrange と yrange の両方も設定してしまいます。両者は、[-(rmax-rmin) : +(rmax-rmin)] になります。しかし、これの後で xrange や yrange を変更しても (例えば拡大するために)、それは rrange を変更しないので、データ点は rrange に関してクリッピングされたままとなります。他の軸とは違い、r 軸の自動縮尺では常に rmin = 0 となります。**reverse** では自動縮尺フラグは無視されます。注意: rmin を負の値を設定すると、予期せぬ結果を生む可能性があります。

Rtics

コマンド **set rtics** は、動径軸に沿って目盛りを配置します。目盛りとその見出しは原点の右側に描かれます。キーワード **mirror** は、それらを原点の左側にも描きます。その他のキーワードに関する話については以下参照: **polar (p. 274)**, **set xtics (p. 307)**, **set mxtics (p. 249)**。

サンプル数 (samples)

関数のグラフは、与えられた個数の x の値での関数値のサンプリングと、それらの値 f(x0)..f(x1)..f(x2).. を結ぶ線分を描画することで構成しています。関数、またはデータの補間に関するデフォルトのサンプリング数は、コマンド **set samples** で変更できます。**plot** や **splot** コマンドの個々の要素のサンプリング範囲 (sampling range) を変更するには、以下参照: **plot sampling (p. 182)**。

書式:

```
set samples <samples_1> {,<samples_2>}
show samples
```

デフォルトではサンプル数は 100 点と設定されています。この値を増やすとより正確な描画が出来ますが遅くなります。このパラメータはデータファイルの描画には何の影響も与えませんが、補間/近似のオプションが使われている場合はその限りではありません。以下参照: **plot smooth (p. 171)**, **set cntrparam (p. 204)**, **set dgrid3d (p. 213)**。

2 次元のグラフ描画が行なわれるときは <samples_1> の値のみが関係します。

隠線処理なしで曲面描画が行なわれるときは、samples の値は孤立線毎に評価されるサンプル数の指定になります。各 v-孤立線は <samples_1> 個のサンプル点を持ち、u-孤立線は <samples_2> 個のサンプル数を持ちます。<samples_1> のみ指定すると、<samples_2> の値は <samples_1> と同じ値に設定されます。以下も参照: **set isosamples (p. 227)**。

グラフ領域サイズ (size)

書式:

```
set size {{no}square | ratio <r> | noratio} {<xscale>,<yscale>}
show size
```

<xscale> と <yscale> は描画全体の拡大の倍率で、描画全体とはグラフとラベルと余白の部分を含みます。

歴史的な注意: gnuplot の以前の版では、**set size** の値を、出力する描画領域 (キャンバス) のサイズを制御するのにも使っていた出力形式もありました。現在は、**'set size'** と **'set term ... size'** の 2 つは、はっきり違う属性を設定します。

set term <terminal_type> size <x 単位 >, <y 単位 > は、出力ファイルのサイズ、または "キャンバス" のサイズを制御します。サイズパラメータの有効な単位については、個々の出力形式のヘルプを参照してください。デフォルトでは、グラフはそのキャンバス全体に描画されます。

set size <xscale>, <yscale> は、描画自体をキャンバスのサイズに対して相対的に伸縮させます。1.0 より小さい伸縮値を指定すると、グラフはキャンの一部分のみを埋めます。これは、**multiplot** モードで小さいグラフを大きいグラフの中に挿入したり、小さいグラフをいくつか並べたりするのに使えます。1.0 より大きい伸縮値はサポートしておらず、エラーを引き起こす可能性があります。

ratio は、指定した <xscale>, <yscale> の描画範囲内で、グラフのアスペクト比 (縦横比) を <r> にします (<r> は x 方向の長さに対する y 方向の長さの比)。

<r> の値を負にするとその意味は違って来ます。<r>=-1 のとき、x 軸、y 軸の双方の単位 (つまり 1) の目盛りの長さが同一、すなわち等長 (isotropic) になるよう設定します。以下も参照: **set isotropic (p. 228)**。

これは、3 次元用のコマンド **set view equal xy** の 2 次元での同等物です。<r>=-2 のとき、y 軸の単位目盛りの長さは x 軸の単位目盛りの長さの 2 倍に設定されます。<r> が負の値に関して以下同様です。以下も参照: **set isotropic (p. 228)**。

gnuplot が指定されたアスペクト比のグラフをちゃんと書けるかは選択される出力形式に依存します。グラフの領域は出力の指定された部分にちゃんと収まり、アスペクト比が <r> であるような最大の長方形となります (もちろん適当な余白も残しますが)。

set size square は **set size ratio 1** と同じ意味です。

noratio と **nosquare** はいずれもグラフをその出力形式 (terminal) でのデフォルトのアスペクト比に戻しますが、<xscale> と <yscale> はそのデフォルトの値 (1.0) には戻しません。

ratio と **square** は 3 次元描画では意味を持ちませんが、**set view map** を使用した 3 次元描画の 2 次元射影には影響を与えます。 以下も参照: **set view equal (p. 299)**。これは、3 次元の x 軸と y 軸を強制的に同じスケールにします。

例:

グラフが現在のキャンバスを埋めるような大きさに設定します:

```
set size 1,1
```

グラフを通常の半分の大きさで正方形にします:

```
set size square 0.5,0.5
```

グラフの高さを横幅の 2 倍にします:

```
set size ratio 2
```

クモの巣グラフ (spiderplot)

コマンド **set spiderplot** は、座標の解釈を極座標に切り替え、各データ点は、動径軸に沿った位置に割り当てられます。paxis 1 は通常鉛直向きで、2 から N までの軸は、時計回りに等間隔に配置されます。このコマンドは、描画の前に発行しなければなりません。これは、グラフに以下と同様の効果も追加します。

```
set style data spiderplot
unset border
unset tics
set key noautotitle
set size ratio 1.0
```

描画語にこれらを元の状態に復帰するには、**reset** を使用してください。

描画スタイル設定 (style)

デフォルトの描画スタイルは、**set style data** と **set style function** で設定できます。関数やデータのデフォルトの描画スタイルを個々に変更する方法については、以下参照: **plot with (p. 186)**。スタイルの一覧全体は、以下参照: **plotting styles (p. 90)**, **plot with (p. 186)**。

書式:

```
set style function <style>
set style data <style>
show style function
show style data
```

指定できる描画要素のデフォルトスタイルも設定できます。

書式:

```
set style arrow <n> <arrowstyle>
set style boxplot <boxplot style options>
set style circle radius <size> {clip|noclip}
set style ellipse size <size> units {xy|xx|yy} {clip|noclip}
set style fill <fillstyle>
set style histogram <histogram style options>
set style line <n> <linestyle>
set style rectangle <object options> <linestyle> <fillstyle>
set style textbox {<n>} {opaque|transparent} {{no}border} {fillcolor}
set style watchpoint labels <label options>
```

矢印スタイル設定 (set style arrow)

矢印 (arrow) のスタイルの集合は **set style arrow** を使って定義することができます。各スタイルは、それ自身の幅、点種、色などを持ち、そのためそれらを後で使うときにいちいち同じ情報を繰り返して指定しなくても、それを番号 <index> で参照できます。

書式:

```
set style arrow <index> default
set style arrow <index> {nohead | head | backhead | heads}
                        {size <length>,<angle>{,<backangle>} {fixed}}
                        {filled | empty | nofilled | noborder}
                        {front | back}
                        { {linestyle | ls <line_style>}
                          | {linetype | lt <line_type>}
                            {linewidth | lw <line_width}
                            {linecolor | lc <colorspec>}
                            {dashtype | dt <dashtype>} }
unset style arrow
show style arrow
```

<index> は整数で、それで矢のスタイル (arrowstyle) を特定します。

default を指定すると、全ての arrow スタイルパラメータはそのデフォルトの値になります。

<index> の arrowstyle が既に存在する場合、他の全ては保存されたまま、与えられたパラメータのみが変更されます。<index> が存在しなければ、指定されなかった値はデフォルトの値になります。

コマンド **plot** または **splot** から呼び出した arrow スタイルには、データ毎可変な線色 (**lc variable** や **lc rgb variable**) を入れることもでき、それはそれに対応する **using** 指定によるデータの追加列が必要となります。この場合、**set arrow** で作成する個別の arrow に対しては、そのスタイルは多分有益なものではありません。

nohead を指定することで、矢先のない矢、すなわち線分を書くこともできます。これは描画の上に線分を描く別な方法を与えます。デフォルトでは 1 つの矢先がついています。**heads** の指定で線分の両端に矢先が描かれます。

矢先の大きさは **size <length>,<angle>** または **size <length>,<angle>,<backangle>** で変更できます。**<length>** は矢先の各枝の長さで、**<angle>** は矢先の枝と矢軸がなす角度 (単位は度) です。**<length>** の単位は x 軸と同じですが、それは **<length>** の前に **first**, **second**, **graph**, **screen**, **character** をつけることで変更できます。詳細は、以下参照: **coordinates (p. 41)**。

デフォルトでは、とても短い矢の矢先は小さくしますが、これは、**size** コマンドの後ろに **fixed** を使うことで無効にできます。

<backangle> は、矢先の後ろの部分の矢軸との切り角 (**<angle>** と同じ方向、単位は度) になりますが、スタイルが **nofilled** の場合はこれを無視します。

filled を指定すると、矢先の回りの線 (境界線) を描き、矢先を塗りつぶします。**noborder** を指定すると、矢先は塗りつぶしますが、境界線は描きません。この場合、矢先の先端がベクトルの終点ピッタリの場所に置かれ、その矢先は全体として少し小さくなります。点線で矢を描く場合は、点線の境界線は汚いので、常に **noborder** を使うべきです。矢先の塗りつぶしは、すべての出力形式がサポートしているとは限りません。

線種はユーザの定義したラインスタイルのリストから選ぶこともできますし (以下参照: **set style line (p. 285)**)、用意されている **<line_type>** の値 (デフォルトのラインスタイルのリストの番号) そして **<linewidth>** (デフォルトの幅の倍数) を使ってここで定義することもできます。

しかし、ユーザー定義済のラインスタイルが選択された場合、その属性 (線種、幅) は、単に他の **set style arrow** コマンドで適当な番号や **lt**, **lw** などを指定しても、変更はできないことに注意して下さい。

front を指定すると、矢はグラフのデータの上に描かれます。**back** が指定された場合 (デフォルト) は矢はグラフのデータの下に描かれます。**front** を使えば、密集したデータで矢が見えなくなることを防ぐことができます。

例:

矢先がなく、倍の幅が矢を描くには:

```
set style arrow 1 nohead lw 2
set arrow arrowstyle 1
```

その他の例については、以下参照: **set arrow (p. 194)**。

Boxplot スタイル指定 (boxplot)

コマンド **set style boxplot** により、描画スタイル **boxplot** で生成する描画のレイアウトを変更できます。

書式:

```
set style boxplot {range <r> | fraction <f>}
                  {{no}outliers} {pointtype <p>}
```

```
{candlesticks | financebars}
{medianlinewidth <width>}
{separation <x>}
{labels off | auto | x | x2}
{sorted | unsorted}
```

boxplot の箱は、常にデータ点の第一四分位から第三四分位の値の範囲にかかっています。箱から延長される箱ひげの限界は、2 つの異なる方法で制御できます。デフォルトでは、箱ひげは、その箱のそれぞれの端から、四分位範囲の 1.5 倍 (すなわち、その箱の厳密な垂直方向の高さ) に等しい範囲にまで延長されます。箱ひげそれぞれは、データ集合のある点に属する y の値で終了するように、メジアンに向かって切り捨てられます。四分位範囲の丁度 1.5 倍の値の点がない場合もありますから、箱ひげはその名目上の範囲よりも短くなる場合もあります。このデフォルトは以下に対応します。

```
set style boxplot range 1.5
```

もう一つの方法として、箱ひげがかかる点の総数の割合 (fraction) を指定することができます。この場合、その範囲はメジアン値から、データ集合の指定した分を囲い込むまで、対称に延長されます。このときも、個々の箱ひげはデータ集合内の点の端までに制限されます。データ集合の 95% の点をはるには以下のようにします。

```
set style boxplot fraction 0.95
```

箱ひげの範囲の外にある任意の点は、outliers (孤立した標本値) と見なされます。デフォルトではそれらをひとつひとつ円 (pointtype 7) で描きますが、オプション **nooutliers** はこれを無効にします。描画されない outliers は、autoscaling には影響を与えません。

デフォルトでは boxplot は candlesticks と似たスタイルで描画しますが、financebars と似たスタイルで描画するためのオプションもあります。

箱の境界と同じ線種を使って、メジアンを示す横断線を描きますが、そのメジアン線をより太くしたければ、以下のようにできます。

```
set style boxplot medianlinewidth 2.0
```

メジアン線が必要なければ、それを 0 にセットしてください。

boxplot の using 指定が 4 列目を持つ場合、その列の値は、このデータ点が所属する個々のカテゴリであると見なします。この場合、入力に存在するカテゴリそれぞれに対して一つずつの boxplot を描きます。これらの boxplot は x 軸方向に (x 軸の単位で) 1 の距離だけ間隔を空けて描画しますが、その間隔は、オプション **set style boxplot separation** で変更できます。

オプション **labels** は、これらの boxplot (それぞれデータ集合のある部分に対応する) のどこに、どのようにラベルをつけるかを決定します。デフォルトではカテゴリ識別子を水平軸 (x か x2 のいずれか plot で使われている方) の目盛ラベルとして使います。これはオプションの **labels auto** に対応します。オプション **labels x**, **labels x2** によって、強制的に x 軸、x2 軸にそれぞれ出力させることもできますし、**labels off** でオフにすることもできます。

デフォルトでは、異なるカテゴリに対する boxplot は、データファイルにそのカテゴリが現れる順番に描画します。この挙動はオプション **unsorted** に対応しますが、オプション **sorted** が有効な場合は、まずカテゴリ指定子を辞書順 (アルファベット順) にソートし、その順に boxplot を描画します。

オプション **separation**, **labels**, **sorted**, **unsorted** は、plot に 4 列目の指定を与えた場合のみ効力を持ちます。

以下参照: **boxplot (p. 94)**, **candlesticks (p. 96)**, **financebars (p. 102)**。

データ描画スタイル指定 (set style data)

コマンド **set style data** はデータ描画に対するデフォルトの描画スタイルを変更します。

書式:

```
set style data <plotting-style>
show style data
```

選択項目については、以下参照: **plotting styles (p. 90)**。**show style data** は現在のデフォルトのデータ描画スタイルを表示します。

塗り潰しスタイル指定 (set style fill)

コマンド **set style fill** は、boxes, histograms, candlesticks, filledcurves での描画における描画要素のデフォルトのスタイルの設定に使われます。このデフォルトは、個々の描画に塗り潰しスタイル (fillstyle) を指定することで上書きできます。**set obj** で生成する長方形 (rectangle) の塗り潰しスタイルには、別のデフォルトがあることに注意してください。 以下参照: **set style rectangle (p. 287)**。

書式:

```
set style fill {empty
                | {transparent} solid {<density>}
                | {transparent} pattern {<n>}}
               {border {lt} {lc <colorspec>} | noborder}
```

デフォルトの塗りつぶしスタイル (fillstyle) は **empty** です。

オプション **solid** は、その色での単色塗りを行います。パラメータ <density> は塗りつぶし色の彩度を表しています。<density> が 0.0 なら箱は白 (出力形式によってはその背景色)。<density> が 1.0 なら塗りつぶし領域は現在の線種か塗りつぶし色と完全に同じ色になります。出力形式によっては、この彩度を連続的に変化させられるものもありますが、その他のものは、部分的な塗りつぶしレベルを幾つか実装しているに過ぎません。パラメータ <density> を与えなかった場合はデフォルトの 1 になります。

transparent solid は、<density> をアルファ値と解釈する以外は同様に機能します。<density> 0.0 は完全な透過で、1.0 は透過なしのベタ塗りです。

オプション **pattern** は、出力ドライバによって与えられるパターンでの塗りつぶしを行います。利用できる塗りつぶしパターンの種類と数は出力ドライバに依存します。塗りつぶしの boxes スタイルで複数のデータ集合を描画する場合そのパターンは、複数の曲線の描画における線種の周期と同様、有効なパターンを、パターン <n> から始めて周期的に利用します。

オプション **empty** は、箱を塗りつぶしませんが、これがデフォルトです。

塗り潰し色 (**fillcolor <colorspec>**) は、塗り潰しスタイル (fill style) から分離されています。すなわち、fillstyle は描画要素やオブジェクトで別々の色を持ったまま共有することができます。fillstyle を受けつける大抵の場所で fillcolor を指定できます。fillcolor は **fc** の省略形も使えます。指定しない場合は、塗り潰し色は現在の線種 (linetype) から取ります。例:

```
plot FOO with boxes fillstyle solid 1.0 fillcolor "cyan"
```

Set style fill border キーワード **border** は、塗り潰しオブジェクトを現在の線種と色の実線で囲むようにします。その線の色は、linetype や linecolor を追加指定すれば変更できます。**noborder** は、境界の線を描かないようにします。例:

```
# 塗り潰しは強度半分、境界は同じ色で全強度
set style fill solid 0.5 border
# 塗り潰しは半分透過、境界は実線の黒 (線種 -1)
set style fill transparent solid 0.5 border -1
# 現在の色でのパターン塗り潰し、境界は線種 5 の色で
plot ... with boxes fillstyle pattern 2 border lt 5
# 水色 (cyan) での領域の塗り潰し、境界は青
plot ... with boxes fillcolor "cyan" fs solid border linecolor "blue"
```

注意: fill スタイルの border (境界) 属性は、デフォルトモードの closed (閉曲線) の **with filledcurves** のグラフにしか影響を与えません。

透明化 (set style fill transparent) いくつかの出力形式は、単色塗り領域の **transparent** (透明化) 属性をサポートしています。transparent solid の領域塗りつぶしでは、**density** (密度) パラメータはアルファ値として使用されます。つまり、密度 0 は完全な透明を、密度 1 は完全な不透明を意味します。transparent pattern の塗りつぶしでは、パターンの背景が完全な透明か完全な不透明のいずれかです。

透明な塗りつぶし領域を含むグラフを見たり作ったりするのには、別な制限がありうることに注意してください。例えば、png 出力形式では、"truecolor" オプションが指定されている場合にのみ透明化の塗り潰しが利用できます。PDF ファイルには透明化領域が正しく記述されていても、PDF の表示ソフトによってはそれを正しく表示できないこともありえます。実際に PostScript プリンタでは問題はないのに、Ghostscript/gv ではパターン塗りつぶし領域を正しく表示できません。

関数描画スタイル指定 (set style function)

コマンド **set style function** は関数描画に対するデフォルトの描画スタイル (lines, points, filledcurves など) を変更します。以下参照: **plotting styles (p. 90)**。

書式:

```
set style function <plotting-style>
show style function
```

ヒストグラムスタイル指定 (set style histogram)

以下参照: **histograms (p. 106)**。

線スタイル順指定 (set style increment)

デフォルトでは、同じグラフ上の次の描画は、次の線種で行われます。**set style increment userstyles** はこれを変更し、代わりにユーザ定義ラインスタイル番号に沿って行うようにしていました。

非推奨: gnuplot バージョン 6.0 ではこのコマンドは、configure 時にオプション –enable-backward-compatibility を指定してビルドしたものでのみ有効です。gnuplot が使用する線種の範囲を有用なものに再定義するには、代わりに **set linetype** を使用してください。以下参照: **set linetype (p. 238)**。

線スタイル指定 (set style line)

出力装置にはおのおのデフォルトの線種と点種の集合があり、それらはコマンド **test** で見ることができます。**set style line** は線種と線幅、点種と点の大きさを、個々の呼び出しで、それらの情報を全部指定する代わりに、単なる番号で参照できるようにあらかじめ定義するものです。

書式:

```
set style line <index> default
set style line <index> {{linetype  | lt} <line_type> | <colorspec>}
                       {{linecolor | lc} <colorspec>}
                       {{linewidth | lw} <line_width>}
                       {{pointtype | pt} <point_type>}
                       {{pointsize | ps} <point_size>}
                       {{pointinterval | pi} <interval>}
                       {{pointnumber | pn} <max_symbols>}
                       {{dashtype | dt} <dashtype>}
                       {palette}
unset style line
show style line
```

default は、全てのラインスタイルパラメータをそれと同じ index を持つ線種 (linetype) に設定します。

<index> の linestyle が既に存在する場合、他の全ては保存されたまま、与えられたパラメータのみが変更されます。<index> が存在しなければ、指定されなかった値はデフォルトの値になります。

このようにつくられるラインスタイルは、デフォルトの型 (線種, 点種) を別なものに置き換えることはしないので、ラインスタイル、デフォルトの型、どちらも使えます。ラインスタイルは一時的なもので、コマンド **reset** を実行すればいつでもそれらは消え去ります。線種自体を再定義したい場合は、以下参照: **set linetype (p. 238)**。

線種と点種は、その index 値をデフォルトとします。その index 値に対する実際の記号の形は、出力形式によって異なり得ます。

線幅と点の大きさは、現在の出力形式のデフォルトの幅、大きさに対する乗数です (しかし、ここでの <point_size> は、コマンド **set pointsize** で与えられる乗数には影響を受けないことに注意してください)。

pointinterval は、スタイル **linespoints** でグラフ中に描かれる点の間隔を制御します。デフォルトは 0 です (すべての点が描画される)。例えば、**set style line N pi 3** は、点種が N、点の大きさと線幅は現在の出力形式のデフォルトで、**with linespoints** での描画では点は 3 番目毎に描画されるようなラインスタイルを定義します。その間隔を負の値にすると、それは間隔は正の値の場合と同じですが、点の記号の下になる線を書かないようにします (出力形式によっては)。

pointnumber 属性は、**pointinterval** と似ていますが、N 個おきに描くようにする代わりに、全体の点の個数を N 個に限定するところが違います。

全ての出力装置が **linewidth** や **pointsize** をサポートしているわけではありません。もしサポートされていない場合はそれらのオプションは無視されます。

出力形式に依存しない色を **linecolor** <colorspec> か **linetype** <colorspec> (省略形は **lc**, **lt**) のいずれかを使って割り当てることができます。この場合、色は RGB の 3 つ組で与えるか、gnuplot の持つパレットの色名、現在のパレットに対する小数指定、または cbrange への現在のパレットの対応に対する定数値、のいずれかで与えます。以下参照: **colors (p. 70)**, **colorspec (p. 71)**, **set palette (p. 258)**, **colornames (p. 207)**, **cbrange (p. 317)**。

set style line <n> **linetype** <lt> は、出力形式に依存した点線/破線のパターンと色の両方をセットします。**set style line** <n> **linecolor** <colorspec> や **set style line** <n> **linetype** <colorspec> は、現在の点線/破線のパターンを変更せずに新しい線色を設定します。

3 次元モード (**splot** コマンド) では、"linetype palette z" の省略形として特別にキーワード **palette** を使うことも許されています。その色の値は、splot の z 座標 (高さ) に対応し、曲線、あるいは曲面に沿って滑らかに変化します。

例: 以下では、番号 1, 2, 3 に対するデフォルトの線種をそれぞれ赤、緑、青とし、デフォルトの点の形をそれぞれ正方形、十字、三角形であるとします。このとき以下のコマンド

```
set style line 1 lt 2 lw 2 pt 3 ps 0.5
```

は、新しいラインスタイルとして、緑でデフォルトの 2 倍の幅の線、および三角形で半分の幅の点を定義します。また、以下のコマンド

```
set style function lines
plot f(x) lt 3, g(x) ls 1
```

は、f(x) はデフォルトの青線で、g(x) はユーザの定義した緑の線で描画します。同様に、コマンド

```
set style function linespoints
plot p(x) lt 1 pt 3, q(x) ls 1
```

は、p(x) を赤い線で結ばれたデフォルトの三角形で、q(x) は緑の線で結ばれた小さい三角形で描画します。

```
splot sin(sqrt(x*x+y*y))/sqrt(x*x+y*y) w l pal
```

は、**palette** に従って滑らかな色を使って曲面を描画します。これはそれをサポートした出力形式でしかちゃんとは動作しないことに注意してください。以下も参照: **set palette (p. 258)**, **set pm3d (p. 267)**。

```
set style line 10 linetype 1 linecolor rgb "cyan"
```

は、RGB カラーをサポートするすべての出力形式で、ラインスタイル 10 に実線の水色を割り当てます。

円スタイル指定 (set style circle)

書式:

```
set style circle {radius {graph|screen} <R>}
                 {{no}wedge}
                 {clip|noclip}
```

このコマンドは、描画スタイル "with circles" で使われるデフォルトの半径を設定します。これは、データ描画で 2 列のデータ (x,y) しか与えなかった場合、あるいは関数描画のときに適用されます。デフォルトは、以下のようになっています: "set style circle radius graph 0.02"。**nowedge** は、扇形の円弧部分から中心に向かう 2 本の半径を描かないようにしますが、デフォルトは **wedge** です。このパラメータは完全な円に対しては何もしません。**clip** は円を描画境界でクリッピングしますが、**noclip** はこれを無効にします。デフォルトは **clip** です。

長方形スタイル指定 (set style rectangle)

コマンド **set object** で定義された長方形には別々のスタイルを設定できます。しかし、個別のスタイル指定をしなければ、そのオブジェクトはコマンド **set style rectangle** によるデフォルトを受け継ぎます。

書式:

```
set style rectangle {front|back} {lw|linewidth <lw>}
                    {fillcolor <colorspec>} {fs <fillstyle>}
```

以下参照: **colorspec (p. 71)**, **fillstyle (p. 283)**。**fillcolor** は **fc** と省略できます。

例:

```
set style rectangle back fc rgb "white" fs solid 1.0 border lt -1
set style rectangle fc linestyle 3 fs pattern 2 noborder
```

デフォルトの設定は、背景色での単色塗り (solid fill) で、境界は黒になっています。

楕円スタイル指定 (set style ellipse)

書式:

```
set style ellipse {units xx|xy|yy}
                  {size {graph|screen} <a>, {{graph|screen} <b>}}
                  {angle <angle>}
                  {clip|noclip}
```

このコマンドは、楕円の直径を同じ単位で計算するかどうかを制御します。デフォルトは **xy** で、これは楕円の主軸 (第 1 軸) の直径は x (または x2) 軸と同じ単位で計算し、副軸 (第 2 軸) の直径は y (または y2) 軸の単位で計算します。このモードでは、楕円の両軸の比は、描画軸のアスペクト比に依存します。**xx** か **yy** に設定すれば、すべての楕円の両軸は同じ単位で計算されます。これは、描画される楕円の両軸の比は、回転しても正しいままですが、水平方向か垂直方向の一方の縮尺の変更により正しくなくなることを意味します。

これは、object として定義された楕円、コマンド **plot** によって描画される楕円の両方に影響を与える全体的な設定ですが、**units** の値は、描画毎、オブジェクト毎に設定を再定義できます。

楕円のデフォルトのサイズも、キーワード **size** で設定できます。デフォルトのサイズは、2 列のみのデータ、または関数の plot 命令で適用されます。2 つの値は、楕円の (2 つの主軸、2 つの副軸に向かい合う) 主軸直径と副軸直径として使用されます。

デフォルトは、"set style ellipse size graph 0.05,0.03" です。

最後になりますが、デフォルトの向きをキーワード **angle** で設定もできます。向きは、楕円の主軸とグラフの x 軸の方向となす角で、単位は度で与える必要があります。

clip は楕円を描画境界でクリッピングしますが、**noclip** はこれを無効にします。デフォルトは **clip** です。

楕円の object の定義に関しては以下も参照: **set object ellipse (p. 254)**。2 次元の描画スタイルに関しては以下参照: **ellipses (p. 100)**。

平行座標スタイル指定 (set style parallelaxis)

書式:

```
set style parallelaxis {front|back} {line-properties}
```

これは、**with parallelaxes** グラフの垂直軸を書く際の線種とレイヤーを指定します。以下参照: **with parallelaxes (p. 117)**, **set paxis (p. 265)**。

クモの巣グラフスタイル指定 (set style spiderplot)

書式:

```
set style spiderplot
          {fillstyle <fillstyle-properties>}
          {<line-properties> | <point-properties>}
```

このコマンドは、クモの巣グラフ (spider plot) のデフォルトの見た目を制御します。塗り潰し、線分、点の属性は、plot コマンドの最初の要素で変更できます。全体的なグラフの見た目は、**set grid spiderplot** などの他の設定の影響も受けます。以下も参照: **set paxis (p. 265)**、**spiderplot (p. 122)**。例:

```
# デフォルトのクモの巣グラフを太い境界で塗り潰しなしの多角形に
set style spiderplot fillstyle empty border lw 3
# 以下は各軸に円 (pt 6) を追加する
plot for [i=1:6] DATA pointtype 6 pointsize 3
```

文字列ボックススタイル指定 (set style textbox)

書式:

```
set style textbox {<boxstyle-index>}
          {opaque|transparent} {fillcolor <color>}
          {{no}border {linecolor <colorspec>}}{linewidth <lw>}
          {margins <xmargin>,<ymargin>}
```

このコマンドは、属性 **boxed** による label の表示を制御します。箱付き文字列をサポートしない出力形式はこのスタイルを無視します。注意: いくつかの出力形式 (svg, latex) での実装は不完全です。また、ほとんどの出力形式は、回転した文字列の箱付けが正しくできません。

番号付きの textbox スタイルを 3 種類定義できます。boxstyle 番号 <bs> を指定しないと、デフォルトスタイル (番号なし) を変更します。例:

```
# デフォルトのスタイルは、黒の境界線のみ
set style textbox transparent border lc "black"
# スタイル 2 (bs 2) を明るい青背景で境界なしに
set style textbox 2 opaque fc "light-cyan" noborder
set label 1 "I'm in a box" boxed
set label 2 "I'm blue" boxed bs 2
```

ウォッチポイントスタイル指定 (set style watchpoint)

書式:

```
set style watchpoint nolabels
set style watchpoint labels {label-options}
```

ウォッチポイントの対象 "mouse" については常にグラフ上にラベルを出力します。他のウォッチポイント対象には、スタイルが **label** か **nolabel** のどちらに設定されているかによってラベルを表示、または非表示とします。

ウォッチポイントラベルの見た目は、他の gnuplot label のラベル属性にあるような全機能を使ってカスタマイズできます。例えばフォント、文字色や、実際の x,y 座標をマークする点の点種、点サイズを設定できます。以下参照: **set label (p. 235)**。

plot コマンドでラベル文字列、ラベル関数のどちらも与えなかった場合、gnuplot は、現在のグラフに対する軸の見出しに使用する書式を使用して、文字列 " x-座標 : y-座標" を自動生成します。以下参照: **watchpoint labels (p. 88)**。

例:

```
set style watchpoint labels point pt 4 ps 2
set style watchpoint labels font ":Italic,6" textcolor "blue"
set style watchpoint labels boxed offset 1, 0.5
```

曲面描画 (surface)

コマンド **set surface** は 3 次元描画 (**splot**) にのみ関係します。

書式:

```
set surface {implicit|explicit}
unset surface
show surface
```

unset surface により **splot** は、関数やデータファイルの点に対するどんな点や線も描かなくなります。これは主に、等高線を作る曲面を描く代わりに等高線のみを描く場合に有用です。その場合でも **set contour** の設定によりますが、曲面上に等高線が描かれます。他のものは通常のままで、ある一つの関数やデータファイルの曲面のみをオフにするには、**splot** コマンド上でキーワード **nosurface** を指定してください。等高線を格子の土台に表示したい場合は **unset surface; set contour base** という組が便利でしょう。以下も参照: **set contour (p. 207)**。

3 次元データの組が網目 (格子線) と認識されると、gnuplot はデフォルトでは格子曲面を要求しているものとして、暗黙に **with lines** の描画スタイルを用います。以下参照: **grid_data (p. 325)**。コマンド **set surface explicit** はこの機能を抑制し、入力ファイルのデータの分離されたブロックで記述される孤立線のみを描画します。この場合でも、splot で明示的に **with surface** とすれば格子曲面が描画されます。

テーブルデータ出力 (table)

table モードが有効な場合、**plot** と **splot** コマンドは、現在の出力形式に対する実際の描画を生成する代わりに

```
X Y {Z} <flag>
```

の値の複数列からなる表形式のテキスト出力を行ないます。フラグ文字 <frag> は、その点が有効な範囲内にある場合は "i"、範囲外の場合は "o"、未定義値 (undefined) の場合は "u" です。データの書式は、軸の刻みの書式 (以下参照: **set format (p. 219)**) によって決まり、列は一つの空白で区切られます。これは、等高線を生成し、それを再利用のために保存したいときに便利です。この方法は、補間されたデータを保存するのにも使うことができます (以下参照: **set samples (p. 278)**, **set dgrid3d (p. 213)**)。

書式:

```
set table {"outfile" | $datablock} {append}
          {separator {whitespace|tab|comma|"<char>"}}
plot <whatever>
unset table
```

この後の表形式の出力は、ファイル "outfile" を指定していればそれに書き出しますが、そうでなければ標準出力か、現在の **set output** が指定するものに出力します。**outfile** が既に存在する場合は、**append** キーワードを指定すれば追加出力、指定しなければこの出力が上書きします。他に、表形式出力を名前付きデータブロックにリダイレクトすることもできます。データブロック名は '$' で始まります。以下も参照: **inline data (p. 68)**。現在の出力形式の標準的な描画に戻すには、**unset table** を明示的に行なう必要があります。

separator 文字は、CSV ファイル (コンマ区切り) の出力に使えます。そのモードは、描画スタイル **with table** にのみ影響を与えます。以下参照: **plot with table (p. 290)**。

Plot with table

以下の説明は、特別な描画スタイル **with table** にのみ適用されます。

表データに変換される入力データに対する描画スタイルに依存する処理 (平滑化、誤差線の延長、2 軸範囲のチェック等) を避けるため、あるいは表データに変換できる列の数を増やすには、通常の描画スタイルの代わりに "table" キーワードを使ってください。この場合、範囲内/範囲外/未定義を意味するフラグ **i**, **o**, **u** を含む追加列は、出力にはつきません。その出力先は、最初に **set table <where>** で指定する必要があります。例:

```
set table $DATABLOCK1
plot <file> using 1:2:3:4:($5+$6):(func($7)):8:9:10 with table
```

この場合、実際の描画スタイルがない状態なので、各列には特定の軸は対応しないことになり、よって xrange, yrange 等の設定は無視されます。

文字列に対して **using** で評価した場合は、文字列も表データ化されます。数値データは常に %g の書式で書き出されますが、他の書式を使いたい場合は、そのように書式化された文字列を生成するように sprintf か gprintf を使用してください。

```
plot <file> using ("File 1"):1:2:3 with table
plot <file> using (sprintf("%4.2f",$1)) : (sprintf("%4.2f",$3)) with table
```

CSV ファイルを生成するには、以下のようにします。

```
set table "tab.csv" separator comma
plot <foo> using 1:2:3:4 with table
```

表データ化用にデータ点の一部分のみを選択するには、入力フィルタ条件 (**if** <**expression**>) をコマンドの最後に指定することができます。その入力フィルタは、出力はされないデータ列も参照できることに注意してください。

```
plot <file> using 1:2:($4+$5) with table if (strcol(3) eq "Red")
plot <file> using 1:2:($4+$5) with table if (10. < $1 && $1 < 100.)
plot <file> using 1:2:($4+$5) with table if (filter($6,$7) != 0)
```

出力形式 (terminal)

gnuplot は数多くのグラフィック形式をサポートしています。コマンド **set terminal** を使って **gnuplot** の出力の対象となる形式の種類を選んでください。出力先をファイル、または出力装置にリダイレクトするには **set output** を使ってください。

書式:

```
set terminal <terminal-type> <options>
set terminal push
set terminal pop
show terminal
```

<terminal-type> を省略すると **gnuplot** は利用可能なすべての出力形式の一覧を表示します。出力形式の設定時に、出力形式毎に指定できるたくさんのオプションがあります。以下も参照: **termoption (p. 292)**。

set terminal と **set output** の両方を使う場合、**set terminal** を最初にする方が安全です。それは、出力ストリームを正しく初期化するためには、出力形式がなんであるかを知る必要がある場合があるからです。

例:

```
set terminal pdf font "Times,11" size 5in, 3in
set output "figure_1.pdf"
plot FILE1 with lines
set output "figure_2.pdf"
plot FILE2 with points
unset output
```

警告: 同じ出力形式を 1 度のセッション内で 2 度以上呼び出す場合でも、前に指定した出力形式用のオプションは、後の方でも指定する必要があるかもしれません。

コマンド **set term push** は、現在の出力形式とその設定を **set term pop** によって復帰するまで、事前に設定したすべてのオプションとともに記憶しています。これは、印刷時の助けになります。例えば、違う出力形式に切り替えるときに、前のオプションを繰り返さなくて済みます。gnuplot は起動時に、デフォルト、または **startup** ファイルに書かれた出力形式を自動的に記憶 (push) するので、明示的に出力形式を記憶させなくても、任意のプラットホーム上でデフォルトの出力形式が **set term pop** で復帰すると期待するスクリプトを可搬性を失わずに書くことが出来ます。

例:

```
set terminal kitty anchor fontscale 0.5 size 700,500
load "myplot.gp"
set terminal push
set terminal pdf font "Times,11" size 5in, 3in
set output "myplot.pdf"
replot
unset output      # 必要! kitty 出力を PDF ファイルに書かないため
set term pop      # kitty 出力形式とオプションを復帰
```

詳細は、以下参照: **complete list of terminals (p. 335)**。

出力形式へのオプション (termoption)

コマンド **set termoption** は、現在使用している出力形式の振舞いを、新たな **set terminal** コマンドの発行なしに変更することを可能にします。このコマンド一つに対して一つのオプションのみが変更できます。そしてこの方法で変更できるオプションはそう多くはありません。現在使用可能なオプションは以下のもののみです。

```
set termoption {no}enhanced
set termoption font "<fontname>{,<fontsize>}"
set termoption fontscale <scale>
set termoption {linewidth <lw>}{lw <lw>} {dashlength <dl>}{dl <dl>}
set termoption {pointscale <scale>} {ps <scale>}
```

極座標方位制御 (theta)

極座標グラフは、デフォルトでは角の 0 の方向 ($\theta = 0$) がグラフの右側を指すよう向きづけられ、角の増加は反時計回りに行われ、$\theta = 90$ が真上を向くようになっています。**set theta** により、極座標の角の座標に関する 0 の向きと増加方向を変更できます。

```
set theta {right|top|left|bottom}
set theta {clockwise|cw|counterclockwise|ccw}
```

unset theta は、デフォルトの状態 "set theta right ccw" に復帰します。

全軸目盛り制御 (tics)

コマンド **set tics** は、全ての軸の目盛りの刻みとラベルを一度に制御します。

目盛りは **unset tics** で消え、**set tics** で目盛りがつきます (デフォルト)。個々の軸の目盛りは、これとは別のコマンド **set xtics**, **set ztics** などを使って制御できます。

書式:

```
set tics {axis | border} {{no}mirror}
         {in | out} {front | back}
         {{no}rotate {by <ang>}} {offset <offset> | nooffset}
         {left | right | center | autojustify}
         {format "formatstring"} {font "name{,<size>}"} {{no}enhanced}
         { textcolor <colorspec> }
set tics scale {default | <major> {,<minor>}}
unset tics
show tics
```

オプションは、個々の軸 (x, y, z, x2, y2, cb) にも適用できます。例:

```
set xtics rotate by -90
unset cbtics
```

軸の刻みの線属性は、すべてグラフの境界 (以下参照: **set border (p. 198)**) と同じものを使って描かれます。

tics の **back** または **front** の設定は、2D 描画 (splot は不可) にのみすべての軸に 1 度適用されます。これは、目盛りと描画要素が重なった場合に目盛りを描画要素の前面に出すか、奥に置くかを制御します。

axis と **border** は **gnuplot** に目盛り (目盛りの刻み自身とその見出し) を、それぞれ軸につけるのか、境界につけるのかを指示します。軸が境界にとても近い場合、**axis** を使用すると境界が表示されていれば (以下参照: **set border (p. 198)**) 目盛りの見出し文字を境界の外に出してしまうでしょう。この場合自動的なレイアウトアルゴリズムによる余白設定は大抵よくないものとなってしまいます。

mirror は **gnuplot** に反対側の境界の同じ位置に、見出しのない目盛りを出力するよう指示します。**nomirror** は、あなたが想像している通りのことを行ないます。

in と **out** は目盛りの刻みを内側に描くか外側に描くかを切り変えます。

set tics scale は、目盛りの刻みの大きさを制御します。最初の <major> の値には、自動的に生成され、またユーザも指定できる大目盛り (レベル 0) を指定し、2 つ目の <minor> の値には、自動的に生成され、またユーザも指定できる小目盛り (レベル 1) を指定します。<major> のデフォルトは 1.0 で、<minor> のデフォルトは <major>/2 です。さらに値を追加すれば、レベル 2, 3, ... の目盛りの大きさになります。**set tics scale default** でデフォルトの目盛りの大きさに復帰します。

rotate は、文字列を 90 度回転させて出力させようとします。これは、文字列の回転をサポートしている出力ドライバ (terminal) では実行されます。**norotate** はこれをキャンセルします。**rotate by** <**ang**> は角度 <ang> の回転を行ないますが、これはいくつかの出力形式 (terminal) でサポートされています。

x と y 軸の大目盛りのデフォルトは **border mirror norotate** で、x2, y2 軸は **border nomirror norotate** がデフォルトです。z 軸のデフォルトは **nomirror** です。

<offset> は x,y かまたは x,y,z の形式ですが、それに座標系を選択して、その前に **first**, **second**, **graph**, **screen**, **character** のいずれかをつけることもできます。<offset> は、目盛りの見出し文字列のデフォルト

の位置からのずらし位置で、そのデフォルトの単位系は **character** です。詳細は、以下参照: **coordinates (p. 41)**。**nooffset** は offset を OFF にします。

