

gnuplot 入門

桂田 祐史

2002年5月19日, 2008年7月23日, 2010年1月9日, 6月8日, 2019年2月6日,
2025年11月26日

この文書の最新版は

<https://m-katsurada.sakura.ne.jp/labo/howto/intro-gnuplot/>

で読める。印刷用 PDF ファイルは

<https://m-katsurada.sakura.ne.jp/labo/howto/intro-gnuplot.pdf>

で入手可能である。

1 gnuplot とは

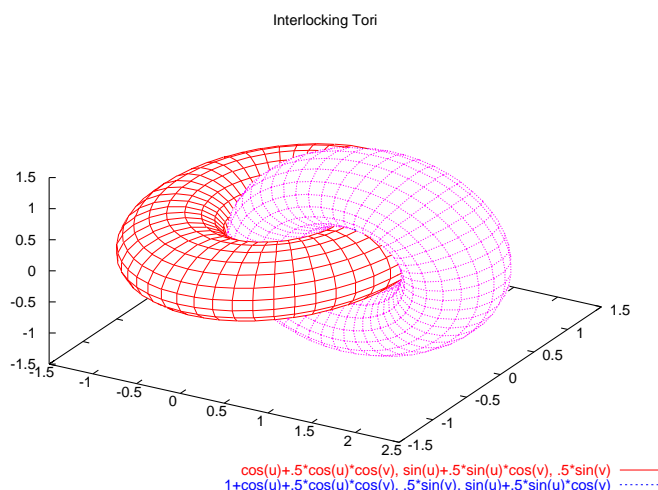
gnuplot¹ は、各種関数のグラフなど曲線・曲面の描画、計算データのプロットなどをするために便利なツールです。

gnuplot はフリーソフトで、色々な環境 (例えば、各種 UNIX のみならず、Win32 環境, WINDOWS 3.1, MS-DOS, MacOS などのパソコン OS でも使えます) に移植されています。使い方を覚えておくと重宝します。

2 文法メモなど

- 起動・終了は次の二つのいずれか

対話的利用 ターミナル等から `gnuplot` で起動し、プロンプト `gnuplot>` に対して、`quit` または `exit` コマンドを入力して終了する。



¹<http://www.gnuplot.info/>

```

Script started on Sun May 26 02:00:26 2002
oyabun% gnuplot

G N U P L O T
Unix version 3.7
patchlevel 1 (+1.1.9 1999/11/08)
last modified Fri Oct 22 18:00:00 BST 1999

Copyright(C) 1986 - 1993, 1998, 1999
Thomas Williams, Colin Kelley and many others

Type 'help' to access the on-line reference manual
The gnuplot FAQ is available from
<http://www.ucc.ie/gnuplot/gnuplot-faq.html>

Send comments and requests for help to <info-gnuplot@dartmouth.edu>
Send bugs, suggestions and mods to <bug-gnuplot@dartmouth.edu>

Terminal type set to 'x11'
gnuplot> load "surface2.dem"
Hit return to continue (1)
Hit return to continue (2)
(中略)
Hit return to continue (8)
Hit return to continue (9)
gnuplot> quit
oyabun%

```

バッチ処理 gnuplot コマンドファイル名 で起動。コマンド・ファイルの最後まで実行すれば終了する。

- 基本的に 1 行 1 コマンド。
- 長いコマンドは “\” で次の行に継続して記述できる²。
- 名前は大文字、小文字を区別する。
- “#” から行末まではコメントとなる。これはコマンド・ファイルのみならず、数値データ・ファイルについても該当する。
- オンライン・マニュアルがあり、help あるいは help キーワードとして使える。
- ファイルの名前は引用符 " " で括る。
plot "bessel.data" や load "bessel.g" のように使う。
- replot コマンドで直前の描画コマンドを再実行できる (オプションを変えて試すのに便利。また重ね描きも可能。)

²JIS 端末では、バックスラッシュ “\” は “¥” に見える。

- `set` コマンドで変更したオプションをすべて破棄するには、`reset` コマンドを用いる。

3 関数のグラフを描こう

(現象数理学科 Mac の場合は Mathematica が利用できるのですが、そちらの方が便利かもしれないけれど…)

3.1 1 変数関数のグラフ

`plot x` の式 とすると、1 変数関数のグラフが描ける³。

```
gnuplot> plot sin(x)
```

描いたものは `clear` コマンドで消せるが、`plot` コマンドを使うと古い図は消えるので、あまり使う必要はないかもしれない。

式をカンマで区切って複数書くことで、複数のグラフを重ねて描ける。

```
gnuplot> plot sin(x),cos(x)
```

変数 (x 座標) の範囲を指定することができる。

```
gnuplot> plot [-pi:pi] sin(x),cos(x)
```

あるいは事前に

```
gnuplot> set xrange [-pi:pi]
```

のように指定しておくこともできる (`unset xrange` でクリアできる)。

ここで `pi` という組み込みの定数を用いた。

おまけ: `pi` について確認

```
gnuplot> print pi ← print コマンドで式の値が表示できる  
3.14159265358979 → 確かに円周率
```

複雑な式の関数を何度も使う場合は、関数定義をしておくと便利である。

```
gnuplot> f(x)=x**3-3*x**2+4*x-5 ←  $f(x) = x^3 - 3x^2 + 4x - 5$  を定義  
gnuplot> plot f(x)
```

```
gnuplot> f(x)=exp(-x*x/2)/sqrt(2*pi) ← 標準正規分布の確率密度関数を定義  
gnuplot> plot [-3:3] f(x)
```

値 (y 座標) の範囲も指定することができる。

³`gnuplot` がどういう組み込み関数を持っているか、`help functions` で調べられる。基本的には UNIX の数学関数ライブラリと同じ。

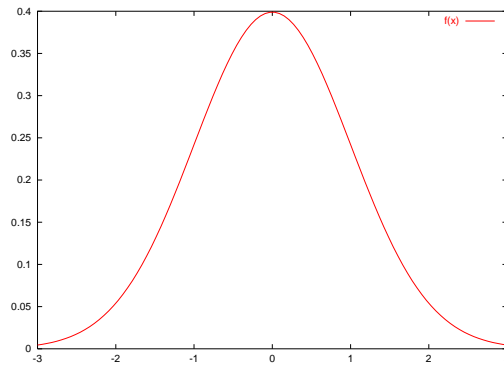


図 1: 正規分布の確率密度関数 ($\pm 3\sigma$ の範囲)

```
gnuplot> plot [-pi/2:pi/2] [-10:10] tan(x) ← x, y 両方とも指定
gnuplot> plot [] [-10:10] tan(x) ← y 座標のみ指定
```

簡単な場合分けは、C 言語風の三項演算子 `?` で実現できる。

```
gnuplot> f(x)=(abs(x)<1)? (x**2-1)**4:0 ←  $|x| < 1$  で  $(x^2 - 1)^4$ , それ以外で 0
gnuplot> plot [-2:2] [-0.1:1.1] f(x)
```

変化が激しい関数の場合、関数値を計算する点の個数を (デフォルトの 100 から) 増やす必要がある。

```
gnuplot> plot sin(100*x)
gnuplot> set samples 10000
gnuplot> plot sin(100*x)
```

`replot` というコマンドがある。画面に表示して確認したものをファイルに保存する目的で、再描画するために使われることが多いが (後述する)、重ね書きする目的で使うこともできる。

```
gnuplot> plot [-1:1] x**2
gnuplot> replot x**3
gnuplot> replot x**4
```

同じことは (上で説明したように) `,` で区切って並べても良い。

```
gnuplot> plot [-1:1] x**2,x**3,x**4
```

この場合は `for` による繰り返しを使っても実現できる。

```
gnuplot> plot [-1:1] for [i=2:4] x**i
```

さらに `for` の少し凝った使い方 (`title` を用いた凡例指定の中で繰り返しの変数を用いる)。

```
gnuplot> plot for [i=1:10] 0.2*i*exp(x) title sprintf("%.1f*exp(x)", i*0.2)
```

(`sprintf()` は C 言語で同じものの書式指定で文字列を作る関数である (説明略)。)

3.2 2変数関数のグラフ

`splot x` と `y` の式 とすると、2 変数関数のグラフが描ける。`set hidden3d` とすると、いわゆる隠線消去を行なう。また `set contour` とすると、等高線を描く⁴。

```
gnuplot> splot x**2-y**2
gnuplot> set hidden3d           ← 隠線消去を指定
gnuplot> set contour           ← 等高線描画を指定
gnuplot> set isosamples 40,40 ← メッシュを細かく
gnuplot> replot                ← 再描画
gnuplot> set nohidden3d        ← 隠線消去指定を外す
gnuplot> set nocontour        ← 等高線描画指定を外す
```

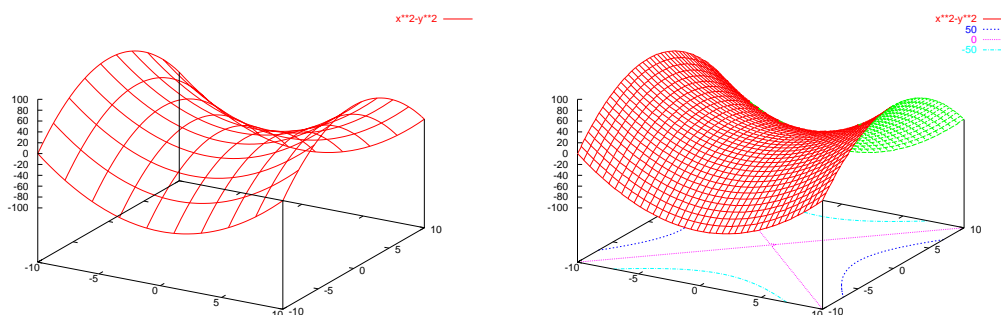


図 2: $z = x^2 - y^2$ のグラフ (素朴 vs 隠線消去&等高線)

4 数値データをプロットしよう

4.1 説明用の (詰まらないけれど我慢してください) 例

$[0, \pi/2]$ の範囲を 90 等分して、各分点における \sin , \cos の値を計算したデータ・ファイル “table.data” があつたとする (0° から 90° までの 1° ごとの三角関数表)。

⁴このような `splot` の描画のコントロールについて知りたければ、もちろん `help splot` とする

```
table.data
#度 sin cos (# から行末まではコメントとなる)
0 0.000000 1.000000
1 0.017452 0.999848
2 0.034899 0.999391
3 0.052336 0.998630
4 0.069756 0.997564
5 0.087156 0.996195
(中略)
85 0.996195 0.087156
86 0.997564 0.069756
87 0.998630 0.052336
88 0.999391 0.034899
89 0.999848 0.017452
90 1.000000 0.000000
```

```
gnuplot> plot "table.data"
```

とすると sin のグラフが描ける。

点を結んで折れ線グラフにするには、with lines (with l と短縮可) か、with linespoints (with lp と短縮可) をつける⁵。

なお with boxes とすると、棒グラフが描ける。

```
gnuplot> plot "table.data" with lines
```

1, 3 列目のデータを選ぶには using 1:3 とする。これで cos のグラフが描ける。

```
gnuplot> plot "table.data" using 1:3 with lines
```

関数のグラフ描画のときと同様に、複数データのプロットが出来る。“,” で区切って並べるだけである。

```
gnuplot> plot "table.data" using 1:2 with lines, \ ← “\” でコマンドを次の行に継続できる
          "table.data" using 1:3 with lines
```

