図形描画ライブラリィ fplot

桂田祐史

1998年5月20日, 2009年6月4日

目 次

(必要なファイルは `/usr/local/meiji/text/katsurada/fplot/` にある。)

1 fplot ライブラリィ, ccx コマンド, f77x コマンド

FORTRAN プログラムや C プログラムから利用できる、図形描画用のサブルーチン・ライブラリィ **fplot**[1]を紹介する。

6701 号室のワークステーションでは、C プログラムに対しては "**ccx**", FORTRAN プログラムに対しては "**f77x**" というコマンドを使うことで、この fplot ライブラリィがリンクされる[2]

[1] この名前の頭文字は、最初にプログラムを書いた (桂田の後輩の) 藤尾秀洋君の姓、座標のデータ型 (floating point numbers) という 2 つの "f" にかけてある。

[2] 東海林先生の講義では `f77g`, `ccg` というコマンドが紹介されているようだが、`f77x`, `ccx` はその兄弟分にあたる。ほぼ互換性があると考えて良い。

例えば heat.c という C プログラムをコンパイルするには

```
oyabun% ccx heat.c
```

とする。heat.c に誤りがなければ heat という名前の実行形式のプログラムが出来る。

例えば reidai1.f というプログラムをコンパイルするには

```
oyabun% f77x reidai1.f
```

とする。エラーがなければ reidai1 という名前の実行形式のプログラムが出来る。

2 関数・サブルーチン一覧

fplot には以下の関数・サブルーチンが含まれている。以下の引数 (x0, y0, r 等) は文字列である s を除いて単精度実数型 (C の float, FORTRAN の real) です[3]。

openpl() fplot ライブラリィの初期化をする。

closepl() fplot ライブラリィの後始末をする。

erase() 画面をクリアする。

fspace(x0,y0,x1,y1) 左下端が (x0, y0), 右上端が (x1, y1) である長方形が描けるように座標を割り当てる。この際、縦横の拡大比が等しくなるような調節が行なわれる。 x0<x1, y0<y1 でなければならない。

fspace2(x0,y0,x1,y1) スクリーンの左下端を (x0, y0), 右上端を (x1,y1) とするように座標を割り当てる (fspace() のような調節は行なわない)。この関数を利用した場合は fcircle() や farc() は使えない。 x0<x1, y0<y1 でなければならない。

label(s) 現在点に文字列を表示する。s は文字列。s は "Hungry?", "Cup noodle!" などの文字列 (残念ながら日本語は使えません)。

linemod(s) 線分のパターンを指定する。s として指定できるのは"dotted", "solid", "longdashed", "shortdashed" と"dotdashed" である[4]。

fline(x1,y1,x2,y2) 点 (x1,y1) から点 (x2,y2) までの線分を描く。現在点は (x2,y2) となる。

fcircle(x,y,r) 点 (x,y) を中心とする半径 r の円を描く。現在点は (x,y) となる。

farc(x,y,x0,y0,x1,y1) 中心が (x,y) の円弧 (始点 (x0, y0), 終点 (x1,y1)) を描く。描画は反時計回りに行なわれる。

fmove(x,y) 現在点を (x,y) に変更する。

fcont(x,y) 現在点から点 (x,y) まで線分を描く。現在点は (x,y) になる。

[3]伝統的な C 言語との互換性を考慮して、実際には引数は float ではなく、double 型で渡すようにしてある。ANSI 規格の C 処理系を使う場合は注意が必要である。

[4]今のところ solid 以外は全部ただの点線になってしまう。そのうちきちんと描き分けるつもり、、、

fpoint(x,y) 点 (x,y) に点を描き、そこを現在点とする。

座標を指定するのに倍精度実数型 (C では double, FORTRAN では double precision あるいは real*8 と宣言する) を使うことも出来るように、先頭の文字が "d" のサブルーチン (dspace, dline, dcircle, darc, dmove, dcont, dpoint) も用意してある。使い方は、先頭の文字が "f" であるものに準じる。

以下にあげるサブルーチンは、少し特殊なものなので、最初のうちは無視してしまっても構わない (最初の二つのうちのどちらかは、お世話になるかも知れない)。

xflush() それまで発行した描画命令を実際に X サーバーに送り出す (X のリクエスト・バッファーをフラッシュする)。アニメーションで切りの良いところで実行すると動きが滑らかになる。

xsync() それまで発行した描画命令を完全に実行し終わる (実際にウィンドウに図形が表示される) まで待つ。アニメーションで切りの良いところで実行すると動きが滑らかになる (少し遅くなるけど)。

fmark(x,y) 点 (x,y) にマーカーを描き、そこを現在点とする。

xor() 今後二重に描いたところは消すようにする。図形の部分消去を実現できる。元に戻すには call set とする。

3 図の印刷法

fplot には次の関数もある。

mkplot(s) 現在までに描いた図を plot(5) 形式でファイルに出力する。ファイルの名前は文字列 s で指定する。

つまり mgraph コマンドが出力するような形式でデータが出力されるので、xplot や plot2ps を利用して、画面に図を最表示したり、プリンターで印刷したり出来る。

例えば call mkplot("heat.plot") として出力したファイル heat.plot の内容を画面に表示するには

```
oyabun% cat heat.plot | xplot
```