デフォルトでは見出しラベルは、美しい結果を生むように、軸と回転角に依存した位置に自動的に揃えられますが、気にいらなければ、明示的に **left**, **right**, **center** のキーワードにより位置揃えを変更できます。**autojustify** でデフォルトの挙動に復帰します。

オプションなしの **set tics** は、第 1 軸に対する鏡映 (mirror) と内側向き目盛りの刻みをデフォルトの挙動に復帰しますが、その他の全てのオプションは、直前の値を保持します。

大目盛り (ラベルのつく) の他の制御に関しては、以下参照: **set xtics (p. 307)**。小目盛りの制御に関しては、以下参照: **set mxtics (p. 249)**。これらのコマンドは、各軸の独立な制御を提供します。

Ticslevel

現在は推奨されていません。以下参照: **set xyplane (p. 312)**。

Ticscale

コマンド **set ticscale** は現在は推奨されていません。代わりに **set tics scale** を使ってください。

タイムスタンプ (timestamp)

コマンド **set timestamp** は現在の時刻と日付をグラフの余白に表示します。

書式:

```
set timestamp {"<format>"} {top|bottom} {{no}rotate}
              {offset <xoff>{,<yoff>}} {font "<fontspec>"}
              {textcolor <colorspec>}
unset timestamp
show timestamp
```

書式文字列 (format) は、書かれる日付と時刻の書式に使用されます。デフォルトは asctime() が使用する "%a %b %d %H:%M:%S %Y" です (曜日、月名、日、時、分、秒、4 桁の西暦)。**top** と **bottom** を使って、日付の配置を左上、左下のいずれの余白にするかを選択できます (デフォルトは左下)。**rotate** は、日付を垂直方向に書き出します。定数 <xoff>、<yoff> はずれ (offset) を意味し、これによってより適切な位置決めが行えます。<font> は日付が書かれるフォントを指定します。

例:

```
set timestamp "%d/%m/%y %H:%M" offset 80,-2 font "Helvetica"
```

日付の書式文字列に関する詳しい情報については、以下参照: **set timefmt (p. 295)**。

日時データ入力書式 (timefmt)

このコマンドは、日時データの入力で使用するデフォルトの書式を設定します。以下参照: **set xdata time** (p. **302**), **timecolumn** (p. **56**)。

書式:

```
set timefmt "<format string>"
show timefmt
```

timefmt と **timecolumn** の両方で有効な書式は以下の通りです:

時系列データ書式指定子	
書式	説明
`%d`	何日, 1–31
`%m`	何月, 1–12
`%y`	何年, 0–99
`%Y`	何年, 4 桁
`%j`	1 年の何日目, 1–365
`%H`	何時, 0–24
`%M`	何分, 0–60
`%s`	Unix epoch (1970-01-01, 00:00 UTC) からの秒数
`%S`	何秒 (出力では 0–60 の整数、入力では実数)
`%b`	月名 (英語) の 3 文字省略形
`%B`	月名 (英語)
`%p`	2 文字の am AM pm PM のいずれか

任意の文字を文字列中で使用できますが、規則に従っている必要があります。\t (タブ) は認識されます。バックスラッシュ + 8 進数列 (\nnn) はそれが示す文字に変換されます。日時要素の中に分離文字がない場合、%d, %m, %y, %H, %M, %S はそれぞれ 2 桁の数字を読み込みます。%S での読み込みで小数点がそのフィールドについている場合は、その小数点つきの数を小数の秒指定だと解釈します。%Y は 4 桁、%j は 3 桁の数字を読み込みます。%b は 3 文字を、%B は必要な分だけの文字を要求します。

空白 (スペース) の扱いはやや違います。書式文字列中の 1 つの空白は、ファイル中の 0 個、あるいは 1 つ以上の空白文字列を表します。すなわち、"%H %M" は "1220" や "12 20" を "12 20" と同じように読みます。

データ中の非空白文字の集まりそれぞれは、**using n:n** 指定の一つ一つの列とカウントされます。よって **11:11 25/12/76 21.0** は 3 列のデータと認識されます。混乱を避けるために、日時データが含まれる場合 **gnuplot** は、あなたの **using** 指定が完璧なものであると仮定します。

日付データが曜日、月の名前を含んでいる場合、書式文字列でそれを排除しなければいけませんが、"%a", "%A", "%b", "%B" でそれらを表示することはできます。**gnuplot** は数値から月や曜日を正しく求めます。これら、及び日時データの出力の他のオプションの詳細に関しては、以下参照: **set format** (p. **219**)。

2 桁の西暦を %y で読む場合、69-99 は 2000 年未満、00-68 は 2000 年以後と見なします。注意: これは、UNIX98 の仕様に合わせたものですが、この慣例はあちこちで違いがあるので、2 桁の西暦値は本質的にあいまいです。

書式 %p が "am" か "AM" を返す場合、12 時は 0 時と解釈します。書式 %p が "pm" か "PM" を返す場合、12 未満の時刻には 12 追加します。

他の情報については、以下も参照: **set xdata** (p. 302), **time/date** (p. 85), **time_specifiers** (p. 221)。

例:

```
set timefmt "%d/%m/%Y\t%H:%M"
```

は、gnuplot に日付と時間がタブで分離していることを教えます (ただし、あなたのデータをよーく見てください。タブだったものがどこかで複数のスペースに変換されていませんか ? 書式文字列はファイル中に実際にある物と一致していなければなりません)。以下も参照時系列データ (time data) デモ。

グラフタイトル (title)

コマンド **set title** は、描画の上の真中に書かれる描画タイトルを生成します。**set title** は **set label** の特殊なもの、とみなせます。

書式:

```
set title {"<title-text>"} {offset <offset>} {font "<font>{,<size>}"}
          {{textcolor | tc} {<colorspec> | default}} {{no}enhanced}
show title
```

<offset> を x,y かまたは x,y,z の形式で指定した場合は、タイトルは与えられた値だけ移動されます。それに座標系を選択して、その前に **first**, **second**, **graph**, **screen**, **character** のいずれかをつけることもできます。詳細は、以下参照: **coordinates** (p. 41)。デフォルトでは **character** 座標系が使われます。例えば、"**set title offset 0,-1**" はタイトルの y 方向の位置のみ変更し、大ざっぱに言って 1 文字分の高さだけタイトルを下に下げます。1 文字の大きさは、フォントと出力形式の両方に依存します。

<font> はタイトルが書かれるフォントを指定するのに使われます。<size> の単位は、どの出力形式 (terminal) を使っているかによって変わります。

textcolor <colorspec> は、文字の色を変更します。<colorspec> は、線種、rgb 色、またはパレットへの割当のいずれかが指定できます。以下参照: **colorspec** (p. 71), **palette** (p. 51)。

noenhanced は、拡張文字列処理 (enhanced text) モードが有効になっている場合でも、タイトルを拡張文字列処理させないようにします。

set title をパラメータなしで使うとタイトルを消去します。

バックスラッシュ文字列の作用、及び文字列を囲む単一引用符と二重引用符の違いについては、以下参照: **syntax** (p. 84)。

Tmargin

コマンド **set tmargin** は上の余白のサイズをセットします。詳細は、以下参照: **set margin** (p. 242)。

Trange

書式: set trange [tmin:tmax] 媒介変数 t の範囲は、以下の 3 つの状況で有効です。

- 媒介変数モードでは、これがコマンド **plot** の両方の生成関数の標本範囲を制限します。以下参照: **set parametric (p. 265)**, **set samples (p. 278)**。
- 極座標モードでは、これがコマンド **plot** の偏角変数θ の入力中の許容可能な範囲を制限、または定義します。この範囲外のθ を持つデータ点は、それが描画境界の内側に入っていたとしても、グラフからは除外されます。以下参照: **polar (p. 274)**。
- **plot** または **splot** コマンドでは、疑似ファイル "+" による 1 次元データの標本化に使用します。以下参照: **sampling 1D (p. 182)**, **special-filenames (p. 174)**。

Ttics

コマンド **set ttics** は、極座標グラフの周囲に目盛り刻みをつけます。それは、**set border polar** とした場合は境界になりますが、そうでなければ極座標格子の、r 軸に沿って最も右端の刻みの場所に書かれる最も外側の円周になります。以下参照: **set grid (p. 223)**, **set rtics (p. 277)**。角の位置は、常に度の単位でラベル付けされます。現在の角の範囲の設定 (trange) に関係なく、同心円全体に刻みラベルをつけることができます。刻みラベルが必要な範囲の設定は、下に示す例のようにしてください。刻みに関する追加属性を設定することもできます。以下参照: **xtics (p. 307)**。

```
set ttics -180, 30, 180
set ttics add ("Theta = 0" 0)
set ttics font ":Italic" rotate
```

Urange

書式: set urange [umin:umax] 媒介変数 u と v の範囲は、以下の 2 つの状況で有効です。1) 媒介変数モードの **splot**。以下参照: **set parametric (p. 265)**,

‘set isosamples‘。

2) **plot** または **splot** コマンドでは、疑似ファイル "++" による 2 次元

標本化データの生成に使用します。以下参照: ‘sampling 2D‘。

Version

コマンド **show version** は現在起動している gnuplot のバージョン、最終修正日、著作権者と、FAQ や info-gnuplot メーリングリスト、バグレポート先のメールアドレスを表示します。対話的にプログラムが呼ばれているときはスクリーン上にその情報を表示します。

書式:

```
show version {long}
```

show version **long** では、その **gnuplot** がコンパイルされたときに使われたオペレーティングシステム、設定、コンパイルオプションなども表示します。

Vgrid

書式:

```
set vgrid $gridname {size N}
unset vgrid $gridname
show vgrid
```

指定した名前付き格子 $gridname が既に存在していれば、こえはそれを有効にし、これに続く **vfill** と **voxel** の操作で使えるようにします。サイズを新たに指定した場合、存在するデータを 0 で埋めた N x N x N 格子に置き換えます。その名前の格子がまだない場合は、N x N x N 格子 (デフォルトは N=100) の領域をメモリに割り当て、その中身を 0 にし、有効にします。格子名は、'$' から始めなければいけないことに注意してください。

show vgrid は、現在定義済みのボクセル格子すべての一覧を表示します。出力例:

```
$vgrid1: (active)
         size 100 X 100 X 100
         vxrange [-4:4]  vyrange[-4:4]  vzrange[-4:4]
         non-zero voxel values:  min 0.061237 max 94.5604
         number of zero voxels:  992070   (99.21%)
```

unset vgrid $gridname は、そのボクセル格子に関係するすべてのデータ領域をメモリから開放します。そのデータ領域は、**reset session** でも開放されます。関数 **voxel(x,y,z)** は、その座標に一番近い有効な格子点の値を返します。以下も参照: **splot voxel-grids (p. 326)**。

視線方向 (view)

コマンド **set view** は **splot** の視線の角度を設定します。これは、グラフ描画の 3 次元座標をどのように 2 次元の画面 (screen) に投影するかを制御します。これは、描画されたデータの回転と縮尺の制御を与えてくれますが正射影しかサポートしていません。3 次元射影、および 2 次元描画的地図上への 2 次元直交射影がサポートされています。

書式:

```
set view <rot_x>{,{<rot_z>}{,{<scale>}{,<scale_z>}}}
set view map {scale <scale>}
set view projection {xy|xz|yz}
set view {no}equal {xy|xyz}
set view azimuth <angle>
show view
```

ここで <rot_x> と <rot_z> は、画面に投影される仮想的な 3 次元座標系の回転角 (単位は度) の制御で、最初は (すなわち回転が行なわれる前は) 画面内の水平軸は x, 画面内の垂直軸は y, 画面自身に垂直な軸が z となっています。最初は x 軸の周りに <rot_x> だけ回転されます。次に、新しい z 軸の周りに <rot_z> だけ回転されます。

コマンド **set view map** は、グラフを地図のように表示するのに使います。これは等高線 (**contour**) のグラフや、pm3d モードによる 2 次元温度分布 (heatmap) などで **with image** よりもむしろ有用です。後者では、

入力データ点のフィルタ用の **zrange** の設定、および色の範囲の縮尺に関する **cbrange** の設定を適切に行うことに注意してください。

<rot_x> は [0:180] の範囲に制限されていて、デフォルトでは 60 度です。<rot_z> は [0:360] の範囲に制限されていて、デフォルトでは 30 度です。<scale> は **splot** 全体の伸縮率を制御し、<scale_z> は z 軸の伸縮のみを行ないます。伸縮率のデフォルトはどちらも 1.0 です。

例:

```
set view 60, 30, 1, 1
set view ,,0.5
```

最初の例は 4 つの全てをデフォルトの値にしています。2 つめの例は縮小率のみを 0.5 に変更しています。

Azimuth

```
set view azimuth <angle-in-degrees>
```

azimuth の設定は、3 次元グラフ (splot) の z 軸の向きに影響します。デフォルトは azimuth = 0 で、グラフの z 軸はスクリーンの水平方向に対して垂直な平面に含まる、すなわち、z 軸の 2 次元射影がスクリーンの鉛直方向になります。0 でない azimuth は視界を原点に関して回転し、z 軸の射影は鉛直方向ではなくなります。azimuth = 90 では z 軸は鉛直方向ではなく、水平方向になります。対話型の表示の際、ホットキー **z** は azimuth を 0 にリセットします。

Equal_axes

コマンド **set view equal xy** は x 軸と y 軸の単位の長さが強制的に等しくなるように縮尺を合わせ、グラフがページに丁度合うようにその縮尺を選択します。コマンド **set view equal xyz** は、さらに z 軸も x と y 軸に合うようにしますが、z 軸の範囲が、描画境界の範囲に合う保証はありません。以下も参照: **set isotropic (p. 228)**。デフォルトでは、3 つの軸は独立に有効な領域を埋めるように伸縮します。

以下も参照: **set xyplane (p. 312)**。

Projection

書式:

```
set view projection {xy|xz|yz}
```

3 次元グラフの視角を回転し、主平面 xy, xz, yz のいずれかがグラフの平面に乗るようにします。軸の目盛りとラベルの配置はそれに従って調整され、それに垂直な第 3 軸の目盛りとラベルは無効になります。そのグラフは、'plot' が同じ軸の範囲で生成するものとほぼ一致するサイズにスケール変換します。**set view projection xy** は、**set view map** と同じです。

オブジェクトやラベル、矢、その他の描画要素を指定する x, y の座標の両方が "graph" 座標の場合、それは射影面では "x/y" ではなく、"水平/鉛直" 値として解釈します。

```
set key top right at graph 0.95, graph 0.95      # 任意の射影で動作
```

Vrange

書式: set vrange [vmin:vmax] 媒介変数 u と v の範囲は、以下の 2 つの状況で有効です。1) 媒介変数モードの splot。以下参照: **set parametric (p. 265)**,

‘set isosamples‘。

2) **plot** または **splot** コマンドでは、疑似ファイル "++" による 2 次元
標本化データの生成に使用します。以下参照: ‘sampling 2D‘。

Vxrange

書式: set vxrange [vxmin:vxmax]

これは、現在有効なボクセル格子が占める x 座標の範囲を設定します。ボクセル格子の他の 2 つの方向用に、これと同様のコマンド **set vyrange**, **set vzrange** があります。最初の **vclear**, **vfill**, **voxel(x,y,z) =** のコマンドの前に明示的な範囲が設定されていなかった場合、vmin と vmax は現在の **xrange** の範囲をコピーします。

Vyrange

以下参照: **set vxrange (p. 300)**。

Vzrange

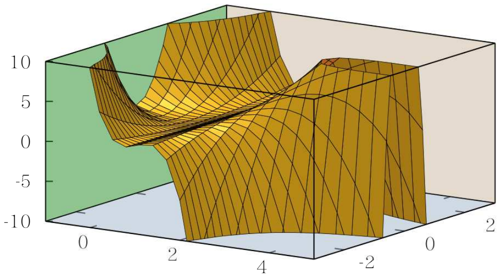
以下参照: **set vxrange (p. 300)**。

Walls

書式:

```
set walls
set wall {x0|y0|z0|x1|y1} {<fillstyle>} {fc <fillcolor>}
```

splot が描く 3 次元曲面は、x, y, z 軸の範囲とは無関係に正規化された単位立方体の中に置きます。そしてこの立方体の境界壁は、グラフ座標の x == 0 や x == 1 等の平面となります。コマンド **set walls** は、これらの壁を x0, y0, z0 を、単色塗りの曲面として描画します。デフォルトではこれらの壁は、半透明 (fillstyle transparent solid 0.5) とします。このコマンドでどの壁を描画するか、そして個別の色や個別の塗り潰しスタイル (fillstyle) をカスタマイズできます。壁の描画を有効にする場合は、**set xyplane 0** も設定するといいでしょう。例:



```
set wall x0; set wall y1; set wall z0 fillstyle solid 1.0 fillcolor "gray"
splot f(x,y) with pm3d fc "goldenrod"
```

Watchpoints

コマンド plot の各描画要素毎に、一つ以上のウォッチポイントを設定できます。コマンド **show watchpoints** で、すべてのウォッチポイント対象と直前の plot コマンドでヒットした点の要約を見ることができます。

例:

```
plot DATA using 1:2 smooth cnormal watch y=0.25 watch y=0.5 \
     watch y=0.75
show watchpoints

    Plot title:     "DATA using 1:2 smooth cnormal"
      Watch 1 target y = 0.25         (1 hits)
              hit 1   x 50.6  y 0.25
      Watch 2 target y = 0.5          (1 hits)
              hit 1   x 63.6  y 0.5
      Watch 3 target y = 0.75         (1 hits)
              hit 1   x 68.3  y 0.75
```

最初のウォッチポイント (y=0.25) を満たすすべての点の座標を配列 WATCH_1 に保存します。y=0.5 を満たす点は配列 WATCH_2 に保存し、以下同様です。

各ヒット点は、x 座標を実数部分、y 座標を虚数部分とする複素数として保存し、よってウォッチポイント 2 の最初のヒット点は、x = real(WATCH_2[1]), y = imag(WATCH_2[1]) となります。この例では、ヒット点の x 座標のみが関心事で、y 座標は常に対象となる y の値に一致します。しかし、ウォッチポイント対象が関数 f(x,y) の z 値である場合、x, y 座標はどちらも事前にはわかりません。

X2data

コマンド **set x2data** は x2 (上) 軸のデータを時系列 (日時) 形式に設定します。詳細は、以下参照: **set xdata (p. 302)**。

X2dtics

コマンド **set x2dtics** は x2 (上) 軸の目盛りを曜日に変更します。詳細は、以下参照: **set xdtics (p. 303)**。

X2label

コマンド **set x2label** は x2 (上) 軸の見出しを設定します。詳細は、以下参照: **set xlabel (p. 303)**。

X2mtics

コマンド **set x2mtics** は、x2 (上) 軸を 1 年の各月に設定します。詳細は、以下参照: **set xmtics (p. 304)**。

X2range

コマンド **set x2range** は x2 (上) 軸の表示される水平範囲を設定します。コマンドオプションのすべての説明については、以下参照: **set xrange (p. 305)**。以下も参照: **set link (p. 239)**。

X2tics

コマンド **set x2tics** は x2 (上) 軸の、見出し付けされる大目盛りの制御を行ないます。詳細は、以下参照: **set xtics (p. 307)**。

X2zeroaxis

コマンド **set x2zeroaxis** は、原点を通る x2 (上) 軸 (y2 = 0) を描きます。詳細は、以下参照: **set zeroaxis (p. 315)**。

軸毎のデータ種類指定 (xdata)

このコマンドは x 軸のデータ形式の解釈を制御します。他の軸それぞれにも同様のコマンドが機能します。

書式:

```
set xdata {time}
show xdata
```

ydata, **zdata**, **x2data**, **y2data**, **cbdata** にも同じ書式が当てはまります。

time オプションはデータが秒単位の日時データであることを伝えます。gnuplot バージョン 6 は時刻をミリ秒の精度保存します。

キーワード **time** なしの **set xdata** は、データの解釈方法を通常の形式に戻します。

日時データ (time)

set xdata time は、x 座標がミリ秒精度の日時データであることを意味します。**set ydata time** という同様のコマンドもあります。

日時データの入力と出力の解釈には、別々の書式機構があります。ファイルからの入力データは、全体に通用する **timefmt** を使うか、または plot コマンド内で timecolumn() 関数を使って読み込みます。この入力機構は、軸の範囲 (range) を設定するときに時間の値を使用する際にも適用されます。以下参照: **set timefmt (p. 295)**, **timecolumn (p. 56)**。

例:

```
set xdata time
set timefmt "%d-%b-%Y"
set xrange ["01-Jan-2013" : "31-Dec-2014"]
plot DATA using 1:2
```

または

```
plot DATA using (timecolumn(1,"%d-%b-%Y")):2
```

出力、すなわち軸に沿った目盛りのラベルや、マウス操作での座標出力については、デフォルトでは、秒での内部時刻から日時を表現する文字列への変換には、関数 'strftime' (unix でそれを調べるには "man strftime" とタイプしてください) を使います。**gnuplot** はこれを適当に意味のある書式で表示しようとしますが、**set format x** か **set xtics format** のいずれかを使ってカスタマイズすることもできます。特別な時間書式指定子に関しては、以下参照: **time_specifiers (p. 221)**。他の情報については、以下も参照: **time/date (p. 85)**。

曜日軸目盛り (xdtics)

コマンド **set xdtics** は x 軸の目盛りの刻みを曜日に変換します (0=Sun, 6=Sat)。6 を越える場合は 7 による余りが使われます。**unset xdtics** はその見出しをデフォルトの形式に戻します。他の軸にも同じことを行なう同様のコマンドが用意されています。

書式:

```
set xdtics
unset xdtics
show xdtics
```

ydtics, **zdtics**, **x2dtics**, **y2dtics**, **cbdtics** にも同じ書式が当てはまります。

以下も参照: **set format (p. 219)**。

軸ラベル (xlabel)

コマンド **set xlabel** は x 軸の見出しを設定します。他の軸にも見出しを設定する同様のコマンドがあります。

書式:

```
set xlabel {"<label>"} {offset <offset>} {font "<font>{,<size>}"}
           {textcolor <colorspec>} {{no}enhanced}
           {rotate by <degrees> | rotate parallel | norotate}
show xlabel
```

同じ書式が **x2label**, **ylabel**, **y2label**, **zlabel**, **cblabel** にも適用されます。

<offset> を x,y かまたは x,y,z の形式で指定した場合は、見出しは与えられた値だけ移動されます。それに座標系を選択して、その前に **first**, **second**, **graph**, **screen**, **character** のいずれかをつけることもできます。詳細は、以下参照: **coordinates (p. 41)**。デフォルトでは **character** 座標系が使われます。例えば、"**set xlabel offset -1,0**" は見出しの x 方向の位置のみ変更し、大ざっぱに言って 1 文字分の幅だけ見出しを左にずらします。1 文字の大きさは、フォントと出力形式の両方に依存します。

<font> は見出しが書かれるフォントを指定するのに使われます。フォントの <size> (大きさ) の単位は、どんな出力形式を使うかに依存します。

noenhanced は、拡張文字列処理 (enhanced text) モードが有効になっている場合でも、ラベル文字列を拡張文字列処理させないようにします。

見出しを消去するには、オプションをつけずに実行します。例: "**set y2label**"

軸の見出しのデフォルトの位置は以下の通りです:

xlabel: x 軸の見出しはグラフの下の真中

ylabel: y 軸の見出しはグラフの左の真中で、水平方向に書かれるか垂直方向になるかは出力形式依存。グラフの左側に回転させない ylabel 文字列を置くには、十分なスペースがない場合もありえます。その場合は、**set lmargin** で揃えられます。

zlabel: z 軸の見出しは軸の表示範囲より上で、見出しの真中が z 軸の真上

cblabel: 色見本 (color box) の軸の見出しは箱に沿って中央揃えされ、箱の向きが水平なら下に、垂直なら右に

y2label: y2 軸の見出しは y2 軸の右。その位置は、出力形式依存で y 軸と同様の規則で決定。

x2label: x2 軸の見出しはグラフの上で、タイトルよりは下。これは、 改行文字を使えば、それによる複数の行からなる描画タイトルで x2 軸の見出しを生成することも可能。例:

```
set title "This is the title\n\nThis is the x2label"
```

これは二重引用符を使うべきであることに注意してください。この場合、もちろん 2 つの行で同じフォントが使われます。

2 次元描画の場合の x, x2, y, y2 軸のラベルの方向 (回転角) は、**rotate by** < 角度 > を指定することで変更できます。3 次元描画の x, y 軸のラベルの方向はデフォルトでは水平方向になっていますが、**rotate parallel** を指定することで軸に平行にすることができます。

もし軸の位置のデフォルトの位置が気に入らないならば、代わりに **set label** を使ってください。このコマンドは文字列をどこに配置するかをもっと自由に制御できます。

バックスラッシュ文字列の作用、及び文字列を囲む単一引用符と二重引用符の違いに関するより詳しい情報については、以下参照: **syntax (p. 84)**。

月軸目盛り (xmtics)

コマンド **set xmtics** は x 軸の目盛りの見出しを月に変換します。1=Jan (1 月)、12=Dec (12 月) となります。12 を越えた数字は、12 で割ったあまりの月に変換されます。**unset xmtics** で目盛りはデフォルトの見出しに戻ります。他の軸に対しても同じ役割をする同様のコマンドが用意されています。

書式:

```
set xmtics
unset xmtics
show xmtics
```

x2mtics, **ymtics**, **y2mtics**, **zmtics**, **cbmtics** にも同じ書式が適用されます。

以下も参照: **set format (p. 219)**。

軸範囲指定 (xrange)

コマンド **set xrange** は表示される水平方向の範囲を指定します。他の軸にも同様のコマンドが存在しますし、極座標での動径 r, 媒介変数 t, u, v にも存在します。

書式:

```
set xrange [{{<min>}:{<max>}}] {{no}reverse} {{no}writeback} {{no}extend}
           | restore
show xrange
```

ここで <min> と <max> は定数、数式、または '*' で、'*' は自動縮尺機能を意味します。日時データの場合、範囲は **set timefmt** の書式に従った文字列を引用符で囲む必要があります。<min> や <max> を省略した場合は、現在の値を変更しません。自動縮尺機能に関する詳細は下に述べます。以下も参照: **noextend (p. 197)**。

yrange, **zrange**, **x2range**, **y2range**, **cbrange**, **rrange**, **trange**, **urange**, **vrange** は同じ書式を使用します。

x と x2 軸、あるいは y と y2 軸の範囲をリンクするオプションについては以下参照: **set link (p. 239)**。

オプション **reverse** は、自動縮尺の軸の方向を逆にします。例えば、データ値の範囲が 10 から 100 であるとき、これは、set xrange [100:10] としたのと同じように自動縮尺します。**reverse** は、自動縮尺ではない軸に対しては機能しません。

自動縮尺機能: <min> (同様のことが <max> にも適用されます) がアスタリスク "*" の場合は自動縮尺機能がオンになります。その場合のその値に、下限 <lb>、または上限 <ub>、またはその両方の制限を与えられます。書式は以下の通りです。

```
{ <lb> < } * { < <ub> }
```

例えば

```
0 < * < 200
```

は <lb> = 0, <ub> = 200 となります。そのような設定では、<min> は自動縮尺されますが、その最終的な値は 0 から 200 の間になります (記号は '<' ですが両端の値も含みます)。下限か上限を指定しない場合は、その '<' も省略できます。<ub> が <lb> より小さい場合は、制限はオフになり、完全な自動縮尺になります。この機能は、自動縮尺だけれども範囲に制限がある測定データの描画や、外れ値のクリッピング、またはデータがそれほどの範囲を必要としていなくても最小の描画範囲を保証するのに有用でしょう。

オプション **writeback** は、**set xrange** で占められているバッファの中に自動縮尺機能により作られた範囲を保存します。これは、いくつかの関数を同時に表示し、しかしその範囲はそのうちのいくつかのものから決定させたい場合に便利です。**writeback** の作用は、**plot** の実行中に機能するので、そのコマンドの前に指定する必要があります。最後に保存した水平方向の範囲は **set xrange restore** で復元できます。例を上げます。

```
set xrange [-10:10]
set yrange [] writeback
plot sin(x)
set yrange restore
replot x/2
```

この場合、y の範囲 (yrange) は sin(x) の値域として作られた [-1:1] の方になり、x/2 の値域 [-5:5] は無視されます。上記のそれぞれのコマンドの後に **show yrange** を実行すれば、上で何が行なわれているかを理解する助けになるでしょう。

2 次元描画において、**xrange** と **yrange** は軸の範囲を決定し、**trange** は、媒介変数モードの媒介変数の範囲、あるいは極座標モードの角度の範囲を決定します。同様に 3 次元媒介変数モードでは、**xrange**, **yrange**, **zrange** が軸の範囲を管理し、**urange** と **vrange** が媒介変数の範囲を管理します。

極座標モードでは、**rrange** は描画される動径の範囲を決定します。<rmin> は動径への追加の定数として作用し、一方 <rmax> は動径を切り捨てる (clip) ように作用し、<rmax> を越えた動径に対する点は描画されません。**xrange** と **yrange** は影響されます。これらの範囲は、グラフが r(t)-rmin のグラフで、目盛りの見出しにはそれぞれ rmin を加えたようなものであるかのようにセットされます。

全ての範囲は部分的に、または全体的に自動縮尺されますが、データの描画でなければ、パラメータ変数の自動縮尺機能は意味がないでしょう。

範囲は **plot** のコマンドライン上でも指定できます。コマンドライン上で与えられた範囲は単にその **plot** コマンドでだけ使われ、**set** コマンドで設定された範囲はその後の描画で、コマンドラインで範囲を指定していないもの全てで使われます。これは **splot** も同じです。

例 (examples)

例:

x の範囲をデフォルトの値にします:

```
set xrange [-10:10]
```

y の範囲が下方へ増加するようにします:

```
set yrange [10:-10]
```

z の最小値には影響を与えずに (自動縮尺されたまま)、最大値のみ 10 に設定します:

```
set zrange [:10]
```

x の最小値は自動縮尺とし、最大値は変更しません:

```
set xrange [*:]
```

x の最小値を自動縮尺としますが、その最小値は 0 以上にします。

```
set xrange [0<*:]
```

x の範囲を自動縮尺としますが、小さくても 10 から 50 の範囲を保持します (実際はそれより大きくなるでしょう):

```
set xrange [*<10:50<*]
```

自動縮尺で最大範囲を -1000 から 1000、すなわち [-1000:1000] 内で自動縮尺します:

```
set xrange [-1000<*:*<1000]
```

x の最小値を -200 から 100 の間のどこかにします:

```
set xrange [-200<*<100:]
```

Extend

set xrange noextend は、**set autoscale x noextend** と全く同じです。以下参照: **noextend** (p. 197)。

軸主目盛り指定 (xtics)

x 軸の (見出しのつく) 大目盛りは コマンド **set xtics** で制御できます。目盛りは **unset xtics** で消え、**set xtics** で (デフォルトの状態の) 目盛りがつきます。y,z,x2,y2 軸の大目盛りの制御を行なう同様のコマンドがあります。

書式:

```
set xtics {axis | border} {{no}mirror}
          {in | out} {scale {default | <major> {,<minor>}}}
          {{no}rotate {by <ang>}} {offset <offset> | nooffset}
          {left | right | center | autojustify}
          {add}
          {  autofreq
           | <incr>
           | <start>, <incr> {,<end>}
           | ({"<label>"} <pos> {<level>} {,{"<label>"}...) }
          {format "formatstring"} {font "name{,<size>}"} {{no}enhanced}
          { numeric | timedate | geographic }
          {{no}logscale}
          { rangelimited }
          { textcolor <colorspec> }
unset xtics
show xtics
```

同じ書式が **ytics**, **ztics**, **x2tics**, **y2tics**, **cbtics** にも適用されます。

axis と **border** は **gnuplot** に目盛り (目盛りの刻み自身とその見出し) を、それぞれ軸につけるのか、境界につけるのかを指示します。軸が境界にとても近い場合、**axis** を使用すると目盛りの見出し文字を境界の外に出してしまうでしょう。この場合自動的なレイアウトアルゴリズムによる余白設定は大抵よくないものとなってしまいます。

mirror は **gnuplot** に反対側の境界の同じ位置に、見出しのない目盛りを出力するよう指示します。**nomirror** は、あなたが想像している通りのことを行ないます。

in と **out** は目盛りの刻みを内側に描くか外側に描くかを切り変えます。

目盛りの刻みのサイズは **scale** で調整できます。<minor> の指定が省略された場合は、それは 0.5*<major> になります。デフォルトのサイズは、大目盛りが 1.0 で小目盛りが 0.5 で、これは **scale default** で呼びだせます。

rotate は、文字列を 90 度回転させて出力させようとします。これは、文字列の回転をサポートしている出力ドライバ (terminal) では実行されます。**norotate** はこれをキャンセルします。**rotate by <ang>** は角度 <ang> の回転を行ないますが、これはいくつかの出力形式 (terminal) でサポートされています。

x と y 軸の大目盛りのデフォルトは **border mirror norotate** で、x2, y2 軸は **border nomirror norotate** がデフォルトです。z 軸には、**{axis | border}** オプションは無効で、デフォルトは **nomirror** です。z 軸の目盛りをミラー化したいなら、多分 **set border** でそのための空間をあける必要があるでしょう。

<offset> は x,y かまたは x,y,z の形式で指定しますが、それに座標系を選択して、その前に **first**, **second**, **graph**, **screen**, **character** のいずれかをつけることもできます。<offset> は刻み文字のデフォルトの位置からのずれを表し、デフォルトの座標系は **character** です。詳細は、以下参照: **coordinates (p. 41)**。**nooffset** はずらしを無効にします。

例:

xtics をより描画に近づける:

```
set xtics offset 0,graph 0.05
```

軸の目盛りとグラフ自身の描画の相対的な順番を変更するには、コマンド **set grid** に 'front', 'back', 'layerdefault' 等のオプションを使用してください。ただし、異なる軸の目盛りや格子線に異なるレイヤーを割り当てるオプションはありません。

デフォルトでは見出しラベルは、美しい結果を生むように、軸と回転角に依存した位置に自動的に揃えられますが、気にいらなければ、明示的に **left**, **right**, **center** のキーワードにより位置揃えを変更できます。**autojustify** でデフォルトの挙動に復帰します。

オプションなしで **set xtics** を実行すると、目盛りが表示される状態であれば、それはデフォルトの境界、または軸を復元し、そうでなければ何もしません。その前に指定した目盛りの間隔、位置 (と見出し) は保持されます。

目盛りの位置は、デフォルト、またはオプション **autofreq** が指定されていれば自動的に計算されます。

目盛りの位置の列は、目盛りの間隔のみ、または開始位置と間隔と終りの位置、のいずれかを指定することができます (以下参照: **xtics series (p. 308)**)。

明示的な位置のリストを与えることで、個々の目盛りの位置を個別に指定することもできます。各位置には、それに対する見出しラベルを指定することもできます。以下参照: **xtics list (p. 309)**。

しかし指定しても、表示されるのはあくまで描画範囲のものだけです。

目盛りの見出しの書式 (または省略) は **set format** で制御されます。ただしそれは **set xtics (<label>)** の形式の明示的な見出し文字列が含まれていない場合だけです。

(見出し付けされない) 小目盛りは、**set mxtics** コマンドで自動的に追加するか、または位置を手動で **set xtics ("" <pos> 1, ...)** の形式で与えることもできます。

刻みの見た目 (線種、幅等) は、それを軸の上に描く場合であっても、境界線によって決定されます (以下参照: **set border (p. 198)**)。

Xtics series

書式:

```
set xtics <incr>
set xtics <start>, <incr>, <end>
```

暗示的な <start>, <incr>, <end> 形式は、目盛りの列を <start> から <end> の間を <incr> の間隔で表示します。<end> を指定しなければ、それは無限大とみなされます。<incr> は負の値も可能です。<start>

と <end> の両方が指定されていない場合、<start> は -∞、<end> は +∞ とみなされ、目盛りは <incr> の整数倍の位置に表示されます。軸が対数軸の場合、目盛りの間隔 (増分) は、倍数として使用されます。

負の <start> や <incr> を、数値の後ろに指定すると (例えば **rotate by <angle>** とか **offset <offset>** の後ろ)、gnuplot の構文解析器は、その値からその負の <start> や <incr> の値の引き算を行おうとする間違いを犯します。これを回避するには、そのような場合は、**0-<start>** や **0-<incr>** のように指定してください。

例:

```
set xtics border offset 0,0.5 -5,1,5
```

最後の ',' のところで失敗します。代わりに

```
set xtics border offset 0,0.5 0-5,1,5
```

か

```
set xtics offset 0,0.5 border -5,1,5
```

のようにしてください。これらは、ちゃんと指示通りに、目盛りを境界に、目盛り見出し文字列を 0,0.5 文字分だけずらして、start, increment, end をそれぞれ -5,1,5 に設定します。

例:

目盛りを 0, 0.5, 1, 1.5, ..., 9.5, 10 の位置に生成

```
set xtics 0,.5,10
```

目盛りを ..., -10, -5, 0, 5, 10, ... に生成

```
set xtics 5
```

目盛りを 1, 100, 1e4, 1e6, 1e8 に生成

```
set logscale x; set xtics 1,100,1e8
```

Xtics list

書式:

```
set xtics {add} ("label1" <pos1> <level1>, "label2" <pos2> <level2>, ...)
```

明示的な ("label" <pos> <level>, ...) の形式は、任意の目盛りの位置、あるいは数字でない見出しの生成も可能にします。この形式では、目盛りは位置の数字の順に与える必要はありません。各目盛りは位置 (pos) と見出し (label) を持ちますが、見出しは必須ではありません。

見出しは引用符で囲んだ文字列か、または文字列値の数式です。それには、"%3f clients" のようにその位置を数字に変換する書式文字列を入れても構いませんし、空文字列 "" でも構いません。より詳しい情報については、以下参照: **set format (p. 219)**。もし、文字列を指定しなければ、デフォルトの数字の見出しを使用します。

明示的な形式では 3 つ目のパラメータとしてレベルを指定できます。デフォルトのレベルは 0 で、これは大目盛りを意味し、レベル 1 の場合は小目盛りを生成します。ラベルは、小目盛りには決して付きません。大目盛りと小目盛りは gnuplot が自動的に生成しますが、ユーザが明示的に指定もできます。レベルが 2 以上の目盛りは、ユーザが明示的に指定しなければならず、自動生成の目盛りよりも高い優先度を持ちます。各レベルの目盛りの刻みの大きさは、**set tics scale** で制御します。

例:

```
set xtics ("low" 0, "medium" 50, "high" 100)
set xtics (1,2,4,8,16,32,64,128,256,512,1024)
set ytics ("bottom" 0, "" 10, "top" 20)
set ytics ("bottom" 0, "" 10 1, "top" 20)
```

2 番目の例では、全ての目盛りが見出し付けされます。3 番目の例では、端のものだけが見出し付けされます。4 番目の例の、見出しのない目盛りは小目盛りになります。

通常明示的な (手動の) 目盛り位置が与えられた場合、自動的に生成される目盛りは使われません。逆に、**set xtics auto** のようなものが指定された場合は、以前に手動で設定した目盛りは消されてしまします。この手動の目盛りと自動的な目盛りを共存させるにはキーワード **add** を使用してください。これは追加する目盛りのスタイルの前に書かなければいけません。

例:

```
set xtics 0,.5,10
set xtics add ("Pi" 3.14159)
```

これは自動的に目盛りの刻みを x 軸に 0.5 間隔でつけますが、π のところに明示的な見出しも追加します。

Xtics time

時間と日付は内部では秒数として保持されています。

入力: 非数値の日時値は、入力時に **timefmt** で指定した書式を用いて秒数に変換します。軸の範囲、目盛りの配置、グラフの座標も **timefmt** で解釈される日時で、引用符で囲んで与えることができます。

出力: 軸の目盛りラベルは、**set format** か **set xtics format** のいずれかで指定された、別の書式を使って生成します。デフォルトでは、それは通常の数値書式指定であると認識しますが (**set xtics numeric**)、他に、地理座標 (**set xtics geographic**) や、日時データ (**set xtics time**) のオプションがあります。

注意: 以前の版の gnuplot との互換性のため、コマンド **set xdata time** も暗黙に **set xtics time** を実行しますし、**set xdata** や **unset xdata** は暗黙に **set xtics numeric** へリセットします。しかし、これはその後に **set xtics** を呼び出すことで変更できます。

例:

```
set xdata time           # 入力データの解釈の制御
set timefmt "%d/%m"      # 入力データの読み込みの書式
set xtics timedate       # 出力書式の解釈の制御
set xtics format "%b %d" # 目盛りラベルで使う書式
set xrange ["01/12":"06/12"]
set xtics "01/12", 172800, "05/12"

set xdata time
set timefmt "%d/%m"
set xtics format "%b %d" time
set xrange ["01/12":"06/12"]
set xtics ("01/12", "" "03/12", "05/12")
```

これらは両方とも "Dec 1", "Dec 3", "Dec 5", の目盛りを生成しますが、2 番目の例 "Dec 3" の目盛りは見出し付けされません。

<start>, <incr>, <end> 形式を使う場合、<incr> はデフォルトでは秒単位ですが、**minutes**, **hours**, **days**, **weeks**, **months**, **years** の明示的な時間の単位を後ろに追加することもできます。これは、間隔 <incr> のみを指定する場合も同じです。

例

```
set xtics time 5 years      # 5 年間隔で目盛りラベルを配置
set xtics "01-Jan-2000", 1 month, "01-Jan-2001"
```

小目盛用の特別な時刻モードもあります。以下参照: **set mxtics time (p. 250)**。

地理座標 (geographic)

set xtics geographic は、x 軸の値が度の単位の地理座標であることを意味します。その軸の刻みの見出しの表現の指定には、**set xtics format** か **set format x** を使います。地理座標データに関する書式指定子は以下の通り:

```
%D                      = 度の整数表示
%<width.precision>d     = 度の浮動小数表示
%M                      = 分の整数表示
%<width.precision>m     = 分の浮動小数表示
%S                      = 秒の整数表示
%<width.precision>s     = 秒の浮動小数表示
%E                      = +/- でなく E/W のラベル
%N                      = +/- でなく N/S のラベル
```

