実践論文

データ描画用小型ソフトウェアの開発

Development of Small Graphic Software for Data Analysis

中村　健治[*1]
Kenji Nakamura

e-mail:k13002@dokkyo.ac.jp

データの解析結果を図示化する小型のソフトウェアの開発について述べる。本ソフトウェアはUNIX上でC言語で書かれており postscript 言語を使って postscript ファイルを出力する。プログラム群はヘッダーファイルに入っている。このためヘッダーファイルをコピーするだけで他の PC への移植ができる。これは大きな利点となっている。表示は、時系列表示、散布図表示、また分布状況のカラー画像表示が可能である。カラーのパレットの変更、地図の付加、なども可能である。不特定多数の人が使うことを想定した大型のソフトウェアとは異なり個人用として開発されており、nesting が浅い、など個人の嗜好に沿ったものとなっている。このような個人用描画ソフトウェアの意義は、様々な描画ソフトウェアが存在する現在は小さいとは思われる。しかし、大型で高機能のソフトウェアにより処理が定型化されつつあるなかで、個人の嗜好に沿った小型のソフトウェアは新たなシーズ開拓の一端となることが期待される。

A small graphic software is developed using the postscript language. The program is written in C language and works on a UNIX system. Since whole the source code is included in a header file, it is very easy to transfer the program to other machines by simply copying the header file. The software has capabilities to show sequential data, scattergrams, color images of distribution of some quantities, and others. The color palette can be changed. The world map can also be added. The software is developed for personal use, and the structure is strongly dependent on the preferences of the user. Nowadays, powerful graphic software is available, and the presentations tend to be in a fixed pattern. This kind of small software could work to help to provide new seeds for new presentation of data.

*1: 獨協大学 経済学部国際環境経済学科

１．はじめに

現在、データ処理の結果をグラフ化あるいは描画するソフトウェアは沢山ある。身近なところではマイクロソフトのエクセルがある。エクセルもPCの普及と高性能化により、非常に大きなファイルを扱うことができるようになり、図示化の機能も充実している。

Gnuplotは一般によく使われている。関数のグラフなども簡単に作成することができる。他のソフトウェアでデータを整えてからgnuplotで図示化することもよく行われる。例えばフリーの数式処理ソフトウェアであるMAXIMAではグラフの標準出力として使われている。使い方は対話的である。

データ解析においては、データを多面的に見るためのツールがいろいろと準備されている。IDL（Interactive Data Language）は市販のソフトウェアであり数値計算結果や衛星データなどの大容量のベクトルデータの可視化にかなり広く使われているソフトウェアである（日本語の説明websiteは例えばhttp://www.kwasan.kyoto-u.ac.jp/~nishida/idl/）。

私は1990年頃にMcIntosh 上で動くSpyglass Dicerという商用のデータ画像表示ソフトウェアを一時使い、３次元画像の視点の変更、断面の表示機能、色変更機能に驚いた。このソフトウェアは現在もSlicer Dicerという名称でPIXOTECという米国の会社からUS$500程度で販売されている。これはCTスキャンのデータのような3次元画像のいろいろな断面を表示して実態を把握するためには有効なソフトウェアと思える。

私の属する大気物理の分野ではいくつかのソフトウェアが一般に使われている。もっともよく使われているソフトウェアにGrADS（The Grid Analysis and Display System）（http://www.iges.org/grads）が、次いでGMT（Generic Mapping Tools）がある。GrADSは時空間のグリッドデータを可視化するソフトウェアであり、組み込み関数を持っており流れの場の解析で重要な渦度や発散などを計算して結果も示すことができる。大気の力学的、熱力学的データの表示で広く使われている。データセットを事前に揃えた後でコントロールファイルを使って描画する。GMTはコマンドラインで使うものでライブラリではなくプログラムの集合である。コンパイルにnetCDFと呼ばれるライブラリが必要である。地球流体電脳ライブラリ（https://www.gfd-dennou.org/）は我が国の地球流体力学関係の研究者により開発されたプログラムであり、Fortranをベースにしているが最近C版もだされた模様である。他にも例えばS言語はBell研の開発した統計処理言語でありGNUプロジェクトではR言語となっている。図示化機能も充実しており統計処理の統合環境を提供している。