(この長たらしいコマンドにぞっとしている人に: 長いコマンドはファイルに入れてから実行しよう。第 5 節「コマンド・ファイルを実行する」を見よ。)

4.2 対数グラフを描く

量 y が量 x の中に比例する場合、それを確認するには、両側対数目盛りのグラフ用紙にプロットすれば良いというのは、理科の実験の常識であるが、それを gnuplot で実行するには、set logscale xy を実行すれば良い。対数目盛りを解除するには set nologscale とする。

⁵打つ点のタイプは、pointtype 番号 として指定する。

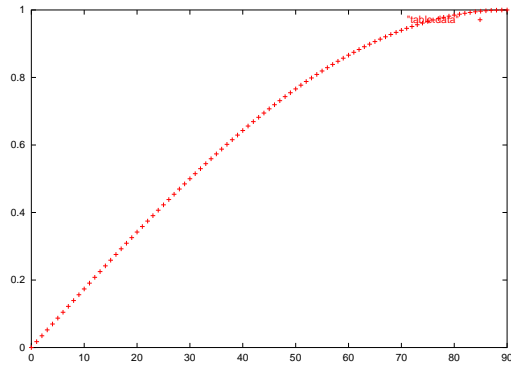


图 3: plot "table.data"

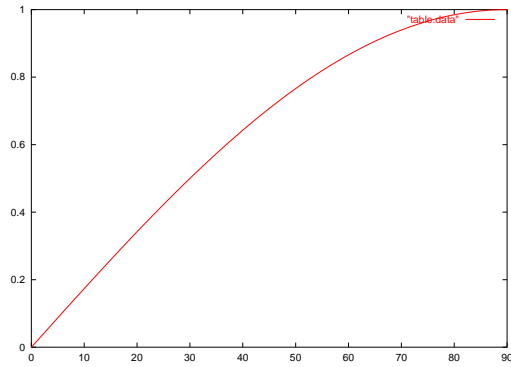


图 4: plot "table.data" with lines

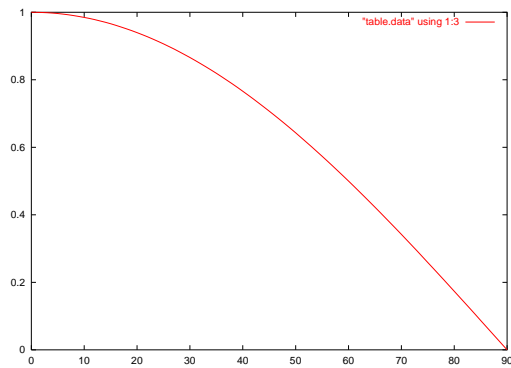


图 5: plot "table.data" using 1:3 with lines

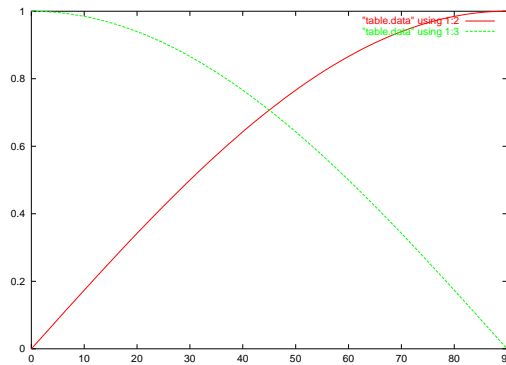


図 6: `plot "table.data" using 1:2 with lines, "table.data" using 1:3 with lines`

“error.tbl” を通常の日盛りと両側対数日盛りの両方でプロットする

```
gnuplot> plot "error.tbl" with lines
```

```
gnuplot> set logscale
```

```
gnuplot> plot "error.tbl" with lines
```

```
gnuplot> set nologscale
```

← 通常の日盛りに戻しておく

(縦軸横軸両方とも対数日盛りにするために、以前は `set logscale xy` と指定していたが、単に `set logscale` で良い、というのを学生に教わった。)

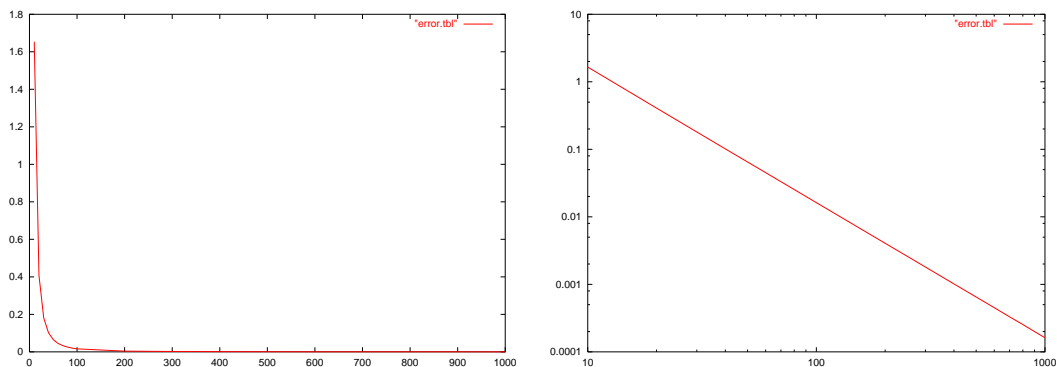


図 7: 通常の日盛りと両側対数日盛り

片側対数グラフ 横軸だけ対数日盛りにするには `set logscale x`, 縦軸だけ対数日盛りにするには `set logscale y` とすれば良い。

対数グラフで負の値を処理するには 例えば誤差を符号付きで記録しているファイルがあるとき、普通に `using 1:2` としたのでは (対数関数の引数に負の値は渡せない) エラーが発生する。これを避けるには `using ($1):(abs($2))` のように、絶対値を取ることを明示すればよい。

(2016/5/12 記) 軸の日盛りに、従来は指数形式で数値を表示していた (と記憶している) が、普通の小数で書くようになってしまっている。0.0000000000000001 なんてのをスクリーンで見て、0 がいくつあるか数えるのは大変なので、指数形式で表示したい。

```
set format y "10^{%L}"
```

```
set format x "10^{%L}"
```

のように指定すると指数形式で書いてくれる。

(2019/2/6 記) 対数プロットして直線状になったとして、傾きを測る or 確認することがしばしば行われる。真面目にやるならば1次回帰曲線を求めたりするのもかもしれないが、大まかに調べるために、脇に三角形を描く方法が最近良く使われている。例えば上のデータの場合、誤差が概ね N^{-2} に比例すると考えられるので、斜辺の傾きが -2 の直角三角形を描く。

```
draw-error.gp
# draw-error.gp

set logscale
set format x "10^{%L}"
set format y "10^{%L}"

# 斜辺の傾きが -2 の三角形の準備
a=1000.0; # グラフの中に入るように三角形の高さを調整
x1=500; x2=900;
y1=a*x1**(-2); y2=a*x2**(-2);
# print sprintf("x1=%f, y1=%e, x2=%f, y2=%e", x1,y1,x2,y2);
set object 1 polygon from x1,y1 to x2,y1 to x2,y2 to x1,y1 fs empty border

plot "error.tbl" with linespoints

set term png
set output "draw-error.png"
replot
```

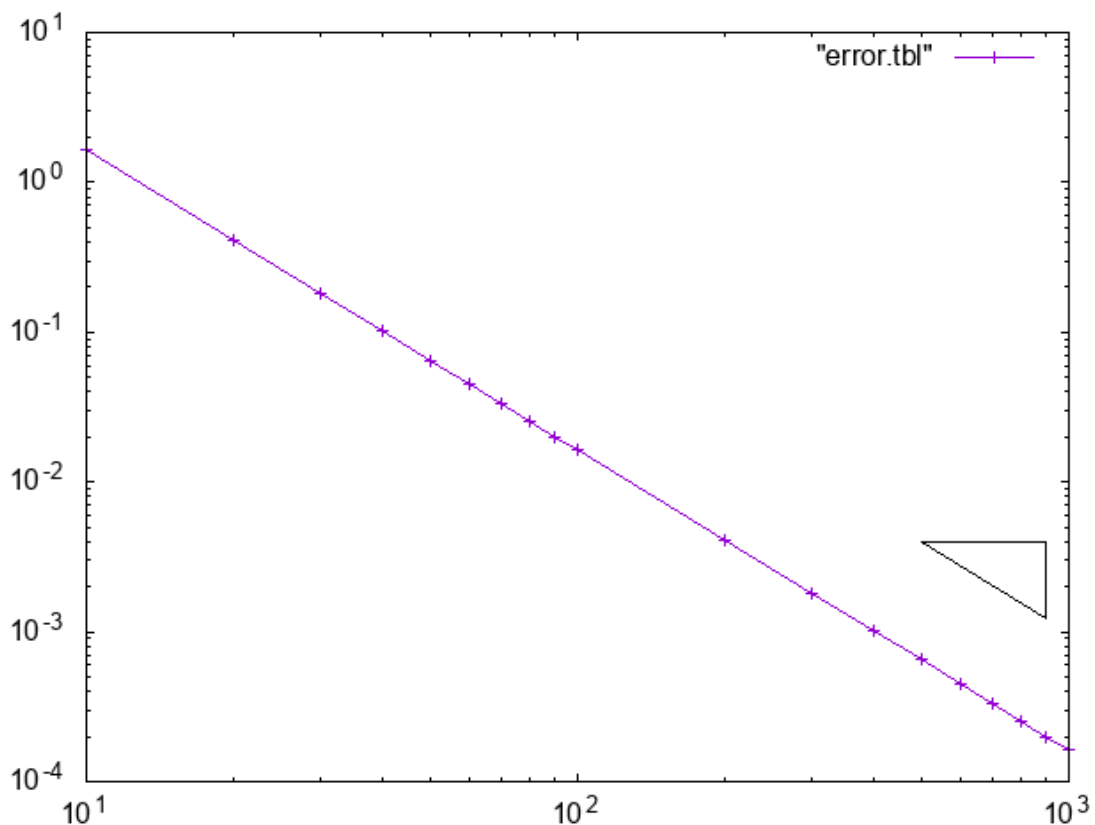


図 8: draw-error.gp の結果: 斜辺の傾きが -2 の直角三角形を添える

(もっとうまい描き方はないものか…)

4.3 例: 有限要素法で解いた Poisson 方程式の境界値問題の解の可視化

(工事中: 説明が舌足らずですが、とりあえず実例を。なお、付録 B.2 に差分法の解の可視化例の紹介をしている。そちらは格子点 (x_i, y_j) での値 $u(x_i, y_j)$ からグラフを描く話で、本来はそれを詳しく解説すべきものだったかも…)

単位円盤領域 $\Omega := \{(x, y) \in \mathbf{R}^2; x^2 + y^2 < 1\}$ における Poisson 方程式の境界値問題

$$-\Delta u(x, y) = f(x, y) \quad \text{in } \Omega, \quad u(x, y) = 0 \quad \text{on } \Gamma := \partial\Omega$$

