システム工学概論

4. Cによる数値計算と可視化（基礎編）

山田泰司

taiji@aihara.co.jp

株式会社あいはら 研究開発チーム

数値計算の可視化とグラフィックライブラリ

- X11(X Window System) - **ポータブル・ネットワーク透過ウインドウシステム**
- GTK+(Gimp Tool Kit+) - Gnome **デスクトップ環境の基盤技術**
- Qt(**キュート**) - KDE **デスクトップ環境の基盤技術**
- SDL(Simple DirectMedia Layer) - **ポータブル・高性能マルチメディアライブラリ**

その他の選択肢として

- Win32 API, MFC, DirectX(Windows)
- Quartz, Cocoa(Mac OS X)
- OpenGL(Mac OS X, X11, Windows) - **高性能な** 3D **グラフィックス**

グラフィックライブラリとしての X11

グラフィックライブラリとしての X11：

- **「X11 を直接叩く」という言い回しがあるように、低レベルライブラリ故の面倒な印象が、現在では確かにある。**
- **それはあくまで GTK+, Qt 等と比較して、特定の GUI、つまりデスクトップ環境に沿ったアプリケーション作成時における問題に過ぎない。**
- **特に数値計算を主眼とするプログラムであれば、グラフィックライブラリとしての X11 は特に難しい事はない。**

X11 プログラミングができるようになる事の利点：

- **Unix, Cygwin/X11, Mac 等においてグラフィックスが扱えるようになる**
- **ネットワーク透過性により、プログラムがリモートで動く**
- **枯れたテクノロジーなので、プログラムの長寿命が期待できる**

X11 基本ライブラリ Xlib の学び方

現在入手が容易な日本語の入門書：

▷ **大村 正道**, 「X Window **練習帳**」, I/O **ブックス**, 2006.

実際の学び方：

▷ **ヘッダファイル** `X11/Xlib.h` **をざっと読む。**

```
$ less /usr/X11R6/include/X11/Xlib.h
```

▷ **そこで書かれている関数のマニュアルを読む。**

```
$ man XOpenDisplay
```

▷ **深く知る必要があるなら** `X11` **のソースコードを読む。**

但し Solaris **の場合、**`/usr/X11R6` → `/usr/openwin` **に読み変える。**

（例題１）２次元グラフィックス

以下のグモウスキー・ミラの写像と呼ばれる差分方程式：

$$\begin{aligned} x_{t+1} &= y_t + 0.008(1 - 0.05y_t^2)y_t + F(x_t) \\ y_{t+1} &= -x_t + F(x_{t+1}) \end{aligned}$$

(但し、$F(x) = \mu x + 2(1-\mu)x^2/(1+x^2)$**) において** $\mu = -0.8$, $x_0 = 0.1, y_0 = 0$ **における** $t < 50000$ **までの値を** X11 **のディスプレイにて可視化せよ。**

但し、何かキーが押されたら、プログラムが終了するようにせよ。

（例題１）の解答例 1/8

まず、可視化に必要な変数を詰め込むための構造体を用意する。

```
          :
#include <X11/Xlib.h>
          :
struct view {
  int w, h;
  Display *display;
  int screen_number;
  Window parent_window, window;
  GC gc;
  long event_mask;
  XEvent event;
};
```

（例題１）の解答例 2/8

可視化のための初期設定：

```
void view_init(struct view *v)
{
  if (!(v->display = XOpenDisplay(NULL))) exit(1);
  v->screen_number = DefaultScreen(v->display);
  v->parent_window = RootWindow(v->display, v->screen_number);
  v->window = XCreateSimpleWindow(v->display, v->parent_window,
                                  0, 0, v->w, v->h, 0,
                                  BlackPixel(v->display, v->screen_number),
                                  WhitePixel(v->display, v->screen_number));
  v->gc = DefaultGC(v->display, v->screen_number);
  XSetForeground(v->display, v->gc, BlackPixel(v->display, v->screen_number));
  v->event_mask = ExposureMask;
  v->event_mask |= KeyPressMask;
  XSelectInput(v->display, v->window, v->event_mask);
  XMapWindow(v->display, v->window);
  XFlush(v->display);
  while (!0) {
    XNextEvent(v->display, &v->event);
    switch (v->event.type) {
    case Expose: return; break;
    }
  }
}
```

（例題１）の解答例 3/8

可視化の後始末：

```
void view_term(struct view *v)
{
  XUnmapWindow(v->display, v->window);
  XCloseDisplay(v->display);
}
```

