gnuplot と C 言語

東京電機大学 松田七美男

平成 25 年 3 月 8 日

目 次

1　gnuplot を利用した計算結果の可視化

結果を画面上のグラフに表示するためには，本来グラフィックライブラリを使うというのが正道かもしれません．実際に過去に何度も，公的にグラフィックの標準化という試み（例えば，GKS,CGM ）が行われましたが，ついに普及することはありませんでした．唯一例外的に，SGI（グラフィック業界を席巻した企業）のライブラリをオープン化した **OpenGL** のみが普及したといえましょう．

OpenGL を使ったグラフィックは大変楽しいのですが，それだけで半期の講義となってしまいますので，ここではグラフを描くツールを**子プロセスで起動してパイプ接続し**コマンドを送りつける方法を選択します．このような用途には **Gnuplot** がうってつけです．

1.1　データファイルのプロット

1.1.1　データファイルの準備

手始めにファイルに保存されたデータをグラフ化してみましょう．データには，$\boldsymbol{E}\times\boldsymbol{B}$ ドリフトの $t, x(t), y(t), z(t)$ を例に取り上げます．`rk4v6EB.c` を $t, x(t), y(t), z(t)$ が出力されるように手直ししてください．出力リダイレクトを利用して，データファイル `sample1.dat` を作成してください．

```
$ rk4v6_1 > sample1.dat
```

1.1.2　gnuplot の起動：対話モード

GNOME 端末などの端末上で，オプションを指定せずに gnuplot を起動してください．

```
$ gnuplot
```

すると，図 1 のように greeting message を表示した後に，gnuplot のプロンプトが表示され，コマンド入力待ちとなります．

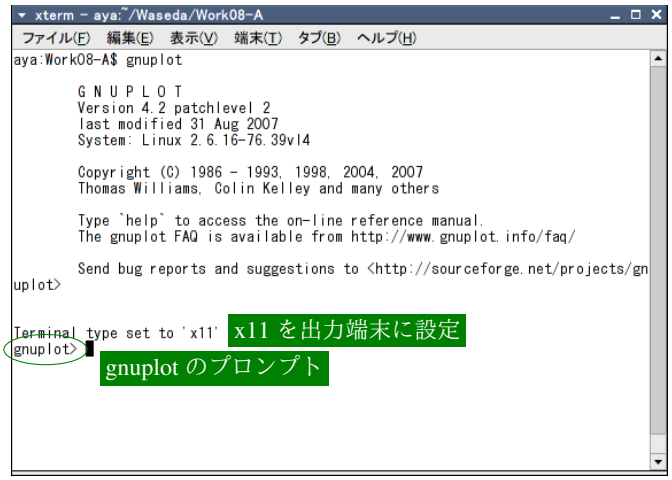


図 1　gnuplot の初期起動画面：対話モードの開始

今まで，端末からコマンドを起動する場合には次のコマンドが実行できるようにするため，行末に & を加えて処理をバックグラウンドに回すようにしてきました．が，gnuplot を対話モードで使用する場合には，gnuplot が端末を占有しますから，& を行末につけるのは無意味となります．

1.1.3　2 次元プロット：plot

2 次元のグラフのプロット命令は，**plot** です．データをプロットする場合には，ファイル名を引数に指定します．データファイルには，データが 1 行ごとに収められ，各行は空白文字（タブも可）でフィールドが区切られていることを前提に処理されます．default では第 1 列を横軸，第 2 列を縦軸にして散布図を描きます．

```
plot "データファイル名"　オプション
```

数多くのオプションがありますが，良く使うものを表 1 に示します．

表 1　2 次元プロット命令 plot の主なオプション

オプション	文字列	引数
種類	with	points | lines | linespoints |*etc.*
記号種	pointtype	整数
記号サイズ	pointsize	実数
線種	linetype	整数
線幅	linewidth	実数
データ列	using	列番号:列番号

