美しいグラフ描画法

– C 言語から呼び出すポストスクリプト描画法 –

水島二郎

平成 23 年 9 月 19 日

まえがき

分野を問わず，具体的なデータから何らかの結論を得て，その結果を他人に説明し，自分の意見を伝えるにはデータを分かり易くしかも美しくグラフに表すことが重要である．たとえば，理工学あるいは経済学などの分野の研究においては，実験や計算あるいは統計調査の結果をグラフに表し，論文にまとめて専門誌に発表を行う．そのようなとき，自分の主張や新しい発見を読者に分かり易く伝えるためには，いくつかの要件が必要となる．それらは，グラフが正しく描かれていることと自分の主張が明確に伝わることだけでなく，グラフが美しく描かれていることが必要である．

論文の原稿をタイプライターで作成していた 1980 年代初期ごろまでは，烏口という製図用具やロットリングという製図用のペンを用いて手作業でグラフを描いていた．原稿を作成するのにコンピュータを用いるようになってからはカルコンプ社や渡辺測器などのプロッタを使用して図を描くようになった．最近では，収集したデータ・実験データ・計算結果をエクセルなどの表計算ソフトで整理し，付属の描画プログラムあるいはグラフ描画専用ソフトを用いて描いたグラフをレーザープリンタで出力するのが一般的となってきた．

この本で説明するのは，レーザープリンタやディスプレイに文字やグラフィックスを美しく描くために開発されたポストスクリプト (PostScript) と呼ばれる言語を用いてグラフや図形を描く方法である．ポストスクリプトは図形を描くために作られているので，図形を美しく描くための道具が揃っており優れた言語であるが，その言語形式として逆ポーランド形式を採用しているため，他の一般的な C や Java，Fortran などに馴染んだ人には得手勝手が異なり，その習得に長い時間が必要である．したがって，ここでは C・Java・Fortran のプログラムからポストスクリプトによるプログラムを出力する方法について説明する．

この本の一番大きな目的は美しいグラフの描き方を説明することである．グラフの縦横比・文字の挿入の仕方・線の種類と太さの違いによってグラフは良い印象を与えたり，悪い印象を与えることもある．離散的なデータを直線や曲線で結ぶと間違った結果を伝えてしまうことにもなりかねない．美しく正しいグラフの描き方を読者と共に考えていくつもりである．

著者は 1985 年頃からポストスクリプトを用いてグラフを描いており，2000 年に Fortran と C からポストスクリプト命令を呼び出す方法についてホームページ

`http://www1.doshisha.ac.jp/~jmizushi`

でそのマニュアルを公開して来たが，今回作成したプログラムは旧版と異なっているので注意していただきたい．また，このマニュアルは執筆の途中であり，暫定的なものなので，予告なく修正および追加を行う．

2011 年 3 月 著者

目 次

第1章　ページ記述言語–ポストスクリプト

1.1　ページ記述とポストスクリプト

人とひとが自分の思いや考えを伝えるときには，言葉を使うが，写真・図形・グラフなどを用いた方が分かり易いことも多い．ある場所を明確に伝えるには地図を描き，物の形を伝えるときには図形を描く．また，抽象的な概念も図にすると分かり易いこともある．科学の多くの分野では，グラフという概念は重要な位置を占めており，数学を頻繁に使用する理工学の分野はもちろんのこと，ほとんど数学を使わない分野においても，いくつかの量の間の関係を調べたり，因果関係を調べるときにはグラフを使用する．

このように，言葉や図形・グラフなどを用いて自分の思いを伝えるときには，文字や図形・写真などを美しく，分かり易く配置をしてページを構成する．本や雑誌・パンフレットや報告書はこのようにして作ったページの集まりである．あるいは，コンピュータのディスプレイの表示・ホームページなどもページの集まりと考えることができる．

コンピュータを用いて文章を作るときにはワープロを使用し，図を描くのにはドローソフト，写真を修正したり絵を描くときにはペイントソフトを用い，グラフを描くにはグラフ描画ソフトを使用する．それらのソフトを使用して得られた文章・図・絵・グラフなどを適切に配置してページを構成した後，コンピュータのファイルとして保存する．このとき，そのページを記述する言語がページ記述言語である．例えば，ホームページを記述するときには HTML (HyperText Markup Language) を使用する．HTML をもう少し一般化した言語が XML (eXtensible Markup Language) であり，ワープロで作った文章の保存などにも使用されている．

ポストスクリプト (PostScript) もこのような情報のレイアウトを目的としてアドビ社によって作られた言語である．**ポストスクリプト**[1]はアドビシステムズ社が主にプリンタ制御用に開発したページ記述言語である．レーザープリンタについては現在もまだプリンタ製造各社が独自の言語を使ってプリントするように設定している。たとえば、キャノンは LIPS という仕様、エプソンなどは ESC/P という仕様などを用いている。ポストスクリプトは現在市販されている多くのコンピュータ用プリンタに採用されており，コンピュータの画面を記述する言語としても採用されている．コンピュータの画面とプリンタ出力の両方で，同じ言語が使用されているときには，画面で確認したページと同じページがプリンタ出力として得られるので非常に便利である．

1.2　ポストスクリプト

ポストスクリプトはページ記述言語であり，C や Fortran のようにプログラミング言語の一種である．ただし，少々特徴のある言語であり，逆ポーランド記述式という記述法を採用しており，スタックという手法を多用し，C や Fortran などになじんだ人にとっては慣れるのに時間を要する．したがって，この章ではポストスクリプト言語を簡単に紹介し，ポストスクリプトによる描画法についても説明するが，次章では C を用いてポストスクリプト言語のプログラムを出力する方法に

[1]ポストスクリプト (PostScript) はアドビシステムズ社 (Adobe Systems Inc.) の登録商標である．

ついて説明する．科学技術の分野では計算のためにフォートラン (Fortran) やCが使用される．これらの言語は手続き形の言語と呼ばれ，人間が考える手順に従ってコンピュータが計算を行うようにコンピュータに指示をするための言語である．ポストスクリプトも同様に手続き形の言語である．しかし，フォートランやCは変数を多用して手順を記述するのに対して，ポストスクリプトはスタックと呼ばれる方法を使用する．この方法は料理をするときにお皿を積み重ねていった後，積み重ねた順と逆の順番にお皿を使用する方法によく似ている．一世代前のコンピュータでよく用いられたアゼンブラー言語を経験した人たちにはなじみ深いものであり，今でも電卓を使うときの方法にもよく似ている．この方法は逆ポーランド形式と呼ばれるものである．逆ポーランド形式言語とフォートランやCとはその考える手順がずいぶんと異なるため，一方の言語に慣れた人にとっては他方が非常に取っつきにくく感じられる．

1.3　ポストスクリプトの基本

ここではポストスクリプト言語の使い方について具体例を用いて説明を行う．前にも説明したように，ポストスクリプトでは逆ポーランド記述を用いる．逆ポーランド記述の方法を説明するために，ポストスクリプト言語を用いて簡単な計算を行う．例えば，$1 \times 2 + (3 + 4)/7$ をポストスクリプト言語を用いて表現すると次のようになる．

【プログラム例 1.1】

```
1:   1 2 mul 3 4 add 7 div add
```

このポストスクリプトによる表現は、“1 と 2 をかけたものと 3 と 4 を加えて 7 で割ったものを加える” という日本語による表現の語順に一致している。しかし、ポストスクリプト言語は科学技術計算によく用いられるプログラミング言語である C 言語やフォートラン (Fortran) 言語とは、記述法が全く異なる言語であることがよくわかる。この解説書は、科学技術計算で良く使われる、C 言語やフォートラン (Fortran) 言語を用い、図形やグラフを描くのに、それらの言語からポストスクリプト言語を呼び出して使う方法について説明する。

ポストスクリプトでは、仮想的なページであるカレントページに描画を行う。プログラム開始時点では、カレントページは空の状態である。このカレントページ上にカレントパスで直線や曲線などのパスを定義する。この定義では、まだ紙面に直線や曲線などのパスで定義した図形は描画されていない。このパスをカレントページに後述の stroke オペレータや fill オペレータを用いて、線を描いたり、パス内を塗りつぶしたりして使用する。ポストスクリプト言語では、さまざまな線や文字を描くのに座標を用いる。特に指定しなければ座標は、図 1.1 に示すように紙 (A4: 21.0cm × 29.7cm) の左下を原点にし、座標の単位は、ポイント (pt) であり 1 ポイントが 1/72 インチ（1 インチ ∼ 2.5cm）の大きさである。ただし、単位としてはセンチやインチなどを定義して使うことも可能である。

1.3.1　線と正方形

直線

最も簡単で基本的な例として 2 点間を直線で結ぶ例を考える。(72,144) から (216,144)(単位は pt) まで直線を引くプログラムは次のようになる。

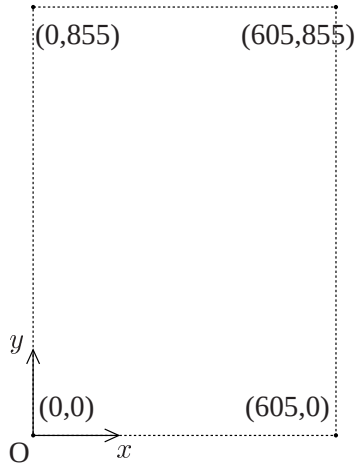

図 1.1 ポストスクリプトにおける座標. 単位はポイント (pt). 1 pt=1/72 inch =0.0353 cm

【プログラム例 1.2】

```
1: %!
2: newpath
3: 72 144 moveto
4: 216 144 lineto
5: stroke
6: showpage
```

図 1.2 直線.

出力結果を図 1.2 に示す。このプログラムを一行づつ説明する。1 行目の ”%! ”は、これ以降がポストスクリプト言語で記述されていることを示しており、表示プログラム (Ghostscript) やプリンタはポストスクリプト翻訳モードに入る。2 行目に newpath オペレータが呼び出されている。このオペレータはカレントパスの内容を空にし、新しいパスを開始することを宣言している。3 行目と 4 行目の数字が座標を表している。moveto オペレータで、その指定された座標にペンをおき、lineto オペレータの指定する座標まで線を引くことになる。つまりこの線は、用紙の左下から x 方向に 72pt(=1 インチ)、y 方向に 144pt(=2 インチ) の点から x の正の方向に 144pt(=2 インチ) の線を引くことになる。4 行目にある stroke オペレータは、作成したパスをカレントページ上に描画する働きをする。これにより、作成したパスが目に見える線になる。最後の showpage により、カレントページを表示する。

点線

プログラム 2 に 1 行加えると点線を書くことができる。

【プログラム例 1.3】

```
1: %!
2: newpath
3: [10 5] 0 setdash
4: 72 144 moveto
5: 216 144 lineto
6: stroke
7: showpage
```


図 1.3 [10-5] 点線.　10 ポイントの長さの線を引き，5 ポイントあけて再び 10 ポイントの長さの線を引く.

出力結果は図 1.3 に示すようになる。オペレータ setdash は今後描く線が点線であることを示す。[　] 内の数字で点線の間隔を設定し、[　] の右横の数字で点線をどこから書き始めるかを決めるのである。この例では、幅 10(pt) の線分を引き、5 の隙間をあける点線で、0 進んだところから点線を書き始める。同じ間隔の点線で、書き始めの位置を変えたものを並べると、「書き始め」の意味が理解できるだろう (図 1.3)。

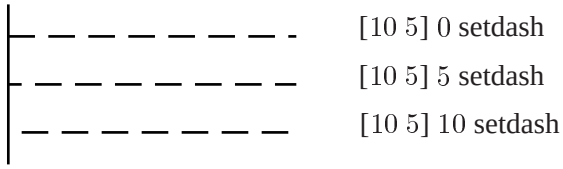

図 1.4 書き始めを変えたときの点線.

[　] 内の数字は 2 つでなくてもよく、[　] 内に並べられた数字を左から順に線、隙間、線、隙間、. . . とみなして点線を描く。例えば、上の例 [10 5] の代わりに [15 3 3 3] を用いると次のようになる (図 1.5)。

図 1.5 15-3-3-3 点線.

このように [ ] 内の数字の並びを考えれば、さまざまな点線を書くことが可能である。特に [ ] 0 setdash とするとこれは実線となる。

正方形

次に、上の例を応用して正方形を書くと以下のようになる (図 1.6)。

【プログラム例 1.4】

```
1: %!
2: newpath
3: 72 144 moveto
4: 72 504 lineto
5: 432 504 lineto
6: 432 144 lineto
7: 72 144 lineto
8: stroke
9: showpage
```

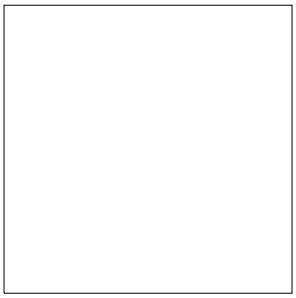

図 1.6 正方形.

このプログラムは、正方形を左下から時計まわりに描いている。ここで、始点と終点である左下を拡大したものを図 1.7 に示す。

図 1.7 より正方形の左下が欠けていることがわかる。これは線が太さを持っているために起こる。この問題は次のように closepath オペレータを用いて書き換えると解消される。

図 1.7 正方形の拡大図.

【プログラム例 1.5】

```
 1: %!
 2: newpath
 3: 72 144 moveto
 4: 72 504 lineto
 5: 432 504 lineto
 6: 432 144 lineto
 7: closepath
 8: 12 setlinewidth
 9: stroke
10: showpage
```

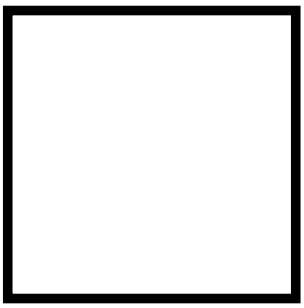

図 1.8 正確な正方形.

プログラム 5 の出力結果を図 1.8 に示す。オペレータ closepath は、始点（moveto オペレータ）と終点（最後の lineto オペレータ）を直線で結ぶ役割をする。そのため、不完全な正方形のプログラムにある最後の lineto オペレータがいらなくなる。closepath オペレータの次に新しく setlinewidth オペレータが呼び出されている。このオペレータは線の太さを決めるもので、この場合は 12/72 インチの太さで線を引くように命令している。

正方形の内部を塗りつぶすこともできる。次のプログラム 6 が正方形内部を塗りつぶす例である。

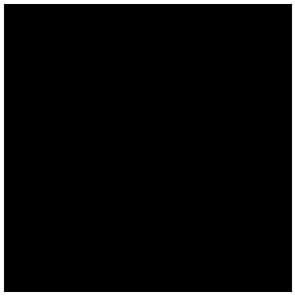

図 1.9 塗りつぶされた正方形.

【プログラム例 1.6】

```
 1: %!
 2: newpath
 3: 72 144 moveto
 4: 72 504 lineto
 5: 432 504 lineto
 6: 432 144 lineto
 7: closepath
 8: fill
 9: showpage
```

ここでは、パスをストロークするかわりに fill オペレータが呼び出されている。このオペレータはカレントパスの内部をインクで塗りつぶす働きをするので内部が黒く塗りつぶされた。色は黒だけでなく濃さを変えることもできる。

【プログラム例 1.7】

```
 1: %!
 2: newpath
 3: 72 144 moveto
 4: 72 504 lineto
 5: 432 504 lineto
 6: 432 144 lineto
 7: closepath
 8: 0.7 setgray
 9: fill
10: showpage
```

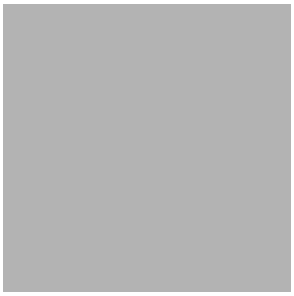

図 1.10 灰色の正方形.

オペレータ setgray で灰色の濃さを決定することができる。この例では、濃さを 0.7 としており、0 が黒に、1 が白に対応する。

これまでは、moveto と lineto を用いて正方形を描く方法を紹介してきたが、ポストスクリプトには矩形に関する便利なオペレータがいくつか用意されており、矩形のパス、矩形塗りつぶしなどは、次のように一行で記述することもできる。

```
%% (10,20) から、x 方向 50、y 方向 40 の長さの長方形を描画する
  10 20 50 40 rectstroke
%% (10,20) から、x 方向 50、y 方向 40 の長さの長方形の内部を塗る
  10 20 50 40 rectfill
```

1.3.2　曲線と円

円弧を使ってつくる曲線と円について説明する。円弧の曲率中心点の座標が $(144, 144)$ であり、半径が 216 である円弧を描くには次のプログラム 9 のようになる。出力結果を図 1.11 に示す。

【プログラム例 1.8】

```
1: %!
2: newpath
3: 144 144 216 0 90 arc
4: 15 setlinewidth
5: stroke
6: showpage
```

プログラムの 3 行目の arc オペレータが円弧を描く働きをする。その arc オペレータの前に 5 つの数字が並んでいる。左から円弧の曲率中心点の座標 (x,y)、曲率半径、x 軸の正方向から反時計回りに測った開始点と終点の角度である。円は、角度の開始点と終点を 0 °から 360 °にすると描くことができる。

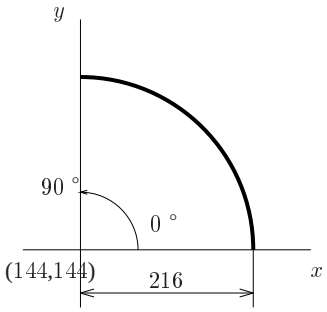

図 1.11 円弧.

また、arc オペレータの代わりに、arcn オペレータを使うと図 1.12 が描ける。この図は、曲率中心点を (288,288) にとり、半径 144 の円弧である。arc オペレータは、角度の開始点と終点を反時計回りに結ぶのに対して、arcn オペレータは時計回りに結ぶ。

【プログラム例 1.9】

```
1: %!
2: newpath
3: 288 288 144 0 90 arcn
4: 15 setlinewidth
5: stroke
6: showpage
```

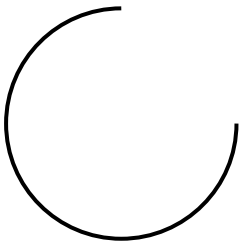

図 1.12 円弧（逆回転）.

