Rによるグラフィックスの作成

わし

作図を行う前に，R の作図の仕組みを簡単に説明する．

0.1　作図を行うには

R の作図機能は極めて多彩である．さまざまな種類の統計グラフを描述することが出来，全く新しい種類のグラフを描くことも出来る．R で作図を行なうには作図関数，作図デバイスの 2 つを用いて図を描く．

作図関数 ：　実際に何かを描画する関数
作図デバイス ：　図を出力する装置 (デバイス)[*1]

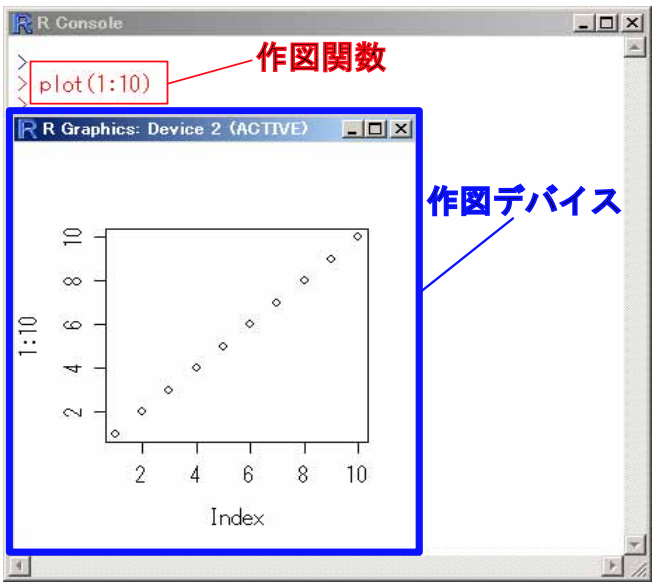


図 1　作図関数と作図デバイス

0.2　作図関数について

作図関数は出力装置に依存せず，例えば X ウインドウ上へ描画する場合でも，ポストスクリプト形式で作図出力を得る場合でも，同じ関数を使って同じ手順で作図が行なえるようになっている．作図命令は以下の 2 つの基本的なグループに分けられる[*2]．

高水準作図関数 ：　一枚の完成された図を作成するための関数で，例えば散布図や関数，ヒストグラムを描く．
低水準作図関数 ：　高水準作図関数で描いた図に追記するための関数で，線や点，文字列や多角形の塗りつぶしを行ったり，座標軸を描く，タイトルを記入するといった図の一部分を描いたりする．

[*1] 作図関数は high (low) level plotting function と graphics function の 2 通りの英語があるので困るのだが，作図関数と作図デバイスはそれぞれ，グラフィックス関数やグラフィックスデバイスという訳が適切なのかもしれない・・・ので，適宜読み替えて頂きたい．ただし，パラメータに関しては RjpWiki の記事にあったので，グラフィックスパラメータと訳している．

[*2] 対話的作図関数の説明は割愛する．

0.3 作図デバイスについて

R はいろいろな作図関数を備えており，作図結果を様々な出力装置から得ることが出来る．出力する装置のことを R では作図デバイスあるいは作図機器と呼ぶ．R が起動されたとき，例えば UNIX , Linux では自動的に関数 X11() で，Windows では関数 windows() でデバイスドライバが初期化され，デバイスが稼動する．一度デバイスが稼動し始めれば，R の作図命令で様々なグラフ表示や自分でグラフ表示を定義することが出来るようになる．通常，作図デバイスは R が起動されたときに自動的に呼び出されるので，グラフィックスを行う際に作図デバイス云々を気にする必要はほとんどない．

R がサポートしている作図デバイスを呼び出す関数は help("Devices") で調べることが出来るが，代表的なデバイス関数には以下のようなものがある．

デバイス	説明	デバイス	説明
bitmap()	ビットマップ	bmp()	ビットマップ
dev2bitmap()	ビットマップ	jpeg()	JPEG
pdf()	ADOBE PDF	pictex()	LaTeX ファイル
png()	PNG	postscript()	POSTSCRIPT
quartz()	(Mac OS 版のデフォルトドライバ)	windows()	(Windows 版のデフォルトドライバ)
win.graph()	(Windows 版のドライバ)	win.metafile()	Windows 版のメタファイル形式
win.print()	(Windows 版のドライバ)	x11()	(UNIX のデフォルトドライバ)
X11()	(UNIX のデフォルトドライバ)	xfig()	Xfig

あるデバイスの利用が終了した場合に dev.off() でデバイスを閉じることも出来る．例えば postscript() でポストスクリプトファイルを作成する場合，ファイルが壊れる不具合を防ぐには dev.off() でデバイスをすぐに閉じればよい．

```
> data(cars)                                              # PS デバイスを開き，出力 file を指定
> postscript("myplot.eps", horizontal=FALSE, height=9, # horizontal = FALSE を指定しないと
              width=14,    pointsize=15)                 # TeX に取り込んだ際，横にひっくり返る
> plot(cars, main = "Speed and Stopping Distances of Cars")
> dev.off()          # 必要な出力がすべて終ったらすぐにデバイスを閉じると不具合が起こりにくい
null device
          1
```

(注) 作業ディレクトリのフルパスを指定しない限り，画像ファイル (この場合は myplot.eps) は現在の作業ディレクトリに保存される．作業ディレクトリの参照・変更方法は **??** 頁の「作業ディレクトリの変更」を参照のこと．

0.4 複数のデバイスドライバ

同時に複数の作図デバイスを持つということがある．もちろん一度に一つの作図デバイスだけが作図命令を受け入れることが出来る (このデバイスをカレントデバイスという) のだが，複数のデバイスが開かれているときは，それらは待機状態としてデバイスリストを成していることになる．複数のデバイスを操作するための主要な命令は以下の通りである．

デバイス	説明
デバイス関数（bmp()，jpeg()，pdf() など）	新しい作図デバイスを開いて待機状態を表すデバイスリストを更新し，現在のデバイス (カレントデバイス) に指定する．今後はこのデバイスにグラフ出力が送られることになる．
dev.list()	アクティブなデバイスの番号と名前を返す．
dev.next()	カレントデバイスの次の位置の作図デバイスの番号と名前を返す．
dev.prev()	カレントデバイスの前の位置の作図デバイスの番号と名前を返す．
dev.set(which=k)	デバイスリストの k 番目の作図デバイスの番号と名前を返す．
dev.off(k)	デバイスリストの k 番目の作図デバイスを終了させる．
dev.copy(device, which=k)	作図デバイス k のコピーを作成する．device は postscript などのデバイス関数を引数にする．
dev.copy2eps(file="")	eps 出力を指定した dev.copy() と同じ動作を行う．様々なフォント名を指定することが出来る．
dev.print(device,which=k)	アクティブな作図デバイス k のコピーを作成し，他のグラフィックスデバイス (既定では postscript) に再出力する．複製されたデバイスは直ちに閉じられ，終了処理がなされる．
graphics.off()	無効なデバイスを取り除き，リスト中の全てのデバイスを終了する．

以下に，図を PDF ファイルに保存する例を挙げる．後者の方法ならば，png() ，bmp() ，jpeg() を指定することで，それぞれの形式の画像ファイルが得られる．

```
> plot(1:10)
> dev.copy(pdf, file="finename.pdf")
> dev.off()                                    # 必要な出力がすべて終ったらすぐにデバイスを閉じる
>                                              # または，前述の作図デバイスで同じことをやると ...
> pdf()                                        # pdf デバイスを開く ()
> plot(1:10)                                   # プロット -> Rplots.pdf に出力
> dev.off()                                    # 必要な出力がすべて終ったらすぐにデバイスを閉じる
```

また，図を eps 形式で保存する場合は以下のようにすればよい．

```
> X11()                                        # X11 デバイスを開く（ OS によっては省略可）
> plot(1:10)
> dev.copy2eps(file="finename.eps", width=6) # 幅と高さの一方を省略してもよい
>                                              # または... dev.print() でコピー出力すると...
> X11()                                        # X11 デバイスを開く（ OS によっては省略可）
> plot(1:10)                                   # X11 デバイスに出力
> dev.print(file="finename.eps",               # 既定で EPS ファイルに出力
            width=10, height=10, horizontal=FALSE)
```

Windows 版 R ならば，図を右クリックしてメニューから出力することが出来る．メニューの機能は上から『 metafile としてクリップボードにコピー』『 bmp としてとしてクリップボードにコピー』『 metafile (.emf) 形式で保存』『 postscript (.ps , .eps) 形式で保存』『プリントアウト』となっている．

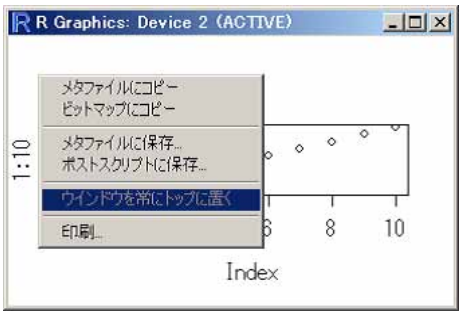


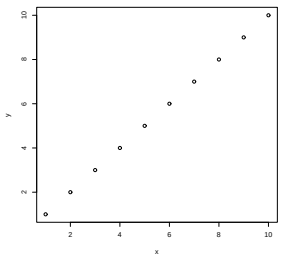
図 2　Windows 版 R の右クリックメニュー

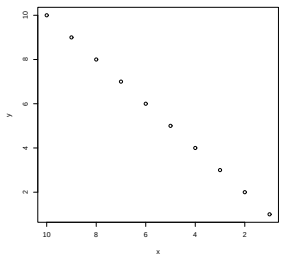
0.5　とりあえず plot()

R で一番良く使われる高水準作図関数が関数 plot() である．最も基本的で機能も多い関数も plot() である．この関数を使って散布図や折れ線グラフなどを描くことが出来る．例えばデータが入っているベクトル x，y を点の座標として以下の様に入力する．

```
> x <- 1:10
> y <- 1:10                  # plot(x 軸のデータ，y 軸のデータ，オプション)
> plot(x, y)                 # 範囲は自動で決まる (xlim=c(1,10) を指定した場合と同じ)
> plot(x, y, xlim=c(10,1)) # x 軸の正の向きを左向きにすることも出来る
```

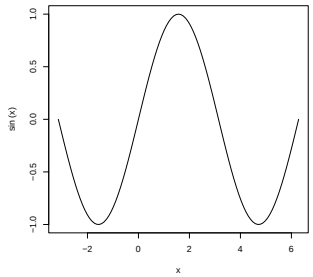
すると散布図の出力が得られる．プロット範囲は引数 xlim, ylim で決めることが出来る．