作成したデータファイル名をプロットしましょう．gnuplot の対話モード上で，

```
gnuplot > plot "sample1.dat"
```

と命令すると，図 2 のような Window が X11 の画面に表れます．

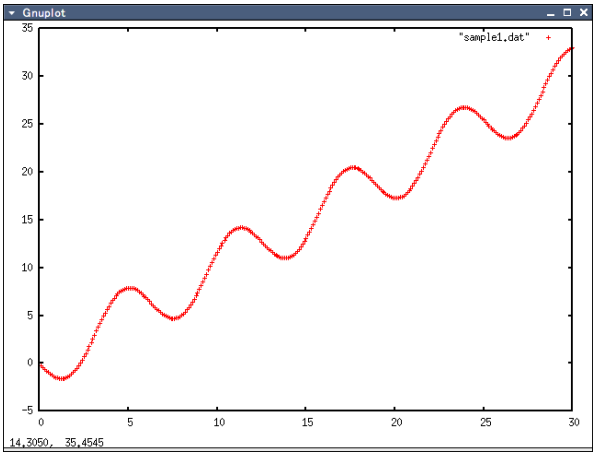


図 2　データファイル”sample1.dat”のプロット

オプション文字列は長いので略記できます．例えば，大きさ 1 の○記号を指定して，折れ線と記号の両方を描かせるには，以下のように記述します．

```
gnuplot > plot "sample1.dat" w lp pt 6 ps
1.0
```

with を w，linespoints を lp，pointtype を pt，pointsize を ps と略記しています．

$x(t)$ を横軸，$y(t)$ を縦軸に指定して軌跡を描いてみましょう．$x(t), y(t)$ はデータファイルの第 2，3 列に保存されていますから，using オプションを用いて，折れ線で描かせます．

```
gnuplot > plot "sample1.dat" using 2:3 w l
```

```
plot "データファイル名" using n:m
```

すると，次のような妙な図が表れます（図 3）．

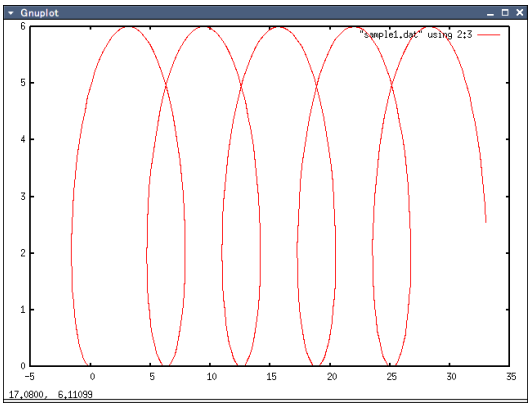


図 3　”sample1.dat”のプロット：軌跡 (x, y) を描こうとしたが，軸範囲の設定が不適切なため，縦に伸びてしまった．

これは軸の範囲を全てのデータが収まるように自動で設定する機能 autoscale が働くからです．軸の範囲は以下のように set コマンドを用いて設定します．

```
set xrange[端の値:端の値]　　(yrange も同様)
```

縦横比がほぼ 4:3 であることから，以下のように軸の範囲を設定すれば，縦横の縮尺がおおよそ等しい図が得られるはずです．

```
gnuplot > set xrange [0:40]
gnuplot > set yrange [-15:15]
```

ついでに，x, y 軸に名前をつけましょう．

```
gnuplot > set xlabel "X-axis"
gnuplot > set ylabel "Y-axis"
gnuplot > replot
```

最後の replot は，最近の plot 命令を実行するコマンドです．

```
set xlabel "文字列（x 軸名）"
set ylabel "文字列（y 軸名）"
replot :　最近のプロットを再描画する命令
```

1.1.4　複数のプロット

1 つのプロットに複数の線を載せたい場合があります．この例では，データの密度が高いので散布図では記号が重なってしまいます．そこで間引いて描かせましょう．データの開始と増分はオプション every を用いて制御します．

```
plot every {<point_incr>}
           {:<block_incr>}
             {:<start_point>}
               {:<start_block>}
                 {:<end_point>}
                   {:<end_block>}
```

ここに block はデータブロックを指します．gnuplot は一つのファイルに含まれる複数のデータ（区切りは空行）を認識します．**point**部の記述により，1 つのデータブロックにおいて描画データの開始位置，終了位置，位置の増分を指定できるというわけです．例えば，散布図で 5 つ毎に記号を描画するなら，

```
plot ”***” using 2:3 every 5 w p
```

とすればよいことになります．複数の線を描くには区切りに半角英数のカンマ「,」を用いて列記します．

```
plot  "データファイル名 1" オプション,\
      "データファイル名 2" オプション
```

