数理解析研究所講究録
第1674巻 2010年 89-98

PostScript を用いた数学関係の図の作成について

神奈川工科大学 情報学部情報メディア学科 平野 照比古
Department of Information Media
Kanagawa Institute of Technology

1 はじめに

数学の教材を作成するときには数式の記述が楽なソフトウェアが必要である。著者はこのような教材は LaTeX[3] を用いて作成し、結果を pdf 形式のファイルにしている。出力結果を PDF 形式にするため LaTeX で書かれた文書内の図は PostScript で記述している。こうすることで出版社に渡す書籍は図も含めた単一の pdf ファイルとすることが出来た ([6])。また、授業で配布する資料も授業時には解答がないものを配布し、解答がついたものは後日 WEB 上で公開するということも容易に出来るようになった ([7] 参照)。

ここでは関数のグラフを多数 PDF ファイルに取り込むために採用した方法について簡単に解説をする。また、より正確な図を描くために数式処理を用いる方法についても述べる。

2 PostScript とは

PostScript は Adobe Systems が開発したプログラミング言語で、言語仕様の中にグラフィックスに関するものが含まれていることが特徴である [1]。名称の由来はプログラムの記述が逆ポーランド記法を用いていることである。PostScript の入門書としては [2] がある。

PostScript では通常の言語における関数は operator とよばれる。operator はスタック上にあるデータをいくつか引数として使い、結果をスタックの上に置く。

2.1 スタックの処理と変数

PostScript では数字などの単純なものを記述するとそれはスタック上に順次置かれる。置かれた順序を入れ替えたりするための基本的な operator は次の通りである。

Operator 名	処理後のスタック	意味
OP_1 dup	OP_1 OP_1	スタックの一番上のデータをコピーする
OP_1 OP_2 exch	OP_2 OP_1	スタックの一番上とその次のデータを入れ替える
OP_1 pop	-	スタックの一番のデータを取り除く
OP_1 OP_2 def	-	OP_1をOP_2で置き換える。OP_1は/で始まる文字列であり、これが変数を定義することになる。OP_2を{}でくくっておくとそれらが一つのものとして扱われ実質的に関数が定義できる

2.2 演算

PostScript では四則演算や関数も引数を先に記述する。なお、PostScript で扱う数はすべて実数である。

Operator 名	処理後のスタック	意味
OP_1 OP_2 add	$OP_1 + OP_2$	OP_1と OP_2を加える
OP_1 OP_2 sub	$OP_1 - OP_2$	$OP_1 - OP_2$ を計算する
OP_1 OP_2 mul	$OP_1 OP_2$	OP_1と OP_2の積を計算する
OP_1 OP_2 div	OP_1/OP_2	OP_1/OP_2 を計算する。整数同士であっても演算は実数として計算される
OP_1 neg	$-OP_1$	OP_1の符号を変えた数を計算する

このほかにも sin や cos の三角関数、平方根をとる sqrt など通常必要な関数はそろっている。詳しくは [1] を参照のこと。

2.3 論理演算と制御構造

PostScript には通常のプログラミング言語にある論理演算と制御構造が用意されている。論理演算は一部の制御構造で用いられる。

論理演算は次の通りである。

Operator 名	処理後のスタック
OP_1 OP_2 eq	$OP_1 = OP_2$ のとき true それ以外は false
OP_1 OP_2 ne	$OP_1 \neq OP_2$ のとき true それ以外は false
OP_1 OP_2 gt	$OP_1 > OP_2$ のとき true それ以外は false
OP_1 OP_2 ge	$OP_1 \geqq OP_2$ のとき true それ以外は false
OP_1 OP_2 lt	$OP_1 < OP_2$ のとき true それ以外は false
OP_1 OP_2 le	$OP_1 \leqq OP_2$ のとき true それ以外は false

制御構造は次の通りである。

Operator 名	意味
OP_1 OP_2 `if`	OP_1が`true`のときはOP_2が実行される
OP_1 OP_2 OP_3 `ifelse`	OP_1が`true`ときはOP_2がそれ以外はOP_3が実行される
OP_1 OP_2 `repeat`	OP_1の回数だけOP_2が実行される
OP_1 `loop`	OP_1を繰り返す。中断するためには`exit`を用いる
OP_1 OP_2 OP_3 OP_4 `for`	OP_1を初期値としOP_2が正のときはOP_2ずつ値を増加し、OP_3の値を以下の間OP_4が実行される。OP_2が負のときはOP_3以上の間OP_4が実行される。 なお、途中で生成される数は毎回スタックの上に設定される。