ベジェ曲線

４点で構成される３次のベジェ曲線のパスを描くこと方法について説明する。図 1.9 のように、４点 P_0、P_1、P_2、P_3 が与えられたときに、線分 $\mathrm{P}_0\mathrm{P}_1$、$\mathrm{P}_1\mathrm{P}_2$、$\mathrm{P}_2\mathrm{P}_3$ を $t : 1-t$ の比で内分する点を Q_0、Q_1、Q_2 とし、線分 $\mathrm{Q}_0\mathrm{Q}_1$、$\mathrm{Q}_1\mathrm{Q}_2$ を $t : 1-t$ の比で内分する点を R_0、R_1 とし、さらに、線分 $\mathrm{R}_0\mathrm{R}_1$ を $t : 1-t$ の比で内分する点 S_0 とする。このようにして描かれる点 $\mathrm{S}_0(t)$ が $0 \leq t \leq 1$ の範囲で t を変化させたときに描く曲線が３次のベジェ曲線である。

ポストスクリプトでは、curveto オペレータによってこのようなベジェ曲線を描くことができる。カレントポイントを $\mathrm{P}_0(x_0, y_0)$ として、

```
x1 y1 x2 y2 x3 y3 curveto
```

とすれば、４点 $\mathrm{P}_0(x_0, y_0)$、$\mathrm{P}_1(x_1, y_1)$、$\mathrm{P}_2(x_2, y_2)$、$\mathrm{P}_3(x_3, y_3)$、からなる３次のベジェ曲線を描くことができる。具体的には、図 1.9 のようなベジェ曲線は、次のようにして描くことができる。

```
10 10 moveto %% カレントポイントを (10,10) に移動
80 100 140 80 160 20 curveto
        %% カレントポイントと３点 (80,100)-(140,80)-(160,20) を
        %% 使ってベジェ曲線のパスを作成する
stroke  %% パスを描画する
```

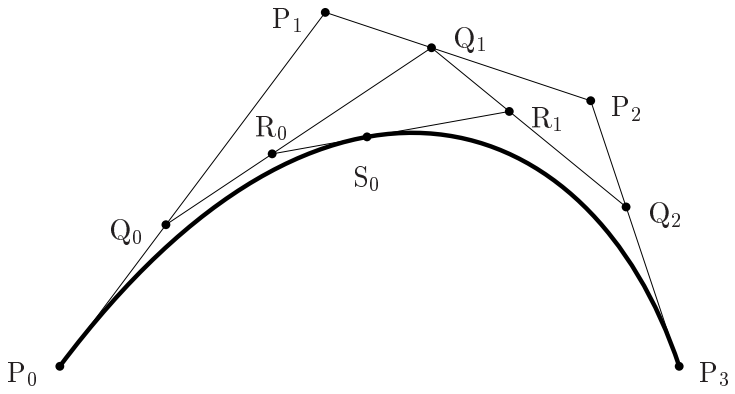

図 1.13 ベジェ曲線

1.3.3　線の形状

線の太さ

これまでにも何度か使ってきたが、線の太さを指定するには、setlinewidth オペレータを用いる。

```
1   setlinewidth %% 線の太さを 1   (pt) に指定
3.5 setlinewidth %% 線の太さを 3.5(pt) に指定
0   setlinewidth %% 線の太さを 0   (pt) に指定
```

太さ 0pt は、線が描画されないわけではなく、表示できる最も細い線で描かれる。そのため、0pt に指定して描画された線は、いくら拡大表示しても線は太くならない。

線の端点の形状

太さのある線の始点と終点の形状を setlinecap で指定することができる。端点の形状は３種類あり、図 1.3.3 のようになる。パスと描画される（太さのある）線との関係がわかりやすいように、パスを白線で、パスの端点を白丸で描いている。

```
0 setlinecap %% Butt cap
1 setlinecap %% Round cap
2 setlinecap %% Projecting square cap
```

図 1.14 端点の形状

折線の角の形状

太さのある折線の角の形状を setlinejoin で指定することができる。端点の形状は３種類あり、図 1.3.3 のようになる。パスと描画される（太さのある）線との関係がわかりやすいように、パスを白線で、パスの端点および角になる点を白丸で描いている。

```
0 setlinejoin %% Miter join
1 setlinejoin %% Round join
2 setlinejoin %% Bevel join
```

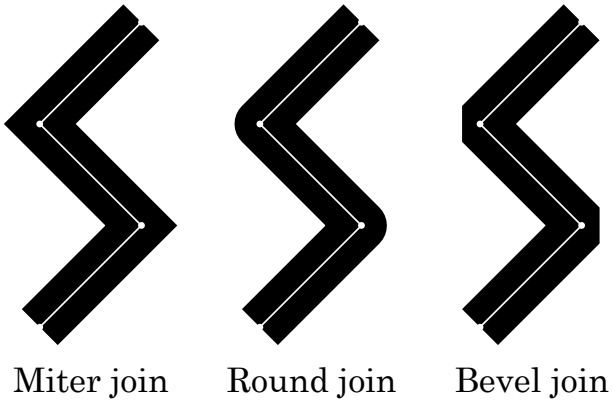

図 1.15 角の形状

1.3.4　塗りつぶし

ポストスクリプトで描いたパスの内部を塗りつぶす方法には fill と eofill の 2 通りの方法がある。

ワインディング規則

fill オペレータは、カレントパスが含む領域全体をカレントカラーで塗りつぶす。図 1.3.4（左）に示すようなパスが自分自身と交差する星型のパスの塗りつぶしでは、その内部が全て塗りつぶされる。また、カレントパスが、１つのパスと内部のもう１つのパスで構成されるような場合は、図 1.3.4（中）のように、各パスの方向が同方向であれば、内部領域も塗りつぶされるが、図 1.3.4（右）に示すように、各パスの方向が逆方向であれば、内側のパスの内部は塗りつぶされない。このような、パスの内部領域の判定方法をワインディング規則と呼ぶ。

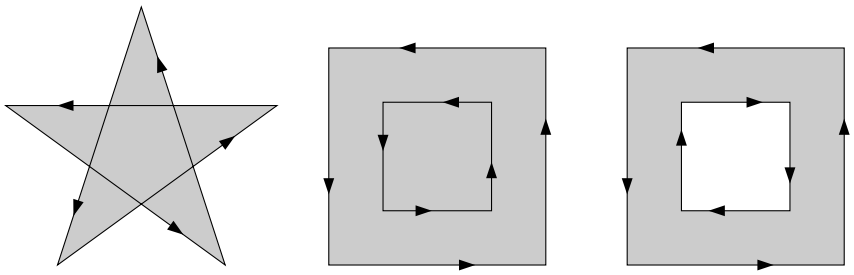

図 1.16 fill による塗りつぶし（ワインディング規則）

奇偶規則

一方、eofill オペレータでは、図 1.3.4 に示すように、パスが交差する度に塗りつぶしの判定が変わり、星型のパスで分割される内部領域は塗りつぶされなくなる。また、パスの内部に別のパスが含まれる場合は、パスの方向に関係なく内部領域は塗りつぶされない。このような、パスの内部領域の判定方法を奇偶規則と呼ぶ。

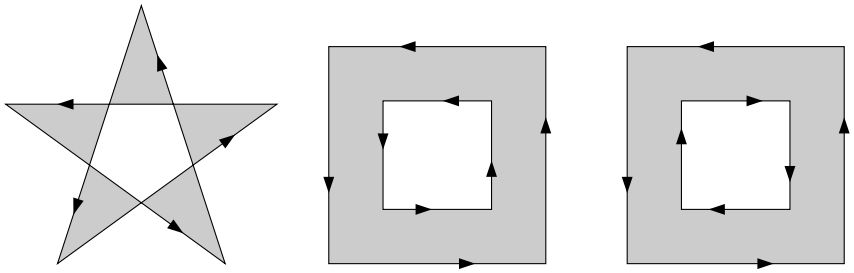

図 1.17 eofill による塗りつぶし（奇偶規則）

1.3.5 文字列の挿入

フォントの設定

文字列を表示するには、あらかじめ利用するフォントを宣言しておく必要がある。最も簡単な方法は、「名前」と「大きさ (pt)」を使って指定する方法で、

```
/Times findfont 8 scalefont setfont
```

のようにする方法である。上記の例では、Times の 8pt のフォントに設定する。実際にカレントポイントに文字列を表示するには、次のように表示させたい文字列を括弧（）でくくり、その後に show オペレータを実行すればよい。

```
(ABC) show
```

図 1.3.5 に、さまざまなフォントを指定したときの英数字の出力結果の一覧を示す。

Courier - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Courier-Italic - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Courier-Bold - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Courier-BoldItalic - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

Times - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Times-Italic - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Times-Bold - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Times-BoldItalic - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

Century - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Century-Oblique - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Century-Bold - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Century-BoldOblique - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

Helvetica - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Helvetica-Oblique - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Helvetica-Bold - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Helvetica-BoldOblique - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Helvetica-Narrow - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

Palatino - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Palatino-Italic - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Palatino-Bold - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Palatino-BoldItalic - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

AvantGarde-Book - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
AvantGarde-BookOblique - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
AvantGarde-Demi - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
AvantGarde-DemiOblique - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

Bookman-Light - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Bookman-LightItalic - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Bookman-Demi - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Bookman-DemiItalic - 0.123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

Symbol – 0.123456789 ΑΒΧΔΕΦΓΗΙϑΚΛΜΝΟΠΘΡΣΤΥςΩΞΨΖ αβχδεφγηιφκλμνοπθρστυϖωξψζ

図 1.18 さまざまなフォントの表示例

第2章　C で描くポストスクリプト図形

C を用いてグラフや図を描くためにはそれぞれのコンパイラに付属のグラフィックス・ライブラリや OpenGL と呼ばれる汎用ライブラリを用いたり，あるいは OS がウィンドウズ (Windows) の場合にはグラフィカル デバイス インターフェイス (GDI) を使うのが一般的であるが，それにはかなりの熟練を要する．この章では，ポストスクリプト言語により簡単にグラフや図を描くための C の道具 (ツール) を使用する方法を説明する．この道具を使うといくつかの関数を呼び出すだけで簡単にきれいな図を描くことができる．すなわち，ポストスクリプト言語による描画プログラムを出力するプログラムを C で記述するのである．

2.1　C で書いたプログラムをコンパイルする方法

第 1 章でも説明したように，**ポストスクリプト**はアドビシステムズ社が主にプリンタ制御用に開発したページ記述言語であり，C や Fortran のようなプログラミング言語の一種である．ただし，ポストスクリプトは少々特徴のある言語であり，逆ポーランド記述式という記述法を採用しており，スタックという手法を多用し，C や Fortran などになじんだ人にとっては慣れるのに時間を要する．したがって，ここでは，C からポストスクリプト描画を行うための関数を呼び出すだけで，ポストスクリプト言語による描画プログラムを出力する方法について説明する．この方法はプリプロセッサの方法に似ており，著者が 25 年程前に開発し，図形を描くのに使用してきた方法である．この章で説明する関数を使用するときに必要となるライブラリ `pssub.c` とヘッダファイル `pssub.h` は著者のホームページ (URL: `http://www1.doshisha.ac.jp/~jmizushi/`) からダウンロードすることができる．これらはこれまで著者が提供してきた `psbasic.c` と `ps.h` の改訂版にあたる．改訂版の `pssub.c` は旧版の `psbasic.c` に比べて，描画の手順と関数名が少し異なるので，旧版の利用者には注意が必要である．また，同じホームページには旧版のポストスクリプトの説明，サンプルプログラム，ポストスクリプトのマニュアルなども置かれている．新版のマニュアル等については順次整備をしていく予定である．

ポストスクリプトにより図形を描くための関数を用いて，C でプログラムを書き，そのファイル名を `sample.c` としよう．このプログラムの中では，ヘッダファイル `pssub.h` を読み込んでいるので，`pssub.h` をプログラム `sample.c` と同じフォルダ (ディレクトリ) に置くか，C の標準関数のヘッダファイルを入っているフォルダにコピーしておく．ここでは，ファイル `sample.c`, `pssub.c`, `pssub.h` は作業フォルダ `E:\ps>` にあるものとする．次のコマンドを実行してコンパイルを行う．

```
E:\ps>cl sample.c pssub.c
```

ここで，`cl` は Visual C++ のコンパイル命令である．これ以外のコンパイラを使用しており，コンパイル命令が `cc` であれば，`cl` の代わりに `cc` を用いる．もし，コンパイル時にエラーがなければ実行ファイル `sample.exe` ができるので，これを実行する．

```
E:\ps>sample
```

あるいは，先に pssub.c のみをコンパイルして pssub.obj を作成し，後に sample.c をコンパイルし，pssub.obj と結合する方法もある．このときは，

```
E:\ps>cl/c pssub.c
```

とすると，psbasic.obj ができるので，次に

```
E:\ps>cl sample.c pssub.obj
```

として，実行ファイル sample.exe を作る．実行の方法はこれまでと同様である．このように，いくつかに分かれたプログラムの内，main 関数を含まない関数のみからなるプログラムを先にコンパイルし，最後に main 関数を含むプログラムをコンパイルして，結合する方法を**分割コンパイル**という．

2.2 パスと図形出力

ポストスクリプトを用いて図を描くときには**ポインタ** (pointer) と**パス** (path) という考え方が大切である．ポインタを移動することにより，パスというポインタの軌跡を描く．パスで軌跡を描くだけではまだ何の図形も出力されないが，パスで表された図形に墨入れ (stroke) または塗りつぶし (fill) をすることによってパスで表された図形が出力される．ポストスクリプトによる描画を簡単に理解するには，銅版画をイメージするのが便利である．まず，新しいパスを描き始めようとするときは，newpath と記述し，次に，パスの出発点 (ポインタの初期値) を決める．出発点から彫刻刀を用いてパスを描いていくことになるが，パスには彫刻刀で彫り込みを入れながら描くパス plot(x,y,2) (ポストスクリプト命令では lineto) と彫り込みを入れずに表面のみをなぞるパス plot(x,y,3)(moveto) がある．パスを描き終えると，墨を入れる (stroke) ことになるが，彫り込みを入れたパスには墨が入っており，表面をなぞったパスには墨が入らないので，銅版に紙を重ねて擦ると彫り込みを入れた線だけが紙に転写される (showpage)．パスが閉じているときに，その内部を掘り下げて (fill) しまうと，内部全体に墨が入り，塗りつぶされてしまうことになる．ここで，墨を入れる (stroke) タイミングに注意が必要である．あるパスを描いた後に，墨を入れる (stroke) 前に新しいパスを宣言する (newpath) とせっかく描いたパスに墨が入らずに消えてしまう．したがって，パスを描いた後，新しいパスを宣言する前に必ず墨を入れるか内部を塗りつぶす必要がある．また，各パスはそれぞれ，墨の濃さや線の太さなどの特性をもっているが，あるパスの途中でそれらの特性を変更することはできない．たとえば，薄い墨で描く線と濃い墨で描く線を混在させるときには，一つのパスに薄い墨を入れた後に，新しいパスを描いてそのパスに濃い墨を入れることになる．

2.3 世界座標と機器座標

この節では 2 次元図形処理の基本である 3 つの座標系，すなわち世界座標系・正規座標系・機器座標系について説明する．これから描こうとする図に座標系 (**世界座標**，図 2.1(a)) を導入する．このことは，現実の世界または図形を矩形領域に切り出すことに対応している．この座標系は任意

にとることができる．長さの単位も任意であるが，領域をできるだけ正方形に近く切り出しを行うと，図の縦横比のバランスが良い美しい図が得られる．世界座標系に描かれた図をコンピュータの機器座標系 (出力座標，図 2.1(c)) に変換してグラフィック出力する．すなわち，切り出した世界座標における矩形領域は**機器座標**に線形写像される．元の世界座標系における図形の縦横比が機器座標系での出力図形と同じ縦横比を保つためには，世界座標系と機器座標系における矩形領域の縦横比を同じにする必要がある．世界座標で描かれた図形を機器座標に線形写像するために，まず世界座標を正規座標に変換する．正規座標系で図形は左下が $(0,0)$ であり，右上が $(1,1)$ である正方形領域に写像される．世界座標系における矩形領域の左下を (x_1, y_1) とし，右上を (x_2, y_2) とすれば，世界座標の点 (x, y) から正規座標 (ξ, η) への変換は

$$\xi = \frac{x - x_1}{x_2 - x_1}, \quad \eta = \frac{y - y_1}{y_2 - y_1} \tag{2.3.1}$$

で表される．また，機器座標系における矩形領域の左下を (X_1, Y_1) とし，右上を (X_2, Y_2) とすれば，正規座標 (ξ, η) から機器座標 (X, Y) への変換は

$$X = \xi(X_2 - X_1) + X_1, \quad Y = \eta(Y_2 - Y_1) + Y_1 \tag{2.3.2}$$

で表される．出力機器は通常 A4 の紙 (21.0 cm $\times$ 29.7 cm) を想定しているため，$X_1 = 0$，$X_2 = 21.0$，$Y_1 = 0$，$Y_2 = 29.7$ であるが，ポストスクリプト描画関数では覚えやすくするために，$X_1 = 0$，$X_2 = 1$，$Y_1 = 0$，$Y_2 = 1$ と定義している．

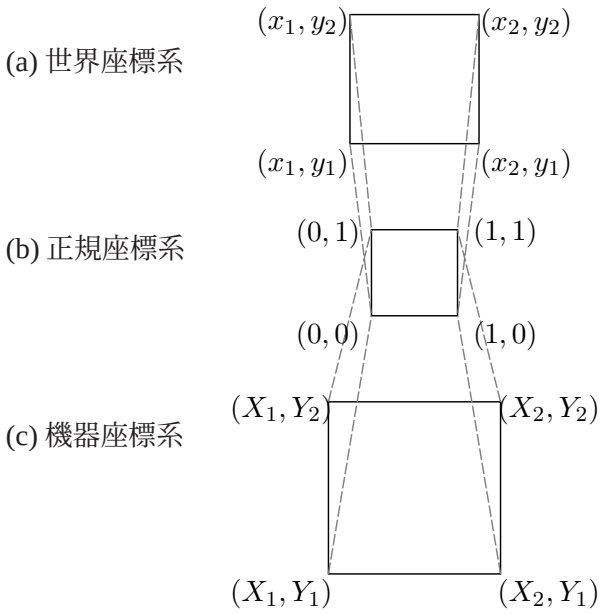

図 2.1 世界座標と正規座標と機器座標.