例えば、コマンド **set format x "%Ddeg %5.2mmin %E"** は、x 座標の -1.51 という値を " **1deg 30.60min W**" のように表示します。

xtics がデフォルトの状態のまま (**set xtics numeric**) の場合は、座標は 10 進数の度で表示し、**format** も上の特別な記号ではなく、通常の数値書式が使われているとみなされます。

マップ上にラベルを置くなど、軸の目盛りとは異なる場所で度/分/秒の出力をするには、strptime に相対的時間書式指定 %tH %tM %tS を使用できます。以下参照: **time_specifiers (p. 221)**, **strptime (p. 50)**。

Xtics logscale

対数軸に沿う刻みに対して **logscale** 属性を設定すると、刻みの間隔は公差ではなく、公比と解釈されます。例:

```
# y=20 y=200 y=2000 y=20000 に刻み列を作成
set log y
set ytics 20, 10, 50000 logscale
```

y=50000 は 2*10^x の数列には含まれないので、そこには刻みはつかないことに注意してください。logscale 属性が無効の場合、軸の増分は、たとえ対数軸であっても公差として扱われます。例:

```
# y=20 y=40 y=60 ... y=200 に刻みを作成
set log y
set yrange [20:200]
set ytics 20 nologscale
```

logscale 属性は、コマンド **set log** で自動的に設定されるので、2 つ目の例のような軸の刻み間隔に強制的にしたい場合でなければ、通常はこのキーワードは必要ありません。

Xtics rangelimited

このオプションは、自動的に生成される軸の目盛りの見出しと、描画されたデータで実際に与えられる範囲に対応する描画境界の両方を制限します。これは描画に対する現在の範囲制限とは無関係であることに注意してください。例えばデータ "file.dat" のデータがすべて $2 < y < 4$ の範囲にあるとすると、以下のコマンドは、左側の描画境界 (y 軸) は y の範囲全体 ([0:10]) のこの部分 ([2:4]) のみが描画され、この範囲 ([2:4]) の軸の目盛りのみが作られる描画を生成します。つまり、描画は y の範囲全体 ([0:10]) に拡大されますが、左の境界は 0 から 2 の間、4 から 10 の間は空白領域となります。このスタイルは、範囲枠 グラフ (range-frame) とも呼ばれます。

```
set border 3
set yrange [0:10]
set ytics nomirror rangelimited
plot "file.dat"
```

Xy 平面位置 (xyplane)

set xyplane コマンドは 3D 描画で描かれる xy 平面の位置を調整するのに使われます。後方互換性のために、"set ticslevel" も同じ意味のコマンドとして使うことができます。

書式:

```
set xyplane at <zvalue>
set xyplane relative <frac>
set ticslevel <frac>        # set xyplane relative と同等
show xyplane
```

set xyplane relative <frac> は、xy 平面を Z 軸の範囲のどこに置くかを決定します。<frac> には、xy 平面と z の一番下の位置との差の、z 軸の範囲全体に対する割合を与えます。デフォルトの値は 0.5 です。負の値も許されていますが、そうすると 3 つの軸の目盛りの見出しが重なる可能性があります。

もう一つの形式である **set xyplane at <zvalue>** は、現在の z の範囲を気にすることなく、指定した z の値の位置に xy 平面を固定します。よって、x,y,z 軸を共通の原点を通るようにするには、**set xyplane at 0** とすればいいことになります。

以下も参照: **set view (p. 298)**, **set zeroaxis (p. 315)**。

Xzeroaxis

コマンド **set xzeroaxis** は $y = 0$ の直線を描きます。詳細に関しては、以下参照: **set zeroaxis** (p. **315**)。

Y2data

コマンド **set y2data** は y2 (右) 軸のデータを時系列 (日時) 形式に設定します。詳細は、以下参照: **set xdata** (p. **302**)。

Y2dtics

コマンド **set y2dtics** は y2 (右) 軸の目盛りを曜日に変更します。詳細は、以下参照: **set xdtics** (p. **303**)。

Y2label

コマンド **set y2label** は y2 (右) 軸の見出しを設定します。詳細は、以下参照: **set xlabel** (p. **303**)。

Y2mtics

コマンド **set y2mtics** は y2 (右) 軸の目盛りを 1 年の各月に変更します。詳細は、以下参照: **set xmtics** (p. **304**)。

Y2range

コマンド **set y2range** は y2 (右) 軸の表示される垂直範囲を設定します。コマンドオプションのすべての説明については、以下参照: **set xrange** (p. **305**)。以下も参照: **set link** (p. **239**)。

Y2tics

コマンド **set y2tics** は y2 (右) 軸の、見出し付けされる大目盛りの制御を行ないます。詳細は、以下参照: **set xtics** (p. **307**)。

Y2zeroaxis

コマンド **set y2zeroaxis** は、原点を通る y2 (右) 軸 ($x2 = 0$) を描きます。詳細は、以下参照: **set zeroaxis** (p. **315**)。

Ydata

コマンド **set ydata** は y 軸のデータを時系列 (日時) 形式に設定します。以下参照: **set xdata** (p. **302**)。

Ydtics

コマンド **set ydtics** は y 軸の目盛りを曜日に変更します。詳細は、以下参照: **set xdtics** (p. **303**)。

Ylabel

このコマンドは y 軸の見出しを設定します。以下参照: **set xlabel** (p. **303**)。

Ymtics

コマンド **set ymtics** は、y 軸の目盛りを月に変更します。詳細は、以下参照: **set xmtics** (p. **304**)。

Yrange

コマンド **set yrange** は、y 方向の垂直範囲を設定します。詳細は、以下参照: **set xrange** (p. **305**)。

Ytics

コマンド **set ytics** は y 軸の (見出し付けされる) 大目盛りを制御します。詳細は、以下参照: **set xtics** (p. **307**)。

Yzeroaxis

コマンド **set yzeroaxis** は $x = 0$ の直線 (y 軸) を書きます。詳細は、以下参照: **set zeroaxis** (p. **315**)。

Zdata

コマンド **set zdata** は z 軸のデータを時系列 (日時) 形式に設定します。以下参照: **set xdata** (p. **302**)。

Zdtics

コマンド **set zdtics** は z 軸の目盛りを曜日に変更します。詳細は、以下参照: **set xdtics** (p. **303**)。

Zzeroaxis

コマンド **set zzeroaxis** は (x=0,y=0) を通る直線を描きます。これは、2D 描画、および **set view map** での splot では効力を持ちません。詳細は、以下参照: **set zeroaxis (p. 315)**, **set xyplane (p. 312)**。

Cbdata

このコマンドはカラーボックス軸のデータを時系列 (日時) 形式に式に設定します。以下参照: **set xdata (p. 302)**。

Cbdtics

コマンド **cbdtics** はカラーボックス軸の目盛りの刻みを曜日に変換します。詳細は、以下参照: **set xdtics (p. 303)**。

ゼロ閾値 (zero)

zero の値は、0.0 に近いデフォルトの閾値を表します。

書式:

```
set zero <expression>
show zero
```

gnuplot は、(複素数値を持つ点の描画においては) その値の虚数部分の絶対値が **zero** 閾値より大きい場合 (つまり実数でない値を持つ点) は、その点を描画しません。この閾値は **gnuplot** の他の様々な部分においてその (大まかな) 数値誤差の閾値としても使われています。デフォルトの **zero** の値は 1e-8 です。1e-3 (= 典型的なビットマップディスプレイの解像度の逆数) より大きい **zero** の値は設定すべきではないでしょうが、**zero** を 0.0 と設定するのは意味のないことではありません。

ゼロ軸 (zeroaxis)

x 軸は **set xzeroaxis** によって描かれ、**unset xzeroaxis** によって削除されます。同様の y, x2, y2, z 軸用のコマンドが同様の働きをします。**set zeroaxis ...** (前置詞なし) は、x, y, z 軸すべてに機能します。

書式:

```
set {x|x2|y|y2|z}zeroaxis { {linestyle | ls <line_style>}
                           | {linetype | lt <line_type>}
                             {linewidth | lw <line_width>}
                             {linecolor | lc <colorspec>}
                             {dashtype | dt <dashtype>} }
unset {x|x2|y|y2|z}zeroaxis
show {x|y|z}zeroaxis
```

デフォルトでは、これらのオプションはオフになっています。選択された 0 の軸は <line_type> の線の型、<line_width> の線の幅、<colorspec> の色、<dashtype> の点線/破線パターンで (いずれも現在使用している出力形式がサポートしていれば)、あるいはあらかじめ定義された <line_style> のスタイルで描かれます。以下参照: **set style line (p. 285)**。

線の型を指定しなければ、軸は通常の軸の線の型 (型 0) で描かれます。

例:

y=0 の軸を見えるように簡単に書く場合:

```
set xzeroaxis
```

太い線にして、違った色、または点線パターンにしたい場合:

```
set xzeroaxis linetype 3 linewidth 2.5
```

Zlabel

このコマンドは z 軸の見出しを設定します。以下参照: **set xlabel (p. 303)**。

Zmtics

コマンド **set zmtics** は z 軸の目盛りを月に変更します。詳細は、以下参照: **set xmtics (p. 304)**。

Zrange

コマンド **set zrange** は z 軸方向に表示される範囲を設定します。このコマンドは **splot** にのみ有効で、**plot** では無視されます。詳細は、以下参照: **set xrange (p. 305)**。

Ztics

コマンド **set ztics** は z 軸の (見出し付けされる) 大目盛りを制御します。詳細は、以下参照: **set xtics (p. 307)**。

Cblabel

このコマンドはカラーボックス軸の見出しを設定します。以下参照: **set xlabel (p. 303)**。

Cbmtics

コマンド **set cbmtics** はカラーボックス軸の目盛りの見出しを月に変換します。詳細は、以下参照: **set xmtics (p. 304)**。

Cbrange

コマンド **set cbrange** は、スタイル **with pm3d**, **with image** や **with palette** などによって現在のパレット (**palette**) を使って色付けされる値の範囲を設定します。その範囲外の値に対しては、最も近い限界の値の色が使用されます。

カラーボックス軸 (cb-軸) が **splot** で自動縮尺されている場合は、そのカラーボックスの範囲は **zrange** が使われます。**splot ... pm3d**|**palette** で描画される点は、異なる **zrange** と **cbrange** を使うことでフィルタリングできます。

set cbrange の書式に関する詳細は、以下参照: **set xrange (p. 305)**。以下も参照: **set palette (p. 258)**, **set colorbox (p. 206)**。

Cbtics

コマンド **set cbtics** はカラーボックス軸の (見出し付けされる) 大目盛りを制御します。詳細は、以下参照: **set xtics (p. 307)**。

シェルコマンド (shell)

shell コマンドは対話的なシェルを起動します。**gnuplot** に戻るには、Unix ならば **exit** もしくは END-OF-FILE 文字を、MS-DOS か OS/2 ならば **exit** を入力して下さい。

コマンド **shell** は、それ以外の gnuplot コマンドライン上のものをすべて無視します。そうでなく、シェルに直ちにコマンド文字列を渡したい場合は、関数 **system** か、ショートカット **!** を使用してください。以下参照: **system (p. 330)**。

例:

```
shell
system "print previous_plot.ps"
! print previous_plot.ps
current_time = system("date")
```

Show

ほとんどの **set** コマンドには、それに対応する、特別なオプションを持たない show コマンドがあります。例:

```
show linetype 3
```

は、以下のように、前のコマンドから現在有効な属性を報告します:

```
set linetype 3 linewidth 2 dashpattern '.-'
```

この形式とは離れる 2, 3 の **show** コマンドについては別に説明します。

Show colornames

gnuplot は 100 程度の色名を持っています (以下参照: **colornames (p. 207)**)。コマンド **show colornames** を使えば、その出力形式に対する色名の一覧を出力できます。新しい色名を設定する方法は今はありません。

Show functions

show functions コマンドはユーザーが定義した関数とその定義内容を表示します。

書式:

```
show functions
```

gnuplot における関数の定義とその使い方については、以下参照: **expressions (p. 45)**。以下も参照ユーザ定義関数でのスプライン (`spline.dem`)

および関数と複素変数を翼に使用 (`airfoil.dem`)。

Show palette

書式:

```
show palette
show palette palette {<ncolors>} {{float | int | hex}}
show palette gradient
show palette rgbformulae
test palette
```

コマンド **test palette** は、現在のパレットの R,G,B 成分の対応状態 (profile) を描画し、その値をデータブロック $PALETTE に残させます。

Show palette gradient

show palette gradient は、コマンド **set palette defined** で前に定義した部分的なグラデーションを表示します。現在のパレットが、rgbformulae に基づくか、または前に定義した値の集合に基づく場合は、このコマンドは何もしません。

Show palette palette

```
show palette palette {<ncolors>} {{float | int | hex}}
```

show palette palette <n> は、現在のパレットの各エントリの色成分の表を、画面、または **set print** で指定されたファイルに書き出します。デフォルトでは、連続なパレットを 128 区切りに標本化します。<ncolors> を指定すると、パレットを (128 でなく) その数の区切りに均等に標本化します。デフォルトは、以下の長形式での一覧表示です:

```
0. gray=0.0000, (r,g,b)=(0.0000,0.0000,0.0000), #000000 =   0   0   0
1. gray=0.1111, (r,g,b)=(0.3333,0.0014,0.6428), #5500a4 =  85   0 164
2. gray=0.2222, (r,g,b)=(0.4714,0.0110,0.9848), #7803fb = 120   3 251
...
```

後にオプションキーワード **float**, **int**, **hex** をつけると、代わりに各エントリに一つの色成分表現のみを表示します。

```
  int:       85       0     164
float:   0.3333  0.0014  0.6428
  hex: 0x5500a4
```

set print を使ってこの出力をファイルに書き出すことで、gnuplot の現在のカラーパレットを Octave のような他の画像処理アプリケーションに取り込むことができます。

set print を使ってこの出力をデータブロックに書き出すことで、現在のパレットを保存できますが、それにより有効なパレットを再定義した後でも、後の plot コマンドで保存したパレットを使えるようになります。これは、カラーボックスがただ現在有効なパレットだけを表現する一方で、複数のパレットで描画するグラフを作成することを可能にします。

Show palette rgbformulae

show palette rgbformulae は、定義済で利用できる、灰色階調値からカラーへの変換公式を表示します。これは、現在のパレットの状態を表示「してはいません。」

Show plot

コマンド **show plot** は直前の描画コマンド、すなわち **replot** コマンドで再現される、直前に行われた **plot** や **splot** コマンドを表示します。

さらにコマンド **show plot add2history** は、この現在の描画コマンドを **history** に書き出します。これは、**replot** を使って直前の描画コマンドに曲線を追加した場合、そしてコマンド行全体をすぐに編集したい場合に便利です。

Show variables

show variables コマンドはユーザ定義変数と内部変数の現在の値の一覧を表示します。gnuplot は、GPVAL_, MOUSE_, FIT_, TERM_ で始まる名前を持つ変数を内部で定義しています。

書式:

```
show variables      # GPVAL_ で始まるもの以外の変数を表示
show variables all  # GPVAL_ で始まるものも含め全ての変数を表示
show variables NAME # NAME で始まる変数のみを表示
```

Splot

splot は 3 次元描画のためのコマンドです (もちろんご存知でしょうが、実際にはその 2 次元への射影)。それは、**plot** コマンドの 3 次元版です。**splot** は、それぞれ単一の x, y, z 軸を提供するだけで、**plot** で用意されている第 2 軸 x2, y2 のようなものはありません。

2 次元と 3 次元描画の両方で使える多くのオプションについては、以下参照: **plot (p. 155)**。

書式:

```
splot {<ranges>}
      {<iteration>}
      <function> | {{<file name> | <datablock name>}
                     {datafile-modifiers}}
                 | <voxelgridname>
                 | keyentry
      {<title-spec>} {with <style>}
      {, {definitions{,}} <function> ...}
```

コマンド **splot** は、関数から生成されたデータ、またはデータファイルから読み込んだデータ、または事前に保存された名前付きデータブロックのデータを処理します。データファイル名は、通常引用符で囲んだ文字列として与えます。関数は 1 本の数式ですが、媒介変数モード (parametric) では 3 つの数式の組として与えます。

バージョン 5.4 以降、**splot** はボクセルデータの操作ができます。以下参照: **voxel-grids (p. 326)**, **set vgrid (p. 298)**, **vxrange (p. 300)**。有効なボクセル格子データは、スタイル **with dots**, **with points**, **with isosurface** のいずれかを使って描画できます。ボクセル格子データ値は、これら以外の描画スタイルでも **using** 指定を使えば参照でき、例えば色を割り当てるなどに利用できます。

デフォルトでは、**splot** は描画されるデータの下に完全な xy 面を描きます。z の一番下の目盛りと xy 平面の位置関係は **set xyplane** で変更できます。**splot** の射影の向きは **set view** で制御できます。詳細は、以下参照: **set view (p. 298)**, **set xyplane (p. 312)**。

splot コマンドの範囲の指定の書式は **plot** の場合と同じです。媒介変数モード (parametric) でなければ、範囲指定は以下の順で、

```
splot [<xrange>][<yrange>][<zrange>] ...
```

媒介変数モード (parametric) では、範囲指定は以下の順で与えなければいけません:

```
splot [<urange>][<vrange>][<xrange>][<yrange>][<zrange>] ...
```

title オプションも **plot** と同じです。**with** も **plot** とほぼ同じですが、2 次元の描画スタイル全部が使えるわけではありません。

datafile オプションにはさらに違いがあります。

媒介変数モード (parametric) や関数を利用して曲面を描く別の方法に、疑似ファイル '++' を利用して xy 平面の格子の上に標本点を生成するやり方があります。

以下も参照: **show plot (p. 319)**, **set view map (p. 298)**, **sampling (p. 182)**。

データファイル (datafile)

plot と同じように、**splot** でファイルからグラフを生成できます。

書式:

```
splot '<file_name>' {binary <binary list>}
                      {{nonuniform|sparse} matrix}
                    {index <index list>} {every <every list>}
                    {using <using list>}
                    {smooth <option>} {if (<expression>)}
```

"" や "-" といった特別なファイル名も **plot** と同様に許されます。以下参照: **special-filenames (p. 174)**。

キーワード **binary** や **matrix** はそのデータが特別な形であることを、**index** は多重データ集合ファイルからどのデータ集合を選んで描画するかを、**every** は各データ集合からどの部分行を選んで描画するかを、**using** は各データ行からどのように列を選ぶかを指定します。

index と **every** オプションは **plot** の場合と同じように振舞います。**using** も、**using** のリストが 2 つでなく 3 つ必要であるということを除いては同様です。

plot のオプションである **smooth** は **splot** では利用できません。しかし、**cntrparam** や **dgrid3d** が、制限されてはいますが平滑化のために用意されています。

データファイルの形式は、各点が (x,y,z) の 3 つ組である以外は、本質的に **plot** と同じです。もし一つの値だけが与えられれば、それは z として使われ、ブロック番号が y として、そして x はそのブロック内での番号が使われます。もし 2 つ、あるいは 4 つの値が与えられれば、**gnuplot** はその最後の値を pm3d plot での色の計算に使います。3 つの値は (x,y,z) の組と見なされます。他に値があれば、それは一般に誤差と見なされます。それは **fit** で使うことが可能です。

splot のデータファイルでは、1 行の空行はデータのブロック分離子です。**splot** は個々のブロックを、関数の y-孤立線と同じものとして扱います。1 行の空行で分離されている点同士は線分で結ばれることはありません。全てのブロックが全く同じ点の数を持つ場合、**gnuplot** はブロックを横断し、各ブロックの対応する点同士を結ぶ孤立線を描きます。これは "grid data" と呼ばれ、曲面の描画、等高線の描画 (**set contour**)、隠線処理 (**set hidden3d**) では、この形のデータであることが必要となります。以下も参照: **splot grid_data (p. 325)**。

Matrix

テキストファイルやバイナリファイルから、matrix データを色々な形式 (**uniform**, **nonuniform**, **sparse**) で入力することができます。

1 つ目の形式は、x, y の座標が一様 (uniform) であると仮定して、その値をこの一様な格子の matrix のそれぞれの要素 M[i,j] に割り当てる方法です。割り当てられる x 座標は [0:NCOLS-1] の範囲の整数です。割り当てられる y 座標は [0:NROWS-1] の範囲の整数です。これは、テキストデータに対してはデフォルトですが、バイナリデータに対してはそうではありません。例や追加キーワードについては以下参照: **matrix uniform (p. 322)**。

2 つ目の形式は、x, y 座標が明示された非一様 (nonuniform) な格子で、入力データの最初の行を y 座標、最初の列を x 座標とみなします。バイナリ入力データでは、1 行目の最初の要素は、列数でなければいけません。**binary matrix** 入力ではこれがデフォルトですが、テキスト入力データに対しては追加キーワード **nonuniform** が必要になります。例に関しては以下参照: **nonuniform (p. 322)**。

sparse matrix の形式は一様な格子を定義しますが、その格子内の任意の個数のそれぞれの点の値を、入力ファイルから、任意の順の一行に一つのデータとして、読み取ります。これは、主に不完全なデータから温度分布図 (heatmap) を生成する目的のものです。例に関しては以下参照: **sparse (p. 323)**。

Uniform matrix 一様 (uniform) な matrix データを描画するコマンドの例:

```
splot 'file' matrix using 1:2:3          # テキストデータ
splot 'file' binary general using 1:2:3  # バイナリデータ
```

一様な格子の matrix データでは、各ブロックの z の値は一行で一度に読まれます。すなわち、

```
z11 z12 z13 z14 ...
z21 z22 z23 z24 ...
z31 z32 z33 z34 ...
```

等。

テキストデータに対しては、1 行目がデータでなく列ラベルを持つ場合、追加キーワード **columnheaders** を使ってください。同様に、各行の最初の要素がデータでなくラベルである場合は、追加キーワード **rowheaders** を使用してください。以下は、その両方を使用する例です:

```
$DATA << EOD
xxx A   B   C   D
aa  z11 z12 z13 z14
bb  z21 z22 z23 z24
cc  z31 z32 z33 z34
EOD
plot $DATA matrix columnheaders rowheaders with image
```

テキストデータでは、空行やコメント行は配列データを終了させ、新たなデータブロックを開始します。いつものことですが、**splot** コマンドの **index** オプションを使ってファイル内のデータブロックを自由に選択できます。オプション columnheaders がある場合は、それは最初のデータブロックにのみ適用します。

Nonuniform matrix 入力データの最初の行は y 座標を持ちます。入力データの最初の列は x 座標を持ちます。バイナリ入力データでは、1 行目の最初の要素は列数でなければいけません (テキストデータではその番号は無視されます)。

非一様 (nonuniform) な matrix データを描画するコマンドの例:

```
splot 'file' nonuniform matrix using 1:2:3  # テキストデータ
splot 'file' binary matrix using 1:2:3      # バイナリデータ
```

よって、非一様な matrix データの構造は以下のようになります:

```
<N+1>  <x0>   <x1>   <x2>  ...  <xN>
 <y0> <z0,0> <z0,1> <z0,2> ... <z0,N>
 <y1> <z1,0> <z1,1> <z1,2> ... <z1,N>
  :      :      :      :   ...    :
```

これらは以下のような 3 つの数字の組に変換されます:

```
<x0> <y0> <z0,0>
<x0> <y1> <z0,1>
<x0> <y2> <z0,2>
 :    :     :
<x0> <yN> <z0,N>
<x1> <y0> <z1,0>
<x1> <y1> <z1,1>
 :    :     :
```

そして、これらの 3 つの数字の組は **gnuplot** の孤立線に変換され、その後 **gnuplot** が通常の方法で描画の残りを行います。

Sparse matrix 書式:

```
sparse matrix=(cols,rows) origin=(x0,y0) dx=<delx> dy=<dely>
```

\\

sparse な matrix 形式は、**plot** や **splot** のコマンドラインの一部分として、一様な格子を定義します。最初は格子は空です。その後で入力ファイルから個々の点の値を任意個、一行から一つずつ読み込み、それをそれに最も近い格子点に割り当てて行きます。すなわち、データ行の

```
x y value
```

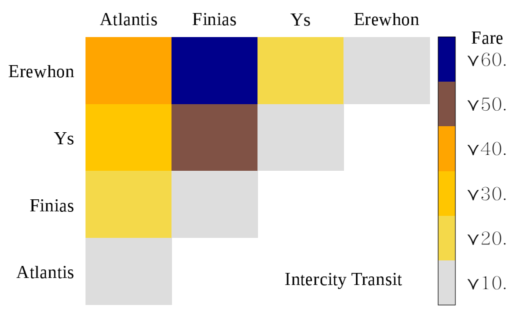
が、

```
i = (x - x0) / delx
j = (y - y0) / dely
matrix[i,j] = value
```

のように評価されます。matrix のサイズは必須です。**origin** (オプション) のデフォルトは origin=(0,0) です。**dx** (オプション) のデフォルトは dx=1 です。**dy** (オプション) のデフォルトは dy=dx です。

この形式は、順序がバラバラで、不完全であってもよいデータから **image**, **rgbimage**, **rgbalpha** 描画スタイルなどを用いて温度分布図 (heatmap) を生成するのが本来の使用目的です。以下の例は、与えらえた上三角成分のみから 4x4 の温度分布図の形式の離散 matrix を生成します。

```
$DATA << EOD
1 1 10
1 2 20
1 3 30
1 4 40
2 2 10
2 3 50
2 4 60
3 3 10
3 4 20
4 4 10
EOD
plot $DATA sparse matrix=(4,4) origin=(1,1) with image
```



Every キーワード **every** は、matrix データに対して使用すると特別な意味を持ちます。データの点やブロックに適用するのでなく、データの行、列に適用します。matrix データの行と列の番号は、0 から始まり、よって番号 N の列は、(N+1) 番目であることに注意してください。書式:

```
plot 'file' matrix every {<column_incr>}
                         {:{<row_incr>}
                           {:{<start_column>}
                             {:{<start_row>}
                               {:{<end_column>}
                                 {:<end_row>}}}}}
```

例:

```
plot 'file' matrix every :::N::N   # N 番の行のすべての値を描画
plot 'file' matrix every ::3::7    # すべての行の 3-7 列を描画
plot 'file' matrix every ::3:0:7:4 # [3,0], [7,4] 枠の部分行列
```

Examples 行列やベクトルの操作のサブルーチン (C による) が **binary.c** に用意されています。バイナリデータを書くルーチンは

```
int fwrite_matrix(file,m,nrl,nrl,ncl,nch,row_title,column_title)
```

です。これらのサブルーチンを使う例が **bf_test.c** として用意されていて、これはデモファイル **demo/binary.dem** 用に複数のバイナリファイルを生成します。

plot での使用法:

```
plot 'a.dat' matrix
plot 'a.dat' matrix using 1:3
plot 'a.gpbin' {matrix} binary using 1:3
```

これらは配列の行を描画し、using 2:3 とすれば配列の列を描画、using 1:2 は、点の座標を描画します (多分無意味です)。オプション **every** を適用することで明示的に行や列を指定できます。

例 – テキストデータファイルの配列の軸の拡大:

```
splot `a.dat` matrix using (1+$1):(1+$2*10):3
```

例 – テキストデータファイルの配列の第 3 行の描画:

```
plot 'a.dat' matrix using 1:3 every 1:999:1:2
```

(行は 0 から数えられるので、3 ではなくて 2 を指定します)。

Gnuplot は、**array**, **record**, **format**, **filetype** などの general バイナリ形式を特定するようなキーワードをつけずにオプション **binary** を使うことで、matrix バイナリファイルを読み込むことができます。その他の変換用の general バイナリキーワードは、matrix バイナリファイルにも適用できるでしょう。(詳細は、以下参照: **binary general (p. 157)**)

データファイルの例

以下は 3 次元データファイルの描画の単純な一つの例です。

```
splot 'datafile.dat'
```

ここで、"datafile.dat" は以下を含むとします:

```
# The valley of the Gnu.
   0 0 10
   0 1 10
   0 2 10

   1 0 10
   1 1 5
   1 2 10

   2 0 10
   2 1 1
   2 2 10

   3 0 10
   3 1 0
   3 2 10
```

この "datafile.dat" は 4*3 の格子 (それぞれ 3 点からなるブロックの 4 つの行) を定義することに注意して下さい。行 (ブロック) は 1 行の空行で区切られます。

x の値はそれぞれのブロックの中で定数になっていることに注意して下さい。もし y を定数の値とし、隠線処理が有効な状態で描画すると、その曲面は裏返しで書かれることになります。

格子状データ (grid data) に対して、個々のブロック内で x の値を定数としておく必要はありませんし、同じ場所の y の値を同じ値に揃えておく必要もありません。**gnuplot** は個々のブロック内の点の数が等しいということを必要としているだけです。しかし、等高線を導くのに用いられる曲面の網目は、対応する点を列的に選んで結ぶため、不揃いの格子データに対する曲面の描画への影響は予想できません。それはケースバイケースの原理でテストすべきでしょう。

格子状データ (grid data)

3 次元描画のためのルーチンは、個々の網目の格子においては一つの標本点と一つのデータ点がある、という形の格子状データ用に設計されています。各データ点は、関数の値を評価すること (以下参照: **set isosamples (p. 227)**)、またはデータファイルを読み込むこと (以下参照: **splot datafile (p. 320)**) によって生成されます。"孤立線" という言葉は関数に対しても、データに対してもその網目の線を表すものとして用いられます。網目は、必ずしも x, y に関する長方形でなくてもよく、u,v で媒介変数表示されても構わないことに注意して下さい。以下参照: **set isosamples (p. 227)**。

しかし、**gnuplot** はそのような形式を必ずしも必要とはしません。例えば関数の場合は、**samples** は **isosamples** と違っていても構いません。すなわち、x-孤立線のうち、1 本の y-孤立線と交わないものがいくつかあることがあります。データファイルの場合は、個々のブロックのばらついた点の個数が全て同じであれば、"孤立線は" はブロックの点を結び、"横断孤立線" は各ブロックの対応する点同士を結び、"曲面" を作ろうとします。

どちらの場合でも、等高線、および隠線処理モードは点が意図したフォーマットであった場合とは違った描画を与えることになります。

ばらつきのあるデータは、描画前に格子に合わせることが可能です。以下参照: **set dgrid3d (p. 213)**。

等高線に関するコードは、y-孤立線の点と、それに対応する隣の y-孤立線上の点の間の線分に沿っての z の張力を計測します。よって、x-孤立線に、y-孤立線との交点とはならないような標本点があるような曲面に対しては、**splot** の等高線はそのような標本点を無視することになります。以下を試してみて下さい:

```
set xrange [-pi/2:pi/2]; set yrange [-pi/2:pi/2]
set style function lp
set contour
set isosamples 10,10; set samples 10,10;
splot cos(x)*cos(y)
set samples 4,10; replot
set samples 10,4; replot
```

Splot の曲面 (splot surfaces)

splot は点の集まりとして、あるいは、それらの点を結ぶことによって曲面を表示することができます。**plot** と同様に、点はデータファイルから読むこともできますし、指定された区間で関数の値を評価して得ることもできます。以下参照: **set isosamples (p. 227)**。曲面は、各点を線分で結ぶことで近似的に作られます。以下参照: **set surface (p. 289)**。そしてその場合曲面は、**set hidden3d** で不透明にもできます。3 次元曲面を眺める向きは、**set view** で変更できます。

さらに、格子状のデータ点に対しては、**splot** は同じ高さを持つ点を補間することができ (以下参照: **set contour (p. 207)**)、そしてそれらを結んで等高線を描くことができます。さらに、その結び方には真直な線分や滑らかな線を使うことができます (以下参照: **set cntrparam (p. 204)**)。関数は、常に **set isosamples** と **set samples** で決定される格子状データとして評価されます。一方、ファイルのデータは、**data-file** に書かれているような格子状データフォーマットにするか、あるいは格子データを生成する (以下参照: **set dgrid3d (p. 213)**) ということをしなければそうはなりません。

等高線は曲面の上に表示することもできますし、底面に射影することもできます。底面への射影は、ファイルに書き出すこともでき、そしてそれを **plot** で再び読み込んで **plot** のより強い整形能力を生かすこともできます。

ボクセル格子データ (voxel-grid)

書式:

```
splot $voxelgridname with {dots|points} {above <threshold>} ...
splot $voxelgridname with isosurface {level <threshold>} ...
```

ボクセルデータは、指定した閾値 (threshold; デフォルトは 0) よりも大きい値の個々のボクセルに印を付けるように、with dots か with points で描画できます。色、点種、線幅の属性は、通常と同様に追加指定できます。

多くの視方向に対して、ボクセル格子点は、ディスプレイ上で互いを隠したりモアレを生成したりします。それらの副作用は、jitter を導入してドットや点を実際のボクセル格子座標からランダムにずらすことで避けることが可能です。以下参照: **set jitter (p. 228)**。

密なボクセル格子は、**pointinterval** 属性 (省略形 **pi**) を使うことで、点の数を減らすよう標本化レベルを下げることが可能です。

```
splot $vgrid with points pointtype 6 pointinterval 2
```

with isosurface は、指定された閾値より大きいボクセル全体を包む、3 次元モザイク型曲面を生成します。この曲面は、閾値それ自体を通過するよう、線形補間により調整されて配置されます。

以下参照: **set vgrid (p. 298)**, **vfill (p. 333)**。以下のデモも参照してください: **vplot.dem**, **isosurface.dem**。

Stats (簡単な統計情報)

書式:

```
stats {<ranges>} 'filename' {matrix | using N{:M}} {name 'prefix'}
      {{no}output}
stats $voxelgridname {name 'prefix'}
```

このコマンドは、ファイルの 1 列、または 2 列のデータの簡単な統計情報を提供します。using 指定子は、plot コマンドと同じ形で解釈されますが、**index**, **every**, **using** 指定に関する詳細については以下参照: **plot (p. 155)**。データ点は、その解析の前に xrange, yrange に従ってフィルタにかけられます。以下参照: **set xrange (p. 305)**。その情報はデフォルトではスクリーンに出力されますが、コマンド **set print** を先に使うことで出力をファイルにリダイレクトしたり、オプション **nooutput** を使うことで出力しないようにすることもできます。

ファイルが見つからない、あるいはファイルから読み込めない場合は、致命的ではない警告を発行します。これは、プログラムエラーを伴わないファイルの存在確認に利用できます。以下参照: **stats test (p. 329)**。

画面出力に加え、gnuplot は個々の統計情報を 3 つの変数グループに保存します。1 番目の変数グループは、どんなデータが並んでいるかを示しますが、先頭行の配列は、**set datafile columnheaders** が有効な場合にのみ作成されます:

STATS_records	N	範囲内のデータ行の総数 N
STATS_outofrange		範囲外として除かれた行数
STATS_invalid		無効/不完全/欠損データ行の総数
STATS_blank		空行の総数
STATS_blocks		ファイル内のデータの index ブロック数
STATS_columns		データ先頭行の列数
STATS_column_header		先頭行に見つかった文字列を保持する配列

2 番目の変数グループは、1 つの列の、範囲内のデータの性質を示します。この列は y の値として扱われます。y 軸が自動縮尺の場合は、対象とする範囲に限界はありませんが、そうでなければ範囲 [ymin:ymax] 内の値のみを対象とします。

2 つの列を同時に 1 回の **stats** コマンドで解析する場合は、各変数名に"_x", "_y" という接尾辞を追加します。例えば STATS_min_x は、1 つ目の列のデータの最小値で、STATS_min_y は 2 つ目の列のデータの最小値を意味します。この場合、点は xrange と yrange の両方で検査することでふるいにかけます。

`STATS_min`		$\min(y)$	範囲内のデータ点の最小値
`STATS_max`		$\max(y)$	範囲内のデータ点の最大値
`STATS_index_min`		$i \mid y_i = \min(y)$	data[i] == STATS_min となる添字 i
`STATS_index_max`		$i \mid y_i = \max(y)$	data[i] == STATS_max となる添字 i
`STATS_mean`	$\bar{y} =$	$\frac{1}{N}\sum y$	範囲内のデータ点の平均値
`STATS_stddev`	$\sigma_y =$	$\sqrt{\frac{1}{N}\sum(y-\bar{y})^2}$	範囲内のデータ点の標本標準偏差
`STATS_ssd`	$s_y =$	$\sqrt{\frac{1}{N-1}\sum(y-\bar{y})^2}$	範囲内のデータ点の不偏標準偏差
`STATS_lo_quartile`			第一 (下の) 四分位境界値
`STATS_median`			メジアン値 (第二四分位境界値)
`STATS_up_quartile`			第三 (上の) 四分位境界値
`STATS_sum`		$\sum y$	和
`STATS_sumsq`		$\sum y^2$	平方和
`STATS_skewness`		$\frac{1}{N\sigma^3}\sum(y-\bar{y})^3$	範囲内のデータ点の歪度
`STATS_kurtosis`		$\frac{1}{N\sigma^4}\sum(y-\bar{y})^4$	範囲内のデータ点の尖度
`STATS_adev`		$\frac{1}{N}\sum\lvert y-\bar{y}\rvert$	範囲内のデータ点の平均絶対偏差
`STATS_mean_err`		$\sigma_y/\sqrt{N}$	平均値の標準誤差
`STATS_stddev_err`		$\sigma_y/\sqrt{2N}$	標準偏差の標準誤差
`STATS_skewness_err`		$\sqrt{6/N}$	歪度の標準誤差
`STATS_kurtosis_err`		$\sqrt{24/N}$	尖度の標準誤差

3 番目の変数グループは、2 つの列のデータの解析専用です。

`STATS_correlation`	x と y の不偏相関係数
`STATS_slope`	回帰直線 y = Ax + B の係数 A
`STATS_slope_err`	A の不確かさ
`STATS_intercept`	回帰直線 y = Ax + B の係数 B
`STATS_intercept_err`	B の不確かさ
`STATS_sumxy`	積和 (x*y の和)
`STATS_pos_min_y`	y の最小値を与える x 座標
`STATS_pos_max_y`	y の最大値を与える x 座標

キーワード **matrix** は、入力が行列形式であることを指示します (以下参照: **matrix (p. 321)**)。通常の統計情報は、すべての行列要素を考慮して生成します。行列のサイズは、変数 STATS_size_x, STATS_size_y に保存します。

`STATS_size_x`	行列の列数
`STATS_size_y`	行列の列数

STATS_index_xxx で示される添字の値は、plot コマンドの第 0 疑似列 ($0) の値に対応し、最初の点は添字は 0、最後の点の添字は N-1 となります。

メジアンと四分位境界値を探す際はデータの値をソートし、点の総数 N が奇数の場合は、その (N+1)/2 番目の値をメジアン値とし、N が偶数の場合は、N/2 番目と (N+2)/2 番目の値の平均値をメジアン値とします。四分位境界値も同様に処理します。

その後の描画に注釈をつけるためにコマンド **stats** を利用した例については、以下を参照してください。stats.dem。

現在のバージョンの gnuplot の **stats** コマンドでは、日時フィールド (**set xdata time** や **set ydata time**) でなければ対数軸のデータも処理できます。この制限は、将来のバージョンで緩和されるでしょう。

接頭辞名 (name)

2 つ以上のファイルやデータ列からの統計情報を並列に扱うことができれば便利ですので、変数のデフォルトの接頭辞である "STATS" を、オプション **name** でユーザが指定する文字列に置き換えることができるようになっています。例えば、異なる 2 つのファイルのそれぞれの 2 列目のデータの平均値は以下のようにして比較できます:

```
stats "file1.dat" using 2 name "A"
stats "file2.dat" using 2 name "B"
if (A_mean < B_mean) {...}
```

name として文字列定数を用意する代わりに、キーワード **columnheader** や関数 **columnheader(N)** により、データファイルの 1 行目から取得した任意の文字列から接頭辞を生成することもできます:

```
do for [COL=5:8] { stats 'datafile' using COL name columnheader }
```

ファイルの存在確認 (test for existence of a file)

存在しない、あるいは読み込めないファイルを描画しようとすると、それはエラーとなり、スクリプトや繰り返しの先を停止してしまいます。それを避けるには、以下の例のように stats コマンドが利用できます。

```
do for [i=first:last] {
    filename = sprintf("file%02d.dat", i)
    stats filename nooutput
    if (GPVAL_ERRNO) {
        print GPVAL_ERRMSG
        continue
    }
    plot filename title filename
}
```

Voxelgrid

```
stats $vgridname {name "prefix"}
```

コマンド stats は、ボクセル格子の内容を問い合わせるのにも使えます。これは、**show vgrid** と同じ情報を生成しますが、これはそれをスクリプトで利用できる変数に保存できます。

STATS_min	格子のすべてのボクセルの 0 でない最小値
STATS_max	格子のすべてのボクセルの最大値
STATS_mean	格子の 0 でないボクセルの平均値
STATS_stddev	0 でないボクセル値の標準偏差
STATS_ssum	格子のすべてのボクセルの和
STATS_nonzero	0 でないボクセル数

System

書式:

```
system "command string"
! command string
output = system("command string")
show variable GPVAL_SYSTEM
```

system "command" は、オペレーティングシステムのデフォルトシェルを呼び出し、そのサブプロセスとして "command" を実行します。関数として **system("command")** を呼び出した場合は、そのサブプロセスの標準出力からの文字ストリームを文字列として返します。最後に改行がついている場合は、それは結果文字列からは除去されます。以下も参照: **backquotes (p. 82)**。

そのサブプロセスの終了コードは、変数 GPVAL_SYSTEM_ERRNO と GPVAL_SYSTEM_ERRMSG に保存されます。しかし、もしコマンド文字列が 2 つ以上のプログラムを呼び出している場合は、そのうちの一つのプログラムがエラーを出しても、"成功" を返す可能性があることに注意してください。例えば、file = system("ls -1 *.plt | tail -1") は、*.plt ファイルが一つもない場合でも "成功" が返ります。それは、**ls** が失敗しても **tail** は成功するからです。

