Calcompインターフェース対応ライブラリ

寒川　光

芝浦工業大学　電子情報システム学科

目 次

数値計算プログラムの実行結果を可視化するために，グラフや変形図を描く目的で古くから Calcomp インターフェースが用いられた．ペンプロッタに対する古典的なインターフェースで，有限差分法や有限要素法などの数値解析の教科書のサンプルプログラムにはこれを用いて作図する例が紹介されている [1]．本書のサンプルプログラムの出図でも Calcomp インターフェースを用いるが，ここでその最終的な表示方法の例として，PostScript ファイルの生成法を説明する[1]．HPC プログラミングの特徴は，**陽に見えるプログラミングの世界よりも下位の階層の機能をブラックボックス化することなく，コンパイラの機能を通してハードウェアの動きまで考慮して高速化する**ところにある．作画にもこのプログラミングの姿勢を当て嵌めると，言語処理系を通して表示装置や印刷装置（ハードウェア）の動きを意識すること，と言うことができる．本節では，プロッタや印刷機などの装置とプログラミング言語や計算機モデルの関係に対しても関心を抱くという問題意識で，作画プログラミングの世界を覗いてみたい．1 章に印刷装置とそれに対するインターフェース，2 章で Calcomp インターフェースの PostScript 用ライブラリ calcomps.c，3 章で Calcomp インターフェースの X Window（X 窓）用ライブラリ calcompx.c について説明する．なお例題として plotest.f，ball.f，starpalette.f，PlotLens.f を用いる．

1　印刷装置の仕様とインターフェース

初期のペンプロッタは磁気テープのフレーム（6 または 8 ビット）の情報を単位として動作する単純な機械であった．小容量の Read Only Memory（ROM）が使用できるようになると，表示装置のインテリジェンスが向上し，英数字フォントがプロッタ側に実装された．Calcomp インターフェースはこの時代（1970 年代）のものである．当時の大型計算機（メインフレーム）では漢字印刷のためにレーザプリンタを使用していたが，まだ高価だったため，作画にはペンプロッタが一般的であった．Adobe Systems 社から PostScript が発表されたのは 1984 年である．この時代になるとプログラミング理論も進歩してオブジェクト指向言語が登場する．コンピュータグラフィックスはオブジェクト指向が成功した分野でもある．レーザプリンタにマイクロプロセッサが組み込まれて，曲線を描いたり，曲線に囲まれた図形を塗りつぶすことが手軽にできるようになった．

Calcomp プロッタと PostScript が前提としたレーザープリンタではハードウェアの機能は大きく異なるが，後者の機能のほうが豊富なので，PostScript で Calcomp をエミュレートすることは簡単である．Calcomp インターフェースは PostScript 以外でも，X-Window や LaTeX など多種の装置を対象にするので，それらに共通する機能だけをインターフェースとして括りだし，それ以外の PostScript 独特の作画機能を利用したいところには PostScript 用にインターフェースを設ける（穴を開ける）ことにする．

1.1　ペンプロッタ

Calcomp（California Computer）社のドラム型プロッタ 565 型は 1959 年に出荷されたが，これは計算機の図形を出力する目的で販売された最初の製品だったかもしれない．ドラム型のペンプロッタはドラムに巻きつけられたロール紙を用いる．x 軸方向はドラムの回転で紙を移動させ，y 軸方向はドラムの軸方向をペンを移動させて線画を描く．ペンの上げ下げはソレノイドで紙にペンを押し付ける．ペンや紙の移動量はインクリメンタルで，その量は 0.01 インチであった．ペンだけが動けば y 方向，紙だけが動けば x 方向，ペンと紙の両方が同時に動けば x と y の中間 45 度の方向に，合計 8 方向の短い線分を描くことができる．磁気テープの 1 フレームの情報に対応させてペンと紙の移動とソレノイドを制御する．この動作が作画の最下位の機能単位なので，これが分解能となり解像度も低かった．後継機種ではインクリメン

[1] 作画は calcomps.c プログラムをコンパイルしてできる calcomp.o をサンプルプログラムにリンクして実行すれば，calcomp.ps ファイルを出力するので，これを GSview などで画面出力したり PostScript 対応の印刷機で印刷することができる．

ト量を細かくする方法と，移動方向を 24 方向に増加させる方法によって，解像度は向上した．Calcomp 社はこれらのプロッタに対する標準的なインターフェースを提供した[2].

初期のペンプロッタ　初期のインターフェースは「始点と終点の座標値を与えてその間を直線で結ぶ」. 1970 年代の代表的なプログラミング言語 Fortran で次のように呼び出すことで，ペン位置を現在点（カレントポイント）から指定点 (x, y) に移動するが，第 3 引数でペンを上げるか，下げるかを指定する.

```
call plot(x,y,2)      ペンダウンで (x,y) に移動
call plot(x,y,3)      ペンアップで (x,y) に移動
```

x, y は単精度浮動小数点数である.

文字を書くにも，フォントの形状をアプリケーション・プログラムで用意して，例えば “A” の文字を描く場合，文字配列に (0,0)，(3,6)，(5,2)，(1,2)，(5,2)，(6,0) の 6 点の座標値を持ち，6×6 のフォントを描く（A の横棒は往復して 2 度書き）という方法によっていた.

作画 “call plot(...)” に先立って次の初期化ルーチンを呼び出す.

```
call plots(n)         初期化，n は出力する磁気テープのユニット（装置）番号.
```

作画情報は磁気テープに書かれ，ユーザは磁気テープをオフラインでプロッタにマウントして作画した.

このほか，次の 2 種類を第 3 引数としてよく使用した.

```
call plot(x,y,–3)     原点を (x,y) に移動（再定義）
call plot(x,y,999)    終了
```

ペンプロッタはロール紙を用いたので，ページの概念がない．2 枚目の図を描く場合，ペンアップで次の図の原点位置に移動し，その位置を新たな原点として定義する.

拡大縮小は座標値の単位を変更することで行う.

```
call factor(s)        現行の単位を s 倍（s は単精度浮動小数点数）する.
```

またペンの現在位置を知ることができる.

```
call where(cx,cy)     (cx,cy) に現在点が返る
```

文字のサポート　プロッタ側にフォントを格納したメモリ（ROM）が実装されるようになると，“call symbol(...)” によって英数字が書けるようになった.

```
call symbol(x,y,'ABC')      (x, y) に “ABC” と描く
```

「指定点に文字列を置く」という動作である．指定点 (x, y) とフォントの位置合わせは，最初の文字の基準位置（文字の左下）が (x, y) に一致する．これでは格子点位置やグラフ上の特定の位置に印をつけるには使い辛いので，(x, y) を “◯” や “×” の中心に一致させる**センターシンボル**も用意された.

さらにペンプロッタ（ハードウェア）に複数のボールペンを切り替える機能が付くと，ライブラリにはこれをサポートするエントリが追加され，多色で作図できるようになった.

[2]IBM は Calcomp 565 を IBM 1627 として IBM 1620 計算機や 1130 計算機の付属品として販売したが，これは IBM が端末装置として販売した最初の非 IBM 製品だったかもしれない.

```
call newpen(ipen)       ペンを ipen に変更する
```

1.2 ページ記述言語 PostScript

PostScript はレーザプリンタ上で稼働するインタープリータに埋め込まれるページ記述言語で，BASIC のようにインタープリータ型（解釈型）言語として設計された．つまり，PostScript の演算子（実行制御，スタック操作，算術演算，論理演算，および描画などのオペレータ）は印刷機に搭載されるマイクロプロセッサによって実行される．プログラミング言語としての特徴は後置記法，スタック指向，辞書と呼ばれるデータ構造，インタープリータ型にある．これらの特徴は走査型出力装置上で図形オブジェクトを取り扱うのに適している．発売以来多くのイラストレータに愛用され，20 年以上前に設計されにもかかわらず現在でも様々な形で使用されている．この理由として，柔軟なページ記述言語であるため，作画ツール中に仮想デバイスを実現することが容易なことを指摘できる（本節でもそのような使い方をする）．

後置記法（postfix notation） 数学で用いられる記法には，演算子（関数）とその引数の並び方によって，前置記法，中置記法，後置記法の区別がある．前置記法は $\sin\theta$ や $f(x, y, \ldots)$ のように変数や引数の前に関数（演算子）が置かれる．中置記法は四則演算 9×9 のようにオペランドの間に演算子が置かれる．後置記法は演算子の前にすべての引数が置かれる．前置記法をポーランド記法，後置記法を逆ポーランド記法ということもある．電卓は部分的には後置記法が採用されている．たとえば $\sqrt{2}$ は “2”，“$\sqrt{}$” の順でキーを押す．四則演算 $2 \times 3 + (4 + 5)/2$ は後置記法では次のように表す．

```
2 3 mul 4 5 add 2 div add
```

後置記法のほうが計算機の処理系に負担をかけないので，インタープリータに適している．

スタック スタックは PostScript によって直ちに利用されるデータを入れておくために用意されたメモリの一部を指す．このメモリ領域はラストイン・ファーストアウト（LIFO）である．上記のコードを実行するプロセッサの側から見てみよう．まず最初のトークン 2 をスタックに置く．次に 3 が来ると 2 をスタックの 2 番目にプッシュしてスタックトップに 3 を置く．次に **mul** が来ると，これは 2 つの入力をとり 1 つを出力する算術演算オペレータなので，スタックの上 2 つにある 2 と 3 をスタックから取り出して加え，和 6 をスタックトップに置く．次に 4 が来ると 6 をプッシュしてスタックトップに 4 を置く[3]．

