明大数学科計算機室ユーザーのための GLSC の紹介 Version 6.3

桂田 祐史

2009 年 5 月 5 日, 2013 年 12 月, 平成 27 年 4 月 18 日

この文書の最新版は

http://nalab.mind.meiji.ac.jp/~mk/labo/howto/intro-glsc/

で読めます。なお

http://nalab.mind.meiji.ac.jp/~mk/labo/howto/index.html#GLSC

も適宜参照してください。

この文書を最初に書きだしたのは遥か大昔で、「明大数学科計算機室」なんて取るべきだと思うのだけど、まだそういう事情が残っているので、この文書を整理する時間が取れるまでは、このタイトルのままにしておきます。

目 次

1　はじめに — GLSC とは

GLSC (Graphic Library for Scientific Computing) とは、龍谷大学数理情報学科のグループ[1]が作成した、科学技術計算の結果を可視化するためのライブラリィです[2]。

ワークステーション環境 (具体的には UNIX + X) で、C や Fortran で図を描く手段には色々なものがありますが、GLSC は実際に数値解析の分野で仕事をしてきた人達が作ったものだけに、数値シミュレーション結果の可視化には本当に便利なものだと筆者は思っています。具体的には、2 変数関数の可視化 (等高線、鳥瞰図(ちょうかんず)の描画) のためのサブルーチン、関数が揃っています。

マニュアル (PostScript フォーマット) も公開されていて、それを PDF に変換したものを

http://nalab.mind.meiji.ac.jp/~mk/labo/howto/GLSC.pdf

においてあります (機械的に PDF に変換しただけで、何も手を入れていませんが、ないよりはましでしょう)。

glscwin — Windows 版 GLSC

オリジナルの開発者の一人である高橋大輔氏の研究室によって、Win32 環境[3]に移植された **glscwin** については、付録の E を見て下さい。

2　身の回りの環境での使い方

数学科計算機室 (6701 号室) にある Solaris 2.6 で動いているワークステーションや、Linux マシン、さらには桂田研のノートパソコン (Cygwin であったり、Knoppix であったり) には、GLSC がインストールされています。

(ソース・ファイルからのインストールの仕方を付録にでも書いておきたい…)

[1] 小林亮、高橋大輔、中野浩、松本平淳太。

[2] ftp://ftp.st.ryukoku.ac.jp/pub/ryukoku/software/math/ から ftp で入手できます。

[3] 具体的な OS の名前で言うと、Windows 95, Windows 98, Windows Me, Windows NT, Windows 2000 のことを指します。

具体的には
1. インクルード・ファイル　/usr/local/include/glsc.h
2. インクルード・ファイル　/usr/local/include/glsc_ftn.h
3. ライブラリィ・アーカイブ　/usr/local/lib/libglscd.a
4. ライブラリィ・アーカイブ　/usr/local/lib/libglscs.a
5. 画像変換ユーティリティ　/usr/local/bin/g_out

という 5 つのファイルです。

また、PostScript 形式のマニュアルが用意されています。これは ftp://ftp.st.ryukoku.ac.jp/pub/ryukoku/software/math/ から入手できます。桂田研の学生には印刷したものを配布することにしています。

プログラムの書き方 (1) 詳しくは GLSC のマニュアルを読むべきですが[4]、インクルード・ファイルの読み込みと、C や C++ の場合の浮動小数点数の宣言の問題を解決するおまじないについて説明しておきます。

(i) C の場合は、"`#define G_DOUBLE`"[5]、それに続けて "`#include <glsc.h>`" とする。

(ii) C++ の場合も "`#define G_DOUBLE`"、それに続けて

```
extern "C" {
#include <glsc.h>
};
```

とする (まあ、C++ で C のファイルをインクルードするときの定跡ですが)。

(iii) UNIX 伝統の FORTRAN (f77) の場合は、

```
include '/usr/local/include/glsc_ftn.h'
```

とする。

プログラムの書き方 (2) 以下述べることはどの言語にも共通しています。

(i) 最初に "`g_init("`*metafilename*`"`, ウィンドウの横幅, ウィンドウの高さ)" を呼び出します (サイズの単位は mm で、引数の型は浮動小数点

[4] GLSC のマニュアルにはサンプル・プログラムのソース・ファイルと実行結果も載っています。また、これらサンプル・プログラムのファイルが http://nalab.mind.meiji.ac.jp/~mk/labo/glsc-sample/ に置いてあります。

[5] この意味は、GLSC に属する関数で、浮動小数点数の型は `float` でなく `double` を使う、ということである。

型です)。メタファイルというのは、描画した図形を記録するためのファイルのことです。

(ii) 出力デバイスを “g_device(出力先)” 呼び出しで指定します。G_BOTH とすると、画面とメタファイルの両方に出力するようになります。

(iii) “g_def_scale()” で座標系を定義します。座標系は複数個定義できて、以下は番号を使って g_sel_scale(番号); として指定できます。
→ GLSC の中で、マルチ・ウィンドウもどきが簡単に実現できます。

(iv) “g_def_line()” を呼び出して、使用する線 (色、太さ、線種) を定義します。複数の線が定義できて、後から番号で呼び出せます。

(v) “g_cls()” で画面のクリアをします。

(vi) 既に “g_def_ほげほげ” で定義したものを “g_sel_ほげほげ” で指定します。

(vii) 色々な描画命令を並べます。

(viii) “g_sleep()” で停止します。

(ix) “g_term()” で GLSC を終了します。

コンパイルの仕方 最もフツーのコンパイルの仕方。

倍精度の場合 “-I/usr/local/include” と “-L/usr/local/lib -L/usr/X11R6/lib -lglscd -lX11 -lm” を指定するのが基本です[6](Solaris 2.6 の場合は -lsocket というのも必要になります)。

(i) gcc の場合は

```
gcc -o myprog -I/usr/local/include myprog.c -L/usr/local/lib -L/usr/X11R6/lib
  -lglscd -lX11 -lm
```

(2 行に分けてますが、1 行コマンドです。)

(ii) g++ の場合は

```
g++ -o myprog -I/usr/local/include myprog.C -L/usr/local/lib -L/usr/X11R6/lib
  -lglscd -lX11 -lm
```

(2 行に分けてますが、1 行コマンドです。)

(iii) FORTRAN の場合は

```
f77 -o myprog myprog.f -L/usr/local/lib -L/usr/X11R6/lib -lglscd -lX11
```

いずれも少々面倒なので、エイリアスを定義したり、Makefile を作ったりして、手間を軽減すべきでしょう。筆者は次のような行を .cshrc に書き加えています。