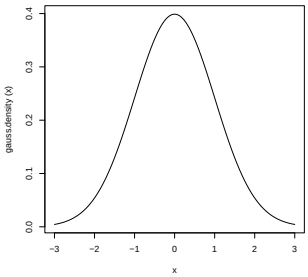




また，数学関数を与えてそのグラフを出力することも出来る．

```
> plot(sin, -pi, 2*pi)                                  # plot(関数名，下限，上限)
> gauss.density <- function(x) 1/sqrt(2*pi)*exp(-x^2/2)  # 標準正規分布の密度
> plot(gauss.density,-3,3)
```



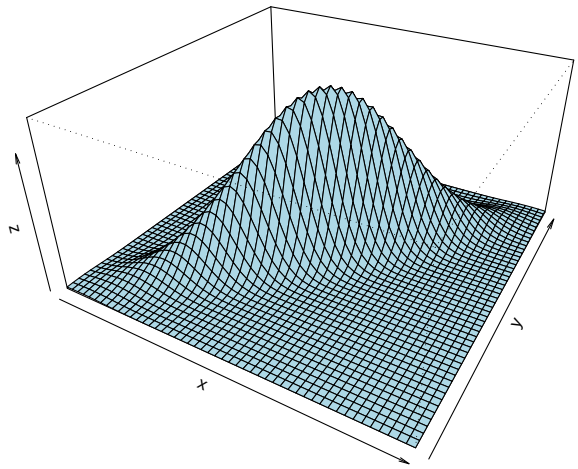


ちなみに，3 次元的にプロットする場合は関数 persp() を用いる．

```
> persp(x 軸のデータ，y 軸のデータ，z 軸のデータ，col = 色，
+       theta = 横回転の角度，phi = 縦回転の角度，expand = 拡大率，border=NA)
```

例として 2 次元正規分布を描く.

```
> x <- seq(-3,3,length=50)      # x 方向の分点
> y <- x                        # y 方向の分点
> rho <- 0.9                    # 2 次元正規分布の定数
> gauss3d <- function(x,y) {    # 2 次元正規分布の関数
+   1/(2*pi*sqrt(1-rho^2))*exp(-(x^2-2*rho*x*y+y^2) / (2*(1-rho^2)))
+ }
> z <- outer(x,y,gauss3d)       # 外積をとって z 方向の大きさを求める
> z[is.na(z)] <- 1              # 欠損値を 1 で補う
> persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue")
```



0.6 plot() の形式指定

関数 plot() の引数 type によって様々な形式でプロットすることが出来る．type には次の 9 種類の文字のいずれかを指定することが出来る.

引数	機能
type=”p”	点プロット (デフォルト)
type=”l”	線プロット (折れ線グラフ)
type=”b”	点と線のプロット
type=”c”	”b” において点を描かないプロット
type=”o”	点プロットと線プロットの重ね書き
type=”h”	各点から x 軸までの垂線プロット
type=”s”	左側の値にもとづいて階段状に結ぶ
type=”S”	右側の値にもとづいて階段状に結ぶ
type=”n”	軸だけ描いてプロットしない (続けて低水準関数でプロットする場合)

以下に例を示す.

```
> x <- rnorm(10)
> plot(x, type="l")
```

0.7 対数軸や軸の範囲の指定

軸に関する設定を行なう為に以下の引数を追加することが出来る.

引数	機能
log=”x”	”x” (x 対数軸) , ”y” (y 対数軸) , ”xy” (両対数軸) の何れかを指定することが出来る (対数は常用対数のみ) .
xlim=c(0, 1), ylim=c(−1, 1)	長さ 2 のベクトルで x 座標, y 座標の最小値と最大値を与える, 他にも xlog, ylog で対数プロットが出来る. ベクトルを降順に並べる (例 : c(2, −2)) と, プロットの向きが逆になる.
axes=FALSE	軸の生成を抑制する. 軸の他に表題, 刻み, 目盛も描くかどうかを論理値で指定する (省略時は TRUE). 他に xaxs, yaxs が指定出来る.

以下に例を示す.

```
> x <- rnorm(10)
> plot(x, ylim=c(-30, 30), type = "l")
```

0.8 タイトルなどの指定

表題などに関する設定を行なう為に以下の引数を追加することが出来る.

引数	機能
main=”Title”	タイトルを与える文字列を指定する. この引数を省略するとは表題は描かれない.
sub=”SubTitle”	サブタイトルを与える文字列を指定する. この引数を省略すると副題は描かれない.
xlab=”X-Label”, ylab=”Y-Label”	それぞれ x 座標名, y 座標名を与える文字列を指定する. 省略すると, x 軸のデータとして与えられた引数の名前が座標名として描かれる.
ann = F	軸のラベルを描かないようにすることも出来る (xlab = ”” , ylab = ”” を同時に指定した場合と同じ.) .
tmag=1.2	プロットの別の注釈するテキストに関する主なタイトルのテキストの拡大率を指定する.

以下に例を示す.

```
> x <- rnorm(10)
> plot(x, main="Simple Time Series")
```

0.9　図の重ね合わせ

図を重ね合わせるには，関数 par() を用いるか，引数に add=T を入れる．

(注) plot() 以外の関数は add=T が効かない場合が多いので注意．

```
> plot(sin, -pi, pi, xlab="", ylab="", lty=2)   # sin(x) を描く
> par(new=T)                                    # 上書き指定
> plot(cos, -pi, pi, xlab="x", ylab="y")        # cos(x) を上書き
> plot(sin, -pi, pi, xlab="x", ylab="y", lty=2) # 新たに sin(x) を描く
> plot(cos, -pi, pi, add=T)                     # cos(x) を上書き
```

0.10　点の色を条件に応じて変える

点の色を条件に応じて変える場合は以下のように条件分岐すればよい．

```
> x <- runif(100)
> y <- runif(100)
> plot(x, y, col = ifelse(y>0.5, "red", "blue")) # y > 0.5 なら赤，その他は青
```

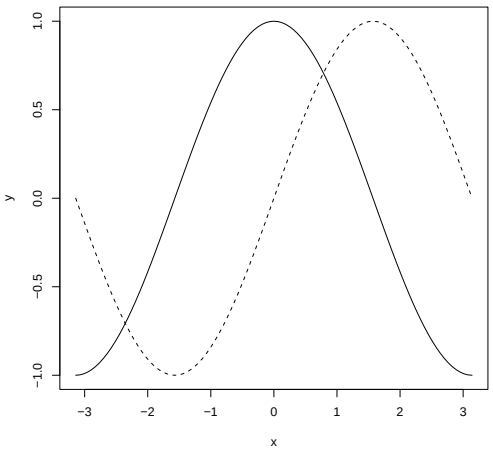


図 3　図の重ね合わせ

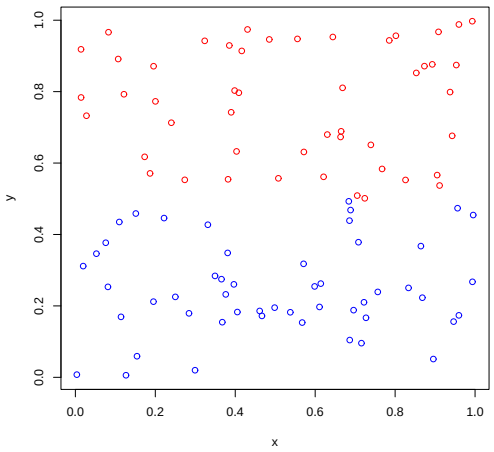


図 4　点の色を条件に応じて変える

0.11　図の消去

プロットした図を消去するには関数 frame() または関数 plot.new() を使う．グラフィックスパラメータ new が TRUE ならば図は消去されない．

```
> frame()
> plot.new()
```

第 1 章

高水準作図関数

以下に種々の高水準作図関数を紹介する．

1.1　散布図

1.1.1　plot()

plot() の詳しい説明は前節を御覧頂きたい．ここでは，plot() への引数の与え方による出力の違いを見てみる．

plot(x) ：　ベクトル x の要素が実数ならば，x は時系列データとみなされ，横軸を自然数，縦軸をデータ x 要素とする時系列プロットが描かれる．

- ベクトル x が時系列データならば，そのまま時系列プロットが描かれる．
- ベクトル x の要素が複素数ならば，横軸を実数，縦軸を虚部とするプロットが描かれる．
- x が 2 列の行列ならば，横軸を一列目，縦軸を 2 列目とするプロットが描かれる．
- x が 2 次元リストならば，その要素を横軸，縦軸としてプロットが描かれる．ただし names() を使ってどちらが x なのか y なのかラベルをつける必要がある．

```
> plot(rnorm(10))
> x <- list(1:5, 3:7);   names(x) <- c("X", "Y")
> plot(x)
```

plot(x, y) ：　データが入っているベクトル x , y やリスト x , y を点の座標として与えると散布図を描く．

```
> x <- rnorm(10);   y <- rnorm(10)
> plot(x,y)
```

plot(y ~ x) ：　以下の様に回帰式として入力することも出来る．

```
> x <- rnorm(10);   y <- rnorm(10)
> plot(y ~ x)
```

plot(f) ：　f は因子オブジェクトである． f の棒グラフを描く．

```
> f <- factor(c(rep("A",3), rep("B",5)))
> plot(f)
```

plot(f, y) ：　f は因子オブジェクト，y は数値ベクトルである．f の各水準に対する y の箱ひげ図を描くときに使う．

```
> x <- factor(c(rep("A",3), rep("B",5)));   y <- rnorm(8)
> plot(f, y)
```

plot(df) ：　df はデータフレームである．データフレーム中の変量のプロットを行う．

plot(˜ expr) ：　expr は ”+” で仕切られたオブジェクト名のリスト (例えば a+b+c) である．名前が与えられたオブジェクトの分布関数のプロットを行う．

```
> plot(~ group, data=sleep)
> plot(~ extra, data=sleep)
```

plot(y ˜ expr) ：　y は任意のオブジェクト，expr は ”+” で仕切られたオブジェクト名のリスト（例えば a+b+c ）である．expr に名前が与えられた全てのオブジェクトに対して y のプロットを行う．

1.1.2　dotchart()

関数 dotchart() でドットチャートを描く棒グラフの棒の代わりに点でプロットする[*1].

```
dotchart(データ, groups=グループのベクトル,
         gdata=NULL, labels=ラベル, ...)
```

```
> dotchart(1:10, group=c(rep(1:2,5)),
+          labels=paste("sample", 1:10))
```

1.1.3　stripchart()

関数 stripchart() で因子レベル別に一次元散布図を描く

```
stripchart(データ ~ 因子ベクトル,
           vertical = F, group.names=,...)
```

```
> y <- rnorm(20); x <- factor(rep(1:2,10))
> stripchart(y ~ x)
```

1.1.4　sunflowerplot()

関数 sunflowerplot() でヒマワリ図を描く．これは 1 点の周りに複数データが対応する際に，点の周りに重なった分だけ花弁を描く．

```
sunflowerplot(x 軸データ, y 軸データ, digits=6,
              xlab = "", xlim = NULL, add = F, ...)
```

```
> x <- round(rnorm(50), d=1)
> y <- data.frame(x, x)
> sunflowerplot(y)
```