Test

このコマンドは、出力形式やパレットでどのような出力が行なえるかを画像でテストし表示します。

書式:

```
test {terminal | palette}
```

test または **test terminal** は、現在使用中の出力形式 (**terminal**) で使える線の種類、点の種類、または利用可能なその他の描画を生成します。

test palette は、R(z),G(z),B(z) (0<=z<=1) の状態を描画します。これらは、**set palatte** で定義した現在のカラーパレットの RGB 成分を示します。また、RGB を灰色階調に写像する NTSC 係数を用いて計算された視光度も描画します。さらにこのコマンドはこの対応関係を $PALETTE という名前のデータブロックに取り込みます。

Toggle

書式:

```
toggle {<plotno> | "plottitle" | all}
```

このコマンドは、対話型出力形式 (qt, wxt, x11) で表示されているグラフの key エントリ上で左クリックしたのと同じ効果を与えます。すなわち、そのグラフが表示されていればそれを消し、グラフが消えていれば再び表示します。**toggle all** は、ホットキー "i" と同様、有効なグラフすべてに作用します。**toggle "title"** の形式は、グラフのタイトルと完全に一致するタイトルを指定する必要がありますが、**toggle "ti*"** の場合は、グラフのタイトルと'*' の前の部分が一致する最初のグラフに作用します。現在の出力形式が対話型でない場合は、コマンド toggle は何もしません。

Undefine

1 つ、または複数の定義済みのユーザ変数を削除します。これは、初期化テストを含むようなスクリプトの状態をリセットするのに便利でしょう。

変数名には、最後の文字としてワイルドカード文字 * を使うことができます。ワイルドカード文字が見つかると、それより前の部分で始まるすべての変数を削除します。これは、共通の接頭語を使っている複数の変数を削除するのに便利でしょう。ただし、ワイルドカード文字は変数名の最後にしか使えないことに注意してください。**undefine** にワイルドカード文字のみを引数として与えた場合は何もしません。

例:

```
undefine foo foo1 foo2
if (!exists("foo")) load "initialize.gp"

bar = 1; bar1 = 2; bar2 = 3
undefine bar*                    # 3 つの変数を全部削除
```

Unset

コマンド set で設定したオプションは、それに対応した **unset** コマンドによってそのデフォルトの値に戻すことが可能です。**unset** コマンドには繰り返し節も利用できます。以下参照: **plot for (p. 183)**。

例:

```
set xtics mirror rotate by -45 0,10,100
...
unset xtics

# 番号 100 から 200 までのラベルを unset
unset for [i=100:200] label i
```

Linetype

書式:

```
unset linetype N
```

以前に単一の線種に割り当てたすべての特性を削除します。この後にこの線種を使用した場合、特性、色は現在の出力形式にデフォルトで設定されているものを使用します (すなわち gnuplot 4.6 より前のバージョンで有効だった、いわゆるデフォルトの線種)。

Monochrome

現在有効な白黒の線種をカラーの線種に切り替えます。**set color** と同等です。

Output

複数のグラフを一つの出力ファイルに書き出すことができる出力形式もあるので、描画の後で出力ファイルを自動的には閉じません。よってそのファイルを安全に印刷等をするためには、まず明示的に **unset out** や **set output** とすることで前のファイルを閉じた上で新しいファイルを開いてください。

Terminal

プログラムの最初に有効になるデフォルトの出力形式は、個々のシステム環境、gnuplot のコンパイルオプション、および環境変数 GNUTERM に依存します。そのデフォルトが何であっても、gnuplot はそれを内部変数 GNUTERM に保存しています。コマンド **unset terminal** は、その初期初期出力形式に復帰し、これは、**set terminal GNUTERM** とすることと同じです。しかし、GNUTERM が出力形式名の後に terminal オプションも含んでいる場合は、その代わりに **set terminal @GNUTERM** とする必要があります。

Warnings

```
set warnings
unset warnings
```

致命的ではないエラーに対する警告メッセージは、通常はファイル名、行番号、およびその警告を引き起こすコマンドラインをエコーした後で stderr に出力します。この警告は、コマンド **unset warnings** で抑制できますが、その場合でも必要ならコマンド **warn "message"** で警告を生成できます。それは明示的に **set warnings** で有効にするまで抑制し続けます。

Update

注意: このコマンドは「非推奨」です。代わりに **save fit** を使用してください。

Vclear

書式:

```
vclear {$gridname}
```

これは、存在するボクセル格子内のすべてのボクセル値を 0 にリセットします。格子名を指定しない場合、現在有効な格子をクリアします。

Vfill

書式:

```
vfill  FILE using x:y:z:radius:(<expression>)
vgfill FILE using x:y:z:radius:(<expression>)
```

コマンド **vfill** は、それがグラフを生成する代わりに現在有効なボクセル格子データを変更する以外は、コマンド **plot** と同様に作用します。入力ファイルから読み込んだ各点に対して、その点を含むボクセル、及び中心が (x,y,z) で指定半径 (radius) の球に含まれるその他すべてのボクセルが以下のように増やされます:

- (x,y,z) からそのボクセルのユーザ座標での原点 (vx,vy,vz) までの距離をユーザ変数 VoxelDistance にセットします。
- (x,y,z) からそのボクセルの格子座標での原点までの距離をユーザ変数 GridDistance にセットします。
- **using** 指定の 5 番目に指定した数式を評価します。この数式は、新しい VoxelDistance や GridDistance の値を使用できます。
- voxel(vx,vy,vz) += その数式 <expression> の評価結果

例:

```
vfill "file.dat" using 1:2:3:(3.0):(1.0)
```

このコマンドは、file.dat 内の各点の半径 3.0 の球の中にあるすべてのボクセル値を 1 増やします。この球と重なるボクセルの個数は、ユーザ座標での格子の間隔に依存します。

```
vgfill "file.dat" using 1:2:3:(2):(1.0)
```

このコマンドは、現在の点を中心とするボクセルの 5x5x5 の立方体内のボクセルの値を 1 増やします。半径の "2" は、x の両方向、y の両方向、z の両方向に丁度 2 ボクセルずつ広げることと解釈し、これはそれらの軸に関するユーザ座標の相対的なスケールとは無関係です。

例:

```
vfill "file.dat" using 1:2:3:4:(VoxelDistance < 1 ? 1 : 1/VoxelDistance)
```

このコマンドは、各点の 4 列目で決まる半径の円内のすべてのボクセル値を変更します。各ボクセルに追加される増分値は、データ点からの距離に従って減少します。

vfill と **vgfill** は、現在のボクセル格子の存在する値を常に増加させることに注意してください。ボクセル一つを 0 にリセットするには、**voxel(x,y,z) = 0** を使用し、すべての格子を 0 にリセットするには、**vclear** を使用してください。

Warn

書式:

```
warn "message"
```

コマンド **warn** は、実質的に **printerr** と同じですが、指定したメッセージを stderr に出力する前に、現在のファイル名か関数ブロック名、および現在の行番号を追加するところが違います。**printerr** とは違い、**warn** の出力は **unset warnings** で抑制されます。

While

書式:

```
while (<expr>) {
    <commands>
}
```

これは、コマンドのブロックを、<expr> が 0 でない値と評価される間、繰り返し実行します。このコマンドは、古い形式 (かっこなし) の if/else 構文と一緒に使うことはできません。以下も参照: **do (p. 136)**, **continue (p. 135)**, **break (p. 133)**。

第 IV 部

出力形式 (Terminal)

出力形式の一覧

gnuplot はとても多くの出力形式をサポートしています。これらは、適切な出力形式を、必要なら機能を変更する追加オプションをつけて選択することにより設定されます。以下参照: **set terminal (p. 291)**。

この文書は、あなたのシステム上で初期設定およびインストールがなされなかったために利用できない出力形式についても記述されているかも知れません。**legacy** (古い) と印のついた出力形式は、最近のバージョンの gnuplot ではデフォルトではビルドされず、実際には使用できないかもしれません。個々の gnuplot 対話形式実行時に、どの出力形式が有効なのかの一覧を見るには、オプションを何もつけずに 'set terminal' と打ってください。

(訳注: この日本語訳に含まれる terminal のマニュアルは、その一覧にはない出力形式のものも含まれているかもしれませんし、逆にその一覧内の出力形式でマニュアルがないものもあるかもしれません。)

TeX/LaTeX 文書システムでの使用のために設計された出力形式がいくつかあります。TeX 用の出力形式に関する要約が以下にあります: http://www.gnuplot.info/docs/latex_demo.pdf

Aifm

注意: 古い (legacy) 出力形式、元々は Adobe Illustrator 3.0+ 用。Adobe Illustrator はレベル 1 の PostScript ファイルを直接認識するので、これの代わりに **set terminal post level1** 使うべきでしょう。

書式:

```
set terminal aifm {color|monochrome} {"<fontname>"} {<fontsize>}
```

Aqua

この出力形式は MacOS 上の表示に関する AquaTerm.app に依存しています。

書式:

```
set terminal aqua {<n>} {title "<wintitle>"} {size <x> <y>}
                  {font "<fontname>{,<fontsize>}"}
                  {linewidth <lw>}"}
                  {{no}enhanced} {solid|dashed} {dl <dashlength>}}
```

<n> は描画するウィンドウの番号 (デフォルトでは 0) <wintitle> はタイトルバーに表示される名前 (デフォルトは "Figure <n>"), <x> <y> は描画サイズ (デフォルトは 846x594 pt = 11.75x8.25 インチ) です。

使用するフォントは <fontname> で指定し (デフォルトは "Times-Roman")、フォントサイズは <fontsize> で指定します (デフォルトは 14.0 pt)。

aqua 出力形式は、拡張文字列処理モード (enhanced text mode; 以下参照: **enhanced (p. 42)**) を、重ね書き以外はサポートしてます。フォントの使用はシステムで有効なフォントに制限されています。文字エンコー

ディングは、**set encoding** で選択できますが、現在は iso_latin_1, iso_latin_2, cp1250 と UTF8 (default) をサポートしています。

曲線は、実線か点線 (デフォルトは実線) のいずれかで描画でき、点線の間隔は倍率 <dashlength> (>0) で変更できます。

Be

出力形式 **be** は、X サーバを利用する **beos** オペレーティングシステム上で gnuplot をコンパイルする人のためのものです。環境変数 **DISPLAY** がセットされているか、環境変数 **TERM** が **xterm** にセットされているか、またはコマンドラインオプションとして **-display** が使われていれば、プログラムの起動時にこの出力形式が選択されます。

書式:

```
set terminal be {reset} {<n>}
```

複数のグラフ描画ウィンドウをサポートしています。**set terminal be <n>** は番号 n のウィンドウに出力します。n>0 の場合、その番号はウィンドウタイトルとアイコン名に **gplt <n>** として付けられます。現在のウィンドウはカーソル記号の変化で区別できます (デフォルトカーソルから十字カーソルへ)。

gnuplot ドライバが別な出力ドライバに変更されても、描画ウィンドウは開いたままになります。描画ウィンドウは、そのウィンドウにカーソルを置いて文字 q を押すか、ウィンドウマネージャのメニューの **close** を選択すれば閉じることができます。**reset** を実行すれば全てのウィンドウを一度に閉じれます。それは実際にウィンドウを管理している子プロセスを終了します (もし **-persist** が指定されていなければ)。

描画ウィンドウは **-persisit** オプションが与えられていなければ、対話の終了時に自動的に閉じられます。

描画サイズとアスペクト比は、**gnuplot** のウィンドウをリサイズすることでも変更できます。

線の幅と点のサイズは **gnuplot** の **set linestyle** で変更可能です。

出力ドライバ **be** に関しては、**gnuplot** は (起動時に)、コマンドライン、または設定ファイルから、geometry や font, name などの通常の X Toolkit オプションやリソースの指定を受け付けます。それらのオプションについては X(1) マニュアルページ (やそれと同等のもの) を参照してください。

他にも **be** 出力形式用の多くの **gnuplot** のオプションがあります。これらは **gnuplot** を呼ぶときにコマンドラインオプションとして指定するか、または設定ファイル ".Xdefaults" のリソースとして指定できます。これらは起動時に設定されるので、**gnuplot** 実行時には変更できません。

コマンドラインオプション (command-line_options)

X Toolkit オプションに加え、以下のオプションが gnuplot の立ち上げ時のコマンドラインで、またはユーザのファイル ".Xdefaults" 内のリソースとして指定できます:

'-mono'	カラーディスプレイ上で強制的に白黒描画
'-gray'	グレイスケールまたはカラーディスプレイ上でのグレイスケール描画 (デフォルトではグレイスケールディスプレイは白黒描画を受け付ける)
'-clear'	新しい描画を表示する前に (瞬間的に) 画面を消去
'-raise'	各描画後に描画ウィンドウを最前面へ出す
'-noraise'	各描画後に描画ウィンドウを最前面へ出すことはしない
'-persist'	gnuplot プログラム終了後も描画ウィンドウを残す

上記のオプション、はコマンドライン上での指定書式で、".Xdefaults" にリソースとして指定するときは異なる書式を使います。

例:

```
gnuplot*gray: on
```

gnuplot は描画スタイル **points** で描画する点のサイズの制御にも、コマンドラインオプション (**-pointsize** <v>) とリソース (**gnuplot*pointsize:** <v>) を提供しています。値 **v** は点のサイズの拡大率として使われる実数値 ($0 < v <= 10$) で、例えば **-pointsize 2** はデフォルトのサイズの 2 倍、**-pointsize 0.5** は普通のサイズの半分の点が使われます。

白黒オプション (monochrome_options)

白黒ディスプレイに対しては **gnuplot** は描画色 (foreground) も背景色 (background) も与えません。デフォルトでは背景は白、描画は黒です。**-rv** や **gnuplot*reverseVideo: on** の場合には背景が黒で描画は白になります。

カラーリソース (color_resources)

カラーディスプレイに対しては、**gnuplot** は以下のリソース (ここではそのデフォルトの値を示します)、または白黒階調 (greyscale) のリソースを参照します。リソースの値はシステム上の BE rgb.txt ファイルに書かれている色名、または 16 進の色指定 (BE のマニュアルを参照) か、色名と強度 (0 から 1 の間の値) をコンマで区切った値を使用できます。例えば **blue, 0.5** は半分の強度の青、を意味します。

gnuplot*background: white
gnuplot*textColor: black
gnuplot*borderColor: black
gnuplot*axisColor: black
gnuplot*line1Color: red
gnuplot*line2Color: green
gnuplot*line3Color: blue
gnuplot*line4Color: magenta
gnuplot*line5Color: cyan
gnuplot*line6Color: sienna
gnuplot*line7Color: orange
gnuplot*line8Color: coral

これらに関するコマンドラインの書式は、例えば以下の通りです。

例:

```
gnuplot -background coral
```

灰色階調リソース (grayscale_resources)

-gray を選択すると、**gnuplot** は、グレイスケールまたはカラーディスプレイに対して、以下のリソースを参照します (ここではそのデフォルトの値を示します)。デフォルトの背景色は黒であることに注意してください。

gnuplot*background: black
gnuplot*textGray: white
gnuplot*borderGray: gray50
gnuplot*axisGray: gray50
gnuplot*line1Gray: gray100
gnuplot*line2Gray: gray60
gnuplot*line3Gray: gray80
gnuplot*line4Gray: gray40
gnuplot*line5Gray: gray90
gnuplot*line6Gray: gray50
gnuplot*line7Gray: gray70
gnuplot*line8Gray: gray30

線描画リソース (line_resources)

gnuplot は描画の線の幅 (ピクセル単位) の設定のために以下のリソースを参照します (ここではそのデフォルトの値を示します)。0 または 1 は最小の線幅の 1 ピクセル幅を意味します。2 または 3 の値によってグラフの外観を改善できる場合もあるでしょう。

gnuplot*borderWidth: 2
gnuplot*axisWidth: 0
gnuplot*line1Width: 0
gnuplot*line2Width: 0
gnuplot*line3Width: 0
gnuplot*line4Width: 0
gnuplot*line5Width: 0
gnuplot*line6Width: 0
gnuplot*line7Width: 0
gnuplot*line8Width: 0

gnuplot は線描画で使用する点線の形式の設定用に以下のリソースを参照します。0 は実線を意味します。2 桁の 10 進数 **jk** (**j** と **k** は 1 から 9 までの値) は、**j** 個のピクセルの描画に **k** 個の空白のピクセルが続くパターンの繰り返しからなる点線を意味します。例えば '16' は 1 個のピクセルの後に 6 つの空白が続くパター

ンの点線になります。さらに、4 桁の 10 進数でより詳細なピクセルと空白の列のパターンを指定できます。例えば、'4441' は 4 つのピクセル、4 つの空白、4 つのピクセル、1 つの空白のパターンを意味します。以下のデフォルトのリソース値は、白黒ディスプレイ、あるいはカラーや白黒階調 (grayscale) ディスプレイ上の白黒描画における値です。カラーディスプレイではそれらのデフォルトの値はほとんど 0 (実線) で、**axisDashes** のみがデフォルトで '16' の点線となっています。

```
gnuplot*borderDashes: 0
gnuplot*axisDashes: 16
gnuplot*line1Dashes: 0
gnuplot*line2Dashes: 42
gnuplot*line3Dashes: 13
gnuplot*line4Dashes: 44
gnuplot*line5Dashes: 15
gnuplot*line6Dashes: 4441
gnuplot*line7Dashes: 42
gnuplot*line8Dashes: 13
```

Block

block 出力形式は、Unicode ブロック要素 (Block element; U+2580-259F) 文字、または点字文字 (Braille Patterns; U+2800-28FF) を使って解像度を向上させた疑似グラフィック出力を作成します。これは端末 (ターミナル) グラフィックの代用品です。それには、UTF-8 が表示できる端末、またはビューワが必要です。描画には、内部ビットマップコードを用います。テキストは **dumb** 出力形式ルーチンを用いて、グラフィックスの一番上に表示します。

書式:

```
      set term block
              {dot | half | quadrants | sextants | octants | braille |
                sextpua | octpua}
              {{no}enhanced}
              {size <x>, <y>}
              {mono | ansi | ansi256 | ansirgb}
              {{no}optimize}
              {[no]attributes}
              {numpoints | charpoints | gppoints}
              {[no]animate}
```

size は、ターミナルサイズを文字セル単位で設定します。

dot, **half**, **quadrants**, **sextants**, **braille** は、疑似グラフィックの生成に用いる文字セットの選択です。**dot** は単純なドットを、**half** は半ブロック文字群を使用します。**quadrants** は、縦横両方向に 2 倍の解像度を生む四分ブロック文字群を使います。**sextants** は、2x3 の六分ブロック文字群を使用します。**braille** は点字記号を使い、これは 2x4 の疑似解像度を提供します。**octants** は、提案された 2x4 ブロック文字を使います。**sectpua** と **octpua** は、KreativeKorp 私用領域 (PUA) 内の 2x3, 2x4 ブロック文字をそれぞれ使用しますが、そのうち利用可能なのは多分 **FairfaxHD** と **KreativeSquare** フォントのみです。

2x3 の六分ブロック文字 ('sextants') は、Unicode 13 にのみ含まれていることに注意してください。よって、それをサポートするフォントは未だ限定的です。それは点字記号についても同様です。利用可能なフォントには、例えば **unscii**, **IBM 3270**, **GNU Unifont**, **DejaVu Sans**, **FairfaxHD** などがあります。2x4 ブロック文字 ('octants') は、まだ 2022 年に将来の Unicode 規格に入ることが決まった段階にすぎません。**FairfaxHD** フォントには含まれています。

オプション **ansi**, **ansi256**, **ansirgb** は、出力に色を表示させるためのエスケープシーケンスを含ませます。これらは、あなたの端末が処理できないかもしれないことに注意してください。デフォルトは **mono** (白黒) です。エスケープシーケンスの一覧については、以下参照: **terminal dumb (p. 352)**。

オプション **attributes** は、太字やイタリック体文字用のエスケープシーケンスをサポートする端末、エミュレーター上でそれらの文字出力を可能にします。以下参照: **terminal dumb (p. 352)**。

ブロック文字群を使うことで、ビットマップの疑似解像度は上がりますが、これはカラーに対してはそうではありません。複数の **pixel** が同じ色を共有する必要があります。これは、ひとつの文字セルで、すべてのピクセルの色の平均を取ることで処理しています。**optimize** をつけると、この出力形式は背景色と前景色の両方の設定によってそれを改善しようとします。この手法は、**half** モードでは完全に機能しますが、**quadrants**, **sextants**, **octants** では難しさが増していきます。また、**braille** では使用できません。

gppoints は、グラフィックコマンドを使って点記号を描きます。端末の低解像度性のため、**braille** と **octant** モードではこれは大抵実用的です。多分エラーバーとしては最も有用でしょう。**charpoints** は、代わりに Unicode 記号文字を使用します。これらも常にグラフィックの一番上に描かれることに注意してください。**numpoints** は上付き下付きの数字を使用し、垂直方向に 2 倍の解像度にします。しかし **sextants** に対しては、文字の枠の中心位置の点は、通常の数字で描く必要があります。**charpoints** か **numpoints** を使っている場合、ビットマップの解像度はそれでも異なり、線と記号は一般に正確には揃わないことに注意してください。

オプション **animate** は、各描画後に、カーソル位置をグラフの左上にリセットし、前のグラフを上にスクロールして消す代わりにスクリーンの同じ領域に次のグラフを上書きするようにします。これは、同じ場所でのアニメーションを作成するためには有用でしょう。

Caca

[試験段階] **caca** 出力形式は、**libcaca** を使ってアスキー文字によるグラフを描く、ほぼ娯楽的な出力モードです。**dumb** 出力形式と比べると、こちらは色、箱の塗り潰し、画像、文字列の回転、多角形の塗り潰し、マウス操作をサポートしています。

書式:

```
     set terminal caca {{driver | format} {default | <driver> | list}}
                       {color | monochrome}
                       {{no}inverted}
                       {enhanced | noenhanced}
                       {background <rgb color>}
                       {title "<plot window title>"}
                       {size <width>,<height>}
                       {charset ascii|blocks|unicode}
```

オプション **driver** (または **format**) は、表示ドライバとして **libcaca** を選択するか、または出力ドライバを選択します。**default** は、**libcaca** にその環境のデフォルトのディスプレイドライバを選択させます。デフォ

ルトのドライバは、**gnuplot** の起動前に環境変数 CACA_DRIVER を設定しておくことで変更できます。**set term caca driver list** を使用すると、サポートする出力モードの一覧を表示します。

オプション **color** と **monochrome** は、カラーか白黒出力を選択します。これは、線の記号も変更することに注意してください。デフォルトの白背景を黒背景にしたい場合は、オプション **inverted** を使ってください。これは、デフォルトの線種の黒を白にも変えます。

拡張文字列処理は、オプション **enhanced** を使うことで有効になります。以下参照: **enhanced text (p. 42)**。

出力ウィンドウのタイトルは、**libcaca** ドライバがサポートしていれば、オプション **title** で変更できます。

オプション **size** は、キャンバスのサイズを文字単位で選択します。デフォルトは 80 x 25 です。バックエンドがサポートしていれば、キャンバスサイズは、現在のウィンドウ/ターミナルのサイズに自動的に合います。"x11" と"gl" ウィンドウのデフォルトのサイズは、環境変数 CACA_GEOMETRY で制御できます。"win32" ドライバでのウィンドウの位置・サイズ情報は、アプリケーションメニューで制御、及び恒常的な変更が行えます。

オプション **charset** は、曲線、点、多角形や長方形の塗り潰し、画像の中間色表現などで使われる文字集合を選択します。バックエンドとターミナルとフォントの組み合わせによっては、**blocks** か **unicode** の文字集合はサポートしない可能性もあることに注意してください。特に Windows では、"Lucida Console" や "Consolas" のような非ラスタフォントの使用を勧めます。

caca 出力形式は、マウス操作をサポートしています。**libcaca** のいくつかのバックエンド (例えば slang や ncurses) はマウスクリックしたマウスの位置しか更新しないことに注意してください。修飾キー (Ctrl, Alt, Shift) は、**libcaca** ではサポートしていないので、利用できません。

caca 出力形式のデフォルトの **encoding** は utf8 です。cp437 **encoding** もサポートしています。

libcaca のサポートする色の数は、バックエンドにより異なります。たいていのバックエンドは、16 色の前景色と 16 色の背景色のみをサポートしていますが、例えば "x11" バックエンドは、Truecolor をサポートしています。

出力形式と **libcaca** バックエンドによっては、背景色 8 色しかサポートさない場合もあります。明色 (背景色の中で最も重要) は、文字を輝かせていると解釈されます。この場合、**background rgb "gray"** を使用してみてください。

以下の libcaca Web サイト`http://caca.zoy.org/wiki/libcaca`

および libcaca 環境変数に関する説明`http://caca.zoy.org/doxygen/libcaca/libcaca-env.html`

も参照してみてください。

Caca limitations and bugs

caca 出力形式には、既知のバグと制限があります:

Unicode のサポートは、ドライバと出力形式依存です。"x11" バックエンドは libcaca version 0.99.beta17 から unicode をサポートしていますが、**libcaca** < 0.99.beta20 でのバグのため、"slang" ドライバは unicode をサポートしていません。**libcaca** < 0.99.beta19 には、不正な 8 ビット列を与えると無限ループを引き起こすというバグがあることに注意してください。

明るい背景色は点滅することがあります。

マウス操作では修飾キーはサポートしません。以下参照: **term caca (p. 340)**。

拡張文字列の回転、および透明化はサポートしていません。**size** オプションは、オンスクリーンディスプレイでは考慮されません。

凡例 (key) の箱を正しく描くには、以下のようにしてください:

```
set key width 1 height 1
```

拡張文字列の位置合わせは、UTF-8 文字列が含まれている場合はうまくいきません。Windows のコンソールウィンドウのリサイズは、libcaca のバグのため正しく機能しません。タイトル行の上で "X" をクリックしてターミナルウィンドウを閉じる機能は wgnuplot を終了させてしまいますので、ウィンドウを閉じるには "q" を打ってください。

Cairolatex

出力形式 **cairolatex** は、cairo と pango の補助ライブラリを使って、EPS (Encapsulated PostScript), PDF, PNG 出力を生成しますが、文字列出力には出力形式 **epslatex** と同じやり方で LaTeX を使用します。

書式:

```
set terminal cairolatex
                {eps | pdf | png}
                {standalone | input}
                {blacktext | colortext | colourtext}
                {header <header> | noheader}
                {mono|color}
                {{no}transparent} {{no}crop} {background <rgbcolor>}
                {font <font>} {fontscale <scale>}
                {linewidth <lw>} {rounded|butt|square} {dashlength <dl>}
                {size <XX>{unit},<YY>{unit}}
                {resolution <dpi>}
```

cairolatex 出力形式は、epscairo 出力形式 (**termnal epscairo**) や pdfcairo 出力形式 (**terminal pdfcairo**) と同等のグラフを出力しますが、テキスト文字列はグラフの中に入れるのではなく、LaTeX に渡します。以下で触れないオプションについては、以下参照: **pdfcairo (p. 372)**。

eps, **pdf**, **png** は、グラフ出力の形式を選択します。latex/dvips 用には **eps** を、pdflatex 用には **pdf** を使用してください。もしあなたのグラフが大量の点数を持つ場合は、ファイルサイズを減らすために **png** を使用してください。オプション **png** を使用した場合、追加オプション **resolution** も受け付け、結果の PNG のピクセル密度を制御できます。**resolution** の引数は整数で、DPI の暗黙の単位で与えます。

blacktext は、カラーモードでもすべての文字列を黒で書くようにします。

cairolatex 出力ドライバは、文字列の位置の特別な制御方法を提供します: (a) '{' で始まるすべての文字列は、'}' もその文字列の最後に必要ですが、その文字列全体を LaTeX で横にも縦にもセンタリングします。(b) '[' で始まる文字列は、その次に位置指定文字 (t,b,l,r,c のうち 2 つまで)、']{'、対象文字列、と続き最後に '}' で閉じますが、この文字列は、LaTeX が LR-box として処理できるものならなんでも構いません。位置合わせを完全に行うには、\rule{}{} も有用でしょう。以下も参照: **pslatex (p. 382)**。複数行に渡るラベルを生成する場合、\shortstack を使用してください。例:

```
set ylabel '[r]{\shortstack{first line \\ second line}}'
```

コマンド **set label** のオプション **back** は、他の出力形式とはやや異なる方法で処理します。**back** を使用したラベルは、他のすべての描画要素の後ろに印字し、**front** を使用したラベルは、他のすべての上に印字します。

このドライバは 2 つの異なるファイルを作成します。一つは図の eps か pdf か png 部分で、もう一つは LaTeX 部分です。その LaTeX ファイルの名前は、コマンド **set output** のものを使用し、eps/pdf/png ファイルの名前は、その拡張子 (通常は '.tex') を '.eps'/'.pdf'/'.png' に置き換えたものを使用します。出力ファイルを指定しなかった場合は、LaTeX 出力はしません。**multiplot** モード以外では、次の plot を行う前に出力ファイルを閉じるのを忘れないでください。

この画像をあなたの LaTeX 文書に取り込むには、'\input{filename}' を使用してください。'.eps'/'.pdf'/'.png' ファイルは、\includegraphics{...} で取り込むので、LaTeX 文書のプリアンブルに \usepackage{graphicx} を入れる必要があります。色付きの文字 (オプション **colourtext**) を使用する場合は、プリアンブルに \usepackage{color} も入れる必要があります。

フォント選択に関する挙動は、ヘッダーモードに依存します。いずれの場合でも、与えられたフォントサイズは適切な大きさを計算するのに使われます。**standalone** モードを使っていない場合は、それを読みこんだところで実際に LaTeX が使用しているフォントとフォントサイズが使われるので、フォントを変更するには LaTeX のコマンドを使用してください。LaTeX 文書の方で 12pt のフォントサイズを使っていれば、オプションとして '", 12"' を指定してください。フォント名は無視されます。'standalone' の場合は、与えられたフォントとフォントサイズを使用します。詳細は以下を参照してください。

文字列を色付けして印字するかどうかは、TeX のブール変数 \ifGPcolor と \ifGPblacktext で制御できます。\ifGPcolor が true で \ifGPblacktext が false のときのみ文字列が色付けされます。これらの変更は、生成された TeX ファイル中で行うか、または大域的にあなたの TeX ファイルのプリアンブルで、例えば以下のようにして設定できます:

```
\newif\ifGPblacktext
\GPblacktexttrue
```

局所的な設定は、大域的な値がない場合にのみ効力を持ちます。

出力形式 cairolatex を使う場合は、コマンド **set output** で TeX ファイル設定する際にファイルの拡張子 (通常は ",tex") をつけてください。グラフのファイル名は、その拡張子を置きかえることで作られます。

standalone モードを使う場合、LaTeX ファイルに完全な LaTeX のヘッダが追加され、グラフファイルのファイル名には "-inc" が追加されます。**standalone** モードは、dvips, pdfTeX, VTeX を使う場合に正しいサイズの出力を作る TeX ファイルを生成します。デフォルトでは **input** で、これは LaTeX 文書から \input コマンドで取り込まれる必要があるファイルを生成します。

"" や "default" 以外のフォントを与えた場合、それは LaTeX のフォント名であるとみなされます。それは ',' 区切りで最大 3 つの部分からなる、'fontname,fontseries,fontshape' の形式です。デフォルトの fontshape や fontseries を使いたい場合は、それらは省略できます。よって、フォント名の実際の書式は、'{fontname}{,fontseries}{,fontshape}' となります。(訳注: より gnuplot 風に言えば '{<fontname>}{,{<fontseries>}{,<fontshape>}}') 名前の各部分の指定法は、LaTeX のフォント系の慣習に従う必要があります。フォント名 (fontname) は 3 から 4 文字の長さで、以下のようになっています: 最初の文字はフォントの供給者、次の 2 つの文字はフォント名用、オプションとして特別なフォント用に 1 文字追加できます。例えば、'j' は古いスタイルの数字用のフォント、'x' はエキスパートフォント用です。多くのフォント名が以下に記述されています: http://www.tug.org/fontname/fontname.pdf

例えば、'cmr' は Computer Modern Roman を、'ptm' は Times-Roman, 'phv' は Helvetica を意味しています。font series は、グリフの太さを表し、多くの場合は、'm' が標準 ("medium")、'bx' か 'b' が太字 (bold) の

フォントを意味します。font shape は、一般的には 'n' が立体 (upright)、'it' がイタリック (italic)、'sl' が斜体 (slanted)、'sc' がスモールキャピタル (small caps) を意味します。異なる series や shapes を提供するフォントもあります。

例:

Times-Roman のボールド体 (周りの文字列と同じ形状) を使うには:

```
set terminal cairolatex font 'ptm,bx'
```

Helvetica, ボールド体、イタリックを使うには:

```
set terminal cairolatex font 'phv,bx,it'
```

周りと同じで斜体の形状のフォントを使うには:

```
set terminal cairolatex font ',,sl'
```

スモールキャピタルを使うには

```
set terminal cairolatex font ',,sc'
```

この方法では、テキストフォントだけが変更されます。数式フォントも変更したい場合は、ファイル "gnuplot.cfg" か、または以下で説明するオプション **header** を使う必要があります。

standalone モードでは、フォントサイズはコマンド **set terminal** で指定したフォントサイズを取ります。指定したフォントサイズを使うためにはファイル "size<size>.clo" が LaTeX の検索パスにある必要があります。デフォルトでは 10pt, 11pt, 12pt をサポートしています。パッケージ "extsizes" がインストールされていれば、8pt, 9pt, 14pt, 17pt, 20pt も追加されます。

オプション **header** は一つの文字列を引数として取り、その文字列を生成する LaTeX ファイルに書き出します。**standalone** モードでは、それはプリアンブルの \begin{document} の直前に書きますが、**input** モードでは、それはグラフに関するすべての設定を局所化するための \begingroup コマンドのの直後に書きます。

例:

T1 フォントエンコーディングを使い、テキストフォントと数式フォントを Times-Roman に、sans-serif フォントを Helvetica に変えるには:

```
set terminal cairolatex standalone header \
"\\usepackage[T1]{fontenc}\n\\usepackage{mathptmx}\n\\usepackage{helvet}"
```

グラフ内では太字 (bold) を使うが、グラフ外のテキストはそうしない:

```
set terminal cairolatex input header "\\bfseries"
```

LaTeX がファイル "gnuplot.cfg" を見つけると、**standalone** モードではそれをプリアンブルに取り込みます。これは、さらに設定を追加するのに使えます。例: 文書のフォントを、数式フォント ("mathptmx.sty" が処理) も合わせて Times-Roman, Helvetica, Courier にするには:

```
\usepackage{mathptmx}
\usepackage[scaled=0.92]{helvet}
\usepackage{courier}
```

ファイル "gnuplot.cfg" は、コマンド **header** で設定するヘッダー情報よりも前に読み込みますので、"gnuplot.cfg" で設定するものを **header** を使って上書きすることができます。

Canvas

出力形式 **canvas** は、HTML5 の canvas 要素上に描画する javascript コマンドの集合を生成します。書式:

```
set terminal canvas {size <xsize>, <ysize>} {background <rgb_color>}
                    {font {<fontname>}{,<fontsize>}} | {fsize <fontsize>}
                    {{no}enhanced} {linewidth <lw>}
                    {rounded | butt | square}
                    {dashlength <dl>}
                    {standalone {mousing} | name '<funcname>'}
                    {jsdir 'URL/for/javascripts'}
                    {title '<some string>'}
```

<xsize> と <ysize> は描画領域のピクセル単位でのサイズを設定します。standalone モードでのデフォルトのサイズは、600x400 ピクセルです。デフォルトのフォントサイズは 10 です。

注: ファイル canvastext.js で提供している Hershey simplex Roman フォントのアスキー部分のフォント一つだけが利用できます。これは、ファイル canvasmath.js で置き換えることもでき、そこには UTF-8 エンコードされた Hershey simplex Greek と math symbols も含まれています。他の出力形式に合わせて、**font "name,size"** の形式も使えるようになっています。今のところ **name** のフォント名部分は無視されますが、そのうちにブラウザが名前付きフォントをサポートしだすでしょう。

デフォルトの **standalone** モードは、HTML 5 の canvas 要素を使用してグラフを描画するような javascript コードを含む HTML ページを生成します。その HTML ページは、2 つの必要な javascript ファイル 'canvastext.js'、'gnuplot_common.js' にリンクします。点線をサポートするためにはさらに追加ファイル 'gnuplot_dashedlines.js' が必要です。デフォルトではそれらはローカルファイルへのリンクで、Unix 互換のシステムでは通常はそれらディレクトリ /usr/local/share/gnuplot/<version>/js にあります。他の環境については、インストールに関する注意を参照してください。この設定は、オプション **jsdir** に別のローカルディレクトリ、あるいは一般的な URL を指定することで変更できます。グラフをリモートクライアントのマシンで見れるようにする場合は、通常は後者の設定が適切でしょう。

canvas 出力形式で生成される描画はすべてマウス操作可能です。キーワード **mousing** を追加すると、**standalone** モードのグラフの下にマウストラッキングボックスをつけます。これは、**canvastext.js** が置かれているのと同じローカルディレクトリ、または URL 内の、'gnuplot_mouse.js' という javascript ファイルへのリンクと 'gnuplot_mouse.css' というマウスボックスに関するスタイルシートも追加します。

オプション **name** は、javascript のみを含むファイルを一つ生成します。それが含む javascript 関数と、それが描画する canvas 要素の id の両方は、以下の文字列パラメータから取られます。例えば以下のコマンド

```
set term canvas name 'fishplot'
set output 'fishplot.js'
```

は、javascript 関数 fishplot() を含むファイルを生成し、その関数はグラフを id=fishplot の canvas 上に描画します。この javascript 関数を呼び出す HTML ページは、上で説明した canvastext.js も読み込まなければいけません。上のように生成した、この fishplot を取りこむ最小の HTML ファイルは以下のようになります:

```
<html>
<head>
    <script src="canvastext.js"></script>
    <script src="gnuplot_common.js"></script>
```

```
</head>
<body onload="fishplot();">
    <script src="fishplot.js"></script>
    <canvas id="fishplot" width=600 height=400>
        <div id="err_msg">No support for HTML 5 canvas element</div>
    </canvas>
</body>
</html>
```

このキャンバス上に描かれるそれぞれのグラフの名前は、fishplot_plot_1, fishplot_plot_2 等となります。外部の javascript ルーチンでそれらを参照することもできます。例: gnuplot.toggle_visibility("fishplot_plot_2")

Cgm

cgm ドライバは CGM 出力 (Computer Graphics Metafile Version 1) を生成します。このファイルフォーマットは ANSI 規格書 X3.122-1986 "Computer Graphics - Metafile for the Storage and Transfer of Picture Description Information" で定義されているものの一部分です。

書式:

```
      set terminal cgm {color | monochrome} {solid | dashed} {{no}rotate}
                       {<mode>} {width <plot_width>} {linewidth <line_width>}
                       {font "<fontname>,<fontsize>"}
                       {background <rgb_color>}
 [deprecated]          {<color0> <color1> <color2> ...}
```

solid は全ての曲線を実線で描き、どんな点線パターンも塗りつぶします; <mode> は **landscape**, **portrait**, **default** のいずれか; <plot_width> はポイント単位でのグラフの仮定されている幅; <line_width> はポイント単位での線幅 (デフォルトは 1); <fontname> はフォントの名前 (以下のフォント一覧参照); そして <fontsize> はポイント単位でのフォントのサイズ (デフォルトは 12) です。

最初の 6 つのオプションはどの順番で指定しても構いません。**default** を選択すると、全てのオプションをそのデフォルトの値にします。

線の色を **set term** コマンドで設定する仕組みは、今は非推奨です。代わりに、背景色は分離されたキーワード background で、線の色は **set linetype** で設定すべきでしょう。この非推奨の仕組みでは色は 'xrrggbb' の形式で受けつけますが、x は文字 'x' そのもの、'rrggbb' は 16 進数での赤、緑、青の成分です。最初の色を背景色として使い、その後に続く色指定を順次線の色ととして割り当てていました。

例:

```
      set terminal cgm landscape color rotate dashed width 432 \
                     linewidth 1  'Helvetica Bold' 12       # デフォルト
      set terminal cgm linewidth 2  14  # やや広い線とやや大きいフォント
      set terminal cgm portrait "Times Italic" 12
      set terminal cgm color solid    # 面倒な点線など消えてしまえ !
```

CGM のフォント (font)

CGM (Computer Graphics Metafile) ファイルの最初の部分、メタファイルの記述部分には、フォントリスト (font table) が含まれています。画像の本体部では、フォントはこのリストにある番号で指定されます。デフォルトではこのドライバは以下の 35 個のフォントリストを生成し、さらにこのリストの **Helvetica**, **Times**, **Courier** の各フォントの **italic** を **oblique** で置き換えたもの、およびその逆による 6 つの追加のフォントが含まれます (Microsoft Office と Corel Draw CGM の import フィルタは **italic** と **oblique** を同じものとして扱うからです)。

CGM fonts	
Helvetica	Hershey/Cartographic_Roman
Helvetica Bold	Hershey/Cartographic_Greek
Helvetica Oblique	Hershey/Simplex_Roman
Helvetica Bold Oblique	Hershey/Simplex_Greek
Times Roman	Hershey/Simplex_Script
Times Bold	Hershey/Complex_Roman
Times Italic	Hershey/Complex_Greek
Times Bold Italic	Hershey/Complex_Italic
Courier	Hershey/Complex_Cyrillic
Courier Bold	Hershey/Duplex_Roman
Courier Oblique	Hershey/Triplex_Roman
Courier Bold Oblique	Hershey/Triplex_Italic
Symbol	Hershey/Gothic_German
ZapfDingbats	Hershey/Gothic_English
Script	Hershey/Gothic_Italian
15	Hershey/Symbol_Set_1
	Hershey/Symbol_Set_2
	Hershey/Symbol_Math