プリンターに印刷するには

```
oyabun% cat heat.plot | plot2ps | lpr
```

とする。

複数の図を一枚の紙に印刷するには、直接印刷しないで

```
oyabun% cat heat.plot | plot2ps > heat.ps
```

のようにして heat.ps のような PostScript データを作っておいてから multi コマンドを利

用する (次の例では一枚の紙に 4 つの図を収めるようにしている。heat5.ps は 2 枚目の紙に出力される。):

```
oyabun% multi -m 4 heat.ps heat2.ps heat3.ps heat4.ps heat5.ps | lpr
```

4 例題による解説

4.1 1 変数関数のグラフ

例題 1: $y = \sin x + \sin 3x + \sin 5x$ のグラフを描きなさい。

Fortran の場合

```
* reidai1.f -- サブルーチン呼び出しによるグラフ描き
      program reidai1
*      変数の宣言
*      a: 区間の左端の座標, b: 区間の右端の座標, margin: 余白の大きさ
      real Pi,a,b,margin
      integer N,i
      real h,x,f
      external f
*
      Pi = 4.0 * atan(1.0)
      a = 0.0
      b = 2.0 * Pi
      margin = (b - a) / 20
*      x 軸の分割の仕方の決定
      write(*,*) ' x 軸の区間の分割数を入力して下さい'
      read(*,*) N
      h = (b - a) / N
*      グラフィックスの初期化
      call openpl
      call fspace2(a - margin, -3.0, b + margin, 3.0)
*       始点のセット
      call fmove(a, f(a))
*       グラフ上の点を順に結ぶ
      do i = 1, N
        x = a + i * h
        call fcont(x, f(x))
      end do
*       図形を記録する
      call mkplot("reidai1.plot")
*       グラフィックスの後始末
      call closepl
      end
************************************************
      real function f(x)
      real x
      f = sin(x) + sin(3.0 * x) + sin(5.0 * x)
      end
```

C の場合

```
/*
 * reidai1.c -- サブルーチン呼び出しによるグラフ描き
 *   How to compile: ccx reidai1.c
 */

#include <stdio.h>
#include <math.h>

main()
{
    /* 変数の宣言
     *  a: 区間の左端の座標, b: 区間の右端の座標, margin: 余白の大きさ
     */
    double Pi, a, b, margin;
    int N, i;
    double h, x, f();

    Pi = M_PI;
    a = 0.0;
    b = 2.0 * Pi;
    margin = (b - a) / 20;
    /* x 軸の分割の仕方の決定 */
    printf("x 軸の区間の分割数を入力して下さい: ");
    scanf("%d", &N);
    h = (b - a) / N;
    /* グラフィックスの初期化 */
    openpl();
    fspace2(a - margin, -3.0, b + margin, 3.0);
    /* 始点のセット */
    fmove(a, f(a));
    /* グラフ上の点を順に結ぶ */
    for (i = 1; i <= N; i++) {
        x = a + i * h;
        fcont(x, f(x));
    }
    /* 図形を記録する */
    mkplot("reidai1.plot");
    /* グラフィックスの後始末 */
    closepl();
}

double f(x)
double x;
{
    return sin(x) + sin(3.0 * x) + sin(5.0 * x);
}
```

全体が、主プログラム (Fortran では、program `reidai1`, C では関数 main()) と関数副プログラム `f` の 2 つの部分からなっている。主プログラムの中で呼び出している `openpl`, `fspace2`, `fmove`, `fcont`, `closepl` が fplot の命令である。

`openpl()` は他のすべてのグラフィックス命令に先立って呼び出す。これによって図を描くためのウィンドウがオープンされる。

`fspace2()` はウィンドウに座標を割り振るために用いる。

Fortran の場合	`call fspace2(x0,y0,x1,y1)`
C の場合	`fspace2(x0,y0,x1,y1);`

とするとウィンドウの左下隅が (x0, y0)、右上隅が (x1,y1) になる (当然 x0<x1, y0<y1 でなければならない)。扱う問題に応じて適当な値を見い出して呼び出す必要がある。次の fmove(), fcont() は後回しにして、最後の closepl() を先に解説しよう。これは openpl() と対になる命令で、図形描画を終了させて後始末をするものである。実行がここに到達すると、プログラムは利用者がマウスでウィンドウをクリックするのを待つ状態になる。

ここまで解説した openpl(), fspace2(), closepl() は実際には画面に何も描かないが、何時でも必要である。つまりプログラムは必ず

Fortran の場合

```
.........
call openpl
.........
call fspace2(x0,y0,x1,y1)
.........
実際の描画命令の列
.........
call closepl
.........
```

C の場合

```
.........
openpl();
.........
fspace2(x0,y0,x1,y1);
.........
実際の描画命令の列
.........
closepl();
.........
```

という形になる。

fmove() は「ペンを移動する (=move)」, fcont() は「線を引きながら (つなぎながら=continue) ペンを動かす」という命令である。つまり、これらの命令では仮想のペンを考えて、それを移動することにより図形を描画する[5]。ではコンパイルしてから実行してみよう。