は、次のような FreeFEM++⁶ プログラムで解ける (ここでは $f(x, y) = xy$ としている)。

FreeFEM++ プログラム poisson-disk.edp

```
// 境界の定義 (単位円), いわゆる正の向き
border Gamma(t=0,2*pi) { x=cos(t); y=sin(t); }
// 三角形要素分割を生成 (境界を 50 に分割)
mesh Th = buildmesh(Gamma(50));
plot(Th,ps="mesh.eps");
plot(Th,wait=1);
// 有限要素空間は P1 (区分的 1 次多項式) 要素
fespace Vh(Th,P1);
Vh u,v;
// Poisson 方程式  $-\Delta u=f$  の右辺
func f = x*y;
// 現在時刻をメモ
real start = clock();
// 問題を解く
solve Poisson(u,v)
  = int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))-int2d(Th)(f*v)
  +on(Gamma,u=0);
// 可視化
plot(u,ps="poisson-disk.eps");
plot(u);
//
{
  ofstream ug("u-g.txt");
  for (int i=0; i<Th.nt; i++) {
    for (int j=0; j<3; j++) {
      ug << Th[i][j].x << " " << Th[i][j].y << " " << u[][Vh(i,j)]<<endl;
    }
    ug << Th[i][0].x << " " << Th[i][0].y << " " << u[][Vh(i,0)]<<"\n\n\n";
  }
}
// 計算時間を表示
cout << " CPU time= " << clock() - start << endl;
```

FreeFEM++ では、等高線描画やベクトル場の表示などは出来るが、グラフの鳥瞰図描画などはサポートしていないようである (最近は出来るようになってきているみたいなので、ここに書いてあることは「古い」かも知れないけれど、参考のため、しばらく残します)。そこで gnuplot を使ってグラフを描いてみる。

“u-g.txt” は次のようなデータである (3次元空間における三角形が並んでいる)。

⁶<http://www.freefem.org/ff++/index.htm>

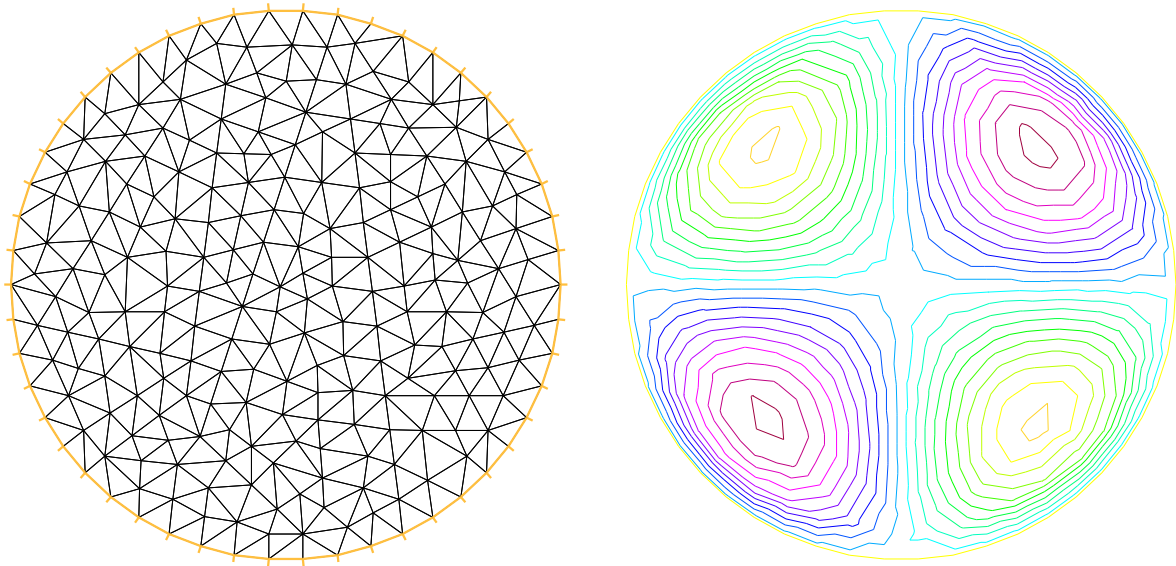


図 9: 円盤の三角形分割と Poisson 方程式の解の等高線表示 (FreeFEM++ による)

```
0.735953 0.529892 0.00584987
0.876307 0.481754 6.33978e-033
0.809017 0.587785 7.53444e-033
0.735953 0.529892 0.00584987
```

```
0.512882 0.46938 0.0104192
0.577982 0.366799 0.00923669
0.656968 0.448345 0.00912418
0.512882 0.46938 0.0104192
```

(中略)

.....

```
0.656968 0.448345 0.00912418
0.447782 0.737755 0.00678195
0.425779 0.904827 4.83201e-033
0.317582 0.777124 0.00599877
0.447782 0.737755 0.00678195
```

```
0.187381 0.982287 3.00659e-033
0.0627905 0.998027 8.0753e-034
0.149059 0.864343 0.00242012
0.187381 0.982287 3.00659e-033
```

このデータを gnuplot で可視化するには、例えば次のようにする (図 10)。

```
poisson-disk.g
set hidden3d
set palette rgbformulae 33,13,10
splot "u-g.txt" with lines palette
```

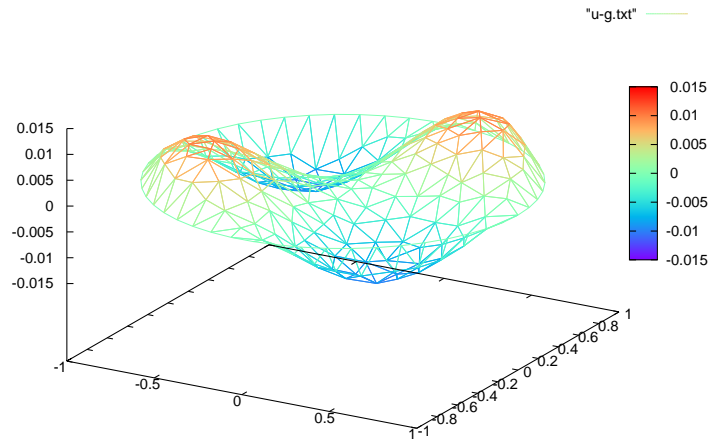


図 10: Poisson 方程式の解のグラフ表示 (gnuplot による)

5 コマンド・ファイルを実行する

4.1 の例のように長いコマンドは入力するのも大変で、修正するのも、後から再現するのも面倒である。このような場合は、コマンドをファイルに書いておいて、そのファイルを読み込んで中に書いてあるコマンドを実行させるとよい。

```
draw.g —
# draw.g -- gnuplot コマンド・ファイル
set term postscript eps color
set output "sincos.eps"
plot "table.data" using 1:2 with lines,\
    "table.data" using 1:3 with lines
set term x11
replot
```

gnuplot を起動してから draw.g を load して実行 —

```
oyabun% gnuplot
(ターミナルから gnuplot を起動する。)
gnuplot> load "draw.g"
```

あるいは、次のように gnuplot を起動するときの引数に指定して実行することもできる。

draw2.g を実行 —

```
oyabun% gnuplot draw2.g
```

この場合、gnuplot はコマンド・ファイル draw2.g の実行が終わると、すぐに終了してしまうので、画面で確認したい場合は、pause コマンドをはさんでおくとよい。

```
draw2.g
# draw2.g --- gnuplot 用のコマンド・ファイル
set term postscript eps color
set output "sincos.eps"
plot "table.data" using 1:2 with lines,\
      "table.data" using 1:3 with lines
set term x11
replot # 再描画
pause -1 "hit Enter key"
```

ここでは `pause -1 "hit Enter key"` とすることで、画面に “hit Enter key” と表示して、キーボードから Enter キーが入力されるまで待っている。

あるいは、`pause -1 "hit Enter key"` 無しでも、

```
oyabun% gnuplot draw2.g -
```

のように、`draw2.g` の後に、標準入力 `'-'` を指定しておけば、プロンプト `gnuplot>` が出て、入力待ちに出来る、というテクニックもある。

6 印刷

6.1 印刷用データの作成, L^AT_EX への取り込み

gnuplot は通常は画面に出力 (表示) するようになっているが、変数 `terminal` (`term` と略記可能, 印刷データのフォーマット), `output` (データの出力先⁷) を適当に設定することで印刷用のデータを作成できる。

gnuplot 起動直後は、UNIX では、`term` は `x11`, `output` は `STDOUT` になっている。Windows では `term` は `win` になっている。Mac では `term` は `aqua` になっている (のかな? 手元の某マシンでは `qt` になっていた…悩ましい)。

印刷用データの形式として、UNIX 環境では、多くの場合 **PostScript** が適当であろう (他に `tgif`, `eepic` などが考えられる)。

UNIX 環境での印刷用 PostScript データの生成

```
gnuplot> plot besj0(x),besj1(x)          ← Bessel 関数  $J_0, J_1$  のグラフを描く
gnuplot> show term                      ← terminal の確認 (通常は必要ない)
      terminal type is x11
gnuplot> show output                   ← 出力先の確認 (通常は必要ない)
      output is sent to STDOUT
gnuplot> set term postscript eps color  ← terminal の指定
gnuplot> set output "bessel.eps"       ← 出力ファイル名の指定
gnuplot> replot                        ← 直前の描画命令の再実行
gnuplot> set term x11                  ← 少なくとも term は元に戻しておく
gnuplot> set output                   ← output を STDOUT に戻す (あるいは unset output)
```

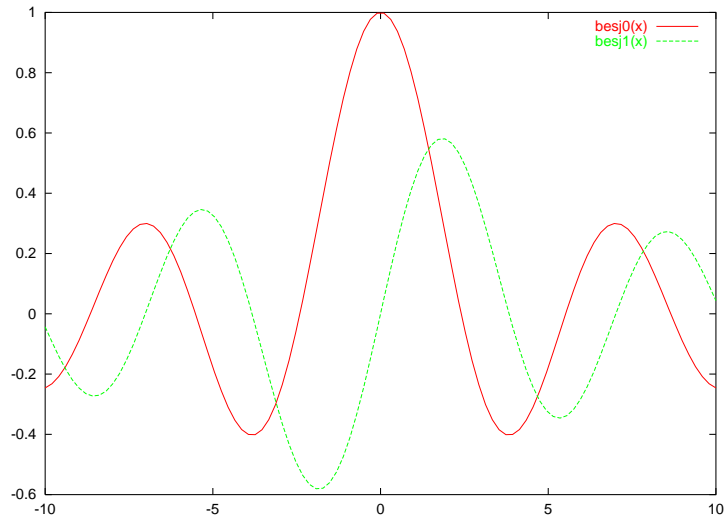
とすると、`bessel.eps` という名前の EPS 形式のファイルが出力できる。これは `lp コマ`

⁷通常はファイルであるが、UNIX 環境では `lp -dlp2` のようなパイプを指定することもできる。

ドで直接プリンターに送っても印刷できるし⁸、 \LaTeX 文書に次のようにして取り込むこともできる。

```
\usepackage[dvips]{graphicx}% プリアンプルには、いつもこう書いておく
...
\includegraphics[width=10cm]{bessel.eps}% bessel.eps を横幅 10cm で描く
```

Windows 環境でも大体同様だが、`term` は `win` に戻すこと、出力形式は PostScript ではなく、**PNG** などの画像形式が良いかもしれない⁹(`set term png` とするわけである)。あるいは、**Netpbm** が使えるならば、`pbm`, `pgm`, `ppm` などのドライバーを使うことも考えられる(`set terminal ppm` とか)。