[6] コンパイル・オプション “-l ほげほげ” で lib ほげほげ.a をリンクすることになります。ここでは libglscd.a, libX11.a, libsocket.a, libm.a をリンクするわけです。

```
alias ccmg 'gcc -O -o \!^:r -I/usr/local/include \!* -L/usr/local/lib
  -L/usr/X11R6/lib -lmatrix -lglscd -lX11 -lm'
```

単精度の場合 "-lglscd" の代わりに "-lglscs" を指定するのが基本です。

3 印刷の仕方 (g_out の使い方)

(昔から悩みの種だったけれど、最近はとりあえず困っていません。)

例えば、C のプログラムで、

```
g_init("Meta", ...);
g_devie(G_BOTH);
```

のようにした場合 (G_BOTH は画面表示とメタファイル出力の両方を行なう、という意味ですが、G_META として画面表示は行なわず、メタファイルの出力のみを行なうことも出来ます)、プログラム終了後に、Meta という名前のファイルが出来ているはずです。

```
oyabun% g_out -v Meta
```

として Meta.ps という PostScript 形式のデータが出来ます。後は

再表示 PostScript 形式のデータは、Ghostscript などのビューアーで表示できます。

gs Meta.ps あるいは ghostview Meta.ps & あるいは
gv Meta.ps & あるいは ggv Meta.ps &

(ここら辺は環境による、ですね。)

MacOS の場合は、GhostScript がなくても、

```
open Meta.ps
```

で表示できます。

印刷 UNIX 風の環境で (最近の Mac もこれに属します)、PostScript プリンターが用意されていれば, lp Meta.ps あるいは lpr Meta.ps で印刷できるはずです。

- カラーのまま印刷したい場合は、`-i` というオプションをつけて Illustrator 形式に変換します。その場合、`Meta.i00` のようなファイル名になります。

- `-v` はポートレート形式にするという意味ですが、BoundingBox がかなりおかしな値になるので、`-v` をつけずに変換して、後から必要に応じて回転する方が良いかもしれません。例えば LaTeX に取り組む場合は、

```
\usepackage[dvips]{graphicx}
...
\includegraphics[angle=90,width=10cm]{Meta.i00}
```

  のように `angle=90` とします (`angle=` と `width=` の順番には意味があります)。

- `g_out` の作る PostScript には、一応 BoundingBox コメントはついていますが、(余白が大きかったりして) 値は使い物になりません。そうして作られた PostScript ファイルも、ps2epsi (あるいは ps2eps — 紛らわしい名前！) というプログラムで (まあまあまともな) BoundingBox のついた EPS 形式に変換できます。

こんな感じ

```
knoppix% g_out -i Meta
knoppix% ps2epsi Meta.i00
knoppix% ggv Meta.i00.epsi          (表示してチェック)
```

LaTeX への取り込みは

```
\includegraphics[angle=90,width=10cm]{Meta.i00.epsi}
```

- BoundingBox に負の座標が含まれている場合、クリップされて図が欠けたり (特に PDF にした場合)、色々と不都合が起こる場合があります。そういう場合は `ps2eps -t=`x`,`y で図を平行移動して (x, y の単位はポイント (1/72 インチ) である)、負の値をなくすと良いでしょう。

```
chronos% ps2eps -t=100,200 Meta.i00
```

  `Meta.i00.eps` というファイルが生成されます。なお、もとのファイルの名前が `Meta.ps` だった場合は `Meta.eps` という名前のファイルの生成されます (元々 `.i00` が風変わりな名前ということなのでしょう)。

- (2012 年 12 月現在の私のお勧め)

```
g_out -i Meta
ps2eps Meta.i00
```

```
\includegrahics[angle=90,width=10cm]{Meta.i00}
```

人にもらった or ずっと昔に作った (メタファイルが残っていない) PostScript ファイルに負の座標の BoundingBox が含まれていたら、`ps2eps -t=100,200 Meta.i00` とします。`\includegraphics` 時に `angle=` を指定するかどうかは、もちろんデータに依ります。

- (2015 年 4 月現在の私のお勧め — 今さら)

```
g_out -i Meta
ps2eps -f -R=- Meta.i00
```

```
\includegrahics[angle=90,width=10cm]{Meta.i00.eps}
```

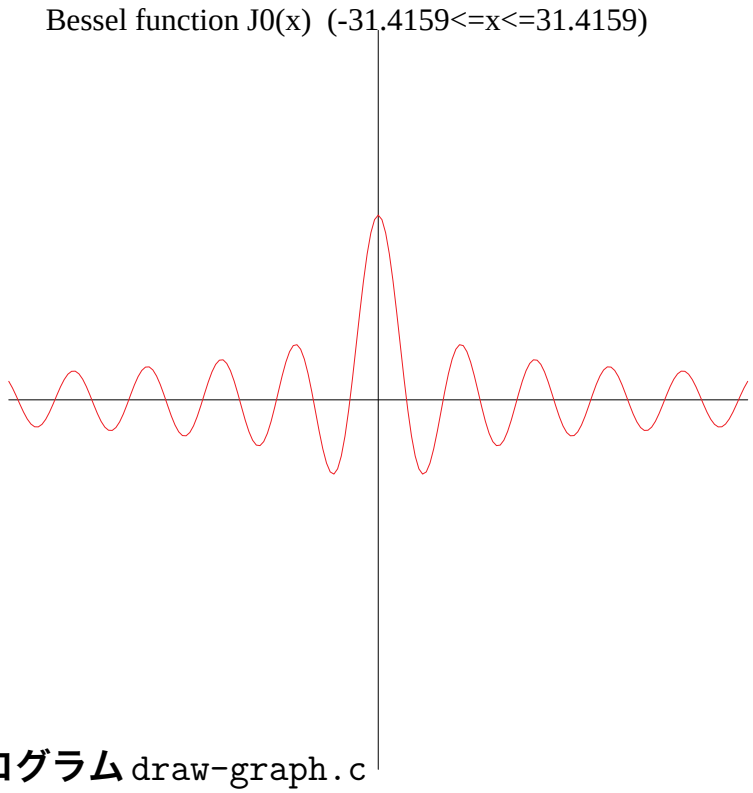
画像の回転は `g_out` の `-v` オプションではなく、`\includegraphics[]` の `angle=` オプションでもなく (LaTeX2HTML と相性が悪い)、`ps2eps` の `-R=` オプションを使うのが一番スッキリする。いくつか試した限りでは、座標の平行移動の必要もないみたい。

4 GLSC のサンプル・プログラム

4.1 何をするプログラムか

ありきたりですが、1 変数関数のグラフを描くプログラム例を示します[7]。0 次 Bessel 関数 `j0()` の $-10\pi \leq x \leq 10\pi$ の範囲のグラフを描きます。