（例題１）の解答例 4/8

最小限のイベントループ：

```
void view_loop(struct view *v)
{
  while (!0) {
    XNextEvent(v->display, &v->event);
    switch (v->event.type) {
    case KeyPress: return; break;
    }
  }
}
```

（例題１）の解答例 5/8

可視化：

```
void view_draw(struct view *v, void *o)
{
  struct deset *des = (struct deset *)o;
  int i, j;
  int x, y;

  XClearWindow(v->display, v->window);
  for (i=0; i<50000; i++) {
    des->de->map(des->p, i, des->v0, des->v1);
    x = (des->v0[0] - des->lb[0])/(des->ub[0] - des->lb[0])*v->w;
    y = (des->v0[1] - des->ub[1])/(des->lb[1] - des->ub[1])*v->h;
    XFillRectangle(v->display, v->window, v->gc, x, y, 1, 1);
    for (j=0; j<des->de->N; j++)
      des->v0[j] = des->v1[j];
  }
  XFlush(v->display);
}
```

（例題１）の解答例 6/8

```
void gm_init(double p[], double v0[], double lb[], double ub[])
{
  mu = -0.8;
  x0 = 0.1; y0 = 0.0;
  lb[0] = -20; ub[0] = 25;
  lb[1] = -14; ub[1] = 12;
}
        :
struct deset {
  struct de *de;
  double *p, *v0, *v1;
  double *lb, *ub;
};
        :
```

（例題１）の解答例 7/8

```
          :
int main(int argc, char *argv[])
{
  struct deset deset = {
    de,
    malloc(sizeof(double)*de->K),
    malloc(sizeof(double)*de->N), malloc(sizeof(double)*de->N),
    malloc(sizeof(double)*de->N), malloc(sizeof(double)*de->N),
  }, *des = &deset;
  struct view view = {
    600, 600,
  }, *v = &view;

  des->de->init(des->p, des->v0, des->lb, des->ub);
  des->de->options(argc, argv, des->p, des->v0);
  view_init(v);       /* 初期化して */
  view_draw(v, des); /* 描画して */
  view_loop(v);       /* 終了待ち */
  view_term(v);       /* 後始末 */
  return 0;
}
```

（例題１）の解答例 8/8

makefile:

```
CPPFLAGS=-I/usr/X11R6/include
LDFLAGS=-L/usr/X11R6/lib
LDLIBS=-lX11 -lm

SRCS=\
v_gm00.c          \

EXES=$(SRCS:.c=)

all: $(EXES)
```

```
$ ./v_gm00
```

（例題２）２次元アニメーション

グモウスキー・ミラの写像と呼ばれる差分方程式において
$\mu = -0.8, x_0 = 0.1, y_0 = 0$ における $t < 50000$ までの値を X11 のディスプレイにて可視化し、μ を徐々に変化させよ。

また、アニメーションに適した描画の工夫も行なうとよい。

さらに、ウインドウのリサイズに対応した工夫を行なうとよい。

（例題２）の解答例 1/3

可視化：

```
void view_draw(struct view *v, void *o)
{
  struct deset *des = (struct deset *)o;
  int i, j;
  int x, y;

  XClearWindow(v->display, v->window);
  for (i=0; i<50000; i++) {
    des->de->map(des->p, i, des->v0, des->v1);
    x = (des->v0[0] - des->lb[0])/(des->ub[0] - des->lb[0])*v->w;
    y = (des->v0[1] - des->ub[1])/(des->lb[1] - des->ub[1])*v->h;
    XFillRectangle(v->display, v->window, v->gc, x, y, 1, 1);
    for (j=0; j<des->de->N; j++)
      des->v0[j] = des->v1[j];
  }
  XFlush(v->display);
  des->p[0] += 0.0001;  /* パラメータ配列の第０要素が呼ばれる度に増加 */
}
```

（例題２）の解答例 2/3

アニメーションのイベントループ：

```
#include <unistd.h>     /* usleep */
        :
void view_loop(struct view *v, void *o)
{
  while (!0) {
    view_draw(v, o); /* ここで描画 */
    usleep(10*1000); /* 10000 マイクロ秒間の CPU 資源解放 */
    while (XPending(v->display)) { /* イベントがある場合 */
      XNextEvent(v->display, &v->event);
      switch (v->event.type) {
      case KeyPress: return; break;
      }
    } /* イベントがない場合 */
  }
}
```