これらのフォントの最初の 13 個は WebCGM で要求されているものです。Microsoft Office の CGM import フィルタはその 13 個の標準フォントと'ZapfDingbats' と 'Script' をサポートしています。しかし、そのスクリプト (script) フォントは '15' という名前でしかアクセスできません。Microsoft の import フィルタの font の置き換えに関するより詳しい情報については、

```
C:\Program Files\Microsoft Office\Office\Cgmimp32.hlp
```

のヘルプファイル、または

```
C:\Program Files\Common Files\Microsoft Shared\Grphflt\Cgmimp32.cfg
```

の設定ファイルなどをチェックしてください。

set term コマンドでデフォルトのフォントリストにないフォント名を指定することも可能です。その場合、その指定したフォントが最初に現われる新しいフォントリストが作られます。そのフォント名に関して、スペル、単語の先頭の大文字化やどこにスペースが入るかなどが、作られる CGM ファイルを読むアプリケーションにとって適切なものであるかをちゃんと確認する必要があります。(gnuplot と任意の MIL-D-28003A 準拠アプリケーションは、フォント名の大文字小文字の違いは無視します。) 新しいフォントをいくつも追加したい場合は、**set term** コマンドを繰り返し使用してください。

例:

```
set terminal cgm 'Old English'
set terminal cgm 'Tengwar'
set terminal cgm 'Arabic'
set output 'myfile.cgm'
plot ...
set output
```

set label コマンドでは新しいフォントを導入することはできません。

CGM のフォントサイズ (fontsize)

フォントは、ページが 6 インチの幅であると仮定して伸縮されます。**size** コマンドでページの縦横比が変更されていたり、CGM ファイルが異なる幅に変換されている場合、結果としてフォントのサイズも拡大されたり縮小されたりすることになります。仮定されている幅を変更するには、**width** オプションを使用してください。

Cgm linewidth

linewidth オプションは線の幅をポイント単位 (pt) で設定します。デフォルトの幅は 1 pt です。**fontsize** や **width** オプションのところで説明されているように、ページの実際の幅によってその縮尺は影響を受けます。

Cgm rotate

norotate オプションはテキストの回転をしないようにします。例えば Word for Windows 6.0c 用の CGM 入力フィルタは回転された文字列を受け付けますが、Word に付属する DRAW エディタはそれを受け付けることができず、グラフを編集すると (例えば曲線に見出しをつける)、全ての回転された文字列は水平方向になって保存されてしまい、Y 軸の見出しはクリップされる境界線を越えてしまうでしょう。**norotate** オプションを使えば、見栄えの良くない場所から Y 軸の見出しが始まってしまいますが、編集によってダメージを受けることはなくなります。**rotate** オプションはデフォルトの挙動を保証します。

Cgm solid

solid オプションは描画の点線の線描画スタイルを無効するのに使います。これは、カラーが有効である場合、また点線にすることでグラフが見にくくなる場合に有用でしょう。**dashed** オプションはデフォルトの挙動を保証し、この場合個々の線種に異なる点線のパターンが与えられます。

CGM のサイズ (size)

CGM グラフのデフォルトのサイズは、横置き (landscape) では幅 32599, 縦 23457、縦置き (portrait) では幅 23457, 縦 32599 です。

Cgm width

CGM ファイルの全ての長さは抽象的な単位を持ち、そのファイルを読むアプリケーションが最終的なグラフのサイズを決定します。デフォルトでは最終的なグラフの幅は 6 インチ (15.24 cm) であると仮定されています。この幅は正しいフォントサイズを計算するのに使われ、**width** オプションで変更できます。キーワード **width** の後に幅をポイント単位で指定します。(ここで、ポイントは PostScript と同様 1/72 インチを意味します。この単位は TeX では"big point" と呼ばれています。) 他の単位から変換するには、gnuplot の数式が使えます。

例:

```
set terminal cgm width 432            # デフォルト
set terminal cgm width 6*72           # 上と同じ値
set terminal cgm width 10/2.54*72     # 10 cm の幅
```

Cgm nofontlist

デフォルトのフォントリスト (font table) は WebCGM で勧告されているフォントを含んでいて、これは Microsoft Office と Corel Draw の CGM (Computer Graphics Metafile) 入力フィルタに適合しています。他のアプリケーションは異なるフォント、あるいは異なるフォント名を使用するかも知れませんが、それはマニュアルには書かれていないかも知れません。オプション **nofontlist** (**winword6** も同じ意味) を使用すると CGM ファイルからフォントリストを削除します。この場合、読み込んだアプリケーションはデフォルトのフォントリストを使用するでしょう。gnuplot はその場合でもフォント番号の選択のために自分のデフォルトのフォントリストを使用します。よって、'Helvetica' が 1 番になり、それがあなたの使用するアプリケーションのデフォルトフォントリストの最初のものになります。'Helvetica Bold' がそのフォントリストの 2 番目のフォントに対応し、他も同様となります。

Context

ConTeXt は (絵の描画のために) Metapost と高度に融合し、高品質な PDF 文書を生成するための TeX のマクロパッケージです。この出力形式は、Metafun ソースを生成しますが、これは手動で編集でき、外部からほとんどのことをあなたが設定できます。

ConTeXt + gnuplot モジュールの平均的なユーザには、このページを読むよりも、**Using ConTeXt** を参照するか、ConTeXt の gnuplot モジュールのマニュアルを参照することを推奨します。

出力形式 **context** は、以下のオプションをサポートしています:

書式:

```
set term context {default}
        {defaultsize | size <scale> | size <xsize>{in|cm}, <ysize>{in|cm}}
        {input | standalone}
        {timestamp | notimestamp}
        {noheader | header "<header>"}
        {color | colour | monochrome}
        {rounded | mitered | beveled} {round | butt | squared}
        {dashed | solid} {dashlength | dl <dl>}
```

```
{linewidth | lw <lw>}
{fontscale <fontscale>}
{mppoints | texpoints}
{inlineimages | externalimages}
{defaultfont | font "{<fontname>}{,<fontsize>}"}
```

standalone でないグラフ (**input**) では、オプションはグラフサイズを選択する **size**、すべてのラベルを倍率 <fontscale> で伸縮する **fontscasle**、および font サイズのみ意味を持ち、他のオプションは警告なく無視されるのみで、それらはそのグラフィックを読み込む .tex ファイルの方で設定してください。元の文書のフォントが 12pt ではない場合は、適切なフォントサイズを指定することを強く推奨します。それにより、gnuplot がラベル用にどれくらいの大きさのスペースを確保すればいいかを知ることができます。

default は、すべてのオプションをデフォルトの値にリセットします。

defaultsize は、描画サイズを 5in x 3in に設定します。**size** <scale> は、描画サイズをデフォルトサイズの <scale> 倍にしますが引数を ',' 区切りで 2 つ与えた場合は、最初のものは横のサイズを、2 つ目のものは垂直サイズを設定します。それらのサイズには、単位としてインチ ('in'), センチ ('cm') が使えますが、省略した場合はデフォルト値に対する比であるとみなします。

input (デフォルト) は、他の ConTeXt 文書から取り込めるグラフを生成します。**standalone** は、それに数行追加し、それ自身がそのままコンパイルできるようにします。その場合、**header** オプションが必要になるかもしれません。

standalone のグラフに設定/定義/マクロを追加したい場合は **header** を使用してください。デフォルトは **noheader** です。

notimestamp は、コメント部分の日時の出力を抑制します (バージョン管理システムを使っている場合、日付だけ違うものを新しいバージョンとして登録したくはないでしょう)。

color (デフォルト) は、カラー描画を生成しますが、**monochrome** は一切 special を入れません。白黒プリンタ用には、その挙動をこんな風に変えた方がもっと良くなるというアイデアを持っている人は、是非提案してください。

rounded (デフォルト) と **mitered**, **beveled** は、線分の接合部の形状を制御し、**round** (default) と **butt**, **squared** は、線分の端の形状を制御します。詳細は、PostScript か PDF のリファレンスマニュアルを参照してください。激しく変化する関数と太い線用には、線分の接合部での尖った角を避けるように **rounded** と **round** を使うといいでしょう。(これに関する一般的な仕組みは、このオプションを各描画スタイル毎に別々に指定できるよう gnuplot がサポートすべきだと思います。)

dashed (デフォルト) は、異なる線種に異なる点線パターンを使い、**solid** は、すべての描画に実線を使用します。

dashlength (または **dl**) は、点線の線分の長さを <dl> 倍します。**linewidth** (または **lw**) は、すべての線幅を <lw> 倍します。(lw 1 は 0.5bp を意味し、これは Metapost の描画のデフォルトの線幅です) **fontscale** は、テキストラベルをデフォルトの文書フォントの <fontscale> 倍に拡大します。

mppoints は、Metapost で描画された定義済みの点の形状を使用します。**texpoints** は、簡単に設定できる記号セットを使用します。これは、以下のようにして ConTeXt で定義できます:

```
\defineconversion[my own points][+,{\ss x},\mathematics{\circ}]
\setupGNUPLOTterminal[context][points=tex,pointset=my own points]
```

inlineimages は、バイナリ画像を文字列として書き出しますが、これは ConTeXt MKIV のみで機能します。

externalimages は、PNG ファイルを外部出力し、これは ConTeXt MKII で機能します。これが動作するためには、gnuplot が PNG 画像出力をサポートしている必要があります。

standalone のグラフでは、**font** でフォント名とサイズを設定できます。standalone でないモード (**input**) では、テキストラベルに十分なスペースを割り当てるためにフォントサイズのみが意味を持ちますコマンド

```
set term context font "myfont,ss,10"
```

は、以下のようになります:

```
\setupbodyfont[myfont,ss,10pt]
```

例えばさらに追加で **fontscale** を 0.8 に設定すると、結果としてフォントは 8pt の大きさになり、

```
set label ... font "myfont,12"
```

は 9.6pt になります。

適当なタイプスクリプトフォント (とヘッダー) を用意するのは自分で行ってさもなくばフォントの切り替えは効果を持ちません。ConTeXt MKII (pdfTeX) の標準フォントは、以下のようにして使えます:

```
set terminal context standalone header '\usetypescript[iwona][ec]' \
    font "iwona,ss,11"
```

フォントの利用に関する ConTeXt の最新の情報については、ConTeXt の文書、wiki、メーリングリスト (アーカイブ) を探してみてください。

例:

```
set terminal context size 10cm, 5cm     # 10cm, 5cm
set terminal context size 4in, 3in      # 4in, 3in
```

UTF-8 エンコードラベルを standalone (ページ全体) グラフで使用するには:

```
set terminal context standalone header '\enableregime[utf-8]'
```

Requirements

ConTeXt 用の gnuplot モジュール: http://ctan.org/pkg/context-gnuplot

と、最新の ConTeXt が必要です。そこから gnuplot を呼び出すには、write18 を可能にする必要があります。ほとんどの TeX 配付物では、これは texmf.cnf で shell_escape=t とすることで設定できます。

この出力形式とより詳しいヘルプと例に関しては以下も参照してください: http://wiki.contextgarden.net/Gnuplot

Calling gnuplot from ConTeXt

ConTeXt 文書でグラフを作成する最も簡単な方法は以下の通り:

```
\usemodule[gnuplot]
\starttext
\title{How to draw nice plots with {\sc gnuplot}?}
```

```
\startGNUPLOTscript[sin]
set format y "%.1f"
plot sin(x) t '$\sin(x)$'
\stopGNUPLOTscript
\useGNUPLOTgraphic[sin]
\stoptext
```

これは自動的に gnuplot を実行し、その結果の画像を文書中に取り込みます。

Debug

このドライバは **gnuplot** のデバッグのために提供されているものです。おそらくソースコードを修正するユーザのみが使用するものでしょう。

Domterm

書式:

```
 set terminal domterm
              {font "<fontname>{,<fontsize>}"} {{no}enhanced}
              {fontscale <multiplier>}
              {rounded|butt|square} {solid|dashed} {linewidth <lw>}
              {background <rgb_color>}
              {animate}
```

domterm 出力形式は、domterm, qtdomterm プログラム等の DomTerm ターミナルエミュレータ上で動作します。これは、ターミナル出力内へ直接 SVG グラフ画像を埋め込むことをサポートしています。http://domterm.org を参照してください。

terminal オプションについては、**svg** 出力形式を参照してください。

Animate

```
set term domterm animate
```

オプション **animate** は、カーソル位置をすべてのグラフの最初の、出力形式左上に戻し、後に続くグラフをスクリーンの同じ場所に上書きするようにします。これは、定位置のアニメーションを作成するためには望ましいでしょう。

Dumb

ダム端末 (**dumb**) ドライバは、ASCII 文字を使用してテキスト領域に描画します。サイズの指定と改行制御用のオプションがあります。

書式:

```
set terminal dumb {size <xchars>,<ychars>} {[no]feed}
                  {aspect <htic>{,<vtic>}}
                  {[no]enhanced}
                  {fillchar {solid|"<char>"}}
                  {[no]attributes}
                  {mono|ansi|ansi256|ansirgb}
```

<xchars>, <ychars> はテキスト領域のサイズを設定し、デフォルトは 79 x 24 となっています。最後の改行は、**feed** オプションが設定されている場合のみ出力されます。

オプション **aspect** は、グラフのアスペクト比の制御に使用します。これは水平軸、垂直軸の目盛り刻みの長さを設定します。整数の指定のみが許されていて、デフォルトは 2,1 で、これは共通のスクリーンフォントのアスペクト比に対応します。

文字 "#" を領域塗り潰しに使いますが、これを terminal フォントで利用できる任意の文字に置き換えることができます。**fillchar solid** は、**fillchar "\U+2588"** (Unicode の全角黒四角) の短縮形です。

オプション **ansi**, **ansi256**, **ansirgb** は色扱うために出力にエスケープシーケンスを挿入しますが、あなたのターミナルでは処理できないかもしれません。デフォルトは **mono** です。**ansi** モードでは、色は **set colorsequence classic** を使うのが一番合います。どのモードを使用するかによって、**dumb** 出力形式は、以下のシーケンスを出力します (空白を追加せずに):

```
ESC [ 0 m           属性をデフォルトにリセット
foreground color:
表示色:
ESC [ 1 m           強調/太字をセット
ESC [ 22 m          強調/太字をオフに
ESC [ <fg> m        30 <= <fg> <= 37 のカラーコード指定
ESC [ 39 m          デフォルトにリセット
ESC [ 38; 5; <c> m  パレットの番号指定 (16 <= <c> <= 255)
ESC [ 38; 2; <r>; <g>; <b> m  成分指定 (0 <= <r,g,b> <= 255)
背景色:
ESC [ <bg> m        40 <= <bg> <= 47 のカラーコード指定
ESC [ 49 m          デフォルトにリセット
ESC [ 48; 5; <c> m  パレットの番号指定 (16 <= <c> <= 231)
ESC [ 48; 2; <r>; <g>; <b> m  成分指定 (0 <= <r,g,b> <= 255)
```

例えば、以下の記述も参照してください: https://en.wikipedia.org/wiki/ANSI_escape_code#Colors

オプション **attributes** は、太字やイタリック体文字用のエスケープシーケンスをサポートする端末、エミュレーター上でそれらの文字出力を可能にします。

```
ESC [ 1 m / 22 m    太字 (bold) のオン/オフ
ESC [ 3 m / 23 m    イタリック体のオン/オフ
```

例:

```
set term dumb mono size 60,15 aspect 1
set tics nomirror scale 0.5
plot [-5:6.5] sin(x) with impulse ls -1
```

```
   1 +-------------------------------------------------+
 0.8 +|||++                   ++||||++                 |
 0.6 +|||||+                 ++|||||||+ sin(x) +-----+ |
 0.4 +||||||+               ++|||||||||+               |
 0.2 +|||||||+             ++|||||||||||+             +|
   0 ++++++++++++++++++++++++++++++++++++++++++++++++++|
-0.2 +        +|||||||||||+              +|||||||||||+ |
-0.4 +         +|||||||||+                +|||||||||+  |
-0.6 +          +|||||||+                  +|||||||+   |
-0.8 +           ++||||+                    ++||||+    |
  -1 +---+--------+--------+-------+--------+--------+-+
        -4       -2        0       2        4        6
```

Dxf

dxf は AutoCad (リリース 10.x) 用の出力ドライバです。このドライバにはオプションはありません。デフォルトの大きさは AutoCad の単位での 120x80 です。**dxf** は 7 色 (白、赤、黄、緑、水色、青、赤紫) を使いますが、これを変更するにはドライバソースファイルを修正する必要があります。白黒の出力装置を使う場合、それらの色は線の太さの違いで表現されます。注意: どなたか、この出力ドライバを DXF 規格 2012 に更新してください！

Emf

emf ドライバは EMF (Enhanced Metafile Format) ファイルを生成します。この形式のファイルは多くの MS-Windows アプリケーションで認識できます。

書式:

```
      set terminal emf {color | monochrome}
                       {enhanced {noproportional}}
                       {rounded | butt}
                       {linewidth <LW>} {dashlength <DL>}
                       {size XX,YY} {background <rgb_color>}
                       {font "<fontname>{,<fontsize>}"}
                       {fontscale <scale>}
```

monochorome モードは折れ線を点線のパターンを循環させて打ち出します。**linewidth <factor>** は全ての線幅をここで指定する値倍にします。**dashlength <factor>** は、太い線には便利でしょう。**<fontname>** はフォント名、**<fontsize>** はポイント単位でのフォントの大きさです。

出力画像の形式的な (名ばかりの) サイズは、デフォルトでは適当な単位での 1024x768 になっています。オプション **size** を使って別な形式的なサイズを指定できます。

拡張文字列処理モード (enhanced text mode) は、プロポーショナル文字間隔を近似しようとします。モノスペースフォントを使う場合、あるいはこの近似を好まない場合、オプション **noproportional** を使うことでこの補正をオフにできます。

デフォルトの設定は、**color font "Arial,12" size 1024,768** で、**default** を選択すると全てのオプションがそのデフォルトの値になります。

例:

```
set terminal emf 'Times Roman Italic, 12'
```

Epscairo

出力形式 **epscairo** は、cairo, pango ライブラリを用いて EPS 出力 (Encapsulated PostScript) を生成します。cairo は version 1.6 以降が必要です。

詳細は、**pdfcairo** 出力形式のヘルプを参照してください。

Epslatex

epslatex ドライバは LaTeX で処理すべき出力を生成します。

書式:

```
set terminal epslatex   {default}
set terminal epslatex   {standalone | input}
                        {level1 | leveldefault | level3}
                        {color | colour | monochrome}
                        {background <rgbcolor> | nobackground}
                        {dashlength | dl <DL>}
                        {linewidth | lw <LW>} {pointscale | ps <PS>}
                        {rounded | butt}
                        {clip | noclip}
                        {palfuncparam <samples>{,<maxdeviation>}}
                        {size <XX>{unit},<YY>{unit}}
                        {header <header> | noheader}
                        {blacktext | colortext | colourtext}
                        {{font} "fontname{,fontsize}" {<fontsize>}}
                        {fontscale <scale>}
```

epslatex 出力形式は、文字列を PostScript コードに含ませる代わりに LaTeX ファイルに移すことを除けば **terminal postscript eps** 同様に描画します。よって、**postscript terminal** と多くのオプションが共通です。

以下のようなエラーメッセージが出た場合:

```
"Can't find PostScript prologue file ... "
```

以下参照: **postscript prologue (p. 382)**。そしてその指示に従ってください。

オプション **color** はカラーを有効にし、**monochrome** は各要素を黒と白描画します。さらに、**monochrome** は灰色の **palette** も使用しますが、これは、明示的に **colorspec** で指定された部品の色を変更しません。

dashlength または **dl** は点線の線分の長さを <DL> (0 より大きい実数) に設定し、**linewidth** または **lw** は全ての線の幅を <LW> に設定します。

デフォルトでは、生成される PostScript コードは、特にフィルタリングや filledcurves のようなでこぼこな領域のパターン塗りつぶしにおいて、PostScript Level 2 として紹介されている言語機能を使います。PostScript Level 2 の機能は条件的に保護されていて、PostScript Level 1 のインタープリタがエラーを出さず、むしろメッセージか PostScript Level 1 による近似であることを表示するようになっています。**level1** オプションは、これらの機能を近似する PostScript Level1 で代用し、PostScript Level 2 コードを一切使用しません。これは古いプリンタや、Adobe Illustrator の古いバージョンなどで必要になるかもしれません。このフラグ **level1** は出力された PostScript ファイルのある一行を手で編集することで、後から強制的に PostScript Level 1 機能を ON/OFF にすることもできます。level 2 のコードが含まれている場合、上の機能は現われないか、このフラグがセットされた場合、あるいは PostScript インタプリタプログラムが level 2 以上の PostScript を解釈するとは言わなかった場合に警告文に置き換わります。**level3** オプションは、ビットマップ画像の PNG 化の機能を有効にします。それにより出力サイズをかなり削減できます。

rounded は、線の端や接合部を丸くし、デフォルトの **butt** は尖った端と角張った接合部を使用します。

clip は、PostScript にすべての出力を BoundingBox (PostScript の外枠) でクリップすることを指示します; デフォルトは **noclip** です。

palfuncparam は **set palette functions** から出力の傾きをどのようにコード化するかを制御します。解析的な色の成分関数 (**set palatte functions** で設定される) は、postscript 出力では傾きの線形補完を用いてコード化されます: まず色の成分関数が <samples> 個の点で標本化され、そしてそれらの点は、結果として線形補完との偏差が <maxdeviation> 以内に収まるように削除されます。ほとんど全ての有効なパレットで、デフォルトの <samples> =2000 と <maxdeviation>=0.003 の値をそのまま使うのが良いでしょう。

PostScript 出力のデフォルトの大きさは 10 インチ x 7 インチです。EPS 出力のデフォルトの大きさは 5 x 3.5 インチです。オプション **size** はこれらをユーザが指定したものに変更します。デフォルトでは X と Y のサイズの単位はインチとみなされますが、他の単位 (現在は cm のみ) も使うことはできます。描画の BoundingBox (PostScript ファイルの外枠) は、サイズが変更された画像を丁度含むように正しく設定されます。スクリーン座標は、オプション **size** で指定された描画枠の全体が 0.0 から 1.0 になります。

blacktext は、たとえカラーモードでも全ての文字列を黒で書きます。

epslatex ドライバは文字列の配置の制御に特別な方法を提供します: (a) '{' で始まる文字列は、'}' で閉じる必要がありますが、その文字列全体が LaTeX によって水平方向にも垂直方向にもセンタリングされます。(b) '[' で始まる文字列の場合は、位置の指定をする文字列 (t,b,l,r,c のうち 2 つまで) が続き、次に ']{'、文字列本体、で最後に '}' としますが、この文字列は LaTeX が LR-box として整形します。\rule{}{} を使えばさらに良い位置合わせが可能でしょう。ドライバ **pslatex** に関する説明も参照。複数行の見出しを作成するには \shortstack を使用してください。例えば、

```
set ylabel '[r]{\shortstack{first line \\ second line}}'
```

set label コマンドのオプション **back** は使えますが、他の出力形式のものとは少し違っています。**front** の場合の見出しが他の全ての要素の上に出力されるのに対して、**back** を使った見出しは他の全ての要素の下に出力されます。

このドライバは 2 つの別のファイルを作ります。1 つは図の eps の部分で、もう一つは LaTeX の部分です。LaTeX ファイルの名前は、**set output** コマンドのものが使われ、eps ファイルの名前はその拡張子 (通常 **.tex**) を**.eps** に置き換えたものになります。出力ファイルを指定しなければ LaTeX 出力は行なわれません！ **multiplot** モード以外では、次の描画を行なう前にその出力ファイルをクローズするのを忘れないでください。

LaTeX の文書で図を取り込むには '\input{filename}' としてください。**.eps** ファイルは \includegraphics{...} コマンドで取り込むので、よって LaTeX のプリアンブルに \usepackage{graphicx} も入れる必要があります。

textcolour オプションで色付きの文字列を使用している場合は、LaTeX のプリアンブルに \usepackage{color} も入れる必要があります。

この eps ファイルから 'epstopdf' を使って pdf ファイルを作ることもできます。graphics パッケージが適切に設定されている場合、その LaTeX のファイルは、変更なしに pdflatex によっても処理でき、その場合 eps ファイルの代わりに pdf ファイルが取り込まれます。

フォントの選択に関する挙動はヘッダーモードに依存します。どの場合でも、与えられたフォントサイズはスペースの計算にちゃんと使用されます。**standalone** モードが使われなかった場合は、include される場所での実際の LaTeX フォントとフォントサイズが使われるので、よってフォントの変更には LaTeX コマンドを使ってください。例えばフォントサイズとして LaTeX 文書中で 12pt を使う場合は、オプション '"" 12' を使います。この場合フォント名は無視されます。**standalone** を使う場合は、与えられたフォントとフォントサイズが使われます。詳細は下記を参照してください。

文字列がカラーで表示されるかどうかは TeX の Bool 値変数 \ifGPcolor と \ifGPblacktext で制御します。\ifGPcolor が true で \ifGPblacktext が false の場合のみ文字列はカラーで表示されます。それらは生成される TeX ファイルを変更するか、またはあなたの TeX ファイルで大域的に与えてください。例えば

```
\newif\ifGPblacktext
\GPblacktexttrue
```

をあなたのファイルのプリアンブルに書きます。部分的な指定は大域的な値が与えられていないときのみ働きます。

epslatex 出力形式を使う場合、**set output** コマンドで TeX ファイルの名前を拡張子付き (通常 ".tex") で与えてください。eps ファイルの名前はその拡張子を ".eps" に置き換えた名前になります。

standalone モードを使う場合、その LaTeX ファイルに完全な LaTeX のヘッダが付加され、eps ファイルのファイル名には "-inc" が追加されます。**standalone** モードは、dvips, pdfTeX, VTeX を使う場合に正しいサイズで出力されるような TeX ファイルを作ります。デフォルトは **input** モードで、これは \input コマンドを使って、別の

LaTeX ファイルから読み込まれるようなファイルを生成します。

"" か "default" 以外のフォント名が与えられた場合、それは LaTeX のフォント名と解釈されます。それは、'fontname,fontseries,fontshape' の、コンマで区切られた 3 つ以下の部分からなります。デフォルトのフォントシェイプ、フォントシリーズが使いたい場合はそれらは省略できます。つまり、フォント名に対する正式な書式は、'[fontname][,fontseries][,fontshape]' となります。そのいずれの部分も名前に関しては LaTeX のフォント体系の慣習に従います。fontname は 3 から 4 文字の長さで、次のような規則で作られています: 1 つ目がフォントの製造元を表し、次の 2 つがフォント名、オプションで追加される 1 つは特別なフォントを意味し、例えば 'j' は旧式の数字を持つフォント、'x' は expert フォント等となっています。以下には、多くのフォントの名前について書かれています。http://www.tug.org/fontname/fontname.pdf

例えば 'cmr' は Computer Modern Roman フォント、'ptm' は Times-Roman, 'phv' は Helvetica 等を表します。フォントシリーズは文字の線の太さを意味し、大半は 'm' で普通 ("medium")、'bx' または 'b' が太字 (bold) フォントを意味します。フォントシェイプは一般に 'n' が立体 (upright)、'it' はイタリック、'sl' は斜体 (slanted)、'sc' は小さい大文字 (small caps) となります。これらとは異なるフォントシリーズやフォントシェイプで与えられるフォントも存在します。

例:

Times-Roman で太字 (シェイプは周りの文字列と同じもの) を使う場合:

```
set terminal epslatex 'ptm,bx'
```

Helvetica で太字でイタリックを使う場合:

```
set terminal epslatex 'phv,bx,it'
```

斜体のシェイプで周りのフォントを使い続ける場合:

```
set terminal epslatex ',,sl'
```

小型の大文字 (small caps) を使う場合:

```
set terminal epslatex ',,sc'
```

この方法では文字列のフォントのみが変更されます。数式のフォントも変更したい場合は、"gnuplot.cfg" ファイルかまたは **header** オプションを使う必要がありますが、これについては以下に書きます。

standalone モードでは、フォントサイズは **set terminal** コマンドで与えられたフォントサイズが使われます。指定したフォントサイズが使えるためには LaTeX の検索パスに "size<size>.clo" というファイルが存在しなければなりません。デフォルトでは 10pt, 11pt, 12pt がサポートされています。パッケージ "extsizes" がインストールされていれば、8pt, 9pt, 14pt, 17pt, 20pt も追加されます。

オプション **header** は文字列引数を取ります。その文字列は、生成される LaTeX ファイルに書き込まれます。**standalone** モードを使う場合、その文字列はプリアンブルの \begin{document} コマンドの直前に書き込まれます。**input** モードでは、その文字列は \begingroup コマンドの直後に置かれ、描画への設定がすべて局所的になるようにします。

例:

T1 フォントエンコーディングを使い、文字列、数式フォントを Times-Roman とし、サンセリフフォントを Helvetica と変更する場合:

```
set terminal epslatex standalone header \
"\\usepackage[T1]{fontenc}\n\\usepackage{mathptmx}\n\\usepackage{helvet}"
```

描画の外の文字列には影響を与えないように描画内で太字 (boldface) フォントを使う場合:

```
set terminal epslatex input header "\\bfseries"
```

ファイル "gnuplot.cfg" が LaTeX によって見つけられると、**standalone** モードを使っている場合は、それは生成される LaTeX 文書のプリアンブルに取り込まれます。それは追加の設定を行なうのに使えます。例えば、文書のフォントを TImes-Roman, Helvetica, Courier と変更し、("mathptmx.sty" で扱われている) 数式フォントを入れる場合:

```
\usepackage{mathptmx}
\usepackage[scaled=0.92]{helvet}
\usepackage{courier}
```

ファイル "gnuplot.cfg" は **header** コマンドで与えられるヘッダ情報の前に読み込まれます。よって、"gnuplot.cfg" で行なわれる設定のいくつかを **header** を使って上書きすることができます。

Epson_180dpi

注意: gnuplot が –with-bitmap-terminals で作られた場合のみ使えます。このドライバはエプソンプリンタのいくつかとそれに類似するものをサポートします。

epson_180dpi と **epson_60dpi** はそれぞれ 180dpi (ドット/インチ), 60dpi の解像度の Epson LQ 型 24 ピンプリンタ用のドライバです。

epson_lx800 は Epson LX-800, Star の NL-10 や NX-1000, PROPRINTER などの適当なプリンタに流用できる、一般的な 9 ピンプリンタドライバです。

nec_cp6 は NEC CP6 や Epson LQ-800 などのプリンタで使える、一般的な 24 ピンプリンタ用のドライバです。

okidata ドライバは 9 ピンの OKIDATA 320/321 標準プリンタをサポートします。

starc ドライバは Star カラープリンタ用です。

tandy_60dpi ドライバは 9 ピン 60dpi の Tandy DMP-130 シリーズ用です。

dpu414 ドライバは Seiko DPU-414 感熱プリンタ用です。

nec_cp6 にはオプションがあります:

書式:

```
set terminal nec_cp6 {monochrome | colour | draft}
```

デフォルトでは白黒 (monochrome) です。

dpu414 にはオプションがあります:

書式:

```
set terminal dpu414 {small | medium | large} {normal | draft}
```

デフォルトは medium (= フォントサイズ) で normal です。おすすめの組み合わせは、**medium normal** か **small draft** です。

Fig

fig ドライバは、対話型描画ツール xfig に取り込める Fig グラフィック言語での出力を生成します。注意:

```
fig 出力形式は、gnuplot バージョン 5.3 で大きく改訂されました。
現在は fig ファイル形式バージョン 3.2 のみサポートしています。
点線/破線パターンを利用するには Xfig 3.2.6 以降が必要です。
```

書式:

```
set terminal fig {monochrome | color}
                 {small | big | size <xsize>{in|cm},<ysize>{in|cm}}
                 {landscape | portrait}
                 {pointsmax <max_points>}
                 {font "<fontname>{,<fontsize>}"} {fontsize <size>}
                 {textnormal | {textspecial texthidden textrigid}}
                 {{linewidth|lw} <multiplier>}
```

デフォルトの設定は、

```
set term fig color small landscape font "Times Roman,10" lw 1.0
```

size は描画範囲を <xsize>*<ysize> に、インチ単位 (デフォルト) かセンチ単位で設定 (変更) します。デフォルトは **size 5in,3in** です。**small** は **size 5in,3in** (portrait モードでは 3in,5in) の省略形、**big** は **size 8in,5in** の省略形です。

pointsmax は、折れ線内の頂点の最大数を設定し、長すぎる折れ線は、分割します。

font は、テキストフォントフェース名を <fontname> に、フォントサイズを <fontsize> ポイントに設定します。その選択は、35 の標準 PostScript フォントに限定されています。**textnormal** はテキストフラグをリセットして postscript フォントを選択し、**textspecial** はテキストフラグを LaTeX special に設定し、**texthidden**, **textrigid** はそれぞれ無表示のテキスト、スケーリングされないテキスト用のフラグを設定します。

linewidth は、すべての線に対する linewidth の値への倍率です。

fig ドライバには **plot** コマンドの **point** スタイルの記号が追加されています。記号の指定は (**pointtype** の値) % 100 の 50 以上の値が使われ、その塗りつぶしの濃さは <pointtype> % 5 の値で制御し、その輪郭は黒 (<pointtype> % 10 < 5 の場合) または現在の色で書かれます。利用可能な記号は以下の通りです。

```
50 - 59:  円
60 - 69:  正方形
70 - 79:  ひし形
80 - 89:  上向きの三角形
90 - 99:  下向きの三角形
```

これらの記号の大きさはフォントの大きさで変更できます。

RGB 色は、それが描画の前に線種として定義されているものではない場合、または登録されている色名やパレット値に一致しなかった場合は灰色で置き換えられます。以下参照: **colornames (p. 207)**。例:

```
set linetype 999 lc rgb '#aabbcc'
plot $data with fillecurve fillcolor rgb '#aabbcc'
```

Gif

書式:

```
set terminal gif
       {{no}enhanced}
       {{no}transparent} {rounded|butt}
       {linewidth <lw>} {dashlength <dl>}
       {tiny | small | medium | large | giant}
       {font "<face> {,<pointsize>}"} {fontscale <scale>}
       {size <x>,<y>} {{no}crop}
       {background <rgb_color>}
       {animate {delay <d>} {loop <n>} {optimize}}
```

PNG, JPEG, GIF 画像は、外部ライブラリ libgd を使って生成されます。GIF の描画は、ImageMagick パッケージのソフト 'display' にその出力を以下のようにパイプで渡すことで対話的に表示させることができます:

```
set term gif
set output '| display gif:-'
```

次の描画コマンドからの出力は、display ウィンドウ上で対話的に <space> を打つことで見ることができます。現在の描画をファイルに保存するには、display ウィンドウで左クリックし、**save** を選択してください。

transparent は、ドライバに背景色の透明化 (transparent) を行うよう指示します。デフォルトは **notransparent** です。

オプション **linewidth** と **dashlength** は拡大率で、描画されるすべての線に影響を与えます。すなわち、これらは様々な描画コマンドで要求される値にかけ算されます。

butt は線分の描画で、その端の点でのはみだしを起こさない描画メソッドを使うようドライバに指示します。この設定は、線幅が 1 より大きい場合にのみ有効です。この設定は、水平線、垂直線の描画の場合に有用でしょう。

出力描画サイズ <x,y> はピクセル単位で与えます。デフォルトは 640x480 です。以下も参照: **canvas (p. 39)**, **set size (p. 278)**。描画終了後の端の余白は、オプション **crop** で取り除くことができ、その結果としてその画像サイズは小さくなります。デフォルトは **nocrop** です。

Animate

```
set term gif animate {delay <d>} {loop <n>} {{no}optimize}}
```

gif 出力形式のオプション **animate** は、複数のフレームからなる単一の gif ファイルを生成します。各画像間の表示間隔は 1/100 秒単位で指定できますが (デフォルトは 5)、、後でそのアニメーションを見るのに使うプログラムが正確に再現するかもしれませんし、しないかもしれません。アニメーションの繰り返し回数も指定できますが、デフォルトは 0 で、それは無限の繰り返しを意味します。改めて言いますが、この値もそれを見るのに使うプログラムが正確に再現するかもしれませんし、しないかもしれません。アニメーション画像列は、次の **set output** か **set term** コマンドによって終了します。

連続する回転を表示する例:

```
set term gif animate loop 0
set output 'rotating_surface.gif'
do for [ang=1:359] {
    set view 60, ang
    splot f(x,y) with pm3d
}
unset output
```

Optimize

```
set term gif animate optimize
```

[非推奨] オプション **optimize** は、出力ファイルを開くときに gd ライブラリに渡します。これは、アニメーションに 2 つの効果を持ちます。

1) アニメーション全体を通じて単一のカラーマップが使用されます。これはアニメーションの全てのフレームで使用される全ての色が最初のフレームで定義されている必要があります。

2) 可能ならば、個々のフレームで一つ前のフレームと違う部分のみがアニメーションファイルに保存されます。これはファイルサイズを小さくしてくれますが、透明化機能を使用している場合には働かないかもしれません。

これら両方の最適化はより小さいサイズの出力ファイルを作ろうとするものですが、多分その減少量は、長いアニメーションのみ意味がある程度でしょう。警告: libgd の最適化の実装はおかしいことが知られていますので、gnuplot でこのオプションを使用することは推奨しません。

Fonts

フォントの選択の詳細は、やや複雑です。より詳しい情報については、**fonts libgd** の下の該当するセクションを参照してください。

例:

```
set terminal gif medium noenhanced size 640,480 background '#ffffff'
```

この例は medium サイズの、大きさ変更不能で回転できない組み込みフォントを使用します。拡張文字処理 (enhanced text) モードは、このフォントでは機能しません。そして、透明化されない背景色として白 (16 進数の 24bit RGB) を使用します。

```
set terminal gif font arial 14
```

これは、'arial' というフェース名のフォントを検索し、フォントサイズを 14pt に設定します。

Gpic

注意: 古い (legacy) 出力形式 (configure で –with-gpic としたときのみ)。**gpic** ドライバは FSF (the Free Software Foundations) の "groff" パッケージの中の GPIC 形式のグラフを生成します。デフォルトの大きさは 5 x 3 インチです。オプションは原点に関するもののみで、デフォルトでは (0,0) です。

書式:

```
set terminal gpic {<x> <y>}
```

ここで **x** と **y** の単位はインチです。

単純なグラフを整形するには以下のようにします。

```
groff -p -mpic -Tps file.pic > file.ps
```

pic からの出力はパイプで eqn に渡すこともできるので、'set label' と **set {x/y}label** コマンドでグラフに複雑な関数の式を入れることも可能です。例えば、

```
set ylab '@space 0 int from 0 to x alpha ( t ) roman d t@'
```

とすれば、以下のコマンドによって y 軸に綺麗な積分が見出し付けされます。

```
gpic filename.pic | geqn -d@@ -Tps | groff -m[macro-package] -Tps
    > filename.ps
```

このようにして作られた図は文書に綺麗に当てはまるように伸縮することができます。pic 言語は簡単に理解できるので、必要なら容易にグラフを直接編集できます。**gnuplot** で作られる pic ファイルの全ての座標は x+gnuplotx, y+gnuploty の形で与えられます。デフォルトでは x, y の値は 0 です。いくつかのファイルに対してその x,y を 0 と設定している行を削除すれば、以下のようにして複数のグラフを一つの図の中に入れてしまうこともできます (デフォルトの大きさは 5.0x3.0 インチ):

```
.PS 8.0
x=0;y=3
copy "figa.pic"
x=5;y=3
copy "figb.pic"
x=0;y=0
copy "figc.pic"
x=5;y=0
copy "figd.pic"
.PE
```

これは、横に 2 つ、縦に 2 つずつ並んだ 4 つのグラフからなる、8 インチの広さの図を生成します。

以下のように x, y を指定することでも同じことができます。

```
set terminal gpic x y
```

Grass

注意: 古い (legacy) 出力形式です。**grass** ドライバは GRASS 地理情報システムのユーザが **gnuplot** を利用することを可能にします。詳しい情報については grassp-list@moon.cecer.army.mil に連絡を取ってください。ページは GRASS グラフウィンドウの現在のフレームに書かれます。オプションはありません。

HP terminals

gnuplot は、Hewlett-Packard 社の古いペンプロッタやプリンタ用に 2 つの包括的な出力形式を提供します。HPGL プリンタ制御言語は 1974 年に導入され、以後多くのプロッタ、プリンタが認識します。以下参照: **set term hpgl (p. 363)**。PCL5 プリンタ制御言語は 1990 年に導入され、HP や他の多くのデバイスの標準になりました。以下参照: **set term pcl5 (p. 371)**。この 2 つの出力形式は、gnuplot にデフォルトで入っています。

特定のプリンタ (hp26 hp2648), インクジェットプリンタ群 (hp500c)、初期のレーザープリンタ (hplj hpljii) をサポートする古い出力形式もありますが、それらはソースコードファイル **term.h** を修正しなければ使えません。

Hpgl

書式:

```
set terminal hpgl {<number_of_pens>} {eject} {fontscale <scale>}
```

hpgl ドライバは、1970 年代以降の HP7475A や多くの他のプロッタのような Hewlett Packard 社製のペンプロッタ用の HPGL 出力を行ないます。HPGL グラフィックデータは、多くのソフトウェアで取り込むこともできます。HPGL コマンド言語は、その後のプリンタでは広く PCL コマンド言語に置き換えられました。以下参照: **set term pcl5 (p. 371)**。

terminal オプションは、使用するペンの数と、終了時にプロッタにページを排出 (eject) させるかどうかを制御します。デフォルトでは、6 つのペンを使い、描画後のページの排出は行ないません。

すべての文字は一様なサイズで描画されます。オプション **fontscale** はこのサイズを大きく、または小さくするための積因子です。gnuplot の現在のエンコーディングが iso_8859_1 か cp850 のいずれかにセットされていれば、非アスキー文字を、対応する文字セットをサポートするいくつかのプリンタモデルで処理することで変換することができます。あなたのプリンタモデルがこれをサポートしていなければ、これらのエンコーディングはセットできません。