世界座標を指定する C のポストスクリプト関数は `xyworld(x1,y1,x2,y2)` である．また，機器座標を指定する関数は `viewport(X1,Y1,X2,Y2)` である．たとえば，`xyworld(-1.0,-1.0,1.0,1.0)` のように世界座標を指定して，機器座標を `viewport(0.2,0.2,0.8,0.8)` と指定すると，世界座標 `(-1,-1,1,1)` を機器座標 `(0.2,0.2,0.8,0.8)` に写像することを意味している．

2.4 ポストスクリプト描画の基本

2.4.1 直線・折れ線とそれらの性質 (線幅・線種・明度・色)

ポストスクリプト描画ライブラリ pssub.c を使用して描画を行うための基本的なプログラムを説明する．一般に，どのような複雑な図形でもいくつかの短い線分に分割してそれらの集合としてパスを描くことが可能である．パスを描くための基本的な関数は plot(x,y,ipen) である．この関数は現在のポインタの位置から座標 (x,y) へポインタを移動して，パスを描く関数であり，ipen が 3 のときは彫り込みを入れずに移動し，2 のときは彫り込みを入れながら移動する．この関数だけを使っても多くの図形を描くことが可能である．最初の例として次の 4 角を描くプログラム (lines1.c) を見てみよう．

【プログラム例 2.1】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:       init();
 5:       viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:       newpath();
 7:       plot(0.1, 0.1, 3); plot(0.9, 0.1, 2);
 8:       plot(0.9, 0.9, 2); plot(0.1, 0.9, 2);
 9:       closepath();
10:       stroke();
11:       fin();
12: }
```

プログラム例 2.1(lines1.c) は最も基本的なプログラムの例であり，このプログラムを理解するだけで多彩な図を描くことも可能である．第 1 行目はポストスクリプト関数の定義を呼び出すための準備 (ヘッダファイル pssub.h の読み込み) である．ヘッダファイル pssub.h はプログラム lines1.c と同じディレクトリにあるとしている．第 4 行目の init() はポストスクリプト描画の開始宣言である．この関数の中で，描画のためのポストスクリプトプログラムを出力するための temp1.ps というファイルを開くことを指定している．第5行目の viewport(0.2,0.2,0.8,0.8) は機器座標の指定であり，A4 の紙 (21cm×29.7cm) の下部 21cm×21cm の正方形の領域を (0, 0)-(1, 1) として，その中の (0.2,0.2)-(0.8,0.8) の範囲に図を描くことを表している．xyworld(0.0,0.0,1.0,1.0) は世界座標の指定であり，(0.0,0.0)-(1.0,1.0) の範囲に描いた図が，機器座標の (0.2, 0.2)-(0.8, 0.8) に写像される．第 6 行目で新しいパスを作る宣言をしている．このとき，ポインタの位置は定まっていない (不定である) ので，第 7 行目で彫り込みを入れずに (ipen=3) ポインタを (0.1,0.1) に移動し，その後に彫り込みを入れながら (ipen=2) ポインタを (0.9,0.1) まで移動する．ここで，注意が必要である．新しいパスを作る宣言したとき，ポインタの位置が不定のまま (関数 plot を ipen=3 を指定して呼ぶ前に)ipen=2 を指定して関数 plot を呼ぶとポストスクリプト表示時にエラーとなる．第 8 行目でも同様に，ポインタを (0.9,0.9) および (0.1,0.9) へ移動する．4 角形を描くために，第 9 行目で closepath() によりパスを閉じる．第 10 行目の stroke() は彫り込みを入れた線に墨を入れる命令であり，第 11 行目の fin() で紙に転写をして，ファイル temp1.ps を閉じる．これ

を出力すると図 2.2(a) のようになる．また，第 10 行目で stroke() の代わりに fill() とすれば，閉じたパスの内部を塗りつぶして，図 2.2(b) のような図形が描かれる．

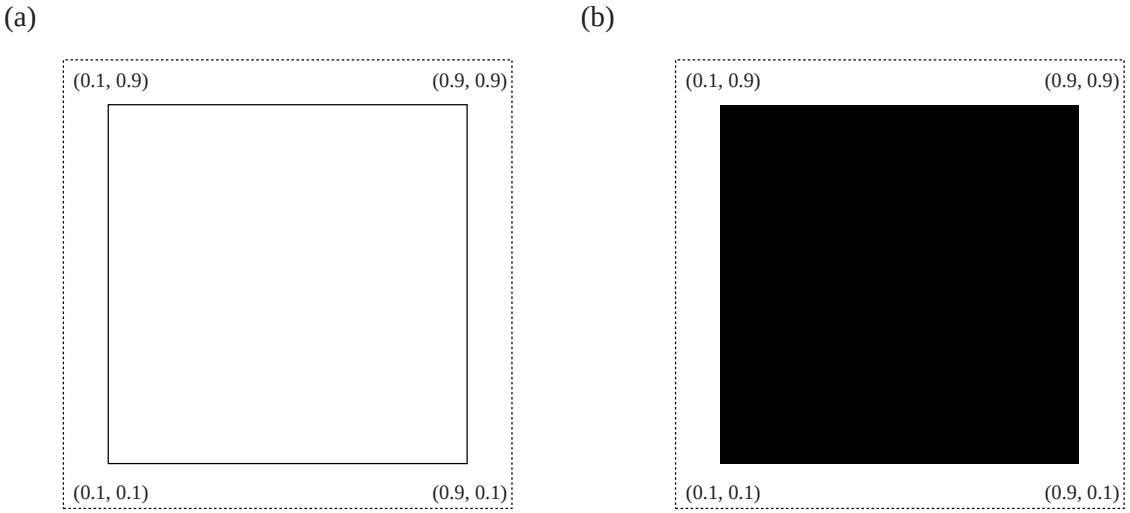

図 2.2 4 角形. (a) 4 角形の外形．(b) 4 角形の塗りつぶし．

プログラム例 2.1 で見たように，関数 plot(x,y,ipen) を用いてポインタを移動することにより，いろいろな図形を簡単に描くことができる．この関数は出力機器である紙の横方向に x 座標をとり，縦に y をとってポインタの位置を指定する．描きたい図形によっては，紙を回転して，回転した座標系でポインタを指定する方が便利なことがある．そのような目的で作った関数が plotrot(x,y,t,ipen) である．この関数は，座標を原点 (ポストスクリプト座標の原点であり，デフォルトでは紙の左下が原点) のまわりに角度 t° だけ反時計まわりに回転した座標 (x,y) でポインタの位置を指定する．関数 plotrot(x,y,t,ipen) の例を次のプログラム (linesrot1.c) で見てみよう．

【プログラム例 2.2】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:       init();
 5:       viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:       newpath();
 7:       plotrot(0.4,0.1,30.0,3);plotrot(0.8,0.1,30.0,2);
 8:       plotrot(0.8,0.5,30.0,2);plotrot(0.4,0.5,30.0,2);
 9:       closepath();
10:       stroke();
11:       fin();
```

```
12: }
```

プログラム例 2.2(linesrot1.c) では，紙の横方向から 30° 回転した座標系から見て，一辺の長さが 0.4 の正方形を描いている．第 7 行目の plotrot(0.4,0.1,30.0,3) は，30° 回転した座標系でポインタを (0.4,0.1) に移動する命令である．その後に彫り込みを入れながら (ipen=2) ポインタを移動することにより閉じた正方形を描く．このプログラムにより，図 2.3 のような図形が描かれる．

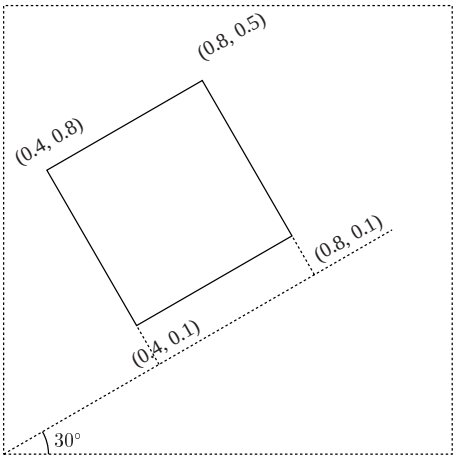

図 2.3 反時計方向に 30° 回転した 4 角形.

図形を描くときには，実線や点線などいろいろな線種を使用することができる．次の例 (lines2.c) でどのような線種が定義されているのか見てみよう．

【プログラム例 2.3】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    init();
 5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:    linety(1); newpath(); plot(0.1,0.1,3); plot(0.9,0.1,2); stroke();
 7:    linety(2); newpath(); plot(0.1,0.2,3); plot(0.9,0.2,2); stroke();
 8:    linety(3); newpath(); plot(0.1,0.3,3); plot(0.9,0.3,2); stroke();
 9:    linety(4); newpath(); plot(0.1,0.4,3); plot(0.9,0.4,2); stroke();
10:    linety(5); newpath(); plot(0.1,0.5,3); plot(0.9,0.5,2); stroke();
11:    linety(6); newpath(); plot(0.1,0.6,3); plot(0.9,0.6,2); stroke();
12:    linety(7); newpath(); plot(0.1,0.7,3); plot(0.9,0.7,2); stroke();
13:    linety(8); newpath(); plot(0.1,0.8,3); plot(0.9,0.8,2); stroke();
```

```
14:     linety(9); newpath(); plot(0.1,0.9,3); plot(0.9,0.9,2); stroke();
15:     stroke();
16:     fin();
17: }
```

プログラム例 2.3(lines2.c) では，9 種の異なる線種で長さ 0.8 の線分を描いている．第 6 行目の (linety(1)) は実線を指定する命令である．パスを開始する命令 newpath() に続いて，plot(0.1,0.1,3) でポインタを (0.1,0.1) に移し，plot(0.9,0.1,2) で彫り込みを入れながらポインタを (0.9,0.1) へ移動する．作成したパスに，stroke() で墨入れを行う．このように，パスを作成した後に stroke() 命令を入れることにより墨入れが行われる．第 7 行目以降，第 14 行目までは線種を変えながら，同じように線分を描いている．各行で，stroke() 命令を忘れて最後に (第 14 行目) だけに stroke() 命令を入れると全ての線が linety(9) で指定された線種となってしまうので，注意が必要である．関数 linety(ik) で，引数 ik と線種との関係は

ik=1: 実線. ik=2: 点線. ik=3: 長い点線. ik=4: 短い破線.
ik=5: 破線. ik=6: 長い破線. ik=7: 短い 1 点鎖線. ik=8: 1 点鎖線.
ik=9: 長い 1 点鎖線.

である．このプログラムの実行結果は図 2.4 のようになる．

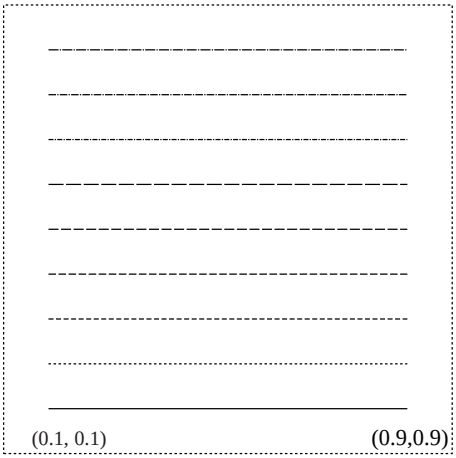

図 2.4 線種. 実線・点線・破線・鎖線.

線を描くときに，いろいろな線種を使用できるだけでなく，線の太さを自由に設定することができる．次のプログラム例 2.4 では 1pt(ポイント，1/72 inch=0.376 mm) から 9pt までの太さを描いている．

【プログラム例 2.4】

```
1:#include "pssub.h"
2:void main()
```

```
 3:{
 4:  init();
 5:  viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0); linety(1);
 6:  linewidth(1.0); newpath(); plot(0.1,0.1,3); plot(0.9,0.1,2); stroke();
 7:  linewidth(2.0); newpath(); plot(0.1,0.2,3); plot(0.9,0.2,2); stroke();
 8:  linewidth(3.0); newpath(); plot(0.1,0.3,3); plot(0.9,0.3,2); stroke();
 9:  linewidth(4.0); newpath(); plot(0.1,0.4,3); plot(0.9,0.4,2); stroke();
10:  linewidth(5.0); newpath(); plot(0.1,0.5,3); plot(0.9,0.5,2); stroke();
11:  linewidth(6.0); newpath(); plot(0.1,0.6,3); plot(0.9,0.6,2); stroke();
12:  linewidth(7.0); newpath(); plot(0.1,0.7,3); plot(0.9,0.7,2); stroke();
13:  linewidth(8.0); newpath(); plot(0.1,0.8,3); plot(0.9,0.8,2); stroke();
14:  linewidth(9.0); newpath(); plot(0.1,0.9,3); plot(0.9,0.9,2); stroke();
15:  fin();
16: }
```

プログラム例 2.4(`lines3.c`) は，プログラム例 2.3 とほぼおなじであり，第 5 行目で `linety(1)` により線種を実線と指定し，第 6 行目以降は `linewidth(w)` で線の太さ (線幅) を指定している．ここで，`w` は実数で，特にその値の範囲に制限はない．普通の線画であれば，`w=1.0` で十分である．このプログラムの実行結果 2.5 を参考にして線幅を決めればよい．

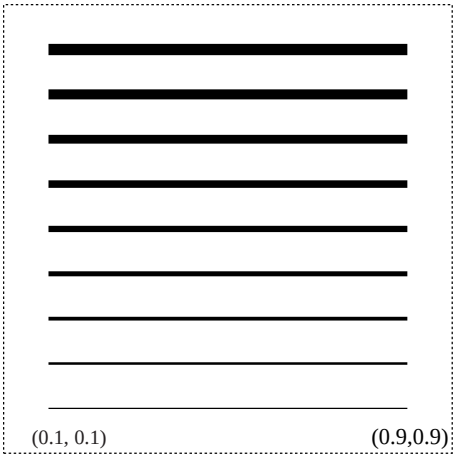

図 2.5 線の太さ．

線の濃さ (黒さ) を設定することもできる．次のプログラム例 2.5 は下から順に黒色から灰色の線を描いている．

【プログラム例 2.5】

```
1:#include "pssub.h"
2:void main()
```

```
 3:{
 4:  init();
 5:  viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0); linety(1);
 6:  linewidth(2.0);
 7:  setgray(0.0); newpath(); plot(0.1,0.1,3); plot(0.9,0.1,2); stroke();
 8:  setgray(0.1); newpath(); plot(0.1,0.2,3); plot(0.9,0.2,2); stroke();
 9:  setgray(0.2); newpath(); plot(0.1,0.3,3); plot(0.9,0.3,2); stroke();
10:  setgray(0.3); newpath(); plot(0.1,0.4,3); plot(0.9,0.4,2); stroke();
11:  setgray(0.4); newpath(); plot(0.1,0.5,3); plot(0.9,0.5,2); stroke();
12:  setgray(0.5); newpath(); plot(0.1,0.6,3); plot(0.9,0.6,2); stroke();
13:  setgray(0.6); newpath(); plot(0.1,0.7,3); plot(0.9,0.7,2); stroke();
14:  setgray(0.7); newpath(); plot(0.1,0.8,3); plot(0.9,0.8,2); stroke();
15:  setgray(0.8); newpath(); plot(0.1,0.9,3); plot(0.9,0.9,2); stroke();
16:  fin();
17: }
```

プログラム例 2.5(`lines4.c`) では，第 5 行目で `linety(1)` により線種を実線と指定し，第 6 行目で `linewidth(2.0)` により線の幅を 2 pt と指定している．第 7 行目以降は `setgray(g)` で線の濃さ (黒さ) を指定している．ここで，`g` は実数で，0.0 から 1.0 の範囲の値をもつ．`g=0.0` は黒であり，`g=1.0` は白を表す．背景が黒のときには，白で線を描くと見えるが，白の背景に白で線を描いても見ることができない．このプログラムの実行結果は図 2.6 のようになる．

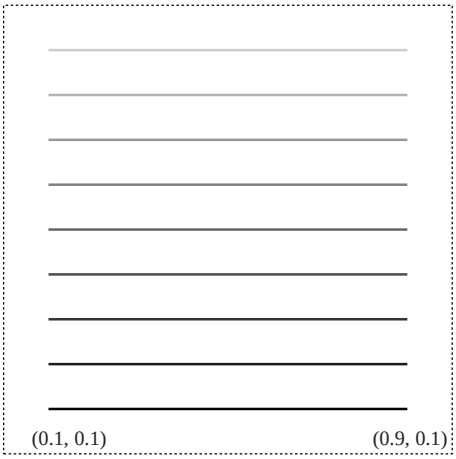

図 2.6 線の濃さ．

線に色を指定することも可能である．次のプログラム例 2.6 では，プリンタなどで色を指定する方法の一つである cmyk という方法で各線の色を指定している．ここで，c はシアン (Cyan)，m はマゼンタ (Magenta)，y はイエロー (Yellow)，k は黒 (キー・プレート，Key Plate) を表している．

【プログラム例 2.6】

```
 1:#include "pssub.h"
 2:void main()
 3:{
 4:  init();
 5:  viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0); linety(1);
 6:  linewidth(2.0);
 7:  setcmyk(1.0,0.0,0.0,0.0); newpath(); plot(0.1,0.1,3); plot(0.9,0.1,2); strok
 8:  setcmyk(0.5,0.5,0.0,0.0); newpath(); plot(0.1,0.2,3); plot(0.9,0.2,2); strok
 9:  setcmyk(0.0,1.0,0.0,0.0); newpath(); plot(0.1,0.3,3); plot(0.9,0.3,2); strok
10:  setcmyk(0.0,0.5,0.5,0.0); newpath(); plot(0.1,0.4,3); plot(0.9,0.4,2); strok
11:  setcmyk(0.0,0.0,1.0,0.0); newpath(); plot(0.1,0.5,3); plot(0.9,0.5,2); strok
12:  setcmyk(0.0,0.0,0.5,0.5); newpath(); plot(0.1,0.6,3); plot(0.9,0.6,2); strok
13:  setcmyk(0.0,0.0,0.0,1.0); newpath(); plot(0.1,0.7,3); plot(0.9,0.7,2); strok
14:  setcmyk(0.5,0.0,0.0,0.5); newpath(); plot(0.1,0.8,3); plot(0.9,0.8,2); strok
15:  fin();
16: }
```

プログラム例 2.6(`lines5.c`) では，第 7 行目以降において `setcmyk(c,m,y,k)` で線の色を指定している．ここで，`c,m,y,k` は実数であり，色をシアン (Cyan)，マゼンタ (Magenta)，イエロー (Yellow)，黒 (Key Plate) に分解したときのそれぞれの色成分の割合である．プリンタではこれらの混合色で各線が描かれる．このプログラムの実行結果である図 2.7 を参考にして色を指定する．

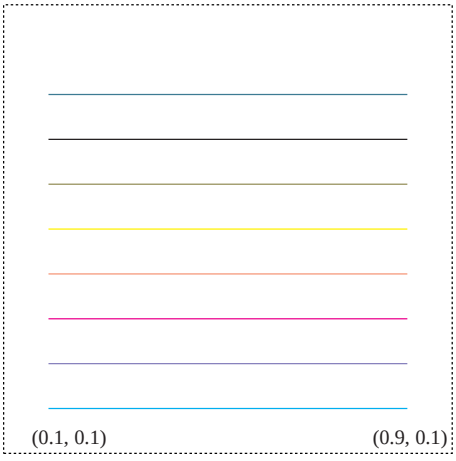

図 2.7 cmyk 法による色の指定.

プログラム例 2.6 では，cmyk という方法で色を指定したが，この方法は染料や塗料などの配色を行うときに使用する方法であり，色を混ぜていくとそれらの原色を混ぜていくと黒に近づく．一方，ディスプレイなどの光源から出る光を混ぜて色を表現する方法には RGB という方法がある．

３原色として，赤 (Red)・緑 (Green)・青 (Blue) の３つの原色を混ぜて色を表現し，これら３つの色を全部混ぜると白となる．次のプログラム例 2.7 では，rgb の方法で色を指定する．

【プログラム例 2.7】