[7] グラフを描く目的でしたら、gnuplot などを使う方が便利です。あくまでも GLSC の解説用のプログラムです。



4.2 ソースプログラム draw-graph.c

以下のプログラムは http://nalab.mind.meiji.ac.jp/~mk/labo/howto/glsc-progs/ に置いてあります。

```
 1  /*
 2   * draw-graph.c -- 1 変数関数のグラフを描く
 3   *    コンパイル:  ccmg draw-graph.c
 4   */
 5
 6
 7  #include <stdio.h>
 8  #include <math.h>
 9
10  #define G_DOUBLE
11  #include <glsc.h>
12
13  double pi;
14
15  int main()
16  {
```

```
17    int i, n;
18    double a, b, c, d;
19    double h, x;
20    double f(double);
21    char title[100];
22    double win_width, win_height, w_margin, h_margin;
23
24    pi = 4 * atan(1.0);
25
26    /* 表示する範囲 [a,b] × [c,d] を決定 */
27    a = - 10 * pi; b = 10 * pi; c = - 2.0; d = 2.0;
28
29    /* 区間の分割数 n */
30    n = 200;
31
32    /* GLSC の開始
33       メタファイル名、ウィンドウ・サイズの決定 */
34    win_width = 200.0; win_height = 200.0; w_margin = 10.0; h_margin = 10.0;
35    g_init("GRAPH", win_width  + 2 * w_margin, win_height + 2 * h_margin);
36
37    /* 出力デバイスの決定 */
38    g_device(G_BOTH);
39
40    /* 座標系の定義: [a,b] × [c,d] という閉領域を表示する */
41    g_def_scale(0,
42                a, b, c, d,
43                w_margin, h_margin, win_width, win_height);
44
45    /* 線を二種類用意する */
46    g_def_line(0, G_BLACK, 2, G_LINE_SOLID);
47    g_def_line(1, G_RED,   0, G_LINE_SOLID);
48
49    /* 表示するための文字列の属性を定義する */
50    g_def_text(0, G_BLACK, 3);
51
52    /* 定義したものを選択する */
53    g_sel_scale(0); g_sel_line(0); g_sel_text(0);
54
55    /* 座標軸を描く */
56    g_move(a, 0.0);  g_plot(b, 0.0);
57    g_move(0.0, c); g_plot(0.0, d);
58
59    /* タイトルを表示する */
60    sprintf(title, "Bessel function J0(x)  (%g<=x<=%g)", a, b);
61    g_text(20.0, 10.0, title);
62
63    /* 刻み幅 */
64    h = (b - a) / n;
65    /* グラフを描くための線種の選択 */
```

```
66    g_sel_line(1);
67    /* 折れ線でグラフを描く */
68    g_move(a, f(a));
69    for (i = 1; i <= n; i++) {
70      x = a + i * h;
71      g_plot(x, f(x));
72    }
73
74    /* ユーザーのマウス入力を待つ */
75    printf("終りました。X の場合はウィンドウをクリックして下さい。\n");
76    g_sleep(-1.0);
77    /* ウィンドウを閉じる */
78    g_term();
79    return 0;
80  }
81
82  double f(double x)
83  {
84    /* 0 次 Bessel 関数 */
85    return j0(x);
86  }
```

4.3 読んでみよう

4.3.1 GLSC を利用するための宣言

```
#define G_DOUBLE
#include <glsc.h>
```

4.3.2 どれが GLSC の命令か？

GLSC の関数はすべて、名前の先頭が g_ となっています。g_init(), g_device(), g_def_scale(), g_def_line(), g_def_text(), g_sel_scale(), g_sel_line(), g_sel_text(), g_text(), g_move(), g_plot(), g_sleep(), g_term() という関数を使っています。

4.3.3 ウィンドウのサイズ

ウィンドウのサイズは直接数値を書き込むと分かりづらくなるので、win_width + 2 * w_margin, win_height + 2 * h_margin のような式で表わしました。ここ

で win_width, win_height はそれぞれ表示領域の幅と高さで、w_margin, h_height はウィンドウの縁に用意する余白 (マージン) の大きさです。具体的な値は

```
win_width = 200.0; win_height = 200.0; w_margin = 10.0; h_margin = 10.0;
```

として与えています。こうして定めたウィンドウのサイズを

```
g_init("GRAPH", win_width  + 2 * w_margin, win_height + 2 * h_margin);
```

として GLSC に指示しています。ここで "GRAPH" は「メタファイル」の名前です。これは描画した図形の情報をファイルに記録する場合にはファイル名として採用されるものです。

4.3.4 出力先

g_device() によって、どこに出力するか指定します。選択肢は G_NONE (出力しない), G_DISP (ディスプレイ), G_META (メタファイル), G_BOTH (ディスプレイとメタファイルの両方) の 4 つです。サンプル・プログラムでは

```
g_device(G_BOTH);
```

として、画面にも表示するし、ファイルにも記録するように指定しています。

4.3.5 座標系の定義

「表示したい対象物の世界」における座標と出力デバイス (画面等) の座標との関係を定義する必要がありますが、これは、辺が座標軸に平行な長方形 (表示領域) を、二つの世界それぞれにおいて指定することで実現されます。

出力デバイスにおける長方形は、ウィンドウの左上隅の頂点からの余白 (水平方向, 垂直方向) と、長方形の辺の長さ (横、縦) の 4 つの数値を使って、指定されます。サンプル・プログラムでは、これらの値は w_margin, h_margin, win_width, win_height という変数に記憶されています。

「表示したい対象物の世界」における長方形は、プログラムの中の変数 a, b, c, d を用いて定義できる $[a, b] \times [c, d]$ です。

これらの情報を関数 g_def_scale() に渡すことになります。

```
/* 座標系の定義: [a,b] × [c,d] という閉領域を表示する */
g_def_scale(0,
            a, b, c, d,
            w_margin, h_margin, win_width, win_height);
```

第 1 の引数である 0 は、座標系の番号を表わします。実は GLSC では、一度に複数の座標系を定義しておいて、必要に応じて番号を使って呼び出すことができるようになっています。

4.3.6 線種、文字種の定義

描画に用いる線には、色、太さ、パターン (実線なのか点線なのか破線なのか) などの属性を持っています。GLSC では、これらの属性を備えた線種を定義することができます。

```
/* 線を二種類用意する */
g_def_line(0, G_BLACK, 2, G_LINE_SOLID);
g_def_line(1, G_RED,   0, G_LINE_SOLID);
```

同様に文字列を表示する場合の文字も、色と大きさの属性を持ち得ます。

```
/* 表示するための文字列の属性を定義する */
g_def_text(0, G_BLACK, 3);
```

4.3.7 定義しておいた座標系、線種、文字の呼び出し

これは簡単で g_sel_某 () という関数に番号を与えて呼び出すだけです。

```
/* 定義したものを選択する */
g_sel_scale(0); g_sel_line(0); g_sel_text(0);
```