(2017/8/26) この文書も化石のようになってしまったけれど(それでも仕立て直しをする時間はないなあ…)、Mac を使っているので少し補足。Mac の場合、画面に出力するには、`term` は `aqua` とすべき場合が多いであろう。こういうのを覚えるのが面倒ならば、

現在の状態を覚えておく

```
gnuplot> set term push
```

覚えておいた状態に戻す

```
gnuplot> set term pop
```

を活用すべきかもしれない (`push`, `pop` は知らないかなあ…)。例えばこの項の最初の例は次のように簡略化(かつ環境非依存に)出来る。

⁸例えば `lp -d プリンター名 bessel.eps` とすると印刷できる。また `ghostview, gv` のようなビューアーを用いて画面に表示することもできる。例えば `gv bessel.eps &` とする。

⁹以前だったら **GIF** を勧めるところだが、特許がらみの騒動で、**GIF** はフリーソフト作製者達から敬遠されつつあり、`gnuplot` もバージョンによっては **GIF** 出力をサポートしていない。現在では **GIF** の特許は切れているので、訴えられる心配はないが、**GIF** は廃れてしまった(ように思える)。

gnuplot> <u>plot besj0(x),besj1(x)</u>	← Bessel 関数のグラフを描く
gnuplot> <u>set term push</u>	← term の状態を覚えておく
gnuplot> <u>set term postscript eps color</u>	← terminal の指定
gnuplot> <u>set output "bessel.eps"</u>	← 出力ファイル名の指定
gnuplot> <u>replot</u>	← 直前の描画命令の再実行
gnuplot> <u>set term pop</u>	← term の状態を戻す。
gnuplot> <u>set output</u>	← output を STDOUT に戻す (省略可能)。

少なくとも term を元に戻しておけば (output を戻さなくても)、画面表示出来るようである。

(2024/11/8 追記) 最近は Mac で set term qt とか、set term wxt とかで使っている。

6.2 印刷に凝る

6.2.1 GNUPLOT+

(2017/8/26 加筆: これは流石に今は使わないですね。)

GNUPLOT で表示するグラフのラベルなどに T_EX 風の数式コマンドを使いたい、という場合には GNUPLOT+ の採用を考えて見よう。

(準備中 — とりあえず「GNUPLOT 日本語化・機能拡張 (PLUS-enhanced) パッチ 公式サイト」¹⁰ を見て下さい。最近更新されなくなって残念です…)

6.2.2 psfrag

2017/8/26 加筆: psfrag は、最近主流の環境では使えない場合が多いので、もう勧められなけれど、昔作ったものを最小限の修正で組版し直すための情報を紹介しておく:

<https://m-katsurada.sakura.ne.jp/knowhow-2016/node3.html>

これは gnuplot というよりも T_EX の話題ですが、gnuplot のユーザーと T_EX のユーザーはかなり重なっていると思うので…

(工事中)

図を作成するのに色々な方法がありますが、図中のテキスト (文章) の表現力については、T_EX のそれに匹敵するものは少ないのが現状です。例えば T_EX で書けるような数式を図中に入れるのは困難なことが多いです。psfrag は、PostScript データ中のテキストを任意の T_EX 表現で置き換える、という少々強引な手法でこの問題を解決するパッケージです。

パッケージですので、L^AT_EX ファイルのプリアンブルに

```
\usepackage{psfrag}
```

と書いておきます。includegraphics で図を差し込む直前に例えば

```
\psfrag{eps2}{$\varepsilon^2$}
```

とすると図中の “eps2” というテキストが ε^2 に置き換わります。

¹⁰<http://www.h2.dion.ne.jp/~yamaga/gnuplot/index.ja.html>

7 パラメーター曲線を描く

7.1 平面曲線

$x = f(t)$, $y = g(t)$ とパラメーター表示される平面曲線を描くには、set parametric としてから、plot f(t),g(t) とすればよい。このモードから抜けるには set noparametric とする。

- set size ratio 正数 でアスペクト比 (縦の長さ / 横の長さ) を指定できる。
- set size ratio 1 とする代わりに set size square とできる (描画領域が正方形になる)。
- 縦と横の縮尺をそろえたい場合は、set size ratio -1 とする。

特に図形の「形」が気になる場合は、set size ratio -1 は必須かも。例えば楕円

$$\begin{cases} x = 3 \cos t \\ y = 2 \sin t \end{cases} \quad (t \in [0, 2\pi])$$

やサイクロイド

$$\begin{cases} x = a(t - \sin t) \\ y = a(1 - \cos t) \end{cases} \quad (t \in \mathbf{R}).$$

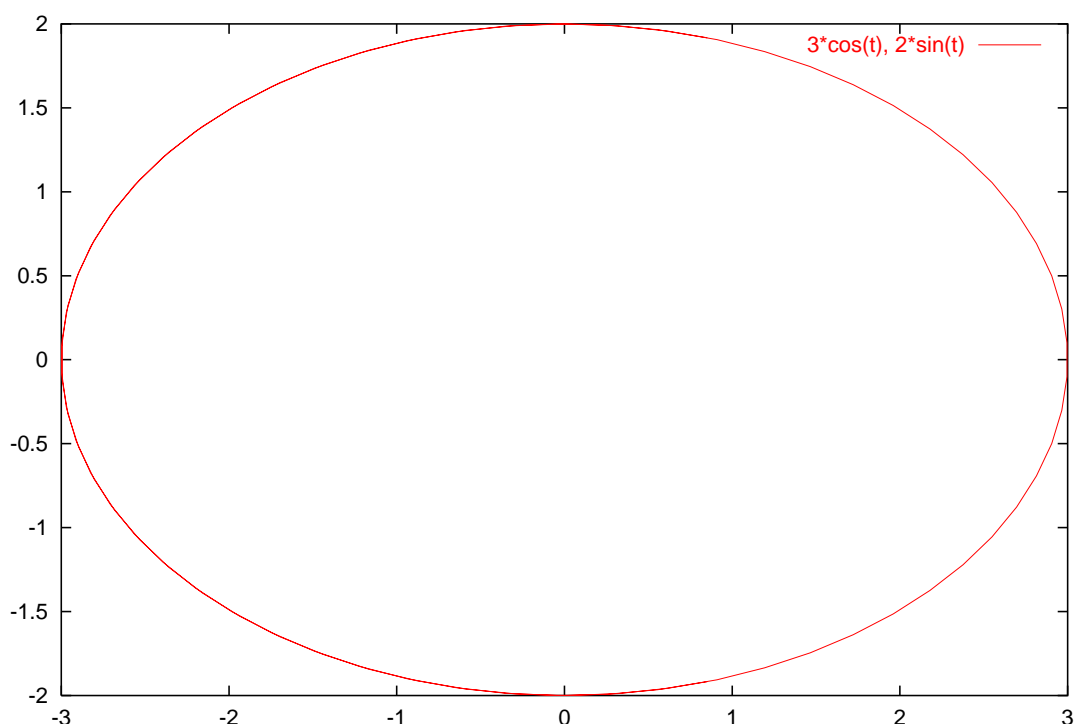


図 11: 楕円 `set parametric;set size ratio -1;plot 3*cos(t),2*sin(t)`

アルキメデス螺旋 $r = \theta$ を二つの方法で描いて比較してみよう。

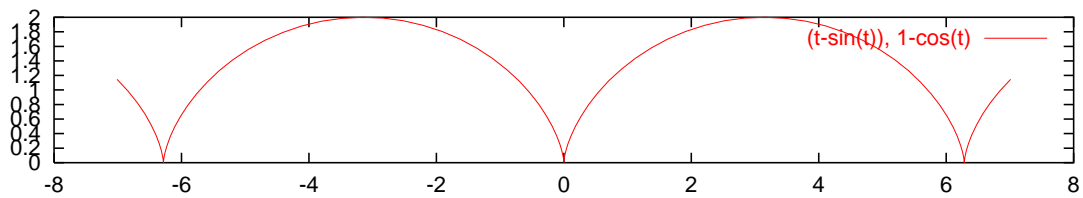


図 12: cycloid set parametric;set size ratio -1;plot [-8:8] t-sin(t),1-cos(t)

```
# archimedes1.gp
set parametric
set samples 200
set xrange [-30:30]
set yrange [-30:30]
set size square
plot [0:10*pi] t*cos(t),t*sin(t)
pause -1
set term postscript eps color
set output "archimedes1.eps"
replot
```

```
# archimedes2.gp
set parametric
set samples 200
set size ratio -1
plot [0:10*pi] t*cos(t),t*sin(t)
pause -1
set term postscript eps color
set output "archimedes2.eps"
replot
```

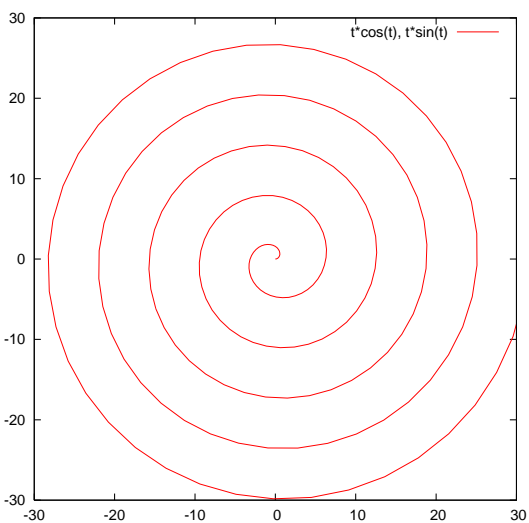


図 13: set size square で

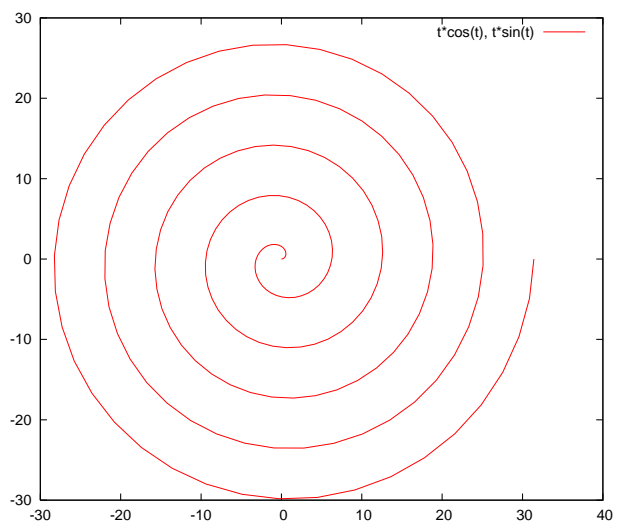


図 14: set size ratio -1 で

双曲線

```
# parametric-2.gp --- 双曲線
set parametric
set size ratio -1
a=3;b=2
plot [-2:2] a*cosh(t),b*sinh(t),-a*cosh(t),b*sinh(t)
pause -1
set term postscript eps color
set output "parametric-2.eps"
replot
```

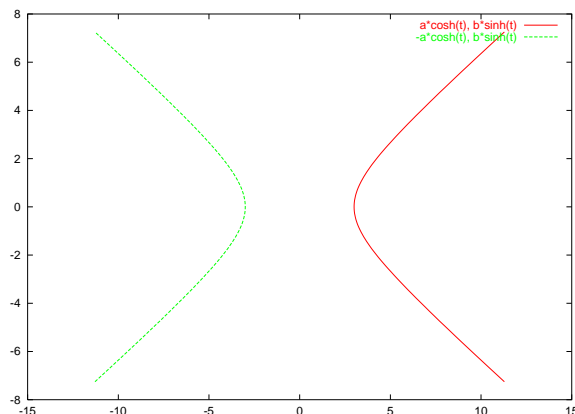


図 15: $x = \pm 3 \cosh t$, $y = 2 \sinh t$ ($t \in [-2, 2]$)