RISC 型のアーキテクチャの計算機では，乗算命令は通常 2 つないし 3 つのレジスタオペランドを持ち，命令にはレジスタの番号が指定される．PostScript ではスタックを番号のないレジスタと見て，2 つの入力オペランドが常に 0 番（スタックトップ）と 1 番（スタックの 2 番目），結果は 0 番の命令と考えてもよい．ただし通常の計算機はデータ型に合致した命令を実行しなと意味がないが，PostScript では変数の型はオブジェクトの属性として管理され，インタープリータが型変換を含めて演算してくれる．

スタック上のデータを LIFO でしか使用できないのでは窮屈なので，スタック操作オペレータが用意され，例えば **exch** はスタックトップと 2 番目を交換する命令である．

辞書とデータ構造 辞書はキーと値の組によってデータを格納した構造である．値として，数値，文字列などの他，PostScript 演算子，配列などを扱うことができ，配列には実行可能な手続きを登録することもできる．辞書に登録するためには **def** オペレータを用いる．def によって定義されたエントリのキーを

[3]次に 5 が来ると 4 と 6 をプッシュしてスタックトップに 5 を置く．add が来ると $4 + 5$ を計算して 9 をスタックトップに積む．2 が来ると 4 と 9 をプッシュしてスタックトップに 2 を積む．div が来ると $9 \div 2$ を求めて 4.5 をスタックトップに置く．add が来ると $6 + 4.5$ を求め， 10.5 をスタックトップに置く．

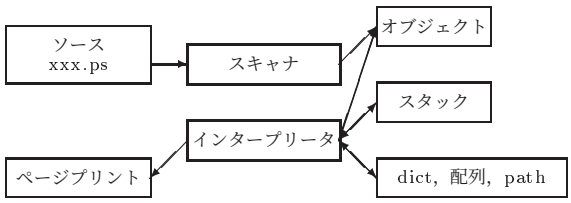

図 1: PostScript 言語の実行モデル（スキャナとインタープリータ）

PostScript インタープリータに与えると，名前検索が行われて対応する値オブジェクトが実行される．本節の例で “5 /Dim exch def” では，5 と /Dim がスタックに積まれ，exch で逆転されて “/Dim 5” としたところで def によって /Dim と 5 が結びつけられる．これ以降 Dim をオペランドとして指定すると，Dim に結びつけられた 5 が参照されるので，通常のプログラミング言語のように変数名によってデータを参照することができる．

インタープリータ型言語　実行は「スキャナ」による読込みと「インタープリータ」による実行の 2 段階に分かれる．スキャナはソースコードを読み込み，トークンを切り出し，オブジェクトを生成して，インタープリータに渡す．図 1 に PostScript 言語の実行モデルを示した．スキャナとインタープリータは交互に実行される．インタープリータを仮想計算機（Virtual Machine:VM）と呼ぶこともできる．

描画オペレータ　PostScript による描画は，カレントページと呼ばれる仮想画面上にパスを作成することから始められる．パスは直線や曲線の集合であり，**newpath** オペレータによって初期化され，**moveto**，**lineto**，**curveto** などの**パス構築オペレータ**によって延長されてゆく．“x y moveto” は現在点から指定された点 (x, y) への移動，“x y lineto” は現在点から指定点 (x, y) への直線である．**rmoveto** や **rlineto** は位置を現在点からの相対的な距離で指定する．“1 1 rlineto” は現在点から x 方向に 1 ポイント，y 方向に 1 ポイントの長さの直線を描く[4]．このようにパスは newpath によって開始されて，現在点に対するアクションとしてプログラミングされるが，**closepath** によって開始点まで延長して閉じた図形にすることもできる．パスは **stroke** オペレータによって塗られるか，**fill** オペレータによって閉じたパスの内側を塗りつぶされる．作画が完成したら **showpage** によってカレントページを完成する．

2　Calcomp インターフェースの PostScript 用ライブラリ

calcomps.c は，Calcomp インターフェースを経由して描かれた図形オブジェクト（直線や文字）を，PostScript の描画オペレータに変換して PostScript ファイルに出力するためのライブラリプログラムである．ペンプロッタの機能を「指定点に直線や文字を置くこと」と要約することができるが，この機能を出力ファイルのプロローグ部分に PostScript の手続きとして定義し，作画はこれを呼び出すコードによって実現する．手続きの定義（プロッタの機能）はライブラリプログラムを改訂すれば拡張することも容易なので，ペンプロッタの機能範囲も柔軟に決めることができる．

ライブラリは C 言語でプログラミングされているが，C または Fortran プログラムから呼び出し可能で，実行後は calcomp.ps ファイルを出力する．これを GSview などの表示プログラムで見たり，PostScript 対応の印刷機で印刷したり，PDF ファイルに変換して配布・印刷することができる．このように最終的な作画は PostScript インタープリータの解釈・実行によって行われるので，Calcomp インターフェースから出図までを階層化の観点で見るとかなり深くなっている．

[4] 1 ポイントは PostScript では 1 インチ（2.54cm）の 72 分の 1 で約 0.35mm である（ビッグポイント）．

2.1 ライブラリ calcomps.c

Calcomp のエントリは plots, factor, plot, where, symbol, newpen であるが, centsym, color, curve, arc, newpage, special を追加した. これらのルーチンで Icol, Ipage の Int 型および Fctr, Curx, Cury の float 型の変数をプロッタ装置の状態を把握する目的でグローバル変数として使用している[5].

plots(char *title, int *ww, int *wh) 出力ファイル calcomp.ps をオープンする. Calcomp の元の仕様を変更して, 第 2,3 引数に図のサイズ (ww, wh) を渡すようにしたが, これを Encapsulated PostScript (EPS) ファイルの仕様である "%%BoundingBox 0 0 ww wh" コマンドに変換する. BoundingBox には左下隅と右上隅の座標値を与える.

plots ルーチンは初期化なので, ここで PostScript の仮想マシンの上に描画オペレータを定義する. 後続の plot による作図はこのコマンドで行う. 現在点から指定された位置 (x,y) に線分を引く pline 手続きを定義する.

```
/pline {newpath moveto lineto setlgray stroke}def
```

pline は, 例えば"10.0 10.0 30.0 10.0 pline" は, 新たなパスを newpath によって起動し, 30.0 10.0 moveto によって現在点を $(30, 10)$ に移動し, 10.0 10.0 lineto によって $(10, 10)$ までパスを延長し, stroke によってパスを描く.

次にフォントファミリーを定義する.

```
/Courier findfont 10 scalefont setfont
```

センターシンボル用に, ○, □, × 印を定義する. 描画オペレータ arc は "x y r from to arc" で, 中心 (x, y) に半径 r の円弧を開始角 from から終了角 to まで描く. したがって "50 60 3 circle" は, スタックに "50 60 3" と積まれたところで "circle" が "newpath 0 360 arc" に置き換えられて実行される. newpath が処理された段階で "50 60 3 0 360 arc" になっているので, 中心 $(50, 60)$ に半径 3 の円が描かれる.

```
/circle{newpath 0 360 arc }def
/square{newpath /Dim exch def moveto Dim Dim rmoveto 0 Dim 2 mul neg rlineto
        Dim 2 mul neg 0 rlineto 0 Dim 2 mul rlineto closepath }def
/xmark{newpath /Dim exch def moveto Dim Dim rmoveto Dim 2 mul neg Dim 2 mul neg
     rlineto 0 Dim 2 mul rmoveto Dim 2 mul Dim 2 mul neg rlineto closepath }def
```

"50 60 3 square" は 3 を /Dim というキーに結びつけた後, "50 60 moveto" によって $(50, 60)$ に移動して, "3 3 rmoveto" によって正方形の右上に移動, 以下 "0 3 2 mul neg rlineto" で右下に直線を引く (以下省略)[6].

.6 setlinewidth 線幅の定義を行う.

以上は **Calcomp ペンプロッタをエミュレートできる仮想マシンの設定**で, PostScript プログラムのプロローグに置かれる. ここで注意したいことは, pline に渡す座標値の変数型が, 整数でも小数点数でもかまわないことにある. そこで calcomps.c では引数に受けた単精度浮動小数点数を小数点数として PostScript に渡している.

factor(float *f) スケールファクター Fctr を f に変更する.