オプションが長い場合には**行の継続の印にバックスラッシュを用いて複数行に分割**できます．また，**同一データファイルを用いる場合には名前を省略可能**です．

1.2　スクリプトファイル

命令行数が多くなってくると修正が繁雑になりますので，スクリプトを gedit などのエディタで編集（修正も）し，そのスクリプトファイル名を引数に渡して gnuplot を立ち上げてみましょう．このような非対話モードはバッチ（batch）モードとも呼ばれます．バッチモードでは，スクリプトファイルの末尾の命令の実行終了とともに全体が終了します．すると，X11 上に開いたプロット Window も一瞬の内に終了してしまいます（開いたかどうかも判らない位短いかもしれません）．

X11 上のプロット Window を暫く眺める余裕を確保するには，2 つの方法があります．

- プロット命令以降任意の場所に処理の一時停止コマンド **pause** s を置く．実数 s は秒単位の停止時間ですが，-1 を与えると gnuplot を起動した端末でエンターキーが押されるまで中断することになります．
- 起動時にオプション-persist を指定する．この場合には，プロット Window は強制終了させなければなりません．

例題 1　今までのコマンドを sample1.plt という名前のスクリプトファイルに記述し，gnuplot をそのスクリプトファイル名を引数に与えて起動しなさい．

[解]　リスト 1　sample1.plt

```
set xrange [0:40]
set yrange [-15:15]
set xlabel "X-axis"
set ylabel "Y-axis"
plot "sample1.dat" using 2:3 w l, \
     "" using 2:3 every 5 w p pt 7 ps 2
pause -1
```

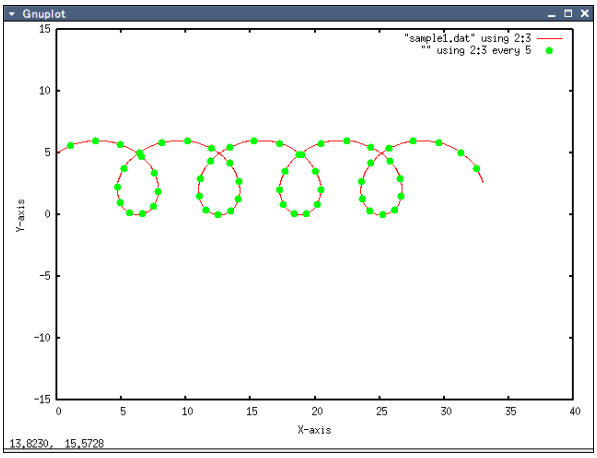


図 4　”sample1.dat”のプロット：軌跡 (x, y)

問題 1　rk4 の出力をデータファイルに保存して，Runge-Kutta 法の数値計算結果と解析解を比較する図を作成しなさい．

1の解　解答例を図 5 に示します．

1.2.1　3 次元プロット：splot

鳥瞰図と呼ばれる 3 次元のプロット命令は，**splot** です．3 次元的な表面を描くのが default ですが，それは後に説明するとして，ここでは 2 次元プロット plot と同様に，3 次元データ (x, y, z) を結んだ線をプロットしてみましょう．ファイル名を引数に与え，using n:m:l のように 3 つの列番号を指定します．

```
splot "データファイル名" オプション
```

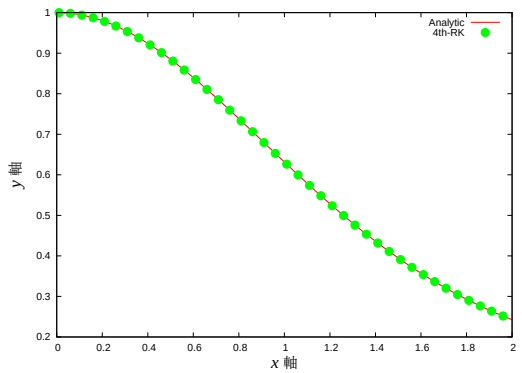


図 5　rk4 の数値計算結果と解析解を比較した図

splot 特有のオプションには，z 軸に関係するものは当然として，図が描かれる z 領域の基底位置を設定する，

```
set ticslevel s（実数）
```