1.1.5　curve()

関数 curve() を用いれば，関数を直接指定してグラフを出力することも出来る．引数 n でポイントの数を指定することが出来る．

```
curve(関数, from=下限, to=上限, n = 点の数,
      add = FALSE, type = "l", xlim = NULL, ...)
```

```
> curve(sin(x^2)*exp(-x^2),-pi,pi)
```

1.1.6　matplot()

matplot() は行列を引数にとり，その各列についてプロットを行う．完成図は，座標設定や列ごとにマーカーの色や形を自動的に設定し，見た目に区別がつくように描かれる．また，上書き用に 関数 matpoints() ，matlines() が用意

[*1] 似たような関数に rug() がある

```
plot(density(faithful$eruptions))
rug(faithful$eruptions, side=1)
```

されている．

```
matplot(行列（もしくは 2 つのベクトル），
        type = "p", lty = 1:5, lwd = 1,
        pch = NULL, col = 1:6, ...)
```

```
> myfunc <- function(x,y) sin(x/20 * pi * y)
> sines  <- outer(1:20, 1:4, myfunc)
> matplot(sines, pch = 1:4, type = "o")
```

1.1.7　出力結果一覧

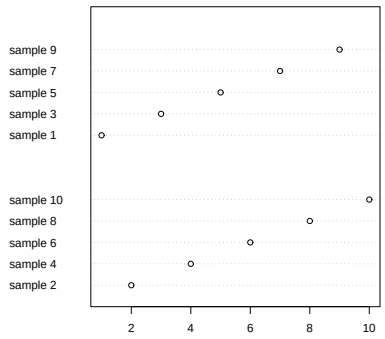


図 1.1　dotchart()

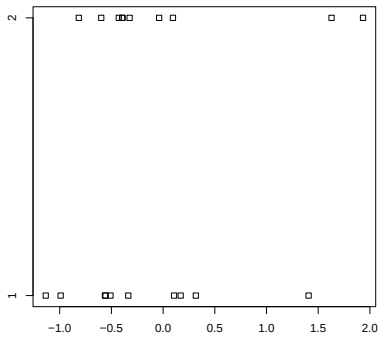


図 1.2　stripchart()

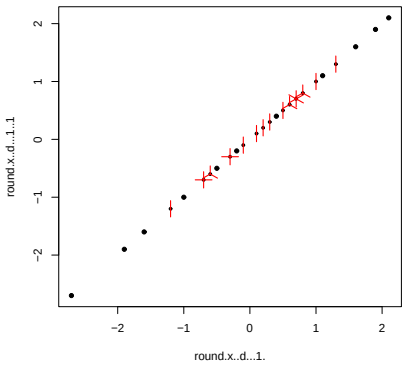


図 1.3　sunflowerplot()

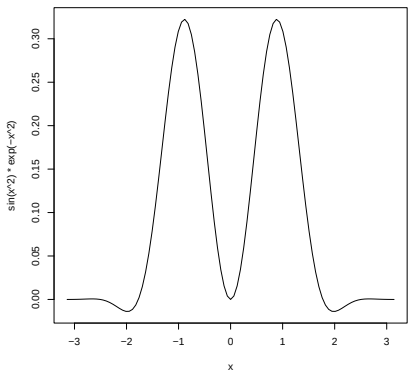


図 1.4　curve()

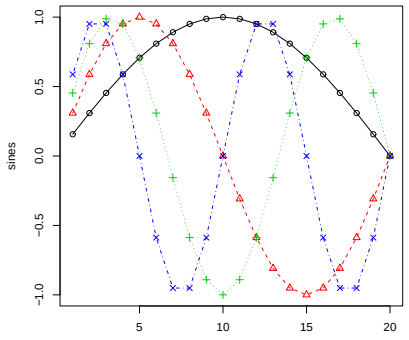


図 1.5　matplot()

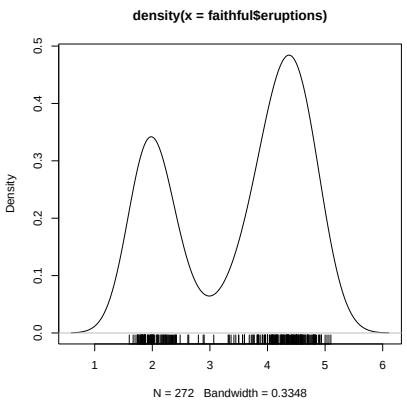


図 1.6　rug()

1.2　一次元データの表現

1.2.1　ヒストグラム：hist()

関数 hist() でヒストグラムを描くことが出来る．異なる区切り幅のヒストグラムを描くことも出来るが，主観的なバイアス (偏り) が入る危険があるのでお勧めは出来ない[*2]．

```
> x <- rnorm(50)                          # 一次元データ
> hist(x, breaks = seq(-3,3,1) )          # -3 から -3 まで 1 ずつの幅で描く
> hist(x, breaks = c(-3,-1,0,0.5,3) )     # 異なる区切り幅（出力例は省略）
```

[*2] 詳しくは『統計解析』の章で述べる．

1.2.2　棒グラフ：barplot()

関数 barplot() で棒グラフ描くことが出来る．データにベクトルを指定すると各要素の長さについて棒グラフが描かれ，データに行列を指定すると，一本（一列）の棒に各行の要素が層別で表示される．

```
barplot(ベクトルデータ, ...)
barplot(行列データ, ...)
```

```
> barplot(1:10)
> barplot(matrix(1:20, 5), col=rainbow(5))
```

関数 barplot() のオプションは以下の通り．ところで，barplot() は各棒の中心位置の x 座標を返すので，この値と lines などの低水準作図関数を使うことで，例えば信頼区間を描くことが出来る．

引数	機能
angle=45	棒を斜線で塗る際の線分の角度を指定する．
beside=F	（行列データで層別表示する場合）T にするとグループ毎に棒が並べて表示され，F にすると一本の棒に層別で表示される．
col=rainbow(10)	棒を塗りつぶす際の色を指定する．（ベクトルでも可）
density=30	棒を斜線で塗る際の線分の密度を指定する．
horiz=F	T にすると棒が水平に表示される．
legend = rownames(x)	凡例を描く．（例はデータ名が x の場合）
names=文字型ベクトル	各棒のラベルを文字型ベクトルで指定する．
space=1	各棒の間の間隔を指定する．
width=数値型ベクトル	各棒の相対的な幅を指定する．

1.2.3　円グラフ：pie()

関数 pie() で円グラフ描くことが出来る．

```
pie(データ, labels = names(x), radius = 0.8,
        density = NULL, angle = 45, col = NULL, ...)
```

```
> pie(1:10, r=1,
      col=rainbow(10))
```

関数 pie() のオプションは以下の通り．

引数	機能
angle=45	棒を斜線で塗る際の線分の角度を指定する．
border=NULL	NA にすると仕切り線が描かれなくなる．
col=rainbow(10)	棒を塗りつぶす際の色を指定する．（ベクトルでも可）
density=30	棒を斜線で塗る際の線分の密度を指定する．
main=”Title”	タイトルを指定する．
radius=0.8	円の半径を −1〜1 の範囲で指定する．負の値にすると，データを表示する開始位置が左端からとなる．

1.2.4　箱ひげ図：boxplot()

複数のデータの分布の違いを比較する際は，関数 boxplot() で箱ひげ図を描く．

```
boxplot(x, range = 1.5, width = NULL,
        horizontal = FALSE, add = FALSE,...)
boxplot(x ~ 因子ベクトル)
```

```
> x <- rnorm(100)
> boxplot(x)
> boxplot(len ~ dose, data = ToothGrowth)
```

関数 boxplot() のオプションは以下の通り．

引数	機能
boxwex = 0.8	箱の幅を指定する．
horizontal=F	T にすると，箱が横向きに描かれる．
notch=F	箱に notch を入れるかどうかを指定する．
outwex = 0.5	outlier の幅を指定する．
range=1.5	「ひげ」が箱の端からどれだけ離れるかを指定する．range が正の値であるなら，「ひげ」は箱から四分位範囲の range 倍（デフォルトは 1.5 倍）である最も端にあるデータ点となる．例えば，上側の「ひげ」の場合は Q3 + 1.5×(Q3−Q1) となる．range が 0 ならば「ひげ」はデータ範囲の一番端の値になる．
staplewex = 0.5	ひげの幅を指定する．
width=数値ベクトル	箱の幅を指定する．

1.2.5　出力結果一覧

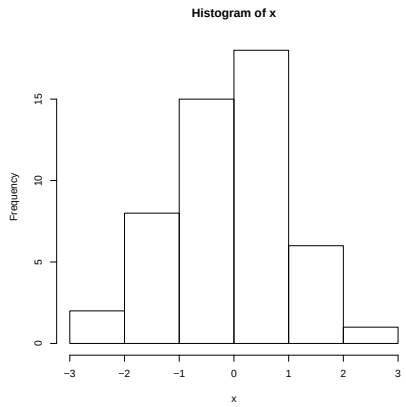


図 1.7　hist()

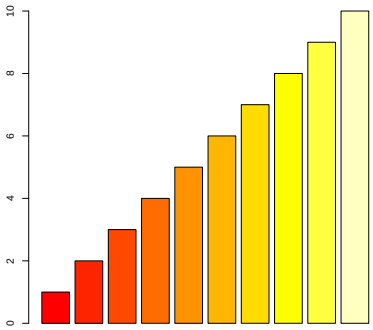


図 1.8　barplot()−1

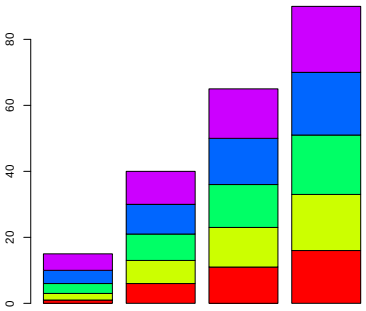


図 1.9　barplot()−2

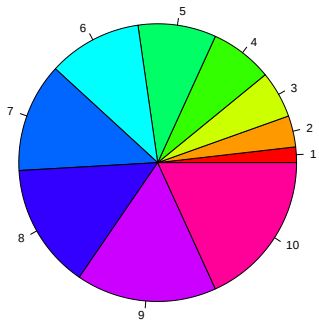


図 1.10　pie()