Fortran の場合

```
oyabun% f77x reidai1.f
reidai1.f:
 MAIN reidai1:
        f:
oyabun% reidai1
  x 軸の区間の分割数を入力して下さい
100
oyabun%
```

C の場合

```
oyabun% ccx reidai1.c
oyabun% reidai1
x 軸の区間の分割数を入力して下さい:  100
oyabun%
```

[5]実は XY プロッターという、そのものズバリ、ペンを紙の上で動かして図形を描く装置があって、fplot ライブラリィはその動作原理をモデルにして作られた。これは UNIX に昔からある plot ライブラリィ関数がお手本になっている。

問題 1: reidai1.f や reidai1.c を修正して、軸や目盛を描くようにしなさい。軸を点線で描くなど工夫してみなさい。

4.2 簡単な動画

ここでは時間発展する系を表示するための簡単な動画 (アニメーション) に利用してみよう。

微分方程式を数値シミュレーションして、微分方程式の解を理解するには、動画にして見るのが有効なことが多く、ここで紹介するテクニック (というほど大したものではないが) はあちこちで役に立つ。

例題 2: 時刻 t, 位置 $x \in [0, 2\pi]$ における変移が $f(x,t) = \sin x \cos t + \sin 3x \cos 3t + \sin 5x \cos 5t$ で与えられる弦の振動を描け[6]。

現代では常識になっていることだが、実際に連続的に図を変化させなくても、十分素早く図を切り変えれば「連続的に動いている」ように見える。適当な時間間隔 Δt を定めて、時刻 $t_j = j\Delta t$ $(j = 0, 1, 2, \cdots)$ における x の関数 $f(x, t_j)$ のグラフを次々に描けばよいであろう。以下では、古いグラフを消すために erase() を用いている。

```
* reidai2.f -- 簡単な動画（紙芝居？）
      program rei2
*      変数の宣言
      integer i,N,j,Jmax
      real Pi,a,b,margin
      real Time,h,dt,t,x,f
      external f
*      Pi=円周率, a=区間の左端, b=区間の右端, margin=余白
      Pi = 4.0 * atan(1.0)
      a = 0.0
      b = 2.0 * Pi
      margin = (b - a) / 20
*      x 軸の区間の分割数、追跡時間の入力
      write(*,*) ' x 軸上の区間の分割数='
      read(*,*) N
      write(*,*) ' 追跡時間='
      read(*,*) Time
*      区間の刻み幅
      h = (b - a) / N
*      グラフィックスの初期化
      call openpl
      call fspace2(a - margin, -2.0, b + margin, 2.0)
*      時間間隔を決め、ループの回数を計算する
      dt = 1.0 / 16.0
      Jmax = Time / dt + 0.5
******** メイン・ループ ********
      do j=0,Jmax
        t = j * dt
        call erase
        call fmove(a, f(a,t))
        do i = 1, N
          x = a + i * h
          call fcont(x, f(x,t))
```

[6] これは両端を固定された弦の、ある 3 つの固有振動の和である。特に $t = 0$ の瞬間には例題 1 の関数になっている。

```
        end do
        call xflush
      end do
****** メイン・ループ終り ******
*      グラフィックスの後始末
      call closepl
      end
****************************************
*      両端を固定された弦のある 3 つの固有振動の和
*      時刻 t=0 では、例題 1 に現われる関数に一致する
      real function f(x,t)
      real x,t
      f = sin(x)*cos(t)+sin(3*x)*cos(3*t)+sin(5*x)*cos(5*t)
      end
```

```
/*
 * reidai2.c -- 簡単な動画（紙芝居？）
 *  How to compile: ccx reidai2.c
 */

#include <stdio.h>
#include <math.h>

main()
{
    /* 変数の宣言 */
    int i, N, j, Jmax;
    double Pi, a, b, margin;
    double Time, h, dt, t, x, f();
    /* Pi=円周率, a=区間の左端, b=区間の右端, margin=余白 */
    Pi = M_PI;
    a = 0.0;
    b = 2.0 * Pi;
    margin = (b - a) / 20;
    /* x 軸の区間の分割数、追跡時間の入力 */
    printf("x 軸上の区間の分割数=");
    scanf("%d", &N);
    printf("追跡時間=");
    scanf("%lf", &Time);
    /* 区間の刻み幅 */
    h = (b - a) / N;
    /*  グラフィックスの初期化 */
    openpl();
    fspace2(a - margin, -2.0, b + margin, 2.0);
    /* 時間間隔を決め、ループの回数を計算する */
    dt = 1.0 / 16.0;
    Jmax = Time / dt + 0.5;
    /* ******* メイン・ループ ******* */
    for (j = 0; j <= Jmax; j++) {
        t = j * dt;
        erase();
        fmove(a, f(a, t));
        for (i = 1; i <= N; i++) {
            x = a + i * h;
            fcont(x, f(x, t));
        }
        xflush();
    }
```

```
    /* ***** メイン・ループ終り ***** */
    /* グラフィックスの後始末 */
    closepl();
}

/*
 * 両端を固定された弦のある 3 つの固有振動の和
 * 時刻 t=0 では、例題 1 に現われる関数に一致する
 */

double f(x, t)
double x, t;
{
    return sin(x) * cos(t) + sin(3 * x) * cos(3 * t) + sin(5 * x) * cos(5 * t);
}
```