[5] Icol はグレイスケールのレベル, Ipage はページ番号, Fctr はスケールファクター, (Curx,Cury) は現在点である.
[6] neg はスタックトップの数値の符号を変更する.

plot(float *x, float *y, int *opr)　第 3 引数 opr が “3” のペンアップの場合は現在点 (Curx,Cury) を更新するだけであるが，“2” のペンダウンの場合は，現在点から指定点 (x, y) への線分を描くためのメッセージ “Curx Cury x y pline” を出力する）．これは plots で出力した “/pline{newpath moveto lineto stroke} def” によってグレイスケールが Icol の線分になる．具体例で述べると “call plot(0.,0.,3)”，“call plot(1.,0.,2)”，“call plot(2.,1.,2)” と呼び出した場合は，“0.0 0.0 1.0 0.0 pline” と “1.0 0.0 2.0 1.0 pline” とが出力される．2 つめは “2.0 1.0 lineto” でもよいのでパスを延長するという考え方からは冗長であるが，ファイルをテキストエディタで直接変更して，直線を削除する用途も可能にしている．

opr が “–3” の場合は新たな原点を “translate” によって定義し，“999” の場合は “showpage” を出力して出力ファイルをクローズする．

where(float *x, float *y)　現在点 (Curx,Cury) の値を (x,y) に返す．

symbol(float *x, float *y, char *text)　文字列 text を (x, y) に置く．Fortran プログラムから呼ばれた場合は，C 言語の文字列（string）の終端記号 “\0” が入っていないことが多い．そこで，文字列の中にブランクがあればこれを終端記号として認識することにした（**ユーザ定義の終端記号**）．C プログラムから呼ばれた場合は “\0” による文字列の終わりとブランク文字の両方を引数の終わりとして認識する．したがって “call symbol(10.5,20.5,’ABC ’)” と “C” の次にブランクを入れて呼んだ場合 “newpath 10.5 20.5 moveto 1 setlgray ABC show stroke” とブランクを削除した 3 文字が出力される．

linewidth(int *iwidth)　線の幅を iwidth （単位はポイント）に変更するために “iwidth setlinewidth” を出力する．

centsym(float *x, float *y, int *mark, float *fm)　サイズ fm の円，正方形，× 印を描く．mark を 1 にして “call centsym(10.5,20.5,1,1.2)” と呼んだ場合 “10.5 20.5 1.2 circle stroke” が，2 にして同様に呼んだ場合 “10.5 20.5 1.2 square stroke” が，3 にして同様に呼んだ場合 “10.5 20.5 1.2 xmark stroke” が出力される．

newpen(int *icolor)　赤から青までを 32 通りに分けて，$r = \mathrm{mod}(32 - icolor, 32)$，$b = \mathrm{mod}(icolor, 32)$ として得た値を使って “r 0 b setrgbcolor” を出力する．引数が 0 の場合は黒である（0 0 0 setrgbcolor）．構造解析などで元の形状と変形後の形状を重ねて描く場合に利用する．

curve(float *x1, float *y1, float *x2, float *y2,float *x3, float *y3)　現在点を (x_0, y_0) とすると，(x_1, y_1)，(x_2, y_2)，(x_3, y_3) の 4 点を 3 次のベジェ曲線の始点，制御点，終点として描く．“newpath x_0 y_0 moveto” と “x_1 y_1 x_2 y_2 x_3 y_3 curveto stroke” が出力される．ただし x_1, y_1 などは Fctr 倍されている．

arc(float *r, float *a1, float *a2)　現在点 (x_0, y_0) を中心として半径 r の円弧を，開始角 $a1$ 終了角 $a2$ で描く．角度の単位は度（degree）で，0 が x 軸方向で反時計回りで指定する．“newpath x_0 y_0 r a_1 a_2 arc stroke” が出力される．ただし r は Fctr 倍されている．

newpage()　引数なしで使用し，改ページを指定する．出力ファイルには “showpage” と “%%Page: ページ番号” を書く[7]

[7] Adobe 社の EPS ファイルの仕様では，ヘッダー部分に “%%Pages:ファイルのページ数” を記述するようになっているが，これなしでもファイルを開くことができる（GSview では DSC Error のメッセージが出るが継続して操作可能であった）．Adobe 社の Acrobat Reader ではエラーになるが， Acrobat では pdf ファイルに変換できた．

special(char *text)　“call special(’ 文字列’)” とすると，出力ファイルに “文字列” を書くだけであるが，これは非常用脱出口（escape hatch）として使用できる便利なインターフェースである．PostScript に簡潔なコマンドが用意されている場合に，それを使用せずに “call plot” による短い線分で描くのは煩わしい．しかし calcomps.c ルーチンのソースコードを改訂して加えると，PostScript 以外の Calcomp インターフェースルーチンでは処理できなくなってしまう．このように PostScript 用にだけ特別にしたいが，ライブラリを変更したくないとき使用するので「非常用」である．

次の例は円弧を arc を使用せずに図示する場合である．レンズ球面の半径とレンズの位置は収差計算プログラムが別途保持しているのでこれを利用した．そこで arc コマンドの開始角と終了角を含めた手続きを “/convex1” で定義し，レンズの球面半径 r1 と位置 c1 を動的書式を用いて文字列変換して “convex1” 命令に埋め込んでレンズ形状を描いた，

```
      character*1 endstr
      data endstr/'\0'/
      call special ('/convex1 {newpath 170 180 arc stroke } def\0')
      write(charg,'(f9.2," 0 ",f9.2," convex1")') c1*f+sox,r1*f
      call special(charg//endstr)
```

Fortran では文字列の扱いが C 言語と異なることに注意されたい．“call special” の引数に直接文字列を引用符で囲って記述する場合，C 言語規格が定義している文字列の終端を記入する．また動的書式を使用した場合も，動的に生成された文字列には終端記号はないので，上の例では data 文で 1 文字変数 endstr に終端記号を作成しておき，これを文字演算子 “//” によって付加している[8]．

Fortran 用のラッパー　GNU の言語環境では，Fortran から呼び出される C プログラムは，エントリ名にアンダースコア “_” をつける必要がある．そこで次のような引数を渡すだけの入り口を設けた．

```
/*     wrapper for GNU Fortran   */
void plots_(char *disp, int *ww, int *wh)
{    plots (      disp,      ww,      wh);}
void plot_(float *x, float *y, int *opr)
{    plot (       x,        y,      opr);}
void symbol_(float *x, float *y, char *text)
{    symbol (       x,        y,       text);}
   以下，省略
```

2.2　例題：ボールの軌跡を表示するプログラム ball.f

ボールを初速 v_0，角度 θ で投げ上げた時の軌跡（放物線）を描いてみる．初速度は $v_0 = 35\mathrm{m/s}$ とし，角度を 5 度，10 度，20 度，30 度，40 度，45 度，50 度，60 度の場合の 8 通りについての軌跡を描く．parameter 文で ncase=8 を定義して，長さ ncase の配列 angle(1:ncase) をとり，これに data 文で 8 通りの角度をセットした．ボールの軌跡は初速度の x 成分を $v_x = v_0 \cos\theta$，y 成分を $v_y = v_0 \sin\theta$ とすると $x = v_x t$ および $y = v_y t - 0.5gt^2$ である（$g = 9.8\mathrm{m/s}^2$）．図のスケールを決定しなけらばならないので，ボールが飛ぶ高さと飛距離の最大値を求める．飛行時間は $y = 0$ となるときから $t = 2v_y/g$，飛距離は $x_{max} = 2v_x v_y/g$ つまり v_0^2/g，高さは $y_{max} = v_y^2/(2g)$ である．初速度 $35\mathrm{m/s} = 126\mathrm{km/s}$ で投げた場合，125 m の飛距離になる．

```
      parameter (ncase=8)
      dimension angle(ncase)
      character*100 ch,charg
```

[8] 引用符の中で引用符を使う場合，2 重引用符が使用可能である．

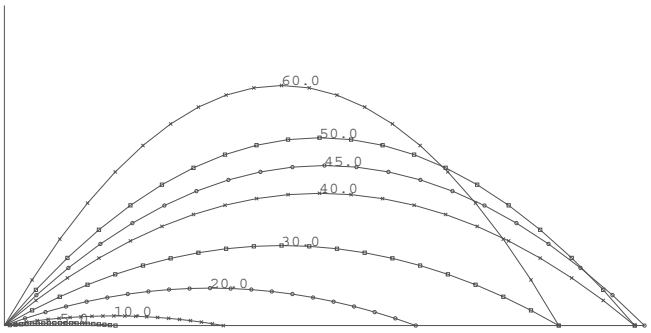

図 2: ボールの軌跡のプロット

```
      character*4 clabel
      data ch/'ball'/,iww/400/,iwh/300/
      data g/9.8/,n/20/,v/35./,angle/5.,10.,20.,30.,40.,45.,50.,60./
      RAD(X)  = X * 3.141592 / 180.0
c                                --- determine figure size  ---
      xaxis=v**2/g; yaxis=v**2/(2*g)
      f=float(iww)/xaxis
      write(*,*) ' Xmax=',xaxis,' Ymax=',yaxis,' f=',f
```

ケースごとに滞空時間を n=20 分割して，ボールの位置をプロットした（放物線ではなく 20 本の直線を描く）．20 分割されたぞれぞれの位置にセンターシンボルを配した．また最高点の位置に，投げ上げた角度を “call symbol” によって書いた．文字型変数は “character*4 clabel” で定義して，書式 “f4.1” によって浮動小数点数を 4 文字の小数に変換した．

```
      call plots (ch,iww,iwh)
      call factor (f)
      call plot  (5.,10.,-3)
      call pline  (0.,0.,xaxis,0.)
      call pline  (0.,0.,0.,yaxis)
      Do k=1,ncase
        vx=v*cos(rad(angle(k))); vy=v*sin(rad(angle(k)))
        time=2*vy/g; dt=time/n; mark=mod(k,3)+1; ymax=0
        call plot  (0.,0.,3)
        do i=1,n
          t=i*dt; x=vx*t; y=vy*t-0.5*g*t*t
          if(y.gt.ymax) then
            at=x; ymax=y
          endif
          call plot (x,y,2)
          call centsym (x,y,mark,0.1*f)
        enddo
        write(clabel,'(f4.1)') angle(k)
        call symbol (at,ymax,clabel)
      Enddo
```