が重要です．s を 0 に設定すると，全領域がプロット領域として使われます．

例題 2　sample1.dat の $(x(t), y(t), z(t))$ を座標点とした軌跡を splot で描きなさい．

[解]

リスト 2　sample1s.plt

```
set size square
set xrange [0:40]
set yrange [-20:20]
set zrange [0:40]
set xlabel "X-axis"
set ylabel "Y-axis"
set zlabel "Z-axis"
set ticslevel 0
splot "sample1.dat" using 2:3:4 w l lw 2, \
     "" using 2:3:4 every 5 w p pt 7 ps 2
pause -1
```

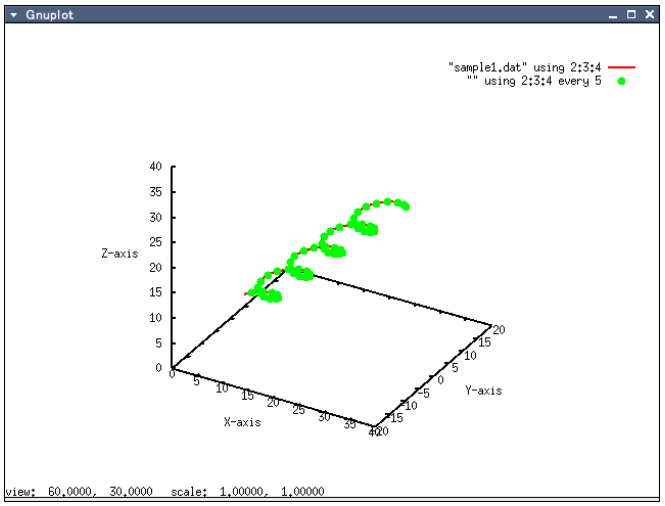


図 6　”sample1.dat”の 3 次元プロット：軌跡 (x, y, z)

1.3　パイプ接続：popen()

データファイルやスクリプトファイルを作成せずに，gnuplot の標準入力にコマンドやデータを出力して図を描いてみましょう．すなわち，親プロセスから gnuplot を起動して，その標準入力へパイプを接続するというものです．この方法を応用すれば，比較的速く画面を更新することができ，その結果，アニメーションが可能になります．

1.3.1　書き込みパイプのオープン

例題 3　書き込みパイプを形成した子プロセスとして gnuplot を起動し，$y = \sin x$ のプロットを画面表示させなさい．

[解] 以下に例を示します．

リスト 3　gnuplot01.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  #define SEC 1000000
6  #define WAIT (10*SEC)
7
8  int main(int argc, char **argv)
9  {
10   FILE *GPLT;
11
12   GPLT = popen("gnuplot", "w");
13   fprintf(GPLT, "plot sin(x)\n");
14   fflush(GPLT);
15   usleep(WAIT);
16   pclose(GPLT);
17
18   return 0;
19 }
```

10 行目：まず，普通のファイルのオープンの場合と同じ様に，ファイル構造体 FILE * GPLT(もちろん名前は任意であり，GPLT でなくても良い) を宣言します．12 行目：次に popen() によりパイプ接続された子プロセス (この場合には gnuplot) を起動します．

```
GPLT = popen("gnuplot", "w");
```

ここで注意，popen() では，書き込み専用あるいは読み込み専用モードでしかオープンできません (読み書き可能にするには，dup() 関数により，ファイルディスクリプタの複製を生成するという少し複雑な操作が必要)．13 行目：gnuplot には，fprintf() を用いてコマンド文字列を書き込みます．ただし，行の終端の印 “\n” を忘れないようにしてください．

```
fprintf(GPLT, ``command\n'')
```

続いて，gnuplot に関数 $f(x)$ を描かせます．これは大変簡単で，

```
plot f(x)
```

とします．*f(x)* には，gnuplot の組込み関数の組合せを表記することができます．また，ユーザー関数を定義して使うこともできます．

gnuplot01 は簡単なプログラムで，すぐに終了しますから，gnuplot も同時に終了してしまいます．したがって，画面表示も一瞬にして消えてしまいます．そこで，usleep() を使って，終了を遅らせます．ところで，折角 usleep を用いても，実はほとんど画面表示がされません．なぜかというと，一般に**入出力はバファリングされる**ことを思い出してください．GPLT への書き込みがバッファされてしまい，gnuplot01 の終了時に強制掃き出しされるまで gnuplot にはコマンドが送られないのです．そこで，