2.4 PostScript におけるグラフィックスの特徴

PostScript におけるグラフィックスの特徴は次の通りである。

- 初期状態の座標系
  - 左下が原点 $(0,0)$ である。
  - 水平方向が x 軸で右に行くほど値が大きい。
  - 垂直方向が y 軸で上に行くほど値が大きい。
  - 座標系の単位は $\frac{1}{72}$ インチである。

  これらの座標系は通常の数学で用いられている座標系と同じである。
- 原点を移動 (`translate`) したり、原点を中心に回転 (`rotate`)、拡大·縮小 (`scale`) の operator がある。
- 回転の角度の単位は 60 分法であり、弧度法ではない。三角関数の引数も同様である。
- ある時期のグラフィックスの環境を保存し (`gsave`)、その後、復元する (`grestore`) operator がある。
- 図形を描くためには道のり (path) を定義し、道のりを描く (`stroke`) か、それによって囲まれた部分を塗る (`fill`)。これらの operator により道のりのデータは消える。

Operator 名	意味
OP_1 OP_2 translate	現在の座標系での (OP_1, OP_2) の位置に座標系の原点を平行移動する
OP_1 rotate	現在の座標系の原点にして $OP_1°$ 回転する
OP_1 OP_2 scale	現在の座標系での x 軸方向に OP_1, y 軸方向に OP_2 拡大・縮小する
OP_1 setlinewidth	stroke での線幅を現在の単位における OP_1 の値に設定する
OP_1 setgray	stroke や fill で塗りつぶす色を設定する。OP_1が0のときは黒、OP_1が0のときは白となる。
OP_1 OP_2 OP_3 setrgbcolor	stroke や fill で塗りつぶす色を OP_1(赤)、OP_2(緑)、OP_3(青) に設定する。値の範囲は 0 ～ 1

3 PostScript を利用した教材の開発

数学関係の教科書で図を扱う場合には次のような点に注意する必要がある。

- 教科書には多数の関数のグラフがある。
- これらのグラフは同じような体裁で表す必要がある。
- したがって、Postscript で図を作成するためには PostScript 用のライブラリーを用意する必要がある。
- PostScript のなかに記述された他のファイルを呼び出す (foo.ps) run は PDF フォーマットの中では正しく動かない。
- したがって、個々のファイル内にライブラリー関数を埋め込む必要がある。

[6] では出版社から提供されたスタイルファイルを元に原稿の入力は PDF ファイルで行った。関数のグラフを記述するに当たり、ライブラリーの関数を埋め込むために次のような処理を行った。

- 関数のグラフは関数の値だけを計算する部分と、グラフを表示する範囲を指定する値程度の指定だけで済ませるようにし、体裁の部分はライブラリーを読み込むこととした。
- EPS 形式に直すのには Ghostview を用いた。
- ライブラリーファイルを変換後の EPS ファイルに埋め込むために PHP([5]) のプログラムを作成した。
- グラフ上に現れる原点の記号などは作成した図の上に載せている。このために picture 環境を用いている。

次のリストはライブラリーファイルを埋め込むために作成した PHP ファイルである。

```
1  <?php
2     $dname = "./ps/";
3     $tmpfname = "tmp000.eps";
4     $FILENAME= opendir($dname);
5     while($fname = readdir($FILENAME)) {
6        if(ereg('.eps$',$fname)) {
7           $fp =fopen("$dname$fname", "r");
8           $ftmp =fopen("$tmpfname", "w");
9           echo "$dname$fname\n";
10          while($line = fgets($fp,1024)) {
11             if(ereg('^\%',$line)) {
12                fputs($ftmp,$line);
13             } else {
14                if(ereg('\((.*)\).*run',$line,$incfname)) {
15                   $fincp = fopen("$dname$incfname[1]","r");
16                   fputs($ftmp, "%%\n%%$line%%\n");
17                   while($line = fgets($fincp,1024)) {
18                      fputs($ftmp,$line);
19                   }
20                   fputs($ftmp, "%%\n%% end of $incfname[1]\n%%\n");
21                   fclose($fincp);
22                } else
23                   fputs($ftmp,$line);
24             }
25          }
26          fclose($ftmp);
27          fclose($fp);
28          copy($tmpfname, "$dname$fname");
29          unlink($tmpfname);
30       }
31    }
32    closedir($FILENAME);
33 ?>
```