“call pline” は “subroutine pline(x1,y1,x2,y2)” によって (x_1, y_1) から (x_2, y_2) に線分を描く．1 ページ目を図 2 に示す．

2 枚目の作図は “call newpage” によって改ページして，グレイスケールを変更して同じ図をプロットした．

```
      call newpage
      call factor (f)
      call plot  (10.,10.,-3)
      call pline  (0.,0.,xaxis,0.)
      call pline  (0.,0.,0.,yaxis)
```

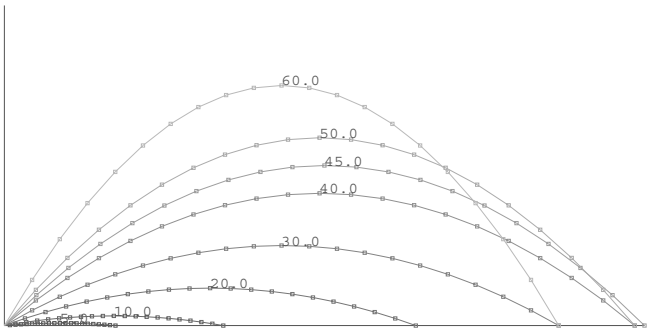

図 3: ボールの軌跡のプロット

```
      Do k=1,ncase
        vx=v*cos(rad(angle(k))); vy=v*sin(rad(angle(k)))
        time=2*vy/g; dt=time/n; ymax=0
        call color  (k*3)
        call plot  (0.,0.,3)
        do i=1,n
          t=i*dt; x=vx*t; y=vy*t-0.5*g*t*t
          if(y.gt.ymax) then
            at=x; ymax=y
          endif
          call plot (x,y,2)
          call centsym (x,y,2,0.1*f)
        enddo
        write(clabel,'(f4.1)') angle(k)
        call symbol (at,ymax,clabel)
      Enddo
c
      call plot (t,y,999)
```

2 ページ目を図 3 に示す.

2.3 例題：レンズの球面収差をプロットするプログラム PlotLens.f

4 種類の球面収差図を 1 枚の図に並べて描くので，“call plot(. . . , −3)” を使用した．凸レンズ単体，凸レンズでできる収差を凹レンズで打ち消す例，テッサー型と呼ばれる 4 枚構成のもの，ガウス型と呼ばれる 1 眼レフの標準レンズに使用される 6 枚で対称に構成したものである．メインプログラムではそれぞれのレンズ種類ごとの収差計算と作画を行うプログラム（lenssingle，lensdouble，lenstesser，lensgauss）の呼び出しを初期化と原点移動と終了ルーチン呼び出しの間に入れる．

```
      data iww/800/, iwh/600/
c
      call plots (ch,iww,iwh)
      call plot (100.,500.,-3)
      call lenssingle
      call plot (0.,-130.,-3)
      call lensdouble
      call plot (0.,-130.,-3)
      call lenstesser
      call plot (0.,-130.,-3)
      call lensgauss
      call plot (0.,0.,999)
      stop
      end
```

ここでは単体の凸レンズの球面収差の計算部分を示す．レンズは両面とも半径 120mm の球面で，屈折率は 1.8 とし，光線は光軸に平行に入れている．

```
      SUBROUTINE lenssingle
      implicit double precision (A-H,O-R,T-Z)
      dimension saber(50)
      data nray/25/,f/5.d0/
c
      call plot (0.,0.,-3)
c   -------------------------
      r1=120.; a=0.d0
      p=-1.d0; c1=r1; r2=r1
      c2=5.d0-c1   ! 5.-100.
      call pline (-50.,0.,500.,0.)   !  X axis
c -------------------------------------------------------------
      call factor (real(f))
      call plot(real(c1),0.,3)     !  move to lens center
      call arc (real(r1),170.,180.)
c -------------------------------------------------------------
      call plot(real(c2),0.,3)     !  move to lens center
      call arc (real(r2),0.,10.)
c -------------------------------------------------------------
      rrat=1.8d0   ! r1=75.7 ==> f=51.6
      dy=16.d0/dble(nray)
c
       do i=1,nray
        y=i*dy; q=y
        call incdnt(c1,r1,p,q,a,x1,y1)
        call snell (c1,r1,a,x1,y1,rrat,g)
        call incdnt(c2,r2,x1,y1,g,x2,y2)
        call snell (c2,r2,g,x2,y2,1.d0/rrat,gg)
        x3=(gg*x2-y2)/gg
c
        call plot (real(p),real(q),3)
        call plot (real(x1),real(y1),2)
        call plot (real(x2),real(y2),2)
        call plot (real(x3),0.,2)
        saber(i)=x3       ! save Aberration point
       enddo
c
      dmx=0.
      call plot(saber(1),0.0,3)
       do i=1,nray
        sy=i*dy
        call plot(saber(i),sy,2)
        if(abs(saber(i)-saber(1)).gt.dmx) dmx=abs(saber(i)-saber(1))
       enddo
c
      write(*,*) 'LensSingle xmax =',dmx
c
      call factor (1.0)
      return
      end
```

レンズの中心から h の高さを通過した光軸に平行な光線が，実際に光軸に交わる位置を $f - g(h)$ として，この $g(h)$ をスネルの法則から計算して作図する．

球面レンズの場合，$g(h)$ は h に関して単調増加である[9]．

[9] 前面を楕円面，後面を球面とする非球面レンズなら球面収差をゼロにできる．

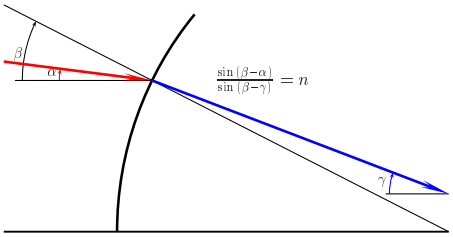

図 4: スネルの法則

サブルーチン incdnt は，レンズ近傍の点 P(p, q) を通る傾き a の光線

$$y - q = a(x - p)$$

が，

$$(x - c)^2 + y^2 = r^2$$

で表されるレンズの断面に入射する点 R(x, y) を求める．x は 2 次方程式

$$(1 + a^2)x^2 - 2(c + a^2p - aq)x + c^2 + a^2p^2 - 2aqp + q^2 - r^2 = 0$$

の根で 2 つあるが，P に近いほうを選択する．サブルーチン snell は，点 R(x,y) でのレンズ面の法線方向 $b = y/(x - c)$ を計算して出射方向 g を求める．図 4 に球面の光軸を含む断面での図を示したが，光軸に α で入射した光線は，屈折率 n のレンズの中では光軸に γ の角度に変化する．球面の中心に向う角度 β が分かれば，三角関数の除算 $\sin(\beta - \alpha)/\sin(\beta - \gamma) = n$ より出射方向を求めることができる．ただし α，β，γ はそれぞれ $a = \tan\alpha$，$b = \tan\beta$，$g = \tan\gamma$ である．入射角は β から α を測り，時計回りが正である．n は入射側が空気の場合ガラスの屈折率，反対の場合は屈折率の逆数を指定する．

球面収差の除去：テッサー型　4 枚のレンズで構成されたテッサー型の例（図 6）がカメラレンズの書に掲載されていた [2][10]．レンズ面が 7 面あり，収差が小さいため，右に収差を 10 倍に拡大して示した．

球面収差の除去：ガウス型　参考書に計算可能なレンズの例（Leitz Canada の特許で，ライカレンズのズミクロン M50mmF2 の基礎となる）を見つけたので計算してみた [2]．1981 年に設計されたようである．表 1 に計算に使用したデータを示す．またこのレンズの形状を図 7 に示した．

レンズはガウスタイプの 4 群 6 枚構成で，加工上の理由で平面を多くしたと書かれている．図 5 の 4 番目に計算結果をプロットしたが，収差がほとんど見られないので，右側に 10 倍に収差を拡大して示した．レンズ面が増えれば，設計パラメータが増加するので，収差補正曲線も高次となる．レンズの端から入射した光線が中央と同じ焦点距離に戻るような設計をフルコレクションという．それを越えて逆方向に振る場合をオーバーコレクションという．図から見られるようにアンダーコレクションとなっている．

[10] 計算可能な数値が記載されていたわけではないが，正確な図面とガラス種が書かれていたので，図を拡大コピーして，レンズ面の半径を読み取って球面計算をしてみた．この作業は金沢工業大学の M 君が行ってくれた．したがって計算に使用したレンズのデータは正確なものではないが，計算された収差曲線（図 5 の 3 番目）は参考書の形に近く，2 枚のレンズからなる曲線（図 5 の 2 番目）よりも高次の曲線が得られた．このレンズ（ロッコール 45mmF2.8）はミノルタハイマチックという虚像を重ねる二重像合致式の測距用のファインダーを持つ機種に搭載された．このカメラは 1962 年にアメリカ初の有人宇宙船「フレンドシップ 7」に搭載され，初めて宇宙から地球を撮影する名誉を得た．設計は 1959 年で，現在のような計算機はなく，卓上計算機によって設計計算が行われたが，参考書によると「洞察力の鋭い優れたレンズ設計者の手になったため，当時としては水準を越えたレンズであったと思われる」と記されている（p. 82）．