7.2 空間曲線

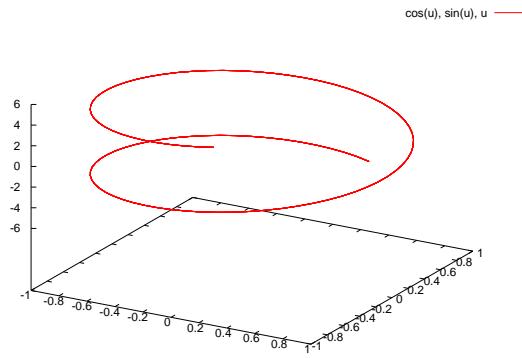
$x = f(u)$, $y = g(u)$, $z = h(u)$ とパラメータ表示される空間曲線を描くには、set parametric としてから、splot f(u),g(u),h(u) とする。

空間内の螺旋

```
# parametric-3.gp --- 空間内の螺旋
set parametric
splot cos(u),sin(u),u
pause -1 "Hit Enter key!"
set term postscript eps color
set output "parametric-3.eps"
replot
```

8 パラメータ曲面を描く

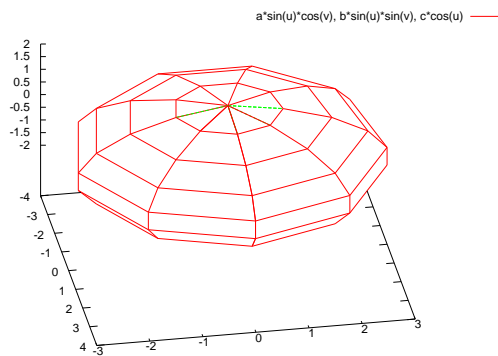
(準備中)



☒ 16: $x = \cos u, y = \sin u, z = u$

$$(x/a)^2 + (y/b)^2 + (z/c)^2 = 1$$

```
# parametric-4.gp ---
set parametric
a=4;b=3;c=2
set urange [0:pi]
set vrange [0:2*pi]
splot a*sin(u)*cos(v),b*sin(u)*sin(v),c*cos(u)
pause -1 "Hit Enter key!"
set term postscript eps color
set output "parametric-4.eps"
replot
```



☒ 17: $(x/a)^2 + (y/b)^2 + (z/c)^2 = 1$

$$x^2 + y^2 - z^2 = 1$$

```
# parametric-5.gp ---
set parametric
set urange [0:2]
set vrange [0:2*pi]
splot sinh(u)*cos(v),sinh(u)*sin(v),cosh(u),\
      sinh(u)*cos(v),sinh(u)*sin(v),-cosh(u)
pause -1 "Hit Enter key!"
set term postscript eps color
set output "parametric-5.eps"
replot
```

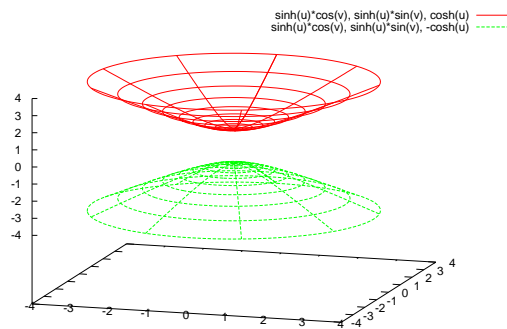


図 18: $x^2 + y^2 - z^2 = 1$

9 テクニック集

9.1 C から gnuplot を呼び出す

C プログラムから、`popen()` でパイプを使って `gnuplot` と通信するという手があります。

FILE 構造体へのポインター変数を用意しておいて、関数 `popen()` でパイプを開き、その後は `fprintf(gp,)` で、`gnuplot` へのコマンドを流し込むわけです。

```
FILE *gp;
...
gp = popen("gnuplot", "w");
```

9.1.1 関数を plot する

次のサンプル・プログラムは `plot sin(x)` させる、というものです。

```

testcallgnuplot0.c
/*
 * testcallgnuplot0.c
 */

#include <stdio.h>

int main(void)
{
    FILE *gp;
    char buf[BUFSIZ];

    gp = popen("gnuplot", "w");
    fprintf(gp, "plot sin(x)\n");
    fflush(gp);
    /* 行入力待つ --- Enter を打つまで待つ */
    fgets(buf, sizeof(buf), stdin);
    /* パイプを閉じる */
    pclose(gp);
    return 0;
}

```

fgets() を入れてあるのは、(図がすぐに消えてしまわないように) ユーザーからの入力を待つことにする、ということですが、X Windows System の環境では、gnuplot -persist と起動して、Cプログラムの終了後も gnuplot のウィンドウを残すことができます。

```

testcallgnuplot1.c
/*
 * testcallgnuplot1.c
 */

#include <stdio.h>

int main(void)
{
    FILE *gp;

    gp = popen("gnuplot -persist", "w");
    fprintf(gp, "plot sin(x)\n");
    pclose(gp);

    return 0;
}

```

9.1.2 計算データを plot する

計算データを元にある程度複雑なグラフを描くには、データファイルを作成して、それを plot するのが良いでしょうが(後述)、plot '-' オプションとして、データを標準入力から読むように指定することも出来ます。

```

testcallgnuplot.c
/*
 * testcallgnuplot.c
 */

#include <stdio.h>
#include <math.h>

int main(void)
{
    int i, n;
    double pi, x, dx;
    FILE *gp;
    char buf[BUFSIZ];

    pi = 4.0 * atan(1.0);
    gp = popen("gnuplot", "w");
    fprintf(gp, "plot '-' with linespoints\n");
    n = 100;
    dx = 2 * pi / n;
    for (i = 0; i <= n; i++) {
        x = i * dx;
        fprintf(gp, "%f %f\n", x, sin(x));
    }
    /* 行頭が 'e' だと標準入力からの読み込みを終了 */
    fprintf(gp, "e\n");
    fflush(gp);

    /* 行入力を待つ --- Enter を打つまで待つ */
    fgets(buf, sizeof(buf), stdin);
    pclose(gp);
    return 0;
}

```

線を途中で切るには改行だけの行(空行)を出力します。また最後に e だけの行を送ります。この場合、線の色は変わらない。

対話的に確認

```
gnuplot> plot '-' with lp
input data ('e' ends) > 1 1
input data ('e' ends) > 2 2
input data ('e' ends) > 3 3
input data ('e' ends) >
input data ('e' ends) > 1 4
input data ('e' ends) > 2 5
input data ('e' ends) > 3 6
input data ('e' ends) > e
```

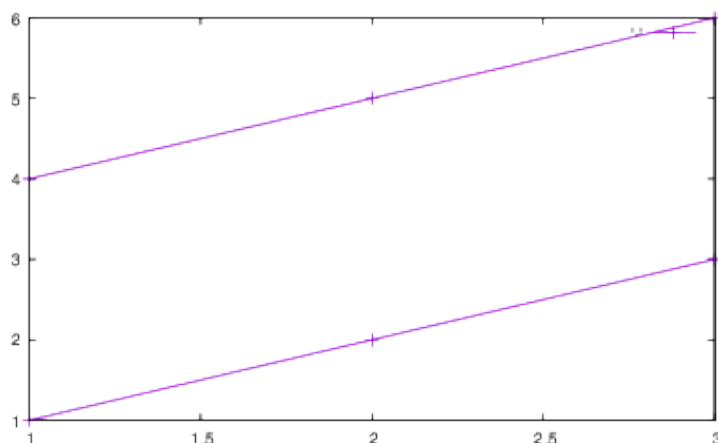


図 19: 1つの plot

次のように2つのものの plot とすると、線の色が変わります。

対話的に確認

```
gnuplot> plot '-' with lp, '-' with lp
input data ('e' ends) > 1 1
input data ('e' ends) > 2 2
input data ('e' ends) > 3 3
input data ('e' ends) > e
input data ('e' ends) > 1 4
input data ('e' ends) > 2 5
input data ('e' ends) > 3 6
input data ('e' ends) > e
```

例: 複数のグラフを plot する 外部ファイルと異なり、'-' は読み直しが出来ないので、データは一つ一つ出力する (対応する 'e' もグラフの個数分必要になる) というのがミソです。

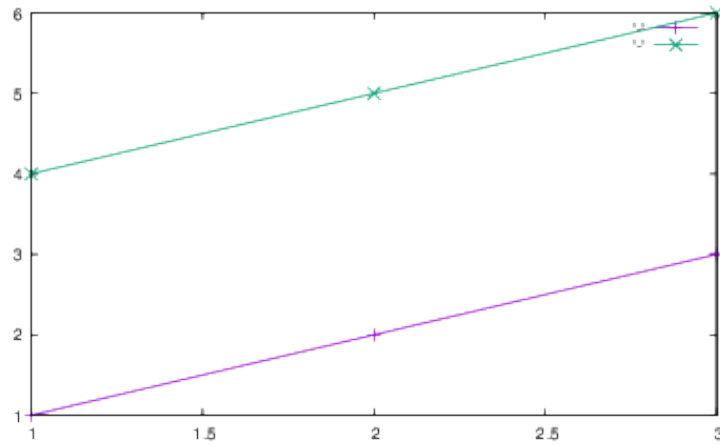


図 20: 2つの plot

3sukumi2.c

```

/*
 * 3sukumi2.c --- 3すくみ方程式の数値シミュレーション
 * http://phi.med.gunma-u.ac.jp/oldlec/ecology_p14.html の 3sukumi.R を
 * C 言語に焼き直し
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N (200)

void runif(double *v, int n, double a, double b)
{
    int i;
    for (i = 0; i < n; i++) v[i] = a + (b - a) * drand48();
}

int main(void)
{
    double N1[N+1], N2[N+1], N3[N], yr[N];
    double cc[3] = {0.01, 0.01, 0.01};
    double a[] = {0.2, 0.4, 0.1, 0.1, 0.2, 0.4, 0.4, 0.1, 0.2};
    double r[] = {0.5, 0.6, 0.7};
    double rand1[N], rand2[N], rand3[N];
    int i;
    FILE *gp;
    double hw;

    N1[0] = 1; N2[0] = 1; N3[0] = 1;
    /* yr = {1,2,...,N} */
    for (i = 0; i < N; i++)
        yr[i] = i + 1;
    /* drand48() の seed 設定 --- 適当 */
    srand48(1L);
    /* [-0.1,0.1) の一様乱数 N 個 */
    hw = 0.1;
    runif(rand1, N, -hw, hw); runif(rand2, N, -hw, hw); runif(rand3, N, -hw, hw);
    /* 微分方程式を Euler 法で解く */
    for (i = 0; i < N-1; i++) {
        N1[i+1] = cc[0] + exp(r[0]+rand1[i]-a[0]*N1[i]-a[1]*N2[i]-a[2]*N3[i])*N1[i];
        N2[i+1] = cc[1] + exp(r[1]+rand2[i]-a[3]*N1[i]-a[4]*N2[i]-a[5]*N3[i])*N2[i];
        N3[i+1] = cc[2] + exp(r[2]+rand3[i]-a[6]*N1[i]-a[7]*N2[i]-a[8]*N3[i])*N3[i];
    }
    /* gnuplot を呼び出して描画 */
    gp = popen("gnuplot -persist", "w");
    fprintf(gp, "plot '-' with lines, '-' with lines, '-' with lines\n");
    for (i = 0; i < N; i++) fprintf(gp, "%f %f\n", yr[i], N1[i]); fprintf(gp, "e\n");
    for (i = 0; i < N; i++) fprintf(gp, "%f %f\n", yr[i], N2[i]); fprintf(gp, "e\n");
}