Imagen

imagen ドライバは Imagen レーザプリンタをサポートします。これは 1 ページに複数のグラフを配置することも可能です。

書式:

```
set terminal imagen {<fontsize>} {portrait | landscape}
                    {[<horiz>,<vert>]}
```

fontsize はデフォルトでは 12 ポイントで、レイアウトのデフォルトは **landscape** です。**<horiz>** と **<vert>** はグラフを横方向と縦方向に何列置くかを指定します。これらのデフォルトは 1 です。

例:

```
set terminal imagen portrait [2,3]
```

これは、1 ページに 6 つのグラフを横に 2 列、縦に 3 列、縦置き (portrait) で配置します。

Jpeg

書式:

```
set terminal jpeg
       {{no}enhanced}
       {{no}interlace}
       {linewidth <lw>} {dashlength <dl>} {rounded|butt}
       {tiny | small | medium | large | giant}
       {font "<face> {,<pointsize>}"} {fontscale <scale>}
       {size <x>,<y>} {{no}crop}
       {background <rgb_color>}
```

PNG, JPEG, GIF 画像は、外部ライブラリ libgd を使って生成されます。大抵の場合、単一の描画なら PNG の方が向いていて、GIF はアニメーション用です。それらは損失の少ない画像形式で、損失のある JPEG 形式よりも上質の画像を生成します。これは、特にベタ塗り潰した背景でのカラーの実線、つまりまさに gnuplot が生成する典型的な画像に対しては注意すべきことです。

オプション **interlace** は、プログレッシブ JPEG 画像を生成します。デフォルトは **nointerlace** です。

オプション **linewidth** と **dashlength** は拡大率で、描画されるすべての線に影響を与えます。すなわち、これらは様々な描画コマンドで要求される値にかけ算されます。

butt は線分の描画で、その端の点でのはみだしを起こさない描画メソッドを使うようドライバに指示します。この設定は、線幅が 1 より大きい場合にのみ意味があります。この逆は **rounded** です。

フォントの選択の詳細は、やや複雑です。以下に同じ意味を持つ簡単な例を示します:

```
set term jpeg font arial 11
set term jpeg font "arial,11"
```

より詳しい情報については、**fonts** の下の該当するセクションを参照してください。

出力描画サイズ <x,y> はピクセル単位で与えます。デフォルトは 640x480 です。以下も参照: **canvas (p. 39)**, **set size (p. 278)**。描画終了後の端の余白は、オプション **crop** で取り除くことができ、その結果としてその画像サイズは小さくなります。デフォルトは **nocrop** です。

Kittycairo

出力形式 **kittycairo** は、kitty グラフィックプロトコルをサポートする端末エミュレータ上で、ウィンドウ内出力を生成します。実際の描画は、2 次元グラフィックライブラリの cairo と文字列レンダリングライブラリの pango を用いて行います。kitty プロトコルは、sixel グラフィックとは別の選択肢で、24 ビット RGB カラーをサポートすること、およびリモートセッションではコンピュータと端末の間の画像データの転送に必要な大域幅が少し小さいことが優位です。

書式:

```
set terminal kittycairo
             {{no}enhanced} {mono|color}
             {font <font>} {fontscale <scale>}
             {linewidth <lw>} {rounded|butt|square} {dashlength <dl>}
             {transparent | background <rgbcolor>}
             {size <XX>,<YY>} {anchor|scroll}
```

この出力形式は、ラベルや他の文字列処理にデフォルトで拡張文字列処理を使用します。以下参照: **enhanced (p. 42)**。

グラフのすべての線分の幅は、**linewidth** の積因子 <lw> で変更できます。フォントサイズも同様に **fontscale** のスケール積因子で一様に変更できます。フォントや文字コードオプションに関する説明については、以下参照: **pdfcairo (p. 372)**。

オプション **rounded** は、線の端や接合部を丸くし、デフォルトの **butt** は尖った端と角張った接合部を生成します。

グラフのサイズは、スクリーンピクセル単位で与えます。デフォルトでは (**anchor**)、それぞれのグラフは、端末ウィンドウの左上に描きます。これはアニメーションや、キーボードを利用した疑似マウス操作 (以下参照: **pseudo-mousing (p. 155)**) では便利です。一方、**scroll** では、各グラフを現在のカーソル位置から開始しますが、それにより端末文字列でそれをスクロールできます。

Kittygd

書式:

```
set terminal kittygd
             {{no}enhanced} {{no}truecolor} {rounded|butt}
             {linewidth <lw>} {dashlength <dl>}
             {font "<face> {,<pointsize>}"} {fontscale <scale>}
             {size <x>,<y>} {anchor|scroll}
             {background <rgb_color>}
```

出力形式 **kittygd** は、kitty グラフィックプロトコルをサポートする端末エミュレータ上で、ウィンドウ内出力を生成します。デフォルトでは、ライブラリは 24 ビット RGB PNG 画像を生成し、出力時に 256 色 (**truecolor**) 減色変換されます。**notruecolor** は、出力の色をさらに少なく制限しますがこれによる明確な優位性はありません。透過型の fill style にはオプション **truecolor** が必要です。以下参照: **fillstyle (p. 283)**。あなたの gnuplot が、cairo グラフィックスをサポートするようにビルドされたものなら、むしろ **kittycairo** の方がいいでしょう。

この出力形式は、ラベルや他の文字列処理にデフォルトで拡張文字列処理を使用します。以下参照: **enhanced (p. 42)**。

グラフのすべての線分の幅は、**linewidth** の積因子 <lw> で変更できます。フォントサイズも同様に **fontscale** のスケール積因子で一様に変更できます。フォントや文字コードオプションに関する説明については、以下参照: **png (p. 375)**。

butt は線分の描画で、その端の点でのはみだしを起こさない描画メソッドを使うようドライバに指示します。この設定は、線幅が 1 より大きい場合にのみ意味があります。これの逆が **rounded** で、多少より均一な曲線を生成しますが、より遅くなります。

グラフのサイズは、スクリーンピクセル単位で与えます。デフォルトでは (**anchor**)、それぞれのグラフは、端末ウィンドウの左上に描きます。これはアニメーションや、キーボードを利用した疑似マウス操作 (以下参照: **pseudo-mousing (p. 155)**) では便利です。一方、**scroll** では、各グラフを現在のカーソル位置から開始しますが、それにより端末文字列でそれをスクロールできます。

Latex

gnuplot は、TeX/LaTeX での利用のための様々な出力形式を提供しています。

(1) PostScript の使用に基づく TeX/LaTeX 互換の出力形式以下参照: **epslatex (p. 355)**, **pslatex (p. 382)**, **pstricks (p. 384)**。

(2) cairo グラフィックに基づく TeX/LaTeX 互換の出力形式以下参照: **cairolatex (p. 342)**。

(3) **tikz** 出力形式は、外部 lua スクリプト (以下参照: **lua (p. 367)**) を用いて PGF と TikZ パッケージ用のファイルを生成します。出力形式オプションを見るには、**set term tikz help** としてください。

(4) **pict2e** 出力形式 (バージョン 5.4 で追加) は昔の gnuplot の古い出力形式 **latex**, **emtex**, **eepic**, **tpic** に置き換わるものです。以下参照: **pict2e (p. 373)**。

(5) その他は、以下参照: **context (p. 349)**。および古い出力形式の以下も参照: **texdraw (p. 390)**。

TeX 系出力形式に関する要約が以下にあります: http://www.gnuplot.info/docs/latex_demo.pdf

Linux cosole

古いバージョンの gnuplot では、linux コンソール上、すなわち X11 がないか、または他のウィンドウ環境でグラフィックスを表示するためには **linux** か **vgagl** の特別な出力形式が必要でしたが、これらの出力形式は既に廃棄されています。

現在、gnuplot を linux コンソールで実行するための推奨する方法は、sixel グラフィックをサポートする yaft (https://github.com/uobikiemukot/yaft) のようなコンソールターミナルエミュレータを使うことです。yaft をコンソールターミナルとして使用すれば、gnuplot を実行し、sixel 出力を行う出力形式を選択できます。以下参照: **sixelgd (p. 387)**。最終的な選択肢として、**set term dumb** を使用することも可能ですが、sixel グラフィックスの方がはるかにいいでしょう。

Lua

この **lua** 出力ドライバは、対象先指定描画ファイルを作成するための、外部 Lua スクリプトとの組み合わせで機能します。現在サポートしている対象は、TikZ -> pdflatex のみです。

Lua に関する情報は、http://www.lua.org で参照できます。

書式:

```
set terminal lua <target name> | "<file name>"
                    {<script_args> ...}
                    {help}
```

スクリプト用に 'target name'、または引用符付きの 'file name' が必須です。スクリプトの 'target name' を与えた場合は、この出力形式は、"gnuplot-<target name>.lua" をまずローカルディレクトリで探し、それに失敗すると環境変数 GNUPLOT_LUA_DIR を探します。

その他のすべての引数は、選択したスクリプトに評価させるように与えられます。例えば、'set term lua tikz help' は、スクリプトそれ自身に、スクリプト用のオプションと選択に関する追加のヘルプを表示させます。

Mp

注意: 古い (legacy) 出力形式 (デフォルトではビルドされない)。**mp** ドライバは Metapost プログラムへの入力を意図した出力を生成し、その出力ファイルに対して Metapost を実行するとグラフを含む EPS ファイルが作られます。デフォルトでは Metapost は全ての文字列を TeX に通します。これはタイトルや見出しに任意の TeX の記号を本質的に使うことができる、という利点を持つことを意味します。この出力形式は、gnuplot バージョン 4 以降十分には更新されておらず、そのためその後追加された機能の多くはサポートしていません。

書式:

```
set term mp {color | colour | monochrome}
            {solid | dashed}
            {notex | tex | latex}
            {magnification <magsize>}
            {psnfss | psnfss-version7 | nopsnfss}
            {prologues <value>}
```

```
{a4paper}
{amstex}
{template "<outputtemplate>"}
{"<fontname> {,<fontsize>}"}
```

オプション **color** は線をカラーで書くことを意味し (それをサポートするプリンタやディスプレイ上で)、**monochrome** (または何も指定しない場合) は黒の線が選択されます。オプション **solid** は線を実線で描き、**dashed** (または無指定) は線を異なるパターンの点線で描き分けます。**solid** が指定されてかつ **color** が指定されなかった場合、ほとんど全ての線が同じものになりますが、これも何かの場合には有用でしょうから認められています。

オプション **notex** は完全に TeX を迂回しますので、このオプションの元では見出しには TeX のコードは使うことができません。これは、古いグラフファイル、あるいは TeX では特殊記号として解釈されてしまう $ や % のような一般的な文字をたくさん使うファイルのために用意されています。

オプション **tex** は、TeX で処理する文字列を出力するように設定します。

オプション **latex** は、LaTeX で処理する文字列を出力するように設定します。これによって TeX では使えないけれど LaTeX では使えるもの、例えば分数を \frac で書いたりすることができます。このオプションを使う場合は、環境変数 TEX に LaTeX の実行プログラム名 (通常は latex) を設定するか、あるいは **mpost –tex=<LaTeX の実行プログラム名 > ...** とすることを忘れないでください。そうでないと metapost はテキストの処理に TeX を使おうとして失敗してしまうでしょう。

TeX におけるフォントサイズの変更は数式には効果がなく、そのような変更を行なうとても簡単な方法は、大域的に拡大率 (magnification factor) を設定する以外にはありません。それがオプション **magnification** の意味です。その場合は拡大率を後ろに指定する必要があります。全ての文字 (グラフではなく) はこの率で拡大されます。数式をデフォルトの 10pt 以外の他のサイズで書きたい場合はこれを使用してください。ただ残念なことに全ての数式が同じサイズになってしまいますが、しかし、以下の MP 出力の編集に関する説明を参照してください。**mag** は **notex** の元でも働きますが、それを行なう意味がないくらい (以下に述べる) フォントサイズオプションはうまく働きます。

オプション **psnfss** は postscript フォントを LaTeX と組み合わせて使用します。このオプションは LaTeX が使われる場合のみ意味を持ちますので、自動的に **latex** オプションが選択されます。このオプションは以下の LaTeX パッケージを使用します: inputenc(latin1), fontenc(T1), mathptmx, helvet(scaled=09.2), courier, latexsym, textcomp

オプション **psnfss-version7** も postscript フォントを LaTeX と組み合わせて使用します (**latex** が自動的に選択されます) が、以下の LaTeX パッケージを使用します: inputenc(latin1), fontenc(T1), times, mathptmx, helvet, courier

オプション **nopsnfss** はデフォルトで、標準的なフォント (何も指定されていなければ cmr10) が使われます。

オプション **prologues** は追加の値を引数に持ち、metapost ファイルに **prologues:=**< その値 > という行を追加します。値として **2** を指定すると metapost は eps ファイルを作るように postscript フォントを使用し、それによりその結果は例えば ghostscript などで参照できるようになります。標準では metapost は TeX のフォントを使用しますので、それを参照するには (La)TeX のファイルに取り込む必要があります。

オプション **noprologues** はデフォルトで、prologue で指定したいかなる行も追加されません。

オプション **a4paper** は [a4paper] を documentclass に追加します。標準では letter 用紙 (デフォルト) が使われます。このオプションは LaTeX でのみ使われますので、自動的に **latex** オプションが選択されます。

オプション **amstex** は、自動的に **latex** オプションを選択し、以下の LaTeX パッケージを使用します: amsfonts, amsmath(intlimits)。デフォルトではこれらは使用されません。

オプション **template** は、**outputtemplate:="<outputtemplate>";** という行をヘッダ行の後に追加します。ここで "<outputtemplate>" はこのオプションの必須の引数です。これにより、出力する metapost ファイルのファイル名が決定されます。例えば, "%j.mps" を指定すると、デフォルトの **.0**, **.1** 等のファイル名ではなく拡張子 **.mps** のファイル名を生成します。複数の出力ファイルを生成するときはテンプレートに "%c" を入れてください。テンプレートを指定しない場合は、"%j.%c" の指定と同等になります。

引用符で囲まれた名前はフォント名を表し、**set label** や **set title** で明示的にフォントが与えられない場合はこのフォントが使われます。フォントは TeX が認識できる (TFM ファイルが存在する) ものを使う必要があります。デフォルトでは **notex** が選択されていなければ "cmr10" が、そうでなければ "pcrr8r" (Courier) が使われます。**notex** の元でも、Metapost には TFM ファイルは必要です。**pcrr8r.tfm** は LaTeX psnfss パッケージの Courier フォント名として与えられています。**notex** のデフォルトからフォントを変更する場合は、少なくとも 32-126 のコード範囲は ASCII エンコーディングに一致するものを選んでください。**cmtt10** もほぼ使えますが、しかしこれはコード 32 (スペース) にスペースではない文字が入っています。

サイズは 5.0 から 99.99 の間の任意の数字を指定でき、省略された場合は 10.0 が使われます。なるべく **magstep** サイズ、つまり 1.2 の整数かまたは 0.5 乗の 10 倍を小数以下 2 桁未満を丸めた値を使用することをお勧めします。それはそれが TeX のシステムで最もよく使われるフォントのサイズだからです。

全てのオプションは省略可能です。フォントを指定する場合はそれは (必要ならサイズもつけて) 一番最後に指定する必要があります。フォント名にそのサイズ情報が含まれていたとしても、サイズを変えるにはフォントサイズを指定する必要があります。例えば **set term mp "cmtt12"** は cmtt12 をデフォルトのサイズである 10 に縮めて使います。それは多分望まないことでしょうし cmtt10 を使う方が良いでしょう。

以下の ascii 文字は、TeX では特別に扱われます:

```
$, &, #, %, _;  |, <, >;  ^, ~,  \, {, }
```

$, #, &, _, % の 5 つは、例えば \$ とすることで容易にそれをエスケープできます。<, >, | の 3 つは、例えば $<$ のように数式モードに入れてやればうまくいきます。残りのものに関しては少し TeX の回避策が必要になりますが、適当なよい TeX の本がそれを指導してくれるでしょう。

Metapost の画像は TeX の文書内で一般に使われています。Metapost はフォントを TeX が行なうのと全く同じ方法で扱い、それは他の大抵の文書整形プログラムとは異なっています。グラフが LaTeX の文書に graphics パッケージで取り込まれ、あるいは epsf.tex を使って plainTeX に取り込まれ、そして dvips (または他の dvi から ps への変換ソフト) で PostScript に変換される場合、そのグラフ内の文字は大抵は正しく扱われているでしょう。しかし、Metapost 出力をそのまま PostScript インタプリタに送っても、グラフ内の文字は出力されないでしょう。

Metapost の使い方

- まず terminal ドライバを Metapost に設定、例えば:

```
set terminal mp "cmtt12" 12
```

- 出力ファイルを選択、例えば:

```
set output "figure.mp"
```

- グラフを作成。各 plot (または multiplot の各グループ) はそれぞれ別な Metapost beginfig...endfig グループに分けられます。そのデフォルトのサイズは 5x3 インチですが、それは **set size 0.5,0.5** とか、そうしたいと思う適当な割合をそのように指定することで変更できます。

- gnuplot を終了。

- gnuplot の出力ファイルに対して Metapost を実行して EPS ファイルを作成:

```
mpost figure.mp  OR  mp figure.mp
```

Metapost プログラム名はシステムに依存し、Unix では通常 **mpost** で、他の多くのシステムでは **mp** です。Metapost は各グラフに対して 1 つずつの EPS ファイルを生成します。

- そのグラフを文書に取り込むには LaTeX graphics パッケージや、plainTeX では epsf.tex を使用:

```
\usepackage{graphics} % LaTeX
\input epsf.tex       % plainTeX
```

TeX DVI 出力を PS に変換するのに、dvips 以外の DVI ドライバを使う場合は、LaTeX ファイルに以下の行を入れる必要があるかも知れません:

```
\DeclareGraphicsRule{*}{eps}{*}{}
```

作られる各グラフは分離したファイルになっていて、最初のグラフのファイルは、例えば figure.0, 2 つ目は例えば figure.1 のような名前になります (出力用に "%j.%c" 以外のものをテンプレートとして指定しなかった場合)。よって、例えば 3 つ目のグラフを文書に取り込むためにあなたがしなければいけないことは以下のみです:

```
\includegraphics{figure.2} % LaTeX
\epsfbox{figure.2}         % plainTeX
```

mp ドライバの postscript ドライバに代わる利点は、もしあるとすれば、それは編集可能な出力であるということでしょう。この出力を可能な限り綺麗にするための、かなりの努力が払われました。Metapost 言語に関するそういった知識のおかげで、デフォルトの線種や色は配列 **lt**[] や **col**[] を編集することで変更できるようになりました。実線/点線、カラー/白黒といった選択も、真偽値として定義されている **dashedlines** や **colorlines** を変更することで行なえます。デフォルトの **tex** オプションが有効な場合、ブロック **verbatimtex...etex** を編集することで、ラベル文字フォントに対する大域的な変更が行なえます。特に、もし望むなら LaTeX のプリアンブルを追加することもでき、その場合 LaTeX の持つサイズ変更コマンドを使えるので最大の柔軟性を発揮できるでしょう。ただし、Metapost に plainTeX でなく LaTeX を実行させるよう、適切な MP 設定変数を設定することを忘れないでください。

Pbm

注意: gnuplot が –with-bitmap-terminals で作られた場合のみ使えます。書式:

```
set terminal pbm {<fontsize>} {<mode>} {size <x>,<y>}
```

<fontsize> は **small** か **medium** か **large** で、<mode> は **monochrome** か **gray** か **color** です。デフォルトの描画サイズは 640 ピクセルの幅で 480 ピクセルの高さです。出力サイズは、x と y の両方を 8 ピクセル倍したものに最も近くなるように空白が追加されます。必要であれば、この空白部分は後で取り除くことができます。

pbm ドライバの出力は <mode> によります: **monochrome** は portable bitmap (PBM; 1 ピクセル 1 ビット) を、**gray** は portable graymap (PGM; 1 ピクセル 3 bit) を、**color** は portable pixmap (PPM; 1 ピクセル 4 ビット) を出力します。

このドライバの出力は、NETPBM によって提供される様々な画像変換、画像処理ツールで使うことができます。 Jef Poskanzer の PBMPLUS パッケージに基づく NETPBM は、上記の PBM 形式から GIF, TIFF, MacPaint, Macintosh PICT, PCX, X11 ビットマップ、その他多くの形式に変換するプログラムを提供します。完全な情報は http://netpbm.sourceforge.net/ にあります。

例:

```
set terminal pbm small monochrome                # デフォルト
set terminal pbm color medium size 800,600
set output '| pnmrotate 45 | pnmtopng > tilted.png'  # NETPBM を利用
```

Pcl5

pcl5 ドライバは、1990 年代以降の Hewlett Packard 社製、あるいはその他のプリンタをサポートします。

書式:

```
set terminal pcl5 {<mode>} {{no}enhanced}
    {size <plotsize> | size <width>{unit},<height>{unit}}
    {font "<fontname>,<size>"} {pspoints | nopspoints}
    {fontscale <scale>} {pointsize <scale>} {linewidth <scale}
    {rounded|butt} {color <number_of_pens>}
```

<mode> は **landscape** か **portrait** です。<plotsize> はグラフの物理的な描画サイズで、それは以下のうちのいずれかです: **letter** は標準の (8 1/2" X 11") 出力、**legal** は (8 1/2" X 14") 出力、**noextended** は (36" X 48") 出力 (letter サイズ比)、**extended** は (36" X 55") 出力 (ほぼ legal サイズ比)、**a4** は (296mm x 210mm) 出力です。さらに、明示的にキャンバスサイズを **width**, **height** オプションで指定することもできます。単位のデフォルトは **in** です。サイズのデフォルトは **letter** です。

<fontname> は stick, univers (デフォルト), albertus, antique_olive, arial, avant_garde_gothic, bookman, zapf_chancery, clarendon, coronet, courier, courier_ps, cg_times, garamond_antigua, helvetica, helvetica_narrow, letter_gothic, marigold, new_century_schlbk, cg_omega, palatino, times_new_roman, times_roman, zapf_dingbats, truetype_symbols, wingdings のいずれかです。フォント名では大文字小文字は区別されず、下線はスペースかダッシュに置き換えられるかまたは取り除かれます。<fontsize> はポイント単位でのフォントの大きさです。

点の種類 (point type) は、**nopspoints** を指定することで制限されたデフォルトの組を使用できますが、**pspoints** を指定すると postscript 出力形式と同じ点種の組を使用することができるようになります。

オプション **butt** (デフォルト) は尖った端と角張った接合部を持つ線を使用し、**rounded** は線の端や接合部を丸くします。

線幅や点、フォントのサイズは、それぞれオプション **linewidth**, **pointscale**, **fontscale** で変更できます。

color は、グラフ内で使用するペンの数 <number_of_pens> を指定します。デフォルトは 8 で、最小は 2 です。

これらのオプションのいくつかの組み込まれたサポートは、プリンタに依存することに注意してください。例えば全てのフォントは恐らく HP Laserjet IV ではサポートされているでしょうが、HP Laserjet III と Designjet

750C では 2,3 (例えば univers, stick) がサポートされているのみでしょう。また、白黒の出力装置ではもちろんカラーも使えませんが、新しい物ならそれをグレイスケールでやってくれるでしょう。

デフォルト: landscape, a4, 8 色, univers, 12 point, pspoints, butt,

スケーリングなし

pcl5 出力形式は、**encoding** の設定に合うフォントを要求しようとします。これは最高の優先度を持つので、結果的に違うフォントの見た目になってしまかもしれないことに注意してください。この出力ドライバのデフォルトの **encoding** は、**HP Roman-8** です。

制限:

この出力ドライバは、透過性のアルファ値はサポートしていません。透過型の塗り潰しは陰影パターンによる疑似的なものです。箱付き文字列は実装していません。

UTF-8 のサポートも制限があります。HP-GL/2 で UTF-8 出力に対するラベルモードが欠けているため、このドライバは 8bit 文字が含まれている文字列に対しては PCL に戻ります。UTF-8 文字列の回転は、0, 90, 180, 270 度の角のみに制限されます。垂直方向の位置合わせも、フォントによっては効かなくなりえます。

拡張文字列のいくつかの機能 (空ボックスや重ね書き) は、HP-GL/2 に加えて

PCL の機能を使うことが必要になります。それは、その能力を持つものには合

いますが、あなたのプリンタやソフトウェアでは機能しないかもしれません。

Pdfcairo

出力形式 **pdfcairo** は、PDF 出力を生成します。実際の描画は、2D グラフィックライブラリである cairo と、文字列の配置とレンダリング用のライブラリ pango を経由して行われます。

書式:

```
set term pdfcairo
             {{no}enhanced} {mono|color}
             {font <font>} {fontscale <scale>}
             {linewidth <lw>} {rounded|butt|square} {dashlength <dl>}
             {background <rgbcolor>
             {size <XX>{unit},<YY>{unit}}
```

この出力形式は、ラベルやその他の文字列を、デフォルトで拡張文字列処理モード (enhanced text mode) で処理します。以下参照: **enhanced (p. 42)**。

描画における全ての線の幅は、**linewidth** で指定する因子 <lw> で変更できます。デフォルトの線幅は 0.5 ポイントです。(1 "PostScript" ポイント= 1/72 インチ = 0.353 mm)

rounded は、線の端や接合部を丸くします。デフォルトの **butt** は尖った端と角張った接合部を使用します。

PDF 出力のデフォルトのサイズは、5inch x 3inch です。オプション **size** は、これをユーザの指定するものへ変更します。デフォルトの X, Y サイズの単位はインチですが、他の単位も使用可能です (現在は cm のみ)。**font** は、"FontFace,FontSize" の書式、つまりフォント名とサイズをカンマで区切った一つの文字列として表記します。FontFace は、'Arial' のような通常のフォント名です。フォント名を与えない場合、pdfcairo 出力形式では'Sans' が使用されます。FontSize はポイント単位でのフォントサイズです。指定しない場合は、pdfcairo

出力形式では 12 ポイントサイズの標準フォントが使用されます。しかし、この出力形式のパラメータ fontscale のデフォルトは 0.5 なので、PDF 出力をフルサイズで見た場合、見かけのフォントサイズは、名目上のポイントサイズよりも小さくなるでしょう。

例 :

```
set term pdfcairo font "Arial,12"
set term pdfcairo font "Serif" # フォント名のみ変更
set term pdfcairo font ",14" # フォントサイズのみ変更
set term pdfcairo font "" # フォント名とサイズをリセット
```

フォントは、通常のフォント処理機構により取得されます。Windows では、フォントはコントロールパネルの "フォント" の項目で構成され見つけられるもので、UNIX では、フォントは "fontconfig" で処理されます。

文字列の処理を行うライブラリ Pango は、utf-8 に基づいているので、必要なら pdfcairo 出力形式は、あなたの文字コードを utf-8 に変換します。入力で想定する文字コードはあなたが使用している 'locale' から取るので、あなたの文字列が実際には違う文字コードならば、あなたがどの文字コードを使っているのかを確実に gnuplot がわかるようにしてください。詳細は以下参照: **encoding (p. 215)**。

pango は、unicode マッピングでないフォントに対しては予期せぬ結果を与えるかもしれません。例えば Symbol フォントに対しては、pdfcairo 出力形式は、文字コードを unicode に変換するために http://www.unicode.org/ で提供されるマッピングを利用します。なお、"the Symbol font" は、Acrobat Reader と一緒に "SY______.PFB" として配布されている Adobe Symbol フォントであると解釈されることに注意してください。この代わりに、OpenOffice.org と一緒に "opens___.ttf" として配布される OpenSymbol フォントが同じ文字を提供しています。Microsoft も Symbol フォント ("symbol.ttf") を配布していますが、これは異なる文字セットになっていて、いくつかは欠けていますし、いくつかは数式記号に変わってしまっています。あなたのデフォルトの設定でなんらかの問題が起きた場合 (例えばデモスクリプト enhancedtext.dem がちゃんと表示されないといった場合) は、Adobe か OpenOffice の Symbol フォントをインストールして、Microsoft の Symbol フォントを削除しないといけないかもしれません。"windings" のような他の非標準のフォントでも動作することが報告されています。

グラフのレンダリングは、アンチエイリアス、オーバーサンプリングの 2 つの機構を持っています。アンチエイリアスは、水平や垂直でない線をより滑らかにします。オーバーサンプリングは、アンチエイリアスと組でピクセルよりも小さいサイズでの精度を提供し、gnuplot が非整数座標の直線を書けるようになります。これは、対角方向の直線 (例えば 'plot x') が左右に揺れるのを避けます。

Pict2e

出力形式 **pict2e** は、picture 環境の LaTeX2e 用の変種を使用します。これは、元々の LaTeX picture 環境に基づく出力形式である **latex**, **emtex**, **tpic**, **eepic** に置き換わるものです。

この出力形式に代わり、より完全に gnuplot の機能をサポートするものとして **tikz**, **pstricks**, **cairolatex**, **pslatex**, **epslatex**, **mp** があります。

書式:

```
set terminal pict2e
             {font "{<fontname>}{,<fontsize>}"}
             {size <XX>{unit}, <YY>{unit}}
```

```
{color | monochrome}
{linewidth <lw>} {rounded | butt}
{texarrows | gparrows} {texpoints | gppoints}
{smallpoints | tinypoints | normalpoints}
```

この出力形式は、以下の LaTeX 標準パッケージを仮定します: **pict2e**, **xcolor**, **graphics/graphicx**, **amssymb**。pdflatex の場合は、透過のサポートに **transparent** パッケージを使用します。

デフォルトでは、グラフはこれが埋め込まれる文書のフォント設定を継承しますが、オプション **font** でフォントを強制的に、例えば cmtt (Courier) や cmr (Roman) などの指定したフォントに変更することもできます。その場合、特定のフォントサイズにも変更できますが、そうでなければ、fontsize 引数はその文字列用に必要なスペースを計算するのに使います。あなたのドライバが、例えば dvips のようには任意のサイズのフォントを利用できない場合は、標準の 10, 11, 12 ポイントサイズに限定してください。

グラフのデフォルトサイズは 5inch x 3inch ですが、オプション **size** でこれをユーザが希望する任意のサイズに変更できます。デフォルトでは、X と Y の単位は inch ですが、他の単位も利用できます (現在は cm のみ)。

texpoints は、"\Diamond" や "\Box" のような LaTeX コマンドを使って点 (point) の記号を書きます。これらは latexsym パッケージで提供され、これは LaTeX の基本配布物ですので、任意の LaTeX の実装に含まれます。他の点記号は、amssymb パッケージの記号を使用します。**gppoints** では、この出力形式は、代わりに gnuplot の内部ルーチンを使って点記号を描画します。

オプション **texpoints** では、3 つの異なる点サイズ指定: **normalpoints**, **smallpoints**, **tinipoints** を選択できます。

color は、gnuplot に \color{...} コマンドを生成させ、それによりグラフの色付けを行います。このオプションを使用するには、あなたの LaTeX 文書のプリアンブルに \usepackage{xcolor} を入れる必要があります。**monochrome** は、色に関するコマンドを一切出力しません。透過色の塗り潰しは、pdflatex を使用すれば利用できます。

linewidth は、線幅の拡大倍率を設定します。**rounded** は、線の端や接合部を丸くします。**butt** は尖った端と角張った接合部を使用し、こちらがデフォルトです。

pict2e は、点線のみサポートし、破線はサポートしていません。デフォルトの線種はすべて実線です。変更するには、**dashtype** 属性をつけて **set linetype** を使用してください。

texarrows は、LaTeX コマンドを使用して矢 (**arrow**) を描きますが、これはやや短かく、オプションをすべてはサポートしていません。**gparrows** は、gnuplot 自身のルーチンを用いて矢を描くことを選択しますが、こちらはすべての機能を備えています。

Pm

pm ドライバは、グラフが描画される OS/2 プレゼンテーションマネージャウィンドウを提供します。そのウィンドウは最初のグラフが描画されたときに開かれます。このウィンドウは印刷、クリップボードへのコピー、そしてそれ自身のオンラインヘルプを持っています。

書式:

```
set terminal pm      {{server} {n} | noserver}
                     {nopersist | persist}
                     {enhanced | noenhanced}
```

```
{font <fontspec>}
{nowidelines | widelines}
{fontscale <scale>}
{linewidth <scale>}
{pointscale <scale>}
{{title} "title"}
```

persist を指定すると、各グラフはそれぞれ自身のウィンドウを持ち、そのすべてのウィンドウは **gnuplot** が終了した後も開いたままになります。**server** を指定すると、全てのグラフは同じウィンドウ内に現われ、それは **gnuplot** 終了後も開いたままになります。このオプションは、さらに追加の数引数を取り、その数字はサーバプロセスのインスタンスになります。よって同時に複数のサーバウィンドウを使うことができます。

widelines を指定すると、全てのグラフは幅の広い線で描かれます。**enhanced** を指定すると、上付き文字や下付き文字、複数のフォントを使うことができます (詳細は、以下参照: **enhanced text (p. 42)**)。コア PostScript フォントのフォント名は 1 文字に省略できます (T/H/C/S はそれぞれ Times/Helvetica/Courier/Symbol を意味します)。

linewidth, **fontscale**, **pointscale** は、線幅、文字サイズ、点記号のサイズの変更に使えます。

title を指定すると、それは描画ウィンドウのタイトルとして使われます。それはサーバインスタンス名としても使われ、それは追加の数引数を上書きします。

gnuplot とは別のドライバである gnupmdrv.exe の場所は、gnuplot 自身が置かれているディレクトリを探しますが、それは環境変数 GNUPLOT_DRIVER_DIR か GNUPLOT を定義することで変更できます。最終的には、gnupmdrv.exe を見つけるのにはカレントディレクトリと PATH を使用します。

Png

書式:

```
set terminal png
       {{no}enhanced}
       {{no}transparent} {{no}interlace}
       {{no}truecolor} {rounded|butt}
       {linewidth <lw>} {dashlength <dl>}
       {tiny | small | medium | large | giant}
       {font "<face> {,<pointsize>}"} {fontscale <scale>}
       {size <x>,<y>} {{no}crop}
       {background <rgb_color>}
```

PNG, JPEG, GIF 画像は、外部ライブラリ libgd を使って生成されます。PNG の描画は、ImageMagick パッケージのソフト 'display' にその出力を以下のようにパイプで渡すことで対話的に表示させることができます:

```
set term png
set output '| display png:-'
```

次の描画コマンドからの出力は、display ウィンドウ上で対話的に <space> を打つことで見ることができます。現在の描画をファイルに保存するには、display ウィンドウで左クリックし、**save** を選択してください。

transparent は、ドライバに背景色の透明化 (transparent) を行うよう指示します。デフォルトは **notransparent** です。

interlace は、ドライバにインターレース GIF を生成するよう指示します。デフォルトは **nointerlace** です。

オプション **linewidth** と **dashlength** は拡大率で、描画されるすべての線に影響を与えます。すなわち、これらは様々な描画コマンドで要求される値にかけ算されます。

デフォルトでは、png 出力形式は、1 ピクセル毎に 24bit の色情報を持つ TrueColor 画像生成します。オプション **notruecolor** は、代わりに 8bit 色のみの画像 (256 個に番号付けられた色) を使用します。透明化塗りつぶしスタイル (transparent fill style) には、オプション **truecolor** が必要です。以下参照: **fillstyle (p. 283)**。背景の透明化は、番号付け画像か TrueColor 画像で可能です。アンチエイリアスにも TrueColor が必要です。

butt は線分の描画で、その端の点でのはみだしを起こさない描画メソッドを使うようドライバに指示します。この設定は、線幅が 1 より大きい場合にのみ意味があります。これの逆が **rounded** で、アンチエイリアスが有効でない場合 (**notruecolor**) 多少より均一な曲線を生成しますが、より遅くなります。

フォントの選択の詳細は、やや複雑です。以下に同じ意味を持つ簡単な例を示します:

```
set term png font arial 11
set term png font "arial,11"
```

より詳しい情報については、**fonts** の下の該当するセクションを参照してください。

出力描画サイズ <x,y> はピクセル単位で与えます。デフォルトは 640x480 です。以下も参照: **canvas (p. 39)**, **set size (p. 278)**。描画終了後の端の余白は、オプション **crop** で取り除くことができ、その結果としてその画像サイズは小さくなります。デフォルトは **nocrop** です。

例

```
set terminal png font "arial,14" size 800,600 background "white"
```

これは、'arial' というフェース名のスケーラブルフォントを検索し、フォントサイズを 14pt に設定します。フォントの検索がどのように行われるかについては以下参照: **fonts (p. 66)**。

```
set terminal png transparent enhanced
```

これは、24 ビット/ピクセルの色情報を使用し、背景を透明化します。そして表示される文字列の配置制御として **enhanced text** モードを使用します。

Pngcairo

出力形式 **pngcairo** は、PNG 出力を生成します。実際の描画は、2D グラフィックライブラリである cairo と、文字列の配置とレンダリング用のライブラリ pango を経由して行われます。

書式:

```
set term pngcairo
             {{no}enhanced} {mono|color}
             {{no}transparent} {{no}crop} {background <rgbcolor>
             {font <font>} {fontscale <scale>}
```

```
{linewidth <lw>} {rounded|butt|square} {dashlength <dl>}
{pointscale <ps>}
{size <XX>{unit},<YY>{unit}}
```

この出力形式は、拡張文字列処理モード (enhanced text mode) をサポートしていて、フォントや書式コマンド (上付、下付など) をラベルや他の文字列に埋め込むことができます。拡張文字列処理モードの書式は他の gnuplot の出力形式と共通です。詳細は、以下参照: **enhanced (p. 42)**。

描画における全ての線の幅は、因子 <lw> で変更できます。

rounded は、線の端や接合部を丸くします。デフォルトの **butt** は尖った端と角張った接合部を使用します。

PNG 出力のデフォルトのサイズは、640 x 480 ピクセルです。オプション **size** は、これをユーザの指定するものへ変更します。デフォルトの X, Y サイズの単位はピクセルですが、他の単位も使用可能です (現在は cm とインチ)。サイズを cm かインチで与えるとそれは、解像度 72 dpi でのピクセル数に変換されます。**size** オプションで指定されたことによる描画の端から端までの領域は、常にスクリーン座標の 0.0 から 1.0 に対応します。

<font> は、"FontFace,FontSize" の書式、つまりフォント名とサイズをカンマで区切った一つの文字列として表記します。FontFace は、'Arial' のような通常のフォント名です。フォント名を与えない場合、pngcairo 出力形式では'Sans' が使用されます。FontSize はポイント単位でのフォントサイズです。指定しない場合は、pngcairo 出力形式では 12 ポイントのサイズが使用されます。

例 :

```
set term pngcairo font "Arial,12"
set term pngcairo font "Arial" # フォント名のみ変更
set term pngcairo font ",12" # フォントサイズのみ変更
set term pngcairo font "" # フォント名とサイズをリセット
```

フォントは、通常のフォント処理機構により取得されます。Windows では、フォントはコントロールパネルの "フォント" の項目で構成され見つけられるもので、UNIX では、フォントは "fontconfig" で処理されます。

文字列のレイアウトに使用されるライブラリ Pango は、utf-8 に基づいていますので、pngcairo 出力形式では、文字コードを utf-8 に変換する必要があります。デフォルトの入力文字コードは、あなたが使用している 'locale' に基づきます。他の文字コードにしたい場合は、あなたがどの文字コードを使っているのかを確実に gnuplot がわかるようにしてください。詳細については、以下参照: **encoding (p. 215)**。

pango は、unicode マッピングでないフォントに対しては予期せぬ結果を与えるかもしれません。例えば Symbol フォントに対しては、pngcairo 出力形式は、文字コードを unicode に変換するために http://www.unicode.org/ で提供されるマッピングを利用します。なお、"the Symbol font" は、Acrobat Reader と一緒に "SY______.PFB" として配布されている Adobe Symbol フォントであると解釈されることに注意してください。この代わりに、OpenOffice.org と一緒に "opens___.ttf" として配布される OpenSymbol フォントが同じ文字を提供しています。Microsoft も Symbol フォント ("symbol.ttf") を配布していますが、これは異なる文字セットになっていて、いくつかは欠けていますし、いくつかは数式記号に変わってしまっています。あなたのデフォルトの設定でなんらかの問題が起きた場合 (例えばデモスクリプト enhancedtext.dem がちゃんと表示されないといった場合) は、Adobe か OpenOffice の Symbol フォントをインストールして、Microsoft の Symbol フォントを削除しないといけないかもしれません。

レンダリングは、cairo と pango ライブラリがサポートする範囲内で、オーバーサンプリングとアンチエイリアス、フォントのヒンティングを行います。

Postscript

postscript ドライバではいくつかのオプションが設定できます。

書式:

```
set terminal postscript {default}
set terminal postscript {landscape | portrait | eps}
                        {enhanced | noenhanced}
                        {defaultplex | simplex | duplex}
                        {fontfile {add | delete} "<filename>"
                         | nofontfiles} {{no}adobeglyphnames}
                        {level1 | leveldefault | level3}
                        {color | colour | monochrome}
                        {background <rgbcolor> | nobackground}
                        {dashlength | dl <DL>}
                        {linewidth | lw <LW>} {pointscale | ps <PS>}
                        {rounded | butt}
                        {clip | noclip}
                        {palfuncparam <samples>{,<maxdeviation>}}
                        {size <XX>{unit},<YY>{unit}}
                        {blacktext | colortext | colourtext}
                        {{font} "fontname{,fontsize}" {<fontsize>}}
                        {fontscale <scale>}
```