- 今回の教科書では PostScript ファイルはすべて ps の下に置いた。3行目でその値を変数$dname に格納している。
- 4行目ではそのフォルダを開いている。
- PHP の関数 readdir() は opendir() で開いたフォルダのファイルを順番に示す

関数である。すべて提示したときには `false` を返すので `while` でループを回すことができる (6 行目)。

- 与えられたファイル名が `.eps` で終わっていれば 8 行目から 31 行目までの処理をする (7 行目)。
- 12 行目では読み込んだ行が `%` で始まっていればそのまま出力し (13 行目)、そうでない場合には `(<filename>) run` の行であるかをチェックする (15 行目)。
- この形式の場合にはこの行の先頭に%を追加した行を出力し (17 行目)、その後、ライブラリーファイルを読み込んでそのまま出力する (18 行目から 20 行目)。
- その後、ライブラリーファイルの終了位置を示す行を書き込む (21 行目)。
- これらの処理は一時ファイルに書き込んでいるので (4 行目、9 行目参照)、最後に一時ファイルを元のファイルに上書きコピーし (29 行目)、一時ファイルを消去する (30 行目)。

図 1 は [6] にある $\dfrac{x+2}{x^2-1}$ のグラフである。

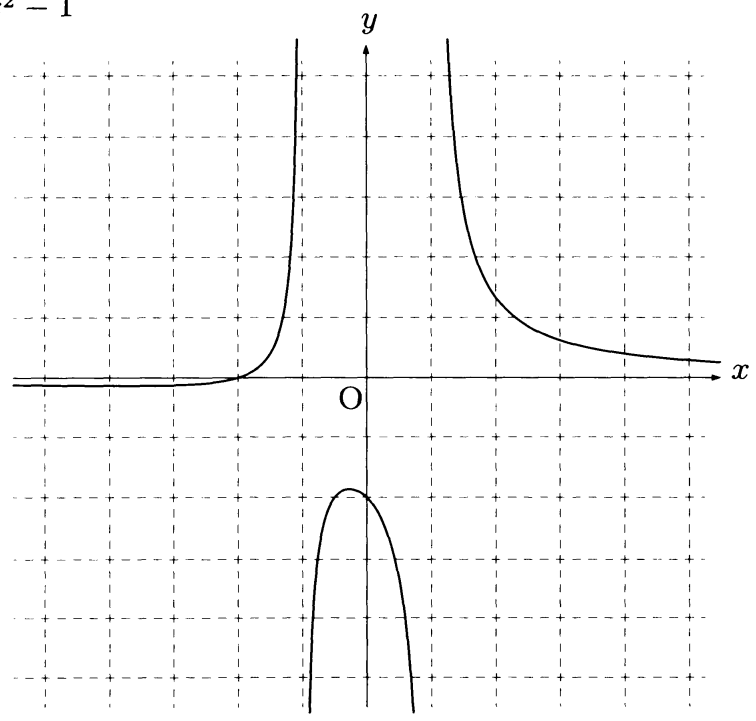

図 1: $\dfrac{x+2}{x^2-1}$ のグラフ

このグラフは次のように作成した。

- グラフは x の値を小さく変化させながら関数の値を計算し、それらの間を線分で結ぶことで描いている。なお、浮動小数点の誤差を避けるため x の値の変化はすべて整数で計算し、最後に 10 や 100 で割ることで浮動小数点に直している。

- 関数の値はすべて PostScript プログラムで計算してある。
- 関数の特異点の部分は手動で計算しないようにしてある。
- 関数のグラフに書き入れる文字は次のような LaTeX のマクロを作成した。

```
\newcommand{\ShowGraph}[7]{%
\setlength{\unitlength}{#1cm}%
\begin{picture}#2#3
 \put#3{\scalebox{#1}{\includegraphics{#7}}}
 \put(#4,#4){0}
 \put(#5,0){$x$}
 \put(0,#6){\makebox[0em]{$y$}}
\end{picture}}
```

この関数は $x < -2$ のところに極小値をもつが、その値が小さいため正確な位置がわからないことが見て取れる。

4 数式処理を用いた教材の例

前節の例では関数が定義できない点を手動で与えていたが、このような有理関数であれば与えられた関数を分子と分母にわけ、分母が 0 になる点は計算しないという方法で計算結果を PostScript のプログラムで出すことが可能である。