```
 1:#include "pssub.h"
 2:void main()
 3:{
 4:  init();
 5:  viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0); linety(1);
 6:  linewidth(2.0);
 7:  setrgb(1.0,0.0,0.0); newpath(); plot(0.1,0.1,3); plot(0.9,0.1,2); stroke();
 8:  setrgb(0.5,0.5,0.0); newpath(); plot(0.1,0.2,3); plot(0.9,0.2,2); stroke();
 9:  setrgb(0.0,1.0,0.0); newpath(); plot(0.1,0.3,3); plot(0.9,0.3,2); stroke();
10:  setrgb(0.0,0.5,0.5); newpath(); plot(0.1,0.4,3); plot(0.9,0.4,2); stroke();
11:  setrgb(0.0,0.0,1.0); newpath(); plot(0.1,0.5,3); plot(0.9,0.5,2); stroke();
12:  setrgb(0.5,0.0,0.5); newpath(); plot(0.1,0.6,3); plot(0.9,0.6,2); stroke();
13:  fin();
14: }
```

プログラム例 2.7(`lines6.c`) では，第 7 行目以降において `setrgb(r,g,b)` で線の色を指定している．ここで，`r,g,b` は実数であり，色を赤 (r, Red)，緑 (g, Green)，青 (b, Blue) の重ね合わせで表現している．したがって，`setrgb(1.0,1.0,1.0)` は白を表し，`setrgb(0.5,0.5,0.5)` は灰色を表す．図 2.8 はプログラム例 2.7 の実行結果である．

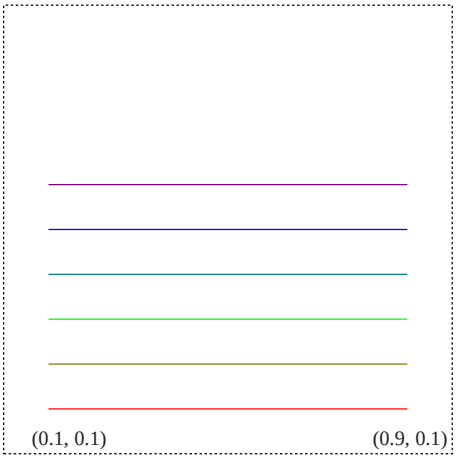

図 2.8 rgb 法による色の指定.

線分を描くとき，その端点の形を指定することができる．次のプログラム例 2.8 (`lines7.c`) では３つの異なる端点の形を描いている．

【プログラム例 2.8】

```
 1:#include "pssub.h"
 2:void main()
 3:{
 4:  init();
 5:  viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:  linety(1); linewidth(6.0);
 7:  linecap(0); newpath(); plot(0.1, 0.1, 3); plot(0.9, 0.1, 2);  stroke();
10:  linecap(1); newpath(); plot(0.1, 0.3, 3); plot(0.9, 0.3, 2);  stroke();
11:  linecap(2); newpath(); plot(0.1, 0.5, 3); plot(0.9, 0.5, 2);  stroke();
12:  fin();
13: }
```

プログラム例 2.8(lines7.c) は，linecap(ik) によって端点の形を指定している．このプログラムの実行結果は図 2.9 のようになる．プログラム例 2.8 における第 7 行目の linecap(0) (ik=0) は端点を矩形とする指定であり，図 2.9 で一番下の線のように plot 文で指定した座標が端点となる．線分の端を丸くしたいときは，linecap(1) (ik=1) と指定する．このとき，線分の端点は plot 文で指定した座標よりも線分の半値幅だけ外側になる．また，linecap(2) (ik=2) と指定すると，線分の端の形は矩形であるが，ik=1 の場合と同じように線分の端点は plot 文で指定した座標よりも線分の半値幅だけ外側になる．

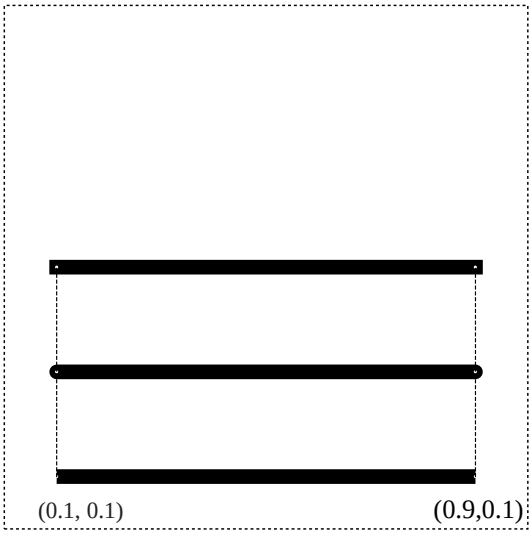

図 2.9 線の端点の形.

折れ線を描くときには，角の形を変えることができる．次の例では 4 つの異なる折れ線の角を描いている．

【プログラム例 2.9】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    init();
 5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0); linety(1);
 6:    linety(1); linewidth(15.0);
 7:    newpath(); plot(0.4, 0.1, 3); plot(0.7, 0.1, 2);
 8:               plot(0.7, 0.1, 3); plot(0.9, 0.4, 2); stroke();
 9:    newpath(); setlinejoin(0);
10:               plot(0.3, 0.3, 3); plot(0.6, 0.3, 2); plot(0.8, 0.6, 2);
11:    stroke();
12:    newpath(); setlinejoin(1);
13:               plot(0.2, 0.5, 3); plot(0.5, 0.5, 2); plot(0.7, 0.8, 2);
14:    stroke();
15:    newpath(); setlinejoin(2);
16:               plot(0.1, 0.7, 3); plot(0.4, 0.7, 2); plot(0.6, 1.0, 2);
17:    stroke();
18:    fin();
19: }
```

プログラム例 2.9(brokenlines1.c) では，４つの折れ線を描いている．これらの折れ線の違いは角部分である．１番下の折れ線は，座標 (0.4,0.1) から (0.7,0.1) までの線分と (0.7,0.1) から (0.9,0.4) までの線分を別々に描いているので，その接続部の角は少し隙間が空いてしまっている．下から２番目の折れ線は２つの線分を続けて描くのでそのような隙間は生じない．ただし，この折れ線を描く前に，setlinejoin(0) としており，この命令は折れ線の角を１点が頂点となるように尖った角 (Mitter join) となるように指定する命令である．同様に setlinejoin(1) は角を丸く (Round join) する命令，setlinejoin(2) は角を２つの頂点をもつように先端を切り取った角 (Bevel join) となるように指定する命令である．プログラムの実行結果 2.10 で，１番上の折れ線の角を見ても小さくて明瞭には見えないが，角に２つの頂点をもっている (図 1.3.3 を参照).

折れ線を描く関数は (brokenlines) であり，折れ線の始点と中間点および終点を指定する配列とそれらの数を引数として呼び出す．次の例は 2 つの中間点をもつ折れ線を描いている．

【プログラム例 2.10】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    int n=4;
 5:    double x[4],y[4];
 6:    x[0]=0.1;y[0]=0.1;x[1]=0.4;y[1]=0.1;x[2]=0.8;y[2]=0.4;x[3]=0.5;y[3]=0.6;
```

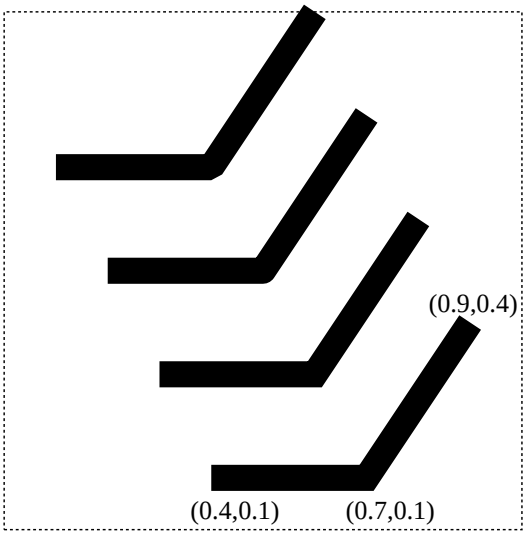

図 2.10 折れ線の角.

```
 7:    init();
 8:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 9:    linety(1); linewidth(2.0);
10:    setlinejoin(0); brokenlines(x,y,n);
11:    stroke();
12:    fin();
13:  }
```

プログラム例2.10(brokenlines2.c) では，実数型の配列 x[4] と y[4] に 4 つの点の座標を指定し，それらの点を折れ線でつないでいる (図2.11(a)). この例では角が尖るように，setlinejoin(0) としているが，この関数の引数を変えるとプログラム例 2.4 のように角の形が変わる．また，プログラム例 2.10 の 10 行目の後ろに closepath(); を付け加えると，図 2.11(b) のように閉じた折れ線，すなわち多角形となる.

2.4.2　曲線の描画法

ポストスクリプトにより曲線を描くときには，円弧を使用する方法やベジェ曲線 (Bézier Curve) を使う方法あるいはラグランジュ補間を用いる方法がある．まず最初に，円弧を使って滑らかな曲線を描く方法を説明する．３つの点を折れ線でつなぐのではなく，折れ曲がり部を円弧で滑らかにつなぐときには (arcto) という関数を用いる．この関数は，２点と円弧の半径を指定すると，現在ポインタが指し示している点からそれらの２点まで折れ線を描く代わりに，その折れ曲がり部分を，引数で指定した半径の円弧で滑らかにつないだ線を描く．次の例 2.11 は (arcto) を用いて円弧でつながった２本の直線を描く例である.

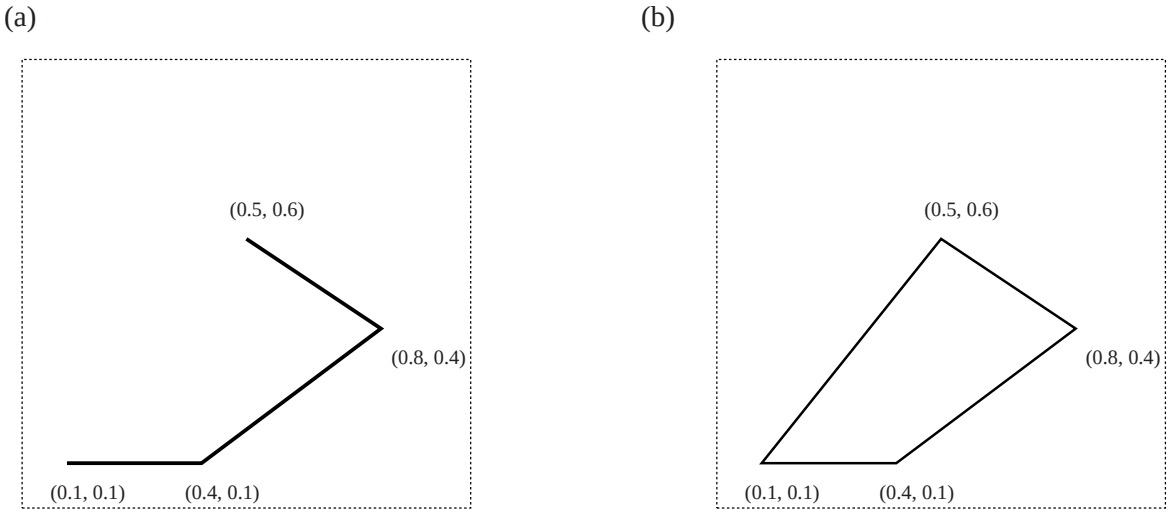

図 2.11 折れ線.

【プログラム例 2.11】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    init();
 5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:    linety(1); linewidth(2.0);
 7:    newpath(); plot(0.1, 0.1, 3); arcto(0.9, 0.1, 0.9, 0.5,0.2);
 8:    plot(0.9, 0.5, 2); stroke();
 9:    fin();
10:   }
```

プログラム例 2.11(arclines1.c) では，実数型の２つの点の座標（x1, y1），（x2, y2）と半径 r1 を指定し，現在ポインタが指し示す点から（x2, y2）までを滑らかな曲線でつないでいる．図 2.12 で実線と点線で示されているように，現在のポインタの位置と（x1, y1）および（x2, y2）を結ぶ折れ線の折れ曲がり部が半径 r1 の円弧で置き換えた図が描かれる．

２点をもっと滑らかな曲線でつなぐ方法は数多くあるが，ベジエ曲線でつなぐ方法について説明しよう．ここでは，３次のベジエ曲線を用いることにすると，あと２点の制御点が必要となる．始点は現在のポインタの位置であり，２点の制御点と終点を指定すると，始点と終点を結ぶ滑らかな曲線を描くことができる．このとき，一般に曲線は制御点を通らない．次の例 2.12 は関数 (curvto) により３次のベジエ曲線を描いている．

【プログラム例 2.12】

```
 1: #include "pssub.h"
```

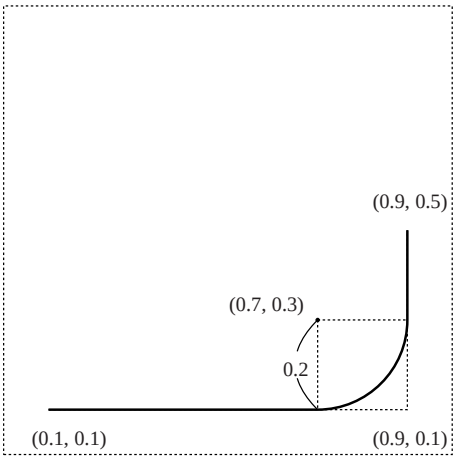

図 2.12 円弧でつなぐ曲線.

```
 2: void main()
 3: {
 4:    init();
 5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:    linety(1); linewidth(2.0);
 7:    newpath(); plot(0.1, 0.1, 3); curvto(0.5,0.1,0.5,0.4,0.9,0.4);
 8:    stroke();
 9:    fin();
10:   }
```

プログラム例 2.12(beziercurves1.c) では，実数型の 2 つの制御点の座標（x1, y1），（x2, y2）と終点の座標（x3, y3）を指定し，現在ポインタが指し示す点から（x3, y3）までを 3 次のベジエ曲線でつないでいる．図 2.13 で実線で描かれている曲線がベジエ曲線である．この曲線は始点と 1 つ目の制御点を結ぶ線分に接しており，同様に 2 つ目の制御点と終点を結ぶ線分にも接している．この関数を用いて自由に図を描くためにはいくつかの図を描いて制御点の配置とできる図の関係を知っておく必要がある．

ベジエ曲線による図形の描画は制御点の選び方が難しく，練習が必要である．3 次のベジエ曲線を用いる例をもう一つ考えておこう．次の例 2.13 は関数 (curvton) により 3 次のベジエ曲線を描いているがこの関数は (n) をパラメターとしてもつので，次数 (n) を指定することができる．

【プログラム例 2.13】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    double x1[100],y1[100];
```

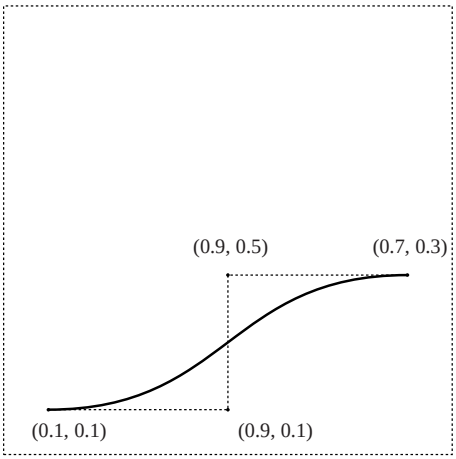

図 2.13 ベジエ曲線.

```
 5:    init();
 6:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 7:    linety(1); linewidth(2.0);
 8:    x1[0]=0.50; y1[0]=0.00; x1[1]=0.00; y1[1]=0.50; x1[2]=0.00; y1[2]=1.00;
 9:    x1[3]=0.50; y1[3]=0.70;
10:    newpath(); plot(x1[0], y1[0], 3);
11:    curvton(x1,y1,4);
12:    stroke();
13:    fin();
14:   }
```

プログラム例 2.13(beziercurves2.c) では，実数型配列で始点（x1[0], y1[0]）と終点（x1[3], y1[3]）を指定し，その中間に 2 つの制御点（x1[1], y1[1]）と（x1[2], y1[2]）を指定している．描く図形は縦の中心線に対して対称図形である．図 2.14 では制御点が少し大きな点で示されている．実線で描かれている曲線を見て，ベジエ曲線の描かれ方を考えてみよう．

ベジエ曲線による図形の描画で，制御点を結ぶ線分の中央部に重なるように描く方法もある．次の例 2.14 は関数 (curvtona) により 3 次のベジエ曲線を描いている．この関数も (n) をパラメタとしてもつので，次数 (n) を指定することができる．また，パラメータ (t) は $0.5 < t < 1$ の値をもち，その値が 1 に近づくほど，各制御点により近づく曲線となる．

【プログラム例 2.14】

```
1: #include "pssub.h"
2: void main()
3: {
4:    double x1[100],y1[100];
```

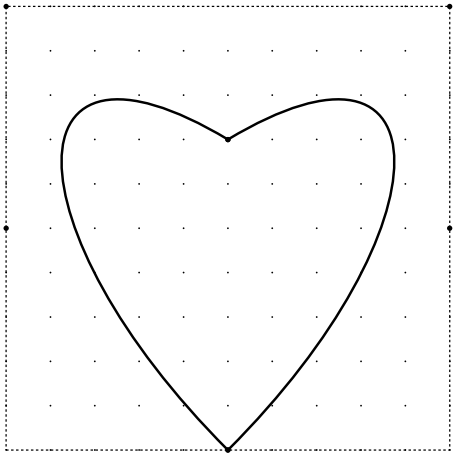

図 2.14 ベジエ曲線.