（例題２）の解答例3/3

```
int main(int argc, char *argv[])
{
        :
  view_init(v);          /* 初期化して */
  view_loop(v, des); /* イベントに応じて描画 */
  view_term(v);          /* 後始末 */
  return 0;
}
```

```
$ ./v_gm01
```

flicker(**ちらつき**) **を防止したい。**

（例題２）の改良：アニメに適した描画 1/2

```
$ diff -u v_gm01.c v_gm02.c
```

```
--- v_gm01.c    2006-11-20 14:01:18.000000000 +0900
+++ v_gm02.c    2006-11-20 13:53:55.000000000 +0900
@@ -65,6 +65,7 @@
   Display *display;
   int screen_number;
   Window parent_window, window;
+  Pixmap pixmap;
   GC gc;
   long event_mask;
   XEvent event;
@@ -79,6 +80,8 @@
                                     0, 0, v->w, v->h, 0,
                                     BlackPixel(v->display, v->screen_number),
                                     WhitePixel(v->display, v->screen_number));
+  v->pixmap = XCreatePixmap(v->display, v->window, v->w, v->h,
+                            DefaultDepth(v->display, v->screen_number));
   v->gc = DefaultGC(v->display, v->screen_number);
   XSetForeground(v->display, v->gc, BlackPixel(v->display, v->screen_number));
   v->event_mask = ExposureMask;
```

（例題２）の改良：アニメに適した描画 2/2

```
@@ -99,15 +102,18 @@
   int i, j;
   int x, y;

-  XClearWindow(v->display, v->window);
+  XSetForeground(v->display, v->gc, WhitePixel(v->display, v->screen_number));
+  XFillRectangle(v->display, v->pixmap, v->gc, 0, 0, v->w, v->h);
+  XSetForeground(v->display, v->gc, BlackPixel(v->display, v->screen_number));
   for (i=0; i<50000; i++) {
     des->de->map(des->p, i, des->v0, des->v1);
     x = (des->v0[0] - des->lb[0])/(des->ub[0] - des->lb[0])*v->w;
     y = (des->v0[1] - des->ub[1])/(des->lb[1] - des->ub[1])*v->h;
-    XFillRectangle(v->display, v->window, v->gc, x, y, 1, 1);
+    XFillRectangle(v->display, v->pixmap, v->gc, x, y, 1, 1);
     for (j=0; j<des->de->N; j++)
       des->v0[j] = des->v1[j];
   }
+  XCopyArea(v->display, v->pixmap, v->window, v->gc, 0, 0, v->w, v->h, 0, 0);
   XFlush(v->display);
   des->p[0] += 0.0001;
 }
```

```
$ ./v_gm02
```

（例題２）の改良：リサイズに対応

```
$ diff -u v_gm02.c v_gm03.c
```

```
--- v_gm02.c    2006-11-20 13:53:55.000000000 +0900
+++ v_gm03.c    2006-11-20 13:54:04.000000000 +0900
@@ -86,6 +86,7 @@
   XSetForeground(v->display, v->gc, BlackPixel(v->display, v->screen_number));
   v->event_mask = ExposureMask;
   v->event_mask |= KeyPressMask;
+  v->event_mask |= StructureNotifyMask;
   XSelectInput(v->display, v->window, v->event_mask);
   XMapWindow(v->display, v->window);
   XFlush(v->display);
@@ -126,6 +127,14 @@
       XNextEvent(v->display, &v->event);
       switch (v->event.type) {
       case KeyPress: return; break;
+      case ConfigureNotify:
+       v->w = v->event.xconfigure.width;
+       v->h = v->event.xconfigure.height;
+       XFreePixmap(v->display, v->pixmap);
+       v->pixmap = XCreatePixmap(v->display, v->window, v->w, v->h,
+                                 DefaultDepth(v->display, v->screen_number));
+       view_draw(v, o);
+       break;
       }
     }
   }
```

（例題３）２次元アニメ＋色空間

グモウスキー・ミラの写像と呼ばれる差分方程式において $\mu = -0.8, x_0 = 0.1, y_0 = 0$ **における** $t < 50000$ **までの値を、距離** $|(x_{t+1}, y_{t+1}) - (x_t, y_t)|$ **に応じて着色して可視化し、** μ **を徐々に変化させよ。**

☞ **色空間のモデル：**RGB, CMY, HSV, HLS, YUV, CIE **など**

☞ HSV **色空間**: Hue(**色相**), Saturation(**彩度**), Value(**明度**) **による。色相が** RYGCBM **の円環によるグラデーションカラーなので、ひとつのスカラー量を色相に対応させることにより視覚化する。**

（例題３）の解答例 1/5

２点の N 次元座標値の距離を算出する関数：

```
          :
#include <math.h>       /* sqrt, modf */
          :
double vec_dist(int N, double a[], double b[])
{
  int i;
  double d = 0;

  for (i=0; i<N; i++)
    d += (a[i] - b[i])*(a[i] - b[i]);
  return sqrt(d);
}
```