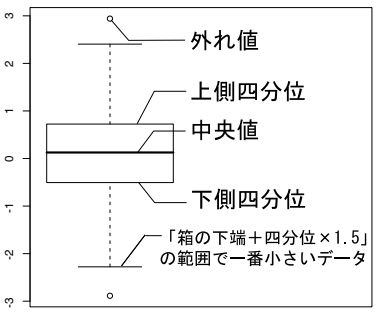


図 1.11　boxplot()−1

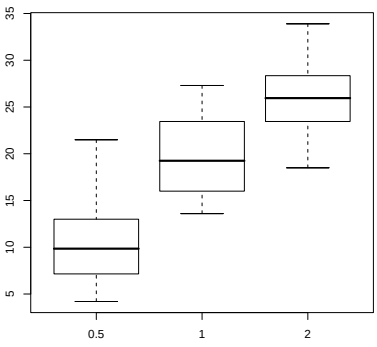


図 1.12　boxplot()−2

1.3　分割表データの図示

1.3.1　fourfoldplot()

関数 fourfoldplot() で 1 個以上の層について，2 変数間の関係を考慮に入れた 2 × 2 のグラフを生成する．

```
fourfoldplot(2*2 分割表データ)
```

```
> x <- matrix(1:4,2)
> fourfoldplot(x, col=1:2)
```

1.3.2　mosaicplot()

関数 mosaicplot() で 分割表データをモザイクプロットとして表示する．引数 dir=”h” (=”v”) で分割方向が変わる．

```
mosaicplot(分割表データ)
mosaicplot(モデル式, data=データ名)
```

```
> mosaicplot(Titanic, color = T)
> mosaicplot(~ Sex+Age+Survived, data = Titanic)
```

1.3.3　assocplot()

関数 assocplot() で分割表のデータについて，Cohen-Friendly の連関プロット（Association Plots）を行う．

```
assocplot(分割表データ)
```

```
> x <- margin.table(HairEyeColor, c(1,2))
> assocplot(x)
```

1.3.4　出力結果一覧

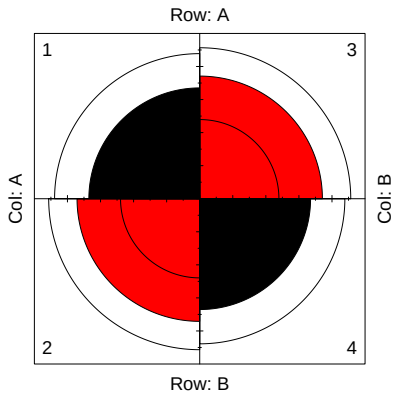


図 1.13　fourfoldplot()

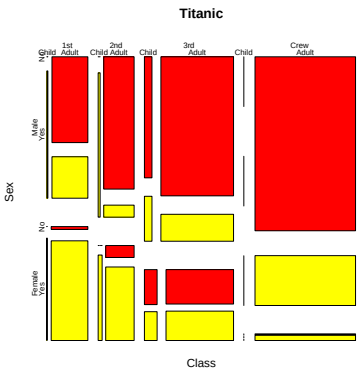


図 1.14　mosaicplot()

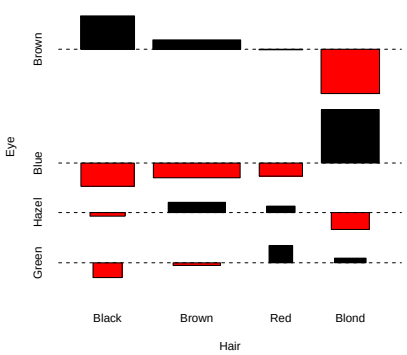


図 1.15　assocplot()

1.4　多変量データの図示

1.4.1　stars()

関数 stars() を用いることで，くもの巣プロットを描いて全体の傾向を見たり，星形図を描いてサンプルの分類を行うことが出来る．

```
> stars(mtcars[, 1:6], locations = c(0,0), radius = T,
+       key.loc=c(0,0.0), main="Motor Trend Cars")        # くもの巣プロット
> stars(mtcars[, 1:7], len = 0.8, key.loc = c(12, 1.5),
+       main = "Motor Trend Cars", draw.segments = TRUE)  # 星形図
```

1.4.2　symbols()

関数 symbols() で多変量データを図示する散布図を描くことが出来る．ただし，点や線の代わりに，円や星，箱ひげ図で散布図が描かれる．

```
> x <- 1:10;  y <- sort(10*runif(10));  z <- runif(10)
> symbols(x, y, thermometers=cbind(0.5,1,z), inches=0.5, fg=1:10)
```

温度計の記号以外にも様々な記号で表すことが出来る．

記号・機能	引数の指定方法
円	circles=数値ベクトル（半径）
正方形	squares=数値ベクトル（半径）
長方形	rectangles=数値ベクトル（半径）
星	stars=3 列以上の行列（星の中央からの線の長さ，0 以上 1 以下）
温度計	thermometers=c(幅，高さ，塗りつぶす高さの比率) または thermometers=c(幅，高さ，塗りつぶす比率 1，塗りつぶす比率 2)
箱ひげ図	boxplots=c(幅，高さ，下ひげの長さ 1，上ひげの長さ 2，中央線の位置)
シンボルの色	fg=数値ベクトル
塗りつぶす色	bg=数値ベクトル
単位	inches=F ならば単位は x 軸の単位が使われる．inches=T ならば最大のシンボルが高さ 1 インチであるようにシンボルが指定される．inches=数値ならば最大のシンボルが（インチ単位で）指定した数値の高さになるように調節される．

1.4.3　pairs()

関数 pairs(行列) で各列同士の組合せ全てについて散布図を描く．関数 pairs(モデル式) で，細かい条件を指定した上で，各列同士の組合せ全てについて散布図を描く．

```
> pairs(iris[1:4])                                          # 行列で指定
>
> pairs(~ Fertility + Education + Catholic, data = swiss,
+       subset = Education < 20, main = "Education < 20")  # モデル式で指定
```

1.4.4　coplot()

関数 coplot() で共変量プロットを描く．この関数は数値ベクトル x，y，z (z は因子オブジェクトでも可) を引数としてとり，z の与えられた値における y に対する x の散布図を複数生成する．

coplot(y ~ x | z) ：　z が因子ならば，z の水準ごとに y が x に対してプロットされる．z が数値ならば，いくつかの『条件を与える区間』に分割され，その区間に含まれる z の値に対して y が x に対してプロットされる．

coplot(y ~ x | $z_1 * z_2$) ：　2 変量 z_1，z_2 に条件付けされた，x に対する y の散布図を生成する．

```
> x <- 1:10;  y <- rnorm(10);  z <- x*y
> coplot(y ~ x | z)
```

1.4.5　出力結果一覧

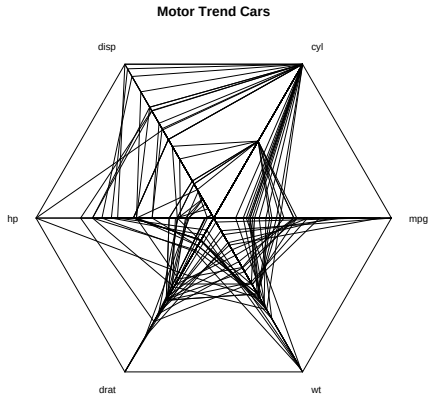


図 1.16　stars()−1

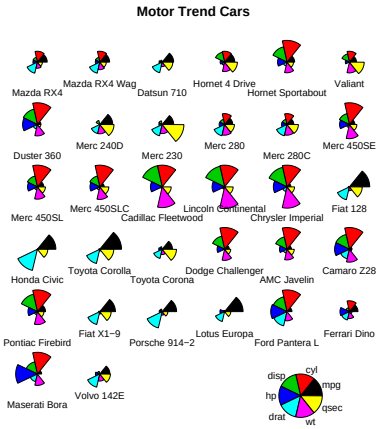


図 1.17　stars()−2

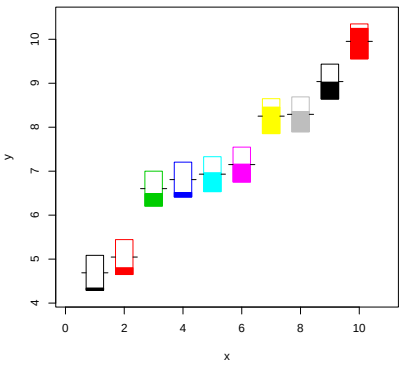


図 1.18　symbols()

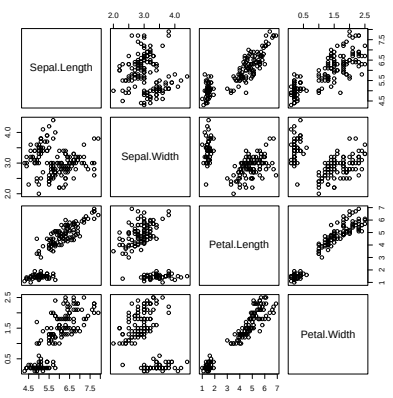


図 1.19　pairs()−1

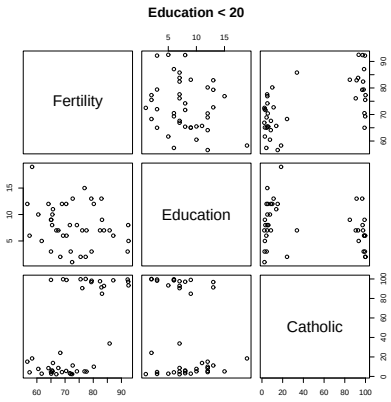


図 1.20　pairs()−2

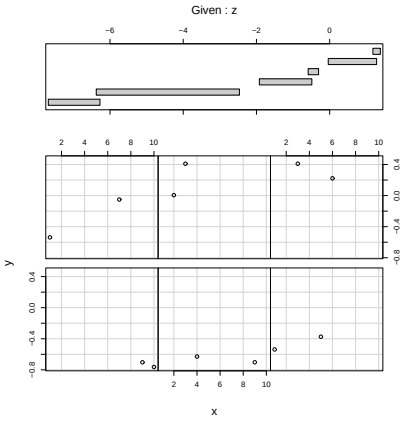


図 1.21　coplot()

1.5　3 次元データの図示

1.5.1　image()

関数 image() で 3 次元データをカラーイメージで図示する.

```
> x <- 10*(1:nrow(volcano));   y <- 10*(1:ncol(volcano))
> image(x, y, volcano, col = terrain.colors(100), axes = FALSE)
```

1.5.2　persp()

関数 persp() で 3 次元立体図を描く.

```
> x <- seq(-10, 10, length= 50);   y <- x
> f <- function(x, y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
> z <- outer(x, y, f);
> persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = rainbow(50), border=NA)
```

1.5.3　contour()

関数 contour() で 3 次元データを等高線図で図示する. 関数 filled.contour() も同様の関数である.

```
> x  <- -6:16
> contour(outer(x, x), method = "edge", vfont = c("sans serif", "plain"))
```

1.5.4　出力結果一覧

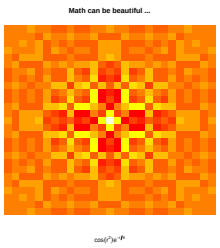


図 1.22　image()