```
fflush(GPLT)
```

が必要となります．fflush によって gnuplot へのコマンドを即時に送出してこそ usleep が活きるのです．

1.3.2　gnuplot の標準入力：’-’

例題 4　主プログラムで計算を実行し，結果を gnuplot に送ってプロットさせてみなさい．

[解] gnuplot の特別なファイル（標準入力）「’-’」を用いた例を示します．**リスト 4**　gnuplot02.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <unistd.h>
5
6  #define DIV 200
7  #define SEC 1000000
8  #define WAIT 10*SEC
9
10 int main(int argc, char **argv)
11 {
12   int i;
13   double x, y;
14   FILE *GPLT;
15
16   GPLT = popen("gnuplot", "w");
17   fprintf(GPLT, "plot '-'\n");
18   for (i = 0; i < DIV; i++) {
19     x = i / (double)DIV;
20     y = cos(4 * M_PI * x);
21     fprintf(GPLT, "%f %f\n", x, y);
22   }
23   fprintf(GPLT, "e\n");
24   fflush(GPLT);
25   usleep(WAIT);
26   pclose(GPLT);
27
28   return 0;
29 }
```

gnuplot に標準入力から 1 行ずつデータを読ませるには，特別なファイル名「’-’」を使います．’e’ で始まる行に出会うとデータ読み込みが終了します．

```
plot '-'  ⇐ 標準入力から読み込む指定
x1  y1
...
e  ⇐ 標準入力からデータ読み込みの終了
```

Gnuplot の紹介デモの中には，簡易世界地図（大陸海岸線）のデータ world.dat が入っています．gnuplot のデモにおいては，このデータを 3 次元プロット splot を用いて（地）球面上に描くデモが展開されます．ここでは，地図帳などでよく見かける 2 次元の世界地図を描いてみることにしましょう．

例題 5　world.dat をそのまま x–y 平面に描くプログラムを作成しなさい（正距円筒図法・方眼図法）．

[解]　素直に plot 命令でデータファイル world.dat の内容を折れ線で描くように，コマンドを送ればよいだけです．

リスト 5　map_plain.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv)
{
  FILE *GPLT;

  GPLT = popen("gnuplot", "w");
  fprintf(GPLT, "set grid;\n");
  fprintf(GPLT, "plot 'world.dat' w l\n");
  fprintf(GPLT, "pause 5\n");
  fflush(GPLT);
  pclose(GPLT);

