R と RStudio の使い方入門
グラフの作成(1)
標準パッケージ graphics

標準関数によるグラフ作成

（Windows 11 の PC を使用）

R と RStudio の使い方入門－ 1

1　R におけるグラフ作成環境
2　標準パッケージ graphics の関数
3　関数 plot() の使い方
4　関数 plot() 以外の関数の使い方
5　標準関数によるグラフ作成

★本 PDF の内容については、
自己責任で利用してください
★PowerPoint のスピーカーノートを
PDF の注釈に変換してあります
PDF を**ダウンロード**して**最新の**
Adobe Acrobat Readerで利用して下さい

はじめに

- R スクリプトとコード

  R スクリプト

  複数の R 関数、演算子、制御文などを
  組み合わせたコードの集まり

  シンプルな散布図を作成：plot(x, y)

  データの読み込み、加工、可視化、統計解析等
  特定の目的を達成する

  R スクリプトを作成してデータ解析を実行

  スクリプトは１つのコードで済む場合もあれば、
  目的によってては数十行にも及ぶ
  必ずしも白紙の状態から作り上げるわけではない
  再利用が基本

３つの層別散布図と確率楕円を表示する R スクリプト

```
90  ## (d) iris：確率楕円を表示した層別散布図
91  species <- levels(iris$Species)      # 品種の名前
92  colors <- c("red", "green", "blue") # 品種の色
93  names(colors) <- species
94
95  plot(iris[, c("Sepal.Length", "Sepal.Width")],
96       type = "n",   # 散布図の枠のみ描画
97       xlab = "がく片の幅", ylab = "がく片の長さ",
98       las = 1)
99
100 for (s in species) {
101   df <- subset(iris, Species == s)  # 品種抽出
102
103   points(df$Sepal.Length, df$Sepal.Width,
104          col = colors[s], pch = 16) # プロット
105
106   dataEllipse(                          # 確率楕円
107     df$Sepal.Length, df$Sepal.Width,
108     levels = 0.5,       # 50% 確率楕円
109     add = TRUE,         # 重ね書き
110     col = colors[s],    # 品種ごとの色
111     fill = TRUE,        # 楕円の塗りつぶし
112     fill.alpha = 0.1)   # 楕円の塗りつぶしの透明度
113 }
114
115 legend("topright", col = colors, pch = 16,
116      cex = 0.5,legend = species, title = "Species")
```

はじめに

●R スクリプトの作成と再利用

典型的な R スクリプトや過去のプロジェクトで作成した R スクリプトを保存・再利用
AI を活用して、目的を実現する R スクリプトを生成（AI 支援を前提とした学習）
↓
既存のサンプルデータで正常に動作するかを確認、R スクリプトの内容とその動作を理解
↓
目的に合わせて R スクリプトをカスタマイズ
(i) 対象データに合わせた修正 （ファイルの読込、列名・データ型の違いに対する変更など）
(ii) データ解析の目的に合わせた修正 （集計の条件変更、グラフのカスタマイズの変更など）
(iii) 複数のスクリプトを統合・連携 （例：A のスクリプトで前処理→B のスクリプトで可視化）

R のスクリプトは再利用が基本
R 言語の学習を進める上では、既存のスクリプトやAI が生成したスクリプトを把握して、
自分の目的に合わせて修正・応用する能力を身に着けることを意識して学習

はじめに

● 本セミナーの学習ポイント

1 R の標準関数によるグラフ作成の全体像を把握、「どんなグラフが描けるか」を広く理解

2 実践的な学習：ハンズオン

説明に使う全てのスクリプトとデータを提供、実際に自分の手でコードを実行して理解する

3 コードは「暗記」ではなく「理解」を

何十行にもわたるスクリプトの説明、関数の詳細な使い方などの説明もあるが、暗記は不要

「こんな機能をもつ関数がある」「この部分を指定する引数がある」という程度の理解で十分

4 該当するスクリプトを参照できること（思い出せること）が重要

グラフを作成するときに、「あのスクリプトが使えそうだ」と思い出す

該当するスクリプトを探し出して、改めてその内容を確認・理解してグラフを得る

5 今後、データ解析はAI と人 との協働作業に発展

AI とのコミュニケーション能力を身に着ける（AI への的確な指示、AI の回答を正しく判断）

そのための基礎的な知識・スキルを学習　　（入門者はAI に頼りすぎないで自分で考える）

はじめに

●本セミナーの説明範囲

R と RStudio を使って、標準関数によるグラフ作成の基本システムについて説明
データに適したグラフの選択、グラフから読み取れる内容などについては次回に説明
（関連する専門書、ネット上のサイト、AI の回答などを参照）
注）「R と RStudio の使い方－1」と「R と RStudio の使い方－2」の内容を
理解しているという前提で説明

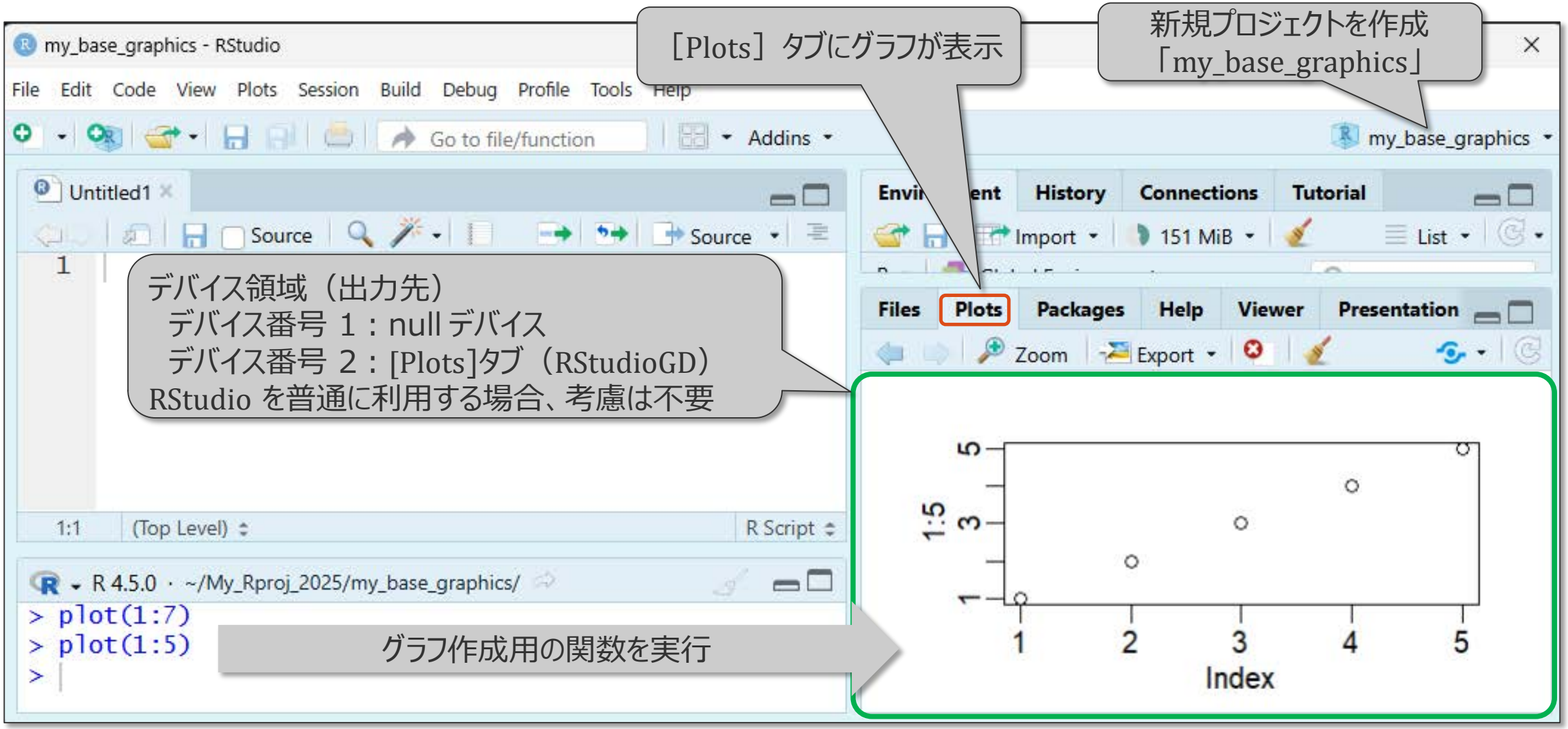
●関連ファイルのダウンロード

この PDF をダウンロードしたサイトから「Rスクリプト」をダウンロード
ダウンロードした圧縮ファイル（zip ファイル）を解凍
R スクリプトファイルを 3 つ得る
my_base_graphics1.R、my_base_graphics2.R、my_base_graphics3.R