4.3.8 線分を描く、グラフを描く

GLSC では、XY プロッター[8] 風の、「現在点から指定した点までの線分を引く」という機能を持った関数 g_plot(double x, double y) が用意されています。線を描かずに現在点のみを移動する機能の関数 g_move(double x, double y) と一緒に使うことで、自由に線分が描けます。

例えば x 軸を表示する目的で、サンプル・プログラムでは二点 $(a, 0)$, $(b, 0)$ を端点とする線分を

```
g_move(a, 0.0);  g_plot(b, 0.0);
```

として描いています。

g_move(), g_plot() のような関数があるときに、1 変数のグラフを描く手順は、次のようになります (定跡的であると言って良いでしょう)。

```
/* 折れ線でグラフを描く */
g_move(a, f(a));
for (i = 1; i <= n; i++) {
  x = a + i * h;
  g_plot(x, f(x));
}
```

4.3.9 マウスの入力待ち

何かを見せる目的のプログラムでは、ユーザーが好きな間だけ見ていられるように、他に何もしないで「待つ」ことが必要になります。GLSC ではそのために g_sleep(double t) という関数が用意されています (t として待ち時間を秒を単位として与えます)。

```
g_sleep(-1.0);
```

のように t として負の数を与えると、「ユーザーが GLSC のウィンドウをマウスでクリックされるまで待つ」ことになります。

[8] 今では XY プロッターと言っても目にする機会がなくなってしまったが、紙の上にペンをアームで動かすことにより、2 次元の線画を描く機械であった。現在ペンのある位置 (2 次元的な) が「現在点」のわけである。

4.3.10 終了 (ウィンドウの削除)

描画用のウィンドウを削除するには単に

```
g_term();
```

とするだけです。

5 GLSC+ について

5.1 その目的

当数学科での GLSC 利用の「歴史」も結構長くなって来ました。最初のうちは GLSC をオリジナルのままに使っていたのですが、しばしば等高線や鳥瞰図を描く関数が使いづらいと感じていました。その理由は、GLSC が C 言語で書かれているため、Fortran のような**整合配列**が利用できないためと言えるでしょう。格子点上の数値データや行列のような二重添字を持つ数列を二次元配列ではなく、長い 1 次元配列に (ユーザーが自分の責任で) 詰め込んで扱うのは、本質的でない複雑さが生じます。

そこで、長い 1 次元配列の代わりに

```
typedef G_REAL **matrix;
```

として定義した、G_REAL へのポインターのポインター (matrix 型) を使ったインターフェイス (C 言語用) を用意しました。

5.2 インストールの仕方

http://nalab.mind.meiji.ac.jp/~mk/program/graphics/glsc+2.tar.gz を入手する。

```
tar xzf glsc-3.5.tar.Z
tar xzf glsc+2.tar.gz -C glsc-3.5
cd glsc-3.5
cat README_GLSC+
```

後は README_GLSC+ の指示に従って下さい (最後の vi Makefile は emacs Makefile かも)。

5.3 新しく導入した関数

とにかく 2 変数関数のグラフの鳥瞰図と等高線が描ければよい、と考えたので、次の 3 つだけです。

1. **g_hidden2()**
2. **g_fake_hidden2()**
3. **g_contln2()**

使い方は名前の末尾に 2 がついていない関数とほとんど同じです。ただ一点 G_REAL へのポインターでなく、G_REAL へのポインターのポインターを引数として受け取るところが異なります。

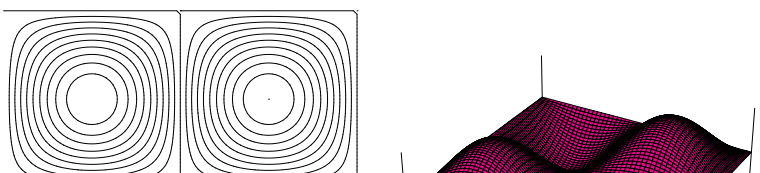
5.4 サンプル・プログラム・ソース

```
/*
 * test-contln2.c
 *   cc test-contln2.c -lmatrix -lglscd -lX11 -lm
 */

#define G_DOUBLE
#include <stdio.h>
#include <math.h>
#include "glsc.h"
#include <matrix.h>

#define W0 80.0
#define H0 80.0
#define W1 80.0
#define H1 80.0
#define W_MARGIN 10.0
#define H_MARGIN 10.0

double pi;

void compute(double (*)(), matrix,
        double, double, double, double,
        int, int);
double f(double, double);

double max(double x, double y) { return (x > y) ? x : y; }

int main()
{
    int m, n, k;
```

```
double xmin, xmax, ymin, ymax, f();
matrix u;

pi = 4 * atan(1.0);

/* 分割数 */
m = 100; n = 100;
/* 定義域は [-π , π] × [-π , π] */
xmin = - pi; xmax = pi; ymin = - pi; ymax = pi;

/* 格子点における数値を納める変数 */
if ((u = new_matrix(m+1,n+1)) == NULL) {
    fprintf(stderr, " 行列のためのメモリーが確保できませんでした。\n");
    return 1;
}
/* GLSC */
g_init("Meta", W0 + W1 + 3 * W_MARGIN, max(H0, H1) + 2 * H_MARGIN);
g_device(G_BOTH);
/* ウィンドウ 0 */
g_def_scale(0,
            xmin, xmax, ymin, ymax,
            W_MARGIN, H_MARGIN, W0, H0);
g_def_scale(1,
            xmin, xmax, ymin, ymax,
            W_MARGIN + W0 + W_MARGIN, H_MARGIN, W1, H1);
/* */
g_def_line(0, G_BLACK, 0, G_LINE_SOLID);
g_def_text(0, G_BLACK, 2);
/* 定義したものを呼び出す */
g_sel_scale(0);
g_sel_line(0);
g_sel_text(0);
/* title */
g_text(W_MARGIN + W0 * 0.6, H_MARGIN / 2, "contour and bird view");
/* 格子点上での関数値を計算する */
compute(f, u, xmin, xmax, ymin, ymax, m, n);
/* 等高線 */
for (k = -10; k <= 10; k++)
    g_contln2(xmin, xmax, ymin, ymax, u, m+1, n+1, 0.1 * k);
/* 鳥瞰図 */
g_hidden2(1.0, 1.0, 0.4,
          -1.0, 1.0,
          /* 視点 (距離, 方角を表わす ( θ , φ ) ) */
          5.0, 30.0, 30.0,
          W_MARGIN + W0 + W_MARGIN, H_MARGIN,
          W1, H1,
          u, m + 1, n + 1, 1,
          G_SIDE_NONE, 2, 1);
```

```
    printf("終了したらウィンドウをクリックして終了してください。\n");
    g_sleep(-1.0);
    g_term();

    return 0;
}

/*
 * [xmin,xmax] × [ymin,ymax] を x 軸方向に m 等分、y 軸方向に n 等分して
 * 各格子点上の f の値を u に格納する。
 */
void compute(double (*f)(), matrix u,
        double xmin, double xmax, double ymin, double ymax,
        int m, int n)
{
    int i, j;
    double dx, dy, x, y;

    dx = (xmax - xmin) / m;
    dy = (ymax - ymin) / n;
    for (i = 0; i <= m; i++) {
        x = xmin + i * dx;
        for (j = 0; j <= n; j++) {
            y = ymin + j * dy;
            u[i][j] = f(x, y);
        }
    }
}

double f(double x, double y)
{
    return sin(x) * sin(y);
}
```