このような中で、個人用の小型の描画ソフトウェアを開発した。これはデータ解析の統合環境ではなく、単純な図示化のための関数の集合であり、現在の複雑化・高性能化した描画ソフトウェアへのアンチテーゼでもある。ちなみに現在広く使われているUNIXももとはMultixと呼ばれた複雑で結局は失敗したシステムのアンチテーゼとして個人開発されたものであった。本稿ではこの小型描画ソフトウェアの概要とその意義について記す。

２．開発経緯

1970年代は計算機の出力はラインプリンタであり、画像化はラインプリンタで文字を並べる、あるいは行送りを抑制して文字の重ね書きを行うことによっていた。当時、私は修士課程の学生であり2次元乱流の数値実験を行っており、その出力を画像化するために気象庁で作られた数値予報の気圧図などを出力するためのFortranプログラムを利用していた。その頃から計算結果の出力の画像化の必要性を感じていた。国立の研究所に就職し1970年代に渡辺測器（現：グラフテック株）のプロッタを使ったことが本描画プログラムの開発の基礎となった。私はこのプロッタにデータを出力するためにFortranからサブルーティンを呼び出していた。1980年代にレーザープリンタが登場し、レーザープリンタでグラフを出力するようになった。レーザープリンタでは最初は文字を適当に出力することによりグラフを作成していた。この頃は商用のソフトウェアは存在したが、購入しなければならないこと、自分のプログラムに組み込んでグラフ等を自由に描くことはできないこと、で不満足であった。その後postscript言語で直接に図形を描けることを知った。Postscript言語はインタープリタ型であるがスタックを活用した後置記法により実行は高速であった。後置記法は逆ポーランド記法とも呼ばれ、ヒューレットパッカード（HP）の関数電卓で使われた。またForthという簡易言語もあった。私はHPの関数電卓を使用しておりまたForthもわずかにかじっていたためpostscript言語のコマンドにはそれほど違和感は無かった。Postscript言語のコマンド群はプロッタルーティン群と似ており、またテキストであることから、バグ出し、あるいは問題が起こった時には直接に画像ファイルを調べることができる。これからプロッタ用のサブルーティンをpostscript言語を使って書き替えればpostscriptレーザープリンタで、あるいはghostscriptを使って一般のレーザプリンタで図を描くことができるだろうと考えた。

その頃、私の使うコンピュータ言語はFortranからC言語に移っており、開発はUNIX上でC言語を用いて1999年秋から行い、2002年2月には一応完成した。その後は基本構造の大きな変更はなく、既存のルーティンを若干変更したルーティンを新たに加えることでversionが上がっていった。これは必要性に対処するために自然に改訂されていっただけといえる。この改訂は未だに続いている。

現在のversionは1.18.0である。新しいルーティンを加えるなどした時は小数点以下の数字を一つ上げる。コメント追加、チェックの追加、図形や文字の位置の調整など実行には支障のない変更は二つ目の小数点以下の数字を一つ上げる、という表記である。Major番号は１である。時間が経つと肥大化・複雑化し、またあ

まり使わないルーティンが溜まるので、ある段階で整理する時にMajor番号を上げることを考えていたが、実際は上がっていない。

３．プログラムの概要

プログラムは40個以上の関数から成っており、ヘッダーファイル（psHeader.h）に入れてある。通常はヘッダーファイルには宣言等を入れてプログラム本体は入れないものであろうが、直接にヘッダーファイルに入れてある。このヘッダーファイルはhome directoryの直下にPSの名前のdirectoryに入っていることを想定している。プログラムが使う一部のファイルは外に出してある。それはカラー画像用のパレットとそのテストルーティンをいくつか含めたpalettes directoryと世界地図のデータ及び地図の一部の切り出しを行うプログラムを入れたmap directoryである。これらは$HOME/PS/palettes/また$HOME/PS/map/としている。Rainbowとgreyのパレットしか使わないならばpalettes/は無くてよい。また地図を使わないならばmap/も無くてよい。実際には別のLinuxシステムに移行するときにはdirectory $HOME/PS/をそのまま新しいシステムのhome directoryにコピーしている。