以下のようなエラーメッセージが出た場合:

```
"Can't find PostScript prologue file ... "
```

以下参照: **postscript prologue (p. 382)**。そしてその指示に従ってください。

landscape と **portrait** は出力が横置か、縦置かを選択します。**eps** モードは EPS (Encapsulated PostScript) 出力を生成しますが、これは通常の PostScript に、それを他の多くのアプリケーションで取り込むことができるようにいくつかの行を追加したものです (追加される行は PostScript のコメント行なので、よってそれ自身もちゃんと印刷できます)。EPS 出力を得るには **eps** モードを使用し、1 つのファイルには 1 つのグラフのみ、としてください。**eps** モードではフォントも含めてグラフ全体がデフォルトの大きさの半分に縮められます。

enhanced は拡張文字列処理モード (上付き文字、下付き文字、および複数のフォントの利用) の機能を有効にします。詳細は、以下参照: **enhanced (p. 42)**。**blacktext** は、たとえカラーモードでも全ての文字列を黒で書きます。

PostScript の両面印刷命令 (duplex) は、プリンタで 1 枚の紙に両面印刷することを可能にします。**defaultplex** はプリンタのデフォルトの設定を使用し、**simplex** は紙の片面のみ印刷、**duplex** は 両面印刷を行ないます (あなたのプリンタがそれを行なえないなら無視されます)。

"<**fontname**>" は有効な PostScript フォントの名前で、<**fontsize**> は PostScript ポイント単位でのフォントの大きさです。標準的な postscript フォント以外に、数式を表現するのに便利な oblique Symbol フォント ("Symbol-Oblique") が定義されています。

default は全てのオプションを以下のデフォルトの値に設定します: **landscape**, **monochrome**, **dl 1.0**, **lw 1.0**, **defaultplex**, **enhanced**, "Helvetica", 14pt。PostScript のグラフのデフォルトの大きさは、10 インチ

の幅で 7 インチの高さです。オプション **color** はカラーを有効にし、**monochrome** は各要素を黒と白描画します。さらに、**monochrome** は灰色の **palette** も使用しますが、これは、明示的に **colorspec** で指定された部品の色を変更しません。

dashlength または **dl** は点線の線分の長さを <DL> (0 より大きい実数) に設定し、**linewidth** または **lw** は全ての線の幅を <LW> に設定します。

デフォルトでは、生成される PostScript コードは、特にフィルタリングや filledcurves のようなでこぼこな領域のパターン塗りつぶしにおいて、PostScript Level 2 として紹介されている言語機能を使います。PostScript Level 2 の機能は条件的に保護されていて、PostScript Level 1 のインタープリタがエラーを出さず、むしろメッセージか PostScript Level 1 による近似であることを表示するようになっています。**level1** オプションは、これらの機能を近似する PostScript Level1 で代用し、PostScript Level 2 コードを一切使用しません。これは古いプリンタや、Adobe Illustrator の古いバージョンなどで必要になるかもしれません。このフラグ **level1** は出力された PostScript ファイルのある一行を手で編集することで、後から強制的に PostScript Level 1 機能を ON/OFF にすることもできます。level 2 のコードが含まれている場合、上の機能は現われないか、このフラグがセットされた場合、あるいは PostScript インタプリタプログラムが level 2 以上の PostScript を解釈するとは言わなかった場合に警告文に置き換わります。**level3** オプションは、ビットマップ画像の PNG 化の機能を有効にします。それにより出力サイズをかなり削減できます。

rounded は、線の端や接合部を丸くし、デフォルトの **butt** は尖った端と角張った接合部を使用します。

clip は、PostScript にすべての出力を BoundingBox (PostScript の外枠) でクリップすることを指示します; デフォルトは **noclip** です。

palfuncparam は **set palette functions** から出力の傾きをどのようにコード化するかを制御します。解析的な色の成分関数 (**set palatte functions** で設定される) は、postscript 出力では傾きの線形補完を用いてコード化されます: まず色の成分関数が <samples> 個の点で標本化され、そしてそれらの点は、結果として線形補完との偏差が <maxdeviation> 以内に収まるように削除されます。ほとんど全ての有効なパレットで、デフォルトの <samples> =2000 と <maxdeviation>=0.003 の値をそのまま使うのが良いでしょう。

PostScript 出力のデフォルトの大きさは 10 インチ x 7 インチです。EPS 出力のデフォルトの大きさは 5 x 3.5 インチです。オプション **size** はこれらをユーザが指定したものに変更します。デフォルトでは X と Y のサイズの単位はインチとみなされますが、他の単位 (現在は cm のみ) も使うことはできます。描画の BoundingBox (PostScript ファイルの外枠) は、サイズが変更された画像を丁度含むように正しく設定されます。スクリーン座標は、オプション **size** で指定された描画枠の全体が 0.0 から 1.0 になります。

fontfile や **fontfile add** で指定されたフォントは、そのフォントのフォント定義を直接 postscript Type 1, TrueType フォントから gnuplot の postscript 出力の中にカプセル化します。よって、その埋め込まれたフォントは見出し、タイトルなどに使うことができます。詳細は、以下参照: **postscript fontfile (p. 380)**。**fontfile delete** によってフォントファイルを埋め込まれるファイルの一覧から取り除くことができます。**nofontfiles** は埋め込みフォントのリストをクリアします。

postscript ドライバは約 70 種類の異なる点種をサポートしていて、これは **plot** や **set linetype** の **pointtype** オプションで選択できます。

gnuplot と Postscript に関する多分有用と思われるファイルが **gnuplot** の配布物、またはその配布サイトの /docs/psdos サブディレクトリ内にいくつか含まれています。そこには "ps_symbols.gpi" (実行すると **postscript** ドライバで使える全ての記号を紹介する "ps_symbols.ps" というファイルを生成する **gnuplot** のコマンドファイル)、"ps_guide.ps" (拡張された書式に関する要約と、文字列内で 8 進コードで生成されるもの、symbol フォント等を含む PostScript ファイル)、"ps_file.doc" (**gnuplot** で作られる PostScript ファイル

の構造の説明を含むテキストファイル)、"ps_fontfile_doc.tex" (数式フォントの文字の一覧と LaTeX のフォントの埋め込みに関する短い説明を含む LaTeX ファイル) があります。

PostScript ファイルは編集可能で、一度 **gnuplot** でそれを作れば、それを望むように修正することは自由に行なえます。そのためのヒントを得るには、以下参照: **editing postscript (p. 380)**。

PostScript の編集 (editing postscript)

PostScript 言語はとても複雑な言語で、ここで詳細を記述することはとてもできません。それでも、**gnuplot** で作られる PostScript ファイルには、致命的なエラーをそのファイルに導入してしまう危険性のない変更を行なうことが可能な部分があります。

例えば、PostScript の文 "/Color true def" (**set terminal postscript color** コマンドに答えてファイルに書き込まれます) を変更して、その描画を白黒のものにする方法はおわかりでしょう。同様に、線の色、文字の色、線の太さ (weight)、記号のサイズも、本当に簡単に書き換えられるでしょう。タイトルや見出しなどの文字列の誤植や、フォントの変更も編集可能でしょう。任意のものの配置も変更できますし、もちろん、任意のものを追加したり、削除したりもできますが、それらの修正は PostScript 言語の深い知識が必要でしょう。

gnuplot によって作られる PostScript ファイルの構成に関しては、gnuplot のソース配布物内の docs/ps ディレクトリのテキストファイル "ps_file.doc" に述べられています。

Postscript fontfile

```
set term postscript ... fontfile {add|delete} <filename>
```

オプション **fontfile** または **fontfile add** は 1 つのファイル名を引数として持ち、そのファイルを postscript 出力内にカプセル化して埋め込み、それによって様々な文字列要素 (ラベル、目盛り見出し、タイトル等) をそのフォントで出力することを可能にします。オプション **fontfile delete** も 1 つのファイル名を引数に持ち、そのファイル名をカプセル化されるファイルのリストから削除します。

postscript 出力ドライバはいくつかのフォントファイル形式を認識します: ASCII 形式の Type 1 フォント (拡張子 ".pfa")、バイナリ形式の Type 1 フォント (拡張子 ".pfb")、TrueType フォント (拡張子 ".ttf")。pfa ファイルは直接認識されますが、pfb と ttf ファイルは gnuplot の実行中に並行して変換され、そのために適切な変換ツール (下記参照) がインストールされている必要があります。ファイル名は拡張子も含めて完全な形で指定する必要があります。各 **fontfile** オプションはちょうど一つのフォントファイル名に対応しますので、複数のフォントファイルを埋め込むためにはこのオプションを複数回使って下さい。

フォントファイルを見つけるための検索順は以下の通りです。(1) 絶対パス名、または現在の作業ディレクトリ (カレントディレクトリ) (2) **set loadpath** で指定したディレクトリのすべて (3) **set fontpath** で指定したディレクトリ (4) 環境変数 GNUPLOT_FONTPATH に指定されているディレクトリ

埋め込まれたフォントファイルを使うには、フォント名を指定する必要がありますが、それは通常ファイル名と違います。対話モードで **fontfile** オプションを使ってフォントを埋め込んだ場合、フォント名はスクリーンに表示されます。

例:

```
set term post "VAGRoundedBT_Regular" 14 fontfile "bvrr8a.pfa"
  Font file 'bvrr8a.pfa' contains the font 'VAGRoundedBT_Regular'.
  Location: /usr/share/fonts/Type1/bvrr8a.pfa
```

pfa や pfb フォントでは、フォント名はフォントファイル内に見つけることができます。フォントファイル中に "/FontName /URWPalladioL-Bold def" のような行がありますが、この真中の物から / を除いたものがフォント名です。この例の場合は "URWPalladioL-Bold" となります。TrueType フォントでは、フォント名はバイナリ形式で保存されているので見つけるのは容易ではありません。さらに、その名前は多くの場合、Type 1 フォント (実行中に TrueType が変換される形式である) ではサポートされていない、スペースを含んだ形式になっています。そのため、フォント名はそこからスペースを取り除いた形に変換されます。gnuplot で使うために生成されたフォント名が何であるかを知る最も簡単な方法は、gnuplot を対話モードで起動して、以下のように入力することです: "set terminal postscript fontfile '<filename.ttf>'".

フォントファイル (ttf, pfb) を pfa 形式に変換するために、フォントファイルを読んで、そして変換結果を標準出力に吐き出す変換ツールが必要になります。その出力を標準出力に書き出すことができない場合、実行中の変換はできません。

pfb ファイルに対しては、例えば "pfbtops" が使えます。それがシステムにインストールされていれば、実行中の変換はうまく行くはずです。pfb ファイルのカプセル化をちょっとやってみましょう。もしプログラムの変換時に正しくツールを呼び出していない場合は、どのようにツールを呼び出したら良いかを環境変数 GNUPLOT_PFBTOPFA に、例えば "pfbtops %s" のように定義して下さい。%s はフォントファイル名に置き換えられますので、これはその文字列に必ず必要です。

実行中の変換をしたくなくて、けれども pfa 形式のファイルは必要である場合、"pfb2pfa" という C で書かれた簡単なツールを使えば良いでしょう。これは大抵の C コンパイラでコンパイルでき、たくさんの ftp サーバに置いてあります。例えば`ftp://ftp.dante.de/tex-archive/fonts/utilities/ps2mf/`

実際に "pfbtopfa" と "pfb2ps" は同じ作業を行います。"pfbtops" は結果の pfa コードを標準出力に出力しますが、"pfbtopfa" はファイルに出力します。

TrueType フォントは、例えば "ttf2pt1" というツールを使って Type 1 pfa フォーマットに変換できます。これは以下にあります: `http://ttf2pt1.sourceforge.net/`

もし gnuplot に組み込まれている変換手順がうまく行かない場合、変換コマンドは環境変数 GNUPLOT_TTFTOPFA で変更できます。ttf2pt1 を使う場合は、それを "ttf2pt1 -a -e -W 0 %s - " のように設定して下さい。ここでも%s はファイル名を意味します。

特殊な用途のために、パイプも使えるようになっています (パイプをサポートしている OS 上で)。ファイル名を "<" で始め、その後にプログラム呼び出しを追加します。そのプログラム出力は標準出力への pfa データでなければいけません。結果として pfa ファイルを、例えば以下のようにしてアクセスできることになります: **set fontfile "< cat garamond.pfa"**。

Type 1 フォントを取り込むことは、例えば LaTeX 文書中に postscript ファイルを取り込む場合に使えます。pfb 形式の "european computer modern" フォント ("computer modern" フォントの一種) が各地の CTAN サーバに置かれています。`ftp://ftp.dante.de/tex-archive/fonts/ps-type1/cm-super/`

例えば、ファイル "sfrm1000.pfb" は、中太、セリフ付き、立体の 10 ポイントのフォント (フォント名 "SFRM1000") です。computer modern フォントは今でも数式を書くのに必要ですが、それは以下にあります: `ftp://ftp.dante.de/tex-archive/fonts/cm/ps-type1/bluesky`

これらによって、TeX 用の任意の文字も使えます。しかし、computer modern フォントは少しエンコーディングがおかしくなっています (このため、文字列には cmr10.pfb の代わりに sfrm1000.pfb を使うべきです)。TeX フォントの使用法はいくつかのデモの一つで知ることができます。**gnuplot** のソース配布物の /docs/psdoc に含まれるファイル"ps_fontfile_doc.tex" に TeX 数学フォントの文字の一覧表が含まれています。

フォント "CMEX10" (ファイル "cmex10.pfb") を埋め込むと、gnuplot は追加フォント "CMEX10-Baseline" も定義します。それは、他の文字にあうように垂直方向にずらされたものです (CMEX10 は、記号の天辺にベースラインがあります)。

PostScript prologue ファイル

各 PostScript 出力ファイルは %%Prolog セクションを含みますし、例えば文字エンコーディングなどを含む追加ユーザ定義セクションを含むかもしれませせん。これらのセクションは、gnuplot の実行ファイル中にコンパイルされている、あるいはあなたのコンピュータの別のところに保存されている PostScript prologue ファイル群からコピーされます。これらのファイルが置かれるデフォルトのディレクトリは、gnuplot のインストール時に設定されますが、このデフォルトは gnuplot コマンドの **set psdir** を使うか、環境変数 GNUPLOT_PS_DIR を定義することで変更できます。以下参照: **set psdir (p. 276)**。

Postscript adobeglyphnames

この設定は、UTF-8 エンコーディングでの PostScript 出力にのみ関係します。これは、0x00FF より大きい Unicode エントリポイント (つまり Latin1 集合外のすべて) の文字を記述するのに使われる名前を制御します。一般に、unicode 文字は一意の名前を持たず、それは unicode 識別番号しか持ちません。しかし、Adobe は、ある範囲の文字 (拡張ラテン文字、ギリシャ文字等) に名前を割り当てる推奨規則を持っています。フォントによってはこの規則を利用しているものありますし、そうでないものもあります。gnuplot はデフォルトでは Adobe グリフ名を使用します。例えば、ギリシャ文字の小文字のアルファは /alpha となります。**noadobeglyphnames** を指定した場合、この文字に対して gnuplot は代わりに /uni03B1 を使おうとします。この設定でおかしくなったとすれば、それはその文字がフォント内にあるにもかかわらずそれが見つからない場合です。Adobe フォントに対しては、デフォルトを使うのが常に正しいかもしれませんが、他のフォントでは両方の設定を試してみないといけないかもしれません。以下も参照: **fontfile (p. 380)**。

Pslatex and pstex

pslatex ドライバは LaTeX で後処理される出力を生成し、**pstex** ドライバは TeX で後処理される出力を生成します。**pslatex** は dvips と xdvi で認識可能な \special 命令を使用します。**pstex** で生成される図は、任意の plain-TeX ベースの TeX (LaTeX もそうです) で取り込むことができます。

書式:

```
set terminal [pslatex | pstex] {default}
set terminal [pslatex | pstex]
                        {rotate | norotate}
                        {auxfile | noauxfile}
                        {level1 | leveldefault | level3}
                        {color | colour | monochrome}
                        {background <rgbcolor> | nobackground}
                        {dashlength | dl <DL>}
                        {linewidth | lw <LW>} {pointscale | ps <PS>}
```

```
{rounded | butt}
{clip | noclip}
{palfuncparam <samples>{,<maxdeviation>}}
{size <XX>{unit},<YY>{unit}}
{<font_size>}
```

以下のようなエラーメッセージが出た場合:

```
"Can't find PostScript prologue file ... "
```

以下参照: **postscript prologue (p. 382)**。そしてその指示に従ってください。

オプション **color** はカラーを有効にし、**monochrome** は各要素を黒と白描画します。さらに、**monochrome** は灰色の **palette** も使用しますが、これは、明示的に **colorspec** で指定された部品の色を変更しません。

dashlength または **dl** は点線の線分の長さを <DL> (0 より大きい実数) に設定し、**linewidth** または **lw** は全ての線の幅を <LW> に設定します。

デフォルトでは、生成される PostScript コードは、特にフィルタリングや filledcurves のようなでこぼこな領域のパターン塗りつぶしにおいて、PostScript Level 2 として紹介されている言語機能を使います。PostScript Level 2 の機能は条件的に保護されていて、PostScript Level 1 のインタープリタがエラーを出さず、むしろメッセージか PostScript Level 1 による近似であることを表示するようになっています。**level1** オプションは、これらの機能を近似する PostScript Level1 で代用し、PostScript Level 2 コードを一切使用しません。これは古いプリンタや、Adobe Illustrator の古いバージョンなどで必要になるかもしれません。このフラグ **level1** は出力された PostScript ファイルのある一行を手で編集することで、後から強制的に PostScript Level 1 機能を ON/OFF にすることもできます。level 2 のコードが含まれている場合、上の機能は現われないか、このフラグがセットされた場合、あるいは PostScript インタプリタプログラムが level 2 以上の PostScript を解釈するとは言わなかった場合に警告文に置き換わります。**level3** オプションは、ビットマップ画像の PNG 化の機能を有効にします。それにより出力サイズをかなり削減できます。

rounded は、線の端や接合部を丸くし、デフォルトの **butt** は尖った端と角張った接合部を使用します。

clip は、PostScript にすべての出力を BoundingBox (PostScript の外枠) でクリップすることを指示します; デフォルトは **noclip** です。

palfuncparam は **set palette functions** から出力の傾きをどのようにコード化するかを制御します。解析的な色の成分関数 (**set palatte functions** で設定される) は、postscript 出力では傾きの線形補完を用いてコード化されます: まず色の成分関数が <samples> 個の点で標本化され、そしてそれらの点は、結果として線形補完との偏差が <maxdeviation> 以内に収まるように削除されます。ほとんど全ての有効なパレットで、デフォルトの <samples> =2000 と <maxdeviation>=0.003 の値をそのまま使うのが良いでしょう。

PostScript 出力のデフォルトの大きさは 10 インチ x 7 インチです。EPS 出力のデフォルトの大きさは 5 x 3.5 インチです。オプション **size** はこれらをユーザが指定したものに変更します。デフォルトでは X と Y のサイズの単位はインチとみなされますが、他の単位 (現在は cm のみ) も使うことはできます。描画の BoundingBox (PostScript ファイルの外枠) は、サイズが変更された画像を丁度含むように正しく設定されます。スクリーン座標は、オプション **size** で指定された描画枠の全体が 0.0 から 1.0 になります。

rotate が指定されると y 軸の見出しが回転されます。<font_size> は希望するフォントの (ポイント単位での) 大きさです。

auxfile が指定されると、ドライバは PostScript コマンドを、LaTeX ファイルに直接出力する代わりに、補助ファイルに書き出すようになります。これは、dvips がそれを扱えないくらい大きいグラフである場合に有

用です。補助 PostScript ファイルの名前は、**set output** コマンドで与えられる TeX ファイルの名前から導かれるもので、それはその最後の **.tex** の部分 (実際のファイル名の最後の拡張子の部分) を **.ps** で置き換えたもの、または、TeX ファイルに拡張子がないならば **.ps** を最後に付け足したものになります。**.ps** ファイルは \special{psfile=...} という命令で **.tex** ファイルに取り込まれます。**multiplot** モード以外では、次の描画を行なう前にその出力ファイルをクローズするのを忘れないでください。

pslatex ドライバは文字列の配置の制御に特別な方法を提供します: (a) '{' で始まる文字列は、'}' で閉じる必要がありますが、その文字列全体が LaTeX によって水平方向にも垂直方向にもセンタリングされます。(b) '[' で始まる文字列の場合は、位置の指定をする文字列 (t,b,l,r のうち 2 つまで) が続き、次に ']{'、文字列本体、で最後に '}' としますが、この文字列は LaTeX が LR-box として整形します。\rule{}{} を使えばさらに良い位置合わせが可能でしょう。

ここに記述されていないオプションは **Postscript terminal** のものと同一ですので、それらが何を行なうのかを知りたければそちらを参照してください。

例:

```
set term pslatex monochrome rotate        # デフォルトに設定
```

PostScript コマンドを "foo.ps" に書き出す:

```
set term pslatex auxfile
set output "foo.tex"; plot ...; set output
```

見出しの位置合わせに関して: gnuplot のデフォルト (大抵それなりになるが、そうでないこともある):

```
set title '\LaTeX\ -- $ \gamma $'
```

水平方向にも垂直方向にもセンタリング:

```
set label '{\LaTeX\ -- $ \gamma $}' at 0,0
```

位置を明示的に指定 (上に合わせる):

```
set xlabel '[t]{\LaTeX\ -- $ \gamma $}'
```

他の見出し – 目盛りの長い見出しに対する見積り:

```
set ylabel '[r]{\LaTeX\ -- $ \gamma $\rule{7mm}{0pt}}'
```

線幅と点の大きさは **set style line** で変更できます。

Pstricks

pstricks ドライバは TeX、または LaTeX の "pstricks.sty" マクロパッケージと共に使われることを意図しています。"pstricks.sty" は必要ですが、もちろん PostScript を解釈するプリンタ、または Ghostscript のような変換ソフトも必要です。

PSTricks は以下にあります。http://tug.org/PSTricks/。

このドライバは、PSTricks パッケージの全ての能力を使おうとなどとは全く考えていません。

書式:

```
set terminal pstricks
               {unit | size <XX>{unit},<YY>{unit}}
               {standalone | input}
               {blacktext | colortext | colourtext}
               {linewidth <lw>} {rounded | butt}
               {pointscale <ps>}
               {psarrows | gparrows}
               {background <rgbcolor>}
               {pstricks | pdftricks2}
```

オプション **unit** は、内部サイズ 1x1 のグラフを生成します。デフォルトのグラフは、**size 5in,3in** です。

standalone は、そのままコンパイルできる、複数のグラフも可能な LaTeX ファイルを生成します。デフォルトは **input** で、これは他の LaTeX ファイから include できるものを生成します。

blacktext は、すべての文字列を強制的に黒で書き出し、**colortext** は色付きの文字を可能にします。デフォルトは **blacktext** です。

rounded は、線の端や接合部を丸くし、デフォルトの **butt** は尖った端と角張った接合部を使用します。

linewidth と **pointscale** は、それぞれ線幅と点 (point) 記号のサイズを伸縮します。

psarrows は、**arrow** (矢) を PSTricks コマンドで描きますが、これは少し短かく、そしてすべてのオプションを無視します。**gparrows** は、それに代わり、gnuplot 自身のサブルーチンによる、すべての gnuplot 用の機能に対応する arrow を描きます。

オプション **pdftricks2** を使用すると、**pdftricks** マクロパッケージ用の出力を生成し、これは pdflatex/lualatex で利用できます。そうでない場合は伝統的な tex/latex、あるいは xelatex で使用する **pstricks** パッケージ用の出力を生成します。

古いオプション **hacktext** は、新しいデフォルトの書式 (%h) で置き換えられています。以下参照: **format specifiers** (p. **220**)。

透過色の使用は、Ghostscript、または PDF に変換する他のものがそれをサポートしている必要があります。

Qt

qt 出力形式は、Qt ライブラリを用いて別ウィンドウへの出力を生成します。

書式:

```
set term qt {<n>}
            {size <width>,<height>}
            {position <x>,<y>}
            {title "title"}
            {font <font>} {{no}enhanced}
            {rounded|butt}
            {{no}replotonresize}
            {{no}antialias}
            {linewidth <lw>} {dashlength <dl>}
            {{no}persist} {{no}raise} {{no}ctrl}
```

```
{close}
{widget <id>}
```

複数の描画ウィンドウもサポートしていて、**set terminal qt <n>** とすれば番号 n の描画ウィンドウへ出力します。

デフォルトのウィンドウタイトルは、このウィンドウ番号に基づいています。そのタイトルはキーワード "title" でも指定できます。

描画ウィンドウは、**gnuplot** の出力形式を別なものに変更した後でも開いたまま残ります。描画ウィンドウは、そのウィンドウが入力フォーカスを持っている状態で文字 'q' を打つか、ウィンドウマネージャメニューで **close** を選択するか、または **set term qt <n> close** とすることで閉じることができます。

描画領域のサイズはピクセル単位で与えます、デフォルトは 640x480 です。それに加えて、ウィンドウの実際のサイズには、ツールバーやステータスバー用のスペースも追加されます。ウィンドウのサイズを変更すると、描画グラフもウィンドウの新しいサイズにぴったり合うようにすぐに伸縮されます。**qt** 出力形式はフォント、線幅も含めて描画全体を伸縮しますが、全体のアスペクト比は一定に保ちます。その後 **replot** とタイプするか、ターミナルツールバーの **replot** アイコンをクリックするか、新たに **plot** コマンドを入力すると、その新しい描画では完全にそのウィンドウに合わせられますが、フォントサイズや線幅はそれ ぞれのデフォルトにリセットされます。

position オプションは描画ウィンドウの位置を設定するのに使えます。これはコマンド **set term** 後の最初の描画にのみ適用されます。

現在の描画ウィンドウ (**set term qt <n>** で選択されたもの) は対話型で、その挙動は、他の出力形式と共通です。詳細は、以下参照: **mouse (p. 244)**。それには追加のアイコンもいくつかついていますが、それらはそれ自体が説明的なものになっているはずです。

この出力形式は、拡張文字列処理モード (enhanced text mode) をサポートしていて、フォントや書式コマンド (上付、下付など) をラベルや他の文字列に埋め込むことができます。拡張文字列処理モードの書式は、gnuplot の他の出力形式と共通です。詳細は、以下参照: **enhanced (p. 42)**。

<font> は "FontFace,FontSize" の形式で、FontFace と FontSize とをコンマで分離して一つの文字列として書きます。FontFace は、'Arial' のような通常のフォント名です。FontFace を与えない場合は、qt 出力形式は'Sans' を使用します。FontSize はポイント単位のフォントサイズです。FontSize を与えない場合は、qt 出力形式は 9 ポイントを使用します。

例 :

```
set term qt font "Arial,12"
set term qt font "Arial" # フォント名のみ変更
set term qt font ",12" # フォントサイズのみ変更
set term qt font "" # フォント名、、フォントサイズをリセット
```

dashlength は、点線/破線パターンのユーザ定義にのみ影響を与え、Qt が内部に持っているパターンには影響を与えません。

可能な限り最も良い出力を生成するために、このレンダリングはアンチアリアス、オーバーサンプリング、ヒンチングの 3 つの機構を持っています。オーバーサンプリングは、アンチエイリアスと組でピクセルよりも小さいサイズでの精度を提供し、gnuplot が非整数座標の直線を書けるようになります。これは、対角方向の直線 (例えば 'plot x') が左右に揺れるのを避けます。ヒンティングは、オーバーサンプリングによって引き起こされる水平、垂直方向の線分のぼかしを避けます。この出力形式は、これらの直線を整数座標に揃え、それに

より、1 ピクセル幅の直線は本当に 1 つ (1 つより多くも少なくもない) のピクセルで描画します。

butt は、求める線分の端点がはみ出さないような線分の描画方法を使うようドライバに指示します。この設定は、線幅が 1 より大きい場合にのみ適用します。この設定は、水平か垂直の線を書く場合に最も有用です。デフォルトは、**rounded** です。

オプション **replotonresize** は、描画ウィンドウのサイズが変更されるとデータを replot します。このオプションがない状態では、アスペクト比を変えないウィンドウサイズ変更の後で、ウィンドウの一部分しか描画されないかもしれません。このオプションをつけると、gnuplot は個々のサイズ変更イベント毎に完全に replot を行い、スペースを適切に利用した結果になります。このオプションは、サイズ変更の間の再描画に対する CPU の内在的な集中に心配がある場合を除いては、一般的に望ましいものです。replot は、ホットキーの'e' を使ったりコマンド 'replot' によって手動で実行することも可能です。

デフォルトでは、描画が行われたときにウィンドウはデスクトップの一番上 (最前面) に表示されます。これは、キーワード "raise" で制御できます。キーワード "persist" は、すべての描画ウインドウを明示的に閉じない間は、gnuplot が終了しないようにします。

<space> キーは gnuplot コンソールウィンドウを上に上げます (MS Windows のみ)。'q' は描画ウィンドウを閉じます。これらのホットキーは、terminal オプションキーワドの "{no}ctrl" を使うことで、ctl-space や ctrl-q に変更できます。しかし、'q' の代わりに ctrl-q を選択するよりもよい方法は、描画ウィンドウのツールウィジェットのトグルスィッチを使うことです。

プログラムのコンパイル時に選択した場所に、gnuplot の外部ドライバ gnuplot_qt がインストールされますが、環境変数 GNUPLOT_DRIVER_DIR を設定することで置き場所を変更することもできます。

Regis

注意: 古い (legacy) 出力形式です。**regis** ドライバは REGIS グラフィック言語での出力を生成します。このドライバには色を 4 色使うか (デフォルト) 16 色使うかのオプションがあります。

書式:

```
set terminal regis {4 | 16}
```

Sixelgd

書式:

```
set terminal sixelgd
       {{no}enhanced} {{no}truecolor} {rounded|butt}
       {linewidth <lw>} {dashlength <dl>}
       {tiny | small | medium | large | giant}
       {font "<face> {,<pointsize>}"} {fontscale <scale>}
       {size <x>,<y>} {anchor|scroll} {{no}transparent}
       {background <rgb_color>}
```

sixel 出力フォーマットは、元々 DEC のターミナルやプリンタで使われていたものです。gnuplot の sixelgd ドライバは、gd ライブラリを内部的に使用して作成した PNG 画像を変換して sixel 出力列を生成します。sixel

出力列は、それを作成したときにターミナル上に表示させることができますが、またはその出力列をファイルに書きだしておいて、後からそのファイルをターミナルに吐き出すことで表示に置き換えることもできます。

sixel 出力形式は、ウィンドウシステムか、またはグラフィックディスプレイマネージャが有効でない場合に、gnuplot のグラフを linux コンソールで表示させたい場合にも便利です。以下参照: **linux console (p. 367)**。

オプション **linewidth** と **dashlength** は拡大率で、描画されるすべての線に影響を与えます。これらは、描画コマンドで要求される値にかけ算されます。

デフォルトでは、sixel 出力は、番号付きの 16 色を使用します。オプション **truecolor** では、24bit RGB PNG 画像を作成し、それを 256 色の sixel 画像出力に減色します。透過型塗り潰しスタイルには、オプション **truecolor** が必要です。以下参照: **fillstyle (p. 283)**。

butt は線分の描画で、その端の点でのはみだしを起こさない描画メソッドを使うようドライバに指示します。この設定は、線幅が 1 より大きい場合にのみ意味があります。これの逆が **rounded** で、多少より均一な曲線を生成しますが、より遅くなります。

gdlib を使用する出力形式におけるフォントの選択の詳細は、やや複雑です。詳細は、以下参照: **fonts (p. 66)**。

出力描画サイズ <x,y> はピクセル単位で与えます。デフォルトは 640x480 ですが、あなたのターミナルウィンドウのサイズより多分小さいでしょう。

transparent は、ドライバに背景色の透明化 (transparent) を行うよう指示しますが、ほとんどの端末エミュレータはこれをサポートしていません。デフォルトは **notransparent** です。

gnuplot の sixelgd 出力は、以下の端末エミュレータ上でテストされ正常動作しています: konsole, mlterm, mintty, vt340 モードの xterm (注意: 元々付属する xterm は sixel グラフィックをサポートするようにビルドされていないかもしれません)。KDE konsole terminal の sixel サポートは、バージョン 22.04.0 で追加されました。

デフォルトでは (**anchor**) 各グラフは、ウィンドウの左上の領域に上書きする形で描画します。これは、再描画することでの場所固定式アニメーション (in-place animation) の作成や、**pause mouse** の間に矢印キーを使った疑似マウス操作を可能にします。それに対しオプション **scroll** は、各グラフを現在のカーソル位置に描くことで、その後のグラフスクロールをテキストを挟むことで行えるようにします。

Svg

このドライバは W3C SVG (Scalable Vector Graphics) フォーマットを生成します。

書式:

```
set terminal svg {size <x>,<y> {|fixed|dynamic}}
           {mouse} {standalone | jsdir <dirname>}
           {name <plotname>}
           {font "<fontname>{,<fontsize>}"} {{no}enhanced}
           {fontscale <multiplier>}
           {rounded|butt|square} {solid|dashed} {linewidth <lw>}
           {background <rgb_color>}
```

ここで <x> と <y> は生成される SVG グラフのサイズですが、**dynamic** は svg ビューワに描画のリサイズを許し、**fixed** は絶対サイズを要求します (デフォルト)。

linewidth <w> は図の中で使用される全ての線の幅を因子 <w> だけ増加させます。

<font> はデフォルトとして使われるフォント名 (デフォルトでは Arial)、<fontsize> はポイント単位でのフォントサイズ (デフォルトは 12) です。svg ビューワソフトは、そのファイルの表示の際には別の代用フォントを使うことになるでしょう。

拡張文字列処理 (enhanced) モードの書式指定は、他の出力形式の場合と同じです。詳細は、以下参照: **enhanced (p. 42)**。

オプション **mouse** は、マウストラッキング機能と、対応する key 上でクリックすることでそれぞれのグラフの描画を On/Off にする機能をサポートを追加することを gnuplot に指示します。デフォルトではローカルディレクトリ、通常は /usr/local/share/gnuplot/<version>/js 内のあるスクリプトを指すリンクを取り込むことで行われますが、オプション **jsdir** に別のローカルディレクトリか、通常の URL を指定することでこれは変更できます。SVG 画像を Web ページに入れるのであれば、普通は後者の URL の方を指定します。一方でオプション **standalone** は、マウス操作プログラムを SVG 文書自体に埋め込み、外部リソースへのリンクは行いません。

SVG ファイルを何かの外部ファイルと組み合わせて利用したい場合、例えばそれがある Web ページや親文書の javascript コードから参照されているような場合、他の SVG グラフへの参照との衝突を避けるために一意的な名前が必要になります。その場合はオプション **name** を使って固有の名前を確保してください。

Svga

古い (legacy) 出力形式です。**svga** ドライバは SVGA グラフィックの PC をサポートします。これは、通常は DJGPP のみでコンパイルされ、GRX グラフィックライブラリを使用しますが、Windows 32bit 用の変種もあり、それは主にデバッグ用です。ベースになるライブラリは、X11 や Linux コンソール、SDL もサポートしていますが、現在それらのターゲットはサポートしていません。

書式:

```
set terminal svga {font "<fontname>"}
                  {{no}enhanced}
                  {background <rgb color>}
                  {linewidth|lw <lw>}
                  {pointscale|ps <scale>}
                  {fontscale|fs <scale>}
```

enhanced オプションにより、拡張文字列処理のサポートが有効になります。以下参照: **enhanced text (p. 42)**。拡張文字列内でのフォントサイズの変更は現在はサポートしていないことに注意してください。よって、上付き、下付き文字も同じサイズになります。

パラメータ **linewidth** は線の幅を拡大します。パラメータ **pointscale** は point 記号の拡大率を設定します。ビットマップフォントを拡大するには、**fontscale** を使うことができます。それは高解像度ディスプレイを使っている場合は有用でしょう。整数倍の拡大率が最も良い結果を生むことに注意してください。

Tek40

このドライバ群は VT-ライクな端末のいくつかをサポートします。**tek40xx** は Tektronix 4010 とその他ほとんどの TEK エミュレータをサポートします。**vttek** は VT-ライクな tek40xx 端末エミュレータをサポートします。以下のものは、gnuplot のコンパイル時に選択されたもののみが利用できます: **kc-tek40xx** はカラーの MS-DOS Kermit Tek4010 ターミナルエミュレータを、**km-tek40xx** はその白黒版をサポートします。**selanar** は Selanar グラフィック端末をサポートします。**bitgraph** は BBN Bitgraph 端末をサポートします。いずれもオプションはありません。

Tek410x

tek410x ドライバは Tektronix 410x, 420x ファミリーをサポートしています。オプションはありません。

Texdraw

texdraw ドライバは (La)TeX texdraw 環境をサポートします。それは texdraw パッケージと共に利用されることを想定しています。以下を参照してください: https://www.ctan.org/tex-archive/graphics/texdraw/ 。

```
set terminal texdraw
               {size <XX>{unit},<YY>{unit}}
               {standalone | input}
               {blacktext | colortext | colourtext}
               {linewidth <lw>} {rounded | butt}
               {pointscale <ps>}
               {psarrows | gparrows} {texpoints | gppoints}
               {background <rgbcolor>}
```

注意: グラフィックは、灰色階調のみです。文字列は常に黒です。箱 (box) と多角形の塗り潰しは、ベタ塗り (solid) の灰色階調のみで、パターン塗りは使えません。

数ある中で、点 (point) は、LaTeX のコマンド "\Diamond", "\Box" などを使って描かれます。これらのコマンドは現在は LaTeX2e のコアには存在せず latexsym パッケージに含まれていますが、このパッケージ基本配布の一部であり、よって多くの LaTeX のシステムの一部になっています。このパッケージを使うことを忘れないでください。他の点種は、amssymb パッケージの記号を使用します。plain TeX との互換性のためには、オプション **gppoints** を指定する必要があります。

standalone は、コンパイルの用意ができた、複数のグラフを持つ可能性のある LaTeX ファイルを生成します。デフォルトは **input** で、他のファイルから include できる TeX ファイルを生成します。

blacktext は、強制的にすべの文字列を黒で書きますが、**colortext** は、「色付き」の文字列を可能にします。デフォルトは **blacktext** で、ここでの「色」は、実際には灰色階調を意味します。

rounded は線の端や接合部を丸くし、**butt** は尖った端と角張った接合部を使用し、これがデフォルトです。

linewidth と **pointscale** は、線幅と点記号のサイズをそれぞれ伸縮します。**pointscale** は、**gppoints** にのみ適用します。

psarrows は、TeXdraw コマンドを用いて **arrow** (矢) を書きます。これはは簡単ですが、どんなオプションも提供しません。**gparrows** は、代わりに、gnuplot 自身の、すべての機能を備えたサブルーチンによる arrow の描画を選択します。同様に、**texpoints** と **gppoints** は LaTeX の記号と gnuplot の点描画ルーチンの選択です。

Tgif

古い (legacy) 出力形式です (gnuplot の configure 時に –with-tgif をつけたときのみ使用可能)。tgif は、Xlib ベースの対話型、2 次元ベクトル形式のドローイングツールでビットマップ画像を取り込んだり、結果をビットマップ化したりできます。

tgif ドライバは、フォント指定、フォントサイズ指定、1 ページ内の複数のグラフ描画をサポートしています。軸の比率は変更されません。

書式:

```
set terminal tgif {portrait | landscape | default} {<[x,y]>}
                  {monochrome | color}
                  {{linewidth | lw} <LW>}
                  {solid | dashed}
                  {font "<fontname>{,<fontsize>}"}
```

<[x,y]> にはそのページ内の x 方向、y 方向のグラフの数を指定し、**color** はカラー機能を有効にし、**linewidth** は全ての線幅を <LW> 倍し、"<fontname>" には有効な PostScript フォント名、<fontsize> はその PostScript フォントの大きさを指定します。**defaults** は全てのオプションの値をデフォルトの値にセットします。デフォルトは **portrait**, **[1,1]**, **color**, **linewidth 1.0**, **dashed**, **"Helvetica,18"** です。

solid オプションは、編集作業中にそうであるように、線がカラーである場合に普通使われます。ハードコピーは白黒になることが多いので、その場合は **dashed** を選択すべきでしょう。

多重描画 (multiplot) は 2 種類の方法で実装されています。

その一つは、標準的な gnuplot の多重描画のやり方です:

```
set terminal tgif
set output "file.obj"
set multiplot
set origin x01,y01
set size  xs,ys
plot ...
     ...
set origin x02,y02
plot ...
unset multiplot
```

より詳しい情報については、以下参照: **set multiplot (p. 247)**。

もう一つの方法はドライバの [x,y] オプションです。この方法の長所は、原点 (origin) や大きさ (size) の設定をしなくても全てのグラフが自動的に縮尺され配置されることです。グラフの比 x/y は、自然な比 3/2 (または **set size** で設定されたもの) が保持されます。

両方の多重描画の実装が選択された場合、標準的なやり方の方が選択され、警告のメッセージが表示されます。

単一描画 (または標準的な多重描画) の例:

```
set terminal tgif                     # デフォルト
set terminal tgif "Times-Roman,24"
set terminal tgif landscape
set terminal tgif landscape solid
```

ドライバの持つ多重描画の仕組みを利用する例:

```
set terminal tgif portrait [2,4]  # 縦置、x-方向に 2 つ、y-方向
                                  # に 4 つのグラフ描画
set terminal tgif [1,2]           # 縦置、x-方向に 1 つ、y-方向
                                  # に 2 つのグラフ描画
set terminal tgif landscape [3,3] # 横置、両方の方向に 3 つのグ
                                  # ラフ描画
```

Tikz

このドライバは、TeX のグラフィックマクロの TikZ パッケージとともに使用する出力を生成します。現在は、外部 lua script によって実装されていて、**set term tikz** は **set term lua tikz** の省略形です。完全な説明は、以下参照: **term lua tikz (p. ??)**。出力形式オプションの表示は、**set term tikz help** を使用してください。

Tkcanvas

このドライバは、以下のスクリプト言語のうちの一つの Tk canvas widget コマンドを生成します: Tcl (デフォルト), Perl, Python, Ruby, REXX。