contour and bird view



A　よく使う関数の説明

g_init

書式 `g_init(char *filename, G_REAL window_width, G_REAL window_height)`

機能 GLSC の初期化を行なう。
filename はメタファイルの名前。*window_width*, *window_height* はウィンドウのサイズ (単位は mm)。

例
```
g_init("Meta", 340.0, 220.0);
```

g_device

書式 `g_device(int device)`

機能 出力先デバイスを指定する。
device は出力先を指定する数値。以下の定数がインクルード・ファイルに定義されている。

G_NONE	出力しない
G_META	メタファイル
G_DISP	画面
G_BOTH	画面とメタファイル

例
```
g_device(G_BOTH);
```

g_def_scale

書式 g_def_scale(int scale_id,
G_REAL x_left, G_REAL x_right, G_REAL y_bottom,
G_REAL y_top,
G_REAL left_margin, G_REAL top_margin, G_REAL
width, G_REAL height)

機能 座標変換を定義する。
scale_id は座標変換につける番号 (複数定義して、番号で区別することができる)。ユーザー座標系の $[x_{\mathrm{left}}, x_{\mathrm{right}}] \times [y_{\mathrm{bottom}}, y_{\mathrm{top}}]$ という長方形閉領域を $[\mathrm{left\_margin}, \mathrm{left\_margin} + \mathrm{width}] \times [\mathrm{top\_margin}, \mathrm{top\_margin} + \mathrm{height}]$ という長方形閉領域に写像する。

例

```
g_def_scale(0, -1.6, 1.6, -1.0, 1.0,
                  10.0, 10.0, 320.0, 200.0);
```

g_def_line

書式 `g_def_line(int line_id, int color_id, int line_width, int line_type)`

機能 線 (というよりはペン) を定義する。
line_id は線につける番号 (複数定義して、番号で区別することができる)。
色が `color_id`、太さが `line_width`, 種類が `line_type` の線に `line_id` という番号をつける。

- 色としては

G_BLACK	黒	0
G_RED	赤	1
G_GREEN	緑	2
G_BLUE	青	3
G_MAGENTA	マゼンタ (赤紫)	4
G_YELLOW	黄	5
G_CYAN	シアン (澄んだ青緑色、赤の補色)	6
G_WHITE	白	7

が使える。

- 線の太さは 0 から 3 まで。0 と 1 は太さ同じだが 0 の方が速い。
- 線種としては

G_LINE_SOLID	実線
G_LINE_DOTS	
G_LINE_DASHED	
G_LINE_LONG_DASHED	
G_LINE_THIN_DOTS	
G_LINE_DOT_DASHED	
G_LINE_D_DOT_DASHED	

例

```
g_def_line(0, G_BLACK, 0, G_LINE_SOLID);
```

g_cls

書式 g_cls()

機能 画面の消去を行なう (これまでに描かれたものを削除する)。

例

```
g_cls();
```

g_sleep

書式 g_sleep(G_REAL time)

機能 time が正の場合、time 秒停止する。time が負の場合、マウスをクリックするまで停止する。

例

```
g_sleep(-1.0); /* マウスをクリックするまで待つ */
```

g_term

書式 g_term()

機能 GLSC を終了する。

例

```
g_term();
```

B PostScript への変換 (g_out に関するノウハウ)

(3 「印刷の仕方 (g_out の使い方)」を読むことをお勧めします。)

関数 g_init() の第一引数で指定した「メタファイル」には、描画した図形データの内容が記録されていて、コマンド g_out により、PostScript 形式に変換できる。

-v を指定するとポートレイト形式になる。ただし PostScript に変換後の座標が負になったりするので、使わないほうが無難かもしれない (ps2epsi に失敗するようになる原因となる？PDF にすると図が欠ける？)。TEX で \includegraphics を用いて取り込む場合には、

```
\includegraphics[width=10cm,angle=90]{mygraph.i00}
```

のように angle=角度 オプションで回転できるので、あえて中途半端な -v オプションに頼る必要はないかもしれない (私は最近は使わなくなりました)。

出来上がった PostScript ファイルの BoundingBox コメントは、“A4 紙 1 枚全部” というものらしく、LaTeX に取り込むには余白が生じるのが普通で不適当なので、ps2epsi コマンドで直すか、ghostview などで表示させて測った値をテキスト・エディターで書き込むのがよい。

特に -i オプションを指定すると、Adobe のイラストレーター形式でセーブされるという。これも一種の PostScript であることに違いはないが、カラーで描いた図がモノクロのデータに変換されることがないので、色つきの画像データを作るときに使える。また、複数ページからなるメタファイルが、1 ページ毎のばらばらのファイルに変換されることも、場合によっては便利である。

線の太さや文字の大きさなどの情報は、PostScript に変換すると消えてしまうが、例えば使用している文字をすべて大きくして構わないのならば、直接 PostScript ファイルを編集して直すことが比較的容易である。

熱方程式や波動方程式のような発展系の数値計算結果を可視化した場合、一つのメタファイルに複数の図が記録される。このとき、一枚の紙に複数の図を連ねて表示する「紙芝居」を作るには、-f 行の数,列の数 と -m 倍率 というオプションを使うとよい。例えば

```
g_out -vfm 4,3 0.4 Meta
```

とすると、1 ページあたり 4 段 (行？) 3 列、全部で 12 の図が入った PostScript ファイルが出来る。

C 桂田研学生向け

桂田研の学生にはおなじみの heat1d-e.c の GLSC 版です。

```
/*
 * heat1d-e-glsc.c -- 1 次元熱伝導方程式の初期値境界値問題を陽解法で解く。
 *   コンパイル:  ccmg heat1d-e-glsc.c
 *
 * オリジナルは fplot ライブラリィを利用した
 *  http://www.math.meiji.ac.jp/%7Emk/program/fdm/heat1d-e.c
 */
```