これもコンパイルして実行してみよう。

Fortran の場合

```
oyabun% f77x reidai2.f
reidai2.f:
 MAIN rei2:
        f:
oyabun% reidai2
  x 軸上の区間の分割数=
100
  追跡時間=
6.28                    ← (一周期分)
oyabun%
```

C の場合

```
oyabun% ccx reidai2.c
oyabun% reidai2
x 軸上の区間の分割数=100
追跡時間=6.28
oyabun%
```

長い追跡時間を指定した場合は、停止させたくなったときに C-C (コントロール C) を打てば止めることができる。

問題 2: 自分で何か動画を作ってみよ。例えば

(a) 振り子の運動

(b) はずむボール

5　サンプル・プログラム (解説抜き)

5.1　定数係数線型常微分方程式の相図

連立常微分方程式の初期値問題

$$\begin{cases} \dfrac{dx}{dt} &= ax + by \\ \dfrac{dy}{dt} &= cx + dy \end{cases} \quad (t \in \mathbf{R}), \qquad \begin{cases} x(0) &= x_0 \\ y(0) &= y_0 \end{cases}$$

(a, b, c, d, x_0, y_0 は既知定数) を Runge-Kutta 法で解いて、相図を描く。

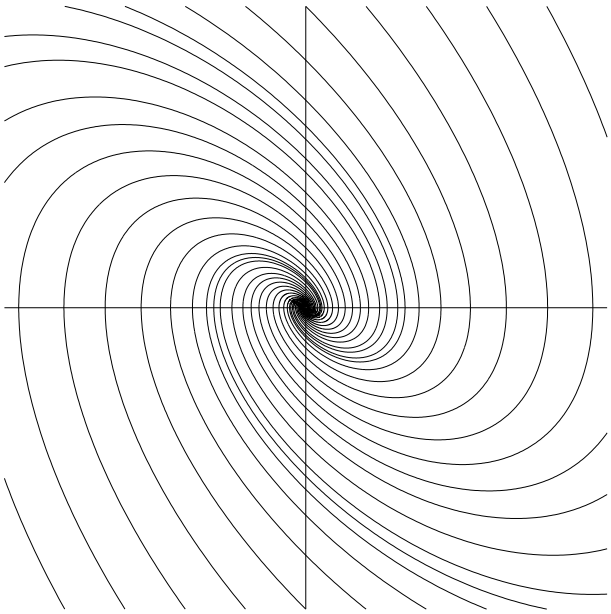
```
/*
 * dynamics.c
 */

/*****************************************************************************
 *****************************************************************************
 *
 *   2 次元の定数係数線形常微分方程式
 *       x'(t) = a x + b y
 *       y'(t) = c x + d y
 *  に初期条件
 *       x(0)=x0
 *       y(0)=y0
 *  を課した常微分方程式の初期値問題を解いて、相図を描く。
 *
 *  このプログラムは次の 4 つの部分から出来ている。
 *     main()
 *       主プログラム
 *       行列の係数の入力、ウィンドウのオープン等の初期化をした後に、
 *       ユーザーにメニュー形式でコマンドを入力してもらう。
 *       実際の計算のほとんどは他の副プログラムに任せている。
 *     draworbit(x0,y0,h,tlimit)
 *       (x0,y0) を初期値とする解の軌道を描く。
 *       刻み幅を h、追跡時間を tlimit とする。
 *       近似解の計算には Runge-Kutta 法を用いる。
 *     fx(x,y)
 *       微分方程式の右辺の x 成分
 *     fy(x,y)
 *       微分方程式の右辺の y 成分
 *****************************************************************************/

#include <stdio.h>
#include <math.h>

/* 係数行列 A の成分（common 文により function fx,fy と共有する） */
double a, b, c, d;
/* ウィンドウに表示する範囲（common 文により draworbit と共有する） */
double xleft, xright, ybottom, ytop;

main()
{
    /* 初期値 */
    double x0, y0;
    /* 時間の刻み幅、追跡時間 */
```

```
double h, tlimit, newh, newT;
/* メニューに対して入力されるコマンドの番号 */
int cmd;
/* マウスのボタンの状態 */
int lbut, mbut, rbut;
/* ウィンドウに表示する範囲の設定 */
xleft = -1.0;
xright = 1.0;
ybottom = -1.0;
ytop = 1.0;
/* 時間刻み幅、追跡時間（とりあえず設定） */
h = 0.01;
tlimit = 10.0;
/* 行列の成分を入力 */
printf(" a,b,c,d=");
scanf("%lf %lf %lf %lf", &a, &b, &c, &d);
/* ウィンドウを開く */
openpl();
fspace2(xleft, ybottom, xright, ytop);
/* x 軸、y 軸を描く */
fline(xleft, 0.0, xright, 0.0);
fline(0.0, ybottom, 0.0, ytop);
xsync();
/* メイン・ループの入口 */
while (1) {
    /* メニューを表示して、何をするか、番号で選択してもらう */
    printf("したいことを番号で選んで下さい。\n");
    printf(" -1:メニュー終了, 0:初期値のキー入力, 1:初期値のマウス入力,");
    printf("  2:刻み幅 h, 追跡時間 T 変更 (現在 h=%7.4f, T=%7.4f)\n",
        h, tlimit);
    scanf("%d", &cmd);
    /* 番号 cmd に応じて、指示された仕事をする */
    if (cmd == 0) {
        /* 初期値の入力 */
        printf(" 初期値 x0,y0=");
        scanf("%lf %lf", &x0, &y0);
        draworbit(x0, y0, h, tlimit);
    } else if (cmd == 1) {
        while (1) {
            printf("マウスの左ボタンで初期値を指定して下さい（右ボタンで中止)\n");
            fmouse(&lbut, &mbut, &rbut, &x0, &y0);
            if (lbut == 1) {
                printf("(x0,y0)=(%g,%g)\n", x0, y0);
                draworbit(x0, y0, h, tlimit);
            } else if (mbut == 1) {
                printf("(x0,y0)=(%g,%g)\n", x0, y0);
                draworbit(x0, y0, -h, tlimit);
            } else {
                printf("マウスによる初期値の入力を打ち切ります。\n");
                break;
            }
        }
    } else if (cmd == 2) {
        /* 時間刻み幅、追跡時間の変更 */
        printf(" 時間刻み幅 h, 追跡時間 T= ");
        scanf("%lf %lf", &newh, &newT);
        if (newh != 0 && newT != 0) {
            h = newh;
```

```
                tlimit = newT;
                printf("新しい時間刻み幅 h = %g, 新しい追跡時間 T = %g\n",
                    h, tlimit);
            } else {
                printf(" h=%g, T=%g は不適当です。\n", newh, newT);
            }
        } else if (cmd == -1) {
            /* 終了 -- メイン・ループを抜ける */
            break;
        }
    }
    mkplot("dynamics.plot");
    printf("fplot ウィンドウを左ボタンでクリックして下さい\n");
    closepl();
    return 0;
}

/*************************************************************************/
/*************************************************************************/
/*      指示された初期値に対する解軌道を描く */
draworbit(x0, y0, h, tlimit)
double x0, y0, h, tlimit;
{
    double x, y, fx(), fy();
    double k1x, k1y, k2x, k2y, k3x, k3y, k4x, k4y, t;
    /* 時刻を 0 にセットする */
    t = 0.0;
    /* 初期値のセット */
    x = x0;
    y = y0;
    /* 初期点を描く */
    fpoint(x, y);
    /* ループの入口 */
    do {
        /* Runge-Kutta 法による計算 */
        /* k1 の計算 */
        k1x = h * fx(x, y);
        k1y = h * fy(x, y);
        /* k2 の計算 */
        k2x = h * fx(x + k1x / 2.0, y + k1y / 2.0);
        k2y = h * fy(x + k1x / 2.0, y + k1y / 2.0);
        /*  k3 の計算 */
        k3x = h * fx(x + k2x / 2.0, y + k2y / 2.0);
        k3y = h * fy(x + k2x / 2.0, y + k2y / 2.0);
        /*  k4 の計算 */
        k4x = h * fx(x + k3x, y + k3y);
        k4y = h * fy(x + k3x, y + k3y);
        /*  (Xn+1,Yn+1) の計算 */
        x += (k1x + 2.0 * k2x + 2.0 * k3x + k4x) / 6.0;
        y += (k1y + 2.0 * k2y + 2.0 * k3y + k4y) / 6.0;
        /* 解軌道を延ばす */
        fcont(x, y);
        /* 時刻を 1 ステップ分進める */
        t += h;
        /* まだ範囲内かどうかチェック */
    }    while (abs(t) <= fabs(tlimit));
    /* サブルーチンを抜ける */
    /* 確実に描画が終るのを待つ */
```

```
    xsync();
}

/*********************************************************************/
/*********************************************************************/
/* 微分方程式の右辺のベクトル値関数 f の x 成分 */
double fx(x, y)
double x, y;
{
    return a * x + b * y;
}

/* 微分方程式の右辺のベクトル値関数 f の y 成分 */
double fy(x, y)
double x, y;
{
    return c * x + d * y;
}
```

マウスを使って入力できるのがちょっと面白い。



5.2 熱方程式の初期値境界値問題の可視化

熱方程式の初期値境界値問題

$$\begin{aligned} u_t &= u_{xx} \quad ((x,t) \in (0,1) \times (0,\infty)) \\ u(0,t) &= u_(1,t) = 0 \quad (t \in (0,\infty)) \\ u(x,0) &= f(x) \quad (x \in [0,1]) \end{aligned}$$