```
 5:    init();
 6:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 7:    linety(1); linewidth(2.0);
 8:    x1[0]=0.50; y1[0]=0.00; x1[1]=0.00; y1[1]=0.50; x1[2]=0.00; y1[2]=1.00;
 9:    x1[3]=0.50; y1[3]=0.70;
10:    newpath(); plot(x1[0], y1[0], 3);
11:    curvtona(x1,y1,0.7,4);
12:    stroke();
13:    fin();
14:   }
```

プログラム例**??**(bezierc urves3.c) もプログラム例 2.14 と同様に, 実数型配列で始点（x1[0], y1[0]）と終点（x1[3], y1[3]）を指定し，その中間に 2 つの制御点（x1[1], y1[1]）と（x1[2], y1[2]）を指定している．第 11 行目の関数 curvtona(x1,y1,0.7,4）では，$t = 0.7$ としているが，この数字が 1 に近いほど角張って，制御点に近づく曲線となる．図 2.15 でも制御点が少し大きな点で示されている．

４点を制御点にもつベジエ曲線は始点と終点を滑らかな曲線でつなぐが，一般には中間にある２点を通らない曲線となる．与えられたいくつかの点を通る曲線を描く代表的な方法はラグランジュ補間の方法法とスプライン補間の方法がある．次の例 2.15 では，ラグランジュ補間の方法を用いて４点を滑らかな曲線で結ぶ関数 (lgcurves) の使い方を説明する．

【プログラム例 2.15】

```
1: #include "pssub.h"
2: void main()
3: {
```

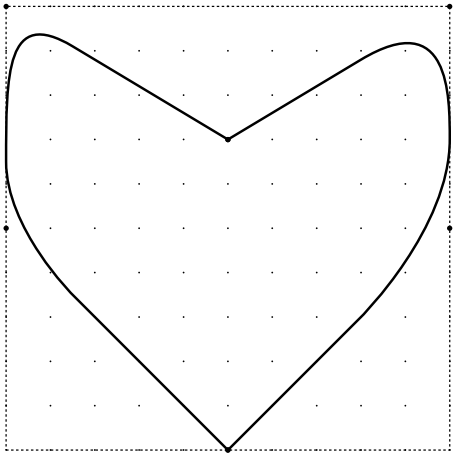

図 2.15 ベジエ曲線.

```
 4:    int n=4;
 5:    double x[4]={0.1, 0.5, 0.5, 0.9},y[4]={0.1, 0.1, 0.4, 0.4};
 6:    init();
 7:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 8:    linety(1); linewidth(2.0);
 9:    newpath(); lgcurves(x, y, n);
10:    stroke();
11:    fin();
12:   }
```

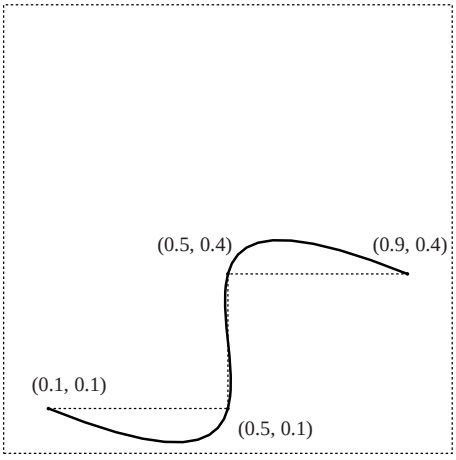

図 2.16 ラグランジュ補間で描く曲線.

プログラム例 2.15(`lgcurves1.c`) では，２つの実数型配列 `x[4]` と `y[4]` を定義し，それぞ

れが 4 つの点の x 座標と y 座標を表している．ここでは，4 点を通る曲線を描くために，$n = 4$ とし，その配列の大きさも 4 としているが，点の数 n は 3 以上の整数であれば良い．ただし，あまり大きな数を用いると曲線が複雑に曲がり，思った結果が得られないことが多く，$n = 3$ または 4 あるいは 5 程度とする方が望ましい結果を得ることができる．図 2.16 で実線で描かれているように，与えた 4 点を滑らかに結んでいることが分かる．ベジエ曲線とは異なり，始点と終点でそれぞれの前後の点を結ぶ線分には接していない．

2.5　基本図形 – 円・多角形・矩形・記号 –

円を描く関数は (circ) であり，引数として，中心の座標 ((x,y)) と半径 (r) を与える．

【プログラム例 2.16】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    init();
 5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:    linety(1); linewidth(1.0);
 7:    newpath(); circ(0.5, 0.5, 0.4);  stroke();
 8:    setgray(0.5); newpath(); circ(0.5, 0.7, 0.05); fill();
 9:    fin();
11:   }
```

プログラム例2.16(circs1.c) において，第7行目で circ(0.5,0.5,0.4) により，関数 circ を呼び出して (0.5,0.5) を中心とする半径 0.4 の円を描いている．また，8行目では (0.5,0.7) を中心とする半径 0.2 の円の内部を灰色 (濃さ 0.5) で塗りつぶしている．円に限らず，図形内部を塗りつぶすとその内部に描かれている図形はすべて塗りつぶされて見えなくなることに注意する必要がある．

円弧を描くときには，関数 arc を用いる．この関数の引数には中心の座標 (x,y) と半径 r の他に描画開始角度 t1 終点角度 t2 を与える．

【プログラム例 2.17】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    init();
 5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:    linewidth(2.0);
 7:    linety(3); newpath(); arc(0.5, 0.5, 0.2, 45.0, 135.0); stroke();
```

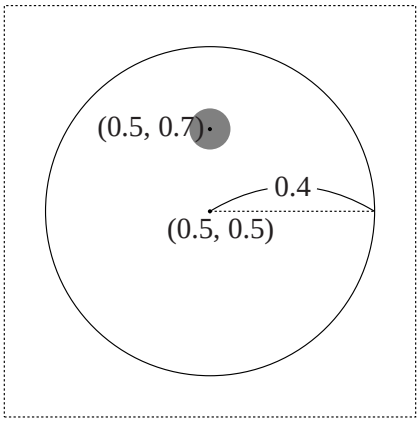

図 2.17 円.

```
 8:    linety(1); newpath(); arc(0.5, 0.5, 0.4, 0.0,  135.0); stroke();
 9:    fin();
11:   }
```

プログラム例2.17(arcs1.c)では，第7行目で破線を指定した後，関数arc(0.5,0.5,0.2,45.0,135.0)により，(0.5,0.5)を中心として半径 0.2 で，始点角度 45° から終点角度 135° まで，頂角が 90° となるように円弧を描いている．また，8行目では実線で，(0.5,0.5)を中心とする半径 0.4 の円弧を 0° から 135° まで描いている．

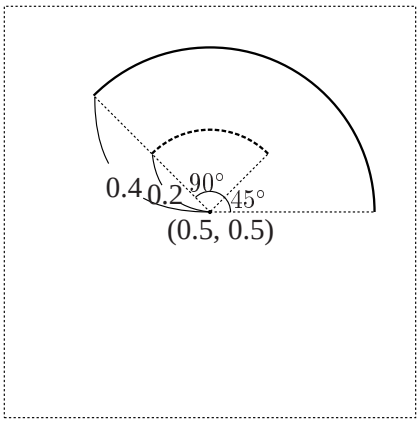

図 2.18 円弧.

楕円を描く関数は ellipse であり，中心の座標 (x,y) と，x 方向の軸長 r1 と y 方向の軸長 r2 をそれぞれ指定する．また，回転した座標系で楕円を描くときには関数 ellipse を使用する．このときには，中心の座標 (x,y) と 2 つの軸長 r1 と r2 の他に，回転角度 t を指定する．次のプログラム例でこれらの楕円の描き方を説明する．

【プログラム例 2.18】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    init();
 5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:    linety(1); linewidth(2.0);
 7:   linety(1); newpath(); ellipse(0.5, 0.5, 0.4, 0.2); stroke();
 8:    linety(3); newpath(); ellipse(0.5, 0.5, 0.1, 0.2); stroke();
 9:    fin();
10:  }
```

プログラム例 2.18(ellipse1.c) では，第 6 行目で中心が (0.5,0.5)，x 方向軸長が 0.4，y 方向軸長が 0.2 の楕円を描いている．このプログラムで描かれる図形は図 2.19(a) のようになる．

また，紙下端の線から $t°$ 回転した楕円を描く関数は (ellipserot) であり，プログラム例 2.19(ellipserot1.c) のように，紙下端の線から反時計回りに $t°$ 回転した座標系で矩形の左下と右上を指定する．

【プログラム例 2.19】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    init();
 5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:    linety(1); linewidth(2.0);
 7:    linety(1); newpath(); ellipserot(0.6,0.3,0.4,0.2,30.0); stroke();
 8:    linety(3); newpath(); ellipserot(0.6,0.3,0.1,0.2,30.0); stroke();
 9:    fin();
10:  }
```

プログラム例 2.19(ellipserot1.c) では，反時計回りに 30° 回転した座標系で中心が (0.6,0.3)，x 方向軸長が 0.4，y 方向軸長が 0.2 の楕円を描いている．このプログラムの出力は図 2.19(b) のようになる．

正方形は円と同様に基本的な図形の構成要素として使ったり，グラフの中で記号のように使ったりする．正方形を描く関数は (square) である．関数 (square) では，正方形の中心座標 (x,y) と一辺の長さ ell を指定する．また，回転した座標系で正方形を描く関数は (squarerot) であり，この関数では，中心座標 (x,y) と一辺の長さ ell の他に回転角度 t を指定する．次のプログラムはこれらの関数を使用する例である．

【プログラム例 2.20】

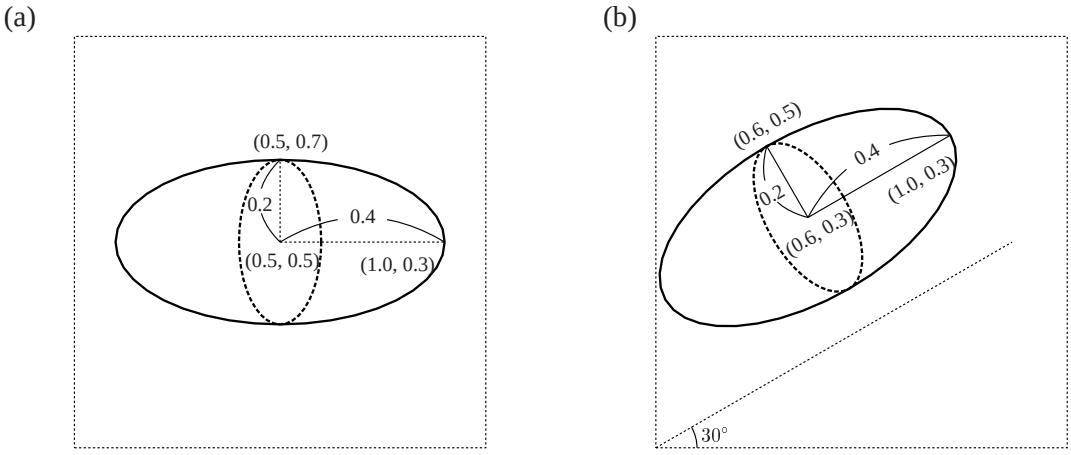

図 2.19 楕円.

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    init();
 5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:    linety(1); linewidth(2.0);
 7:    newpath(); square(0.2, 0.1, 0.2); stroke();
 8:    newpath(); squarerot(0.6, 0.1, 0.2, 20.0); stroke();
 9:    newpath(); setrgb(1.0,0.0,0.0);squarerot(0.9, 0.1, 0.2, 40.0);
10:               fill();stroke();
11:    newpath(); setrgb(0.0,0.0,1.0);squarerot(1.0, 0.1, 0.2, 60.0);
12:               fill();stroke();
13:    fin();
14:   }
```

プログラム例 2.20(squares1.c) では，第 7 行目で（0.2,0.1）を中心とした一辺の長さが 0.2 の正方形を描いている．第 8 行目で描くのは（0.6,0.1）を中心とした一辺の長さが 0.2 の正方形を角度 20° 回転した正方形である．同様に，第 9 行目から第 12 行目においては，それぞれ中心が（0.9,0.1）と（1.0,0.1）にあり，一辺の長さが 0.2 である正方形が 40° と 60° 回転した図形を描き，赤と青で塗りつぶしている．このプログラムで描かれる図形は図 2.20 のようになる．

三角形や四角形だけでなく，正 n 角形を描く関数は (polygon) である．関数 (polygon) では，正方形の中心座標（x,y）と中心から頂角までの長さ (r) および角の数 n を指定する．この関数では，中心座標の正 n 角形の頂点は中心座標の真上に位置するように図形が描かれる．回転した座標系で正 n 角形を描く関数は (polygonrot) であり，この関数では，中心座標（x,y）と 中心から頂角までの長さ (r) と 回転角度 t および角の数 n を指定する．次のプログラムはこれらの関数を使用する例である．

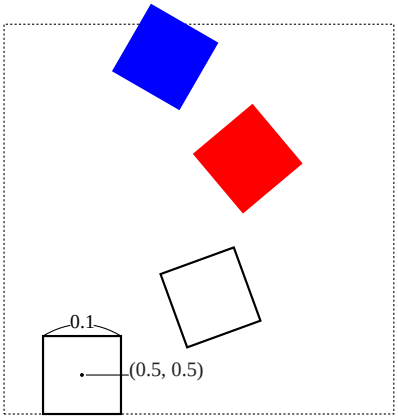

図 2.20 正方形.

【プログラム例 2.21】

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    init();
 5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:    linety(1); linewidth(2.0);
 7:    newpath(); polygon(0.3,0.3,0.1,3);stroke();
 8:    linety(3); newpath(); polygon(0.7,0.3,0.1,4);stroke();
 9:    setrgb(1.0,0.0,0.0);newpath();polygon(0.7,0.7,0.1,5);fill();
10:    setrgb(0.0,0.0,1.0);newpath();polygon(0.3,0.7,0.1,6);fill();
11:    fin();
12:   }
```

プログラム例 2.21(polygons1.c) は，第 7 行目で（0.3,0.3）を中心として，中心から頂点までの長さが 0.1 の正三角形を描いている．第 8 行目は（0.7,0.3）を中心とし，中心から頂点までの長さが 0.1 （0.7,0.3）の正四角形を描くプログラムである．同様に，第 9 行目と第 10 行目で，五角形と六角形を描き，それぞれ赤と青で内部を塗りつぶしている．このプログラムで描かれる図形は図 2.21 のようになる．

n 個の頂点をもつ星形図形を描くときは，関数 star あるいは starx を用いる．星形図形の中心座標（x,y）と中心から頂角までの長さ (r) および角の数 n を指定する．この関数では，正 n 角形の頂点は中心座標の真上に位置するように図形が描かれる．回転した座標系で正 n 角形を描く関数は (polygonrot) であり，この関数では，中心座標（x,y）と 中心から頂角までの長さ (r) と 回転角度 t および角の数 n を指定する．次のプログラムはこれらの関数を使用する例である．

【プログラム例 2.22】

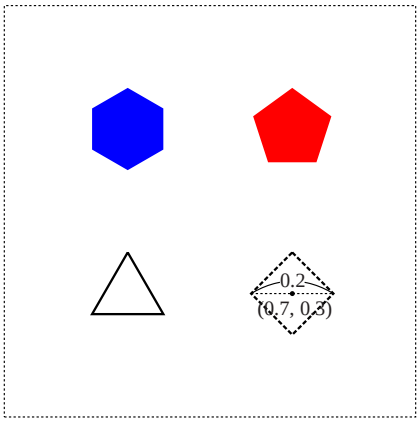

図 2.21 正 n 角形.

```
 1: #include "pssub.h"
 2: void main()
 3: {
 4:    init();
 5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
 6:    linety(1); linewidth(2.0);
 7:    newpath(); star(0.25, 0.25, 0.2, 0.0,3); stroke();
 8:    newpath(); star(0.75, 0.25, 0.2, 0.0,4); stroke();
 9:    newpath(); star(0.25, 0.75, 0.2, 0.0,5); stroke();
10:    newpath(); star(0.75, 0.75, 0.2, 0.0,6); stroke();
11:    fin();
12:   }
```

プログラム例 2.22(stars1.c) の第 7 行目では，(0.25,0.25) を中心として，中心から頂点までの長さが 0.2 である 3 つの頂角をもつ星形図形を描いている．4 つめの引数は中心と頂点の一つを結ぶ線が y 軸となす角度であり，第 7 行目では 0.0 を指定しているので，中心の真上に頂角がある．第 8 行目から第 10 行目までで，4 つの頂点，5 つの頂点，6 つの頂点をもつ星形図形を描いている．このプログラムで描かれる図形は図 2.22(a) と 2.22(b) のようになる．ただし，図 2.22(b) では，関数 star の代わりに starx を用いており，描いている星形図形は，5 個から 8 個の頂点をもっている．

矩形はグラフの外枠などによく用いる．矩形を描く関数は (rect) である．関数 (rect) では，矩形の左下と右上の ((x,y)) 座標をそれぞれ指定する．また，回転した座標系で矩形を描く方が便利なこともある．そのときは関数 (rectrot) を使用する．これら 2 つの関数の使用例を説明する．

【プログラム例 2.23】

(a) (b)

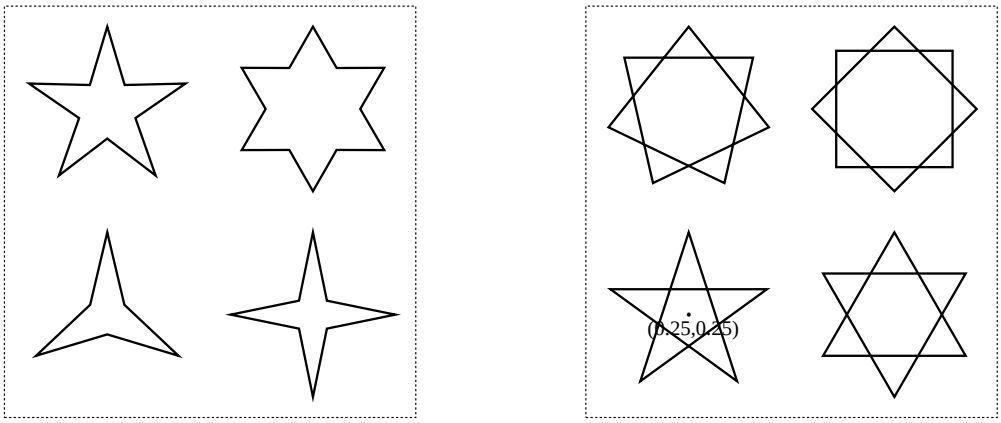

図 2.22 星形図形.