ヘッダーファイルを読み込むだけなのでインストールは非常に簡単である。事前にインストールしておかなければならないライブラリは無い。C言語のコンパイラgccが動けばよい。試してはいないが、iccなどgcc以外のCコンパイラでも動く筈である。ヘッダーファイルには描画ルーティンがすべて入っているので、使う時は描画ルーティンも毎回コンパイルすることになるが、プログラムの行数は3千行程度、コメント行を除くと約2300行、と小さく、また現在のコンパイラは十分に速いので問題は生じない。必要ならば動的リンクライブラリを作ることも可能であり、そうすることが普通であろうが、問題が起きたときに直接にコードを見ること、出力に不満があるときに描画ルーティンを直接変更すること、また適当に変えながらテストを行うことがあるので、毎回ヘッダーファイルを含めてコンパイルしている。

既存のソフトウェアの場合は新しい表示法が欲しい場合に、それに見合う表示能力があるかどうかをマニュアルで調べなければならない。そして無い場合にはお手上げとなってしまう。自作の場合はそのようなことは生じない。必要なルーティンの有無は分かっており、もし無ければ既存のルーティンを土台にして新たに作ればよいだけである。

関数はおおよそ以下のように分けられ、この順で使われる。

開始終了：psOpen、psClose、他
初期セットアップ：
　psPutComment、psSetLandscape、psSetMapFile、
　psSetPalette、他
基本ルーティン：
　psPutLine、psPutCircle、psPutEllips、
　psPutInGrid、他
グラフ等の出力
時系列、散布図：
　psPutDataMarks、psPutDataLines、他
画像出力：
　psPutImage、psPutContour、psPutArrows、
　psPutProfiles、他

それぞれの関数の機能は関数の名前を見れば想像はつくと思うが、実際の描画を行っているところはグラフ等の出力ルーティン群である。実際は似た名前で同様の機能を持つが若干異なる関数がいくつもある。Option パラメータをつけて対応することも可能であるが、引数が増えてしまうので簡素性を保つために関数そのものを増やしている。実際に使用しているとGrADSでコントロールファイルのコントロールパラメータが多数あることが理解できる。Appendixにもう少し詳細に記す。

出力するページはA4サイズのみである。これも自分の出力環境が基本的にA4サイズのみであることから決まっている。Postscript言語の上に作っているので何でもできるが、個人の使用環境に沿った形でプログラムが作られている。

psHeader.hの内部はできるだけフラットな構造をとるようにしている。プログラムは構造化すべきという話がある。大がかりなときはいわゆるobject orientedな方法により各モジュールのブラックボックス化、強靭化を図ることもある。しかし、個人開発ではボックスが多くなるとシステム全体の把握が困難になる。また個々のプログラムにおいても構造化を進めるとnestが深くなり可読性が悪くなる。このため似たようなルーティンでも同様のコードを重複を避けずに単純に他のルーティンと並べて書いてある。個々の関数の内部もなるべく関数を使わないようにしている。また履歴を最初に、引数の説明を各関数の最初に入れるなどしてある。説明のコメント行は##で始まっており、

```
cat psHeader.h | grep ¥#¥# | tr -d ¥# | less
```

とすれば各ルーティンの説明のみが表示されるようになっている。自分で作成していても引数などはすぐに忘れるので重宝する。

以下にpsHeader.hを使った y=0.01*x^2 のグラフを出力する極く簡単なプログラム例を示す。また図１にはその結果を示す。

```
// demo.c
#include <stdio.h>
#include <stdlib.h>
#include "/home/nakamura/PS/psHeader.h"
#define    NDATA 100
void       putGraph();
char       comment[200];
float      dtX[NDATA], dtY[NDATA];
```

```
void main()
{
    int    i;
    sprintf(comment,"demo.c");
    for(i=0; i<NDATA; i++) {
        dtX[i]=(float)i;
        dtY[i]=0.01*(float)(i*i);
    }
    putGraph();
}

void putGraph()
{
    char    title[100], strX[100], strY[100];
    int     x0, y0, dx, dy, nGridX, nGridY;
    float   sX, eX, sY, eY;

    // location and size setup
    x0=150; y0=500; dx=250; dy=200;
    nGridX=5; nGridY=2;
    // min and max for x and y values
    sX=0.; eX=100.; sY=0.; eY=100.;
    sprintf(title,"Example");
    sprintf(strX,"x-value");
    sprintf(strY,"y-value");
    psOpenD();
    psPutComment(comment);
    psPutInGridYL(x0, y0, dx, dy, sX, eX, sY, eY,
      nGridX, nGridY, title, strX, strY);
    psPutDataMarks(dtX, dtY, NDATA, x0, y0, dx, dy,
      sX, eX, sY, eY, "tSquare");
    psClose();
}
```