書式:

```
set terminal tkcanvas {tcl | perl | perltkx | python | ruby | rexx}
                      {standalone | input}
                      {interactive}
                      {rounded | butt}
                      {nobackground | background <rgb color>}
                      {{no}rottext}
                      {size <width>,<height>}
                      {{no}enhanced}
                      {externalimages | pixels}
```

結果を表示するには、以下の Tcl/Tk コマンド列を実行します:

```
package require Tk
# 以下の 2 行は、外部画像を使用する場合にのみ必要
package require img::png
```

```
source resize.tcl
source plot.tcl
canvas .c -width 800 -height 600
pack .c
gnuplot .c
```

Perl/Tk の場合は、以下のようにします:

```
use Tk;
my $top = MainWindow->new;
my $c = $top->Canvas(-width => 800, -height => 600)->pack;
my $gnuplot = do "plot.pl";
$gnuplot->($c);
MainLoop;
```

Perl/Tkx の場合は、以下のようにします:

```
use Tkx;
my $top = Tkx::widget->new(".");
my $c = $top->new_tk__canvas(-width => 800, -height => 600);
$c->g_pack;
my $gnuplot = do "plot.pl";
$gnuplot->($c);
Tkx::MainLoop();
```

Python/Tkinter の場合は、以下のようにします:

```
from tkinter import *
from tkinter import font
root = Tk()
c = Canvas(root, width=800, height=600)
c.pack()
exec(open('plot.py').read())
gnuplot(c)
root.mainloop()
```

Ruby/Tk の場合は、以下のようにします:

```
require 'tk'
root = TkRoot.new { title 'Ruby/Tk' }
c = TkCanvas.new(root, 'width'=>800, 'height'=>600) { pack  { } }
load('plot.rb')
gnuplot(c)
Tk.mainloop
```

Rexx/Tk の場合は、以下のようにします:

```
/**/
```

```
call RxFuncAdd 'TkLoadFuncs', 'rexxtk', 'TkLoadFuncs'
call TkLoadFuncs
cv = TkCanvas('.c', '-width', 800, '-height', 600)
call TkPack cv
call 'plot.rex' cv
do forever
    cmd = TkWait()
    if cmd = 'AWinClose' then leave
    interpret 'call' cmd
end
```

gnuplot が生成するコード (上の例では、"plot.<ext>" として書き出されているものです) は、以下のような手続き関数を含んでいます:

gnuplot(canvas)

引数として canvas の名前を取ります。
これを呼び出すと、その canvas をクリアし、canvas のサイズを探し、その中にグラフを描き、それに合うように伸縮します。

gnuplot_plotarea()

canvas スクリーン座標での描画領域の境界 (xleft, xright, ytop, ybot) を含むリストを返します。
2 次元グラフ描画 (`plot`) に対してのみ動作します。

gnuplot_axisranges()

グラフ座標での 2 つの軸の範囲 (x1min, x1max, y1min, y1max, x2min, x2max, y2min, y2max) を返します。
2 次元グラフ描画 (`plot`) に対してのみ動作します。

オプション **standalone** を使えば、自己完結した最小スクリプトを生成できます。デフォルトは **input** で、これは取り込まれるべきスクリプトを作ります (すなわち、load されるか call されるか、または選択した言語用のなんらかの適切な方法で)。

オプション **interactive** を指定すると、一つの線分上でマウスクリックしたときにその中点の座標が標準出力に出力されるようになります。この動作は、user_gnuplot_coordinates という手続き関数を定義することで、別なものに置き換えることも可能です。その手続き関数には以下の引数が渡されます:

```
win id x1s y1s x2s y2s x1e y1e x2e y2e x1m y1m x2m y2m,
```

これらは、canvas の名前、線分の id、2 つの座標系でのその線分の開始点の座標、終了点の座標、中点の座標です。中点の座標は、対数軸に対してのみ与えられます

デフォルトでは、canvas は **transparent** ですが、オプション **background** で、明示的に背景色を設定することもできます。

rounded は、線の端や接合部を丸くします。デフォルトの **butt** は、尖った端と角張った接合部を使用します。

オプション **rottext** で、文字列の任意角での回転を有効にできますが、それには Tcl/Tk 8.6 以降が必要です。デフォルトは **norottext** です。

オプション **size** は、目盛りの刻みとフォントサイズを、指定された canvas サイズに対して最適化なものにしようとします。デフォルトでは、出力サイズは 800 x 600 ピクセルとしています。

enhanced は、拡張文字列処理を選択します (デフォルト) が、これは今のところ Tcl でのみ利用可能です。

オプション **pixels** (デフォルト) は、フェールセーフなピクセル毎の画像処理ルーチンを選択します。以下参照: **image pixels (p. 114)**。オプション **externalimages** は、画像を外部 PNG 画像として保存し、それをあとで tkcanvas コードが読み込んで伸縮します。このオプションは、Tcl でのみ有効で、しかも Tk の画像処理ルーチンが任意の伸縮を提供していないために、ある状況では遅くなります。この場合、スクリプトで、提供される rescale.tcl を取り込まなければいけません。

対話型モードは、Python/Tk, Rexx/Tk ではまだ実装されていません。Ryby/Tk の対話型モードは、まだ user_gnuplot_coordinates をサポートしていません。

Webp

webp 出力形式は、単一フレーム、またはアニメーションを生成します。実際の描画は、2 次元グラフィックライブラリ cairo と、レイアウト、文字の描画ライブラリ pango が担当します。

書式:

```
set term webp
            {size <x_pixels>,<y_pixels>}
            {font <font>} {fontscale <scale>} {{no}enhanced}
            {{no}transparent} {background <rgbcolor>
            {linewidth <lw>} {rounded|butt|square} {dashlength <dl>}
            {pointscale <ps>}

            {{no}animate {quality <q>} {delay <msec>} {loop <n>}}
```

ンネル画像として、pngcairo 出力形式と共有するルーチンを使って作ります。フォントや terminal オプションに関する詳細は、以下参照: **set terminal pngcairo (p. 376)**。その後、そのフレームを出力時に webp 形式に変換します。

オプション **animate** は、複数のフレームからなる webp ファイルを生成します。個々のフレームは、個別の **plot**、または **splot** コマンドで生成したものです。そのアニメーション列は、コマンド **set output** か、**set terminal** で終了します。

quality (1..100) は、出力ファイルのサイズに影響します。1-74 の q の値は、損失 (loss) のある圧縮を使用します。より小さい値を指定すると、描画する画像の細部の喪失を犠牲にして、より小さいファイルを作成します。75-100 の q の値は、損失のない (lossless) 圧縮を行います。これはすべて同じ画像品質です (lossless!)。より大きな値を指定すると、減らすファイルサイズにおける価値の減少のためにより多くの計算時間を費します。デフォルトは 75 で、余分な計算なしに損失のない圧縮を行います。

サブオプション **delay** は、再生時のフレーム間隔をミリ秒単位で設定します (デフォルトは 50 ミリ秒)。

サブオプション **loop** は、再生時にアニメーション列を何回繰り返して再生するかを指定します。デフォルトの 0 はループし続けます。

Windows

出力形式 **windows** は、グラフ描画と文字列描画に Windows GDI を使用する高速な対話型出力ドライバです。Windows では、複数の環境で動作する **wxt** 出力形式、**qt** 出力形式もサポートされています。

書式:

```
set terminal windows {<n>}
                     {color | monochrome}
                     {solid | dashed}
                     {rounded | butt}
                     {enhanced | noenhanced}
                     {font <fontspec>}
                     {fontscale <scale>}
                     {linewidth <scale>}
                     {pointscale <scale>}
                     {background <rgb color>}
                     {title "Plot Window Title"}
                     {{size | wsize} <width>,<height>}
                     {position <x>,<y>}
                     {docked {layout <rows>,<cols>} | standalone}
                     {close}
```

複数のウィンドウ描画がサポートされています: **set terminal win <n>** で出力が番号 n の描画ウィンドウに送られます。

color, **monochrome** は、カラー出力か白黒出力かの選択で、**dashed** と **solid** は、点線と実線の選択です。**color** では **solid** がデフォルトで、**monochrome** では **dashed** がデフォルトです。**rounded** は、線の端や接合部を丸くし、デフォルトの **butt** は尖った端と角張った接合部を使用します。**enhanced** は拡張文字列処理 (enhanced text mode) の機能 (上付、下付文字やフォントの混在) を有効にします。詳細は以下参照: **enhanced text (p. 42)**。**<fontspec>** は "<fontface>,<fontsize>" の形式で、"<fontface>" は有効な Windows のフォント名で、**<fontsize>** はポイント単位でのフォントの大きさです。この両要素はいずれも省略可能です。以前の版の gnuplot では、**font** キーワードは省略可能で、<fontsize> は引用符なしの数値で与えることができましたが、現在はその形式はサポートしていませんので注意してください。**linewidth**, **fontscale**, **pointscale** で、線の幅、文字サイズ、点記号の大きさを伸縮できます。**title** は、グラフウィンドウのタイトルを変更します。**size** はウィンドウ内の描画領域のピクセル単位での幅と高さを、**wsize** はウィンドウ自身の実際のサイズを、**position** はウィンドウの原点、すなわち左上角のスクリーン上のピクセル単位での位置を定義します。これらのオプションは、ファイル **wgnuplot.ini** のデフォルトの設定を上書きします。

docked は、グラフウィンドウを、wgnuplot のテキストウィンドウの中に埋め込み、**size** と **position** のオプションを無視します。**docked** は、コンソール版の gnuplot では利用できないことに注意してください。このオプションを設定すると、新規のウィンドウに対してデフォルトの値を変更します。最初のデフォルトは、**standalone** です。ドッキングモードでは、グラフの最小列数、行数は追加オプション **layout** で保持できます。指定されたレイアウト以上にグラフがある場合は、行を追加します。複数のグラフは、数値 ID でソートし、行方向に埋めていきます。

他のオプションもグラフメニューや初期化ファイル **wgnuplot.ini** で変更できます。

Windows 版は、非対話型モードでは通常、コマンドラインから与えたファイルの最後に達すると直ちに終了しますが、コマンドラインの最後に **-** を指定した場合は別です。また、このモードではテキストウィンドウは表示せず、グラフのみの表示となりますが、オプションとして **-persist** (x11 版の gnuplot と同じオプション; 従来の Windows のみのオプション **/noend** や **-noend** を使うこともできます) を指定すると gnuplot は終了しなくなります。この場合他の OS での gnuplot の挙動とは異なり、-persist オプション後も gnuplot の対話型コマンドラインを受け付けます。

コマンド **set term** で gnuplot の出力形式を変更した場合、描画ウィンドウはそのまま残りますが、**set term windows close** で描画ウィンドウを閉じることができます。

gnuplot は、Windows 上での出力の生成のためのいくつかの方法をサポートしています。以下参照: **windows printing (p. 398)**。windows 出力形式は、クリップボードや EMF ファイルを通して他のプログラムとのデータの交換をサポートしています。以下参照: **graph-menu (p. 397)**。EMF ファイルを生成するには、**emf** 出力形式を使うこともできます。

グラフメニュー (graph-menu)

gnuplot graph ウィンドウでマウスの右ボタン (*) を押すか、システムメニューやツールバーから **Options** を選択すると以下のオプションを持つポップアップメニューが現われます:

Copy to Clipboard クリップボードにビットマップや EMF 画像をコピー

Save as EMF... 現在のグラフウィンドウをメタファイル (EMF か EMF+) として保存

Save as Bitmap... 現在のグラフをビットマップファイルとして保存

Print... グラフィックウィンドウを Windows プリンタドライバでプリントアウト。プリンタと拡大率の選択が可能。以下も参照: **windows printing (p. 398)**。

Bring to Top チェックを入れるとグラフウィンドウを他の全ての描画ウィンドウの手前に表示

Color チェックを入れるとカラー出力が有効、チェック無しだとすべて灰色階調表示。これは例えば白黒のプリントアウトの見た目の確認に有用です。

GDI backend 古典的な GDI API を使うもので、非推奨であり、このバージョンでは無効になっています。

GDI+ backend GDI+ Windows API を使うスクリーン描画。これは、アンチエイリアス、オーバーサンプリング、透過、点線/破線パターンのカスタマイズのすべてを完全にサポートします。バージョン 5.0, 5.2 ではデフォルトでした。

Direct2D backend Direct2D と DirectWrite API を使用する描画。これは、グラフィックカード加速機能を使用し、よって通常は最も高速です。Direct2D は EMF データを作れませんので、EMF データのクリップボードへのコピーの際には、D2d でビットマップデータを生成している間 GDI+ で作業します。現在はこれを推奨していて、バージョン 5.3 からはこれがデフォルトのバックエンドになります

Oversampling これにより、対角線は非整数のピクセル位置に描画され、左右にふらつくことが避けられます。しかし鉛直線、水平線は、ぼやけた線にならないよう整数のピクセル位置に置かれます。

Antialiasing 折れ線や線の端の平滑化を可能にします。これは描画を遅くすることに注意してください。**Antialiasing of polygons** 多角形描画のアンチエイリアスは、デフォルトでは有効ですが、GDI+ のバックエンドでは描画を遅くする可能性があります。

Fast rotation グラフウィンドウをマウスで回転している際にアンチエイリアスを一時的にオフにします。これは、マウスボタンを離した後に追加の再描画が行われますが、相当に描画を早くしてくれます。

Background... ウィンドウ背景色の設定

Choose Font... グラフィックウィンドウで使うフォントの選択

Update wgnuplot.ini 現在のウィンドウの位置、ウィンドウの大きさ、テキストウィンドウのフォントとそのフォントサイズ、グラフウィンドウのフォントとそのフォントサイズ、背景色を初期化ファイル **wgnuplot.ini** に保存

(*) このメニューは、**unset mouse** によって右マウスボタン押ししか使えなくなるので注意。

印刷 (printing)

好みにより、グラフは以下のような方法で印刷できます。

1. gnuplot のコマンド **set terminal** でプリンタを選択し、**set output** で出力をファイルにリダイレクト

2. gnuplot graph ウィンドウから **Print...** コマンドを選択。テキストウィンドウからこれを行なう特別なコマンド **screendump** もある。

3. **set output "PRN"** とすると出力は一時ファイルに出力され、gnuplot を終了するかまたは **set output** コマンドで出力を他のものへ変更すると、ダイアログ (対話) ボックスが現われ、そこでプリンタポートを選択。そこで OK を選択すると、出力はプリントマネージャでは加工されずにそのまま選択されたポートでプリントアウトされる。これは偶然 (または故意) に、あるプリンタ用の出力を、それに対応していないプリンタに送り得ることを意味する。

テキストメニュー (text-menu)

gnuplot text ウィンドウでマウスの右ボタンを押すか、システムメニューから **Options** を選択すると以下のオプションを持つポップアップメニューが現われます:

Copy to Clipboard マークしたテキストをクリップボードにコピー

Paste 打ち込んだのと同じようにクリップボードからテキストをコピー

Choose Font... テキストウィンドウで使うフォントの選択

System Colors 選択するとコントロールパネルで設定したシステムカラーをテキストウィンドウに与える。選択しなければ白背景で文字は黒か青。

Wrap long lines 選択すると現在のウィンドウ幅よりも長い行を折り返す

Update wgnuplot.ini 現在の設定を、ユーザのアプリケーションデータディレクトリにある初期化ファイル **wgnuplot.ini** に保存

メニューファイル wgnuplot.mnu

メニューファイル **wgnuplot.mnu** が **gnuplot** と同じディレクトリにある場合、**wgnuplot.mnu** に書かれているメニューが読み込まれます。メニューコマンドは以下の通り:

`[Menu]`	次の行の名前で新しいメニューを開始
`[EndMenu]`	現在のメニューを終了
`[--]`	水平なメニューの仕切りを入れる
`[|]`	垂直なメニューの仕切りを入れる
`[Button]`	メニューに押しボタンを入れ、それに次のマクロを割り当てる

マクロは 2 行で書き、最初の行はマクロ名 (メニューの見出し)、2 行目がマクロ本体です。先頭の空白列は無視されます。マクロコマンドは以下の通り:

[INPUT] [EOS] か {ENTER} までをプロンプトとして出力し文字列を入力 [EOS] 文字列の終り (End Of String)。何も出力しない [OPEN] 開くファイル名を取得。最初の [EOS] までが対話ウィンドウのタ

イトル、そこから次の [EOS] か {ENTER} までがデフォルトのファイル名

[SAVE] セーブファイル名を取得 ([OPEN] 同様) [DIRECTORY] ディレクトリ名を取得。[EOS] か {ENTER} までが対話ウィンドウ

のタイトル

マクロ文字の置き換えは以下の通り:

`{ENTER}`	復帰 `'\r'`
`{TAB}`	タブ `'\011'`
`{ESC}`	エスケープ `'\033'`
`{^A}`	`'\001'`
`...`	
`{^_}`	`'\031'`

マクロは展開後の文字数が最大 256 文字に制限されています。

Wgnuplot.ini

Windows テキストウィンドウと windows 出力形式は、オプションのいくつか **wgnuplot.ini** の [**WGNUPLOT**] セクションから読み込みます。このファイルは、ユーザのアプリケーションデータディレクトリに置きます。**wgnuplot.ini** ファイルのサンプル:

```
[WGNUPLOT]
TextOrigin=0 0
TextSize=640 150
TextFont=Consolas,9
TextWrap=1
TextLines=400
TextMaximized=0
SysColors=0
GraphOrigin=0 150
GraphSize=640 330
GraphFont=Tahoma,10
```

```
GraphColor=1
GraphToTop=1
GraphGDI+=1
GraphD2D=0
GraphGDI+Oversampling=1
GraphAntialiasing=1
GraphPolygonAA=1
GraphFastRotation=1
GraphBackground=255 255 255
DockVerticalTextFrac=350
DockHorizontalTextFrac=400
```

以下の設定は wgnuplot のテキストウィンドウのみに適用されます。

TextOrigin と **TextSize** は、テキストウィンドウの位置とサイズの指定です。**TextMaximized** が 0 でない場合、ウィンドウは最大化されます。

TextFont は、テキストウィンドウのフォントとサイズの指定です。

TextWrap は、長いテキスト行の折り返しを選択します。

TextLines は、テキストウィンドウの内部バッファに何行 (折り返しなし) 保持するかを指定します。現在は、この値を wgnuplot 内からは変更できません。

以下参照: **text-menu (p. 398)**。

DockVerticalTextFrac と **DockHorizontalTextFrac** は、それぞれ千分率単位での、テキストウィンドウ内に垂直方向、水平方向に確保される割合です。

GraphFont は、フォント名とポイント単位のフォントサイズの指定です。

以下参照: **graph-menu (p. 397)**。

Wxt

wxt 出力形式は、個々のウィンドウへの出力を生成します。ウィンドウは wxWidgets ライブラリで生成されます (これが **wxt** の名前の由来です)。実際の描画は、2D グラフィックライブラリ cairo と、文字列配置/レンダリングライブラリ pango が処理します。

書式:

```
      set term wxt {<n>}
                   {size <width>,<height>} {position <x>,<y>}
                   {background <rgb_color> | nobackground}
                   {{no}enhanced}
                   {font <font>} {fontscale <scale>}
                   {title "title"}
                   {linewidth <lw>} {butt|rounded|square}
                   {dashlength <dl>}
                   {{no}persist}
```

```
{{no}raise}
{{no}ctrl}
{close}
```

複数の描画ウィンドウもサポートしていて、**set terminal wxt <n>** とすれば番号 n の描画ウィンドウへ出力します。

デフォルトのウィンドウタイトルは、このウィンドウ番号に基づいています。そのタイトルは "title" キーワードでも指定できます。

描画ウィンドウは、**gnuplot** の出力形式を別なものに変更しても残ったままになります。それを閉じるには、そのウィンドウに入力フォーカスがある状態で 'q' を入力するか、ウィンドウマネージャのメニューで **close** を選択するか、**set term wxt <n> close** としてください。

描画領域のサイズはピクセル単位で与えます。デフォルトは 640x384 です。それに加えて、ウィンドウの実際のサイズには、ツールバーやステータスバー用のスペースも追加されます。ウィンドウのサイズを変更すると、描画グラフもウィンドウの新しいサイズにぴったり合うようにすぐに伸縮されます。他の対話型出力形式と違い、**wxt** 出力形式はフォント、線幅も含めて描画全体を伸縮しますが、全体のアスペクト比は一定に保って、空いたスペースは灰色で塗り潰します。その後 **replot** とタイプするかターミナルツールバーの **replot** アイコンをクリックするか新たに **plot** コマンドを入力すると、その新しい描画では完全にそのウィンドウに合わせられますが、フォントサイズや線幅はそれぞれのデフォルトにリセットされます。

position オプションは描画ウィンドウの位置を設定するのに使えます。これはコマンド **set term** 後の最初の描画にのみ適用されます。

現在の描画ウィンドウ (**set term wxt <n>** で選択されたもの) は対話的でその挙動は、他の出力形式と共通です。詳細は、以下参照: **mouse (p. 244)**。それには追加のアイコンもいくつかついていますが、それらはそれ自体が説明的なものになっているはずです。

この出力形式は、拡張文字列処理モード (enhanced text mode) をサポートしていて、フォントや書式コマンド (上付、下付など) をラベルや他の文字列に埋め込むことができます。拡張文字列処理モードの書式は他の gnuplot の出力形式と共通です。詳細は、以下参照: **enhanced (p. 42)**。

<font> は "FontFace,FontSize" の形式で、FontFace と FontSize とをコンマで分離して一つの文字列として書きます。FontFace は、'Arial' のような通常のフォント名です。FontFace を与えない場合は、wxt 出力形式は'Sans' を使用します。FontSize は、ポイント単位のフォントサイズです。FontSize を与えない場合は、wxt 出力形式は 10 ポイントを使用します。

例：

```
set term wxt font "Arial,12"
set term wxt font "Arial" # フォント名のみ変更
set term wxt font ",12" # フォントサイズのみ変更
set term wxt font "" # フォント名、フォントサイズをリセット
```

フォントは通常のフォントサブシステムから取得します。MS-Windows 上ではコントロールパネルの "Fonts" エントリで検索されるので、そこに設定します。Unix 上では、フォントは "fontconfig" が処理します。

文字列のレイアウトに使用される pango ライブラリは utf-8 を基本としていますので、wxt 出力形式ではエンコーディングを utf-8 にする必要があります。デフォルトの入力エンコーディングは、システムの 'locale' によります。他のエンコーディングを使用したい場合は、それを gnuplot に知らせる必要があります。詳細は、以下参照: **encoding (p. 215)**。

pango は、unicode マッピングでないフォントに対しては予期せぬ結果を与えるかもしれません。例えば Symbol フォントに対しては、wxt 出力形式は、文字コードを unicode に変換するために http://www.unicode.org/ で提供されるマッピングを利用します。pango は、その文字を含むフォントを見つけるためにあなたの Symbol フォントを検索し、そして DejaVu フォントのように、幅広く unicode をカバーする他のフォントを探す、といった最善の作業を行おうとします。なお、"the Symbol font" は、Acrobat Reader と一緒に"SY______.PFB" として配布されている Adobe Symbol フォントであると解釈されることに注意してください。この代わりに、OpenOffice.org と一緒に"opens___.ttf" として配布される OpenSymbol フォントが同じ文字を提供しています。Microsoft も Symbol フォント ("symbol.ttf") を配布していますが、これは異なる文字セットになっていて、いくつかは欠けていますし、いくつかは数式記号に変わってしまっています。あなたのデフォルトの設定でなんらかの問題が起きた場合 (例えばデモスクリプト enhancedtext.dem がちゃんと表示されないといった場合) は、Adobe か OpenOffice の Symbol フォントをインストールして、Microsoft の Symbol フォントを削除しないといけないかもしれません。"windings" のような他の非標準のフォントでも動作することが報告されています。

描画のレンダリングは、ツールバーで対話的に変更できます。可能な限り最も良い出力を生成するためにこのレンダリングは、アンチエイリアス、オーバーサンプリング、ヒンティングの 3 つの機構を持っています。アンチエイリアスは、水平や垂直でない線の滑らかな表示を可能にします。オーバーサンプリングは、アンチエイリアスと組でピクセルよりも小さいサイズでの精度を提供し、gnuplot が非整数座標の直線を書けるようになります。これは、対角方向の直線 (例えば 'plot x') が左右に揺れるのを避けます。ヒンティングは、オーバーサンプリングによって引き起こされる水平、垂直方向の線分のぼかしを避けます。この出力形式は、これらの直線を整数座標に揃え、それにより、1 ピクセル幅の直線は本当に 1 つ (1 つより多くも少なくもない) のピクセルで描画します。

デフォルトでは、描画が行われたときにウィンドウはデスクトップの一番上 (最前面) に表示されます。これは、キーワード "raise" で制御できます。キーワード "persist" は、すべての描画ウインドウを明示的に閉じない間は、gnuplot が終了しないようにします。最後に、デフォルトでは <space> キーは gnuplot コンソールウィンドウを上に上げ、'q' は描画ウィンドウを閉じます。キーワード "ctrl" は、それらのキー割り当てを、それぞれ <ctrl>+<space> と <ctrl>+'q' に変更します。これらの 3 つのキーワード (raise, persist, ctrl) は、設定ダイアログ上のやりとりでも設定し、記憶させることができます。

X11

書式:

```
set terminal x11 {<n> | window "<string>"}
                 {title "<string>"}
                 {{no}enhanced} {font <fontspec>}
                 {linewidth LW}
                 {{no}persist} {{no}raise} {{no}ctrlq}
                 {{no}replotonresize}
                 {close}
                 {size XX,YY} {position XX,YY}
set terminal x11 {reset}
```

複数のグラフ描画ウィンドウをサポートしています。**set terminal x11 <n>** は番号 n の描画ウィンドウに出力します。n が 0 でなければ、タイトルが明示的に指定されていなければその番号がウィンドウタイトルと

してつけられ、アイコンには **Gnuplot <n>** とラベル付けされます。現在有効なウィンドウはカーソル記号の変化で区別できます (デフォルトカーソルから十字カーソルへ)。

x11 出力形式は、外部のアプリケーションによって生成されている X のウィンドウの X ID (16 進表記) をオプション **window** の後に文字列として指定することで、そのウィンドウと接続できます。X は複数のクライアントにイベント ButtonPress の選択を認めないため、gnuplot その外部ウィンドウをコンテナとして使用します。この方法により、gnuplot のマウス機能はとりこまれた描画ウィンドウ内でも作動します。

```
set term x11 window "220001e"
```

x11 出力形式は、利用可能なフォントの元で拡張文字列処理モード (以下参照: **enhanced (p. 42)**) をサポートしています。文字列に埋め込まれ、様々な効果を与えるフォントサイズ命令のために、デフォルトの x11 フォントがスケーラブルフォントである必要があります。よって、以下の最初の例はうまくいくでしょうが、2 番目のものはそうではないでしょう。

```
set term x11 enhanced font "arial,15"
set title '{/=20 Big} Medium {/=5 Small}'

set term x11 enhanced font "terminal-14"
set title '{/=20 Big} Medium {/=5 Small}'
```

gnuplot ドライバが別な出力ドライバに変更されても、描画ウィンドウは開いたままになります。描画ウィンドウは、そのウィンドウにカーソルを置いて文字 q を押すか、ウィンドウマネージャのメニューの **close** を選択すれば閉じることができます。**reset** を実行すれば全てのウィンドウを一度に閉じれます。それは実際にウィンドウを管理している子プロセスを終了します (もし **-persist** が指定されていなければ)。コマンド **close** は、個々の描画ウィンドウを番号を指定して閉じるのに使うことができます。しかし、persist のために残っているような描画ウィンドウは **close** コマンドでは閉じることはできません。番号を省略して **close** した場合には現在有効な描画ウィンドウを閉じます。

gnuplot の外にあるドライバ gnuplot_x11 は、プログラムのコンパイル時に選択されたデフォルトの場所が検索されます。これは環境変数 GNUPLOT_DRIVER_DIR を異なる場所と定義することで変更できます。

描画ウィンドウは **-persisit** オプションが与えられていなければ、対話の終了時に自動的に閉じられます。

オプション **persist** と **raise** はデフォルトでは設定されていませんが、それは、デフォルトの値 (persist == no で raise == yes) か、コマンドラインオプション -persist / -raise の指定か、または X のリソース値が使われる、ということを意味します。[no]persist か [no]raise が指定されるとそれはコマンドラインオプションや X リソースの設定よりも優先されます。これらのオプションの設定は直ちに効力を持ちますので、既に起動しているドライバの挙動は変更されます。ウィンドウを前面に出せない場合は、以下参照: **raise (p. 189)**。

オプション **replotonresize** (デフォルトで有効) は、描画ウィンドウのリサイズ時にデータを再描画します。このオプションなしだと、アスペクト比の変わらない拡大であっても、リサイズ後にウィンドウの一部分にしか描画されない可能性があります。このオプションを使えば、gnuplot は各リサイズイベント毎に完全な再描画を行いますので、枠内をより綺麗に使ってくれます。リサイズの間の再描画による潜在的な CPU への負荷が心配でない場合、このオプションは普通は望ましいものです。再描画は、ホットキー 'e' や 'replot' コマンドで手動で実行することも可能です。

オプション **title "<title name>"** は現在の描画ウィンドウに、または番号を指定すればその番号の描画ウィンドウに対するウィンドウタイトル名をつけます。そのタイトルが表示される場所、または表示されるかどうかは、使っている X のウィンドウマネージャに依存します。

オプション size は、描画ウィンドウのサイズを設定するのに使用できます。このオプションは、その後に生成するウィンドウのみに適用されます。

オプション position は、描画ウィンドウの位置を設定するのに使えます。このオプションは、その後に生成するウィンドウのみに適用されます。

描画サイズとアスペクト比は、**gnuplot** のウィンドウをリサイズすることでも変更できます。

線の幅と点のサイズは **gnuplot** の **set linestyle** で変更可能です。

出力ドライバ **x11** に関しては、**gnuplot** は (起動時に)、コマンドライン、または設定ファイルから、geometry や font, name などの通常の X Toolkit オプションやリソースの指定を受け付けます。それらのオプションについては X(1) マニュアルページ (やそれと同等のもの) を参照してください。

他にも **x11** 出力形式用の多くの **gnuplot** のオプションがあります。これらは **gnuplot** を呼ぶときにコマンドラインオプションとして指定するか、または設定ファイル ".Xdefaults" のリソースとして指定できます。これらは起動時に設定されるので、**gnuplot** 実行時には変更できません (**persist** と **raise** 以外は)。

X11 のフォント (x11_fonts)

初期起動時は、システムの設定か、ユーザの .Xdefaults ファイルの設定か、コマンドライン指定か、のいずれかによる X11 リソースによってデフォルトのフォントが選択されます。

例:

```
gnuplot*font: lucidasans-bold-12
```

新しいデフォルトのフォントを、gnuplot 内部から x11 ドライバに以下のようにして指示することもできます:

```
‘set term x11 font "<fontspec>"‘
```

まず x11 ドライバは、与えられたフォントの正式名を X サーバに尋ねます。この問い合わせが失敗した場合、<fontspec> を"<font>,<size>,<slant>,<weight>" と解釈し、以下の形の完全な X11 フォント名を生成しようとします:

```
-*-<font>-<weight>-<s>-*-*-<size>-*-*-*-*-*-<encoding>
```

```
<font> はフォントの基本名 (base name) (例: Times, Symbol)
<size> はポイントサイズ (指定がなければデフォルトは 12)
<s> は <slant>=="italic" なら 'i', <slant>=="oblique" なら 'o', その他は 'r'
<weight> は明示的に指定されれば 'medium' か 'bold'、その他は '*'
<encoding> は現在の文字集合に基づいて設定。
```

よって **set term x11 font "arial,15,italic"** は (デフォルトの encoding だとすれば) -*-arial-*-i-*-*-15-*-*-*-*-*-iso8859-1 に変換されます。<size>, <slant>, <weight> 指定はいずれも必須ではありません。<slant> や <weight> を指定しなかった場合は、フォントサーバが最初に見つけた、変種のフォントを取得するかもしれません。デフォルトのエンコーディングは、対応する X11 リソースを使って設定することもできます。例:

```
gnuplot*encoding: iso8859-15
```

x11 ドライバは、一般的な PostScript フォント名も認識し、それと同等で有効な X11 フォントか TrueType フォントに置き換えます。これと同じ手順が、**set label** の要求によるフォントの生成でも使われています。

あなたの gnuplot が configure の –enable-x11-mbfonts オプションをつけてインストールされたものなら、フォント名の前に "mbfont:" をつけることでマルチバイトフォントを指定することができます。フォント名を複数指定する個ともできますが、その場合はセミコロンで区切ります。マルチバイトフォントのエンコーディングは locale の設定に従いますので、環境変数 LC_CTYPE を適切な値 (例えば ja_JP.eucJP, ko_KR.EUC, zh_CN.EUC など) にに設定する必要があります。

例:

```
set term x11 font 'mbfont:kana14;k14'
      # 'kana14' と 'k14' は日本語の X11 font エイリアス名、';'
      # はフォント名の区切りです。
set term x11 font 'mbfont:fixed,16,r,medium'
      # <font>,<size>,<slant>,<weight> 形式も使用できます。
set title '(mb strings)' font 'mbfont:*-fixed-medium-r-normal--14-*'
```

同じ書式は X のリソースでのデフォルトフォントの設定でも有効です。例:

```
gnuplot*font: \
    mbfont:-misc-fixed-medium-r-normal--14-*-*-*-c-*-jisx0208.1983-0
```

gnuplot が –enable-x11-mbfonts でインストールされた場合、"mbfont:" をつけなくても 2 つの特別な PostScript フォント名 'Ryumin-Light-*', 'GothicBBB-Medium-*' (標準的な日本語 PS フォント) を使うこともできます。

コマンドラインオプション (command-line_options)

X Toolkit オプションに加え、以下のオプションが gnuplot の立ち上げ時のコマンドラインで、またはユーザのファイル ".Xdefaults" 内のリソースとして指定できます (**raise** と **persist** は **set term x11 [no]raise [no]persist** によって上書きされることに注意してください):

'-mono'	カラーディスプレイ上で強制的に白黒描画
'-gray'	グレイスケールまたはカラーディスプレイ上でのグレイスケール描画 (デフォルトではグレイスケールディスプレイは白黒描画を受け付ける)
'-clear'	新しい描画を表示する前に (瞬間的に) 画面を消去
'-tvtwm'	geometry オプションによる位置の指定を、仮想ルートウィンドウ中の 現在の表示部分に対する相対的な位置にする
'-raise'	各描画後に描画ウィンドウを最前面へ出す
'-noraise'	各描画後に描画ウィンドウを最前面へ出すことはしない
'-persist'	gnuplot プログラム終了後も描画ウィンドウを残す

上記のオプションはコマンドライン上での指定書式で、".Xdefaults" にリソースとして指定するときは異なる書式を使います。

例:

```
gnuplot*gray:  on
gnuplot*ctrlq: on
```

gnuplot は描画スタイル **points** で描画する点のサイズの制御にも、コマンドラインオプション (**-pointsize <v>**) とリソース (**gnuplot*pointsize: <v>**) を提供しています。値 **v** は点のサイズの拡大率として使われ

る実数値 (0 < v <= 10) で、例えば **-pointsize 2** はデフォルトのサイズの 2 倍、**-pointsize 0.5** は普通のサイズの半分の点が使われます。

-ctrlq スイッチは、描画ウィンドウを閉じるホットキーを **q** から **<ctrl>q** に変更します。これは、**pause mouse keystroke** によるキーストロークの保存機能を使っている場合には、他のアルファベット文字と同様に **q** を保存できるようになるので有用でしょう。同じ理由で、**-ctrlq** スィッチは <space> ホットキーも <ctrl><space> に置き換えます。

カラーリソース (color_resources)

注意: このセクションは、gnuplot バージョン 5 とは大きくずれています。x11 出力形式は以下のリソース (ここではそのデフォルトの値を示します)、または白黒階調 (greyscale) のリソースを参照します。リソースの値はシステム上の X11 rgb.txt ファイルに書かれている色名、または 16 進の色指定 (X11 のマニュアルを参照) か、色名と強度 (0 から 1 の間の値) をコンマで区切った値を使用できます。例えば **blue, 0.5** は半分の強度の青、を意味します。

```
gnuplot*background: white
gnuplot*textColor: black
gnuplot*borderColor: black
gnuplot*axisColor: black
gnuplot*line1Color: red
gnuplot*line2Color: green
gnuplot*line3Color: blue
gnuplot*line4Color: magenta
gnuplot*line5Color: cyan
gnuplot*line6Color: sienna
gnuplot*line7Color: orange
gnuplot*line8Color: coral
```

これらに関するコマンドラインの書式は、背景 (bacground) に関しては単純で通常の X11 toolkit オプションの "-bg" に直接対応します。他のものも、全て一般的なリソースの上書きオプション "-xrm" を使うことで設定できます。

例:

背景色を変更するには

```
gnuplot -background coral
```

線種 1 番目の色を書き換えるには

```
gnuplot -xrm 'gnuplot*line1Color:blue'
```

灰色階調リソース (grayscale_resources)

-gray を選択すると、**gnuplot** は、グレイスケールまたはカラーディスプレイに対して、以下のリソースを参照します (ここではそのデフォルトの値を示します)。デフォルトの背景色は黒であることに注意してください。

gnuplot*background: black
gnuplot*textGray: white
gnuplot*borderGray: gray50
gnuplot*axisGray: gray50
gnuplot*line1Gray: gray100
gnuplot*line2Gray: gray60
gnuplot*line3Gray: gray80
gnuplot*line4Gray: gray40
gnuplot*line5Gray: gray90
gnuplot*line6Gray: gray50
gnuplot*line7Gray: gray70
gnuplot*line8Gray: gray30

線描画リソース (line_resources)

注意: このセクションは、gnuplot バージョン 5 とは大きくずれています。**gnuplot** は描画の線の幅 (ピクセル単位) の設定のために以下のリソースを参照します (ここではそのデフォルトの値を示します)。0 または 1 は最小の線幅の 1 ピクセル幅を意味します。2 または 3 の値によってグラフの外観を改善できる場合もあるでしょう。

gnuplot*borderWidth: 1
gnuplot*axisWidth: 0
gnuplot*line1Width: 0
gnuplot*line2Width: 0
gnuplot*line3Width: 0
gnuplot*line4Width: 0
gnuplot*line5Width: 0
gnuplot*line6Width: 0
gnuplot*line7Width: 0
gnuplot*line8Width: 0

gnuplot は線描画で使用する点線の形式の設定用に以下のリソースを参照します。0 は実線を意味します。2 桁の 10 進数 **jk** (**j** と **k** は 1 から 9 までの値) は、**j** 個のピクセルの描画に **k** 個の空白のピクセルが続くパターンの繰り返しからなる点線を意味します。例えば '16' は 1 個のピクセルの後に 6 つの空白が続くパターンの点線になります。さらに、4 桁の 10 進数でより詳細なピクセルと空白の列のパターンを指定できます。例えば、'4441' は 4 つのピクセル、4 つの空白、4 つのピクセル、1 つの空白のパターンを意味します。以下のデフォルトのリソース値は、白黒ディスプレイ、あるいはカラーや白黒階調 (grayscale) ディスプレイ上の白黒描画における値です。カラーディスプレイでは dashed:off がデフォルトになっています。

```
gnuplot*dashed: off
gnuplot*borderDashes: 0
gnuplot*axisDashes: 16
gnuplot*line1Dashes: 0
gnuplot*line2Dashes: 42
gnuplot*line3Dashes: 13
gnuplot*line4Dashes: 44
gnuplot*line5Dashes: 15
gnuplot*line6Dashes: 4441
gnuplot*line7Dashes: 42
gnuplot*line8Dashes: 13
```

X11 pm3d リソース (pm3d_resources)

注意: このセクションは、gnuplot バージョン 5 とは大きくずれています。

デフォルトでは **gnuplot** はそのスクリーンのデフォルトの visual を使用します。割り当てることのできる色数は選択された visual クラスによって変わります。12bit を超える深度を持つ visual クラス上では、gnuplot は最大色数である 0x200 (=512) 色で起動します。8bit を超える (12bit 以下で) 深度の visual クラスでは最大色数は 0x100 (=256) 色、8bit 以下のディスプレイでは最大色は 240 (16 色は曲線の色用に取られる) になります。

gnuplot は最初に、上に述べたような最大色を割り当てようと起動します。これに失敗するとその色数は、gnuplot がその全部を割り付けることができるまで、1/2 ずつ減らされます。**maxcolors** を繰り返し 2 で割った結果、**mincolors** よりも小さい数字になった場合、**gnuplot** は private カラーマップを使おうとします。この場合、ウィンドウマネージャは、ポインタが X11 ドライバのウィンドウに入るか出るかでカラーマップを退避 (swapping) させる責任を持つことになります。

mincolors のデフォルトの値は maxcolors / (num_colormaps > 1 ? 2 : 8) で、num_colormaps は gnuplot が現在使用しているカラーマップの数で、これは、x11 のウィンドウが 1 つだけ開いているような通常の場合は 1 です。

X11 の他のリソース (other_resources)

デフォルトでは、現在の描画ウィンドウの内容は、ウィンドウの X イベントに従って X11 クリップボードに送られます。'gnuplot*exportselection' のリソースの値を 'off' か 'false' と設定することによりこれを無効にできます。

デフォルトでは、文字の回転は速くそれを行なう方法が使われますが、背景色によってはその付近が汚れることがあります。これが起こる場合は、リソース'gnuplot.fastrotate' を 'off' にしてみてください。

```
gnuplot*exportselection: off
gnuplot*fastrotate: on
gnuplot*ctrlq: off
```

Xlib

xlib ドライバは X11 Windows System をサポートしています。このドライバは gnuplot_x11 への命令を生成しますが、**set output '<filename>'** を指定するとそれらをファイルに書き出します。**set term x11** は、**set output "|gnuplot_x11 -noevents"; set term xlib** と同値です。**xlib** には **x11** と同じオプションの組を与えることができます。

第 V 部

Index

索 引