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

#define G_DOUBLE
#include <glsc.h>

int main()
{
  int i, n, nMax, N;
  double tau, h, lambda, Tmax;
  double *u, *newu;
  double f(double);
  double win_width, win_height, w_margin, h_margin;
  char message[100];

  /* N, λ を入力する */
  printf("区間の分割数 N = "); scanf("%d", &N);
  printf("λ (=τ/h^2) = "); scanf("%lf", &lambda);

  /* h, τ を計算する */
  h = 1.0 / N;
  tau = lambda * h * h;
  printf("τ=%g\n", tau);

  /* 最終時刻を入力する */
  printf("最終時刻 Tmax = "); scanf("%lf", &Tmax);

  /* ベクトル u, newu を用意する */
  u = malloc(sizeof(double) * (N+1));
  newu = malloc(sizeof(double) * (N+1));

  /* 初期値の代入 */
  for (i = 0; i <= N; i++)
    u[i] = f(i * h);

  /* ***************** グラフィックスの準備 ***************** */
  /* メタファイル名は "HEAT",
   * ウィンドウのサイズは、
      横 win_width  + 2 * w_margin, 縦 win_height + 2 * h_margin */
  win_width = 200.0; win_height = 200.0; w_margin = 10.0; h_margin = 10.0;
  g_init("HEAT", win_width  + 2 * w_margin, win_height + 2 * h_margin);
  /* 画面とメタファイルの両方に記録する */
  g_device(G_BOTH);
  /* 座標系の定義: [-0.1,1.1] × [-0.1,1.1] という閉領域を表示する */
  g_def_scale(0,
              -0.1, 1.1, -0.1, 1.1,
              w_margin, h_margin, win_width, win_height);
  /* 線を二種類用意する */
```

```
  g_def_line(0, G_BLACK, 0, G_LINE_SOLID);
  g_def_line(1, G_BLACK, 0, G_LINE_DOTS);
  /* 表示するための文字列の属性を定義する */
  g_def_text(0, G_BLACK, 3);
  /* 定義したものを選択する */
  g_sel_scale(0); g_sel_line(0); g_sel_text(0);

  /* タイトルと入力パラメーターを表示する */
  g_text(30.0, 30.0,
         "heat equation, homogeneous Dirichlet boundary condition");
  sprintf(message, "N=%d, lambda=%g, Tmax=%g", N, lambda, Tmax);
  g_text(30.0, 60.0, message);

  /* 座標軸を表示する */
  g_sel_line(1);
  g_move(-0.1, 0.0); g_plot(1.1, 0.0);
  g_move(0.0, -0.1); g_plot(0.0, 1.1);
  g_sel_line(0);

  /* t=0 の状態を表示する */
  g_move(0.0, u[0]);
  for (i = 1; i <= N; i++)
    g_plot(i * h, u[i]);

  /* ループを何回まわるか計算する (四捨五入) */
  nMax = rint(Tmax / tau);

  /* 時間に関するステップを進めるループ */
  for (n = 0; n < nMax; n++) {
    /* 差分方程式 (n -> n+1) */
    for (i = 1; i < N; i++)
      newu[i] = (1.0 - 2 * lambda) * u[i] + lambda * (u[i+1] + u[i-1]);
    /* 計算値を更新 */
    for (i = 1; i < N; i++)
      u[i] = newu[i];
    /* Dirichlet 境界条件 */
    u[0] = u[N] = 0.0;
    /* この時刻 (t=(n+1) τ) の状態を表示する */
    g_move(0.0, u[0]);
    for (i = 1; i <= N; i++)
      g_plot(i * h, u[i]);
  }

  printf("終りました。X の場合はウィンドウをクリックして下さい。\n");
  g_sleep(-1.0);
  /* ウィンドウを閉じる */
  g_term();
  return 0;
}
```

```
double f(double x)
{
  if (x <= 0.5)
    return x;
  else
    return 1.0 - x;
}
```

D　Cygwin+XFree86 環境での利用

個人的に GLSC がすごいと思うのは、X の基本的な機能しか使っていないので (その点非常に禁欲的です)、様々なシステム (GLSC の開発後に登場してきたものでも) に対して、無修整あるいは非常に微弱な修正で利用できるようになることです。

最近、普及してきた Cygwin+XFree86 という環境でもごく簡単に利用できます。自力で make するのも簡単ですが、バイナリーを用意しました。http://nalab.mind.meiji.ac.jp/~mk/labo/cygwin/cygwin-glsc+.tar.gz から cygwin-glsc+.tar.gz を入手して、

```
tar xzf cygwin-glsc+.tar.gz -C /usr/local
```

のように展開すれば使えるようになります (ヘッダー・ファイルを /usr/local/include, ライブラリィ・ファイルを /usr/local/lib に格納します。コンパイルは /usr/local/bin/ccmg というスクリプトで可能です。スクリプトを見ればコンパイル&リンクの仕方が分かるでしょう)。

(2006/6/12 追記) Cygwin 環境だと、**glscwin** の方が便利かも知れません。次の節で解説します。

E　glscwin について

E.1　誰が作ったもの？入手するには？

オリジナル GLSC の開発者の一人である高橋大輔氏の研究室によって、GLSC が Win32 環境[9] に移植されたものが **glscwin** です。

[9]具体的な OS の名前で言うと、Windows 95, Windows 98, Windows Me, Windows NT, Windows 2000 のことを指します。

しばらく公開をやめたように思っていたのですが高橋大輔研究室[10]の講義のページにあるのを見つけました。「数値計算法 A[11]内に glscwin-20070914.zip[12]が置かれています。またマニュアル glscm.zip[13]もあります。

龍谷大学にあるページ

http://sparrow.math.ryukoku.ac.jp/~junta/edu/nc2000/glscman/glscm.html

は説明を読むのに重宝しています。

E.2 特徴

glscwin にはオリジナルの GLSC にはない利点が色々あります。

(1) 画像を EMF (Enhanced Meta File) 形式で出力可能。
(Windows で表示が出来、動画を連番ファイルで出力しておくと、紙芝居というか簡易アニメーションが出来る — 何を言っているのか分からないかも。百聞は一見に如かずなのだが…)

(2) マウスのイベントを扱える。

(3) 使える色が多い (いわゆるフルカラー)
例えば `g_density_plot_color()` という、2 変数関数のレベルを色で塗り分ける関数があって便利です。

明治大学数学科の 6701 号室の Windows パソコンには、`glsc_ver0.82.lzh` をインストールしてあります。コンパイルには、`/usr/local/?/cglscs`, `/usr/local/?/cglscd` というスクリプトを使うと良いでしょう。

[10] http://hakotama.jp/
[11] http://hakotama.jp/report/SuuchiKeisanHouA/
[12] http://hakotama.jp/report/glscwin/glscwin-20070914.zip
[13] http://hakotama.jp/report/glscwin/glscm.zip

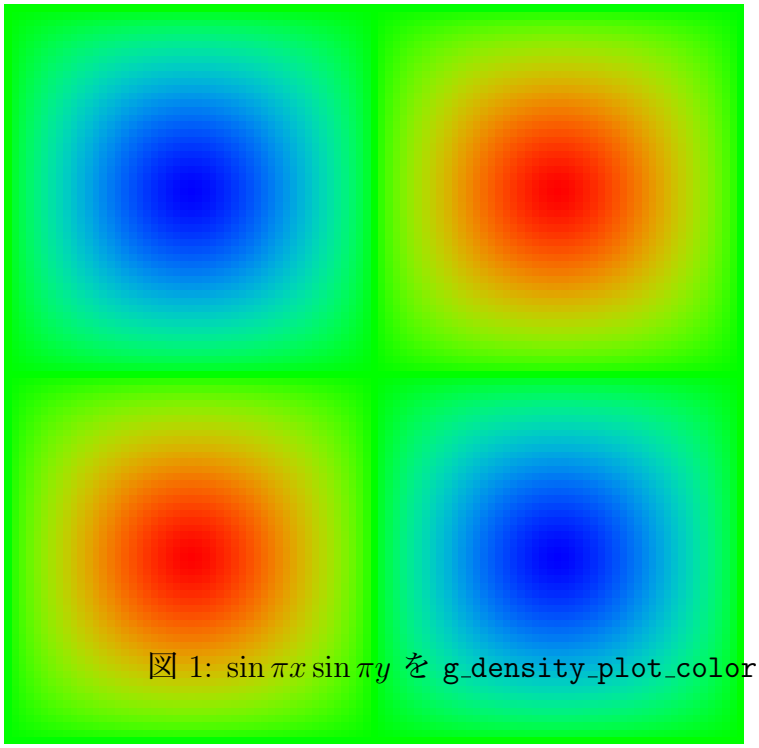


図 1: $\sin \pi x \sin \pi y$ を `g_density_plot_color()` で見る

glscd