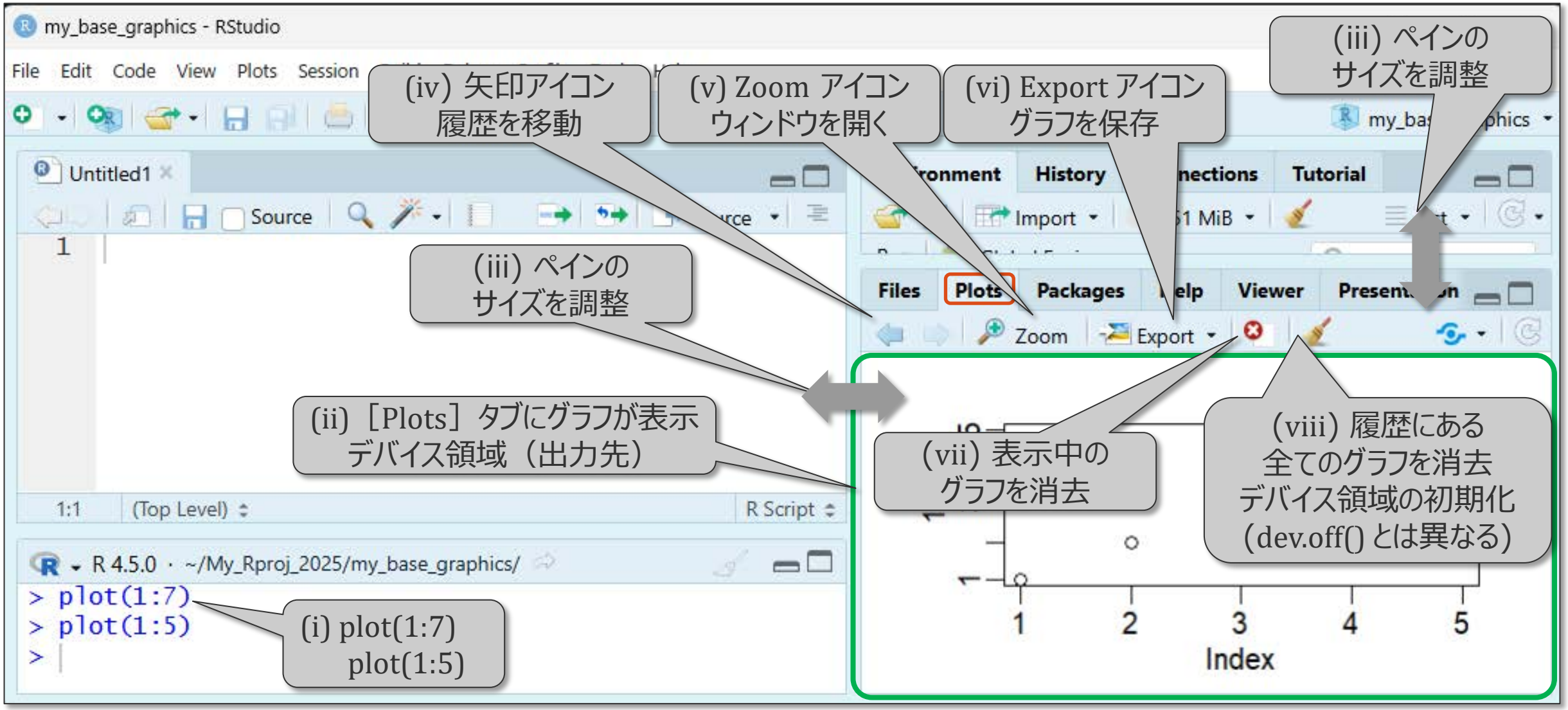
1　R と RStudio におけるグラフ作成環境

RStudio を使用したグラフ作成
R のグラフ作成用パッケージ
graphics・lattice・ggplot2

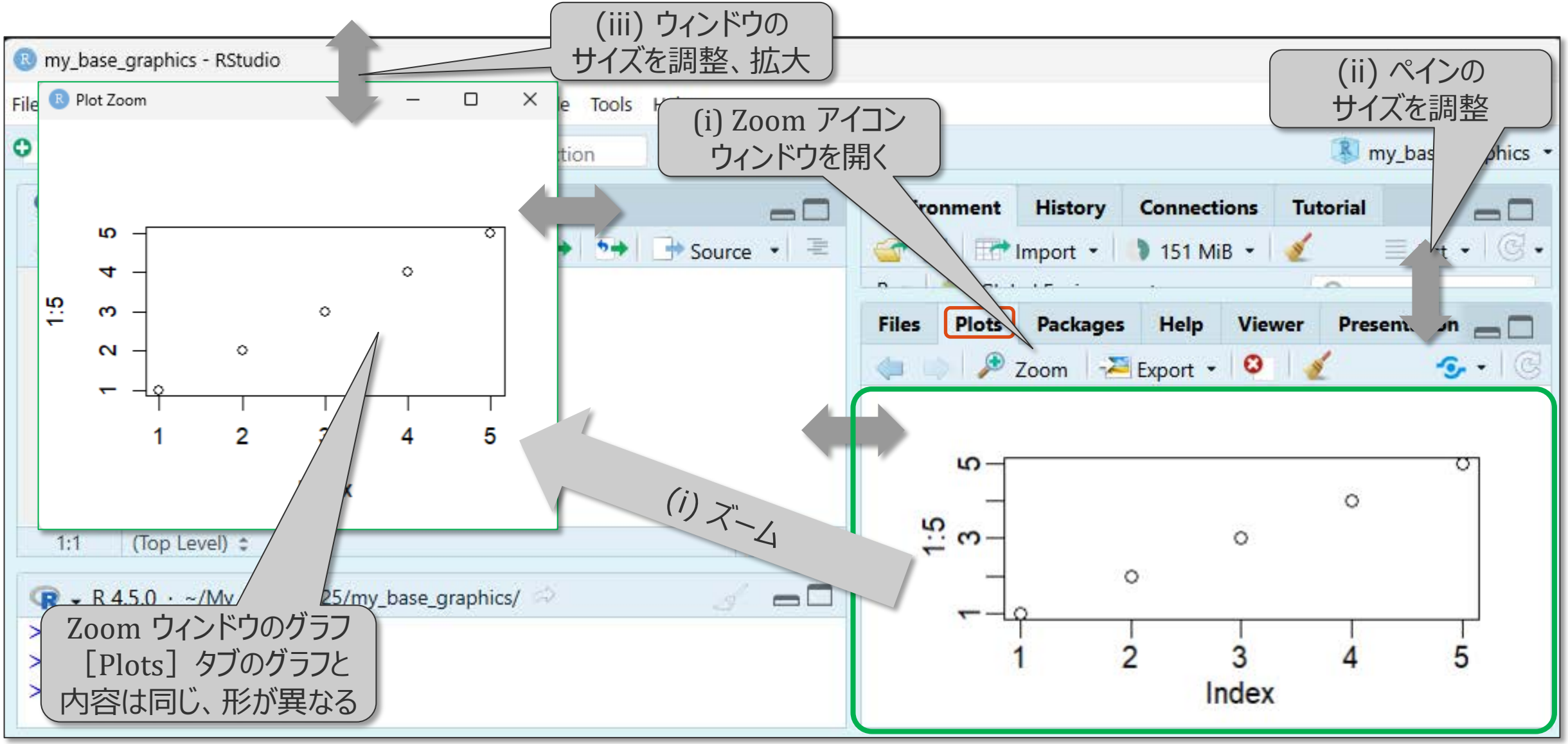
R と RStudio におけるグラフ作成環境



R と RStudio におけるグラフ作成環境



R と RStudio におけるグラフ作成環境



R と RStudio におけるグラフ作成環境

●RStudio でのグラフの表示領域

(i) グラフの表示領域 = デバイス領域

グラフは［Plots］タブの中に表示

［Plots］タブの画面サイズはそのペインのサイズで調整、小さ過ぎると描画不可（エラー）

複数のグラフを連続して作成すると直近のグラフが１つ表示、前のグラフは履歴に移動

矢印アイコン（左右の矢印）で履歴を移動して、前のグラフや後のグラフに切り替え可能

試行錯誤しながら複数のグラフを作成して改良していく際に便利

複数のグラフを同時に表示する場合、［Plots］タブの領域を格子状のグリッドに分割

パッケージごとにシステムが異なり、設定方法が異なる

(ii) ズーム機能

ズームウィンドウを開いて、グラフを拡大表示（［Zoom］アイコンによる）

ズームウィンドウは、元の［Plots］タブのサイズに影響されずに独立してサイズ変更が可能

(iii) グラフをファイルに保存（PNG、JPEG、TIFF 、PDF等 ）「R と RStudio の使い方－2」参照

R と RStudio におけるグラフ作成環境

- R の主なグラフィックパッケージ

  パッケージ base, stats の一部の関数も含む

  代表的なパッケージ：標準パッケージ graphics、推奨パッケージ lattice、外部パッケージ ggplot2

  それぞれのパッケージは異なる設計思想と設計アプローチを持つ

  graphics　R の基本的なグラフ描画機能、軽量かつシンプル（base、stats の一部の関数を含む）

  高水準と低水準の関数の組合せで作図し、短いコードで素早くグラフを作成

  小規模な解析やシンプルなグラフ作成に適すが、高度なカスタマイズには手間がかかる

  lattice　条件付きプロット（カテゴリごとのグラフ分割）に特化

  formula 記法による簡潔な記述、トレリスグラフィックスで複数のグラフを同時表示

  多変量データの視覚化、グループ別比較に適す

  ggplot2　レイヤーを積み重ねて構築する柔軟で強力なグラフ描画システム

  美しく整ったデザインと高いカスタマイズ性

  データサイエンスの現場では、標準ツールの一つとして広く使用されている

R と RStudio におけるグラフ作成環境

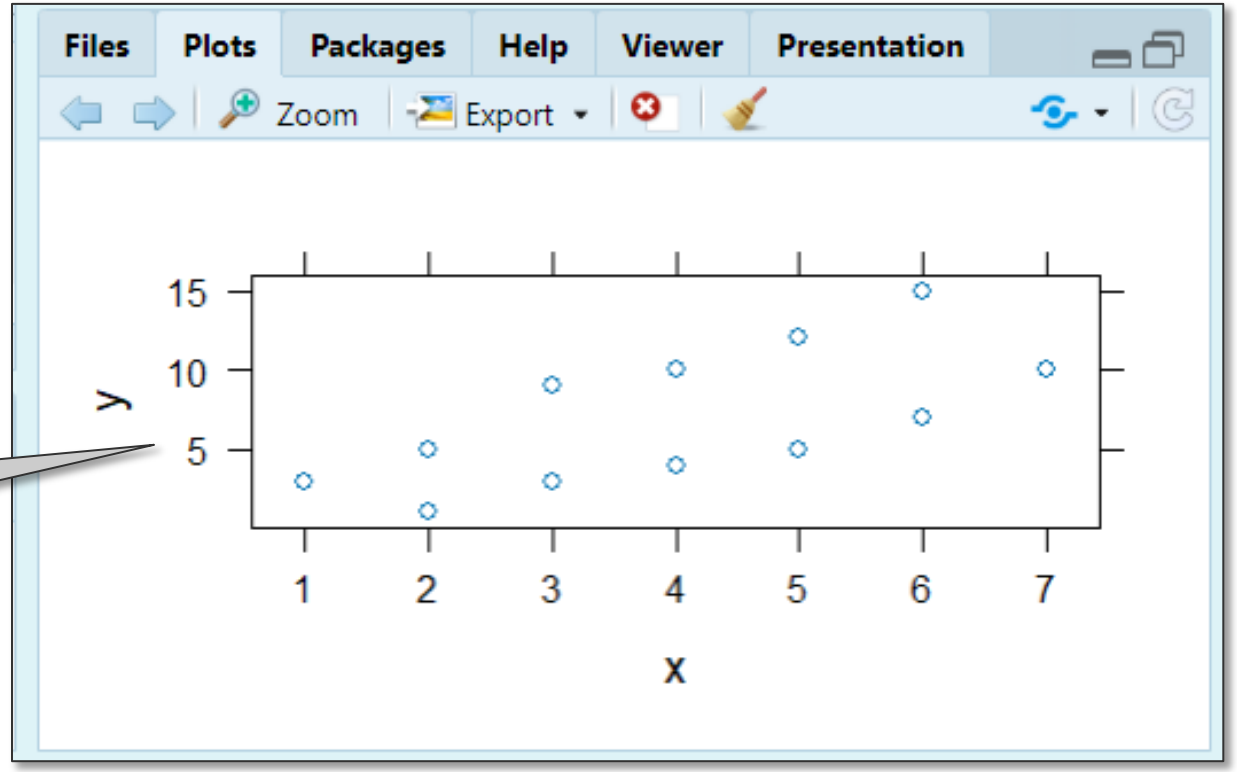
- R の主なグラフィックパッケージ

  代表的なパッケージ：標準パッケージ graphics、推奨パッケージ lattice、外部パッケージ ggplot2

  それぞれのパッケージは異なる設計思想と設計アプローチを持つ

１つのグラフの作成には、
１つのパッケージの関数を使用
異なるパッケージの関数を混在させない
（例外あり）

１つのグラフ（図）の作成には
１つのパッケージの関数を
組み合わせて使用



R と RStudio におけるグラフ作成環境

- R の主なグラフィックパッケージ

代表的なパッケージ：標準パッケージ graphics、推奨パッケージ lattice、外部パッケージ ggplot2

それぞれのパッケージは異なる設計思想と設計アプローチを持つ

１つのグラフの作成には、１つのパッケージの関数群のみを使用

異なるパッケージの関数を混合して使わない

混合すると、座標系や描画方式の違いにより、予期しない結果や描画エラーが生じる

同じデバイス領域に次のグラフを作成する場合、前とは別のパッケージの関数が使える

実践的なアプローチ

(i)　作成するグラフの目的とイメージを明確にする

(ii) その目的とイメージに適するパッケージを選択

(iii) 選択したパッケージの関数のみでグラフを完成させる

パッケージ graphics などで簡易に描いたグラフを、他のパッケージで作図し直すこともある

R と RStudio におけるグラフ作成環境

- R の主なグラフィックパッケージ

代表的なパッケージ：標準パッケージ graphics、推奨パッケージ lattice、外部パッケージ ggplot2
それぞれのパッケージは異なる設計思想と設計アプローチを持つ

graphics	R の基本的なグラフ描画機能、軽量かつシンプル（base、stats の一部の関数を含む） 高水準と低水準の関数の組合せで作図し、短いコードで素早くグラフを作成できる 小規模な解析やシンプルなグラフ作成に適すが、高度なカスタマイズには手間がかかる
lattice	条件付きプロット（カテゴリごとのグラフ分割）に特化 formula 記法による簡潔な記述、トレリスグラフィックスで複数のグラフを同時表示 多変量データの視覚化、グループ別比較に適す
ggplot2	レイヤーを積み重ねて構築する柔軟で強力なグラフ描画システム 美しく整ったデザインと高いカスタマイズ性 データサイエンスの現場では、標準ツールの一つとして広く使用されている

2　標準パッケージ graphics の概要

高水準グラフィックス関数
低水準グラフィックス関数
関数 par()によるグラフィックスパラメータの設定

標準パッケージ graphics の概要

- パッケージのインストールとロード

  標準パッケージ base, stats, graphics

  R をインストールするときに、同時にインストールされる

  → ［Pachages］タブのリストに存在

  R を起動するときに、自動的にロードされる

  → □にチェックが入る

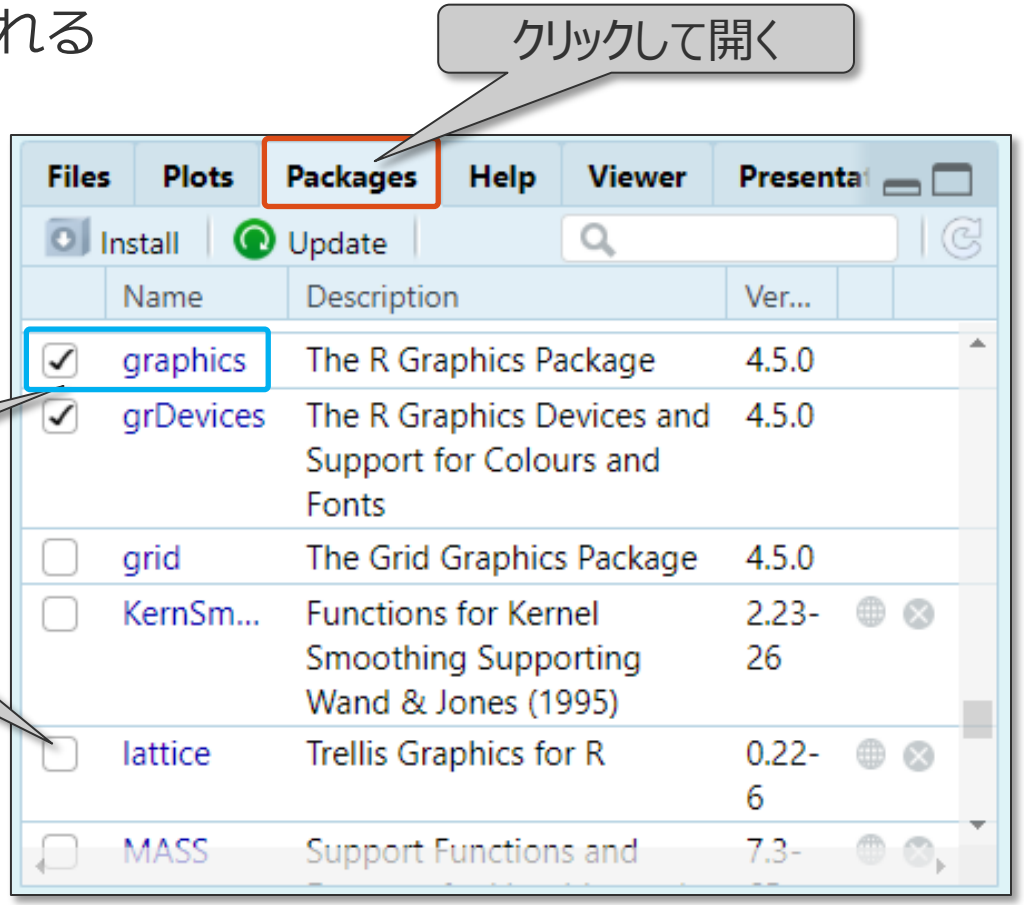
  RStudio の［Pachages］タブを表示

  チェックが入っていることを確認

  □ base、□ stats、□ graphics

  標準パッケージ graphics, base, stats の関数は

  R の起動後に直ちに使える



標準パッケージ graphics の概要

●グラフ作成用の高水準関数と低水準関数（graphics, base, stats の関数 ）

高水準グラフィックス関数、低水準グラフィックス関数、両者の組合せてグラフを作成

(i) 高水準グラフィックス関数
グラフの骨組みを描画
（図の全体構造を作成）

(ii) 低水準グラフィックス関数
装飾やデータの追加

関数 plot() は多機能
渡されたデータの種類に
対応したグラフを描画
ジェネリック関数
（points()、lines()、text()）

高水準グラフィックス関数

主な関数名	パッケージ名	グラフ名（主な用途）
plot()	base, graphics stats	散布図、線グラフ、棒グラフ 箱ひげ図、関数のプロット
pairs()	graphics	散布図行列
boxplot()	graphics	箱ひげ図
stripchart()	graphics	１次元散布図
hist()	graphics	ヒストグラム
barplot()	graphics	棒グラフ
dotchart()	graphics	Cleveland ドットチャート
mosaicplot()	graphics	モザイク図
fourfoldplot()	stats	4分割プロット
qqplot()	stats	２標本 Q-Q プロット
qqnorm()	stats	１標本 Q-Q プロット
contour()	graphics	等高線プロット
persp()	graphics	３次元透視図
image()	graphics	ヒートマップ
curve()	graphics	関数のプロット