  return 0;
}
```

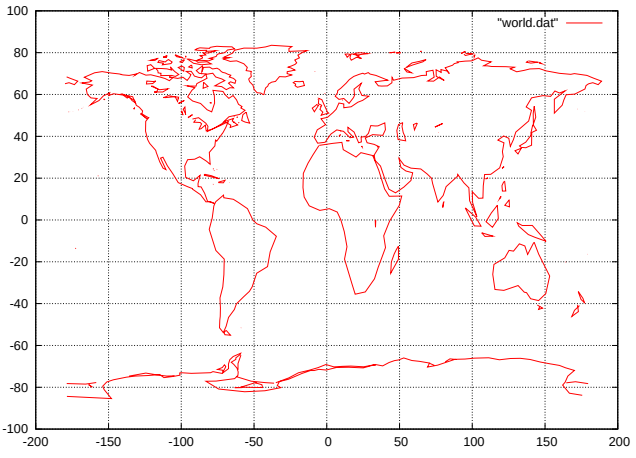


図 7　正距円筒法による世界地図

単純な図法では，緯度の高い地域が拡大されてしまいます．そこで地図帳には，用途に応じて面積（正積），2 点間距離（正距），角度（正角）などがそれぞれ正しく表示されるように工夫された地図が掲載されています．

代表的な正積図法のモルワイデ法では，経度 λ，緯度 α の地点を次のような座標 (x, y) に変換しています．

$$\begin{cases} x = 2\sqrt{2}\,\dfrac{\lambda}{\pi}\cos\theta \\ y = \sqrt{2}\sin\theta \\ \sin 2\theta + 2\theta = \pi\sin\alpha \end{cases} \tag{1}$$

例題 6　world.dat を用いて，モルワイデ法の世界地図を描いてみなさい．外形（楕円形）も描くようにしなさい．

[解]　緯度 α から非線形方程式 (1) により，θ を求めるために newton 法を組み込んでみました．

リスト 6　map_mw.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <string.h>

double newton(double c)
{
  double x = 0, dx;
  int i;

  for (i = 0; i < 20; i++) {
    dx = -(x + sin(x) - c)/(1 + cos(x));
    x += dx;
    if ( fabs(dx) <= DBL_EPSILON ) return x;
  }

  return 0;
}

void mollweide(double *x, double *y)
/* x, y in [rad] */
{
  double th ;

  th = newton(2*asin(2*(*y)/M_PI));
  *x = sqrt(8)/M_PI*(*x)*cos(th);
  *y = sqrt(2)*sin(th);
}

int main(void)
{
  double x, y;
  char buf[81];
  FILE *GPLT;

  GPLT = popen("gnuplot","w");

  fprintf(GPLT,
    "set size ratio -1; set nokey;"
    "unset border; unset xtics; unset ytics;"
    "set xrange[-sqrt(8):sqrt(8)];"
    "set parametric; set trange[0:2*pi];"
    "R=sqrt(2);\n");

  fprintf(GPLT, "plot "
    "2*R*cos(t),R*sin(t) lt 5,"
    " '-' w l lt 3 lw 1\n");
  fflush(GPLT);
  while(fgets(buf, 80, stdin) != NULL) {
    if (strlen(buf) < 2 || buf[0] == '#') {
      fprintf(GPLT, "\n");
    } else {
      sscanf(buf, "%lf %lf", &x, &y);
       x *= M_PI/180;
       y *= M_PI/180;
       mollweide(&x, &y);
      fprintf(GPLT, "%8.4f  %8.4f\n", x, y);
    }
  }
  fprintf(GPLT, "e\n");
  fprintf(GPLT, "pause 10\n");
  fflush(GPLT);
  pclose(GPLT);

```

```
66    return 0;
67  }
```

46 行目は媒介変数を用いて楕円を描いてます．媒介変数に設定する gnuplot のコマンドを，43 行目に記述しています．ついでに媒介変数 t の範囲も設定しています．

```
set parametric   ⇐ 媒介変数モードに設定
```

50 行目で標準入力から 1 行ずつ読み込んだデータを，54 行目で実数 x, y に変換し，さらに 21 行目から定義した関数 `mollweide()` に渡してモルワイデ法の座標値に変換します（55 行目）．

モルワイデ法の座標に変換する元のデータを入力リダイレクトで読み込ませるために以下のように起動します．

```
$ ./map_mw < world.dat
```

すると，図 8 のような地図が画面に表れます．

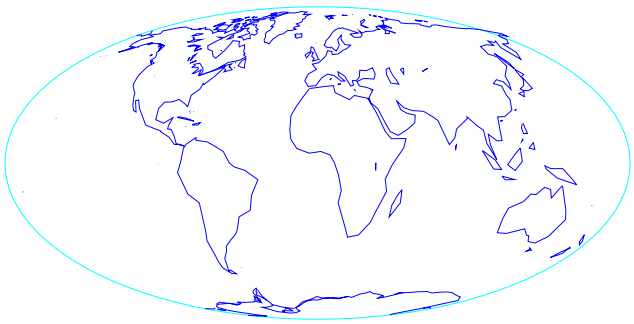


図 8　モルワイデ図法による世界地図

問題 2　モルワイデ図法による世界地図に緯度や経度を表すグリッド線が表示されるよう，修正を加えなさい．

1.3.3　gnuplot によるアニメーション

gnuplot で書き換えを素早く行わせることによって，アニメーションを実現しましょう．

例題 7　速度の 2 乗に比例する抵抗が働く方物運動の軌跡を gnuplot を用いて，アニメーション表示してみなさい．ただし，初期速度を 10 [m · s^{-1}]，投げ上げ角度を 60 度とし，抵抗係数 b_2 とステップ h を起動時に指定できるようにしなさい．なお，運動方程式は以下の通り．

$$\begin{cases} \dot{v}_x = -b_2|v|v_x \\ \dot{v}_y = -g - b_2|v|v_x \end{cases}$$