を差分法で解く。

```
/* heat1d.c -- 陰的スキーム (いわゆる θ 法) で熱方程式を解く
 *     空間 1 次元、同次 Dirichlet 境界条件の問題
 *
 *     「微分方程式と計算機演習」第 11 章 p237 のプログラムを修正・拡張したもの
```

```
 C     ** IMPLICIT FINITE DIFFERENCE METHOD **
 C     **         FOR HEAT EQUATION         **
 C     nfunc : 関数の番号（5 種類の関数を用意している）
 C     theta : スキームのパラメーター
 C     N : 分割の数
 C     t : 時刻
 C     tau : 時間の刻み幅
 C     Tmax : 時刻の上限
 C     h : 空間の刻み幅
 C     AL,AD,AU : 係数行列
 C     u : DIMENSION FOR UNKNOWN FUNCTION                      */

/*
 *   このプログラムのコピーが欲しい場合は
 *      cp /usr/local/meiji/D97/heat1d.c .
 * とすればよい。
 *
 * 細かい拡張
 *   1) 区間の左端、右端の座標を変数 a, b に設定するようにした。
 *   2) グラフを描くごとに、それまでに描いたグラフを消去するかどうか、
 *      変数 erase_always で制御するようにした。
 *   3) 計算した数値データを表示するか変数 print_always で制御するよう
 *      にした。
 *   4) 何ステップおきにグラフを描き換えるか、数値データを表示するか、
 *      変数 skips で制御するようにした。skips == 1 だと、毎回グラフを
 *      書き換える。
 *   5) 初期条件を与える関数 f(x, nfunc) で、登録されている関数の種類
 *      を増やした。nfunc == 5 まである。特に nfunc == 4,5 は非対称な
 *      関数である。
 *
 */

#include <stdio.h>
#include <math.h>

#define ndim    1000

/* 円周率 (math.h にある M_PI の値を利用する
 * あるいは double PI; として、どこかで PI = 4.0 * atan(1.0); と代入
 */

#define PI  M_PI

main()
{
    int N, nfunc, i, j, Jmax;
    double a, b, theta, h, Tmax, tau, c1, c2, c3, c4, lambda, f(), t;
    double AL[ndim], AD[ndim], AU[ndim], ff[ndim], u[ndim + 1];
    double xleft, ybottom, xright, ytop;
    /* 何ステップおきにグラフを書き換えるか */
    int skips = 1;
    double dt;
    /* 以前描いたグラフを消すか？(0=No, 1=Yes) */
    int erase_always = 0;
    /* 数値を表示するか?(0=No, 1=Yes) */
    int print_always = 0;

    /* 区間の左端、右端 */
```

```
a = 0.0;
b = 1.0;

/* 入力 */
printf("入力して下さい : nfunc(1..5)=");
scanf("%d", &nfunc);
printf("入力して下さい : θ=");
scanf("%lf", &theta);
printf("入力して下さい : N(<=%d)=", ndim);
scanf("%d", &N);
if (N <= 1 || N > ndim)
    return 0;
printf("入力して下さい : λ=");
scanf("%lf", &lambda);

h = (b - a) / N;
tau = lambda * h * h;
printf(" 時間の刻み幅τ= %g になりました。\n", tau);

printf("入力して下さい : Tmax=");
scanf("%lf", &Tmax);

printf("図を描く時間間隔 Δ t は : ");
scanf("%lf", &dt);
skips = (dt + 0.1 * tau) / tau;

/* 係数行列の計算 */
c1 = - theta * lambda;
c2 = 1.0 + 2.0 * theta * lambda;
c3 = 1.0 - 2.0 * (1.0 - theta) * lambda;
c4 = (1.0 - theta) * lambda;
for (i = 1; i < N; i++) {
    AL[i] = c1;
    AD[i] = c2;
    AU[i] = c1;
}

/* LU 分解する */
trilu(N-1, AL+1, AD+1, AU+1);

/* 初期値 */
for (i = 0; i <= N; i++)
    u[i] = f(a + i * h, nfunc);

/* 出力 (t=0) */
/* 出力 (ここでは画面に数値を表示すること) は Fortran と C で
   かなり異なるので、直接の翻訳は出来ない。ごたごたするので、
   print100 という関数にまとめた */
t = 0.0;
print100(N, t, u);

/* グラフィックス画面の準備 -- まず画面の範囲を決めてから */
xleft  = a - (b - a) / 10;
xright = b + (b - a) / 10;
ybottom = - 0.1;
ytop    = 1.1;

/* fplot ライブラリィの呼び出し */
```