低水準グラフィックス関数

主な関数	追加する要素
points( )	点
lines( )	線
abline( )	直線
segments( )	線分
arrows( )	矢印
rect( )	矩形
polygon( )	多角形
text( )	文字列
mtext( )	余白の文字列
legend( )	凡例
title( )	タイトル、軸ラベル
rug( )	軸上のラグ
axis( )	軸
box( )	箱
grid( )	グリッド線

標準パッケージ graphics の概要

R 4.0.0 で、関数 plot() は
パッケージ graphics から
パッケージ base に移動

●高水準関数と低水準関数（base, stats の一部の関
高水準グラフィックス関数、低水準グラフィックス関数、[illegible]の組合せてグラフを作成

(i) 高水準グラフィックス関数
グラフの骨組みを描画
（図の全体構造を作成）

(ii) 低水準グラフィックス関数
装飾やデータの追加

関数 plot() は多機能
渡されたデータの種類に
対応したグラフを描画
ジェネリック関数
（points()、lines()、text()）

高水準グラフィックス関数

主な関数名	パッケージ名	グラフ名（主な用途）
plot()	base, graphics stats	散布図、線グラフ、棒グラフ 箱ひげ図、関数のプロット
pairs()	graphics	散布図行列
boxplot()	graphics	箱ひげ図
stripchart()	graphics	１次元散布図
hist()	graphics	ヒストグラム
barplot()	graphics	棒グラフ
dotchart()	graphics	Cleveland ドットチャート
mosaicplot()	graphics	モザイク図
fourfoldplot()	stats	4分割プロット
qqplot()	stats	２標本 Q-Q プロット
qqnorm()	stats	１標本 Q-Q プロット
contour()	graphics	等高線プロット
persp()	graphics	３次元透視図
image()	graphics	ヒートマップ
curve()	graphics	関数のプロット

低水準グラフィックス関数

主な関数	追加する要素
points( )	点
lines( )	線
abline( )	直線
segments( )	線分
arrows( )	矢印
rect( )	矩形
polygon( )	多角形
text( )	文字列
mtext( )	余白の文字列
legend( )	凡例
title( )	タイトル、軸ラベル
rug( )	軸上のラグ
axis( )	軸
box( )	箱
grid( )	グリッド線

標準パッケージ graphics のグラフィックスパラメータ制御

●関数 par()：グラフィックスパラメータの制御

関数 par() で標準パッケージ graphiecs によるグラフ作成を制御（由来 parameters）

グラフの軸、余白、文字サイズ、色、レイアウトなどのグラフに関する要素を制御

通常、他のパッケージ（lattice, gglot2 など）には無効

関数 par() で制御するグラフィックスパラメータの多くは、高水準関数の中でも指定できる

例　描画する色の指定（par() と高水準関数の両方で設定可）

```
par(col = "blue")・・・・・この関数を実行した後に作成するすべてのグラフに有効
plot(x, y, col = "blue") ・・・このグラフのみに有効（par() の設定に対して優先する）
```

作図領域の余白を設定（par() のみで設定可、ごく一部の高水準関数で設定可能）

```
par(mar = c(5, 4, 2, 2)
```

現在のパラメータの設定内容を一時的に保存、その後、保存したパラメータの設定に復元

```
current_par <- par(no.readonly = TRUE)　→　他のスクリプトの実行　→　par(current_par)
```

ホウキ・アイコン（［Plots］タブ）で規定値に戻る

標準パッケージ graphics のグラフィックスパラメータ制御

●関数 par()：グラフィックスパラメータの制御

関数 par() で標準パッケージ graphiecs によるグラフ作成を制御（由来 parameters）

グラフの軸、余白、文字サイズ、色、レイアウトなどのグラフに関する要素を制御

通常、他のパッケージ（lattice, gglot2 など）には無効

関数 par() で制御するグラフィックスパラメータの多くは、高水準関数の中でも指定できる

例　描画する色の指定（par() と高水準関数の両方で設定可）

```
par(col = "blue")・・・・・この関数を実行した後に作成するすべてのグラフに有効
plot(x, y, col = "blue") ・・・このグラフのみに有効（par() の設定に対して優先する）
```

作図領域の余白を設定（par() のみで設定可、ごく一部の高水準関数で設定可能）

```
par(mar = c(5, 4, 2, 2)
```

現在のパラメータの設定内容を一時的に保存、その後、保存したパラメータの設定に復元

```
current_par <- par(no.readonly = TRUE)
```
→　他のスクリプトの実行　→　`par(current_par)`

読み取り専用のパラメータを除いて付値

ホウキ・アイコン（［Plots］タブ）で規定値に戻る

標準パッケージ graphics のグラフィックスパラメータ制御

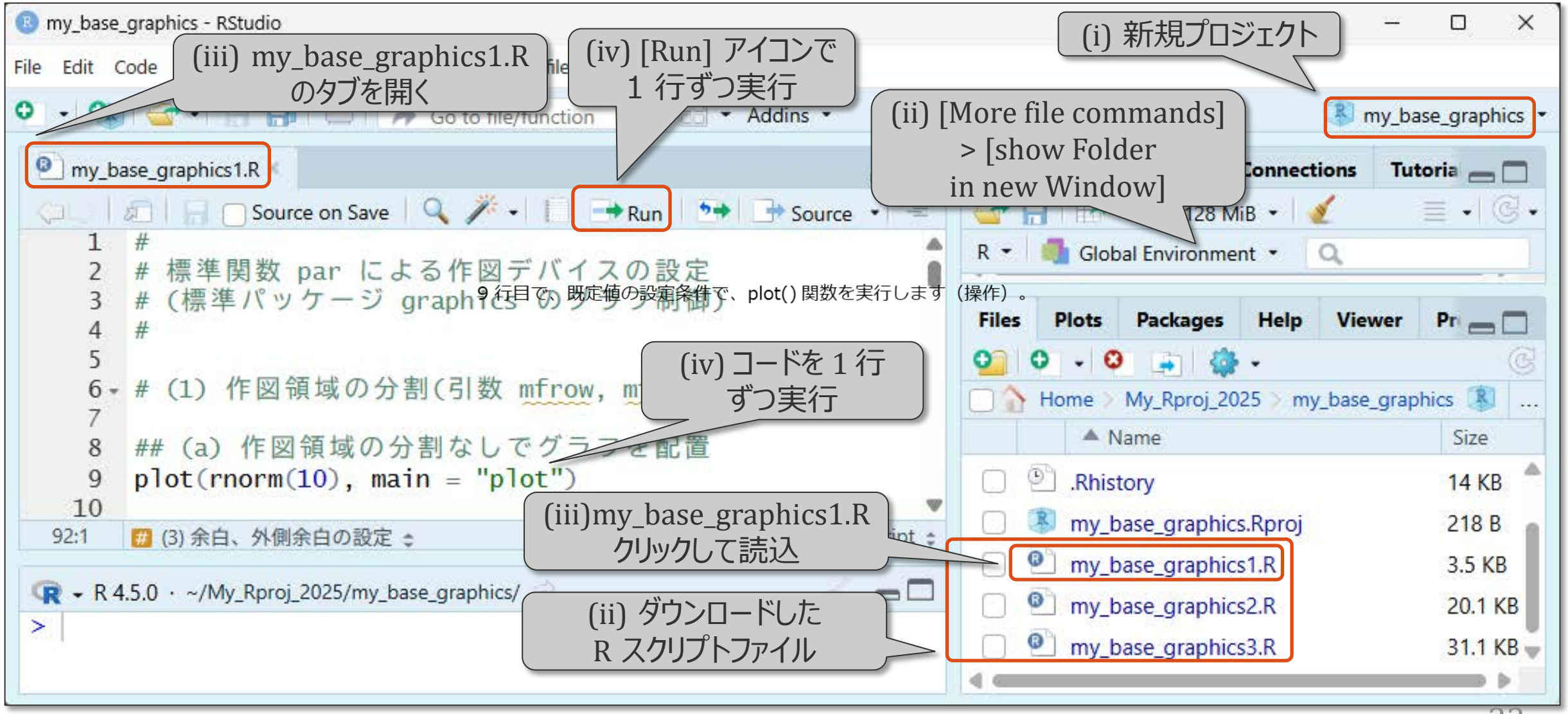
●関数 par() : グラフィックスパラメータの制御

par() の引数と制御する内容

1. プロット領域のレイアウトに関する引数	
mfrow, mfcol	グラフィックスデバイスの行と列の数
mar, mai, oma, omi, omd	グラフ作成の余白、外側余白
mex	余白の文字サイズ係数
fig	プロット領域のサイズと位置
2. テキストに関する引数	
cex	テキストの相対的な拡大率
cex.axis	軸目盛ラベルの文字サイズ係数
cex.lab	軸ラベルの文字サイズ係数
cex.main, cex.sub	タイトル,サブタイトルの文字サイズ係数
font	テキストのフォントスタイル
font.axis, font.lab, font.main, font.sub	各要素のフォントスタイル
col.axis, col.lab, col.main, col.sub	各要素の文字色

3. 線のスタイルに関する引数	
lty	線の種類
lwd	線の太さ
col	線の色
4. 点のスタイルに関する引数	
pch	プロットする点の種類
col、bg	点の色、背景色
5. その他	
las	目盛ラベルの向き
xaxs, yaxs	x軸、y軸のスタイル
xlog, ylog	x軸、y軸を対数軸に変換
ann	メインタイトルや軸ラベルを表示
new	既存のプロットに新しいプロットを追加

標準パッケージ graphics のグラフィックスパラメータ制御



標準パッケージ graphics のグラフィックスパラメータ制御

●関数 par()：(1) 作図領域の分割

(a) 作図領域の分割なしでグラフを配置

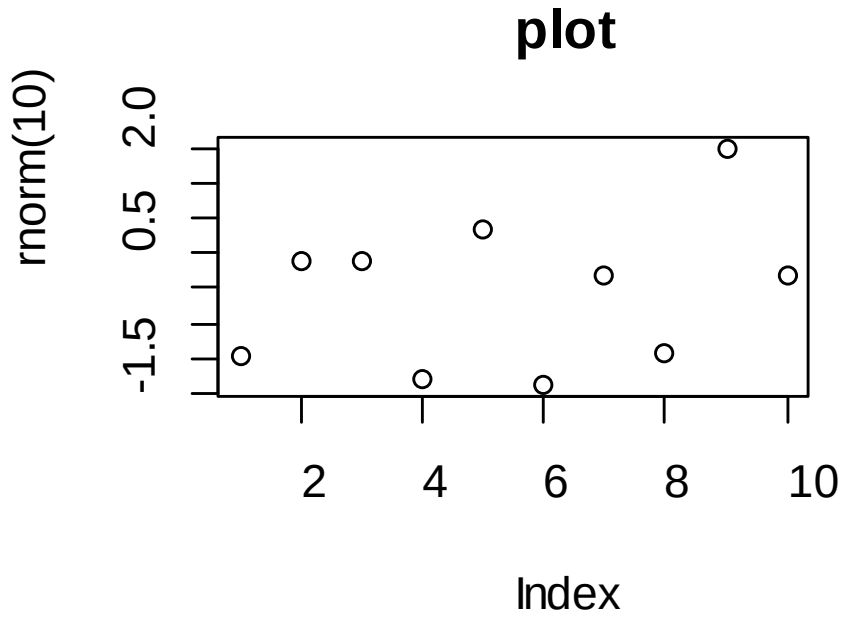
```
1  #
2  # 標準関数 par による作図デバイスの設定
3  # (標準パッケージ graphics のグラフ制御)
4  #
5
6  # (1) 作図領域の分割(引数 mfrow, mfcol) ---------
7
8  ## (a) 作図領域の分割なしでグラフを配置
9  plot(rnorm(10), main = "plot")
10
11 ## (b) 作図領域を2行3列に分割、横方向に配置
12 current_par <-
13   par(no.readonly = TRUE)  # パラメータ保存
14 par(mfrow = c(2,3))        # 作図領域を分割
15
16 plot(rnorm(10), main = "figure 1")
17 plot(rnorm(10), main = "figure 2")
18 plot(rnorm(10), main = "figure 3")
19 plot(rnorm(10), main = "figure 4")
20 plot(rnorm(10), main = "figure 5")
21 plot(rnorm(10), main = "figure 6")
```

[Source エディタ] のスクリーンショット

9 行目を [Run] アイコンで実行

(my_base_graphics1.R：1-21)

(a) 作図領域の分割なしでグラフを配置



標準パッケージ graphics のグラフィックスパラメータ制御

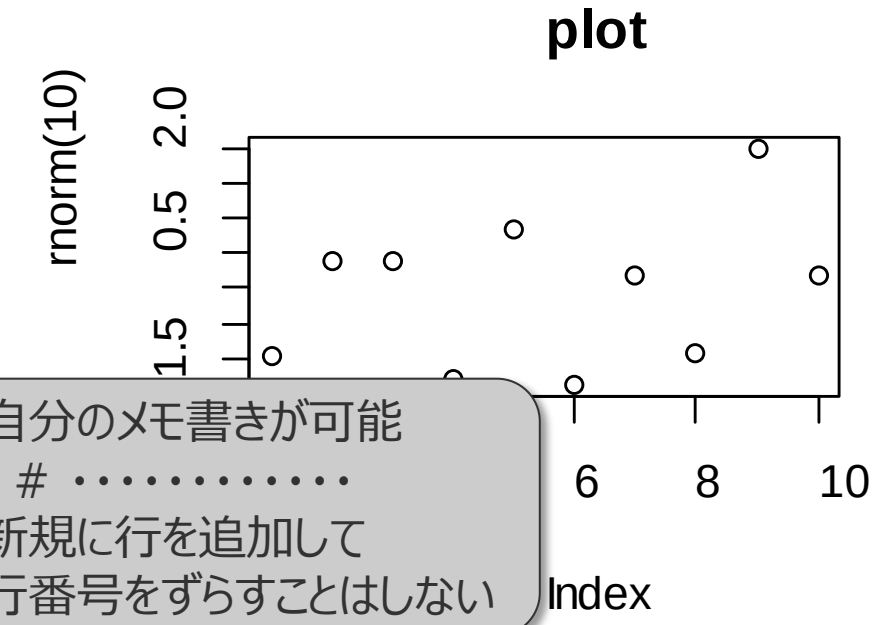
●関数 par()：(1) 作図領域の分割

(a) 作図領域の分割なしでグラフを配置

```
1  #
2  # 標準関数 par による作図デバイスの設定
3  # (標準パッケージ graphics のグラフ制御)
4  #
5
6  # (1) 作図領域の分割(引数 mfrow, mfcol) -----
7
8  ## (a) 作図領域の分割なしでグラフを配置
9  plot(rnorm(10), main = "plot")
10
11 ## (b) 作図領域を2行3列に分割、横方向に配置
12 current_par <-
13   par(no.readonly = TRUE)  # パラメータ保存
14 par(mfrow = c(2,3))         # 作図領域を分割
15
16 plot(rnorm(10), main = "figure 1")
17 plot(rnorm(10), main = "figure 2")
18 plot(rnorm(10), main = "figure 3")
19 plot(rnorm(10), main = "figure 4")
20 plot(rnorm(10), main = "figure 5")
21 plot(rnorm(10), main = "figure 6")
```