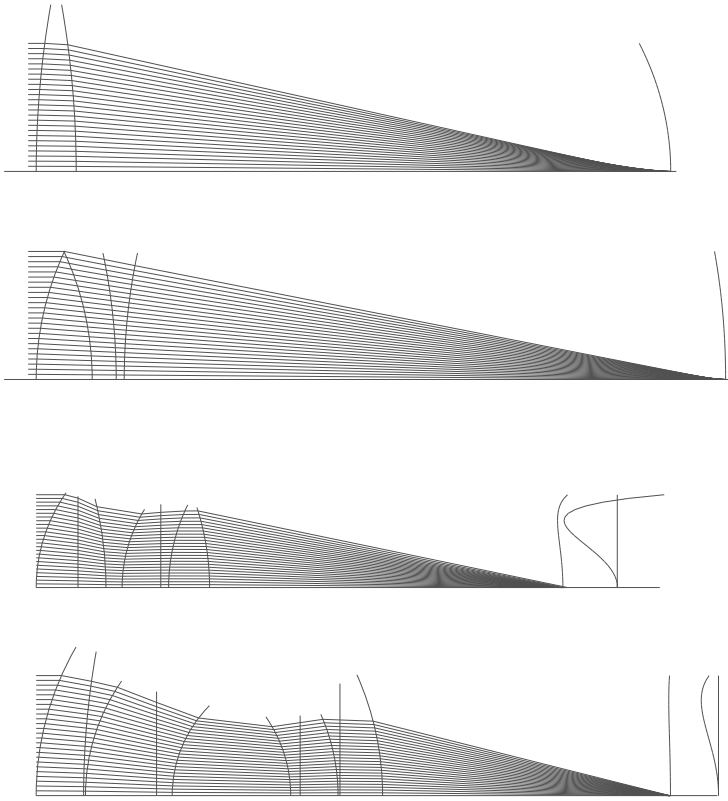

図 5: レンズの球面収差のプロット

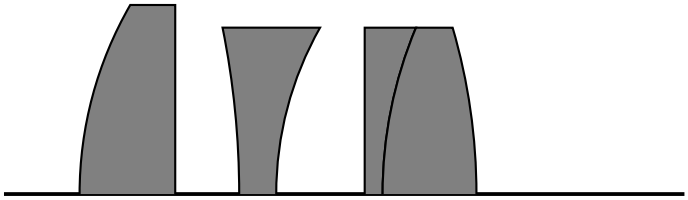

図 6: テッサー型のレンズ:ロッコール 45mmF2.8（ミノルタハイマチック）

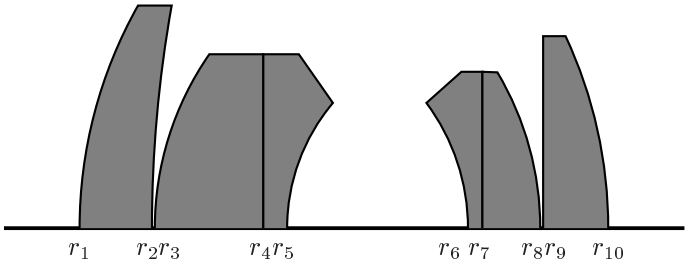

図 7: ガウス型のレンズ:ライカ summicron F2.0

表 1: Leica Summicron M50mm F2 のレンズ構成

$f = 100, F2, 2\omega = 45\,°$

曲率半径		面間隔		屈折率 n_d	分散 ν_d	x_i	c_i	レンズ高		θ_i
r_1	59.94					0.0	59.94			29.417
		d_1	9.57	1.78831	47.39			y_1, y_2	29.44	
r_2	167.31					9.57	176.88			10.135
		d_2	0.38	1.0(空気)						
r_3	40.30					9.95	50.25			34.800
		d_3	14.35	1.66755	41.93			y_3, y_4	23.00	
r_4	∞(平面)					24.30	∞			
		d_4	2.87	1.72825	28.41					
r_5	25.67					27.17	88.00			40.174
		d_5	24.20	1.0				y_5, y_6	16.56	
r_6	−27.69					51.37	23.68			36.730
		d_6	1.91	1.62588	35.70					
r_7	∞					53.28	∞			
		d_7	7.65	1.71700	47.98			y_7, y_8	20.61	
r_8	−40.30					60.93	20.63			30.755
		d_8	0.38	1.0						
r_9	∞					61.31	∞			
		d_9	8.61	1.71700	47.98			y_9, y_{10}	25.39	
r_{10}	−59.94					69.92	9.98			25.064

2.4 例題：複数の図を原点を変更して描くプログラム plotest.f

対話型の作画は，例えば構造解析で変形量の拡大率を，作画してからでないと決められない場合などに要求される．これは PostScript では満たされないが，X 窓であれば対応できる．この場合でも PostScript に作図できなくてはいけない．

プログラム plotest.f が 1 枚目の図を “call testfig” で描いた後 “read(*,*) xx,yy” で標準入力からの入力待ちになる．新しい原点を入力すると 2 枚目の図を描く．このとき原点は入力されたものが使用される．

2.5 ライブラリ calcompt.c

LaTeX 2εでは pspicture 環境で pstricks パッケージを使用して，PostScript の機能で作図できる．機能的には PostScrit を出力するものと同等なので，実装の要点を掻い摘んで紹介する．

plots(char *title, int *ww, int *wh) 対応　出力ファイル calcomt.tex をオープンする．数値の単位はポイントとする．

```
void plots(char *title, int *ww, int *wh) {
int    i;
float  p;
     output=fopen("calcompt.tex","w");
     fprintf(output,"\\documentclass{jarticle}\n");
     fprintf(output,"\\usepackage{pstricks}\n");
     fprintf(output,"\\begin{document}\n");
     fprintf(output,"\\begin{figure}\n");
     fprintf(output,"\\begin{center}\n");
     fprintf(output,"\\psset{unit=1pt}\n");
     Ww=*ww; Wh=*wh;
     fprintf(output,"\\begin{pspicture}(0,0)(%d,%d)\n",Ww,Wh);
     /* Initialize internal variables */
     Iput=0; Fctr=1.; Icol=1; Ipage+=1;
}
```

plot(float *x, float *y, int *opr) 対応　plot は，

```
px=*x*Fctr; py=*y*Fctr;
```

```
switch ( (*opr) ) {
  case -3 : /* new origin will be used, \put(x,y){ */
     printf("\\put(%f,%f){\n",px,py);
     fprintf(output,"\\put(%f,%f){\n",px,py);
     Iput+=1;
     break;
  case 2  :       /* move to (x,y) with pen down */
     xx0 = Curx; yy0 = Cury;
     xx1 = px;   yy1 = py;
     if(Icol<=1) fprintf(output,"\\psline(%f,%f)(%f,%f)\n",xx0,yy0,xx1,yy1);
     if(Icol==2) fprintf(output,"\\psline[linecolor=red](%f,%f)(%f,%f)\n",xx0,yy0,xx1,yy1);
     if(Icol==3) fprintf(output,"\\psline[linecolor=blue](%f,%f)(%f,%f)\n",xx0,yy0,xx1,yy1);
     if(Icol==4) fprintf(output,"\\psline[linecolor=green](%f,%f)(%f,%f)\n",xx0,yy0,xx1,yy1);
     if(Icol==5) fprintf(output,"\\psline[linecolor=cyan](%f,%f)(%f,%f)\n",xx0,yy0,xx1,yy1);
     if(Icol==6) fprintf(output,"\\psline[linecolor=magenta](%f,%f)(%f,%f)\n",xx0,yy0,xx1,yy1);
     if(Icol==7) fprintf(output,"\\psline[linecolor=yellow](%f,%f)(%f,%f)\n",xx0,yy0,xx1,yy1);
     break;
  case 999 :                  /* end of plot                */
     for(i=0; i<Iput; i++) fprintf(output,"}\n");
     fprintf(output,"\\end{pspicture}\n");
     fprintf(output,"\\end{center}\n");
     fprintf(output,"\\end{figure}\n");
     fprintf(output,"\\end{document}\n");
     fclose(output);
     printf("End of plot\n");
     break;
  default :   /*  case 3 'move to (x,y) with pen up' */
     break;        /* OR other *opr is specified   */
}
Curx = px, Cury = py;
```

newpage 対応　figure 環境を閉じ，新しい図とする．

```
void newpage () {
     int i;
     for(i=0; i<Iput; i++) fprintf(output,"}\n");
     fprintf(output,"\\end{pspicture}\n");
     fprintf(output,"\\end{center}\n");
     fprintf(output,"\\end{figure}\n");
     fprintf(output,"%%%%%%%%%%%%%\n");
     fprintf(output,"\\begin{figure}\n");
     fprintf(output,"\\begin{center}\n");
     fprintf(output,"\\psset{unit=1pt}\n");
     fprintf(output,"\\begin{pspicture}(0,0)(%d,%d)\n",Ww,Wh);
     /* Initialize internal variables */
     Iput=0; Fctr=1.; Icol=1; Ipage+=1;
}
```

curve 対応　\psbezier 命令にする．

arc 対応　\psarc 命令にする．

symbol 対応　文字列を uput 命令で置く．

centsym 対応　psdots 命令で点を示すシンボルをオプション引数 dotstyle で与える．

```
     mdot=(*mark)%5+1;
     xx0 = (*x) * Fctr; yy0 = (*y) * Fctr;
```