（例題３）の解答例 2/5

HSV から RGB 色空間へ変換する関数：

```
/*
 * HSV to RGB color
 * in:  double h(hue), s(saturation), v(value) in [0, 1]
 * out: double r(red), g(green), b(blue) in [0, 1]
 */
void HSV2RGB(double h, double s, double v, double *r, double *g, double *b)
{
  double R, G, B;
  double f, i, m, n, k;

  h *= 6;
  f = modf(h, &i);
  m = v*(1 - s);
  n = v*(1 - s*f);
  k = v*(1 - s*(1 - f));
  switch ((int)i) {
  case 0: R = v; G = k; B = m; break;
  case 1: R = n; G = v; B = m; break;
  case 2: R = m; G = v; B = k; break;
  case 3: R = m; G = n; B = v; break;
  case 4: R = k; G = m; B = v; break;
  case 5: R = v; G = m; B = n; break;
  }
  *r = R; *g = G; *b = B;
}
```

（例題３）の解答例 3/5

```
@@ -67,6 +106,8 @@
   Window parent_window, window;
   Pixmap pixmap;
   GC gc;
+  Colormap cmap;
+  XColor colors[1024];
   long event_mask;
   XEvent event;
 };
@@ -83,6 +124,20 @@
   v->pixmap = XCreatePixmap(v->display, v->window, v->w, v->h,
                             DefaultDepth(v->display, v->screen_number));
   v->gc = DefaultGC(v->display, v->screen_number);
+  v->cmap = DefaultColormap(v->display, v->screen_number);
+  {
+    int c, ncolors = sizeof(v->colors)/sizeof(XColor);
+
+    for (c=0; c<ncolors; c++) {
+      double R, G, B;
+
+      HSV2RGB((double)c/ncolors, 1, 1, &R, &G, &B);
+      v->colors[c].red = 65535*R;
+      v->colors[c].green = 65535*G;
+      v->colors[c].blue = 65535*B;
+      XAllocColor(v->display, v->cmap, &v->colors[c]);
+    }
+  }
       :
```

（例題３）の解答例 4/5

```
@@ -27,6 +65,7 @@
    x0 = 0.1; y0 = 0.0;
    lb[0] = -20; ub[0] = 25;
    lb[1] = -14; ub[1] = 12;
+   lb[2] = 0; ub[2] = 50;
  }
  void gm_map(double p[], double t, double v0[], double v1[])
  {
@@ -152,8 +212,8 @@
      malloc(sizeof(double)*de->K),
      malloc(sizeof(double)*de->N),
      malloc(sizeof(double)*de->N),
-     malloc(sizeof(double)*de->N),
-     malloc(sizeof(double)*de->N),
+     malloc(sizeof(double)*(de->N+1)),
+     malloc(sizeof(double)*(de->N+1)),
    }, *des = &deset;
    struct view view = {
      600, 600,
```