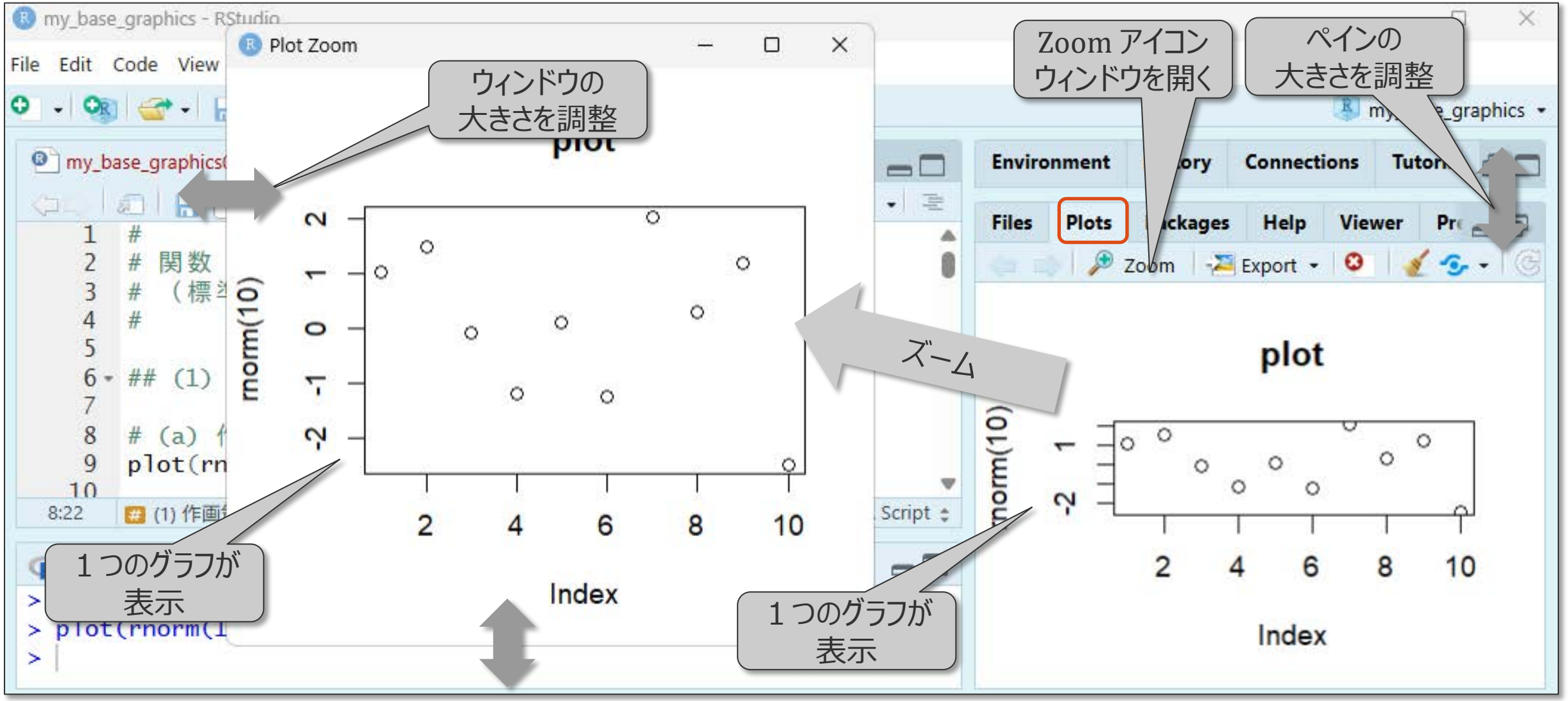
(my_base_graphics1.R：1-21)

(a) 作図領域の分割なしでグラフを配置



自分のメモ書きが可能
・・・・・・・・・・・
新規に行を追加して
行番号をずらすことはしない

標準パッケージ graphics のグラフィックスパラメータ制御



標準パッケージ graphics のグラフィックスパラメータ制御

●関数 par()：(1) 作図領域の分割

関数 par()で作図領域を格子状の行と列に分割

引数 mfrow（multi-figure row）：横方向

引数 mfcol（multi-figure column）：縦方向

(b) 作図領域を２行３列に分割、横方向に配置

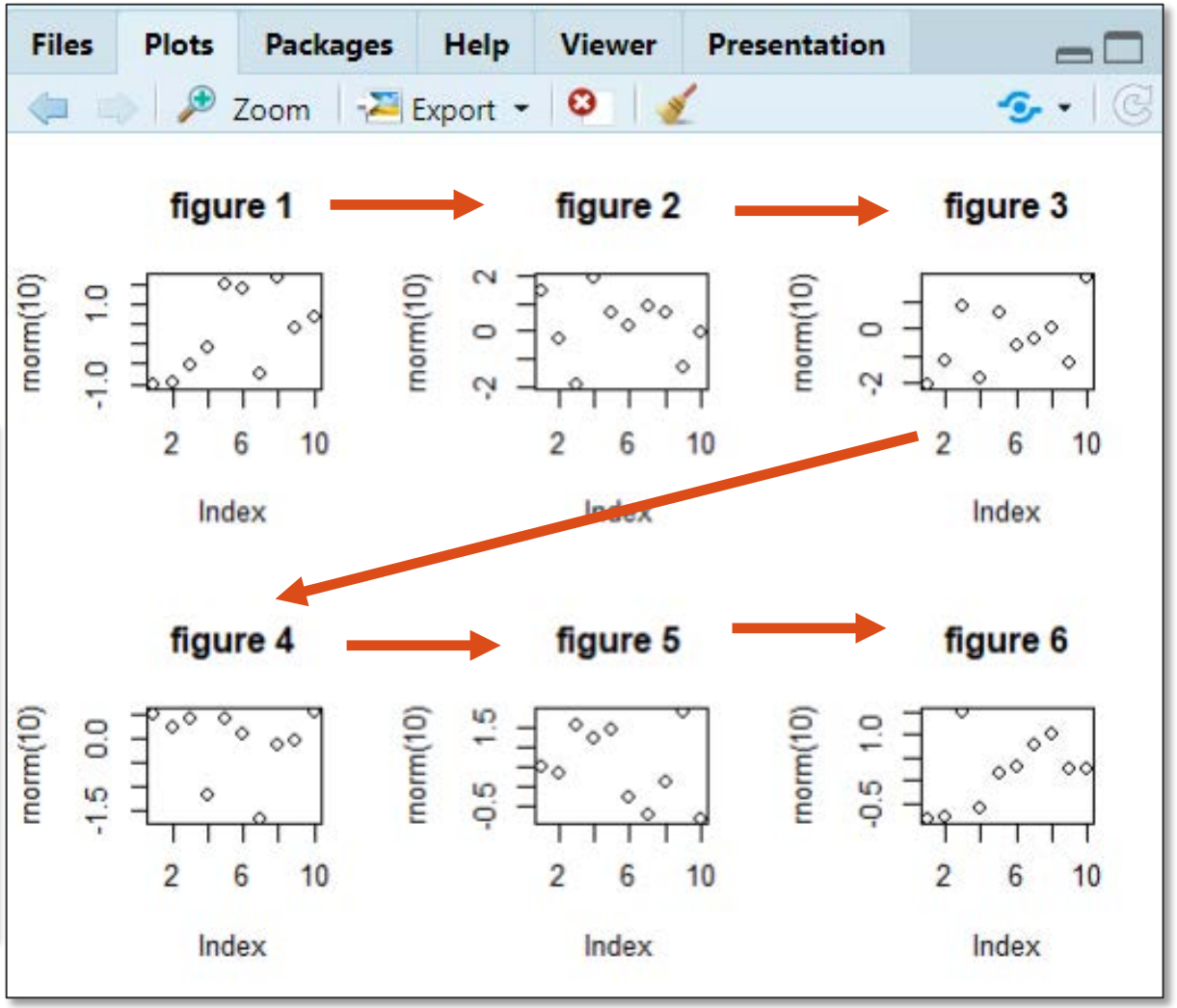
事例：par(mfrow = c(2、3))

```
11  ## (b) 作図領域を2行3列に分割、横方向に配置
12  current_par <-
13    par(no.readonly = TRUE)   # パラメータ保存
14  par(mfrow = c(2,3))         # 作図領域を分割
15
16  plot(rnorm(10), main = "figure 1")
17  plot(rnorm(10), main = "figure 2")
18  plot(rnorm(10), main = "figure 3")
19  plot(rnorm(10), main = "figure 4")
20  plot(rnorm(10), main = "figure 5")
21  plot(rnorm(10), main = "figure 6")
22
23  par(current_par)            # パラメータ復元
```

引数 mfrow

（my_base_graphics1.R：11-23）

(b) 作図領域を２行３列に分割、横方向に配置



標準パッケージ graphics のグラフィックスパラメータ制御

- 関数 par()：(1) 作図領域の分割

  関数 par()で作図領域を格子状の行と列に分割

  引数 mfrow（multi-figure row）：横方向

  引数 mfcol（multi-figure column）：縦方向

  (c) 作図領域を２行３列に分割、縦方向に配置

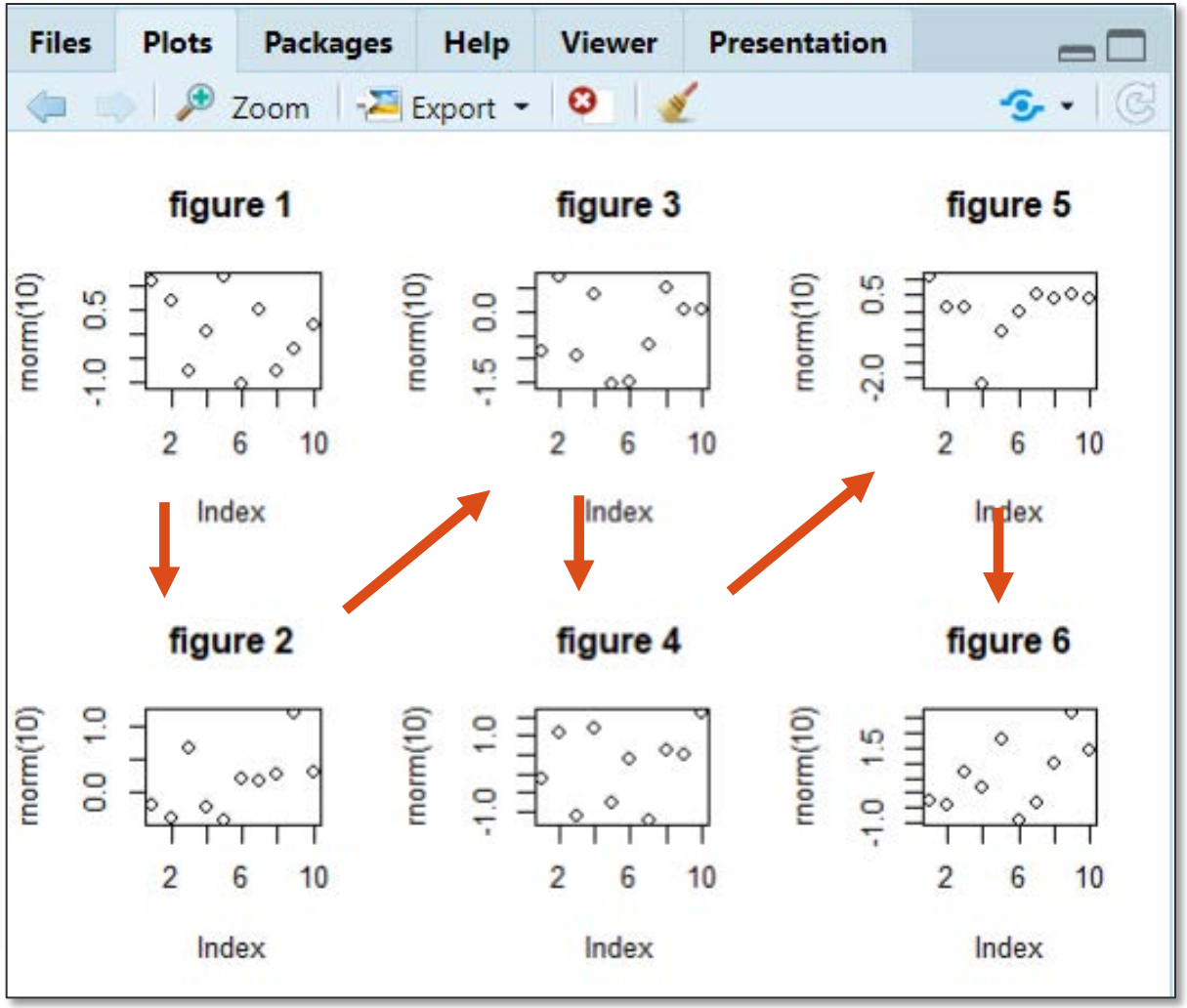
  事例：par(mfcol = c(2、3))

```
25  ## (c) 作図領域を2行3列に分割、縦方向に配置
26  current_par <-
27    par(no.readonly = TRUE)   # パラメータ保存
28  par(mfcol = c(2,3))         # 作画領域を分割
29
30  plot(rnorm(10), main = "figure 1")
31  plot(rnorm(10), main = "figure 2")
32  plot(rnorm(10), main = "figure 3")
33  plot(rnorm(10), main = "figure 4")
34  plot(rnorm(10), main = "figure 5")
35  plot(rnorm(10), main = "figure 6")
36
37  par(current_par)            # パラメータ復元
```