```

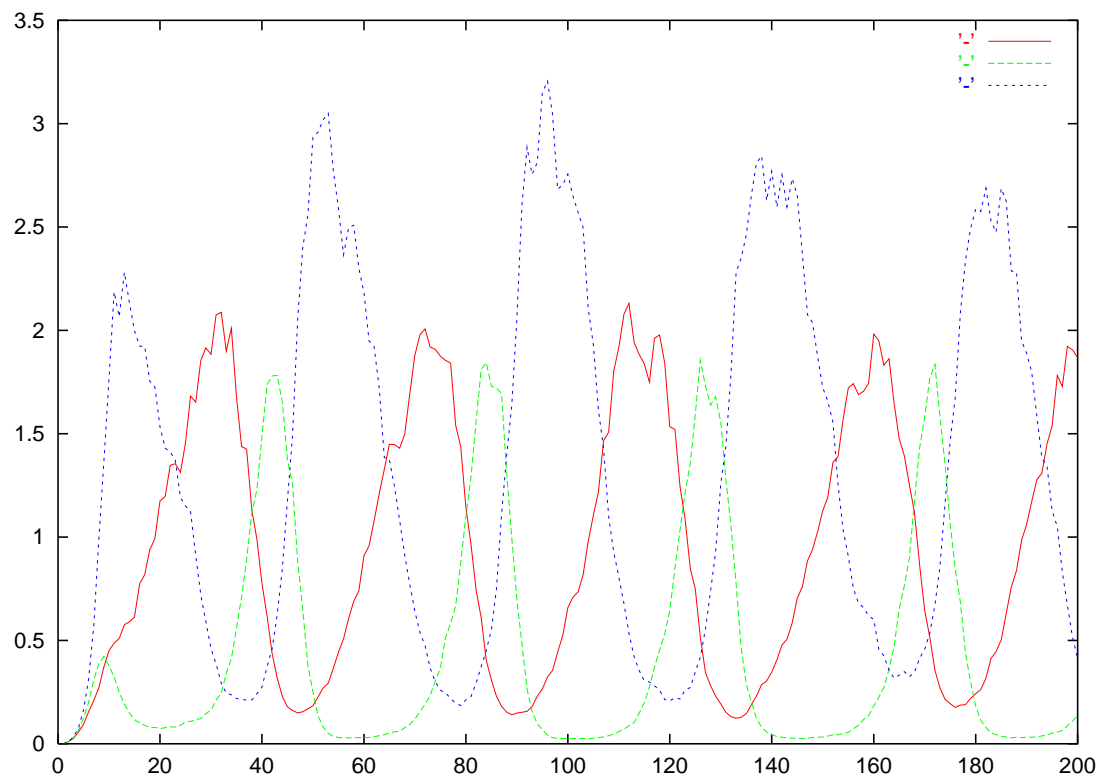


図 21: 3sukumi2.c の実行結果

(2021/4/28 追記) 上の 3sukumi.c は元ネタが自分で書いたものでなくて、どうも良く分からないところがあるので、別のサンプル・プログラムを用意した。

```
lotka-volterra-rk2.c
```

```
/*
 * lotka-volterra-rk2.c (Runge-Kutta method for Lotka-Volterra model)
 * ターミナルで
 *   cc lotka-volterra-rk2.c
 *   ./a.out > lotka-volterra-rk2.data
 *   500
 *   gnuplot
 * gnuplot> に対して
 *   plot "lotka-volterra-rk2.data" using 2:3 with l
 */

#include <stdio.h>

double a,b,c,d;

int main(void)
{
    int i, N;
    double t, x, y, dt, k1x, k1y, k2x, k2y, k3x, k3y, k4x, k4y;
    double fx(double, double, double), fy(double, double, double), x0, y0;
    double Tmax;
    // パラメーター決定
    a = 2; b = 1; c = 3; d = 1;
    // 最終時刻
    Tmax = 10.0;
    // 時間刻み
    printf("# N: "); scanf("%d", &N);
    dt = Tmax / N;
    // 初期値
    x0 = 3.0;
    for (y0 = 0.5; y0 <= 5.0; y0 += 0.5) {
        // 初期値
        t = 0.0;
        x = x0;
        y = y0;
        printf("# t x y\n");
        printf("%f %f %f\n", t, x, y);
        // Euler 法
        for (i = 0; i < N; i++) {
            k1x = dt * fx(x, y, t);
            k1y = dt * fy(x, y, t);
            k2x = dt * fx(x + k1x / 2, y + k1y / 2, t + dt / 2);
            k2y = dt * fy(x + k1x / 2, y + k1y / 2, t + dt / 2);
            k3x = dt * fx(x + k2x / 2, y + k2y / 2, t + dt / 2);
            k3y = dt * fy(x + k2x / 2, y + k2y / 2, t + dt / 2);
            k4x = dt * fx(x + k3x, y + k3y, t + dt);
            k4y = dt * fy(x + k3x, y + k3y, t + dt);
            x += (k1x + 2 * k2x + 2 * k3x + k4x) / 6;
            y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6;
        }
    }
}
```

コンパイル、実行、可視化

```
% cc lotka-volterra-rk2.c
% ./a.out > lotka-volterra-rk2.data
500
% gnuplot
gnuplot> plot "lotka-volterra-rk2.data" using 2:3 with l
```

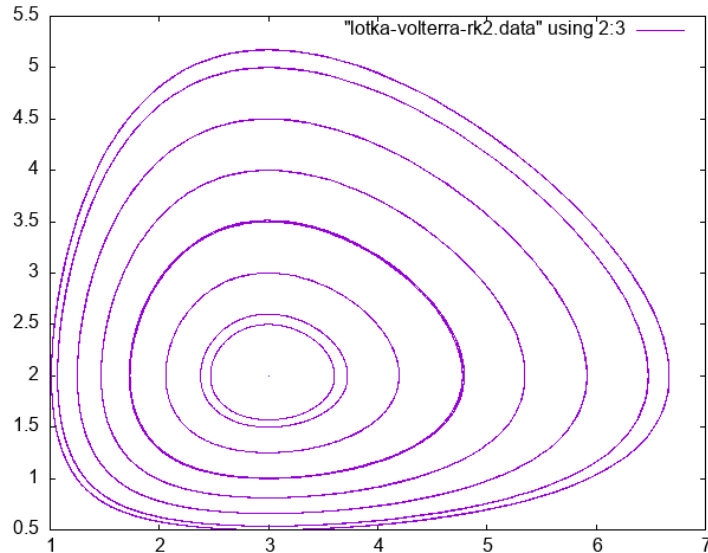


図 22: Lotka-Volterra の方程式の解軌道

9.1.3 計算データをファイルに出力して plot する

データファイルを作成して、それを plot で描画する例として、

<https://m-katsurada.sakura.ne.jp/program/>

にサンプル・プログラム

- https://m-katsurada.sakura.ne.jp/program/graphics/call_gnuplot.c
- https://m-katsurada.sakura.ne.jp/program/graphics/call_gnuplot.h
- <https://m-katsurada.sakura.ne.jp/program/graphics/polar4.c>

を置いておきます (内容は円盤で定義された関数の可視化をする)。これは記憶が確かではないですが、元ネタは GNUPLOT FAQ だったと思います。

なお、“GPTCALL” というものが

<http://phe.phyas.aichi-edu.ac.jp/~cyamauch/gptcall.html>

で公開されています。ちょっと試してみたらなかなか良さそうでした。

9.2 等高線のみを描く

GNUPLOT — not so frequently asked questions —¹¹ にある文書 <http://t16web.lanl.gov/Kawano/gnuplot/plot3d.html#6.5> に載っていた話。

視点を 0,0 にして (真上からのぞくことにして)、曲面自身を描かないようにして (set nosurface あるいは unset surface)、表示するというのが要点。

```
drawcontour.g
# drawcontour.g
set view 0,0
set contour base
set nosurface
set isosamples 100
set cntrparam level 10
splot x**2-y**2
pause -1
set term postscript eps color; set output "contour.eps"; replot
```

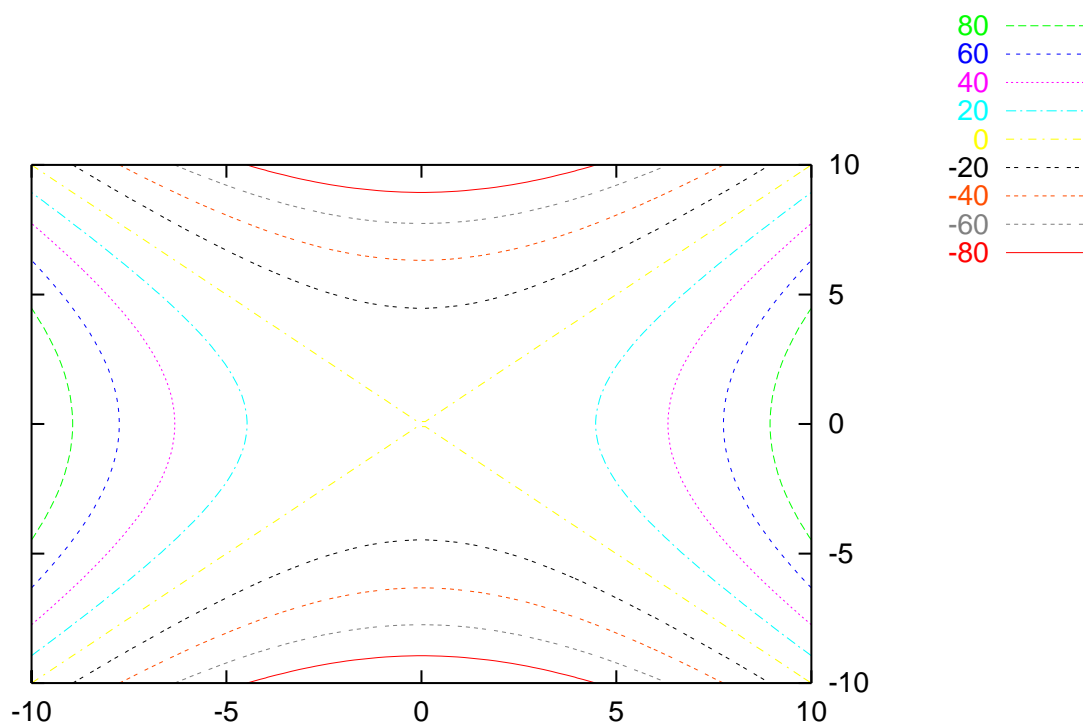


図 23: $x^2 - y^2$ の等高線

描画する等高線の高さ (レベル) は、set cntrparam levels incre -80,10,80 のようにこちらで指定することもできます (-80 から 10 刻みで 80 まで)。