[解]　4 次のルンゲ・クッタ法による数値計算を用いた例を示します．

リスト 7　`rk4v6_3.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <unistd.h>
5  #include "rk4fix.h"
6
7  #define  Tmax    10
8  #define  G       9.8
9  #define  TH0     60.0
10 #define  V0      10
11 #define  IVAL    10000
12
13 double b2;
14
15 vec6 mov(double t, double r[6])
16 {
17   double v;
18   vec6 ret;
19
20   v = hypot(r[3],r[4]);
21   ret.f[0] = r[3];
22   ret.f[1] = r[4];
23   ret.f[2] = 0;
24   ret.f[3] = -b2*v*r[3];
25   ret.f[4] = -G - b2*v*r[4];
26   ret.f[5] = 0;
27
28   return ret;
29 }
30
31 int main(int argc, char **argv)
32 {
33   double r[6] = {0,0,0,0,0,0};
34   double y, v, t = 0, h;
35   FILE *GPLT;
36
37   if (argc < 3) {
38     printf("Usage: %s b2 h\n", argv[0]);
39     exit(0);
40   }
41   b2 = atof(argv[1]);
42   h = atof(argv[2]);
43   r[3]=V0*cos(TH0/180*M_PI);
44   r[4]=V0*sin(TH0/180*M_PI);
45
46   GPLT = popen("gnuplot","w");
47   fprintf(GPLT,"set xrange [0:4]\n");
48   fprintf(GPLT,"set yrange [0:3]\n");
49   while (t < Tmax){
50     rk4fixv6(mov, t, r, h);
51     t += h;
52     fprintf(GPLT, "plot '-' pt 7 ps 3\n");
53     fprintf(GPLT, "%f %f\ne\n", r[0], r[1]);
54     fflush(GPLT);
55     if (r[1] < 0) break;
56     usleep(IVAL);
57   }
58   pclose(GPLT);
59
60   return 0;
61 }
```

41,42 行で b_2, h を起動オプションから取り込んでます．43,44 行は角度を radian に変換しています．以下のようにコンパイルします．

```
$ gcc -o rk4v6_3 rk4v6_3.c rk4fix.o -lm
-Wall
```

実行時には，小さい h を指定してみましょう．赤い丸が少し歪んだ放物線を描いて飛行するアニメーションが表示されるはずです．

```
$ ./rk4v6_3 0.2 0.01
```

問題 3　前記の例題を 3 次元空間でアニメーション表示させてみなさい．

1.3.4　リングバッファ

gnuplot を用いて，オシロスコープのような機能を実現してみましょう．連続した（時系列）データを受け取り，範囲を少しずつずらしながら描画させれば，オシロスコープでデータを表示している画面のように見えます．数値シミュレーションが連続データを発生させる速度は，一般に gnuplot が 1 画面を描画する速度よりも速く，途中にデータをため込む機構，すなわち**バッファリング**が必要です．時系列データをメモリにすべて保存することは大きさに制限があって不可能な場合が多いです．むしろ逆に，バッファに用いるメモリ領域をある大きさに制限する手だてを工夫しなければなりません．そこで，いわゆる**リングバッファ**を用いてみましょう．

例題 8　標準入力から連続して x, y データを受け取り，それをオシロスコープのように少しずつ範囲をずらしながら gnuplot で表示させるプログラムを作成しなさい．

[解] gnuplot で表示させるデータ点の数と同じ大きさのリングバッファを用いた例を示します．

リスト 8　gplt_record_ring.c

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

#define BUFFSIZE 384

int main(int argc, char *argv[])
{
  int i, im, seek=0;
  double x, y;
  double data[BUFFSIZE][2]={{0,0}};
  FILE *GPLT;

  GPLT = popen("gnuplot -geometry 400x300","w");
  setlinebuf(GPLT);
  fprintf(GPLT,
    "set noxtics; set yrange[-5:5];"
    "set nokey;\n");

  while(1){
    fscanf(stdin,"%lf %lf", &x, &y);
    seek %= BUFFSIZE;
    data[seek][0]=x;
    data[seek][1]=y;
    fprintf(GPLT, "plot '-' w l lt 1 lw 2\n");
    for (i = 1; i < BUFFSIZE; i++){
      im = (i + seek) % BUFFSIZE;
      fprintf(
        GPLT, "%f %f\n",data[im][0],data[im][1]);
    }
    fprintf(GPLT,"e\n");
    seek++;
  }
  fclose(GPLT);

  return 0;
}
```

seek が新しいデータを読んで保存する位置を表します（図 9）到着とともに 1 ずつ大きくなっていきますが，剰余演算

```
seek++
seek = seek % BUFFSIZE
```