引数 mfcol

（my_base_graphics1.R：25-37）

(c) 作図領域を２行３列に分割、縦方向に配置



標準パッケージ graphics のグラフィックスパラメータ制御

●関数 par()：(2) 作図する領域

(i) プロット領域（plot region）

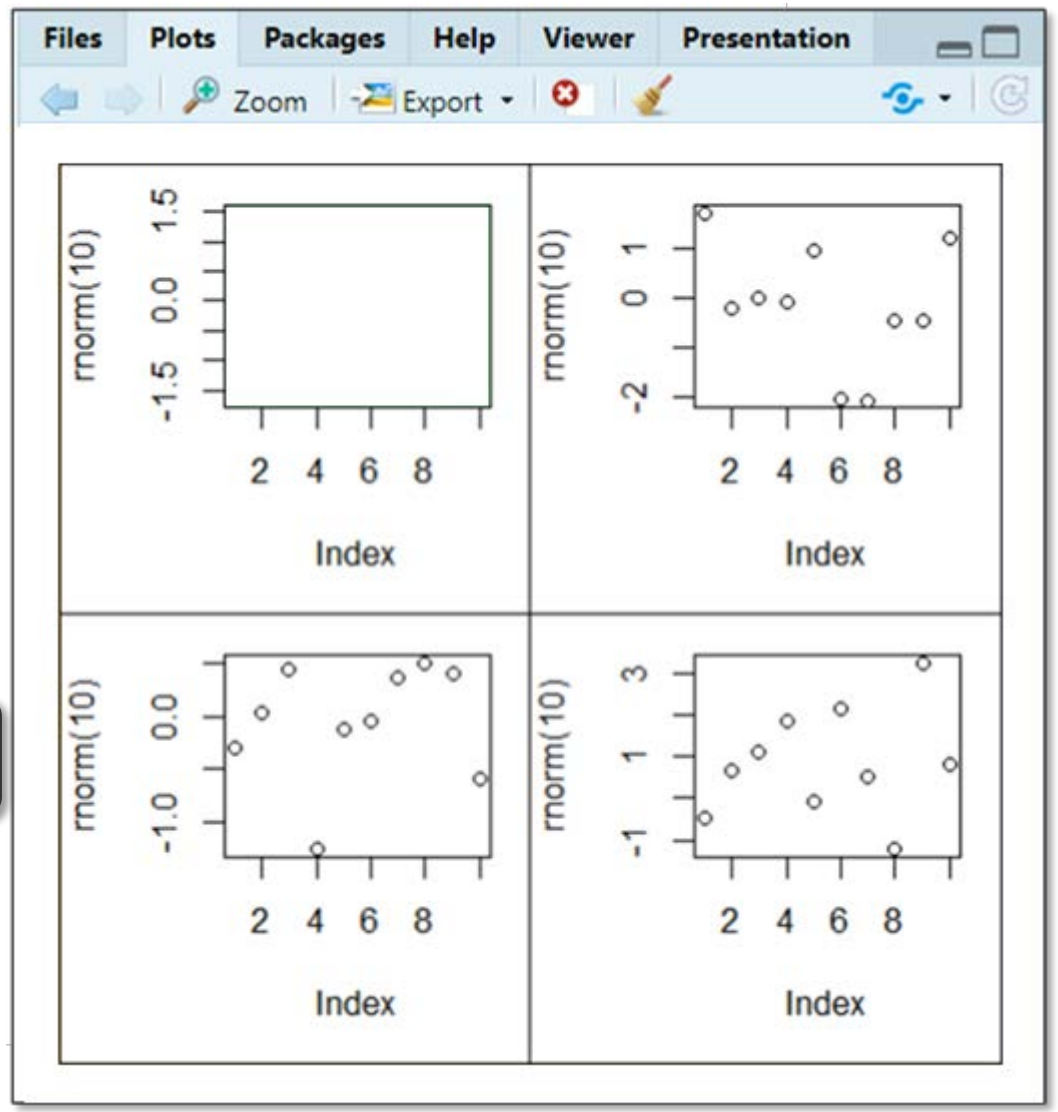
(ii) 作図領域（figure region）

(iii) デバイス領域（device region）

```
40  # (2) 作図する領域 ----------------------------
41  #     デバイス領域、作図領域、プロット領域
42
43  current_par <-
44    par(no.readonly = TRUE)   # パラメータ保存
45
46  par(oma = c(1,1,1,1))       # 外側余白を行数で指定
47  par(mar = c(5,4,1,1))       # 余白を行数で指定
48  par(mfrow = c(2, 2))        # 作図領域を分割
49
50  for (i in 1:4) {
51    plot(rnorm(10))
52    box(which = "figure")     # 作図領域の枠を表示
53  }
54
55  par(current_par)            #パラメータ復元
```

コードの
詳細は省略

（my_base_graphics1.R：40-55）



標準パッケージ graphics のグラフィックスパラメータ制御

●関数 par()：(2) 作図する領域

(i) プロット領域

点、線などが描かれる領域

(ii) 作図領域

プロット領域 ＋軸ラベル、
タイトルなどの領域

(iii) デバイス領域

［plots］タブと一致

2 種類の余白

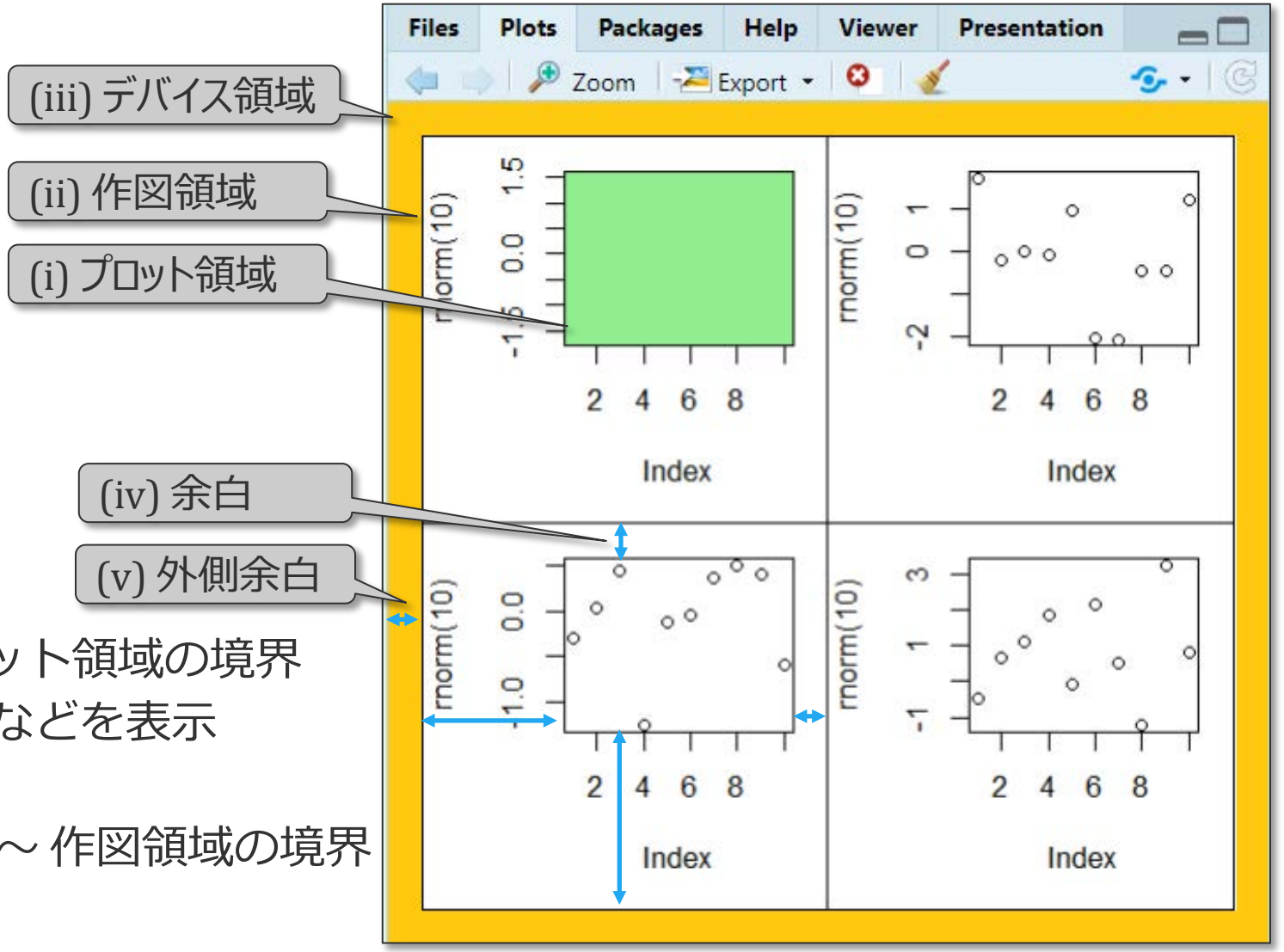
(iv) 余白： 作図領域 の境界～ プロット領域の境界

タイトル、軸ラベル、軸目盛などを表示

下、左、上、右

(v) 外側余白 ：デバイス領域の境界～ 作図領域の境界

下、左、上、右



標準パッケージ graphics のグラフィックスパラメータ制御

●関数 par()：(3) 余白、外側余白、軸

タイトル、横軸と縦軸の軸ラベル、目盛ラベル、
を表示するスペースを勘案して調整

関数 par()での指定方法：par(mar=c(下, 左, 上, 右))

mar　：余白（margin）、行単位（小数点以下有効）

mai　：余白、インチ単位

oma　：外側余白（outer margin）、行単位

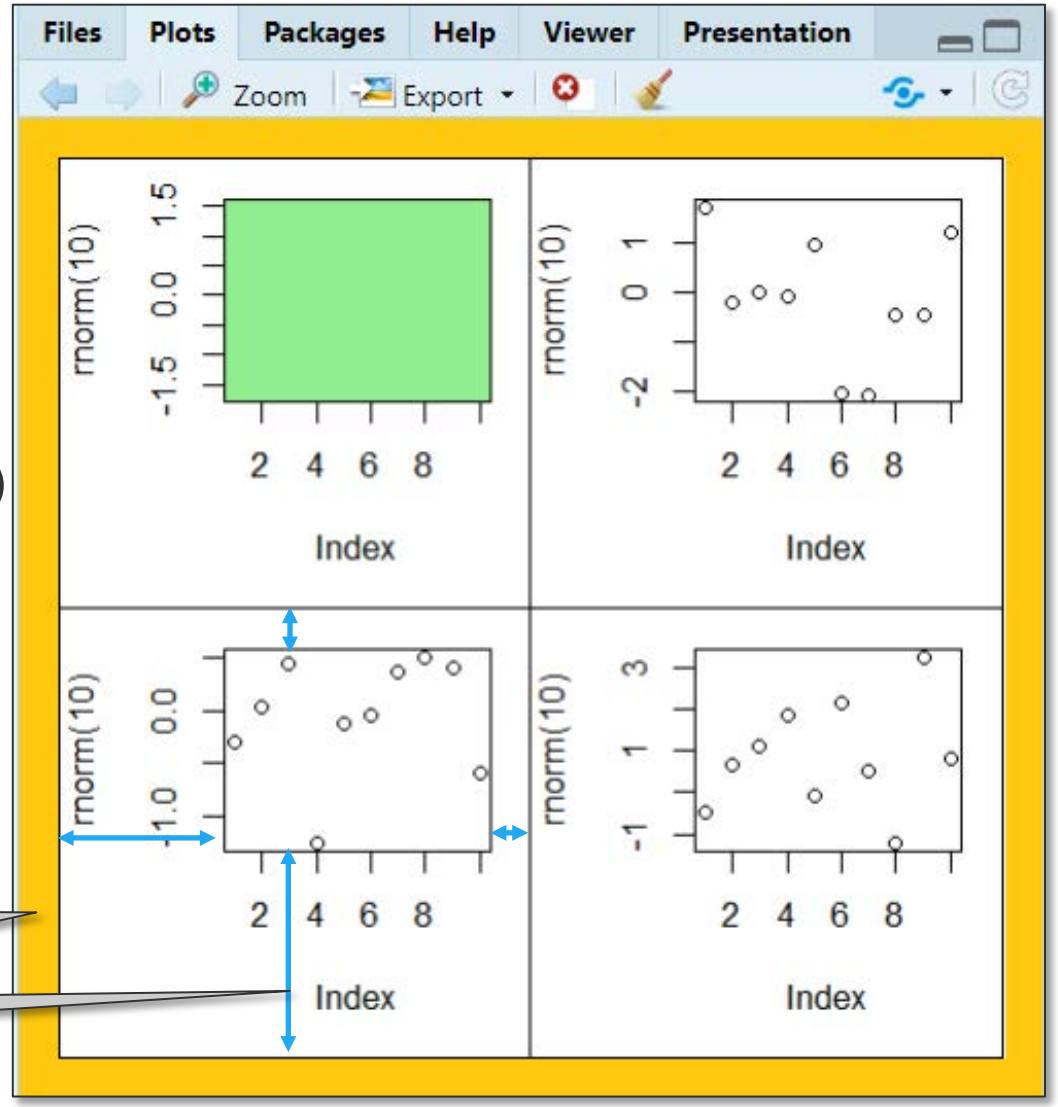
omi　：外側余白、インチ単位

omd　：外側余白、デバイス領域の幅を0～1 で表示

左端が 0、右端が 1

上端が 0、下端が 1

中間の位置は 0.5



標準パッケージ graphics のグラフィックスパラメータ制御

●関数 par()：(3) 余白、外側余白、軸

(a) 既定値（デフォルト）の表示、外側余白、軸

```
> par("mar")
[1] 5.1 4.1 4.1 2.1
> par("mai")
[1] 1.02 0.82 0.82 0.42
> par("oma")
[1] 0 0 0 0
> par("omi")
[1] 0 0 0 0
> par("omd")
[1] 0 1 0 1
```