9.3 細かい技 (凡例、ラベル等)

- いわゆる凡例¹² (legend) を消すには、unset key とする。

¹¹<http://t16web.lanl.gov/Kawano/gnuplot/>

¹²地図や図表などで使用する記号の説明を「凡例」と呼ぶ。

- 個々の凡例は `title` で指定できる。 `plot sin(x) title "graph of sin"`
- 最近の `gnuplot` では、 `plot sin(x) notitle` のようなことも出来るらしい。
- x 軸, y 軸のラベルは、それぞれ `set xlabel "文字列"` と `set ylabel "文字列"`
- 望む位置にラベルを表示できる。 `set label [タグ番号] "文字列" [位置指定]` という形式。 `[]` は省略可能であることを示す。タグ番号を省略すると、自動的に番号がふられる。位置指定を省略すると、ど真ん中にラベルが表示されて(多分)困る。例えば

```
set label "文字列" at screen 0.5,screen 1
```

のように位置を指定することになるだろう。ラベルはどんどん増えていくので、クリアするには `unset label [タグ番号]` とする。

first	座標系の原点が (0,0)
second	右上の点
graph	枠の左下が (0,0) で右上が (1,1)
screen	ウィンドウの左下が (0,0) で右上が (1,1)
(無指定)	そのグラフで採用している座標系

- `set arrow from 0,-1 to 0,1; set arrow from -10,0 to 10,0` のようにして矢印を描いて座標軸がわりにできる。
- C 言語風の `sprintf()` が利用可能である。例えば `s=sprintf("pi=%18.15f, e=%18.15f", pi, exp(1))` とすると、“pi= 3.141592653589793, e= 2.718281828459045” という文字列が出来る。
- フォントを指定すれば日本語も利用可能である。フォント名は環境依存するが、頑張って調べる価値はある。以下は私の Mac で使えるフォントを利用した例。

```
set title "標準正規分布の確率密度関数" font "HiraMinPro-W3, 24"
```

(HiraMinPro-W3 はどこで設定しておいたのか分からない…)

9.4 簡易アニメーション

スクリプトを `reread` することによって、簡単なアニメーションが実現出来ます。偏微分方程式論で有名な熱方程式の基本解

$$U(x,t) = \frac{1}{\sqrt{4\pi t}} \exp\left(-\frac{x^2}{4t}\right)$$

は、時刻 $t=0$ で数直線上の原点に単位熱量をおいて、それ以降は熱伝導によって熱が流れて行った場合の、時刻 t , 場所 x における温度を表します。

```
anim.gp
plot [-10:10] [0:1] u(x,t)
t=t+dt
if (t<Tmax) reread
```

のようなファイル “anim.gp” を用意しておいて、`gnuplot` で

```
gnuplot> u(x,t)=exp(-x*x/(4*t))/sqrt(4*pi*t)
gnuplot> t=0.1
gnuplot> Tmax=5
gnuplot> dt=0.01
gnuplot> load "anim.gp"
```

とすると $t = 0.1$ から 0.01 刻みで $t = 5$ まで、 $u(\cdot, t)$ のグラフが描けます。
ここでは静止面で我慢。

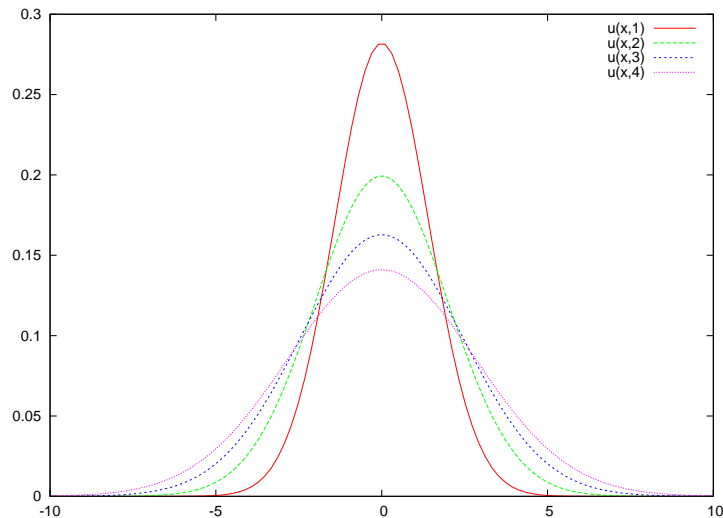


図 24: $t = 1, 2, 3, 4$ での $u(\cdot, t)$ のグラフ

(工夫すれば GIF アニメーションとか作れそうですね。… `set term gif animate` として `set output "なんか.gif"` とするだけでした。)

<https://m-katsurada.sakura.ne.jp/labo/text/heat-kernel.gif>

(2017/11/7 加筆) gnuplot ver 4.6 から、`do for` という新しい構文が導入されて、それを使うと、次のように書ける。

```
Tmax=5.0
Nmax=500
dt=Tmax/Nmax
u(x,t)=exp(-x*x/(4*t))/sqrt(4*pi*t)
do for [i=1:Nmax] {
    t=i*dt
    plot [-10:10] [0:1] u(x,t)
}
```

(最初 $T_{\max}=5$ としたら、 dt が 0 になってハマってしまった。)

練習問題 (偏微分方程式を勉強する人向け) Poisson 核

$$P_r(\theta) = \frac{1}{2\pi} \frac{1-r^2}{1-2r\cos\theta+r^2} \quad (0 \leq r < 1, \theta \in \mathbf{R})$$

がどんなものか表示せよ ($r = 0, 0.1, 0.2, \dots$ に対して、 P_r のグラフを $[-\pi, \pi]$ で描く)。

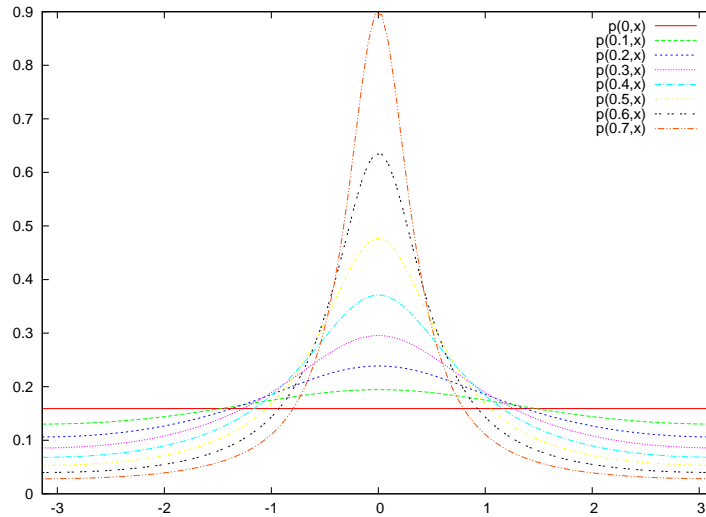


図 25: $\theta \in [-\pi, \pi]$ の範囲で $P(0.1, \cdot), P(0.2, \cdot), \dots, P(0.7, \cdot)$ のグラフを描く

9.5 misc

- 例えば 2, 3 次元程度の微分方程式ならば、Runge-Kutta 法によって微分方程式の数値解を求めて、可視化する程度のことは gnuplot だけで出来てしまうようになっている。
「gnuplot で微分方程式を解く」¹³ を見よ。

A 使ったファイルの置場所

1. <https://m-katsurada.sakura.ne.jp/lab0/howto/intro-gnuplot/table.data>
2. <https://m-katsurada.sakura.ne.jp/lab0/howto/intro-gnuplot/error.tbl>
3. <https://m-katsurada.sakura.ne.jp/lab0/howto/intro-gnuplot/draw.g>
4. <https://m-katsurada.sakura.ne.jp/lab0/howto/intro-gnuplot/draw2.g>

B 実例集

B.1 例: 差分法で解いた 1 次元熱伝導方程式の初期値境界値問題の解の可視化

1 次元区間 $(0, 1) = \{x \in \mathbf{R} \mid 0 < x < 1\}$ における熱伝導方程式の初期値境界値問題

$$\begin{aligned} \frac{\partial u}{\partial t}(x, t) &= \frac{\partial^2 u}{\partial x^2}(x, t) \quad (x \in (0, 1), t \in (0, T_{\max})), \\ u(0, t) &= u(1, t) = 0 \quad (t \in (0, T_{\max})), \\ u(x, 0) &= u_0(x) \quad (x \in [0, 1]). \end{aligned}$$

を差分法 (陽解法) で解いて、それを可視化する。

ここで u_0 は時刻 $t = 0$ での初期温度分布である。

時間発展する系なので、各時刻 $t = t_n$ での差分解 $x \mapsto u(x, t_n)$ のグラフを描くことにする。差分法の説明は、例えば「発展系の数値解析」¹⁴ に載っている。

¹³http://www.ss.scphys.kyoto-u.ac.jp/person/yonezawa/contents/program/gnuplot/diff_eq.html

¹⁴<https://m-katsurada.sakura.ne.jp/lab0/text/heat-fdm-0.pdf>

```

/*
 * heat1d-gnuplot.c --- 1次元領域の熱伝導方程式
 *   u_t=u_{xx}, 同次Dirichlet境界条件
 */

#include <stdio.h>
#include <math.h> // floor()
#include <stdlib.h> // malloc()
#include <unistd.h> // sleep()

// 初期条件
double u0(double x)
{
    if (x < 0.5)
        return x;
    else
        return 1.0 - x;
}

int main(void)
{
    int N, i, n, nMax, nskip;
    double Tmax, tau, h, t, lambda, c, dt;
    double *x, *u, *newu;
    FILE *gp;
    Tmax = 0.5; // 最終時刻
    dt = 0.01; // 描画間隔
    // 分割
    N = 50;
    h = 1.0 / N;
    lambda = 0.5; //  $\lambda = \tau / h^2$ 
    tau = lambda * h * h;
    nskip = rint(dt / tau); if (nskip == 0) nskip = 1;
    nMax = Tmax / tau;
    printf("N=%d, h=%g, tau=%g, nskip=%d\n", N, h, tau, nskip);
    // メモリ確保
    x = malloc(sizeof(double) * (N+1));
    u = malloc(sizeof(double) * (N+1));
    newu = malloc(sizeof(double) * (N+1));
    if (x == NULL || u == NULL || newu == NULL) {
        fprintf(stderr, "メモリ確保に失敗しました。 \n");
        exit(1);
    }
    // gnuplot を呼び出す
    if ((gp = popen("gnuplot -persist", "w")) == NULL) {
        fprintf(stderr, "gnuplot 呼び出しに失敗しました。 \n");
        exit(1);
    }
    // fprintf(gp, "set term gif animate\n");
    // fprintf(gp, "set output \"heat1d.gif\"\n");
    fprintf(gp, "set yrange [-0.01:0.5]\n");
    fprintf(gp, "set xlabel \"x\"\n");
    fprintf(gp, "set ylabel \"u\"\n");
    fprintf(gp, "set label \"1D heat equation\" at screen 0.4,0.95\n");
    // 初期値
    t = 0;
    for (i = 0; i <= N; i++) {
        x[i] = i * h;
        u[i] = u0(x[i]);
    }
    // 初期値を描く
    fprintf(gp, "plot '-' with lines title \"t=0\"\n");
    for (i = 0; i <= N; i++) {