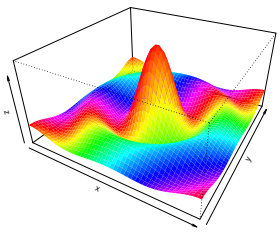


図 1.23　persp()

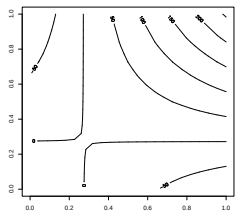


図 1.24　contour()

第 2 章

低水準作図関数

高水準関数で描いた図に文字や図形を描き加える場合は低水準作図関数を用いる．低水準作図関数はグラフィックスパラメータを引数として受け入れることも出来る．

2.1　低水準作図関数一覧

まず，低水準作図関数の一覧表を挙げる．

種類	関数	機能
点	points(x, y) , points(c(x, y))	各点の x 座標と y 座標を指定することで点列を描く (規定では points() に対して，関数の引数 type に ”p” を与える) ．マーカーの形式はグラフィックスパラメータ pch によって指定する．また，points(approx(x, y)) でデータの線形補間が行える．
直線	lines(x, y) , lines(c(x, y))	各点の x 座標と y 座標を指定することで，それぞれの点を通る直線・曲線を描く．点列を描く (規定では lines() に対して，関数の引数 type に ”l” を与える) ．
直線	abline(a, b), abline(c(a, b))	a ，b は切片と傾きを表し，直線 y = a + bx を描く．
直線	abline(h = y)	ベクトルで与えられた y 座標の水平線を引く．
直線	abline(v = x)	ベクトルで与えられた x 座標の垂直線を引く．
直線	abline(result)	result は関数 lm() で直線回帰を行った結果が入ったオブジェクトで，回帰直線が描かれる．
格子	grid(a, b)	a × b 本の格子を描く．引数として col, lty, lwd が指定できる
線分	segments(x0, y0, x1, y1)	始点の座標 (x0，y0) と，終点の座標 (x1 ，y1) を通る線分を描く．
矢印	arrows(x0, y0, x1, y1)	始点の座標 (x0，y0) と，終点の座標 (x1 ，y1) を通る矢印を描く．
矩形	rect(x0, y0, x1, y1)	始点の座標 (x0，y0) と，終点の座標 (x1 ，y1) を通る長方形を描く．引数として col, border, density が指定できる
文字	text(x, y, labels)	座標 (x，y) にラベルが描かれる．座標値と文字列をベクトルで指定することも出来る．text の引数に srt = − (回転角) を入れることで，プロットの x 軸ラベルを 45 度回転させることが出来る．
文字	mtext(text, side = 3, line = 0, at = NA)	書き込む文字列を引数 text で指定し，side に文字列を書き込む余白位置を表す番号 (1 : 下，2 : 左，3 : 上，4 : 右) を指定する．ここで line には図形領域から何行離すかを指定し，at には文字列を書き込む座標を指定することも出来る．

種類	関数	機能
枠	box()	軸や目盛を描かず枠だけ描く．例えば box(lty='1373', col = 'red') として色や線の太さを指定することも出来る．(この場合は図の枠に赤い模様が付く).
題名	title(main, sub)	引数 main に上部余白に描かれるメインタイトルを，引数 sub に下部余白に描かれるサブタイトルを指定して，図の上下の余白部分にタイトルを追加する．省略すると，それぞれメインタイトルとサブタイトルが描かれなくなる．また，引数 tmag でテキストの拡大率を指定することが出来る．
軸	axis(side=4, labels=F)	座標を描く (1 : 下，2 : 左，3 : 上，4 : 右)．引数 labels に FALSE を指定すると目盛のラベルは描かれなくなる．また，引数 at=1:10 で目盛のラベルを指定することが出来る．
軸	axis(side=1, pos=0)	引数 pos で軸を描く位置を指定することが出来る．上 (side=3) か下 (side=1) に軸を描く場合は y 座標を，右 (side=4) か 左 (side=2) に軸を描く場合は x 座標を pos に与えればよい．例えば pos = 0 とすれば原点を通る座標軸を描くことが出来る．
凡例	legend(x, y, legend)	座標 (x，y) に凡例を追加する．引数 legend に文字ベクトルを指定することにより複数行に渡る凡例を，引数 ncol に 2 以上の数値を指定することで複数列に渡る凡例を作ることが出来る．
凡例	legend(locator(1), legend=文字)	関数 locator() により対話的に凡例の位置を決定する．
凡例	legend(..., fill=v , col=v)	箱を塗りつぶす色，点や線を描く色を指定する．
凡例	legend(..., lty=v , lwd=v , pch=v)	線の種類と線の幅，プロット用の文字 (文字ベクトル) を指定する．
図形	polygon(x,y)	(x，y) に多角形の頂点の座標ベクトル（または x，y 成分を持つリストや行列）を指定して多角形を描いて中を塗りつぶす．要素に NA があると，多角形の生成は終了する．
図形	polygon(x, y, density=c(10, 20), angle=c(−45, 45))	引数 density により，ビット/インチで太さを指定したラインを使って多角形の内部に影を入れることが出来る．このとき，引数 angle で角度 (左回りに) を与えて線の傾斜を指定することが出来る．引数は border や col , lty や xpd を指定することが出来る．

2.2　使用例

(例 1) まず，散布図を描いた後，まず関数 axis() で 両軸を描き，次に関数 box() で枠を描く．さらに関数 legend() で凡例を描いた後，最後に関数 polygon() で多角形を描いている．

```
> plot(rnorm(50), rnorm(50), xlim=c(-3,6), ylim=c(-3,3), axes = F, ann=F)
> axis(1, pos = 0, at = -3:6, adj = 0, col = 2)       # 赤で X 軸を描く
> axis(2, pos = 0, at = -3:6, adj = 1, las = 2)       # 黒で Y 軸を描く
> box()
> legend(2, 3, paste("sin(",6:9,"x)"), col=6:9,
+        pch=3, ncol=2, cex=1.1, pt.bg="pink")        # (5,9) に凡例を描画
> polygon(3:6, c(-2,-1,-2,-1), density=c(10, 20), angle=c(-45, 45))
> arrows(5, -1, 4, 2, col="blue")                     # (5,-1) から (4,2) に矢印を描く
```

(例 2) 関数 polygon() を用いることで，グラフの一部に影を付けることが出来る[*1]．まず，標準正規分布の密度関数のグラフを $-4 \leq x \leq 4$ の範囲でプロットする．そのうち $2 \leq x \leq 4$ の範囲に灰色の影を付けるには，以下の様に多数の多角形に分割して関数 polygon() で塗りつぶしをすればよい．具体的には xvals，rep(0,10) の組合せで x 軸上の辺を結び，rev(xvals)，rev(dvals) の組合せでグラフに沿った辺を結べばよい．

```
> plot(dnorm, -4, 4)
> xvals <- seq(2, 4, length=10)          # 領域を x 軸方向に 10 個の多角形（台形）に等分割
> dvals <- dnorm(xvals)                  # 対応するグラフの高さ
> polygon(c(xvals,rev(xvals)),
+ c(rep(0,10),rev(dvals)),col="gray") # 塗りつぶす
> title("Title")                         # タイトルを描く
> mtext("sub-title", side=4)             # 文字を描く
```

以下に (1) と (2) の出力結果を載せる．

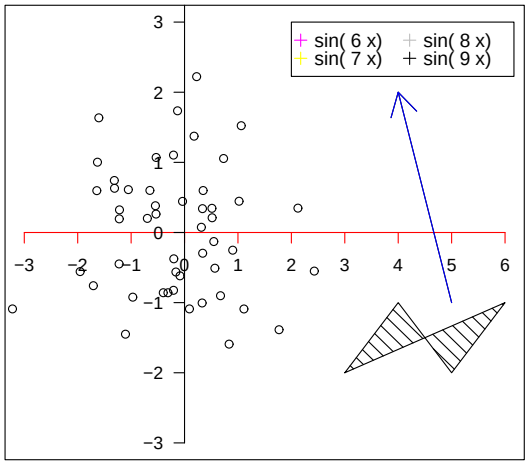


図 2.1　例 (1)

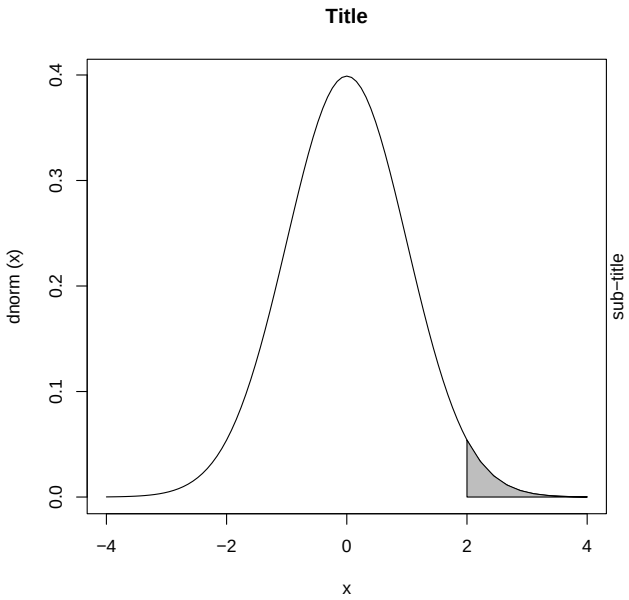


図 2.2　例 (2)

2.3　数式の描画

プロットに数学記号や式を使いたい場合で text()，mtext()，axis()，title() のいずれかを使う場合は，文字列の代わりに expression() を指定すればよい．例えば以下は二項分布の確率関数の公式を書く．

```
> text(x,y,expression(paste(bggroup("(",atop(n,x),")"),p^x, q^{n-x})))
```

一覧はオンラインマニュアルで御覧頂ける．

```
> ?plotmath
> demo(plotmath)
> example(plotmath)
```

[*1] RjpWiki の記事より引用した．

第 3 章

グラフィックスパラメータ

高水準作図関数や低水準作図関数で作図する場合，作図関数固有のパラメータ以外にもグラフィックスパラメータと呼ばれるパラメータを指定することが出来る．これにより作図結果の微妙なカスタマイズを行うことが出来，自分好みの出力結果を得ることが出来る．

3.1　グラフィックスパラメータ事始

グラフィックスパラメータを設定する方法は，作図関数の引数にパラメータを与える方法と，関数 par() を使って設定する方法の 2 通りがある．前者は一時的にパラメータ値が変更され，後者は永続的にパラメータ値が変更される．重要なことは，グラフィックスパラメータの全てがこの 2 通りの方法で変更出来るわけではなく，一部のグラフィックスパラメータは関数 par() を使ってしかパラメータ値を変更することが出来ない点を理解することである[*1]．例えば，色に関するグラフィックスパラメータ col を赤に設定する方法を挙げる．

```
> par(col="red")            # 永続的に（これ以後は）赤色でプロットする
> plot(1:10, col="red")     # 一時的に（このプロットのみ）赤色でプロットする
```