このプログラムでは描画のサブプログラム putGraph()に変数を渡すためにはglobal変数を使っている。putGraph()の内部ではpsファイルを開き、グラフの位置、大きさ、グラフのタイトル、縦軸、横軸の題、最大値、最小値などを引数として描画する関数に与えている。

描画の基本は非常に単純である。出力ファイルを開いてから最初にpsファイルであることの宣言、描画範囲などの設定（具体的にはBoundingBoxの設定)、またglobal変数の宣言などをpsOpenD()で行い、その後実際の描画ルーティンを呼び出す。描画ルーティンはpostscript言語そのものが描画コマンドを持っているのでそれを使うだけである[1]。例えばpostscript言語で座標（x0, y0）から(x1, y1)へ太さwの線を描くにはテキストで

```
newpath
x y moveto
x1 y1 lineto
w setlinewidth
stroke
```

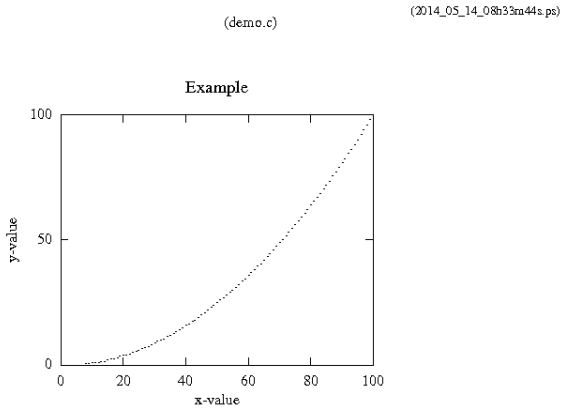

図１　描画の例。y=0.01*x^2のグラフを表示している。上部右には出力日時による画像のファイル名が、また上部中央にはコメント（この場合はプログラム名）が書き込まれている。

とすれば良く、作成したルーティン psPutLine(int x0, int y0, int x1, int y1, float linewidth) はx0、y0などを引数として取り込んで

```
fprintf(fpoPS, "newpath¥n");
fprintf(fpoPS, "%d %d moveto¥n",x0,y0);
fprintf(fpoPS, "%d %d lineto¥n",x1,y1);
fprintf(fpoPS, "%f setlinewidth¥n",linewidth);
fprintf(fpoPS, "stroke¥n");
```

としてファイルにテキストを書き込んでいるだけである。なおここでは出力ファイルへのポインターをfpoPSとしている。

カラー画像は画素毎の数値データを色パレットの表を使って（実際には計算して）RGB値に変換してからファイルに流しこむことで作っている。具体的には柴山（1994）[2]に従ってファイルに以下を書き込んでいる。

```
/picstr XXX string def
X0 Y0 translate
DX DY scale
NUMX NUMY 8
{currentfile}
picstr readhexastring pop
false 3
colorimage
RGBRGB....
```

ここでX0、Y0は原点の位置、DX、DYは図のサイズ、NUMX、NUMYはデータのx、y方向の数、そしてRGBRGB...はRGBのそれぞれ8ビットの値である。

Postscript言語のコマンドは非常にprimitiveなものしか使っていない。逆にいうと凝ったことを要求しなければ単純なコマンドだけで良い、ということであろう。

４．実際の使用

頻繁に必要となる図はカラー画像、時系列図、そして散布図である。また必要な図を作成するためにその都度作成したプログラムも多い。

カラー画像の場合はパレットが必要となる。パレットは何も指定しなければいわゆるrainbowカラーがdefaultとなる。パレットはgreyを含めていくつか作成したが、実際に使用したものはrainbowがほとんどで時々greyを使うだけであった。少し変わったカラー画像出力ルーティンとしていわゆる彩度も取り入れたものも作成した。これは熱帯域の降雨の日周変化の図を作成するときに作ったルーティンである。熱帯域の陸上ではスコールとよばれる午後の雨がある。一方海上では明け方の雨が多い。この特性を表すのに色で降雨の多い時刻を表すことが普通であるが、これでは日周変化の強さ、つまり午後の同じ時刻に雨が多いとしても、その雨が他の時刻に比べて少し多いだけなのかあるいは非常に多いのかは分からない。そこで日周変化の強さの指標を作りそれを「彩度」のパラメータとして取り入れた。例えば午後に強い雨が多いとその場所は明瞭な赤となり、少し強いだけであると赤みがかった灰色となる、ということである。これは予想したほどの視覚効果はなかったものの面白い図となった。