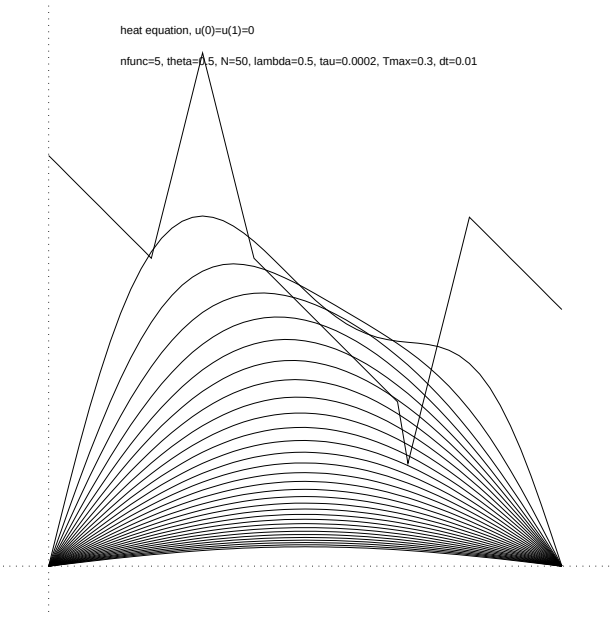
```
    openpl();
    fspace2(xleft, ybottom, xright, ytop);

    /* パラメーター、入力データ等の表示 */
    print_data(xleft, ybottom, xright, ytop,
        nfunc, theta, N, lambda, tau, Tmax, t, dt);

    /* 解 u の t=0 におけるグラフを描く */
    fmove(a, u[0]);
    for (i = 1; i <= N; i++)
        fcont(a + i * h, u[i]);
    xsync();

    Jmax = (Tmax + 0.1 * tau) / tau;

    /* 繰り返し計算 */
    for (j = 1; j <= Jmax; j++) {

        /* 連立一次方程式を作って解く */
        for (i = 1; i < N; i++)
            ff[i] = c3 * u[i] + c4 * (u[i - 1] + u[i + 1]);
        trisol(N-1, AL+1, AD+1, AU+1, ff+1);

        /* 求めた値を u[] に収める */
        for (i = 1; i < N; i++)
            u[i] = ff[i];
        u[0] = 0.0;
        u[N] = 0.0;

        /* 時刻の計算 */
        t = j * tau;

        if (j % skips == 0) {
            /* 数値データの表示 */
            if (print_always)
                print100(N, t, u);

            /* 解のグラフを描く */
            if (erase_always) {
                /* これまで描いたものを消し、パラメーターを再表示する */
                erase();
                print_data(xleft, ybottom, xright, ytop,
                    nfunc, theta, N, lambda, tau, Tmax, t, dt);
            }
            fmove(a, u[0]);
            for (i = 1; i <= N; i++)
                fcont(a + i * h, u[i]);
            xsync();
        }
    }

    save_picture();

    printf("終了したければ、ウィンドウをマウスでクリックして下さい。\n");
    closepl();
    return 0;
}
```

```
/*********************************************************/

/* 初期条件を与える関数。複数登録して番号 nfunc で選択する。 */

double f(x, nfunc)
double x;
int nfunc;
{
    /* f(x)=max(x,1-x) */
    if (nfunc == 1) {
        if (x <= 0.5)
            return x;
        else
            return 1.0 - x;
    }
    /* f(x)=1 */
    else if (nfunc == 2)
        return 1.0;
    else if (nfunc == 3)
        return sin(PI * x);
        /* f(x)= 変な形をしたもの (by M.Sakaue) */
    else if (nfunc == 4) {
        if (x <= 0.1)
            return 5 * x;
        else if (x <= 0.3)
            return -2 * x + 0.7;
        else if (x <= 0.5)
            return 4.5 * x - 1.25;
        else if (x <= 0.7)
            return - x + 1.5;
        else if (x <= 0.9)
            return x + 0.1;
        else
            return -10 * x + 10.0;
    }
    /* やはり非対称な形をしたもの (by M.Sakaue) */
    else if (nfunc == 5) {
        if (x <= 0.2)
            return -x + 0.8;
        else if (x <= 0.3)
            return 4 * x - 0.2;
        else if (x <= 0.4)
            return -4 * x + 2.2;
        else if (x <= 0.7)
            return - x + 1.0;
        else if (x <= 0.8)
            return 4 * x - 2.6;
        else
            return -x + 1.5;
    }
}

/*********************************************************/

/* 配列 u[] の内容 (u[0],u[1],...,u[n]) を一行 5 個ずつ表示する。 */

print100(N, t, u)
int N;
```

```
double t, u[];
{
    int i;
    printf("T= %12.4e\n", t);
    for (i = 0; i < 5; i++)
        printf("  I     u(i)    ");
    printf("\n");
    for (i = 0; i <= N; i++) {
        printf("%3d%12.4e ", i, u[i]);
        if (i % 5 == 4)
            printf("\n");
    }
    printf("\n");
}

/*********************************************************/

/* ウィンドウに座標軸を表示する */
axis(xleft, ybottom, xright, ytop)
double xleft, ybottom, xright, ytop;
