Plot Tips on PSTricks

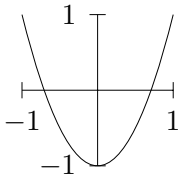
山田 泰司<taiji@aihara.co.jp>

2002年10月7日

この文書は、PSTricksのプロット関係のマクロの使い方を調べたものであり、この文書のTeXソースファイルと一緒に読まれることを前提に書いてあります。

PSTricksの曲線とPostScriptのBézier 3次曲線

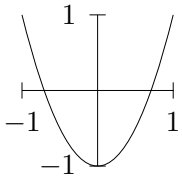
- 放物線 `\parabola`




```
\parabola(-1,1)(0,-1)
```

素のPostScript言語には、放物線を直接描く命令は無い。PSTricksには`\parabola`マクロが提供される。

- Bézier 3次曲線 `\psbezier`




```
\psbezier(-1,1)(-0.33333,-1.66667)(0.33333,-1.66667)(1,1)
```

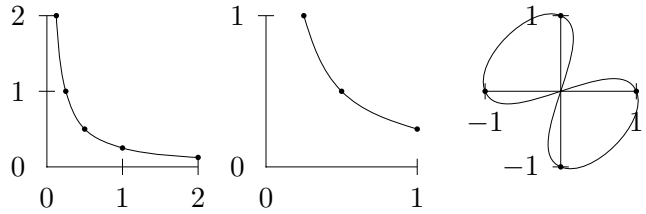
PostScript言語で提供される曲線の描画命令は、Bézier 3次曲線/`curveto`オペレータであり、これはPSTricksの`\psbezier`マクロに対応している。

実際、PSTricksの放物線はBézier 3次曲線によって描かれている。つまり、PostScript言語では上図のどちらも、

```
-1 1 moveto -1 3 div -2 3 div 1 3 div -2 3 div 1 1 curveto stroke
```

のように、始点、Bézier 3 次制御点 (2 点)、終点、/curveto オペレータで描画 (/stroke) されている。

- 補間曲線 \pscurve, \psecurve, \psccurve



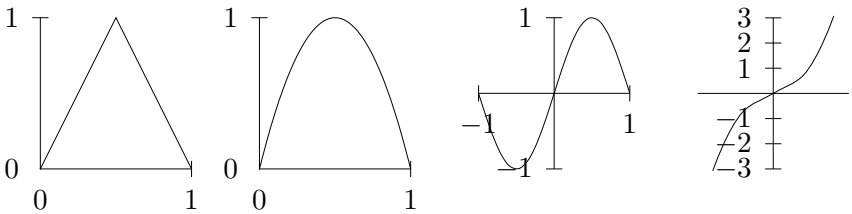

```
\pscurve(.125,2)(.25,1)(.5,.5)(1,.25)(2,.125)
\psecurve(.125,2)(.25,1)(.5,.5)(1,.25)(2,.125)
\psccurve(1,0)(-1,0)(0,1)(0,-1)
```

上図の曲線は、Bézier 3次曲線を基本とした補間曲線である。実際これらは、PostScript 言語で、いくつもの、始点、Bézier 3 次制御点、終点、/curveto オペレータの組み合わせで描画されている。

\psecurve の場合は、始点・終点は実際には描かれない。\psccurve の場合は、始点・終点が閉じた曲線を描く場合に使われる。

PSTricks の関数描画と PostScript の算術オペレータ

- XY プロット \psplot




```
\psplot[plotstyle=line]{0}{1}{-2 x .5 sub abs mul 1 add}
\psplot[plotstyle=curve]{0}{1}{4 x mul 1 x sub mul}
\psplot[plotstyle=curve]{-1}{1}{x 180 mul sin}
\psplot[plotstyle=curve]{-.4}{-4}{x 180 mul tan}
```

PSTricks には、PostScript 言語を使ってデータ点列を生成しプロットするマクロがサポートされており、PostScript の算術オペレータを使えば初等関数であれば簡単に直線補間もしくは曲線補間にて描画することが出来る。上図は順に、

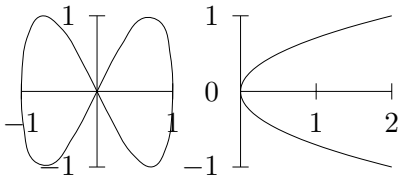
$$
\begin{aligned}
-2|x-0.5|+1 &\Leftrightarrow \texttt{-2 x .5 sub abs mul 1 add}\\
4x(1-x) &\Leftrightarrow \texttt{4 x mul 1 x sub mul}\\
\sin(180x) &\Leftrightarrow \texttt{x 180 mul sin}\\
\tan(180x) &\Leftrightarrow \texttt{x 180 mul tan}
\end{aligned}
$$

である。但し、/tanはPostScriptのオペレータには無いのでマクロ\pscustom内で\codeマクロを使って、独自のPostScript手続き

```
/tan { dup sin exch cos div } bind def
```

を作成し演算をしているので、詳しくはソースを見て頂きたい。

- 媒介変数プロット \parametricplot




```
\parametricplot[plotstyle=ccurve]{0}{360}{t sin t 2 mul sin}
\parametricplot[plotstyle=curve]{-1}{1}{t t mul a mul t b mul}
```