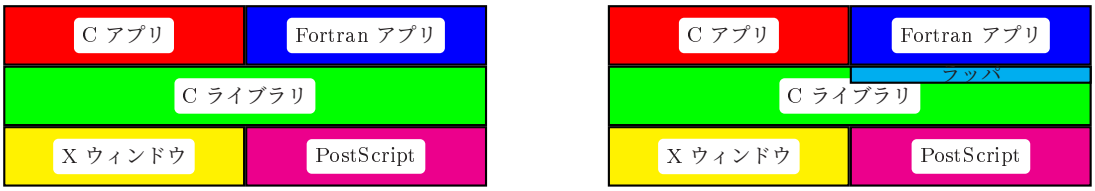

図 8: Fortran コーラブルなラッパー：UNIX 環境（左）と Cygwin 環境（右）

```
switch ( mdot )
{
     case 5  :       /*  triangle* mark  */
         fprintf(output,"\\psdots[dotstyle=triangle*](%f,%f)\n",xx0,yy0);
         break;
     case 4  :       /*  square* mark  */
         fprintf(output,"\\psdots[dotstyle=square*](%f,%f)\n",xx0,yy0);
         break;
     case 3  :       /*  circle*  mark  */
         fprintf(output,"\\psdots[dotstyle=*](%f,%f)\n",xx0,yy0);
         break;
     case 2  :       /*  square  */
         fprintf(output,"\\psdots[dotstyle=square](%f,%f)\n",xx0,yy0);
         break;
     default :       /*  circle  */
         fprintf(output,"\\psdots[dotstyle=o](%f,%f)\n",xx0,yy0);
         break;
 }
 Curx=xx0, Cury=yy0;
```

newpen 対応　線の色を引数を 8 で割った余りで，黒，赤，青，緑，シアン，マゼンタ，黄で変更する．

3　Calcomp インターフェースの X 窓用ライブラリ

X-Window システムは，DEC との共同研究で，Athena プロジェクトの一環として MIT（マサチューセッツ工科大学）のコンピュータサイエンス研究所で開発され，1984 年に初めて発表された．ウィンドウシステムは 1980 年代の初めに登場した “マウス付きのインテリジェント・ビットマップ・グラフィック端末” を前提に開発された[11]．ここでは Calcomp インターフェースで書かれたプログラムの出図を X 端末に出力する例を紹介する．Calcomp インターフェースの PostScript 用ライブラリと同様に，X 端末に出力する．両者のアプリケーション，ライブラリ，作画システムの階層構造を，IBM などのメーカーの UNIX 環境と GNU を使用した UNIX（Cygwin）環境の違いを示す（8）．

3.1　X 窓システム

それ以前のウィンドウシステムは，OS のカーネル部分にウィンドウシステムの制御機能を持たせてシステムコール形式で使用する**カーネル型**と，OS には必要最小限のリソース管理機能だけを持たせ，他は

[11] ビットマップ・グラフィック端末は，文字や絵を一緒に描くことができるように，画面全体をビットマップとして扱うことのできる端末で，「ダム端末」に対比させられる言葉である．

ライブラリサポートにする**ライブラリ型**があった．しかし X 窓システムは**サーバ・クライアント型**として実装された．

サーバ・クライアント型　計算機システムの歴史の中では，1985 年頃まで，計算機は大型化し高価になる傾向が持続していたが，80 年代後半から 2 つの先進的な技術がこの状況を一変させた．強力なマイクロプロセッサと LAN の発明である．これにより中央集中型システムから分散システムへの移行が始まった．X 窓システムが計画されたのもこの時代である．

X 窓システムでは LAN で接続されたエンジニアリング・ワークステーションのネットワーク環境を想定している．ここではマシンを独占して使えるワークステーションがユーザごとに 1 台ある．ワークステーションにはハードディスクは付いていないこともある．ユーザはネットワークごしに，遠隔地の（リモート）ワークステーションに “rlogin マシン名” のようなコマンドを使ってログインする使用法も想定している．

リモートコマンドは基本的にクライアント／サーバ・モデルを基礎としており，遠隔地のホストにローカル端末を接続するという方法で実現されている．具体的には，“rlogin” や “telnet” コマンドによって，自分のワークステーションはリモートの計算機の端末になる．この機能の実現は，rlogin クライアントが擬似端末を通してネットワークごしにサーバーの rlogind と情報をやり取りすることで実装されている．この方式ではサーバプロセスは常時走っており，ウィンドウプログラムはライブラリを介してサーバと通信し，サーバがディスプレイへの描画を行う．サーバはクライアントと同じマシン上で動く必要はなく，ネットワーク越しに使用できる．これは環境変数の DISPLAY にサーバーマシンの IP アドレスを指定すれば可能となる．

最初の X は スタンフォードの Paul Asente によって開発された ”W” というウィンドウパッケージに負うところが大きい．1987 年 9 月，MIT は X11 の最初の版を発表した．X11R2 以降，管理は MIT から 1988 年 1 月に発足した X コンソーシアムに移行している．現在は UNIX のウィンドウ環境の一つで，UNIX，Linux には標準装備されている．またマイクロソフトの Windows でも Cygwin 環境で startxwin.bat を起動すれば利用できるようになった．

インターネット越しでの X の使用　ネットワークは当初は LAN であったが，現在ではインターネット越しで使用する場合も多い．この場合は暗号化された ssh のプロトコルを利用する．この場合は “-X オプション” で X11 のポートフォワード機能を使う．リモートログインしたゲートウェイマシンから，さらに別のマシンへログインする場合でも可能である．筆者は IBM 勤務時代は，自席のノート型 PC（ThinkPad）にインストールした Cygwin 環境で，X 窓から米国 IBM のゲートウェイマシンに，X のポートフォワード機能を用いて “ssh -X ユーザ ID@ホスト名” でログインし，さらにベンチマーク用の計算機に telnet で入って仕事をしていた．このとき ssh で入ったホストで “echo $DISPLAY” によってポート番号（例えば “ホスト名:15.0”）を調べ，telnet で次の計算機にログインしたとき環境変数の設定，”export DISPLAY＝ホスト名:15.0” を入力することで，自分の PC を X サーバとして使用している．

X 端末　ウィンドウプログラム（X クライアント）を実行するマシンと， 画面表示とマウスやキーボードの入力を行う X サーバプログラム を実行するマシン（X サーバ）が分かれているところが X 窓システムの特徴である．X サーバ実行専用マシンを **X 端末**と呼ぶ．X クライアントと X サーバは LAN 接続を前提としているが，両者が同一のマシンでもかまわない（ネットワーク透過性）．

3.2　Calcomp 対応ライブラリの概要

X ライブラリ（以下，Xlib）は C 言語を使ってウィンドウ・プログラミングを実現するためのライブラリで，線分や円弧などの 2 次元オブジェクトの描画， 文字表示，カラーを扱う関数などが用意されている．こ

こで描画のために使用する関数は，線分（XDrawLine），円弧（XDrawArc），長方形（XDrawRectangle），文字列（XDrawString）の 4 種類で，これに色と線幅を設定する関数（XSetLineAttributes）である．

マウスやキーボードを用いた対話形式の操作もサポートされるが，ここでは 2 種類のイベントを受け付て処理する．PostScript では扱いにくい操作として，作画の際にパラメータを変更して何度も作図し，見映えの良いパラメータ値を試行して探すことを可能にした（この目的でキー入力イベントを扱う）．これは例えば構造解析で，原形図に変形図を重ねて描く場合の拡大縮小のスケールを探す場合である．もう 1 つのイベント処理は，表示された図形が他の窓で消された場合の再表示処理のために Expose イベントを扱う．

X 窓の座標系とカラー　X 窓はラスタスキャン型の表示装置を前提に設計されており，x，y 座標値はピクセル座標（整数値）で指定する．したがってドラム型プロッタをエミュレートするためには，浮動小数点数で与えられた x，y 座標値を，ピクセル値（整数）に変換する．また Xlib は**原点が窓の左上**にあり y 軸が下向きなので，上下を反転する．

X 端末に表示されている画面は，グラフィックカードのメモリ（VRAM）に随時記憶されていて，それが画面に随時表示される．グラフィックに対する描画は，具体的にはこの VRAM に色番号を書き込むことなので，ドラム型プロッタでのペン番号の変更とはかなり異なる．色の種類や線の太さはその例である．そこで Calcomp ライブラリでは NCOLOR 種類の色が切り替えられるように “color(icolor)” というインターフェースを追加し，“newpen(ipen)” で指定するのは線の太さとした．

操作の概要　PostScript 出力と同様のインターフェースで X 窓に描画するが，1 ページ単位で表示が停止する．これは描画の完了となる “call plot(x,y,999)”，および “call newpage” によって次の作図に移るときに “while(1)” の無限ループで操作待ちを入れているからで，表示画面に対して任意のキー入力があるまで処理はこのループによって先に進めない．なおここに Expose イベント処理を含めたので，このタイミングで他の X 窓によって隠される部分があれば再表示する（後述）．以下，“call newpage” における処理を示す．

```
void newpage(){
  XFlush (Dpp);                                  バッファの内容を送りだす
  while(1){                                      無限ループに入る
    XNextEvent(Dpp, &ev);
    if(ev.type == Expose) re_draw();             Expose イベントに対する再表示
    if(ev.type == KeyPress) break;               Key 入力があればこの無限ループを出る
  }
  XClearWindow (Dpp,Wd);
  ICOL=0; IPEN=1; Nopr=Nchar=0;                  初期化
  XSetLineAttributes (Dpp,gd[ICOL],1,LineSolid,CapButt,JoinMiter);
  Orgx=Orgy=Curx=Cury=0; Fctr=1;
}
```