{
    linemod("dotted");
    fline(xleft, 0.0, xright, 0.0);
    fline(0.0, ybottom, 0.0, ytop);
    linemod("solid");
}

/* ウィンドウ内の位置 (x,y) から文字列 s を表示する */
fsymbol(x, y, s)
double x, y;
char s[];
{
    fmove(x, y);
    label(s);
}

#define X(x)    (xleft+(x)*(xright-xleft))
#define Y(y)    (ybottom+(y)*(ytop-ybottom))

/* パラメーターの値等をウィンドウに表示する */
print_data(xleft, ybottom, xright, ytop,
nfunc, theta, N, lambda, tau, Tmax, t, dt)
double xleft, ybottom, xright, ytop, theta, lambda, tau, Tmax, t, dt;
int nfunc, N;
{
    char message[80];

    axis(xleft, ybottom, xright, ytop);
    sprintf(message, "heat equation, u(0)=u(1)=0");
    fsymbol(X(0.2), Y(0.95), message);
    sprintf(message,
        "nfunc=%d, theta=%g, N=%d, lambda=%g, tau=%g, Tmax=%g, dt=%g",
        nfunc, theta, N, lambda, tau, Tmax, dt);
    fsymbol(X(0.2), Y(0.9), message);
    sprintf(message, "t = %g", t);
    if (t != 0.0)
        fsymbol(X(0.2), Y(0.85), message);
}
```

```
/* 現在ウィンドウに表示されている図をファイルに保存する */
save_picture()
{
    char filename[200];
    printf("図を保存するファイル名: ");
    scanf("%s", filename);
    mkplot(filename);
}

/************************************************************
 ************************************************************
 ************************************************************/

/*
 *  三重対角行列に対する LU 分解に基づく連立 1 次方程式の求解
 *
 *  (Gauss の消去法を用いているが、ピボットの選択等はしていないので、
 *   係数行列が正定値対称行列でないと結果は保証されない。)
 *
 *  n は未知数の個数。
 *  f は最初は連立 1 次方程式の右辺のベクトル b。最後は解ベクトル x。
 *   (添字は 0 から。i.e. b[0],b[1],...,b[n-1] にデータが入っている。)
 *
 *  AL, AD, AU は係数行列の
 *     下三角部分 (lower part)
 *     対角部分 (diagonal part)
 *     上三角部分 (upper part)
 *  つまり
 *
 *  AD[0] AU[0]   0   ................. 0
 *  AL[1] AD[1] AU[1]   0  ............. 0
 *    0   AL[2] AD[2] AU[2]  0 ......... 0
 *          ..................
 *                    AL[n-2] AD[n-2] AU[n-2]
 *                      0     AL[n-1] AD[n-1]
 *
 *  ここで AL[0], AU[n-1] は意味がないことに注意。
 */

/* 三項方程式 (係数行列が三重対角である連立一次方程式のこと) を解く
 *   入力
 *     n: 未知数の個数
 *     al,ad,au: 連立一次方程式の係数行列
 *       (al: 対角線の下側、 ad: 対角線、 au: 対角線の上側)
 *        al[i] = A_{i,i-1}, ad[i] = A_{i,i}, au[i] = A_{i,i+1},
 *        al[0], au[n-1] は意味がない)
 *        f: 連立一次方程式の右辺の既知ベクトル b
 *   出力
 *     al,ad,au: 入力した係数行列を LU 分解したもの
 *     f: 連立一次方程式の解 x
 *   能書き
 *     一度 call すると係数行列を LU 分解したものが返されるので、
 *     以後は同じ係数行列に関する連立一次方程式を解くために、
 *     サブルーチン trisol が使える。
 *   注意
 *     ピボットの選択をしていないので、係数行列が正定値である
 *     などの適切な条件がない場合は結果が保証できない。
```

```
 */

/* 三項方程式を解く */
trid(n,al,ad,au,f)
int n;
double al[],ad[],au[],f[];
{
    trilu(n,al,ad,au);
    trisol(n,al,ad,au,f);
}

/* 三重対角行列の LU 分解 (pivoting なし) */
trilu(n,al,ad,au)
int n;
double al[],ad[],au[];
{
    int i, nm1 = n - 1;
    /* 前進消去 (forward elimination) */
    for (i = 0; i < nm1; i++) ad[i + 1] -= au[i] * al[i + 1] / ad[i];
}

/* LU 分解済みの三重対角行列を係数に持つ三項方程式を解く */
trisol(n,al,ad,au,f)
int n;
double al[],ad[],au[],f[];
{
    int i, nm1 = n - 1;
    /* 前進消去 (forward elimination) */
    for (i = 0; i < nm1; i++) f[i + 1] -= f[i] * al[i + 1] / ad[i];
    /* 後退代入 (backward substitution) */
    f[nm1] /= ad[nm1];
    for (i = n - 2; i >= 0; i--) f[i] = (f[i] - au[i] * f[i + 1]) / ad[i];
}
```