3.2　グラフィックスパラメータの永続的変更

まず，関数 par() を使わなければ変更できないグラフィックスパラメータを紹介する．この種のパラメータは領域，余白，座標系などを設定する機能であるものがほとんどであり，一旦変更すると，改めて変更するまでは元に戻らない．

引数	機能
par()	全てのグラフィックスパラメータの現在値がリストとして得られる．
par(”文字列”)	グラフィックスパラメータ名を文字列で指定すると，そのグラフィックスパラメータの現在値が得られる．グラフィックスパラメータの値を変更する際は，まずこの命令でパラメータ値を確認する．
par(”文字列”, ”文字列”) par(c(”文字列”, ”文字列”))	グラフィックスパラメータを複数指定することも出来る．
par(col=2)	「パラメータ名 = 値」の形で設定することでグラフィックスパラメータの値を変更する．
par(col=2, lty=3) par(list(col=2, lty=3))	複数のグラフィックスパラメータを一度に変更することも出来る．

[*1] 一部のグラフィックスパラメータには読み取り専用・変更不可なものがあるが，ここでは詳しく扱わないことにする．例えば，par(”cin”)，par(”cra”)，par(csi)，par(”cxy”)，par(”din”) でそれぞれ「インチ単位の既定文字サイズ（幅と高さ）」「ピクセル単位の既定文字サイズ（幅と高さ）」「インチ単位で与えた既定サイズの文字の高さ」「既定文字のサイズ（幅と高さ：par(”cin”)/par(”pin”)）」「インチ単位のデバイスの大きさ（幅と高さ）」を確認することが出来るが，これらの値を明示的に変更することはできない．

3.2.1　関数 par() の使い方

par() の使い方を以下に挙げる[*2].

```
> par(new=T)          # new を TRUE に設定すると，現在の作図に次の作図を上書きする
> par(ask=TRUE)       # 作図する前に『作図してよいですか？』と質問させるようにする
> help(par)           # 全てのグラフィックスパラメータのヘルプが表示される
```

3.2.2　グラフィックスパラメータ値の一時退避と復帰

関数 par() を使わなければ変更できないグラフィックスパラメータの値を変更する場合は，適当な変数に現在のパラメータの値を保存しておくのが得策である．R では便利なことに，グラフィックスパラメータの値を 1 コマンドで一切合財保存することが出来る．

```
> oldpar <- par(no.readonly = TRUE)     # 現在のグラフィックスパラメータ値を退避する
> oldpar <- par(col=1, lty=2)           # 一部だけ保存する
> par(oldpar)                           # 作業前のグラフィックスパラメータ値に戻す
```

関数定義内で，関数 par() を使ってグラフィックスパラメータを変更する場合も，元のグラフィックスパラメータの値を一切合財保存した上で作業をし，関数の最後に元に戻すことをお勧めする．

```
> myfunc <- function () {
+   oldpar <- par(no.readonly = TRUE)   # 現在のグラフィックスパラメータの値を退避
+   on.exit(par(oldpar))                # 関数がエラー中断してもパラメータを復帰
+   par(col=2);  plot(1:10)             # 色を赤に変更してからプロットを行う
+ }                                     # 関数を抜けた後にパラメータ値は元に戻る
```

3.2.3　作図領域・余白・座標系

関数 par() を使わなければ変更できないグラフィックスパラメータは，大抵がプロット領域（関数 plot() で散布図を描いた場合：四角の枠とその中に描かれた点や線などがある領域）と余白（プロット領域の周囲，すなわち枠の外側の目盛や目盛のラベル，軸のラベルやタイトルなどが描かれる部分），作図領域（プロット領域 + 余白）及びデバイス領域に関するパラメータとなっている．以下に概念図を示す．

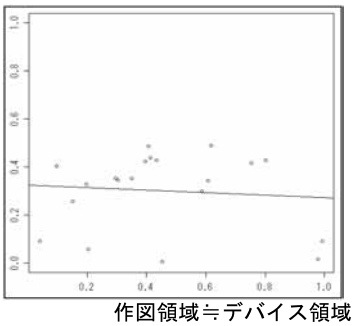


図 3.1　デバイスを分割していない場合

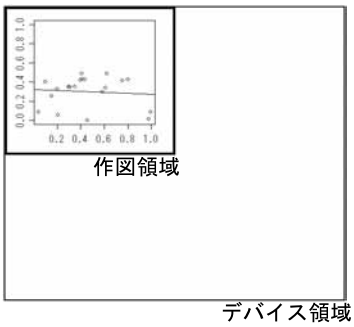
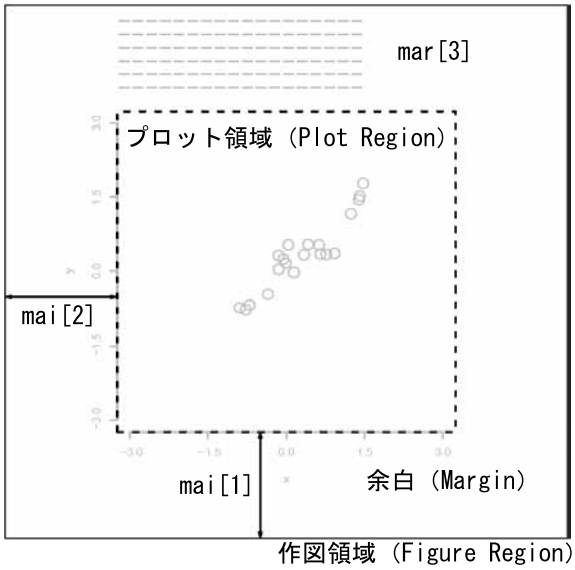


図 3.2　デバイスを分割している場合

[*2] パラメータ new を TRUE にしておくと次に描く図は前の図に上書きして描かれ，パラメータ ask を TRUE にしておくと help の example を閲覧する際に作図例が一瞬で流れてしまうのを防ぐことが出来る．

デバイス領域とは，グラフを描いたときに表示されるウインドウの領域のこと[*3]で，普段はデバイス領域と作図領域はほぼ一致しているが，デバイスを分割している場合に違いが出てくる．次に，作図領域とプロット領域の違いを示す．



3.2.4　作図範囲に関するパラメータ

xpd = F に FALSE を指定するとプロット領域内に収まる部分だけ図が描かれ，TRUE を指定すると作図領域内に収まる部分だけ図が描かれる．NA を指定するとデバイス領域全体に図が描かれる．以下に例を示す．

```
> library(rpart)
> result <- rpart(Species ~ ., data=iris)   # 直前で par(xpd=F) 又は par(xpd=NA) を実行する
> plot(result)
> text(result)
```

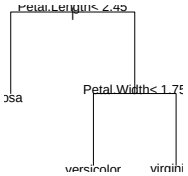


図 3.3　par(xpd=F) としている場合

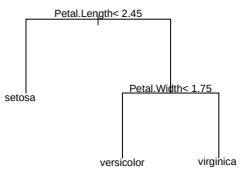


図 3.4　par(xpd=NA) としている場合

3.2.5　プロット領域（plot region）の大きさや位置を指定するパラメータ

引数	機能
pin = c(6,6)	プロット領域の幅と高さをインチ単位で指定する．
plt = c(0.1, 0.9, 0.1, 0.9)	プロット領域の x，y 軸方向の両端の位置の比率を指定する．c(0, 1, 0, 1) と変更すれば余白が全くない出力が得られる．
ps = 12	文字とシンボルの大きさを整数値で指定する．
pty = ”m”	”s” は正方形のプロット領域，”m” は最大のプロット領域を生成する．”m” を設定した場合，プロット領域でも余白でもない空白部分が出来る場合もある．

[*3] デバイスが ps, eps ならば，デバイス領域は用紙の大きさ（用紙の外周）となる．

ちなみに，par(”usr”) でプロット領域を知ることが出来る．

```
> par("usr")
```

3.2.6　作図領域（figure region）の大きさや位置を指定するパラメータ

引数	機能
fig = c(0, 1, 0, 1)	作図領域の x，y 軸方向の両端の位置の比率を指定する．初期値は c(0, 1, 0, 1) で，これは作図領域をフルに使用していることを示しており，c(0, 0.5, 0.5, 1) と変更すれば左上の方に作図されることになる．この値を設定した後，現在のプロットにグラフを描き加えるときは明示的に par(new=T) とする必要がある（この点は S と異なる）．
fin = c(5,5)	作図領域の幅と高さを指定する．例えば par(”fin”) で調べた結果が [6.968749 6.958332] であった場合に c(5, 5) を指定すれば，描画結果は少し小さめの出力となる．この値を設定した後，現在のプロットにグラフを描き加えるときは明示的に par(new=T) とする必要がある（この点は S と異なる）

3.2.7　余白に関するパラメータ

余白をあまり小さくしてしてしまうと軸のラベルなどが描けなくなってしまうので注意．

引数	機能
mai = c(0.85, 0.68, 0.68, 0.35)	底辺，左側，上側，右側の順に余白の大きさをインチ単位で指定する．mar で余白サイズを変えると plt，pin の値が変更される．
mex = 1	プロットの余白の座標を指定するのに使われる文字サイズの拡大率を指定する．このパラメータを変更すると，余白の大きさもそれに応じて変化する．複数図表を使う場合は，このパラメータ値が自動的に 0.5 に変更されることがあるので注意が要る．
mar = c(5, 4, 4, 2)	底辺，左側，上側，右側の順に余白の大きさを行の高さ（mex）で指定する．デフォルトでは高水準作図関数は軸のラベルを 3 行分離して描くので，底辺と左部には最低でも 4 行分 (4 mex 分) の余白が必要となる．mai で余白サイズを変えると plt，pin の値が変更される．
omi = c(0, 0, 0.8, 0)	底辺, 左側, 上側, 右側の順に外周の大きさをインチ単位で表した (outer margins in inches) もので指定する．
oma = c(2, 0, 3, 0)	外周の底辺，左側，上側，右側の順に外側余白（outer margin）を行の高さ（mex）で指定する．
omd = c(0, 1, 0, 1)	ウインドウ全体を 0 から 1 の範囲として，外周を除いた複数図表の両端の位置 (x_1, x_2, y_1, y_2) を指定する．この場合は外周は存在しないことになる．

例えば右側に y 軸座標を描く場合，グラフを描く前に mar の値を設定して，右側の余白を余分に取る必要がある．

```
> par(mar=c(5,4,5,4))
> plot(1:10)
> axis(side=4)
```