“call plot(x,y,999)” に対する処理は，キー入力に対して “exit(1)” によってアプリケーションの実行を終える．

calcompx.c のグローバル変数　Xlib 用の構造体やマクロなどは X11/Xlib.h と X11/Xutil.h で宣言されている．前者は X 端末との通信（X プロトコル），基本的なリソース（グラフィックやフォントなど）の管理，イベントの処理に必要な基本作業を行う．後者はクライアント間通信，アプリケーションユーティリティ関数で使われる関数，型，シンボルを宣言している．

```
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
```

```
#define CHARLEN 100
#define NCOLOR 33
Display   *Dpp;                              画面に窓を表示するために必要な構造体
Window     Wd;                               窓 1 つにつき 1 つ使われる構造体
GC         gd[NCOLOR+1];                     色や線幅，重ねて描く場合の方法などの情報を管理
XEvent ev ;                                  イベントを取得するための構造体
XFontStruct  *fontinfo;                      フォントの幅，高さ，ID 番号などのフォント情報を管理
unsigned long pixels[NCOLOR+1];
int      ICOL, IPEN, Wx=0, Wy=0, Ww, Wh;     グラフィックの状態の把握用
float    Fctr, Orgx, Orgy, Curx, Cury;       グラフィックの状態の把握用
int  screen,black,white;
int  Nopr,Nchar;                             再表示用
char Chxy[10000];                            再表示用
typedef struct{int type,x,y,r,s,t,u;}XY;     再表示用
XY XYopr[5000];                              再表示用
```

GC は Graphic Context の略である．

plots に対応する初期化　XOpenDisplay(NULL) によってこのプログラムを実行しているマシンが，X 端末との通信経路を確立し，さまざまな情報をメモリに格納し，そのアドレスを Display 構造体のポインタで返す．X 窓は他人の画面にも出力することができ，引数にはその端末名を指定するが，NULL の場合は自分が使っている画面になる．

XCreateSimpleWindow は X 端末に新しく作る窓の管理を依頼する．ここでは XDefaultScreen で，普通のウィンドウ（サイズは Ww，Wh）の特質を取得して，黒い背景を用いている．XMapWindow によって窓を画面上に表示し，第 1 引数に与えられた文字列を表示窓のタイトルに出力する．

```
plots(char *disp, int *ww, int *wh){
  XColor   colors;
  Colormap cmap;
  Window   rw;
/*XSetWindowAttributes attr;  */
  int  fontascent,fontheight;
  int  i,mid,pix;
  char dispname[CHARLEN];
  unsigned long plane_masks[NCOLOR+1];
  for(i=0; (i<CHARLEN) && (*(disp+i) != ' '); i++) dispname[i]= *(disp+i);
  dispname[i]='\0';
  Dpp    = XOpenDisplay(NULL);    /* connects to server  */
  screen = XDefaultScreen(Dpp);
  Ww     = *ww; Wh     = *wh;
  black  = XBlackPixel(Dpp,screen);
  white  = XWhitePixel(Dpp,screen);
  cmap   = DefaultColormap(Dpp,screen);
  rw = XDefaultRootWindow (Dpp);          /* window_id for root window  */
  Wd = XCreateSimpleWindow(Dpp,rw,Wx,Wy,Ww,Wh,2,white,black); /* generate window  */
  XStoreName (Dpp, Wd, dispname );        /* Name of window  */
  XChangeWindowAttributes(Dpp, Wd, CWBackingStore, &attr );
  XMapWindow (Dpp,Wd);                /* map window */
  XSelectInput(Dpp,Wd,KeyPressMask|ExposureMask);
```

最後の XSelectInput の指定が，ウィンドウに対してキー入力イベントと Expose イベントを検知するための設定である．

次にフォントの設定，カラーの設定，線の太さの定義を行う．

```
/******  color definition     ************/
     gd[0]  = XCreateGC(Dpp,rw,0,NULL);
```

```
        colors.red   = 65535; colors.green = 65535; colors.blue  = 65535;
        colors.flags = DoRed|DoGreen|DoBlue;
        XAllocColor(Dpp,cmap,&colors);
        XSetForeground(Dpp,gd[0],colors.pixel);

        mid=(NCOLOR+1)/2; pix=65536/(NCOLOR-1);
        for(i=1;i<(NCOLOR+1);i++) {
              gd[i]  = XCreateGC(Dpp,rw,0,NULL);
              if(i <=  mid ) {
                  colors.blue = 65535;
                  colors.green= (2*pix*(i-1))<65535 ? 2*pix*(i-1) : 65535;
                  colors.red  = 0;
              } else {
                  colors.blue = 2*pix*(mid-i);
                  colors.green= 2*pix*(NCOLOR-i);
                  colors.red  = ((i-mid)*2*pix < 65535) ? (i-mid)*2*pix : 65535;
              }
              colors.flags = DoRed|DoGreen|DoBlue;
              XAllocColor(Dpp,cmap,&colors);
              XSetForeground(Dpp,gd[i],colors.pixel);
        }
    /*  Initialize with IPEN=unit width, ICOL=white */
        ICOL = 0 ; IPEN = 1 ;
        XSetLineAttributes (Dpp, gd[ICOL], 1, LineSolid, CapButt, JoinMiter );
```

ここで定義したカラー配合の例は starpalette.f に下部を参照されたい.

factor 対応　スケールファクター Fctr を変更する.

plot 対応　第 3 引数 “–3” の原点の移動は，変数 Orgx，Orgy の更新を行う．第 3 引数 “3” のペンの移動は，default ケースとして現在点を，指定点 (x, y) をスケールファクター倍した座標値で置き換える．第 3 引数 “2” の直線は XDrawLine で線分を描き，描き終わった位置を現在点とする.

```
    void plot(float *x, float *y, int *opr){
      int i,ix0,iy0,ix1,iy1;
      float px,py;
      px=*x*Fctr; py=*y*Fctr;
      switch ( (*opr) ){
        case -3 :                   /* move origin     */
          Orgx+=px; Orgy+=py;
          break;
        case 2  :       /* move to (x,y) with pen down */
          ix0=(int)(Curx+Orgx); iy0=(int)(Cury+Orgy);
          ix1=(int)(px  +Orgx); iy1=(int)(py  +Orgy);
       /* XDrawLine(Dpp,Wd,gd[ICOL],ix0,Wh-iy0,ix1,Wh-iy1);  */
          Xline    (                ix0,Wh-iy0,ix1,Wh-iy1);
          break;
        case 999 :                  /* end of plot     */
          XFlush (Dpp);
          while(1){
            XNextEvent(Dpp, &ev);
            if(ev.type == Expose) re_draw();
            if(ev.type == KeyPress) exit(1);
          }
          for(i=1;i<(NCOLOR+1);i++)
          XFreeGC (Dpp, gd[i]);
          XDestroyWindow (Dpp, Wd);
          XFlush (Dpp) ;
```

```
        XCloseDisplay(Dpp);
        break;
      default : /* case 3 'move to (x,y) with pen up'*/
        break;       /* OR other *opr is specified*/
    }
    Curx=px; Cury=py;
}
```

XDrawLine 関数を XLine 関数に置き換えているのは，“call plot” 以外にも XDrawLine 関数を呼び出すところがあるため，操作の配列保存を 1 箇所にするためである（後述）.

“999” に対応する終了操作では，XFlush でバッファに残っている X 命令を送り出す．Xlib の送り出す描画依頼は関数の実行と同時に X 端末に送られるのではなく，一旦リクエストバッファに蓄えられ，一定数のリクエストが溜まった時点でまとめて送られるので，XFlush をコールすることによって，このバッファに溜まっているリクエストを強制的に X 端末に送り出す.

表示が終わった段階でイベント待ちの状態でループに入る．キー入力によって窓を消す.

where 対応　現在点 Curx と Cury に 原点を加えた座標値を返す．PostScript では新たな原点の定義を translate コマンドで行えるので，where で戻る座標値は Curx と Cury であった．しかしページの概念のない Calcomp をエミュレートするために，X 用の calcompx.c の仕様は calcomps.c とは異なり，(Curx+Orgx,Cury+Orgy) を返す．そこで (Curx,Cury) を知るためにはその時点で定義されている原点に移動してから where を呼ぶ（plotest.f プログラムの例）.

```
c                    To know (Orgx,Orgy)
      call plot (0.,0.,3)
      call where (ox,oy)
      write(*,*) 'orgx=',ox,' orgy=',oy
```

calcomps.c と同じ (Curx,Cury) を知るためには，このようにして得た (Orgx,Orgy) を用いて次のように補正する.

```
c                    To know current point
      call where (x,y)
      write(*,*) 'Raw Current Point(',x,',',y,')'
      write(*,*) 'Current Point(',x-ox,',',y-oy,')'
```

symbol 対応　書くべき文字列とその位置を仮引数に受け取ると，文字列の窓の中での幅（ピクセル数）を XTextWidth によって取得して，指定位置（文字列の左下）と Xlib との調整を行って，XSetFont と XDrawString によって表示する[12]．座標値は Calcomp は単精度浮動小数点数で与えるが，Xlib のインターフェースは整数なので，型変換を行い，ここでスケールファクタの調整も行う．また Xlib は原点が窓の左上にあって y 軸は下を向いているので，窓の高さから y を引いて上下を反転させている.

newpen 対応　plots で初期化した色（gd[icolor 番目]）を，XSetLineAttributes で設定する.

linewidth 対応　引数に指定された線の太さを，XSetLineAttributes で設定して，線の太さを変更するようにした.