自らの需要に応えてその都度ルーティンを加えながら作っているので体系的ではない。いくつかのルーティンは単一の目的のために作られている。例えばpsPutEllips() は楕円を表示するもので、緯度経度の10度グリッド内での降雨域の平均の大きさと方向を表示するためだけに作成した。

仕事柄、地図の上にいろいろな分布の画像を出力するが、使っている地図データは輪郭の緯度経度を並べ、つながっていない別の輪郭のデータとの間に空白行を入れただけの簡単なテキストファイルである。地図を描くルーティンは例外処理が多くもっとも複雑で今や変更不能のルーティンとなっている。コメントにも”below is a spaghetti logic. Don’t touch.” と入っている。例として図２に６～８月の世界の平均降雨強度の分布を示す。もともとの画像データは緯度経度0.1度の解像度である。

特殊用途として流れの場を矢印で表したり、雨の鉛直プロファイルの時系列を表わすのに隠れ線処理を行うことにより疑似３次元表示を行うルーティンもある。

頻繁に使うと考えて作成したがほとんど使わなかった関数もある。その一つはpsSetVerbose()である。これは関数を呼び出す毎に関数名を標準出力に出力するための関数で、デバッグ等で使う予定であったが、直接にプログラムを見た方が早いということでほとんど使うことは無かった。

psHeader.hに含めるか迷ったルーティンもある。画像の中でつながっているグリッドセルに同じ番号を与えるいわゆるラベリングを行うルーティンである。これは再帰的に書くと非常に簡単に書けるが（画像処理アルゴリズムの基本と仕組み、長尾智晴）[3]、ここでは一つのセルの周りのセルを単純にチェックしていく接続表を使った形の方法を用いている。これは雨域の大きさの統計をとる時に必要となった。このルーティンは完成はしたが図示化ルーティンではないとして結局はpsHeader.hには含めなかった。しかし、psHeader.hを入れているdirectoryに置いた。

出力ファイルはテキストファイルであり内容は分かっているので、出力の細かい調整も直接に出力ファイルを変更することで可能である。例えば出力されるコメントの内容を変更することは容易である。また二つの出力を重ねることも簡単である。具体的には最初のファイルの最後に書かれている “showpage”を外して二つ目のファイルのテキストとつなげるだけでできる。

細かい部分で自分の趣味も入っている。例えば画像ファイル名は2014_04_09_09h50m41s.psのように出力日時であり、また図１のようにA4のページの右上に出力時の年月日時分秒のファイル名を自動的に書き出すことができる。これはデータの解析途中では画像を沢山出力するが、最新の画像を探す時や解析結果がある程度まとまった後に必要画像を取りそろえるときに役に立つ。出力した画像ファイルには適当な名前をその場で与えれば良いのであるが、解析途中での出力画像は試行であるものがほとんどであるため適当な名前を与える余裕が無く、後からまとめることは困難である。もっともこのファイル名には、連続処理により図を次々に作成する時に1秒以内に次の画像ファイルが作られるとその前の画像ファイルが上書きされてしまうという弊害もある。このためpsHeader.hを使った連続画像出力プログラムではsystem関数のtime()を使って秒が異なることを確かめてから出力させるなどの余計な処理が必要となっている。これを避けるため秒以下の細かい時刻を使うことも考えたが、ファイル名が長くなること、根本的な解決にはならないこと、から行わなかった。システム関数を呼び出して既存ファイルの名前をチェックして同じ名前があるときには例外処理を行うことも考えたが、プログラムが複雑化するのでこれも避けた。他人が利用することを考えた場合は、複雑となっても例外的な状況にも対応できるようにしておくべきであろうが、自分で使うことを目的としており、随時プログラムの中身を見ることから、簡素性と可読性に重きを置いたことになっている。

図１にもあるようにコメントをpsPutComment()でページ上部に出力することもできる。コメントには通常はプログラムを走らせた時のコマンドラインを入れるようにしている。こうするとプログラム名、処理したデータファイル名、また処理パラメータ等が自動的に入り、出力された画像のback trackingに大いに役立つ。