6 ccx, f77x の正体

ともにシェル・スクリプトである。

/usr/local/meiji/bin/f77x

```
#!/bin/csh
g77 -O -o $1:r $* -I/usr/local/include -L /usr/local/lib -R /usr/local/lib -lfplot -lX
```

/usr/local/meiji/bin/ccx

```
#!/bin/csh
gcc -O -o $1:r $* -I/usr/local/include -L /usr/local/lib -R /usr/local/lib -lfplot -lX
```

7 TEX への取り込み

1. 関数 mkplot() を使って、plot(5) フォーマットで記録する。

```
mkplot("mygraph.plot");
```

2. plot2ps コマンドを使って、PostScript ファイルに変換する。

```
oyabun% plot2ps mygraph.plot > mygraph.ps
```

3. /usr/local/meiji/bin/toeps コマンドを使って、BoundingBox を埋め込む。

```
oyabun% toeps mygraph.ps
```

4. LATEX では、eclepsf スタイルを読み込む。

   LATEX 209 の場合 \documentstyle[eclepsf] のようにする。

   LATEX 2ε の場合 \usepackage[eclepsf] のようにする。

5. 図を取り込みたいところで \epsfile{file=ファイル名}

```
\epsfile{file="mygraph.ps",width=8cm}
```

8 ccg の関数 fsymbol() の実現法

東海林先生の講義で使った ccg コマンドでリンクされるライブラリィには、fsymbol() という関数があるが、ccx にはない。これが使いたい場合は、次のようにすれば良い。

```
fsymbol(x, y, s)
double x, y;
char s[];
{
    fmove(x, y); label(s);
}
```