3.3　グラフィックスパラメータの一時的変更

この節で紹介するグラフィックスパラメータは，関数 par() で設定することが出来るし，関数 plot() や多くの高水準作図関数の引数としてグラフィックスパラメータを設定することも出来る．例えばプロット点の形を指定するパラメータ pch に関しては，以下の 2 通りの設定方法を用いることが出来る．ただし，関数 par() で設定した場合は，設定した後はもう一度パラメータ値を変更するまではそのままの設定値が使われるが，作図関数の引数としてグラフィックスパラメータを設定した場合は，そのときの作図の場合のみ設定値が使われる (それ以後は直前までの設定値が使われる) .

```
> plot(1:10, pch="+")      # 一時的にプロット点の形を "+" に変更する
> par(pch="+")             # これ以後，プロット点の形をずっと "+" に変更する
> plot(1:10)               # プロット点の形は "+" となる
> plot(1:20)               # これもプロット点の形は "+" となる
```

3.3.1　プロットに用いられる色

col と同様の命令で col.axis, col.lab, col.main, col.sub でそれぞれ軸，ラベル，タイトル，サブタイトルの色を指定することが出来る．

引数	機能
col = 1, col = blue	作図に用いる色 (color) を引数 col = 番号または色名で指定する (初期値は 1) ．番号は 1 から順に「黒，赤，緑，青，水色，紫，黄，灰」となっている．
col = rgb(1, 0, 0), col = '#FFFFFF'	引数 col = rgb(赤，緑，青) で色を指定したり，16 進数で色を指定することも出来る．
col = rainbow(10)	以下の表の他に，rainbow(n) ，heat.colors(n) ，terrain.colors(n) ，topo.colors(n) ，cm.colors(n) を col に指定して色を決めることも出来る．
col = hsv(.5, .5, .5), col = gray(0.8)	それぞれ hsv 形式，グレースケールで色を決めることが出来る．
gamma = 1.0	ガンマ補正を行う．

3.3.2　プロットのマーカー

pch = 値 ，**pch = "文字"** で，点をプロットするときに用いるプロット文字を指定する．値は 0 から 25 の整数，文字はピリオドや "+" などで指定する．

pch	出力	pch	出力	pch	出力	pch	出力	pch	出力	pch	出力	pch	出力
0	□	1	○	2	△	3	+	4	×	5	◇	6	▽
7	⊠	8	✳	9	◈	10	⊕	11	✡	12	⊞	13	⊗
14	☑	15	■	16	●	17	▲	18	◆	19	●	20	•
21	○	22	□	23	◇	24	△						

3.3.3　テキスト・フォントに関するパラメータ

文字列に関するグラフィックスパラメータには以下のようなものがある．以下の引数によって文字の大きさや描画方向などを設定することが出来る．

引数	機能
adj = 0	テキスト文字列の調節 (0 : 左揃え，0.5 : 中心揃え，1 : 右揃え) を行う．また，adj = c(x,y) で x 軸および y 軸方向を別々に揃えることも出来る．
cex = 1	標準の大きさを 1 として，文字の拡大率を指定する．csi を変更すると，このグラフィックスパラメータもその値に応じて自動的に変化する．同様の命令で cex.axis, cex.lab, cex.main, cex.sub でそれぞれ軸，ラベル，タイトル，サブタイトルの拡大率を指定する．
ps = 20	テキストと記号の大きさをポイント単位で指定する．
text(x, y, labels=”文字列”, srt=90)	座標 (x, y) に labels で指定した文字列を表示する際に，文字の回転角を (単位は度で) 指定する (x 軸を基準とする) ．また，引数 crt では文字の回転角を (単位は度で) 指定する．
col = 1	色を指定する．同様の命令で col.axis, col.lab, col.main, col.sub でそれぞれ軸，ラベル，タイトル，サブタイトルの色を指定する．
font = 1	フォント番号を指定する (1 : プレイン，2 : ボールド，3 : イタリック，4 : ボールドイタリック，5 : シンボル文字 (Adobe symbol encoding)) ．同様の命令で font.axis, font.lab, font.main, font.sub でそれぞれ軸，ラベル，タイトル，サブタイトルのフォント番号を指定する．
family=””	フォント・ファミリーを指定する．デフォルトは ”” で，デバイスのデフォルト値になるよう設定されている．他には ”serif”，”sans”，”mono”，”symbol” が指定できる（デバイスの種類によっては無視される)．

3.3.4　線分の太さと形式

以下のパラメータが指定できる[*4].

引数	機能
lwd = 1	線分の幅 (line width) を番号で指定する．値が大きいほど太い線になる．
lty=0 ，lty=”blank”	線分の形式 (line type) を無し（透明の線）にする．
lty=1 ，lty=”solid”	線分の形式 (line type) を実線にする．
lty=2 ，lty=”dashed”	線分の形式 (line type) をダッシュにする．
lty=3 ，lty=”dotted”	線分の形式 (line type) をドットにする．
lty=4 ，lty=”dotdash”	線分の形式 (line type) をドットとダッシュにする．
lty=5 ，lty=”longdash”	線分の形式 (line type) を長いダッシュにする．
lty=6 ，lty=”twodash”	線分の形式 (line type) を二つのダッシュにする．
type = ”p”	点プロット (デフォルト) を行う．
type = ”l”	線プロット (折れ線グラフ) を行う．
type = ”b”	点と線のプロットを行う．
type = ”c”	”b” において点を描かないプロットを行う．
type = ”o”	点プロットと線プロットの重ね書きを行う．
type = ”h”	各点から x 軸までの垂線プロットを行う．
type = ”s”	左側の値にもとづいて階段状に結ぶ．
type = ”S”	右側の値にもとづいて階段状に結ぶ．
type = ”n”	軸だけ描いてプロットしない（続けて低水準関数で作図する場合)．

[*4] 他にも ’lend’（線分の終端 ; 0 : ”round”，1 : ”butt”，2 : ”square”)，’lheight’，’ljoin’（線分の結合形式 ; 0 : ”round”，1 : ”mitre”，2 : ”bevel”)，’lmitre’ が指定できる．

3.3.5　枠に関するパラメータ

引数	機能
bg = ”blue”	背景の色を指定する.
fg = ”blue”	前面の色を指定する.
bty = ”o”	枠が箱型，四方が囲まれている形になる.
bty = ”l”	枠が L 字型，左と下部だけに枠の線が引かれる.
bty = ”7”	枠が 7 型，上部と右だけに枠の線が引かれる.
bty = ”c”	枠が C 字型，右部を除き枠の線が引かれる.
bty = ”u”	枠が u 字型，上部を除き枠の線が引かれる.
bty = ”]”	枠が] 字型，左部を除き枠の線が引かれる.
bty = ”n”	枠を描かない (箱を描く) .

3.3.6　軸に関するパラメータ

引数	機能
ann = F	FALSE を指定すると，軸と全体のタイトルを描かなくなる.
las = 0	ラベルを各軸に並行して描く.
las = 1	ラベルをすべて水平に描く.
las = 2	ラベルを軸に対して垂直に描く.
las = 3	ラベルをすべて垂直に描く.
lab = c(5,5,7)	長さ 3 の数値ベクトルの最初の二つで x , y 軸につける目盛の数を指定する．最後の一つはラベルのサイズを指定するはずだが，今のところ (R 2.1.1) は移植されていない.
mgp = c(3,1,0)	長さ 3 の数値ベクトルでそれぞれ軸タイトル，軸ラベル，軸線が描かれる位置を枠から何行分 (mex 単位) 外側にするかを指定する.
tck = NA	軸の目盛線の長さを枠の大きさに対する割合で指定し，値が正ならば線が図の内側に，値が負ならば図の外側に目盛線が描かれる．ここで tck = 1 とすれば，図に格子を入れることが出来る．初期値は NA だが，これは tcl = −0.5 を用いるという意味である.
tcl = −0.5	軸の目盛線の長さをテキスト行の高さ (mex) の割合で指定する．正の値を指定すれば目盛を内側に向かって描き，負の値なら外側に描く．もし tcl = NA とすれば，tck に −0.01 がセットされる.
xaxt = ”s” , yaxt = ”s”	軸を描くか否かを指定する．この場合は通常の軸が使われていることを示す（”l” や ”t” を指定しても同じ働き).
xaxt = ”n” , yaxt = ”n”	軸を描くか否かを指定する．この場合は軸を描かない．xaxt = ”n” と yaxt = ”n” を同時に指定すると axes = F と同じになる.
xaxs = ”r” , yaxs = ”r”	x , y 軸のスタイルを指定する．まずデータ範囲を両側 4 % 広げ，次に範囲内でラベルがきれいに表示できる軸を見つける.
xaxs = ”i” , yaxs = ”i”	x , y 軸のスタイルを指定する．データ範囲内できれいに表示できる軸を見つける.
par(xaxp) , par(yaxp)	x, y 軸の目盛りの区切りを参照する．最初の二つが両端の目盛りの座標，最後の値がその内側の区切りの個数を表す．値自体を変更してもあまり意味が無い.
xlog = F , ylog = F	それぞれ対数 x 軸，対数 y 軸の使用を論理値で指定する．もし TRUE ならば対数軸が使われる．新しいデバイスに対しては既定値は FALSE に設定される.

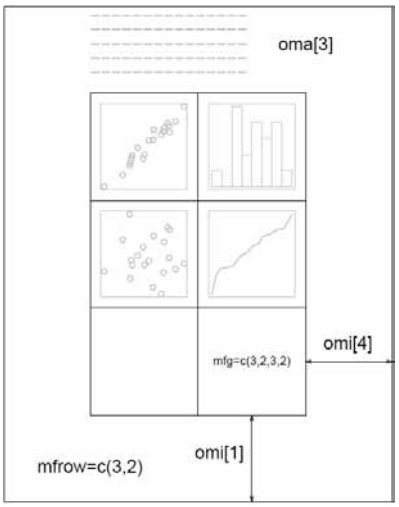
第 4 章

画面の分割方法，重ねた図を描く方法

以下に出てくるパラメータの説明はグラフィックスパラメータと割り付けパラメータの節で扱っている．

4.1　画面（デバイス）を分割して図を描く

まず，画面（デバイス）の外周の余白を指定するパラメータがどこにあるかを紹介する．



タイトルや注釈テキストを入れる場合で，余白を指定する場合は par() の引数に以下のパラメータを入れて指定する．タイトルや注釈テキストを入れない場合は指定する必要は無い．ここで紹介する例では，全体のタイトルを描くため，まず oma で上部に 4 mex 分（mex の初期値は 1 なので 4 文字分）の余白を取っている．

```
> par(oma = c(0, 0, 4, 0))    # 下・左・上・右の順で余白を設定
```