```
#!/bin/sh

GLSCDIR=/usr/local/glscwin-ver0.82
KANJI="-finput-charset=cp932 -fexec-charset=cp932"
#CFLAGS="-W -Wall -O -DG_DOUBLE -I$GLSCDIR/include"
CFLAGS="-O -DG_DOUBLE -I$GLSCDIR/include"
LDFLAGS="-L$GLSCDIR/lib -lglscd -luser32 -lgdi32 -lwinmm -lcomdlg32 -lcomctl32 -lm"

prog=‘basename $1 .c‘

gcc $KANJI $CFLAGS -o $prog "$@" $LDFLAGS
```

E.3 C++ で glscwin を利用する

ずっと以前に書いたメモ 『glscwin を C++ プログラムから使う』[14] を読んで下さい (無保証です)。

E.4 インストール・メモ

手元に glscwin_0.82.lzh というファイルがある。

まずはばらす

```
mkdir glscwin_0.82
cd glscwin_0.82
lha x ../glscwin_0.82.lzh
```

[14] http://nalab.mind.meiji.ac.jp/~mk/labo/howto/GLSCWIN_C++.txt

glsc_ext.c の修正

```
mathpc% diff -c glsc_ext.c.org glsc_ext.c
*** glsc_ext.c.org      Mon Jun 28 03:30:28 1999
--- glsc_ext.c  Tue May 20 23:32:29 2008
***************
*** 797,803 ****
    g_area_rgb(c.r, c.g, c.b);
  }
  /*************************************/
! double g_color_gen_func(x)
  /*************************************/
    G_FLOAT x;
  {
--- 797,803 ----
    g_area_rgb(c.r, c.g, c.b);
  }
  /*************************************/
! G_FLOAT g_color_gen_func(x)
  /*************************************/
    G_FLOAT x;
  {
mathpc%
```

コンパイルしてライブラリィ・アーカイブ・ファイルを作る

```
mathpc% cat make.sh
#!/bin/sh
#CFLAGS=-W -Wall
gcc $CFLAGS -c -O glsc.c
gcc $CFLAGS -c -O glsc_ext.c
#gcc $CFLAGS -c -O glsc_win.c
gcc $CFLAGS -c glsc_win.c
gcc $CFLAGS -c -O ezfont.c
ar cru libglscs.a glsc.o glsc_ext.o glsc_win.o ezfont.o
ranlib libglscs.a

gcc $CFLAGS -c -O -DG_DOUBLE glsc.c
gcc $CFLAGS -c -O -DG_DOUBLE glsc_ext.c
#gcc $CFLAGS -c -O -DG_DOUBLE glsc_win.c
gcc $CFLAGS -c -DG_DOUBLE glsc_win.c
gcc $CFLAGS -c -O ezfont.c
ar cru libglscd.a glsc.o glsc_ext.o glsc_win.o ezfont.o
ranlib libglscd.a
mathpc%
```

glsc_win.c だけコンパイルに -O をつけていないが、

```
    while (TRUE) {
      if (w_lbutton_down == G_YES) {
        break ;
      }
    }
```

というくだりがあるからである。volatile する方が正しいか？

倍精度ライブラリィとコンパイル＆リンクするためのスクリプト例— cglscd

```
#!/bin/sh

GLSCDIR=/usr/local/glscwin-ver0.82
KANJI="-finput-charset=cp932 -fexec-charset=cp932"
#CFLAGS="-W -Wall -O -DG_DOUBLE -I$GLSCDIR/include"
CFLAGS="-O -DG_DOUBLE -I$GLSCDIR/include"
LDFLAGS="-L$GLSCDIR/lib -lglscd -luser32 -lgdi32 -lwinmm -lcomdlg32 -lcomctl32 -lm"

prog=‘basename $1 .c‘

gcc $KANJI $CFLAGS -o $prog "$@" $LDFLAGS
```

E.5 サンプル

以下のプログラムは http://nalab.mind.meiji.ac.jp/~mk/labo/howto/glsc-progs/ に置いてあります。

```
/*
 * testdensity.c --- g_density_plot_color() のテスト
 *   GLSCWIN が必要です。
 *   桂田研パソコンならば glscd testdensity.c でコンパイル可能
 */

#include <stdio.h>
#define G_DOUBLE
#include <glsc.h>

#define N (100)

double pi;

double f(double x, double y)
{
  return sin(pi * x) * sin(pi * y);
}

int main()
```

```
{
  double xmin, xmax, ymin, ymax, w_margin, h_margin, w_width, w_height;
  G_REAL u[N+1][N+1];
  int i, j, n, nx, ny;
  double x, y, dx, dy;

  nx = ny = N;
  pi = 4 * atan(1.0);