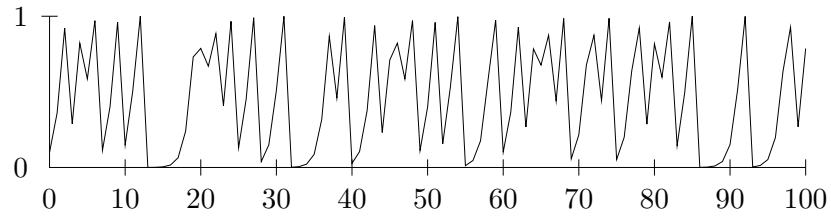
上図は順に、

$$(\sin(t), \sin(2t)) \quad \Leftrightarrow \quad \texttt{t sin 2 t mul sin}$$
$$(at^2, bt) \text{ ここで } a = 2, b = 1 \quad \Leftrightarrow \quad \texttt{t t mul a mul t b mul}$$

である。実際には、定数a, bはマクロ\pscustom内で\codeマクロを使って、一時的なPostScript辞書に定義しているので、詳しくはソースを見て頂きたい。

\psplot、\parametricplotのオプションにてplotstyle=curveまたはccurveとした場合、PostScript言語のBézier 3次曲線を基本とした曲線補間で描画される。plotstyle=ccurveの場合は、始点・終点が閉じた曲線を描く場合に使われる。

- \psplot で離散時間力学系 (Logistic 写像) の時系列を描画



PostScript による算術演算コードを\code マクロ内で定義することによって、上図のような PostScript 言語に演算させた時系列データも描くことが出来る。ここで用いられている Logistic 写像の PostScript コードは以下の通り。

$f(x) = ax(1-x)$ ここで、$a = 4$
⇕

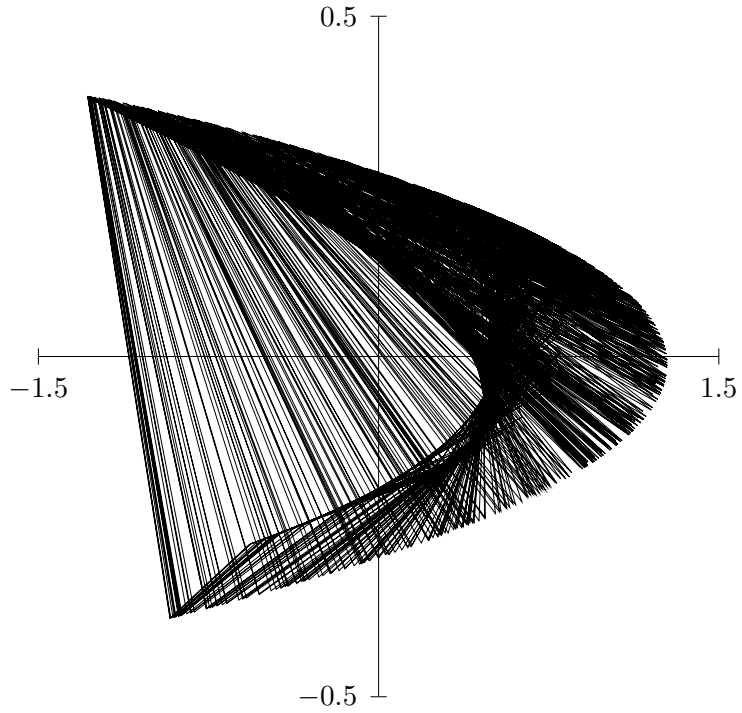
```
/tmp <<
  /a 4                              % パラメータ a=4
  /x .1                             % 状態変数 x と初期値
  /f {                              % ロジスティック写像 f
    a x mul 1 x sub mul
    /x exch def
  } bind
>> def
```

そして、以下のように PSTricks でプロットさせる。

```
\psplot[plotpoints=100]{0}{99}{tmp begin x f end}
```

ここで注意したいのは、例えば\psplot{-1}{1}{x 180 mul sin}のように用いられている時の x と、上例での x とは、まったくの無関係ということである。\psplot の仕組みを鑑みればわかるように、その第 3 引数で指定する PostScript コードは、ひとつの x 軸の値に対する y 軸の値をスタックトップに置くことがその役目であり、\psplot{-1}{1}{x 180 mul sin}の例では PSTricks が指定する x 軸の値を使っての y 軸の値を求めているのに対して、上例では PSTricks が指定する x 軸の値は使っておらず、PostScript 辞書 tmp 内の x の値をスタックトップに置いているだけである。そして、手続き /f が呼ばれるたびにその値は更新される。

- \parametricplot で離散時間力学系 (Hénon 写像) のアトラクタを描画



PostScript による算術演算コードを\code マクロ内で定義することによって、上図のような PostScript 言語に演算させたアトラクタも描くことが出来る。ここで用いられている Hénon 写像の PostScript コードは以下の通り。

$$f(x,y) = \begin{cases} y + 1 - ax^2 \\ bx \end{cases} \quad \text{ここで、} a = 1.4, b = 0.3$$