画面（デバイス）を分割するパラメータは以下の様なものがある．

引数	機能
mfcol = c(m,n), mfrow = c(m,n)	画面を m 行 n 列に分割する．mfcol で指定した場合は列順に，mfrow で指定した場合は行順にグラフが描かれる．1 画面に戻す場合は mfrow=c(1,1) または mfcol=c(1,1) とすればよい．
mfg = c(i,j,m,n)	画面を m 行 n 列に分割している場合で，左上から順番に図を描きたくない場合に使う．次のグラフを i 行 j 列にを描く．
fig = c(4,9,1,4)/10	現在の画面における現在の図表の位置を指定するパラメータで，ページ内の任意の位置に図表をおくためのものである．値は百分率を指定し，「左側・右側・底辺・上側」の順に指定する．例の値はページの右下に図を置くように指定している．

ここでは画面を 2×2 に分割する．

```
> par(mfrow=c(2,2))
```

画面が分割できたら，グラフを 1 つずつ描けば，複数のグラフが（行順に）一度に表示される出力が得られる．

```
> plot(sin)
> plot(cos)
> plot(asin)
> plot(acos)
```

最後に outer = T と指定してから mtext を使うことで全体のタイトルを外周に書き込む．

```
> mtext(side = 3, line=1, outer=T, text = "Title", cex=2)
```

すると以下の左図のようになる．ここで最初の余白 (oma) を設定していなければ右図のようにタイトルがはみ出すことになる．

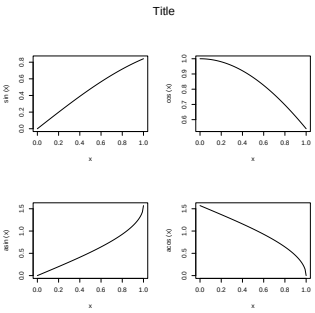


図 4.1　成功例

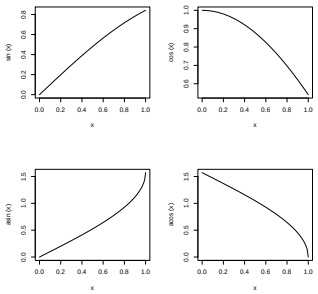


図 4.2　失敗例

(注意 1) 関数 frame() で図を消去すると図が消去されるが，このとき，図表番号の順番も 1 つ後に進んでいることに注意したい．

(注意 2) stars や pairs など，内部で複数図表を用いている高水準作図関数は，複数図表と同時に使うことは出来ない．

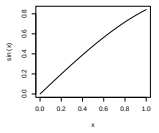
4.2　少し高度な画面（デバイス）分割

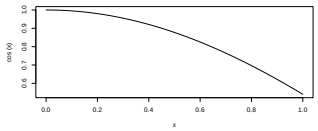
上で紹介したグラフィックスパラメータ mfrow や mfcol を用いることで画面を複数に分割することが出来るのだが，以下で紹介する関数を用いることで規則的な画面分割に限らず，自由に画面を分割することが出来るようになる．

4.2.1　layout() を用いた画面分割

関数 layout() を用いると，行列 mat で行数と列数を指定して画面を分割することが出来る．このとき画面は『行列の行数 × 行列の列数』に分割され，行列の成分が作図の順番となる．

```
> mat <- matrix(c(1,0,2,2), 2, 2, byrow = TRUE)
> mat
     [,1] [,2]           # このとき画面（デバイス）は
[1,]    1    0           #（ 1 番目の図）　（ここは空白のまま）
[2,]    2    2           #（ ---------- 2 番目の図 ---------- ）
> layout(mat)            # と分割される
> plot(sin)
> plot(cos)
```





layout.show(n) でデバイス番号を確認することが出来，また lcm(x) で長さを指定することも出来る．

```
> layout(matrix(c(1,3,2,2), 2, 2, byrow = TRUE), respect=T, widths=lcm(5), heights=lcm(5))
> plot(sin)
> plot(cos)
> plot(acos)
```

4.2.2　split.screen() を用いた画面分割

関数 split.screen() に長さ 2 のベクトル c(m, n) を引数に与えることで画面を分割することも出来る[*1]．分割された画面には番号が振られ，その番号（画面）を指定して図を描くことができる．例えば，画面を上下 2 つに分割する場合は以下のようにする．

```
> split.screen(c(2,1))
[1] 1 2
```

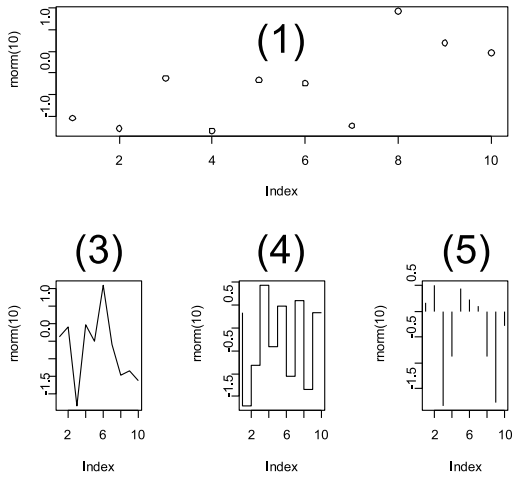
この場合は上側が画面番号 1 ，下側が画面番号 2 になる．ここで引数 screen に分割する画面番号を指定することによって，その画面をさらに分割することも出来る．例えば上の例で出来た画面 2 をさらに 3 つに分割してグラフを描く．

```
> split.screen(c(1,3), screen = 2)
[1] 3 4 5
> screen(1); plot(rnorm(10))              # 画面 1 にプロット
```

[*1] 行列を引数にしても画面分割を行うことも出来る

```
> screen(3); plot(rnorm(10), type="l")   # 画面 3 にプロット
> screen(4); plot(rnorm(10), type="S")   # 画面 4 にプロット
> screen(5); plot(rnorm(10), type="h")   # 画面 5 にプロット
```

以下の出力結果で括弧つきの番号は画面番号を表す (実際の出力には表示されない)．



画面消去をする場合は関数 frame() で消去することが出来る．関数 erase.screen() は図を消去するのではなく，背景色で塗りつぶすだけなのでメモリを食うので使用は控えた方が良い．また，通常の 1 画面形式に戻る場合は，関数 close.screen(all=T) を実行すればよい．

4.3　重ねた図を描く

単純な作図ならば，引数に add=T を入れることで重ねた図を描くことができる．2 つのグラフの x 軸と y 軸の座標は自動的に合わせられる．

```
> plot(cos, -pi, pi, lty=2)
> curve(sin, add=T)
```

しかし，高水準作図関数の中には add=T を引数にもってくることが出来ないものがある．そこで，パラメータ new を指定することで，既存のグラフに新たなグラフを上書きするように指定することが出来る．例えば，以下のようにすることで 2 つのプロットを重ね合わせることが出来る．

(注意 1) par(new=T) で重ね描きする場合，そのままでは軸と軸のラベルが重ね描きされる．2 回目の plot() によって軸と軸のラベルが重ね書きされるのを避けるため，1 回目のプロット時に axes=F , xlab=””，ylab=””（もしくは ann=T ）を指定するのが得策である．

(注意 2) par(new=T) で重ね描きする場合，普通はそれぞれのグラフの座標範囲が異なるため，仕上がりがおかしくなる．重ね描きする全てのグラフが同じ座標範囲となるよう，全ての作図関数で xlim = c(x0, x1)，ylim = c(y0, y1) を指定すること．

```
x <- rnorm(100)                                          # 正規乱数を 100 個生成して
hist(x, xlim=c(-4,4), ylim=c(0,0.5), prob=T, ann=F)      # ヒストグラムを描く
par(new=T)                                               # 上書き可にしてから
plot(density(x), xlim=c(-4,4), ylim=c(0,0.5),
     xlab="", ylab="", main="", col="red")               # 密度推定の曲線を上書きする
```

この後，axis() で 2 回目のグラフ（plot(density(x)...)）が描くはずだった y 軸を図の右側の余白に描くことが出来る．

```
> axis(side=4)
```

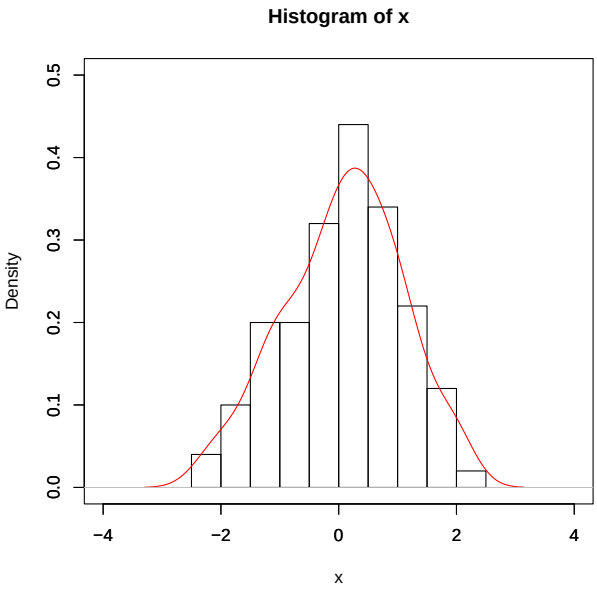


図 4.3　axis() 指定前

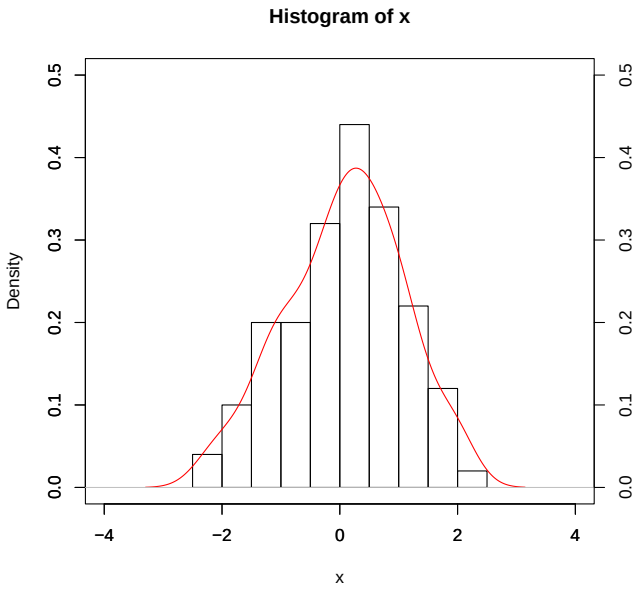


図 4.4　axis() 指定後

(注意) axis() で右側に y 軸を追記する場合，パラメータ mar で右側の余白を空けておいた方が良い．余白を空けないと不具合が生じることがある．

参考文献・参考サイト

- 『S によるデータ解析』渋谷 政昭，柴田里程 著 (共立出版)
- isac S のリファレンスマニュアル：*http://s.isac.co.jp/*
- 『An Introduction to R -Notes on R-』(Version 2.2.0; 2005-10-06)
- RjpWiki (岡田先生) ：*http://www.okada.jp.org/RWiki/index.php?RjpWiki*
- 『工学のためのデータサイエンス入門』 間瀬 茂・神保 雅一・鎌倉 稔成・金藤 浩司 著 (数理工学社)