このほかに数式処理を用いて関数の値を計算する例としてはけた落ちを意識しないでいつも正しい関数の近似値が得られるという利点がある例を示す。

$\sin x$ のテイラー展開では原点のあたりでの振る舞いが強調された話が多い。学生に対して与えられたテイラー展開が見慣れた $\sin x$ のグラフになることを示すためには 1 周期や 2 周期にわたってマクローリン展開による式の結果が見慣れた形になることを示すのが良いのではないだろうか。次のような単純な C 言語のプログラムではこの要請にこたえられないことがわかる。

```
#include <stdio.h>
double func(double x, int deg);
double func2(double x, int deg);

int main(void) {
  int i, deg = 45, scale = 10, xL = -19, xH, xLS, xHS;
  double x;

  xH = -xL;
  xLS = xL*scale;
  xHS = xH*scale;
```

```
  for(i=xLS;i<=xHS;i++) {
    x = (i*1.0)/scale;
    printf("%f %f %f\n",  x, func(x, deg),  func2(x, deg));
  }
}

double func(double x, int deg) {
  double sin, px;
  int j;
  sin = x;
  px = x;
  for(j=3;j<=deg; j+=2) {
    px *= -x*x/(j*(j-1));
    sin += px;
  }
  return sin;
}
double func2(double x, int deg) {
  double sin, px;
  int j;
  if((deg & 1) ==0) deg--;
  sin = 1;
  px = x*x;
  for(j=deg;j>2; j-=2) {
    sin = 1-px/(j*(j-1))*sin;
  }
  return sin * x;
}
```

この出力の一部は次のようになった。

```
18.600000 1.317148 1.317148
18.700000 1.860588 1.860588
18.800000 2.528902 2.528902
18.900000 3.354320 3.354320
19.000000 4.377756 4.377756
```

このようにかなり大きな次数で計算した (この場合には 45 次の項) にもかかわらず $\sin x$ の値が常識の範囲を超えている。計算の次数を下げるともっと前から値は正しくない。これは桁落ちによるものであり、浮動小数点で計算している限りでは避けられないものである。数式処理で x の値を有理数で与えておけば関数の結果も有理数で求められ、桁落ちを心配することはない。最終的に 1.0 をかけて浮動小数点として結果を出力させればよい。

次のプログラムは Risa/Asir による例である。

```
1  def func(Deg,Scale,R) {
2    output("sin-asir")$
3    print("72 2.54 div 3 div dup scale")$
4    print("0.04 setlinewidth")$
5    print(R,0)$
6    print(" 2 add 3 translate")$
7
8    print(R,1)$
9    print(" 0.5 add dup neg 0 moveto 0 lineto stroke")$
10   print("0.06 setlinewidth")$
11   C =" moveto"$
12   for(I=-R*Scale;I<=R*Scale;I++) {
13     X = I/Scale$
14     S = X$
15     Px = X$
16     for(J=3;J<=Deg;J+=2) {
17       Px *= -X*X/(J*(J-1))$
18       S += Px$
19     }
20     print(X*1.0,0)$
21     print(" ",0)$
22     print(S*1.0,0)$
23     print(C)$
24     C = " lineto"$
25   }
26   print("stroke")$
27   output()$
28 }$
29 end$$
```

- 3行目は図形の大きさの単位を 1cm に設定するための PostScript のコマンドを出力している。
- 4行目から 9行目ではグラフの軸を描くための PostScript のコマンドを出力している。
- 12行目から 25行目で $\sin x$ の値を有理数で計算した後 (14行目から 20 行目)、値を有理数に直し (20行目と 22行目)、PostScript のコマンドをその後に付け加えている。

なお、出力ファイルのサイズを小さくする書き方としては repeat を用いる方法もあるがここではわかりやすい方法をえらんである。

出力ファイルを原稿に取り込むことでより正確なグラフが容易にかける。

参考文献

[1] Adobe Systems Incorporated, *PostScript Language Reference Manual third edition*, http://www.adobe.com/devnet/postscript/pdfs/PLRM.pdf

[2] Adobe Systems Incorporated, Adobe Systems Incorporated, *PostScript(R) Language Tutorial and Cookbook*, http://www-cdf.fnal.gov/offline/PostScript/BLUEBOOK.PDF

[3] H. Kopka and P. W. Daly, *Guide to LaTeX Fourth Edition*, Addison-Wesley, 2004

[4] F. Mittelbach and M. Goossens, *The LaTeX Companion Second Edition*,Addison-Wesley, 2004

[5] 日本 PHP ユーザー会, http://www.php.gr.jp/

[6] 平野照比古，工科のための微分積分学第 3 版，学術図書出版，2009 年

[7] 平野研究室のホームページ http://www.hilano.org/hilano-lab