により，0 から BUFFSIZE-1 の間に制限することができます．

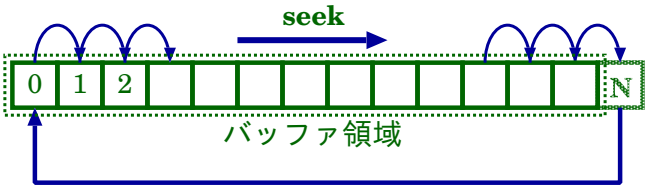


図 9　リングバッファの概念図

例題 9　**【カオス振動：duffing 方程式】**　変位 x の 3 乗に比例する復元力を考慮した，duffing 方程式は，カオス振動を生ずる系として知られています．すなわち，x^3 に比例する復元力と速度に比例する抵抗力が働く振動系に外力を加えると，一定の周期解に収斂しない振動が生ずる場合があり．系は以下の常微分方程式で表されます．

$$\ddot{x} + a\dot{x} + bx^3 = f\cos(\omega t) \qquad (2)$$

この振動状態を数値シミュレーションして，時系列データ $t, x(t)$ を発生させ，gplt_record_ring にパイプで接続して振動の経時変化を表示させなさい．

[解]　以下に 4 次の Runge-Kutta 法 rk4fixv6 を用いた例を示します．**リスト 9**　duffing.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>
#include "rk4fix.h"

double A = 0.05, B = 1.0, F = 7.5;

vec6 duffing(double t, double r[6])
{
  vec6 ret;

  ret.f[0] = r[3];
  ret.f[3] = -A*r[3] - B*pow(r[0],3) + F*cos(t);
  return ret;
}

int main(int argc, char **argv)
{
  double r[6] = {0};
  double t = 0, h;

  if (argc < 3) {
    printf("%s F h\n", argv[0]);
    exit(0);
  }
  F = atof(argv[1]);
  h = atof(argv[2]);

  while(1) {
    rk4fixv6(duffing, t, r ,h);
    t += h;
    printf("%f %f\n", t, r[0]);
```

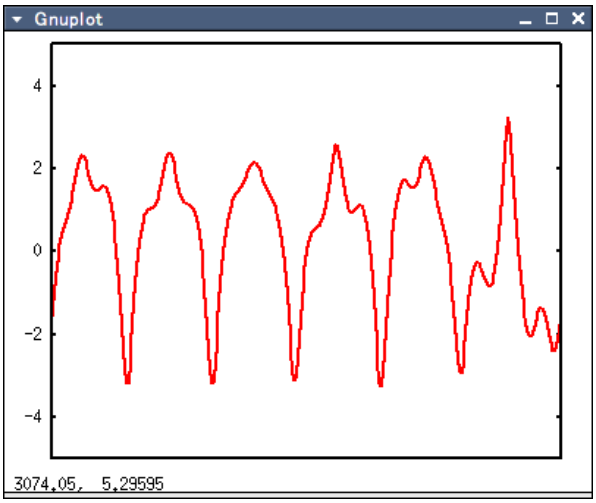
```
34      fflush(stdout);
35    }
36
37    return 0;
38  }
```

このソースでは，パラメータを $a = 0.05$, $b = 1.0$, $\omega = 1$ と固定し，強制振動の振幅 F とステップ h を起動時のオプションで指定する仕様となっています．コンパイル時に rk4fix.o をリンクするのを忘れないでください．

```
$ gcc -o duffing duffing.c rk4fix.o -lm
-Wall
```

コンパイルがうまくいったら，以下のように実行させます．

```
$ ./duffing 7.5 0.1 |./gplt_record_ring
```



duffing は無限ループを形成してますので，終了させるには Ctrl + C とキー入力して強制終了させます．