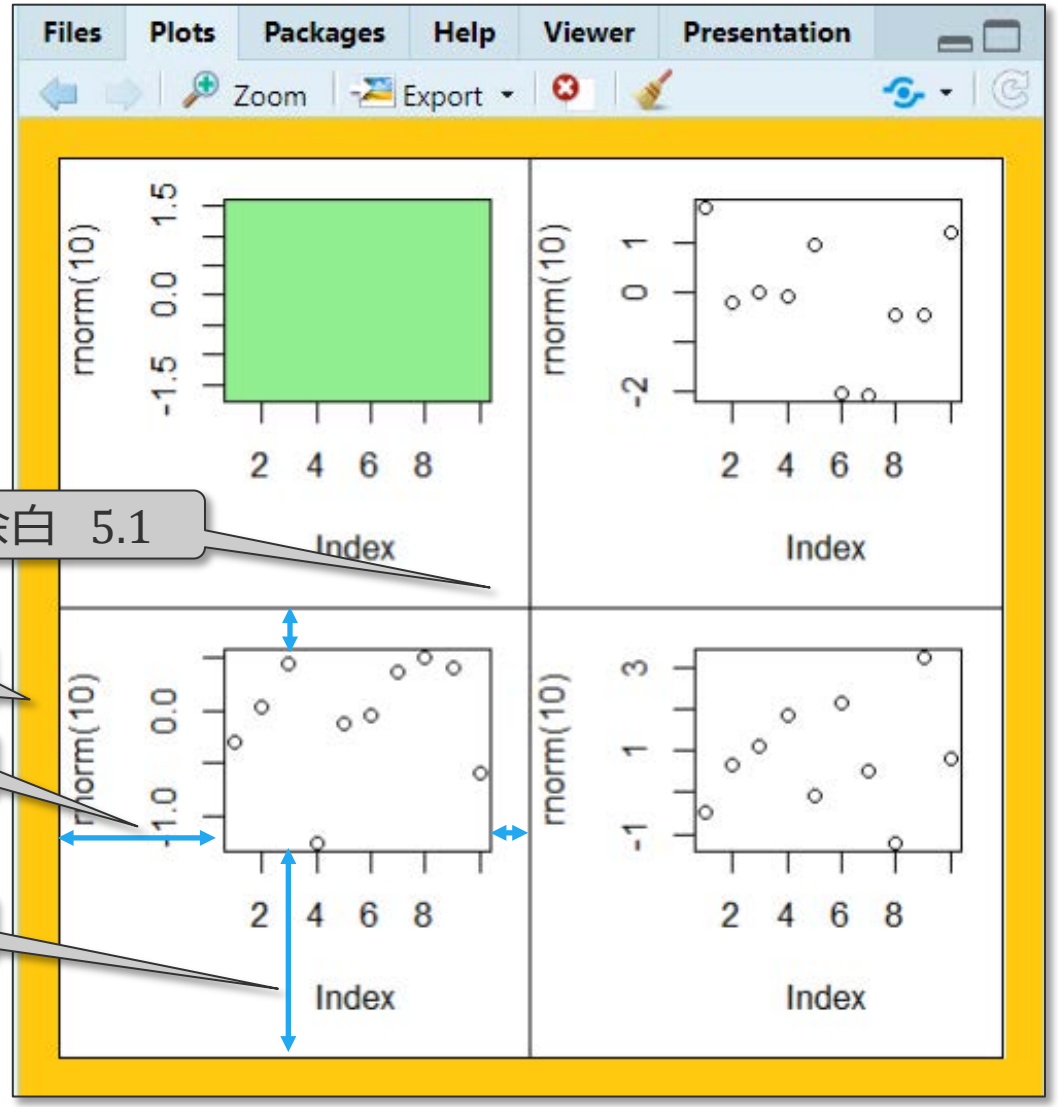
mar = c(5, 4, 4, 2) + 0.1
行単位

外側余白の既定値は 0

左端が 0、右端が 1

```
58 # (3) 余白、外側余白、軸----------------------
59
60 ## (a) 既定値(デフォルト)の表示
61 par("mar")    # 余白、行単位(下、左、上、右)
62 par("mai")    # 余白、インチ単位
63 par("oma")    # 外側余白、行数単位
64 par("omi")    # 外側余白、インチ単位
65 par("omd")    # 外側余白、0～1(左、右、上、下)
```

（my_base_graphics1.R：58-65）



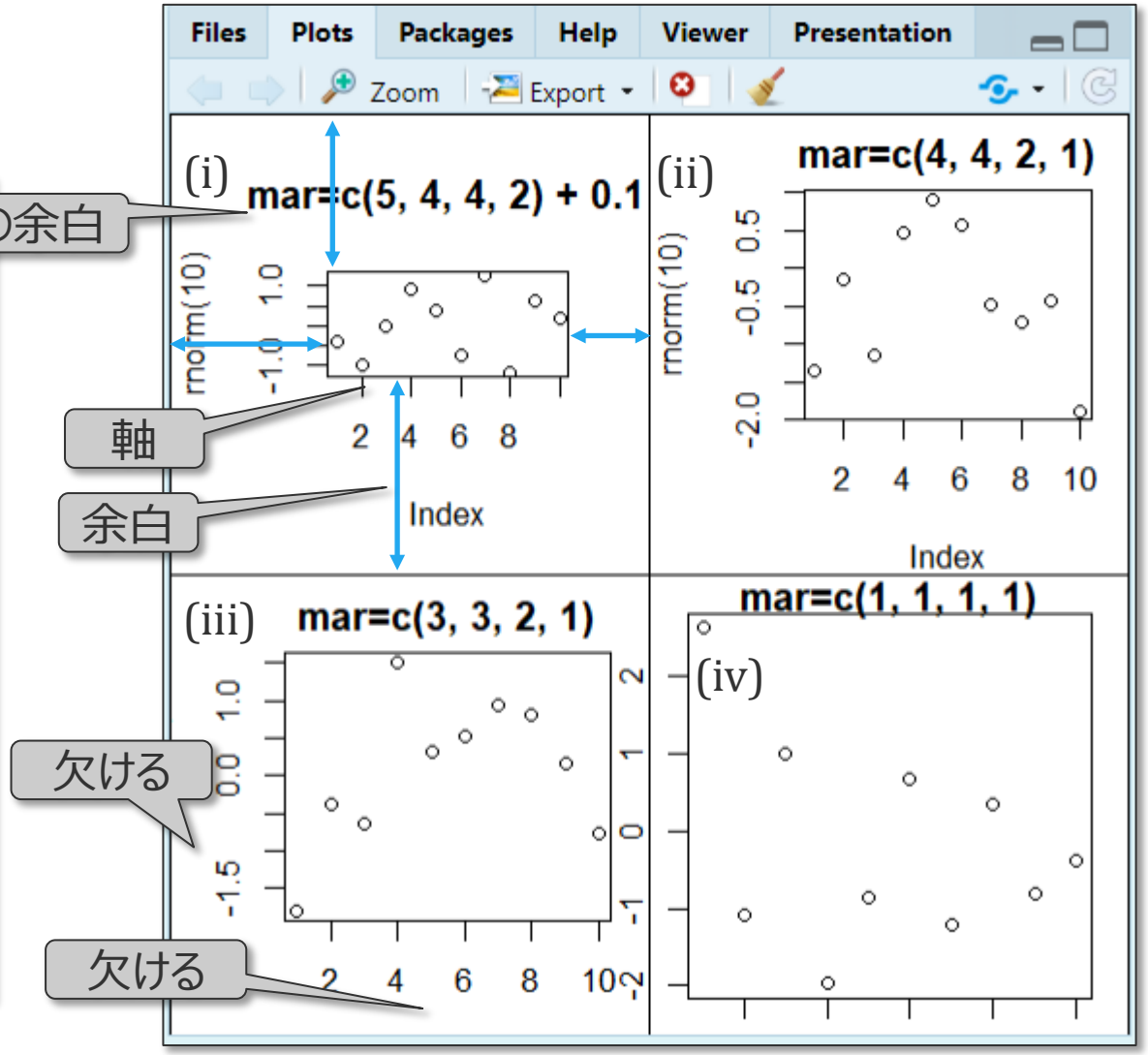
標準パッケージ graphics のグラフィックスパラメータ制御

- 関数 par()：(3) 余白、外側余白、軸

(b) 余白　引数 mar で余白を行単位で指定

```
67  ## (b) 余白
68  current_par <-
69    par(no.readonly = TRUE)     # パラメータ保存
70  par(mfrow = c(2, 2))          # 作画画面を分割
71
72  plot(rnorm(10), main = "mar=c(5,4,4,2)+0.1")
73  box(which = "figure")         # (i) 既定値
74
75  par(mar = c(4, 4, 2, 1))
76  plot(rnorm(10), main = "mar=c(4, 4, 2, 1)")
77  box(which = "figure")         # (ii)
78
79  par(mar = c(3, 3, 2, 1))
80  plot(rnorm(10), main = "mar=c(3, 3, 2, 1)")
81  box(which = "figure")         # (iii)
82
83  par(mar = c(1, 1, 1, 1))
84  plot(rnorm(10), main = "mar=c(1, 1, 1, 1)")
85  box(which = "figure")         # (iv)
86
87  par(current_par)              # パラメータ復元
```

（my_base_graphics1.R：67-87）



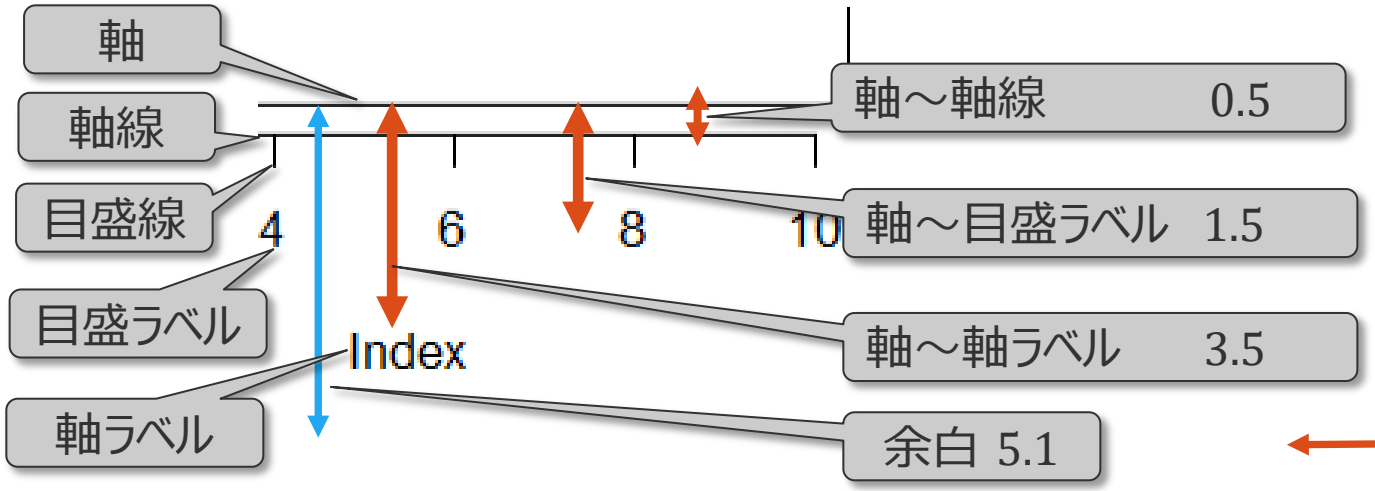
標準パッケージ graphics のグラフィックスパラメータ制御

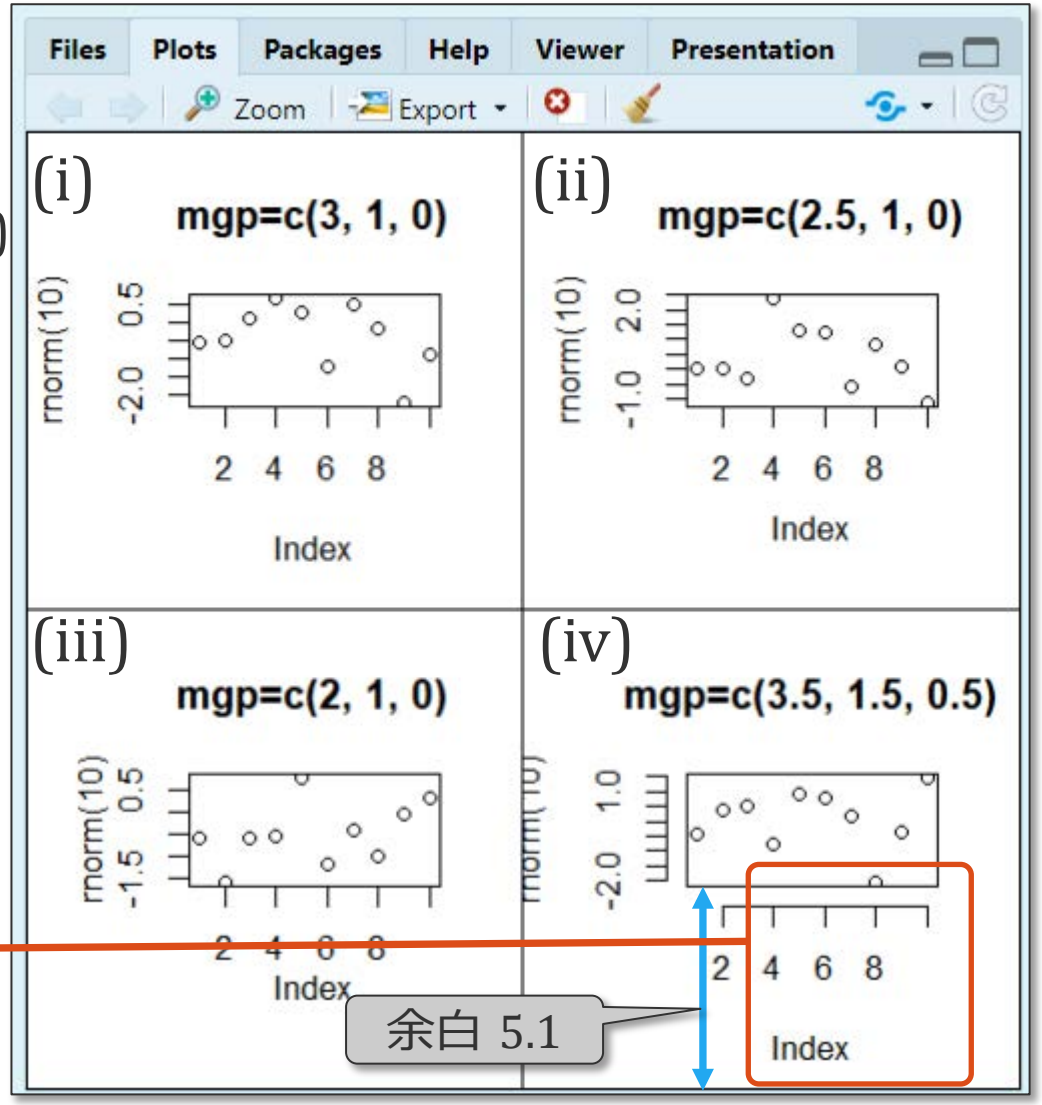
●関数 par()：(3) 余白、外側余白、軸

(c) 軸ラベル、目盛ラベル、軸線の位置の設定

par(mgp=c(軸～軸ラベル, 軸～目盛ラベル, 軸～軸線))

par(mar = c(5, 4, 4, 2) + 0.1)
par(mgp = c(3.5, 1.5, 0.5))