  /* 矩形領域を定める */
  xmin = -1.0; xmax = 1.0; ymin = -1.0; ymax = 1.0;
  /* ウィンドウのサイズ */
  w_margin = 1.0; h_margin = 1.0; w_width = 100.0; w_height = 100.0;
  /* GLSC の開始 */
  g_init("DENSITY", w_width + 2 * w_margin, w_height + 2 * h_margin);
  g_device(G_BOTH);
  /* 座標系を決める */
  g_def_scale(0,
              xmin, xmax, ymin, ymax,
              w_margin, h_margin, w_width, w_height);
  g_sel_scale(0);
  /* 格子点上での関数値の計算 */
  dx = (xmax - xmin) / nx;
  dy = (ymax - ymin) / ny;
  for (i = 0; i <= nx; i++) {
    x = xmin + i * dx;
    for (j = 0; j <= ny; j++) {
      y = ymin + j * dy;
      u[i][j] = f(x,y);
    }
  }
  /* */
  g_density_plot_color((G_REAL *)u, nx + 1, ny + 1, 0, nx, 1, 0, ny, 1,
                       G_NO, G_NO,
                       xmin, xmax, ymin, ymax,
                       -1.0, 1.0);
```

```
  g_sleep(-1.0);
  g_term();
  return 0; // もしマウス・クリックだけで終了したいのならば exit(0); とする。
}
```

E.6 EMF について

glscwin で作ったファイルは、拡張子が .emf のいわゆる Windows メタファイルである。

- Windows 標準のコマンド、例えばペイントでも扱える。
- 他のイメージ・フォーマットに変換するには、Windows 版 ImageMagick の convert を使えば良い (Windows 版でないとダメのようである)。

```
set path=(/cygdrive/c/Program\ Files/ImageMagick-6.2.8-Q16 $path)
convert -geometry "25%" mygraph.emf mygraph.jpg
```

- PostScript に変換するには、convert でも可能であるが、シェアウェアの WMF2EPS[15] を使うこともできる。

E.6.1 wmf2eps のインストール

1. http://www.wmf2eps.de.vu/ から入手する。2008年10月12日現在、WMF2EPS1.32.ZIP が最新版である。
2. wmf2eps.exe を適当な場所にコピーして、実行できるようにする。
3. プリンタ “WMF2EPS Color PS L2” をインストールをする。
   [プリンタと FAX] →[プリンタのインストール] →[プリンタの種類を指定してください] に対して「このコンピュータに接続されているローカルプリンタ」、[次のポートを使用] に対して「FILE:」を選択、[プリンタ ソフトウェアのインストール] で [ディスクの使用] を選び、WMF2EPS1.32¥PSprint¥Win2000¥Standard¥W2kPrint.I を選択する。

[15] http://www.wmf2eps.de.vu/

4. プリンタのプロパティの設定
   WMF2EPS Color PS L2 のプロパティから、全般→印刷設定→詳細設定を開く。

   (a) PostScript オプション→PostScript 出力オプションを “EPS(Encapsulated PostScript)” にする。

   (b) グラフィックスの印刷品質は “1200dpi”

   (c) TruType フォントは “ソフトフォントとしてダウンロード”

   (d) TruType フォントダウンロードオプションは “自動”

以上は、[16] を参考にした。
なお、古いフリーのバージョンがあり、Windows 2000, Windows XP でも使えるらしい。「wmf2eps の旧バージョンを Win2k で使う方法」[17] を見よ。

F　有向線分 (矢印) の描画

```
/*
 * arrow.c --- 有向線分の描画
 */

#include <stdio.h>
#include <math.h>
#define G_DOUBLE
#include <glsc.h>

/* ------------------------------------------------------------ */

/* π, 度とラジアンの換算用の比 */
double arrow_pi, arrow_degree;
int arrow_initialized = 0;

void init_arrow()
{
  arrow_initialized = 1;
  arrow_pi = 4 * atan(1.0); arrow_degree = arrow_pi / 180.0;
}

/* 有向線分 (矢印のある線分) を描く */
void arrow(double p1, double p2,
           double q1, double q2,
           double lambda)
```

[16] http://www.bunmeisha.co.jp/LaTeX2e/latex2e_eps.html
[17] http://rakasaka.fc2web.com/tex/tex.html

```
{
  double e1, e2, norm;
  double fl1, fl2, fu1, fu2, theta, cost, sint;
  double qu1, qu2, ql1, ql2, r1, r2, t;
  double phi;
  double vx[5], vy[5];
  int fill;

  if (!arrow_initialized)
    init_arrow();
  /* */
  theta = 15.0 * arrow_degree; phi = 30.0 * arrow_degree;
  /* e=(p-q)/||p-q|| */
  e1 = p1 - q1; e2 = p2 - q2; norm = hypot(e1, e2);
  e1 /= norm; e2 /= norm;
  /* fl = λ R(θ) e */
  cost = cos(theta); sint = sin(theta);
  fl1 = lambda * (cost * e1 - sint * e2);
  fl2 = lambda * (sint * e1 + cost * e2);
  /* fu = λ R(-θ) e */
  fu1 = lambda * ( cost * e1 + sint * e2);
  fu2 = lambda * (- sint * e1 + cost * e2);
  /* ql = q + fl */
  ql1 = q1 + fl1; ql2 = q2 + fl2;
  /* qu = q + fu */
  qu1 = q1 + fu1; qu2 = q2 + fu2;
  /* r = q + λ cos θ (1-tan θ/tan φ)e */
  t = lambda * cost * (1 - tan(theta) / tan(phi));
  r1 = q1 + t * e1; r2 = q2 + t * e2;
  /* */
  vx[0] = q1; vy[0] = q2;
  vx[1] = qu1; vy[1] = qu2;
  vx[2] = r1;  vy[2] = r2;
  vx[3] = ql1; vy[3] = ql2;
  vx[4] = q1;  vy[4] = q2;
  g_sel_area(0);
  fill = G_YES;
  /* */
  g_move(p1, p2); g_plot(r1, r2);
  g_polygon(vx, vy, 5, G_NO, fill);
}

/* ------------------------------------------------------------ */

int main()
{
  g_init("META", 220.0, 220.0);
  g_device(G_BOTH);
  g_def_scale(0,
```

```
                -1.0, 5.0, -1.0, 5.0,
                10.0, 10.0, 200.0, 200.0);
  g_sel_scale(0);
  /* arrow のために必要 */
  g_def_area(0, G_BLACK);

  g_def_line(0, G_BLACK, 3, G_LINE_SOLID);
  g_sel_line(0);

  arrow(1.0, 1.0, 4.0, 1.0, 0.8);
  g_sleep(-1.0);
  g_term();
  return 0;
}
```

G　GLSC で改善して欲しい点

- ウィンドウの再描画をすること。

言い出しっぺの法則 ???