（例題３）の解答例 5/5

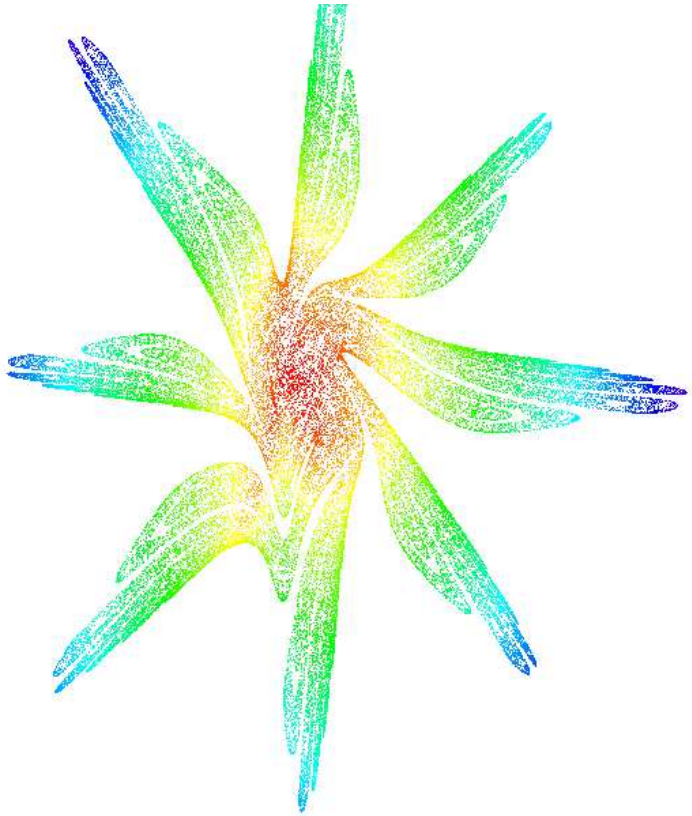
```
@@ -105,15 +160,20 @@
 {
   struct deset *des = (struct deset *)o;
   int i, j;
-  int x, y;
+  int x, y, c, ncolors = sizeof(v->colors)/sizeof(XColor);
+  double distance01;

   XSetForeground(v->display, v->gc, WhitePixel(v->display, v->screen_number));
   XFillRectangle(v->display, v->pixmap, v->gc, 0, 0, v->w, v->h);
   XSetForeground(v->display, v->gc, BlackPixel(v->display, v->screen_number));
   for (i=0; i<50000; i++) {
     des->de->map(des->p, i, des->v0, des->v1);
+    distance01 = vec_dist(des->de->N, des->v0, des->v1);
     x = (des->v0[0] - des->lb[0])/(des->ub[0] - des->lb[0]) * v->w;
     y = (des->v0[1] - des->ub[1])/(des->lb[1] - des->ub[1]) * v->h;
+    c = (distance01 - des->lb[2])/(des->ub[2] - des->lb[2]) * ncolors;
+    XSetForeground(v->display, v->gc, (0<=c&&c<ncolors)?
+                   v->colors[c].pixel:BlackPixel(v->display, v->screen_number));
     XFillRectangle(v->display, v->pixmap, v->gc, x, y, 1, 1);
     for (j=0; j<des->de->N; j++)
       des->v0[j] = des->v1[j];
```

```
$ ./v_gm04
```

（例題３）の結果

ImageMagick の import コマンドで撮影：

```
$ import filename.eps
```



数値計算とグラフィックス：まとめ

数値計算結果の理解や表現を助ける可視化技術 — 自らが理解を深めるには可視化された数値計算結果を積極的に観ることが重要である：

- ▷ ２次元空間に数値計算結果を可視化するための技術
- ▷ 加えて、時間軸を数値計算に活用するアニメーション技術
- ▷ さらに、色空間を数値計算に活用する色管理技術

☞ 実は、どのツールキットを使ってもグラフィックスの基本的な操作はおよそ同じである。

SDLによる（例題１）の解答例 1/2

```
#include <SDL.h>
#include <SDL_gfxPrimitives.h>
        :
struct view {
  int w, h;
  const SDL_VideoInfo *info;
  SDL_Surface *screen;
  SDL_Event event;
};
void view_init(struct view *v)
{
  if (SDL_Init(SDL_INIT_VIDEO) < 0)
    exit(1);
  v->info = SDL_GetVideoInfo();
  v->screen = SDL_SetVideoMode(v->w, v->h, v->info->vfmt->BitsPerPixel, 0);
}
        :
void view_loop(struct view *v)
{
  while (!0) {
    SDL_WaitEvent(&v->event);
    switch (v->event.type) {
    case SDL_QUIT: return; break;
    }
  }
}
void view_term(struct view *v) { SDL_Quit(); }
```

SDL による（例題１）の解答例 2/2

```
        :
void view_draw(struct view *v, void *o)
{
  struct deset *des = (struct deset *)o;
  int i, j;
  int x, y;

  if (SDL_MUSTLOCK(v->screen))
    if (SDL_LockSurface(v->screen) < 0) return;
  SDL_FillRect(v->screen, NULL,
               SDL_MapRGBA(v->screen->format, 255, 255, 255, 255));
  for (i=0; i<50000; i++) {
    des->de->map(des->p, i, des->v0, des->v1);
    x = (des->v0[0] - des->lb[0])/(des->ub[0] - des->lb[0])*v->w;
    y = (des->v0[1] - des->ub[1])/(des->lb[1] - des->ub[1])*v->h;
    pixelRGBA(v->screen, x, y, 0, 0, 0, 255);
    for (j=0; j<des->de->N; j++)
      des->v0[j] = des->v1[j];
  }
  if (SDL_MUSTLOCK(v->screen))
    SDL_UnlockSurface(v->screen);
  SDL_UpdateRect(v->screen, 0, 0, 0, 0);
}
        :
```

SDL による（例題３）の結果

Mac OS X のスクリーンショット機能で PNG 形式にて撮影、そして ImageMagick の convert コマンドで EPS 形式へ

```
$ convert filename.png filename.eps
```