[12] XTextWidth によってフォント情報を取得するところは，TEX が TFM（フォントメトリック）ファイルを参照してフォントを置く位置を決定する処理に対応している.

centsym 対応　第 3 引数が 1（default）の場合は XDrawArc によって半径 fm の円を描く（ fm は第 4 引数）．2 の場合は XDrawRectangle によって一辺が fm の正方形を描く．3 の場合は XDrawLine によって×印を描く．

curve 対応　3 次のベジェ曲線を 64 分割して線分で描く（plot を呼び出す）．

arc 対応　XDrawArc によって円弧を描く．

newpage 対応　“call plot(x,y,999)” ではプログラムの実行を終わるので，複数の作図ができない．複数枚の作図は “call newpage” によって行う．

```
void newpage (){
     XFlush (Dpp);
     while(1){        /*     wait until key in  */
         XNextEvent(Dpp, &ev);
         if(ev.type == Expose) re_draw();
         if(ev.type == KeyPress) break;
     }
     XClearWindow (Dpp,Wd);
     ICOL=0; IPEN=1 ;
     XSetLineAttributes (Dpp,gd[ICOL],1,LineSolid,CapButt,JoinMiter);
     Orgx=Orgy=Curx=Cury=0; Fctr=1;
}
```

XFlush でバッファにたまっている X 命令を全部送り出した段階でイベント待ちの状態でループに入る．Expose イベントの検知で再表示，キー入力イベントの検知によって窓を初期化して次の作画に入る．

newpage による 2 枚目の作画と，アプリケーションに対するパラメータ変更の例を示す（plotest.f プログラムの例）．

```
      call plots ('plotest ',iww,iwh)
c     ----- draw figure at 100,200  -----
      call plot (100.,200.,-3)
      call testfig
      call newpage
c     ----- draw figure at xx,yy    -----
      write(*,*) 'input (x,y) for new origin'
      read(*,*) xx,yy
      call plot (xx,yy,-3)
      call testfig
c     ---------------------------------
      call plot (0.,0.,999)
```

原点を $(100, 200)$ として testfig で最初の作画を行った後，newpage でプログラムは無限ループでキー入力待ちになる．X 窓でキー入力すると実行画面に “input (x,y) for new origin” と表示されて read 文の入力待ちになるので，例えば “200 300” と入力すると，原点を $(200, 300)$ として testfig で 2 つめの作画を行う．

special 対応　何もしない．

追加エントリ xpause　対話形式で使用できるので，ソースコードに描画の一時停止の機能を含めた．処理は newpage と同様であるが，停止するだけで初期化などは行わない．

```
void xpause(){
  while(1){
    XNextEvent(Dpp, &ev);
    if(ev.type == KeyPress) break;
  }
}
```

これを利用すると描画のアルゴリズムを理解するのに役立つことがある．次の例は starpalette.f に含めた diamond サブルーチンの 2 重のループ構造で，外側のループ反復のたびに停止させてどこまで作画が進んでいるかを確認することができる．

```
      subroutine diamond (cx,cy,scale,ncolor)
      integer x(0:22),y(0:22)
      double precision th
      scolor=ncolor/12.0
      call factor (scale)
c
       do i=0,22
        th=2.d0*3.141592d0 * i / 23.d0
        x(i)=cos(th)*150+cx; y(i)=sin(th)*150+cy
       enddo
c
       Do k=12,1,-1
         icolor=scolor*k
         call color (icolor)
         m=0
         do i=0,23
           j=mod(m+k,23)
           if(k.eq.12.and.i.eq.0) then
             call plot (float(x(j)),float(y(j)),3)
           else
             call plot (float(x(j)),float(y(j)),2)
           endif
           m=j
         enddo
         call xpause
       Enddo
      return
```

描画の記憶と再表示　再表示には，操作を行った順序（シーケンス）を保存してその順序で描画を再現する方法と，すべての操作にグラフィックス状態を付加して保存して，再表示ではシーケンスにとらわれずに描画する方法が考えられる[13]．前者では関数の種別をコード化して配列の各レコードに格納し，再表示の際にこのコードによって（switch 文と case）分岐してそれぞれの関数を呼び出す．後者の方法では保存すべきグラフィック状態は色（ICOL）と線幅（IPEN）だけなので，これを含めて関数の種類ごとの配列に引数とともに記憶し，前回の操作と色や線幅に変更があった場合には XSetLineAttribute を呼び出す．どちらの方法でも文字列は長さが変化するので可変長レコードが必要になるので，文字列だけは別配列とした．ここでは前者の方法をとった．線分を描く場合を例に示す（Nopr が操作番号で，$type = 1$ が 線分（XDrawLine）である）．

```
int  Nopr;                操作番号
typedef struct{int type,x,y,r,s,t,u;}XY;
XY XYopr[5000];
```

[13] これ以外にオブジェクト指向的なアプローチも考えられる．

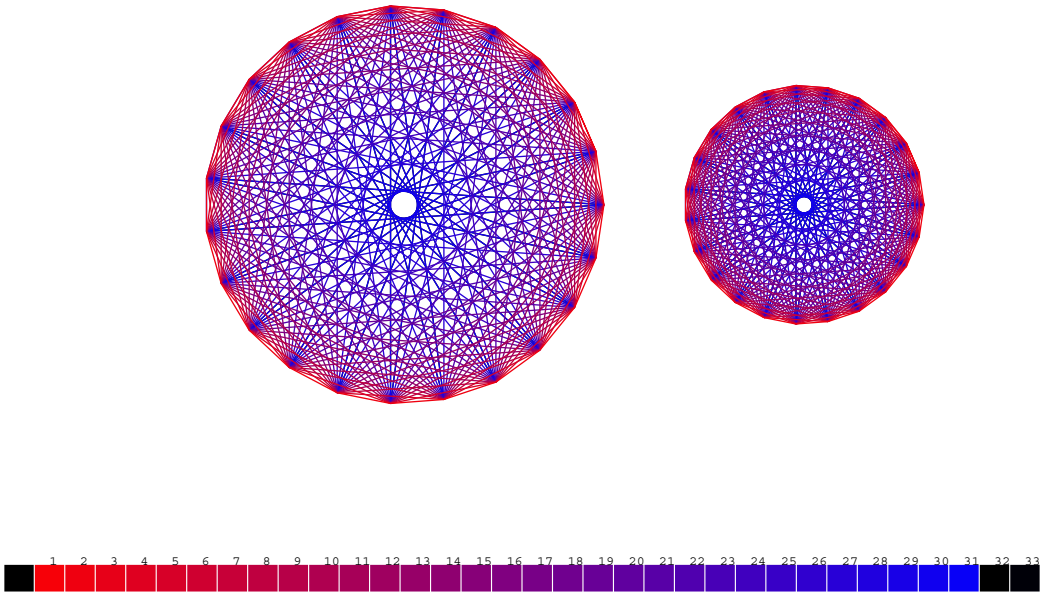

図 9: パレットとダイアモンド

```
Xline (int x0,int y0,int x1,int y1){
  XDrawLine(Dpp,Wd,gd[ICOL],x0,y0,x1,y1);                    線分の描画
  XYopr[Nopr].type=1; XYopr[Nopr].x=x0; XYopr[Nopr].y=y0;    描画操作の保存
  XYopr[Nopr].r=x1; XYopr[Nopr].s=y1; Nopr+=1; }
```

“call plot(x,y,999)” および “call newpage” で無現ループに入ると，Expose イベントを検知すると再表示のために re_draw 関数が呼ばれる．この中で線分を描く場合を示す．

```
void re_draw(){
  int i;
  IPEN=1, ICOL=0;
  for(i=0; i<Nopr; i++){
     switch ( XYopr[i].type ){
       case 1 :
         XDrawLine(Dpp,Wd,gd[ICOL],XYopr[i].x,XYopr[i].y,XYopr[i].r,XYopr[i].s);
         break;
```

文字列の表示と操作の保存は次のように $type = 4$ として，最大で 10000 文字を Chxy に詰め込んで，各レコードに文字数 len を保存している．

```
int  Nchar;
char Chxy[10000];
Xstring (int x0,int y0,char *chtx,int len){
  int i;
  XDrawString (Dpp,Wd,gd[ICOL],x0,y0,chtx,len);
  XYopr[Nopr].type=4; XYopr[Nopr].x=x0; XYopr[Nopr].y=y0; XYopr[Nopr].r=len;
  for(i=0; i<len ; i++ ) Chxy[Nchar+i] = *(chtx+i);
  Nchar+=len; Nopr+=1; }
```

再表示部分を示す．

```
            case 4 :
               len=XYopr[i].r;
               for(j=0; j<len; j++) chtx[j]=Chxy[k+j]; /*  *(chtx+j)=Chxy[k+j]; */
               XDrawString (Dpp,Wd,gd[ICOL],XYopr[i].x,XYopr[i].y,chtx,len);
               k+=len; break;
```

参考文献

[1] 荒川忠一. 数値流体工学. 東京大学出版会, 1994.

[2] 小倉敏布. 写真レンズの基礎と発展. 朝日ソノラマ, 1995.