出力される画像ファイルの名前が出力日時時刻となっていることは意外に便利なことがある。別PCで同時に動かすようなことが無ければファイル名は唯一となり、紛れが無くなる。これを利用すれば一つのdirectoryに全部のpsファイルを集めてしまうことができる。また自然に作成時刻順に並ぶ。現在ハードディスクの容量は大きいので、出力画像ファイルを取捨選択せずにどんどん保存しても問題はないので、試し

出力画像もすべて何も考えずに保存することができる。書類のファイリング手法に、こまめに分類する、とにかく日付順に並べる、使った最新のファイルをもっとも上部に置く、などがあるが、今の場合は、とにかく日付順に並べる手法に対応しているともいえる。しかしこのままではゴミ画像も沢山入ってしまうので、使える画像ができたときにはなるべく頻繁にプリンタ出力するようにしている。そして処理プログラムと出力画像のプリンタ出力を単純に時刻順にファイリングしている。この時、コマンドラインをコメントとして画像内に出力しておくことが有効に働く。紙とインクの消費はかなりあるがそのような問題ではない。また画像ファイルの整理などのためにpsファイルを自動的にps2psを通してから印刷するlprpsあるいはpslpr、もっとも新しいpsファイルをdisplayに出力するあるいは画像保存directoryに移すgsltst、mvpsltst、current directoryにあるpsファイルを一括してプリントするlprpsAll、画像ファイル保存用のdirectoryに一括して移すmvps2Figsなどの小物のコマンドをshell scriptとして作成してある。昔のUNIXは小回りの利くコマンドの塊と言える面があったが、それに似ているとも言えよう。

プログラムの書き方また使用法はほぼ定型化されており自分が使用する分には不都合は無く、また自分の趣味に合っており居心地も良い。逆にいうと他の人がこの描画ルーティンを使った場合には居心地が悪かろう、ということになる。2次元、3次元、4次元のデータを多面的な視点から様々に描画する高級なソフトウェアもあり、現象のメカニズムの理解や解析の方向を探るためには大変有用であるが、私の通常のデータ解析では時系列のグラフと散布図、そして分布の画像出力がもっとも多い。そして作成した描画ルーティン群は私の通常の仕事では十分に実用となっている。

５．小型描画プログラムの意義

この小型の描画ソフトウェアを作成したのは描画ソフトウェアが当時はまだ十分には流布されておらず、特に画像化ソフトウェアが不満足であったためである。振り返ってみて、このような小型のソフトウェアの現代での意義には以下が考えられる。

（1）描画ルーティンの中身が明確に分かる。

このメリットは当然ではあるが、実際に使う場合は図が作られれば良いのであり描画ソフトウェアの中身を知る必要は必ずしも無いので副産物というべきものであろう。マイクロソフトのエクセルによるグラフがどのように作られているかを知る必要は無いことと同様である。

（2）自分の嗜好に合った画像、グラフが得られる。

これは重要ではないが、個人の趣味を満足させているので実際の使用にあたってはストレスが生じない。

（3）自分の嗜好に合った処理手順を踏むことができる。

これは（2）と類似している。連続処理をするときなどは個々人それぞれのやり方があると思われる。私の場合は、実行形式のプログラムを一回走らすだけで図が作成できることを目指している。中間データのファイルを作成し、それに対して新たに描画プログラムを走らすことは、同じ処理を多数行う場合、いくつかのプログラムまたコントロールファイルを用意してまとめておかなければならなくなり、パッケージ化が必要となってくる。複数の人で系統的に処理を行う場合はその方が良いと考えられるが、個人で行うときには手順が複雑になるので避けたい。

（4）新たな表示法を考えることができる。

これは新しい表示法を得るためには重要である。しかしopen sourceで開発されている様々なソフトウェアでは様々の要求に対応するべく開発されているのでこれとの優劣はつけがたい。大手に対するベンチャー活動の意義に似ているというべきであろう。