```

```

    fprintf(gp, "%f %f\n", x[i], u[i]);
}
fprintf(gp, "e\n"); fflush(gp);
sleep(3);
// 時間を進める
c = 1 - 2.0 * lambda;
for (n = 1; n <= nMax; n++) {
    t = n * tau;
    for (i = 1; i < N; i++)
        newu[i] = c * u[i] + lambda * (u[i+1]+u[i-1]);
    newu[0] = newu[N] = 0.0;
    for (i = 0; i <= N; i++)
        u[i] = newu[i];
    if (n % nskip == 0) {
        fprintf(gp, "plot '-' with lines title \"t=%g\"\n", t);
        for (i = 0; i <= N; i++)
            fprintf(gp, "%f %f\n", x[i], u[i]);
        fprintf(gp, "e\n"); fflush(gp);
    }
}
}
pclose(gp);
return 0;
}

```

Mac のターミナルの場合: プログラムの入手、コンパイル、実行

```

curl -Ohttps://m-katsurada.sakura.ne.jp/misc/20191229/heat1d-gnuplot.c
cc heat1vi d-gnuplot.c
./a.out

```

(Xcode の cc, GNU の gcc で動作確認済み)

GIF アニメーション¹⁵

B.2 例: 差分法で解いた 2 次元熱伝導方程式の初期値境界値問題の解の可視化

(有限要素法を出したので、差分法もやっておくべきか、どうして気づかなかったのか?)
正方形領域

$$\Omega = \{(x, y) \in \mathbf{R}^2 \mid 0 < x < 1, 0 < y < 1\}$$

における熱伝導方程式の初期値境界値問題

$$\begin{aligned} \frac{\partial u}{\partial t}(x, y, t) &= \frac{\partial^2 u}{\partial x^2}(x, y, t) + \frac{\partial^2 u}{\partial y^2}(x, y, t) \quad ((x, y) \in \Omega, t \in (0, T_{\max})), \\ u(x, y, t) &= 0 \quad ((x, y) \in \partial\Omega, t \in (0, T_{\max})), \\ u(x, y, 0) &= u_0(x, y) \quad ((x, y) \in \bar{\Omega}). \end{aligned}$$

を差分法 (陽解法) で解いて、それを可視化する。

ここで $\partial\Omega$ は Ω の境界 ((正方形の 4 つの辺の合併)、 u_0 は時刻 $t = 0$ での初期温度分布である。

時間発展する系なので、各時刻 $t = t_n$ での差分解 $(x, y) \mapsto u(x, y, t_n)$ のグラフの鳥瞰図を描くことにする。

¹⁵<https://m-katsurada.sakura.ne.jp/misc/20191229/heat1d.gif>

プログラムは<https://m-katsurada.sakura.ne.jp/misc/20191229/heat2d-gnuplot.c> に置いておく。

Mac のターミナルの場合: プログラムの入手、コンパイル、実行

```
curl -O https://m-katsurada.sakura.ne.jp/misc/20191229/heat2d-gnuplot.c
cc heat2d-gnuplot.c
./a.out
```

(Xcode の cc, GNU の gcc で動作確認済み)

```
/*
 * heat2d-gnuplot.c --- 2次元領域の熱伝導方程式
 *   u_t=u_{xx}+u_{yy}, 同次 Dirichlet 境界条件
 */

#include <stdio.h>
#include <math.h> // rint()
#include <stdlib.h> // malloc()
#include <unistd.h> // sleep()

typedef double *vector;
typedef vector *matrix;

matrix newmatrix(int m, int n)
{
    int i;
    vector ap;
    matrix a;

    if ((a = malloc(m * sizeof(vector))) == NULL)
        return NULL;
    if ((ap = malloc(m * n * sizeof(double))) == NULL) {
        free(a);
        return NULL;
    }
    for (i = 0; i < m; i++)
        a[i] = ap + (i * n);
    return a;
}

// 初期条件
double u0(double x, double y)
{
    return x * (1 - x) * y * (1 - y);
}

int main(void)
{
    int nx, ny, i, j, n, nMax, nskip;
    double Tmax, tau, hx, hy, t, lambdax, lambday, lambda, c, dt, ztop;
    double *x, *y;
    matrix u, newu;
    FILE *gp;
    Tmax = 0.5;
    dt = 0.001; // 描画間隔
    // 分割
    nx = 50; ny = 50;
    hx = 1.0 / nx; hy = 1.0 / ny;
    tau = 0.25 * (hx*hx*hy*hy)/(hx*hx+hy*hy); // λ=0.25
    nskip = rint(dt / tau); if (nskip <= 0) nskip = 1;
    lambdax = tau / (hx * hx);
```

```

lambday = tau / (hy * hy);
c = 1 - 2.0 * (lambdax + lambday);
nMax = Tmax / tau;
printf("hx=%g, hy=%g, tau=%g, nskip=%d\n", hx, hy, tau, nskip);
// メモリ確保
x = malloc(sizeof(double) * (nx+1));
y = malloc(sizeof(double) * (ny+1));
u = newmatrix(nx+1,ny+1);
newu = newmatrix(nx+1,ny+1);
if (x == NULL || y == NULL || u == NULL || newu == NULL) {
    fprintf(stderr, "メモリ不足です。");
    exit(1);
}
// gnuplot を呼び出す
if ((gp = popen("gnuplot -persist", "w")) == NULL) {
    exit(1);
}
// 次の2行の注釈を外すとアニメーション GIF ファイルを作る
// fprintf(gp, "set term gif animate\n");
// fprintf(gp, "set output \"heat2d.gif\"\n");
fprintf(gp, "set xlabel \"x\"\n");
fprintf(gp, "set ylabel \"y\"\n");
fprintf(gp, "set label \"2D heat equation\" at screen 0.4,0.95\n");
fprintf(gp, "set xrange [0:1]\n");
fprintf(gp, "set yrange [0:1]\n");
// 初期値
t = 0;
for (i = 0; i <= nx; i++)
    x[i] = i * hx;
for (j = 0; j <= ny; j++)
    y[j] = j * hy;
for (i = 0; i <= nx; i++)
    for (j = 0; j <= ny; j++)
        u[i][j] = u0(x[i], y[j]);
// 初期値を描く
ztop = pow(10.0, ceil(log10(fabs(u[nx/2][ny/2]))));
fprintf(gp, "set zrange [0:%f]\n", ztop);
fprintf(gp, "splot '-' with lines title \"t=0\"\n");
for (i = 0; i <= nx; i++) {
    for (j = 0; j <= ny; j++)
        fprintf(gp, "%f %f %f\n", x[i], y[j], u[i][j]);
    fprintf(gp, "\n");
}
fprintf(gp, "e\n");
fflush(gp);
sleep(3);
// 時間を進める
for (n = 1; n <= nMax; n++) {
    t = n * tau;
    for (i = 1; i < nx; i++)
        for (j = 1; j < ny; j++) {
            newu[i][j] = c * u[i][j]
                + lambdax * (u[i+1][j]+u[i-1][j])
                + lambday * (u[i][j+1]+u[i][j-1]);
        }
    for (i = 0; i <= nx; i++)
        newu[i][0] = newu[i][ny] = 0.0;
    for (j = 0; j <= ny; j++)
        newu[0][j] = newu[nx][j] = 0.0;
    for (i = 0; i <= nx; i++)
        for (j = 0; j <= ny; j++)
            u[i][j] = newu[i][j];
    if (n % nskip == 0) {

```

```

ztop = pow(10.0, ceil(log10(fabs(u[nx/2][ny/2]))));
fprintf(gp, "set zrange [0:%f]\n", ztop);
fprintf(gp, "splot '-' with lines title \"t=%g\"\n", t);
for (i = 0; i <= nx; i++) {
    for (j = 0; j <= ny; j++)
        fprintf(gp, "%f %f %f\n", x[i], y[j], u[i][j]);
    fprintf(gp, "\n");
}
fprintf(gp, "e\n");
fflush(gp);
}
}
pclose(gp);
return 0;
}

```

GIF アニメーション¹⁶

C Windows 環境での利用

(基本的に仕事は UNIX 上で行っているのですが、Windows 上の gnuplot には詳しくないが…)

以前は gnuplot+ のバイナリーを GNUPLOT+ の “やま・か” 氏のページから入手していた。今だと <http://www.ipc.chiba-u.ac.jp/~yamaga/wgnuplot/wgpl+w32.zip> になるのかな。

例えば、角藤氏のページ¹⁷ から入手した gnuplot-40p0w32.zip ファイルを展開して、現れたフォルダー中の wgnuplot.exe にパスを通すなどして実行すればよい。ただしフォントが適当でない場合がある。このときは gnuplot のウィンドウを右クリックして、フォントを変更してみる (MS ゴシックなどが無難?)。気に入ったものを見つけたら、やはり右クリックして現れるメニューの最後にある Update wgnuplot.ini を実行する。

D 参考となる情報

- <http://ayapin.film.s.dendai.ac.jp/~matuda/Gnuplot/gnuplot.html>
『グラフは Gnuplot にお任せ』
2004 年 4 月現在、日本語のページとしては、一番便利なリンク集。
- <http://t16web.lanl.gov/Kawano/gnuplot/>
『GNU PLOT — not so Frequently Asked Questions —』
- <http://takeno.iee.niit.ac.jp/~foo/gp-jman/gp-jman.html>
『gnuplot のページ (Takeno Labo)』 — 4.00 の日本語マニュアル, Windows 版 gnuplot 日本語化キットなど。
- http://www.geocities.jp/koh_hotta/ghost/#gnuplot
『Appendix 2: 初歩 gnuplot 入門』 by 堀田耕作 (愛知教育大学) 講義の記録だそうですが、曲面描画の例が豊富で参考になります。
- <http://www.linux.or.jp/JF/JFdocs/gnuplot.html>
『GNU PLOT 3.5 マニュアル』 (written by Thomas Williams, Colin Kelley, 翻訳: 田丸博晴, 升谷 保博)
<https://m-katsurada.sakura.ne.jp/howto/gnuplot35j.pdf>

¹⁶<https://m-katsurada.sakura.ne.jp/misc/20191229/heat2d.gif>

¹⁷<http://www.fsci.fuk.kindai.ac.jp/~kakuto/win32-ptex/web2c75.html>

- <https://m-katsurada.sakura.ne.jp/labo/howto/gnuplotj/>
古い日本語マニュアル (以前 fj に流れていたものを HTML に変換しました。)
- <http://www.h2.dion.ne.jp/~yamaga/gnuplot/index.ja.html>
『GNU PLOT 日本語化・機能拡張 (PLUS-enhanced) パッチ 公式サイト』いわゆる **GNU-PLOT+**
- 矢吹道郎監修、大竹つよし著, 使いこなす GNU PLOT, テクノプレス (1996).
- 川原稔, gnuplot パーフェクトマニュアル, ソフトバンク (1999).
- gnuplot に付属するデモ
gnuplot のソースファイル一式の中に demo というディレクトリがあります。そこで gnuplot all.dem とすると、一通り見ることができるはず (昔は確かに出来たはずですが、今やってみたら、一部エラーになりました)。
- Win32 環境用の $\text{T}_{\text{E}}\text{X}$ と言えば、角藤さんが有名ですが、<http://www.fsci.fuk.kindai.ac.jp/~kakuto/win32-ptex/web2c75.html> には gnuplot の Win32 用のバイナリーもおいてあります。