```
1: #include "pssub.h"
2: void main()
3: {
4:    init();
5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
6:    linety(1); linewidth(2.0);
7:    newpath(); rect(0.2, 0.2, 0.8, 0.8); stroke();
8:    fin();
9:   }
```

プログラム例 2.23(rect1.c) では，第 7 行目で左下が（0.2,0.2）右上が（0.8,0.8）の矩形を描いている．このプログラムで描かれる図形は図 2.23(a) のようになる．

また，紙下端の線から $t°$ 回転した矩形を描く関数は (rectrot) であり，プログラム例 2.24(rectrot.c) のように，紙下端の線から反時計回りに $t°$ 回転した座標系で矩形の左下と右上を指定する．

【プログラム例 2.24】

```
1: #include "pssub.h"
2: void main()
3: {
4:    init();
5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
6:    linety(1); linewidth(2.0);
7:    newpath(); rectrot(0.3, 0.0, 0.8, 0.5); stroke();
8:    fin();
9:   }
```

プログラム例 2.24(rectrot1.c) では，反時計回りに 30° 回転した座標系で矩形の左下が（0.3,0.0）右上が（0.8,0.5）の矩形を描いている．このプログラムの出力は図 2.23(b) のようになる．

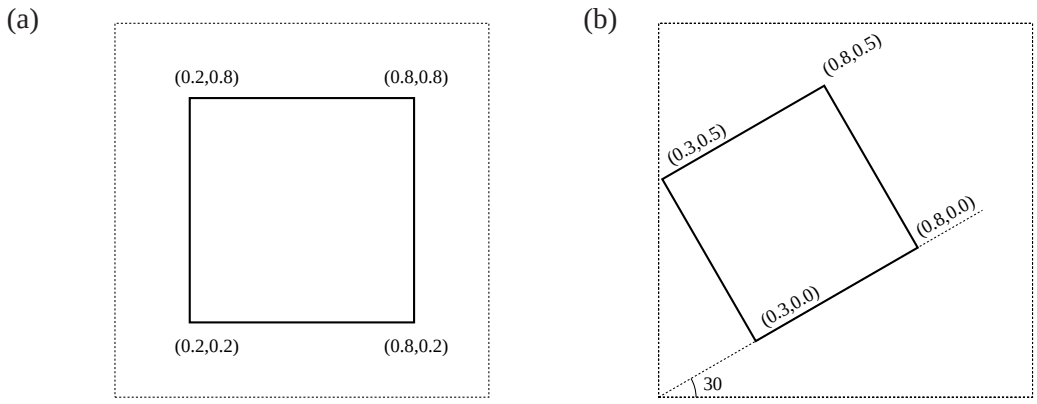

図 2.23 矩形.

角の丸い矩形を描くこともできる．角の丸い矩形を描く関数は (rectround) である．関数 (rectround) では，矩形の左下と右上の ((x,y)) 座標だけでなく，角の丸さを表す半径 (r) も指定する．また，回転した座標系で角の丸い矩形を描くときには，関数 (rectroundrot) を使用する．これら 2 つの関数の使い方をプログラム例 2.25(rectround1.c) と例 2.26(rectroundrot1.c) で紹介する．

【プログラム例 2.25】

```
1: #include "pssub.h"
2: void main()
3: {
4:    init();
5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
6:    linety(1); linewidth(2.0);
7:    newpath(); rectround(0.2, 0.2, 0.8, 0.8, 0.04); stroke();
8:    fin();
9:   }
```

【プログラム例 2.26】

```
1: #include "pssub.h"
2: void main()
3: {
4:    init();
5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
6:    linety(1); linewidth(2.0);
7:    newpath(); rectroundrot(0.3, 0.0, 0.8, 0.5, 0.04, 30.0); stroke();
8:    fin();
9:   }
```

プログラム例 2.25(rectround1.c) では，第 7 行目で左下が (0.2,0.2) 右上が (0.8,0.8) の

矩形で，角を半径 0.04 の円で丸くした曲線を描いている．このプログラムで描かれる図形は図 2.24(a) のようになる．

また，紙下端の線から $t°$ 回転した角の丸い矩形を描く関数は (rectroundrot) であり，プログラム例 2.25 の第 7 行目の (rectround(0.2, 0.2, 0.8, 0.8, 0.04);) の代わりに，(rectroundrot(0.3,0.0,0.8,0.5,0.04,30.0);) と記述すると，紙下端の線から反時計回りに $t°$ 回転した座標系で角の丸い矩形を描くことができ，そのときの出力は図 2.24(b) のようになる．

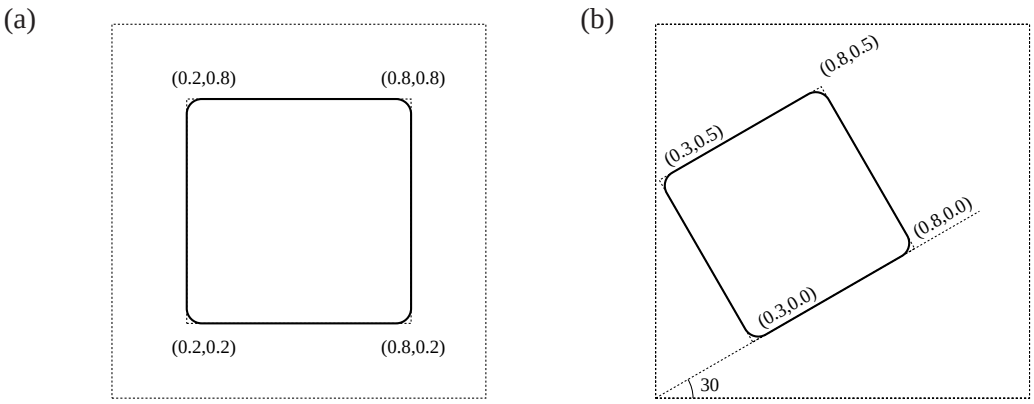

図 2.24 角の丸い矩形.

関数のグラフや実験データのグラフを描くときには，座標軸として矢印を用いる．矢印は手続きの流れ，論理の流れを示すためのプレゼンテーション用としても広く使用される．そのために，いろいろな矢尻の形をもつ矢印や幅の広い矢印などを使い分ける必要がある．矢印を描く関数には，(arrow)，(arrowa)，(arrowb) (arrowc) などがある．これらの関数では矢印の始点 ((x1,y1)) と終点 (矢尻) ((x2,y2)) および矢尻の大きさ (d) を指定する．幅の広い矢印を描く関数は (arrowwide) で，矢印の始点，終点，矢尻の大きさの他に，矢の太さ表す半径 (al) も指定する．また，回転した座標系で矢を描くときには，関数 (arrowrot)，(arrowarot)，(arrowbrot)，(arrowcrot) (arrowwiderot) などを使用する．このときには，座標の回転角 (t1) も指定する必要がある．矢印を描く関数の使い方をプログラム例 2.27(arrow1.c) で紹介する．

【プログラム例 2.27】

```
1: #include "pssub.h"
2: void main()
3: {
4:    init();
5:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0,0.0,1.0,1.0);
6:    linety(1); linewidth(2.0);
7:    newpath(); arrow(0.1, 0.1, 0.9, 0.1, 0.025); stroke();
8:    newpath(); arrowa(0.1, 0.2, 0.9, 0.2, 0.025); stroke();
9:    newpath(); arrowb(0.1, 0.3, 0.9, 0.3, 0.025); stroke();
```

```
10:     newpath(); arrowc(0.1, 0.4, 0.9, 0.4, 0.025); stroke();
11:     newpath(); arrowwide(0.1, 0.5, 0.9, 0.5, 0.05, 0.01); stroke();
12:     fin();
13:    }
```

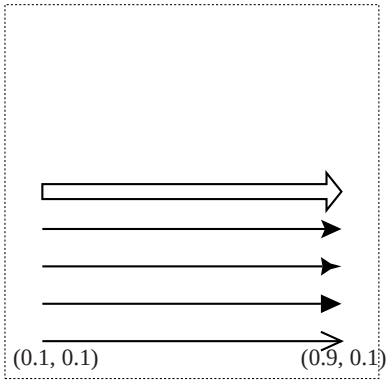

図 2.25 矢印.

プログラム例 2.27(arrow1.c) で描かれる矢印は図 2.25 のようになる．下から４つの矢印はそれぞれ矢尻の形が異なっている．一番上の矢印は幅の広い矢印であり，プレゼンテーションにおいて論理の推移やものごとの変化の方向を表すときに使用する．

第3章　美しいグラフの描き方

ポストスクリプトによる描画が最も力を発揮するのがグラフ描画である．実験データや数値計算結果のデータなどがもつ意味を見つけ出すためにも，その意味を他人に伝えるためにも，データの整理の仕方とグラフの描き方は重要である．

最も一般的なグラフは横軸と縦軸を矢印で描き，２つの矢印が表す座標軸に刻まれたそれぞれの座標値を通る座標軸に平行な２つの線の交点にデータを対応させる．データの性質や種類によっては，それらのデータ点を滑らかな曲線で結びあるいは折れ線で結ぶこともある．たとえば，微分可能な数学関数などは滑らかな曲線で結ぶ．あるいは，測定誤差などを含むために分散する実験データなどは点で描き，それらが統計的にもつ性質を最小自乗法などで求めて，直線や曲線で表現する．まずは，横軸として x 軸と縦軸として y 軸を描いてみよう．

【プログラム例 3.1】

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3: #include <string.h>
 4: #include <math.h>
 5: #include "pssub.h"
 6:
 7: void xaxis(double,double,double,double,int,int);
 8:
 9: void main()
10: {
11:    init();
12:    viewport(0.2,0.2,0.8,0.8); xyworld(-1.2, -1.2, 1.2, 1.2);
13:    xaxis(-1.0,0.0,1.0,0.0,2,10);
14:    fin();
15: }
16:
17: void xaxis(double x1,double y1,double x2,double y2,int m,int n)
18: {
19:     int i;double s,d,x,dx,dy;
20:     setgray(0.0);
21:     s=(x2-x1)*(x2-x1)+(y2-y1)*(y2-y1);s=sqrt(s);
22:     d=s/(double)60;
23:     arrow(x1-s/20.0, y1, x2+s/20.0*2.0, y2, d);
24:     for (i=0;i<=m*n;i++){
25:        x=1.0/(double)(m*n)*(double)i;
```

```
26:         if(i/n*n==i) dy=0.02; else  dy=0.01;
27:         line( -x, -dy, -x, dy);
28:         line( x, -dy, x, dy);
29:         } stroke();
30:      textx(x2+s/10.0*2.0, -s/30.0,"x");
31:      textx((x1+x2)/2.0,(y1+y2)/2.0,"O");
32:      textx(x2/2.0,y2,"pi/2");
33:      textx(x2,y2,"pi");
34:      textx(x1/2.0,y1,"-pi/2");
35:      textx(x1,y1,"-pi");
36:      stroke();
37:      return;
38: }
```

プログラム例 3.1(axisx.c) では，第 7 行目で関数 xaxis のプロトタイプ宣言を行っている．関数 xaxis は，２点（x1,y1）と（x2,y2）と整数 m と n を引数にもつ．第 23 行目で矢印を描く．この矢印は，引数で与える 2 点間の距離を s とすると，矢の根本がが $1/10$ だけ左にあり，矢の先端は $2/10$ だけ右にくるように描く．また，矢印で表す座標軸に $m * n$ 個の刻み線を入れる．このとき，n 個ごとに刻み線の長さを長くする．第 30 行目からは軸の名前と座標の値の数値を表示する．こうして描かれる図形は図 3.1(a) のようになる．同様にして，y 軸を描くと図 3.1(b) のように描くことができる．

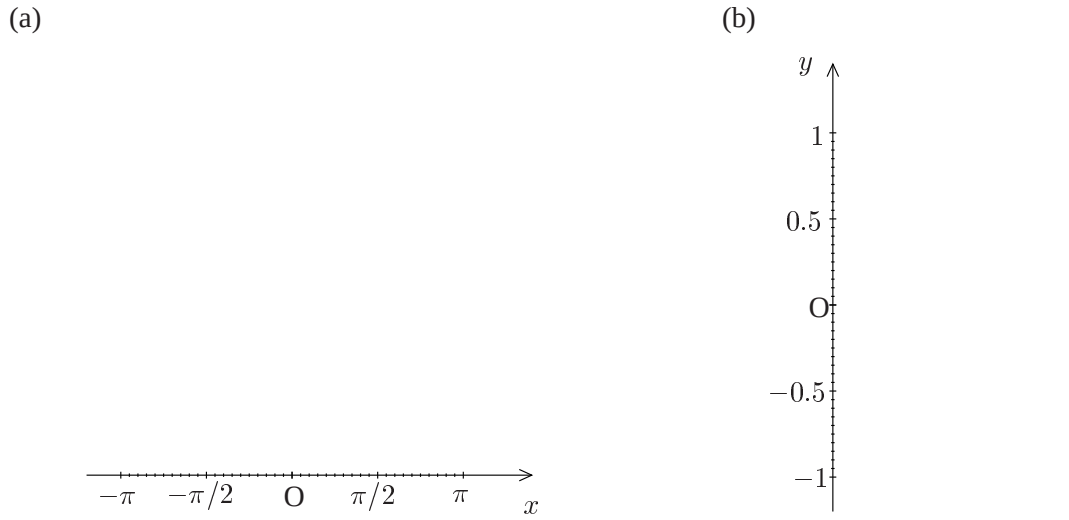

図 3.1 x 座標軸と y 座標軸.

簡単なグラフの例として，関数 $y = \sin x$ を $-pi \leq x \leq \pi$ の範囲で描くプログラムを考える．座標軸 (x 軸と y 軸) を描くときは，プログラム例 3.1 で作成した関数 xaxis とほぼ同様にして作成する関数 yaxis を用いる．また，関数 $y = \sin x$ を描く関数を drawline とする．

【プログラム例 3.2】

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3: #include <string.h>
 4: #include <math.h>
 5: #include "pssub.h"
 6:
 7: void frame(void);
 8: void xaxis(double,double,double,double,int,int);
 9: void yaxis(double,double,double,double,int,int);
10: void drawline(void);
11:
12: void main()
13: {
14:    init();
15:    viewport(0.2,0.2,0.8,0.8); xyworld(-1.2, -1.2, 1.2, 1.2);
16:    frame();
17:    drawline();
18:    fin();
19: }
20:
21: void frame()
22: {
23:     xaxis(-1.0,0.0,1.0,0.0,2,10);
24:     yaxis(0.0,-1.0,0.0,1.0,2,10);
25:     return;
26: }
27:
28: void drawline(void)
29: {
30:     int i,nx=100;double x,y,t;
31:
32:      x=-1.0;  y=0.0;
33:      linewidth(1.5);
34:      setgray(0.0);
35:      plot( x, y, 3);
36:        for (i=-nx+1;i<=nx;i++){
37:          t = (double)i/(double)nx;x = t; y = sin(t*3.14159);
38:          plot( x, y, 2);
39:         } stroke();
40:    return;
41: }
```

プログラム例 3.2 をコンパイル・実行すると，図 3.2 のグラフが得られる．

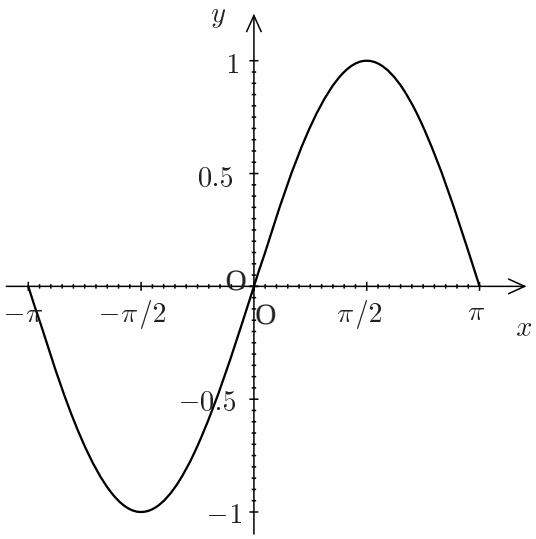

図 3.2 $x = [-\pi, \pi]$ における $y = \sin x$ のグラフ.

図 3.2 のように，横軸と縦軸座標を矢印で表すのがグラフ描画の基本であるが，四角形の枠内にグラフを描く方が印象的な場合もある．特に，グラフの数値を目視で読み取ろうとする場合には，四角形の中に格子状の網目を入れるとわかりやすい．そのようなグラフの例がプログラム例 3.3 である．

【プログラム例 3.3】

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3: #include <string.h>
 4: #include <math.h>
 5: #include "pssub.h"
 6:
 7: double f(double);
 8: void plotd(void);
 9: void frame(void);
10:
11: void main()
12: {
13:    init();
14:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0, -1.0, 1.0, 1.0);
15:    frame();
16:    plotd();
17:    fin();
18: }
19:
21: double f(double x)
22: {   double pi,f1;
```

```
23:     pi=atan(1.0)*4.0;
24:     f1=exp(-x)*sin(4.0*pi*x);
25:     return (f1);
26: }
27: void plotd()
28: {  double x1[21],y1[21];double x=1.0;
29:    int i,n=20;
30:     for (i=0;i<=n; i++) { x1[i]=(double)i/(double)n;
31:                           y1[i]=f(x1[i]); }
32:     linewidth(1.5);
33:     linety(1);
34:     setgray(0.0);
35:     plot(x1[0], y1[0], 3);
36:     for (i=1;i<=n-2;i++) {
37:      spline(x1[i-1],y1[i-1],x1[i],y1[i],x1[i+1],y1[i+1],x1[i+2],y1[i+2],-1);
38:      spline(x1[n-3],y1[n-3],x1[n-2],y1[n-2],x1[n-1],y1[n-1],x1[n],y1[n],0);
39:      spline(x1[n-3],y1[n-3],x1[n-2],y1[n-2],x1[n-1],y1[n-1],x1[n],y1[n],1);
40:      stroke();
41: }
42:
43:void frame()
44: {
45:     double x,y,dx,dy;
46:     int nx=2,ny=2,ix,iy;
47:
48:       linewidth(1.0);
49:       rect(0.0, -1.0, 1.0, 1.0); stroke();
50:       for(ix=1;ix<=nx*10-1;ix++) {
51:         x = (double)ix/((double) nx*10);
52:         dy=1.0;
53:         if(ix/10*10==ix) setgray(0.5); else setgray(0.8);
54:         plot( x, -dy, 3);
55:         plot( x, dy, 2);
56:         stroke();               }
57:      for(iy=1;iy<=ny*10-1;iy++) {
58:         y = (double) iy/((double) ny*10)*2.0-1.0;
59:         dx=1.0;
60:         if(iy/10*10==iy) setgray(0.5); else setgray(0.8);
61:         plot( 0.0, y, 3);
62:         plot( dx,  y, 2);
63:         stroke();               }
64:      setgray(0.0);
65:      textx(0.75, 0.0, "x");
```

```
66:       textx(0.05, 0.02, "0");
67:       textx(0.55, 0.02, "0.5");
68:       textx(0.95, 0.02, "1");
69:       texty(-0.01, 0.5, "y");
70:       texty(0.0, -1.0,  "-1");
71:       texty(0.0, 0.0,  "0");
72:       texty(0.0, 1.0,  "1");
73:     return;
74: }
```