（5）移植が簡単である。

これは重要である。かなり強力なPC Linux のシステムがほぼ個人用に普及しているなかではシステムの維持は自分で行う必要がある。ハードウェアの進歩、UNIXの進歩、そして様々なソフトウェアやライブラリが現れそれがどんどん改訂されていく中でシステムを維持し、また既存のシステムに新しいソフトウェアをインストールすることは往々にして手間がかかる。私は最近、新しい衛星のためのデータ読み出し用のtoolkitを仕事用のPCにインストールしたが、かなりの時間を使った。その衛星のデータはHDF (Hierarchical Data Format) (http://www.hdfgroup.org/) と呼ばれる形式で配布されており、それからデータを読み出すためにHDFのライブラリと、その上で動く米国航空宇宙局(NASA)の開発した当該衛星に特化したtoolkit(tkio)をインストールする必要があった。この時、Linux Kernel、Perlや圧縮解凍用のライブラリzlib、JAVA Runtime Environment (JRE)、gcc、gfortranのそれぞれのある程度新しいversionのものが必要であった。さらに実際に使う際には環境変数を設定したり、pathを設定したり、またgccでコンパイルする際に多数のincludeファイルやlibraryのリンクを指定しなければならないなど面倒なことがあった。ソフトウェアの規模は全く異なるので比較にはならないが作成した小型の描画ルーティンは短いsourceでありパレットデータや地図データ等を含めて小さいdirectory一つをコピーするだけで新しいPCへ移植することができる。Source ファイルをheaderファイルとして取り込む形なのでmakeなどによる新たなコンパイルの必要も無い。少し前まではpsheader.hの一か所でsqrt()を使っているためgcc -lmとしてmath libraryのリンクが必要であったが、すっきりさせるためにsqrt()を顕わに書き替えることによりmath libraryのリンクも不要としている。

関連するがカスタマイズは簡単である。というよりも常にカスタマイズしていると言ってよい。様々の機能をもった大型の汎用のソフトウェアもカスタマイズが可能ではあるが、その都度マニュアルを参照する必要が生じるなどカスタマイズは面倒である。頻繁に作成する図はカラー画像、時系列図、散布図だけであり、

これらは汎用のソフトウェアの出力とは遜色は無いと考えている。しかし文字サイズや自動グリッドなどは汎用のものの方が優れている。また汎用ソフトウェアでは図のカラー化は一般的であるが、本ソフトウェアではカラー画像以外のカラー化はなされていない。

本ソフトウェアは自分用であるが、他人が本ソフトウェアを使用する場合は大きなプログラムではないので必要ならばコードを見る気があれば十分に使用に耐えると考える。しかしコードにまでは立ち入らずに使用する場合には問題がある。他人の場合には自分の場合とは異なる使用となる。また例外処理が不完全な場合や文字サイズなどが使用者の嗜好に合わない、などの問題が生じよう。自分のグループ内での使用ならともかく、そうでない他人の場合にはプログラムの完成度と広がりが重要となる。さらにはversion管理なども必要となる。Public domainで世に出ているソフトウェアは数多くあるが多くは多人数で様々に使用し、改訂に次ぐ改訂によりソフトウェアを鍛え上げている。本ソフトウェアも他人の使用を考えた場合には、まずは少人数で試験使用を行い、完成度を高め、かつ様々な使用に耐えるようカバーする範囲を広げなければならないであろう。

６．おわりに

複雑化、大型化する描画ソフトウェアのアンチテーゼとして極く小規模なデータ画像出力ソフトウェアの開発について述べた。個人用であり直接プログラムの中身を見ることが多いという点で多くの人が使うことを想定した大型のソフトウェアとは設計思想が異なっている。大型のソフトウェアの多機能性には及びもつかないが、自分が必要とする図はすべて作りだすことができている。これは自らの需要に応えて作成したものであるから当然ともいえる。また使用形態を自分の嗜好に合わすことができることは、本質的ではないとはいえデータの解析作業中のストレスをかなり軽減する。

このような個人用描画ソフトウェアの意義は、描画ソフトウェアが限られていた少し前はともかく現在は小さいとは思われる。しかし、大型で高機能のソフトウェアにより処理が定型化されつつあるなかで、個人の嗜好に沿った小型のソフトウェアも新たなシーズ開拓の一端となるのではないかと考える。

謝辞

本研究の一部は、情報科学研究所研究助成によるものである。また査読者からは有益はコメントを頂いた。

参考文献

(1) Adobe Systems、PostScript チュートリアル&クックブック、アスキー出版局、295pp. (1989).
(2) 柴山守、X11 による画像処理、技術評論社、367pp. (1994).
(3) 長尾智晴、画像処理アルゴリズムの基本と仕組み、秀和システム、262pp. (2010).