標準パッケージ graphics のグラフィックスパラメータ制御

●関数 par()：(3) 余白、外側余白、軸

(c) 軸ラベル、目盛ラベル、軸線の位置の設定

```
89  ## (c) 軸ラベル、目盛ラベル、軸線の位置
90  current_par <-
91    par(no.readonly = TRUE)    # パラメータ保存
92  par(mfrow = c(2, 2))
93
94  plot(rnorm(10), main = "mgp=c(3, 1, 0)")
95  box(which = "figure")          # (i) 既定値
96
97  par(mgp = c(2.5, 1, 0))
98  plot(rnorm(10), main = "mgp=c(2.5, 1, 0)")
99  box(which = "figure")          # (ii)
100
101 par(mgp = c(2, 1, 0))
102 plot(rnorm(10), main = "mgp=c(2, 1, 0)")
103 box(which = "figure")          # (iii)
104
105 par(mgp = c(3.5, 1.5, 0.5))
106 plot(rnorm(10), main = "mgp=c(3.5, 1.5, 0.5)")
107 box(which = "figure")          #(iv)
108
109 par(current_par)               # パラメータ復元
```

既定値

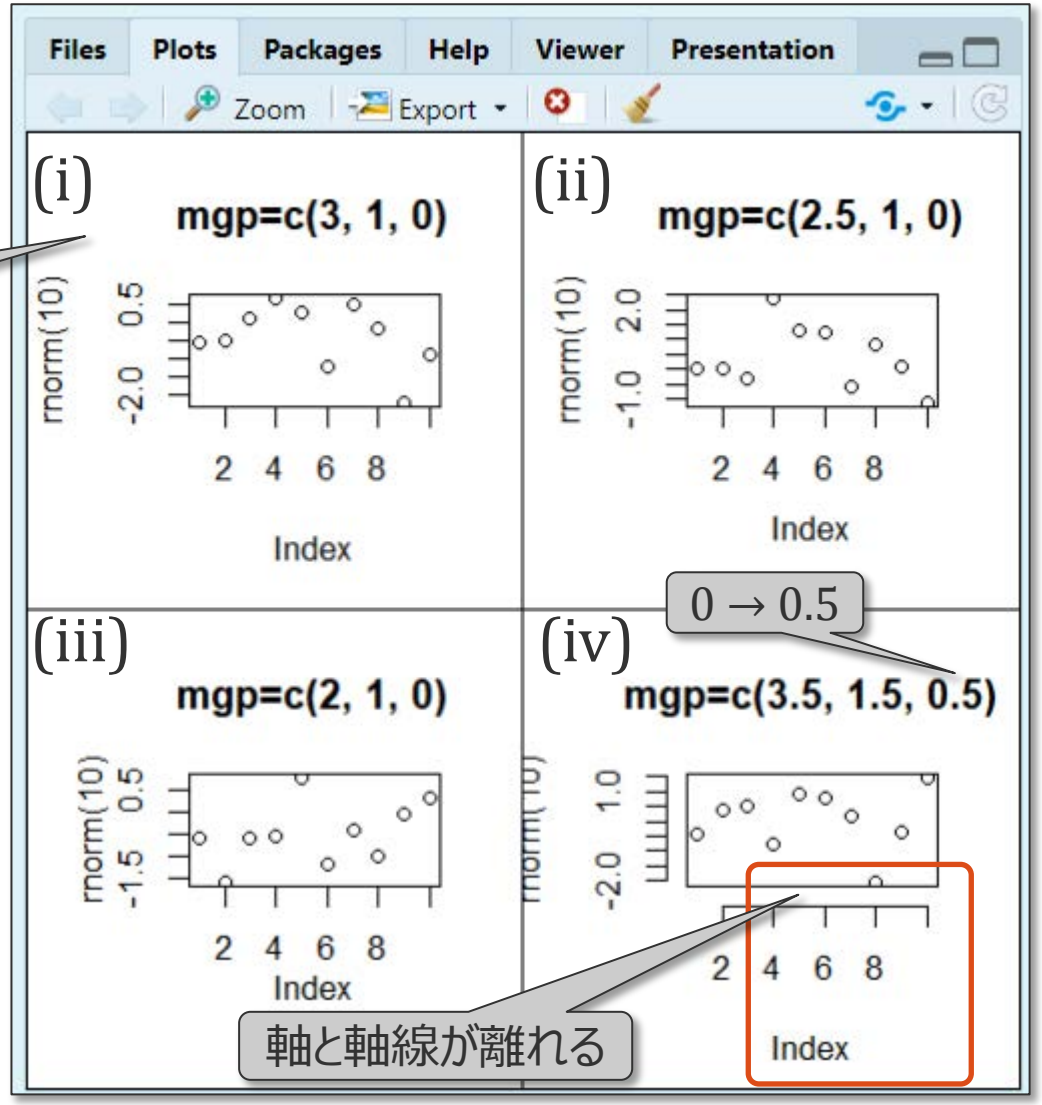
(i) 既定値

(ii)

(iii)

(iv)

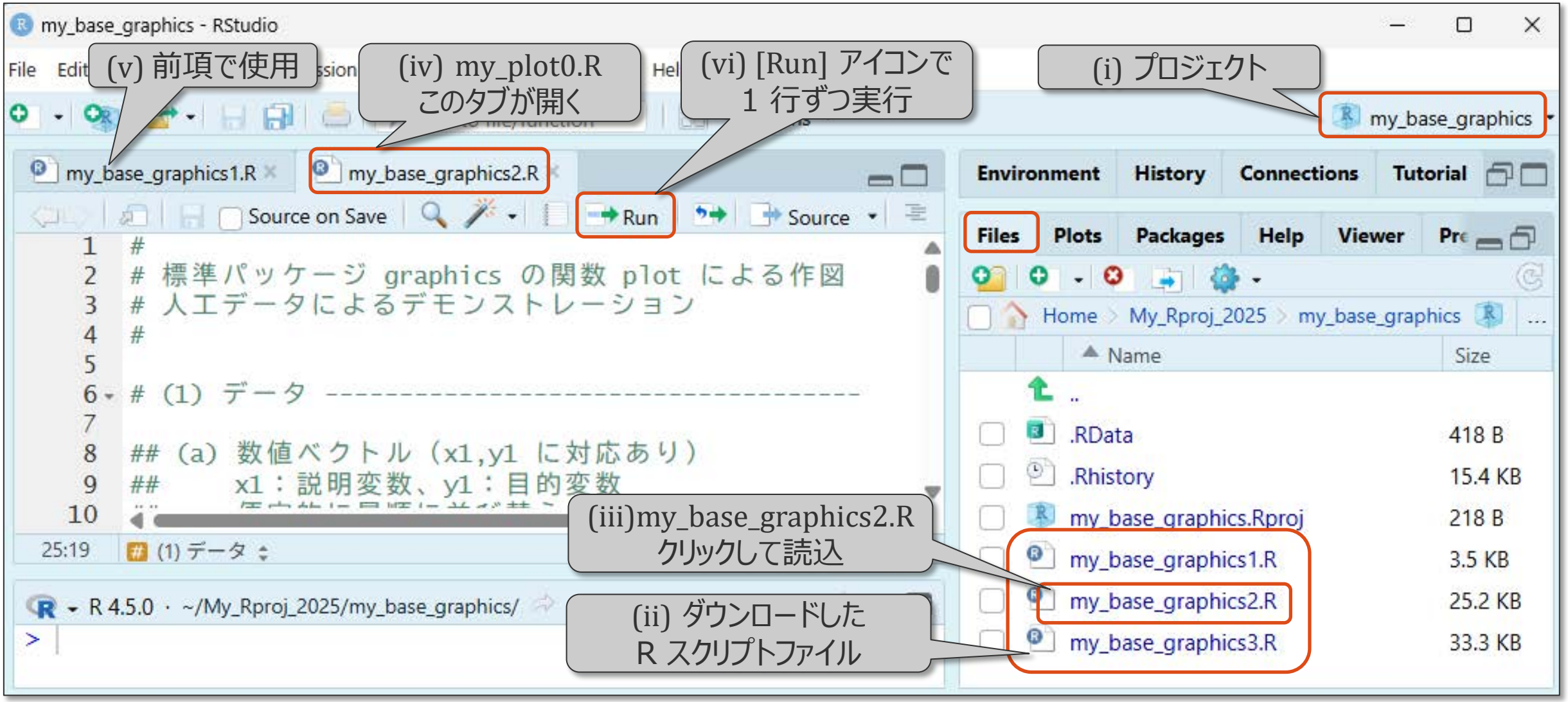
(my_base_graphics1.R：89-109)



3　関数 plot() の使い方

ジェネリック関数の一つ
plot() に渡すデータの種類
plot() の引数によるグラフのカスタマイズ
plot() と低水準グラフィックス関数の組合せ

関数 plot() の使い方


my_base_graphics - RStudio
(v) 前項で使用
(iv) my_plot0.R
このタブが開く
(vi) [Run] アイコンで
1 行ずつ実行
(i) プロジェクト
my_base_graphics
my_base_graphics1.R
my_base_graphics2.R
Source on Save
Run
Source
1 #
2 # 標準パッケージ graphics の関数 plot による作図
3 # 人工データによるデモンストレーション
4 #
5
6 # (1) データ ----------
7
8 ## (a) 数値ベクトル（x1,y1 に対応あり）
9 ## x1：説明変数、y1：目的変数
10
25:19
(1) データ
R 4.5.0 · ~/My_Rproj_2025/my_base_graphics/
>
Environment
History
Connections
Tutorial
Files
Plots
Packages
Help
Viewer
Home > My_Rproj_2025 > my_base_graphics
Name
Size
..
.RData 418 B
.Rhistory 15.4 KB
my_base_graphics.Rproj 218 B
my_base_graphics1.R 3.5 KB
my_base_graphics2.R 25.2 KB
my_base_graphics3.R 33.3 KB
(iii)my_base_graphics2.R
クリックして読込
(ii) ダウンロードした
R スクリプトファイル

関数 plot() の使い方

●準備：(1) データ

(a) 数値ベクトル：x1, y2（x1：説明変数、y1：目的変数、対応あり、x1 の値で昇順ソート）

(b) データフレーム：df（grp：因子ベクトル（group）、 obs：数値ベクトル（observation））

群　　観測値

```
6  # (1) データ ----------------------------------
7
8  ## (a) 数値ベクトル (x1,y1 に対応あり)
9  ##  x1：説明変数、y1：目的変数、x1で昇順に並び替え
10 x1 <- c( 1, 3, 6, 7, 9, 10)
11 y1 <- c( 2, 4, 5, 4, 7, 11)
12
13 ## (b) データフレーム(grp, obs に対応あり)
14 ##     grp：group(群),  obs：observation(観測値)
15 df <- data.frame(
16   grp = factor(
17     c("A1","A1","A1","A1","A1","A2","A2","A2","A2"),
18     levels = c("A1", "A2")
19     ),
20   obs = c(7, 5, 6, 10, 2,  12, 10, 5, 15)
21   )
22 View(df)
```

(my_base_graphics2.R：6-22)

(a) 数値ベクトル

x1	y1
1	2
3	4
6	5
7	4
9	7
10	11

昇順

対応あり

(b) データフレーム

df

grp	obs
A1	7
A1	5
A1	6
A1	10
A1	2
A2	12
A2	10
A2	5
A2	15

関数 plot() の使い方

●準備：(1) データ

(c) 因子ベクトルとテーブルオブジェクト

gender：因子ベクトル、原因　Male/Female

answer：因子ベクトル、結果　yes/no

tb：テーブルオブジェクト

```
24  ## (c) 因子ベクトルとテーブルオブジェクト
25  ##      原因：Male/Female → 結果：yes/no、
26  gender <- factor(
27    rep(c("male", "female"), times = c(4, 5)),
28    levels = c("male", "female")
29    )
30  answer <- factor(
31    c("yes", "no",  "no", "yes",
32      "no",  "no", "yes",  "no",  "no"),
33    levels = c("no", "yes")
34    )
35  tb <- table(gender, answer)
36  print(tb)
```

水準の順序

（my_base_graphics2.R：24-36）

因子ベクトル

gender	answer
male	yes
male	no
male	no
male	yes
female	no
female	no
female	yes
female	no
female	no

集計

テーブルオブジェクト

```
> print(tb)
        answer
gender   no yes
  male    2   2
  female  4   1
```