プログラム例 3.3 をコンパイル・実行すると，図 3.3 のグラフが得られる．このように，ある程度定量的な情報を伝えるためには座標値を表す格子をグラフに描画しておく．

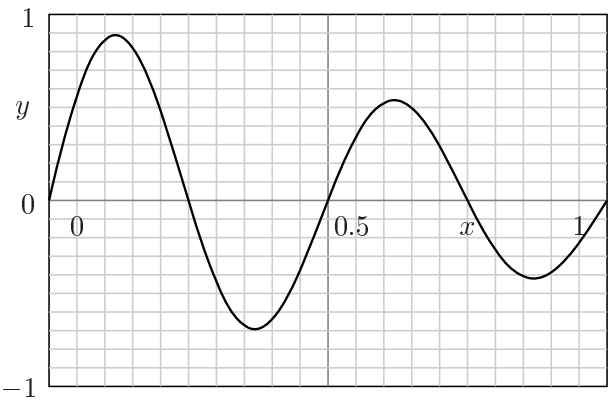

図 3.3 $x = [0, 1]$ における $y = e^{-x} \sin 4\pi x$ のグラフ．

図 3.2 や 3.3 は，横軸と縦軸座標を線形座標，すなわち x および y ととっているが，y が x のべき関数であることが期待できるときやべき関数であることを確かめるためには，横軸と縦軸に対数座標をとる．すなわち，

$$y = cx^a \tag{3.0.1}$$

の関係があれば，式 (3.0.1) の両辺の自然対数をとると，

$$\log_{10} y = a \log_{10} x + \log_{10} c \tag{3.0.2}$$

となるので，両対数で描いた $y(x)$ のグラフは傾きが a の直線となる．逆に，グラフの傾きが a となれば，$y(x)$ は式 (3.0.2) のように，x^a の関数形をもつことがわかる．そのようなグラフの例がプログラム例 3.4 である．

【プログラム例 3.4】

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <string.h>
```

```
 4: #include <math.h>
 5: #include "pssub.h"
 6:
 7: double f(double);
 8: void plotd(void);
 9: void frame(void);
10:
11: void main()
12: {
13:    init();
14:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0, 0.0, 4.0, 4.0);
15:    frame();
16:    plotd();
17:    fin();
18: }
19:
21: double f(double x)
22: {   double fx;
23:     fx=2.0*pow(x,2.0/3.0);
24:     return (fx);
25: }
26:
27: void plotd()
28: {  double x1[21],y1[21];int n,i;
29:    n=20;
30:     for (i=0;i<=n; i++) {
31:        x1[i]=1.0*pow(10.0,(double)i/(double)n *4.0);y1[i]=f(x1[i]); }
32:     for (i=0;i<=n; i++) {
33:        x1[i]=log10(x1[i]);y1[i]=log10(y1[i]);}
34:     linewidth(1.5);linety(1);setgray(0.0);
35:     plot(x1[0], y1[0], 3);
36:     for (i=1;i<=n-2;i++) {
37:      spline(x1[i-1],y1[i-1],x1[i],y1[i],x1[i+1],y1[i+1],x1[i+2],y1[i+2],-1);
38:      spline(x1[n-3],y1[n-3],x1[n-2],y1[n-2],x1[n-1],y1[n-1],x1[n],y1[n],0);
39:      spline(x1[n-3],y1[n-3],x1[n-2],y1[n-2],x1[n-1],y1[n-1],x1[n],y1[n],1);
40:      stroke();
41: }
42:
43:void frame()
44: {
45:     int ix,nx,iy,ny,ixx,iyy; double x,y,dx,dy;
46:        linewidth(1.0);
47:       rect(0.0, 0.0, 4.0, 4.0);stroke();
```

```
48:       nx=4;
49:       for(ix=0;ix<=nx-1;ix++) {
50:         setgray(0.3);
51:         x = (double)ix; dy=4.0;
52:         if(ix!=0) {plot(x,0.0,3);plot(x,dy,2);stroke(); }
53:         for(ixx=2;ixx<=9;ixx++) {
54:             setgray(0.8);x = (double) ix+log10((double) ixx);
55:             plot(x,0.0,3);plot(x,dy,2);stroke();}}
56:       ny=4;
57:       for(iy=0;iy<=ny-1;iy++) {
58:         setgray(0.5);
59:         y = (double)iy; dx=4.0;
60:         if(iy!=0) {plot(0.0,y,3);plot(dx,y,2);stroke(); }
61:         for(iyy=2;iyy<=9;iyy++) {
62:             setgray(0.8);y = (double) iy+log10((double) iyy);
63:             plot(0.0,y,3);plot(dx,y,2);stroke();}}
64:      setgray(0.0);
65:      textx(3.5, 0.0, "x");
66:        textx(0.0, 0.02, "1"); textx(1.0, 0.02, "10");
67:        textx(2.0, 0.02, "100"); textx(3.0, 0.02, "1000");
68:        textx(4.0, 0.02, "10000");
69:      texty(0.0, 3.5, "y");
70:        texty(0.0, 0.0, "1");  texty(0.0, 1.0, "10");
71:        texty(0.0, 2.0, "100");texty(0.0, 3.0, "1000");
72:        texty(0.0, 4.0, "10000");
73:     return;
74: }
```

プログラム例 3.4 の実行結果は図 3.5 のようになり，このグラフの傾きは 1.5 であり，$y \propto x^{1.5}$ であることがわかり，その係数は $x = 1$ における y 軸の切片から求めることができる．このグラフの場合は切片が $y = 2.0$ である．

プログラム例 3.4 では，y が x のべき関数であると考えて，このことを確かめるために横軸と縦軸に対数座標をとった．初等関数では，べき関数の他に指数関数や対数関数が考えられる．特に指数関数は理工学のみならず，経済学や医学などの分野でもよく現れる．y が x の指数関数で

$$y = c\exp(ax) \tag{3.0.3}$$

の場合を考える．式 (3.0.3) の両辺の自然対数をとると，

$$\log_{10} y = a\log_{10} e \times x + \log_{10} c \tag{3.0.4}$$

となり，横軸に線形座標 x をとり，縦軸に対数座標 $\log_{10} y$ をとれば，式 (3.0.4) のグラフは傾きが $a\log_{10} e$ の直線となる．縦軸に $\ln y$ をとれば，グラフの傾きが a となるが，$\log_{10} y$ ととるのは，

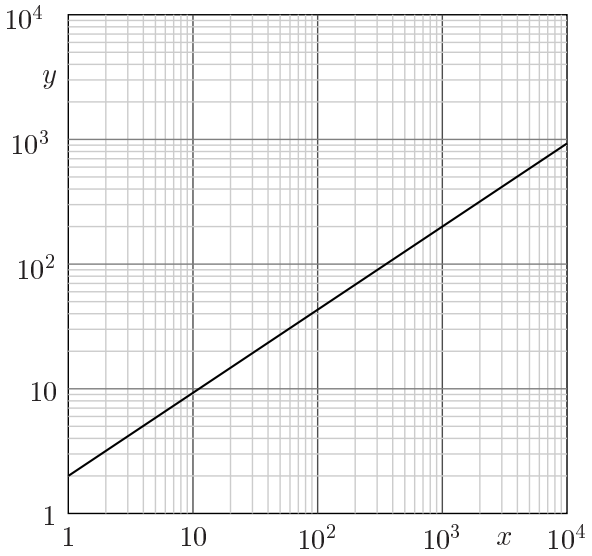

図 3.4 $\log_{10} x = [0, 4]$ ($x = [1, 10^4]$) における $y = 2.0x^{1.5}$ のグラフ.

片対数グラフを用いていたこれまでの習慣によるものであり，必要に応じて縦軸に $\ln y$ をとってもよい．また，$y(x)$ の関数形が $y = c\ln(ax)$ のように対数関数とよそうされるときは，横軸に対数座標 $\log_{10} x$ をとり，縦軸に線形座標 y をとる．プログラム例 3.5 は横軸に線形座標 x をとり，縦軸に対数座標 $\log_{10} y$ をとって，関数 $y = 2exp(4x)$ のグラフを描く例である．

【プログラム例 3.5】

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3: #include <string.h>
 4: #include <math.h>
 5: #include "pssub.h"
 6:
 7: double f(double);
 8: void plotd(void);
 9: void frame(void);
10:
11: void main()
12: {
13:    init();
14:    viewport(0.2,0.2,0.8,0.8); xyworld(0.0, 0.0, 1.0, 4.0);
15:    frame();
16:    plotd();
17:    fin();
18: }
19:
21: double f(double x)
22: {   double fx;
```

```
23:          fx=2.0*exp(4.0*x);
24:     return (fx);
25: }
26:
27: void plotd()
28: {  double x1[21],y1[21];int n,i;
29:    n=20;
30:     for (i=0;i<=n; i++) {
31:        x1[i]=(double)i/(double)n; y1[i]=f(x1[i]); }
32:     for (i=0;i<=n; i++) {
33:        y1[i]=log10(y1[i]);}
34:     linewidth(1.5);linety(1);setgray(0.0);
35:     plot(x1[0], y1[0], 3);
36:     for (i=1;i<=n-2;i++) {
37:      spline(x1[i-1],y1[i-1],x1[i],y1[i],x1[i+1],y1[i+1],x1[i+2],y1[i+2],-1);
38:      spline(x1[n-3],y1[n-3],x1[n-2],y1[n-2],x1[n-1],y1[n-1],x1[n],y1[n],0);
39:      spline(x1[n-3],y1[n-3],x1[n-2],y1[n-2],x1[n-1],y1[n-1],x1[n],y1[n],1);
40:      stroke();
41: }
42:
43:void frame()
44: {
45:     int ix,nx,iy,ny,ixx,iyy; double x,y,dx,dy;
46:        linewidth(1.0);
47:       rect(0.0, 0.0, 1.0, 4.0);stroke();
48:       nx=1;
49:       for(ix=0;ix<=nx-1;ix++) {
50:         setgray(0.5);
51:         x = (double)ix; dy=4.0;
52:         if(ix!=0) {plot(x,0.0,3);plot(x,dy,2);stroke(); }
53:         for(ixx=1;ixx<=9;ixx++) {
54:             setgray(0.8);x = (double) ixx/10.0;
55:             plot(x,0.0,3);plot(x,dy,2);stroke();}}
56:       ny=4;
57:       for(iy=0;iy<=ny-1;iy++) {
58:         setgray(0.5);
59:         y = (double)iy; dx=1.0;
60:         if(iy!=0) {plot(0.0,y,3);plot(dx,y,2);stroke(); }
61:         for(iyy=2;iyy<=9;iyy++) {
62:             setgray(0.8);y = (double) iy+log10((double) iyy);
63:             plot(0.0,y,3);plot(dx,y,2);stroke();}}
64:      setgray(0.0);
65:      textx(0.75, 0.0, "x");
```

```
66:         textx(0.0, 0.02, "0"); textx(0.5, 0.02, "0.5");
67:         textx(1.0, 0.02, "1");
68:       texty(0.0, 3.5, "y");
69:         texty(0.0, 0.0, "1");  texty(0.0, 1.0, "10");
70:         texty(0.0, 2.0, "100");texty(0.0, 3.0, "1000");
71:         texty(0.0, 4.0, "10000");
72:      return;
73: }
```

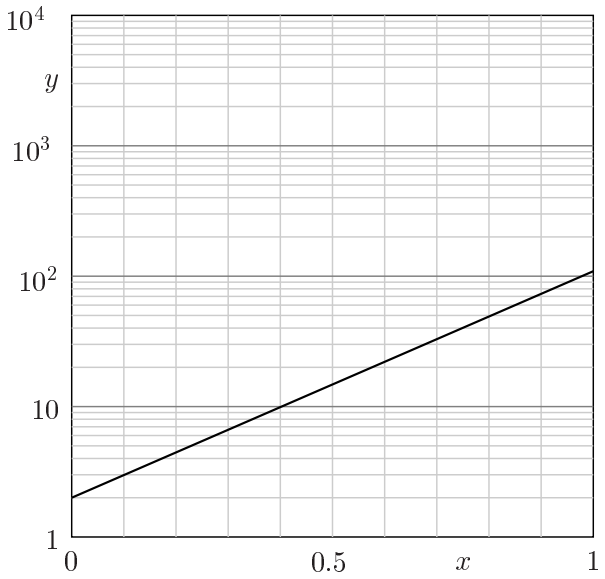

図 3.5 $x = [0, 1]$ における $y = 2.0 \exp(4x)$ のグラフ.

プログラム例 3.5 の実行結果は図**??**のようになり，このグラフの傾きはおよそ 1.74 であり，$\log_{10} e = 0.434294$ なので，指数関数の係数 a はおよそ 4 であることがわかる．また，その係数は $x = 0$ における y 軸の切片で，およそ 2 である．

付 録A　ポストスクリプト描画のための関数

ポストスクリプトを用いて描画するための関数 (ライブラリ pssub.c に含まれる関数) の簡単な説明を行う. 詳しくは著者のホームページ (URL `http://www1.doshisha.ac.jp/~jmizushi`) を参照のこと.

annulus ： 点 (x1, y1) を中心, 半径 r1 と r2 の 2 重円を描く.

呼び出し方: annulus(x1,y1,r1,r2)

(x1, y1)(double): 円の中心の座標.

r1(double): 内側の円の半径.

r1(double): 外側の円の半径.

arc ： 点 (x1, y1) を中心, 半径 r1 の円弧を角 t1 から t2 まで反時計回りに描く.

呼び出し方: arc(x1,y1,r1,t1,t2)

(x1, y1)(double): 円の中心の座標.

r1(double): 円の半径.

t1(double, 単位 [$^c irc$]): 円弧の始点角.

t2(double, 単位 [°]): 円弧の終点角.

arcn ： 点 (x1, y1) を中心, 半径 r1 の円弧を角 t1 から t2 まで時計回りに描く.

呼び出し方: arcn(x1,y1,r1,t1,t2)

(x1, y1)(double, 単位 [cm]): 円の中心の座標.

r1(double): 円の半径.

t1(double, 単位 [°]): 円弧の始点角.

t2(double, 単位 [°]): 円弧の終点角.

arcto ： 現在ポインタの位置と点 (x1, y1) および点 (x2, y2) を結ぶ折れ線曲り部を半径 r1 の円弧で置き換えて描く.

呼び出し方: arcto(x1,y1,x2,y2,r1)

(x1, y1)(double): 点 1.

(x2, y2)(double): 点 2.

r1(double): 円の半径.

arctorot ： 角度 t 回転した座標系で, 現在ポインタの位置と点 (x1, y1) および点 (x2, y2) を結ぶ折れ線曲り部を半径 r1 の円弧で置き換えて描く.

呼び出し方: arcto(x1,y1,x2,y2,r1,t)

(x1, y1)(double): 点 1.

(x2, y2)(double): 点 2.

r1(double): 円の半径.

t(double, 単位 [°]): 座標の回転角度.

arrow ： (x1,y1) から (x2,y2) に線を引き,(x2,y2) 側が矢印となる.

呼び出し方: arrow(x1,y1,x2,y2,d)

(x1, y1)(double): 始点の座標.

(x2, y2)(double): 終点の座標.

d(double): 矢印の大きさ.

arrowa ： (x1,y1) から (x2,y2) に線を引き,(x2,y2) 側が矢印となる. 矢印の先端が三角形.

呼び出し方: arrowa(x1,y1,x2,y2,d)

(x1, y1)(double): 始点の座標.

(x2, y2)(double): 終点の座標.

d(double): 矢印の大きさ.

arrowb ： (x1,y1) から (x2,y2) に線を引き,(x2,y2) 側が矢印となる. 矢印が凹型.

呼び出し方: arrowb(x1,y1,x2,y2,d)

(x1, y1)(double): 始点の座標.

(x2, y2)(double): 終点の座標.

d(double): 矢印の大きさ.

arrowc ： (x1,y1) から (x2,y2) に線を引き,(x2,y2) 側が矢印となる. 矢印の先端が釣り針のモドリ.

呼び出し方: arrowc(x1,y1,x2,y2,d)

(x1, y1)(double): 始点の座標.

(x2, y2)(double): 終点の座標.

d(double): 矢印の大きさ.

arrowfill ： (x1,y1) から (x2,y2) に線を引き,(x2,y2) 側が矢印となる.

呼び出し方: arrowfill(x1,y1,x2,y2,d)

(x1, y1)(double): 始点の座標.

(x2, y2)(double): 終点の座標.

d(double): 矢印の大きさ.

arrowrot ： 角度 t 回転した座標系で,(x1,y1) から (x2,y2) に線を引き,(x2,y2) 側が矢印となる.

呼び出し方: arrowrot(x1,y1,x2,y2,t,d)

(x1, y1)(double): 始点の座標.

(x2, y2)(double): 終点の座標.

t(double, 単位 [°]): 座標の回転角度. d(double): 矢印の大きさ.

arrowwide ： (x1,y1) から (x2,y2) に線を引き,(x2,y2) 側が矢印となる幅が al の太い白抜き矢印となる.

呼び出し方: arrowwide(x1,y1,x2,y2,d,al)

(x1, y1)(double): 始点の座標.