追記

最近新しいプリンタを購入しこれのLinuxドライバーをインストールしてpsファイルを出力しようとしたところうまく出力されなかった。Display上では問題無く表示される。理由は分からないがps2psあるいはps2pdfを使って変換してから出力するとうまくいく。ps2psをかけた後のファイルを見ると最初にかなりの量のテキストが入っており解読する気力が無く、そのままとなっている。

Appendix　プログラム群の構成

描画プログラム群の構成については第３章にあるがより詳しく記す。プログラムは大きく開始、描画、終了に分かれ、この順で使われる。また一部のプログラムのための下位のプログラムがある。この下位プログラムは表には現れない。データのやりとりは引数によっている。以下に頻繁に使っているプログラムを羅列する。これら以外にも余り使用しないプログラムや少し異なる派生プログラムがある。例えばグリッドを描く時、目盛の線を枠の中あるいは外に記す場合は異なったプログラムとなっている。

開始プログラム群

psOpenD()：出力ファイルをオープンすると共にページ上部右に日時によるファイル名を記す。
psPutComment()：コメントをページ上部に記す。通常はコマンドラインを入れる。
psSetLandscape()：横置きを指定する。
psSetMapFile()：地図ファイルを指定する。
psSetPalette()：画像のカラーパレットを指定する。

描画プログラム群

psPutInGridY()：図の枠等を描く。
psPutImageX()：カラー図を描く。
psPutDataMarks()：指定されたマークを指定された位置に置く。散布図、時系列の描画に使用する。
psPutDataLines()：指定されたデータポイントを線で結ぶ。主に時系列の描画に使用する。

終了プログラム

psClose()：出力ファイルの最後に”showpage”を書き込んでからファイルをクローズする。

下位プログラム群

psPutLine()：指定された幅の線を指定された場所に描く。
psPutRectangle()：四角を描く。
psPutColorBar()：カラー画像用のカラーバーを描く。
psPutString()：文字を記す。
strshort_()：グリッドに数字を入れるときに数字の頭の空白を除き、また後ろの０を除く。
psSetMark()：散布図や時系列のマークを指定する
psPutMark()：マークを指定された位置に描く。

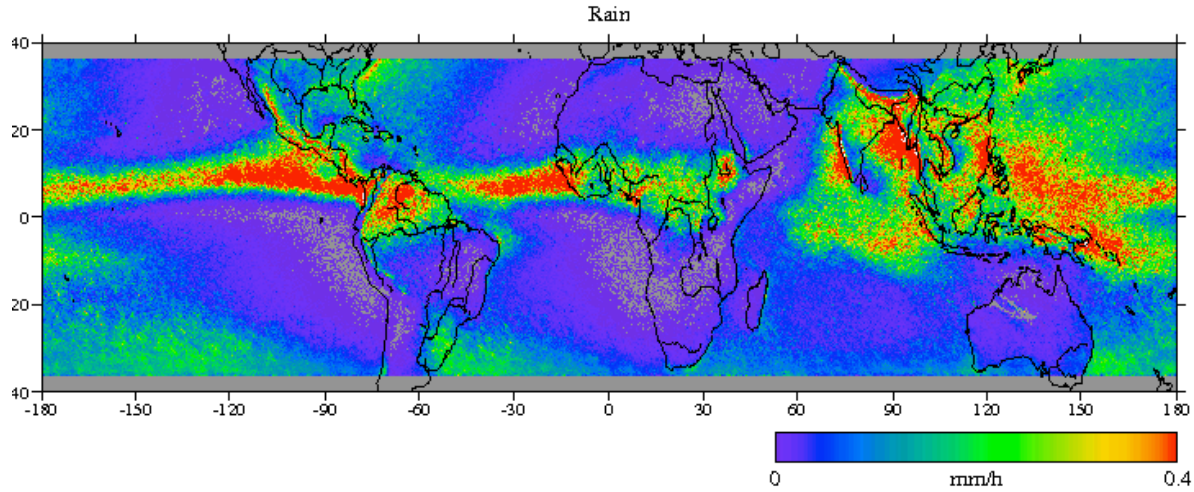

図２　出力画像の例。衛星による世界の夏（6, 7, 8 月）の 1998 年から 2013 年の平均の降雨強度の分布。熱帯域の降水の細かい分布が分かる。

(2014 年 9 月 30 日受付)
(2014 年 12 月 3 日採録)