⇕

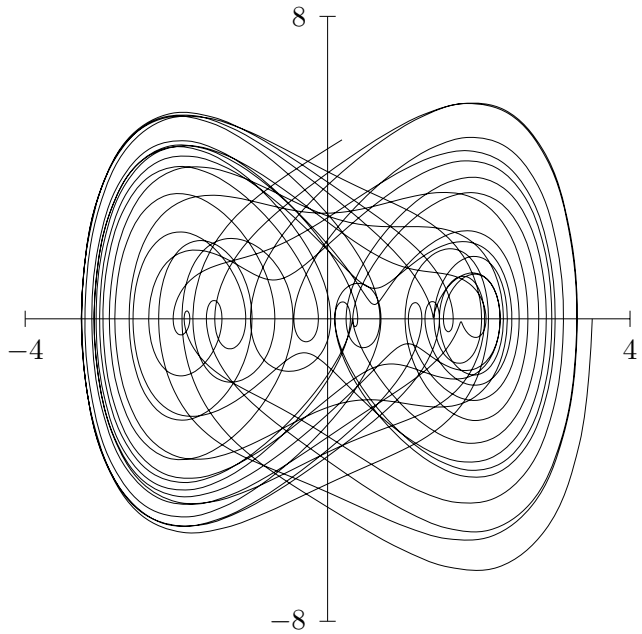
```
/tmp <<
  /a 1.4                          % パラメータ a=1.4
  /b .3                           % パラメータ b=0.3
  /x .1                           % 状態変数 x と初期値
  /y .1                           % 状態変数 y と初期値
  /f {                            % H\'enon 写像 f
    y 1 add a x mul x mul sub
    b x mul
    /x 3 -1 roll def
    /y exch def
  } bind
>> def
```

そして、以下のように PSTricks でプロットさせる。

```
\parametricplot[plotpoints=1000]{0}{999}{tmp begin x y f end}
```

これは、先の Logistic 写像の時系列データのプロットの例を、2 次元写像に拡張し、かつ、2 次元空間上の任意の点をプロットできる`\parametricplot`を使用した例である。これもまた、スタックトップに PostScript 辞書 `tmp` 内の `x` の値と `y` を置いているだけである。そして、手続き `/f` が呼ばれるたびにそれらの値は更新される。

- `\parametricplot` で連続時間力学系 (duffing 方程式) のアトラクタを描画



PostScript による算術演算コードを`\code`マクロ内で駆使することによって、上図のような PostScript 言語に演算させた連続時間力学系のアトラクタも描くことが出来る。ここで用いられている duffing 方程式の PostScript コードは以下の通り。

$$\frac{df(x,y)}{dt} = \begin{cases} y \\ -ky - x^3 + B\cos\omega t \end{cases} \quad \text{ここで、} k = 0.05, B = 7.5, \omega = 180/\pi$$

$$\Updownarrow$$

```
/tmp <<
  /k .05                                    % パラメータ k=0.05
  /B 7.5                                    % パラメータ B=7.5
  /omega 180 3.14159265358979323846 div % パラメータ omega=180/pi
  /df { % [] t df []
    4 dict begin
      /t exch def
      /X exch def
      /x X 0 get def
      /y X 1 get def
      [                                     % duffing 方程式
        y
```

```
        k neg y mul x x mul x mul sub B omega t mul cos mul add
      ]
    end
  } bind
  /X [ 3.5 0 ]                          % 初期値
  /t 0                                  % 時刻
  /h .1                                 % ルンゲ・クッタの刻み幅
  /f {
    X t h /df rk4
    /X exch def
    /t t h add def
  } bind
>> def
```

そして、以下のようにPSTricksでプロットさせる。

```
\parametricplot[plotpoints=1000]{0}{999}{tmp begin X cvx exec f end}
```

これもまた、スタックトップにPostScript辞書`tmp`内の配列`X`の0番目の値と1番目の値を置いているだけである。そして、手続き`/f`が呼ばれるたびにそれらの値は更新されるのだが、実際、その更新には、ここで示していない手続き`/rk4`が使われている。これが、4次のルンゲ・クッタ法の実装したものであるので、詳しくはソースコードを御覧頂きたい。無論、他のプログラムで予め数値計算して生成したデータファイルを用意して、`\fileplot`マクロや`\dataplot`マクロでプロットするのが普通であるが、上述の例ぐらいのことはPostScript言語で可能であるという例としてあげてある。

参考文献

[1] Timothy Van Zandt, "PSTricks: PostScript macros for Generic TeX — User's Guide," Version 0.93a, `http://www.tug.org/applications/PSTricks/`, March 1993.

[2] Adobe Systems Incorporated, "PostScript LANGUAGE REFERENCE third edition," Addison-Wesley Publishing Company, `http://partners.adobe.com/asn/developer/pdfs/tn/PLRM.pdf`

[3] 山田 泰司, "付録 A. C 言語による力学系のシミュレーション," in "カオス学入門," 合原 一幸, 放送大学教育振興協会, March 2001.