(x2, y2)(double): 終点の座標.

d(double): 矢印の大きさ. al(double): 矢印の太さ.

battery ： (x1,y1) から (x2,y2) に電気回路の電池を+極,-極の順に描く.
呼び出し方: battery(x1,y1,x2,y2,d,w)
(x1, y1)(double): 始点の座標.
(x2, y2)(double): 終点の座標.
d(double): 電池の大きさ. w: 線の太さ.

brokenlines ： n 個の実数型配列の座標を指定し, それらを折れ線で結ぶ.
呼び出し方: brokenlines(x1,y1,n)
(x1, y1)(double): 実数型配列の座標.
n(int): 実数型配列の個数.

circ ： 点 (x1, y1) を中心, 半径 r1 の円を反時計回りに描く.
呼び出し方: circ(x1,y1,r1)
(x1, y1)(double): 円の中心の座標.
r1(double): 円の半径.

circn ： 点 (x1, y1) を中心, 半径 r1 の円時計回りに描く.
呼び出し方: circn(x1,y1,r1)
(x1, y1)(double): 円の中心の座標.
r1(double): 円の半径.

clipoff ： クリップの解除.
呼び出し方: clipoff

clipon ： 設定した閉じた領域のパスのみ描画する. この関数を呼び出す前に, 閉じた領域を設定しておくこと.
呼び出し方: clipon()

cliponrec ： 点 (x1, y1) を左下頂点, 点 (x2, y2) を右上頂点とする四角形の内側のパスのみ描画する.
呼び出し方: clipon(x1,y1,x2,y2)
(x1, y1)(double): 長方形の左下頂点の座標.
(x2, y2)(double): 長方形の右上頂点の座標.

closepath ： パスの最初の点と最後に指定した点を結び, パスを閉じる.
呼び出し方: closepath

coil ： (x1,y1) から (x2,y2) に電気回路のコイルを描く.
呼び出し方: coil(x1,y1,x2,y2,d,n)
(x1, y1)(double, 単位 [cm]): 始点の座標.
(x2, y2)(double, 単位 [cm]): 終点の座標.
d(double): コイルの大きさ. n(int): コイルの巻き数.

curvto ： 現在ポインタが指し示す点から終点 (x3, y3) までを 3 次のベジエ (Bézier) 曲線でつなぐ. 現在ポインタが指し示す点である始点において, 始点と点 (x1, y1) を結ぶ線分に接し, 終点において, 点 (x2, y2) と終点を結ぶ線分に接する.
呼び出し方: curvto(x1,y1,x2,y2,x3,y3)
(x1, y1)(double, 単位 [cm]): 始点におけるコントロール点の座標.

(x2, y2)(double, 単位 [cm]): 終点におけるコントロール点の座標.
(x3, y3)(double, 単位 [cm]): 終点の座標.

curvton : n 個のコントロール点を指定し，現在ポインタが指し示す点から終点 (x[n-1], y[n-1]) までをベジエ (Bézier) 曲線で描く． 現在ポインタが指し示す点である始点において，始点と点 (x[0], y[0]) を結ぶ線分に接し，終点において，点 (x[n-2], y[n-1]) と終点を結ぶ線分に接する.
呼び出し方: curvto(x1[],y1[],n)
(x1[], y1[])(double, 1 次元配列): n 個のコントロール点の座標.
n(int): コントロール点の数.

curvtona : n 個のコントロール点を指定し，現在ポインタが指し示す点から終点 (x[n-1], y[n-1]) までをベジエ (Bézier) 曲線で描く． 現在ポインタが指し示す点である始点において，始点と点 (x[0], y[0]) を結ぶ線分に接し，終点において，点 (x[n-2], y[n-1]) と終点を結ぶ線分に接する． $0.5 < t < 1$ であり，t が大きいほど曲線は各コントロール点に近づく.
呼び出し方: curvto(x1[],y1[],t,n)
(x1[], y1[])(double, 1 次元配列): n 個のコントロール点の座標.
t(double, $0.5 < t < 1$): 曲線とコントロール点の近さ． n(int): コントロール点の数.

dims : 円弧を用いた寸法線を描く.
呼び出し方: dims(x1,y1,x2,y2)
(x1, y1)(double): 始点の座標.
(x2, y2)(double): 終点の座標.

dimsrot : 角度 t 回転した座標系で，円弧を用いた寸法線を描く.
呼び出し方: dims(x1,y1,x2,y2,t)
(x1, y1)(double): 始点の座標.
(x2, y2)(double): 終点の座標.
t(double, 単位 [°]): 座標の回転角度.

ellipse : (x1, y1) を中心とし，x 方向の半径 rx，y 方向の半径 ry の楕円を描く.
呼び出し方: ellipse(x1,y1,rx,ry)
(x1, y1): 楕円の中心の座標.
rx: x 方向の半径,ry: y 方向の半径.

ellipserot : 角度 t 回転した座標系で,(x1, y1) を中心とし，x 方向の半径 rx，y 方向の半径 ry の楕円を描く.
呼び出し方: ellipserot(x1,y1,rx,ry,t)
(x1, y1): 楕円の中心の座標.
rx: x 方向の半径,ry: y 方向の半径.
t(double, 単位 [°]): 座標の回転角度.

eoclipon : 指定した閉じたパスの外側のみを描画する.
呼び出し方: clipon()

eocliponrec ： 点(x1, y1)を左下頂点, 点(x2, y2)を右上頂点とする四角形の外側のパスのみ描画する.

呼び出し方: clipon(x1,y1,x2,y2)

(x1, y1)(double): 長方形の左下頂点の座標.

(x2, y2)(double): 長方形の右上頂点の座標.

fill ： 閉領域を塗りつぶす.

呼び出し方: fill

fin ： 基本的には, プログラムの最後に追加する.

呼び出し方: fin

grestore ： gsave で保存されたグラフィック状態に戻す.

呼び出し方: grestore

gsave ： 現在のグラフィック状態を保存.

呼び出し方: gsave

init ： ポストスクリプトの記述を始める. これ以降のポストスクリプトの記述はファイル 'temp1.ps' に書かれる. デフォルトの文字を18ポイントのタイムス(times-roman)とする. 長さの単位として cm が利用可能となる(デフォルトはcmとなる). 用紙はA4(21.0[cm]×29.7[cm])とする. 今後用紙の座標を機器座標と呼ぶことにし, 機器座標は左下を(0,0)右上を(1, 1.4) とする. 物理座標(世界座標)における(0,0)-(1,1)の正方形を機器座標の(0.2,0.2)-(0.8, 0.8)に対応づける. この対応付けを変更するときにはサブルーチン viewport と xyworld を用いる. デフォルトの線種として実線, 線の太さとして1(もっとも細い)を用いる. これらを変更するときは linetyとlinewidth を用いる.

呼び出し方: init

line ： 2点(x1, y1)と(x2,y2)を直線で結ぶ.

呼び出し方: line(x1,y1,x2,y2)

(x1, y1)(double): 始点の座標.

(x2, y2)(double): 終点の座標.

lgcurves ： n個の実数型配列の各座標をラグランジュ補間で描く.

呼び出し方: lgcurves(x1,y1,n)

(x1, y1)(double[]): 座標.

n(int): 結ぶ座標の数.

linerot ： 角度t回転した座標系で, 2点(x1, y1)と(x2,y2)を直線で結ぶ.

呼び出し方: linerot(x1,y1,x2,y2,t)

(x1, y1)(double): 始点の座標.

(x2, y2)(double): 終点の座標.

t(double, 単位[°]): 座標の回転角度.

linecap ： 2点を直線で結ぶ際の端点の形を指定.

呼び出し方: linecap(n)

n(int): 端点のタイプ. n=0:矩形,

n=1:　半円（線分の半値幅だけ外側になる），
n=2:　矩形（線分の半値幅だけ外側になる）．

linety ：　線のタイプを指定する．
呼び出し方：　linety(it)
it(int):　線のタイプ．　it=1:実線，
it=2:点線，it=3:長い点線，
it=4:短い破線，it=5:破線，
it=6:長い破線，it=7:短い１点鎖線，
it=8:１点鎖線，it=9:長い１点鎖線．

linewidth ：　線の太さを指定する．
呼び出し方：　linewidth(w)
w(double):　線の太さ．　w=1.0:細い，
w=2.0:標準，w=5.0:太い．

newpath ：　カレントパスをクリアする．
呼び出し方：　newpath

parabola ：　(x1,y1),(x2,y2),(x3,y3) を通る放物線を描く．
呼び出し方：　parabola(x1,y1,x2,y2,x3,y3)
(x1, y1),(x2, y2),(x3, y3)(double):通過点の設定．

plot ：　現在の点から (x,y) までポインターを移動する．
呼び出し方：　plot(x,y,ipen)
(x, y)(double):　終点の座標．
ipen(int):　ペンの状態．　ipen=2:線を描く（ペンを下ろす），ipen=3:線を描かない（ペンを上げる）．

plotrot ：　角度 t 回転した座標系で，現在の点から (x,y) までポインターを移動する．
呼び出し方：　plotrot(x,y,t,ipen)
(x, y)(double):　終点の座標．
ipen(int):　ペンの状態．　ipen=2:線を描く（ペンを下ろす），ipen=3:線を描かない（ペンを上げる）．
t(double, 単位 [°]):　座標の回転角度．

polygon ：　座標 (x,y) を中心とし，中心か頂点角までの長さを r とする正 n 角形を描く．
呼び出し方：　polygon(x,y,r,n)
(x, y)(double):　中心の座標．
r(double):　中心から頂点角までの長さ．
n(int):　頂点角の数．

polygonrot ：　角度 t 回転した座標系で，座標 (x,y) を中心とし，中心か頂点角までの長さを r とする正 n 角形を描く．
呼び出し方：　polygonrot(x,y,r,t,n)
(x, y)(double):　中心の座標．
r(double):　中心から頂点角までの長さ．

t(double, 単位 [°]): 座標の回転角度.
n(int): 頂点角の数.

rect : 点 (x1, y1) を左下頂点, 点 (x2, y2) を右上頂点とする長方形を描く.
呼び出し方: rect(x1,y1,x2,y2)
(x1, y1)(double): 長方形の左下頂点の座標.
(x2, y2)(double): 長方形の右上頂点の座標.

rectrot : 角度 t 回転した座標系で, 点 (x1, y1) を左下頂点, 点 (x2, y2) を右上頂点とする長方形を描く.
呼び出し方: rectrot(x1,y1,x2,y2,t)
(x1, y1)(double): 長方形の左下頂点の座標.
(x2, y2)(double): 長方形の右上頂点の座標.
t(double, 単位 [°]): 座標の回転角度.

rectround : 点 (x1, y1) を左下頂点, 点 (x2, y2) を右上頂点とする長方形の角を半径 r で丸くして描く.
呼び出し方: recroundt(x1,y1,x2,y2,r)
(x1, y1)(double): 長方形の左下頂点の座標.
(x2, y2)(double): 長方形の右上頂点の座標.
r(double, 単位 [cm]): 頂点角の半径.

rectroundrot : 角度 t 回転した座標系で, 点 (x1, y1) を左下頂点, 点 (x2, y2) を右上頂点とする長方形の角を半径 r で丸くして描く.
呼び出し方: rectroundrot(x1,y1,x2,y2,r,t)
(x1, y1)(double): 長方形の左下頂点の座標.
(x2, y2)(double): 長方形の右上頂点の座標.
r(double): 頂点角の半径.
t(double, 単位 [°]): 座標の回転角度.

resist : (x1,y1) から (x2,y2) に電気回路の抵抗を描く.
呼び出し方: resist(x1,y1,x2,y2,d)
(x1, y1)(double): 始点の座標.
(x2, y2)(double): 終点の座標.
d(double): 抵抗記号の振幅の大きさ.

rotate : 原点を中心に紙を角度 theta 回転する.
呼び出し方: rotate(theta)
itheta=0-360(double):反時計回りの回転角度.

rplot : 現在ポインタの位置 (x1,y1) から (x+dx,y+dy) までポインタを移動する.
呼び出し方: rplot(x1,y1,dx,dy)
(x1, y1)(double): 点1の座標.
(dx, dy)(double): 視点からの相対的な位置.

rplotrot : 角度 t 回転した座標系で, 現在ポインタの位置 (x1,y1) から (x+dx,y+dy) までポインタを移動する.

呼び出し方:　rplotrot(x1,y1,dx,dy,t)
(x1, y1)(double):　点 1 の座標.
(dx, dy)(double):　視点からの相対的な位置.
t(double, 単位 [°]):　座標の回転角度.

scale :　現在描いている図形の縮尺率を指定する.
呼び出し方:　scale(tx,ty)
tx=a:x 方向を a 倍に, ty=b:y 方向を b 倍に拡大, 縮小する.

setchar :　文字のタイプと大きさを指定する.
呼び出し方:　setchar(itype,ipoint)
itype(int):　文字のタイプ.
itype=1:　Times-Roman, itype=2:　Times-Bold, itype=3:　Times-Italic,
itype=4:　Times-BoldItalic,
itype=5:　Helvetica, itype=6:　Helvetica-Bold, itype=7:　Helvetica-Oblique,
itype=8:　Helvetica-BoldOblique,
itype=9:　Courier, itype=10:　Courier-Bold, itype=11:　Courier-Oblique,
itype=12:　Courier-BoldOblique,
itype=13:　Symbol
ipoint(double, 単位 [ポイント,pt]):　文字の大きさ (1pt $\simeq$ 0.3515mm).

setgray :　色（白黒）を指定する.
呼び出し方:　setgray(g)
g(double):　線の明るさ.　g=1.0:白, g=0.0:黒.

setlinejoin :　色（白黒）を指定する.
呼び出し方:　setgray(ljoin)
ljoin(int):　線の角の形.　ljoin=0:Miter join (尖った角), ljoin=1:Round join (丸い角), ljoin=2:Bevel join (切り落とした角).

setrgb :　色（カラー）を指定する.
呼び出し方:　setrgb(r,g,b)
r,g,b(double):　各色の強さ.　r=1.0:赤, g=1.0:緑, b=1.0:青.

setcmyk :　色（カラー）を指定する.
呼び出し方:　setcmyk(c,m,y,k)
c,m,b(double):　各色の強さ.　c=1.0:シアン, m=1.0:マゼンタ, y=1.0:イエロー, k=1.0:黒.

spline :　(x1, y1) から (xn, yn) のデータをスプライン曲線で結ぶ.
呼び出し方:　spline(x1,y1,x2,y2,x3,y3,x4,y4,ipart)
ipart は次のようにする.
(x1, y1)-(x4, y4)(double):ipart=-1,
(x2, y2)-(x5, y5)(double):ipart=-1,
(x2, y2)-(x5, y5)(double):ipart= 0,
(x(n-4), y(n-4))-(x(n-1), y(n-1)):ipart= 0,
(x(n-3), y(n-3))-(xn, yn):ipart= 0,

```
(x(n-3), y(n-3))-(xn, y):ipart= 1,
```

square ： 中心座標 (x1,y1) で一辺の長さが ell の正方形を描く．

呼び出し方： square(x1,y1,ell)

(x1, y1)(double)： 中心の座標．

ell)(double)： 一辺の長さ．

squarerot ： 角度 t 回転した座標系で，中心座標 (x1,y1) で一辺の長さが ell の正方形を描く．

呼び出し方： squererot(x1,y1,ell,t)

(x1, y1)(double)： 中心の座標．

ell)(double)： 一辺の長さ．

t(double, 単位 [°])： 座標の回転角度．

star ： 中心座標 (x1,y1) で頂点角までの長さが r の頂点数 n 個の星形を描く．

呼び出し方： star(x1,y1,r,t,n)

(x1, y1)(double)： 中心の座標．

r)(double)： 中心から頂点までの長さ．

n(int)： 頂点角の数．

starx ： 角度 t 回転した座標系で，中心座標 (x1,y1) で頂点角までの長さが r の頂点数 n 個の星形を描く．

呼び出し方： starx(x1,y1,r,t,n)

(x1, y1)(double)： 中心の座標．

r)(double)： 中心から頂点までの長さ．

t(double, 単位 [°])： 座標の回転角度．

n(int)： 頂点角の数．

stroke ： カレントパスを描画する．

呼び出し方： stroke

text ： 文字列を出力する．

呼び出し方： text(x1,y1,s[])

(x1, y1)(double, 単位 [cm])： 最初の 1 文字の左下の座標．

s(80 文字以内の文字)： 出力する文字列．

text1 ： 1 文字を出力する．

呼び出し方： text1(x1,y1,'s')

(x1, y1)(double, 単位 [cm])： 文字の左下の座標．

s(1 文字)： 出力する文字．

textx ： 文字列を出力する． 特に x 軸のメモリを入れる際に有効．

呼び出し方： textx(x1,y1,s[])

(x1, y1)(double, 単位 [cm])： 文字列の真中が (x1, y1) の少し下側にくる．

s(80 文字以内の文字)： 出力する文字列．

textrot ： 角度 t 回転した座標系で，文字列を出力する．

呼び出し方： text(x1,y1,t,s[])

(x1, y1)(double, 単位 [cm])： 最初の 1 文字の左下の座標．

t(double, 単位 [°])： 座標の回転角度．

s(80 文字以内の文字)： 出力する文字列．

texty ： 文字のタイプと大きさを指定する． 特に y 軸のメモリを入れる際に有効．

呼び出し方： texty(x1,y1,s[])

(x1, y1)(double, 単位 [cm])： ． 文字列の右端が (x1, y1) の少し左側にくる．

n(int)： 文字数．

s(80 文字以内の文字)： 出力する文字列．

translate ： 原点を移動する．

呼び出し方： translate(a,b)

a(double):x 方向に a 移動， b:y 方向に b 移動．

translateo ： 原点を移動する．

呼び出し方： translateo(t)

t(double, 単位 [°])： 傾き角度． a(double):x 方向から t 傾いた方向に a 移動，b:y 方向から t 傾いた方向にに b 移動．

triangle ： 点 (x1, y1) を左下頂点とする一辺 r1 の正三角形を描く．

呼び出し方： triangl(x1,y1,r1,a1)

(x1, y1)(double, 単位 [cm])： 正三角形の左下頂点の座標．

r1(double, 単位 [cm])： 正三角形の一辺の長さ．

a1(double, 単位 [°])： x 軸からの回転角度．

viewport ： 世界座標から切り取った図形をマッピングする機器座標の位置の設定．

呼び出し方： viewport(x1,y1,x2,y2)

(x1,y1):機器座標での左下の座標，(x2,y2):機器座標での右上の座標．

xyworld ： 世界座標において切り出す図形の位置の設定．

呼び出し方： xyworld(x1,y1,x2,y2)

(x1,y1):用紙の左下の座標， (x2,y2):用紙が A4 のときは (x1,y1) を一つの角とする一辺が 21.0[cm] の正方形の右上の座標．

関連図書

[1] Les Hancock・倉骨彰・三浦明美： Ｃ言語入門，アスキー

[2] Ｂ．Ｗ．カーニハン・Ｄ．Ｍ．リッチー（石田晴久訳）： プログラミング言語 C，共立出版

[3] 椋田實： はじめてのＣ（改訂第４版），技術評論社

[4] 石田晴久・後藤良和・高田大二・中島寛和： 入門 ANSI-Ｃ（三訂版），実教出版

[5] 金敷 準一： PAD 入門 初心者のための構造化プログラミング，サイエンス社

[6] アドビ・システムズ： ページ記述言語 PostScript チュートリアル& クックブック入門（野中浩一訳），アスキー

[7] アドビ・システムズ： PostScript リファレンスマニュアル第 3 版（桑沢清志訳），アスキー

索引