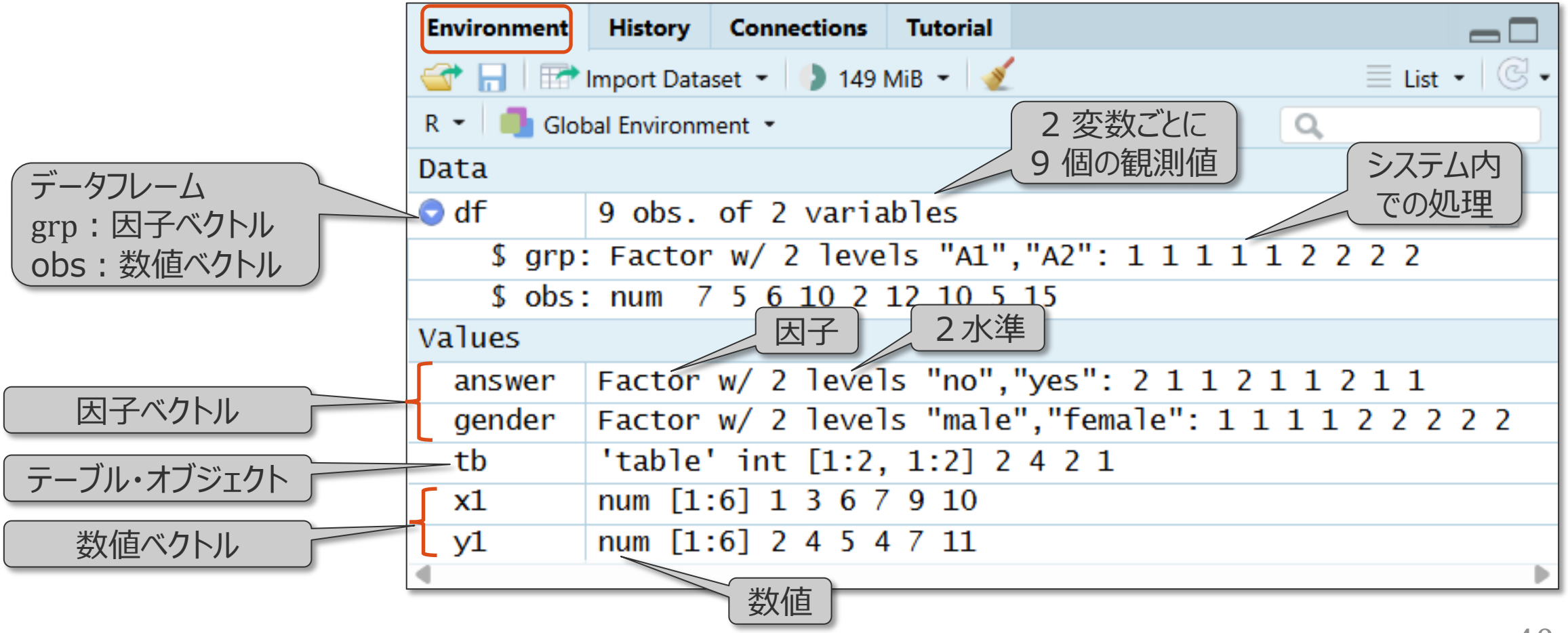
水準の順序

（Console の表示）

関数 plot() の使い方

●準備：(1) データ

スクリプトを実行後、［Environment］タブで作成したオブジェクトを確認



関数 plot() の使い方

●準備：(2) 関数 par() による余白と軸のカスタマイズ

par(mar = c(3, 4, 2, 2), mgp = c(1.5, 0.5, 0))

ベクトルの要素（一部）

mar[2] = 4　・・・ 左の余白

mar[3] = 2　・・・ 上の余白

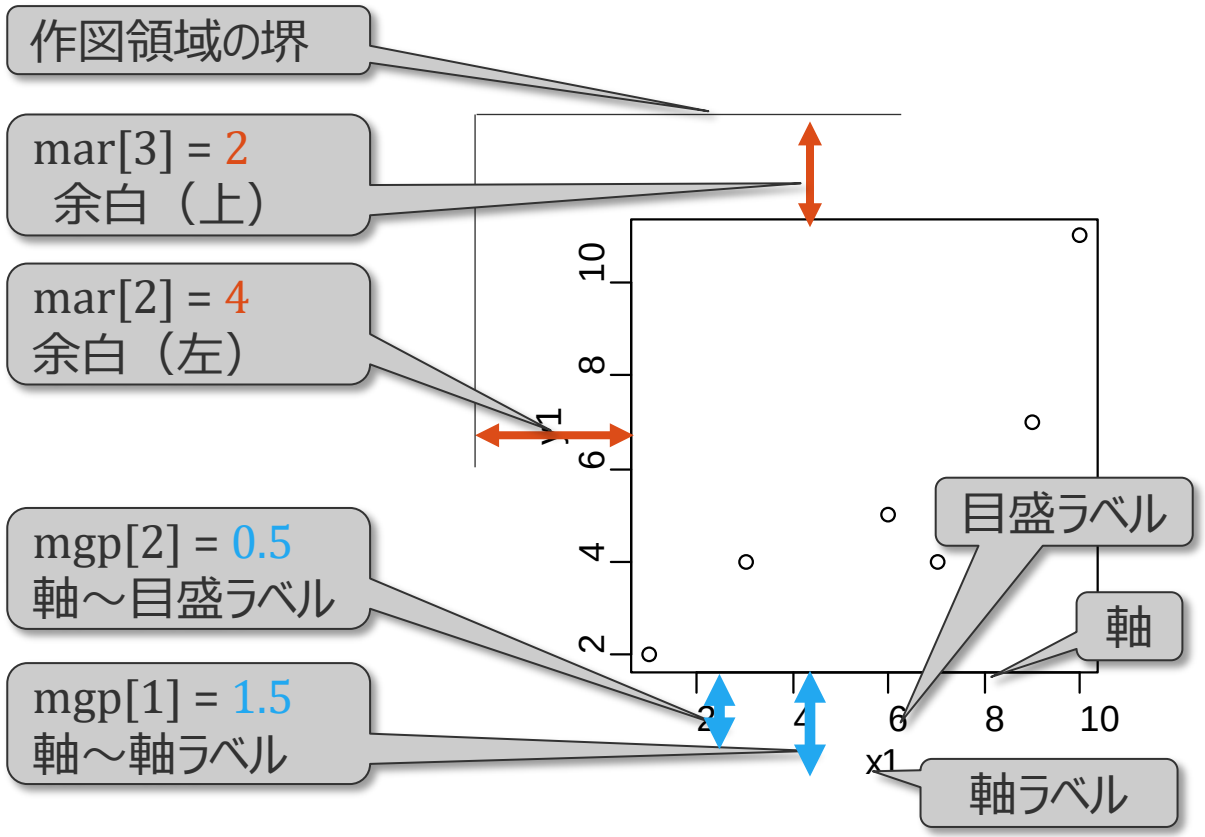
mgp[1]=1.5　・・・ 軸～軸ラベルの距離

mgp[2]=0.5　・・・ 軸～目盛ラベルの距離

単位は「行」、小数点のある値でも可

```
39 # (2) 余白の設定、軸の設定 ----------------------
40
41 par(mar = c(3, 4, 2, 2), mgp = c(1.5, 0.5, 0))
42 ### mar(下余白, 左余白, 上余白, 右余白)、行単位
43 ### mgp(軸～軸ラベル, 軸～目盛ラベル, 軸～軸線)
```

（my_base_graphics2.R：39-43）



関数 plot() の使い方

- 関数 plot() の使い方：(3) 典型的な例
  - (a) 関数 plot() のシンプルなコード
    - (i) plot(x1, y1) ・・・引数の名前を省略（位置引数）
      2 つの数値ベクトルの順序に注意
      先：x 軸に割付、後：y 軸に割付
    - (ii) plot(x = x1, y = y1) ・・名前付き引数

    データを渡すだけのシンプルなコード
    外観は自動設定（既定値）
    必要な情報はグラフに反映される

通常の利用方法

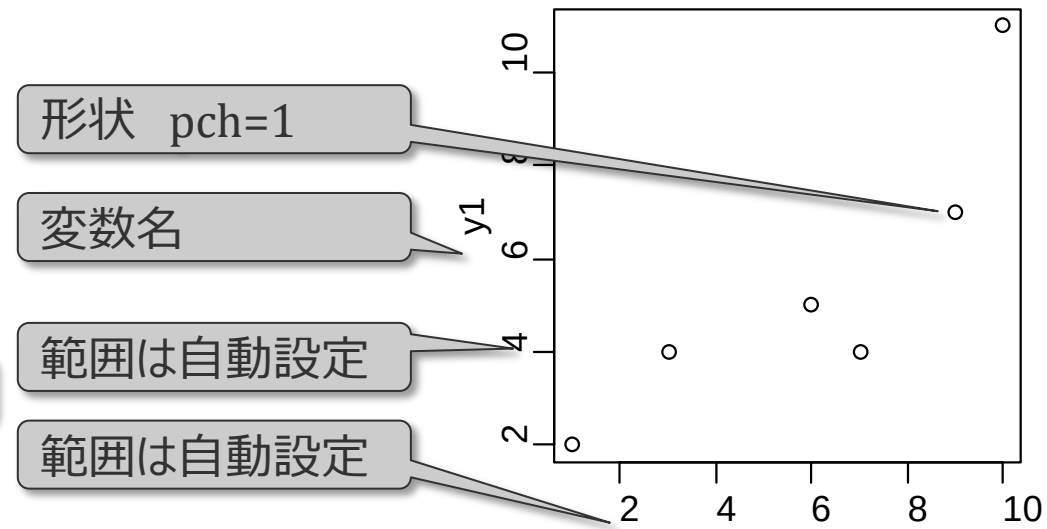
x1	y1
1	2
3	4
6	5
7	4
9	7
10	11

変数名
オブジェクト名

```
46 # (3) 関数 plot の典型的な使い方 ------
47
48 ## (a) plot() の最もシンプルなコード
49 plot(x1, y1)          # 散布図、位置引数
50 plot(x = x1, y = y1) # 散布図、名前付き引数
```

通常の利用方法

（my_base_graphics2.R：46-50）



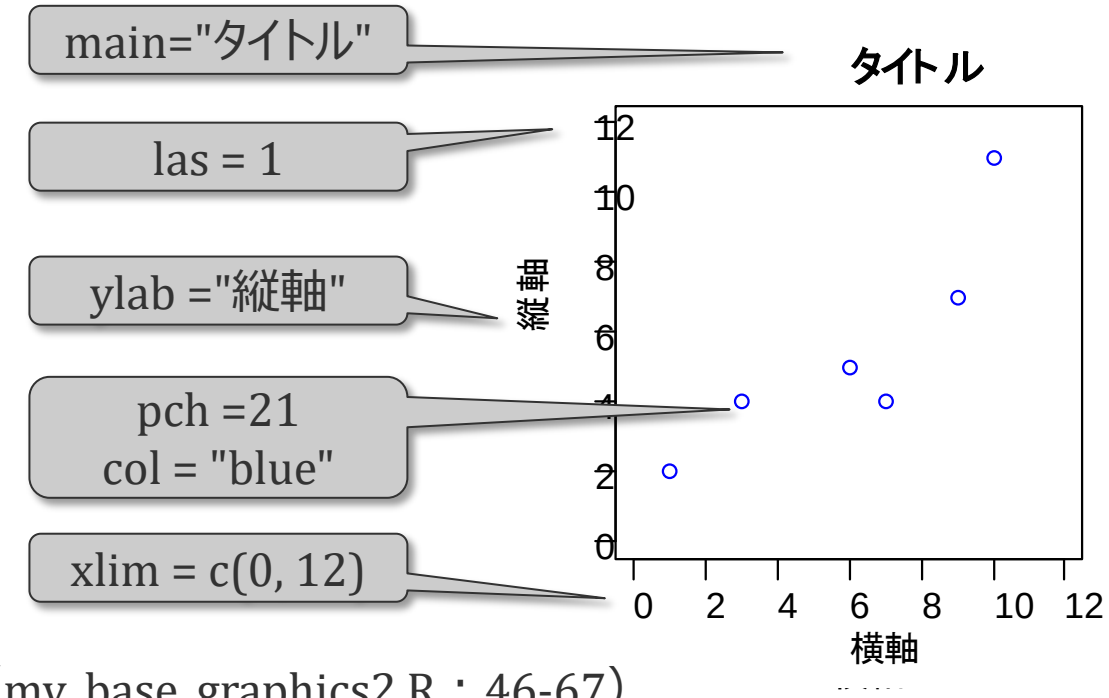
関数 plot() の使い方

●関数 plot() の使い方：(3) 典型的な例

(b) 引数によるカスタマイズと低水準関数との組合せ

```
46  # (3) 関数 plot の典型的な使い方 ----------------
47
48  ## (a) plot() の最もシンプルなコード
49  plot(x1, y1)          # 散布図、位置引数
50  plot(x = x1, y = y1) # 散布図、名前付き引数
51
52  ## (b) 引数によるカスタマイズと低水準関数との組合せ
53  plot(                 # 高水準関数、散布図
54    x1, y1,             # 数値ベクトル(対応あり)
55    xlim = c(0, 12),    # x 軸の範囲の指定
56    ylim = c(0, 12),    # y 軸の範囲の指定
57    main = "タイトル", # タイトルの追加
58    xlab = "横軸",      # x軸の軸ラベルの追加
59    ylab = "縦軸",      # y軸の軸ラベルの追加
60    pch = 21,           # マーカーの種類の指定
61    col = "blue",       # マーカーの色指定
62    las = 1)            # 目盛ラベルの方向
63
64  lm_out <- lm(y1~x1)   # 回帰分析,線形モデルオブジェク
65
66  abline(lm_out,        # 低水準関数
67         col = "blue") # 回帰直線の追加
```

●、○、■、□などの名称
ここでは、「記号」を主として使用
シンボル、マーク、マーカー、



（my_base_graphics2.R：46-67）

関数 plot() の使い方

- 関数 plot() の使い方：(3) 典型的な例
  - (b) 引数によるカスタマイズと低水準関数との組合せ
    - 高水準関数 plot() と低水準関数 abline() による回帰直線の追加
    - 次の高水準関数の実行まで、低水準関数による追加が可能

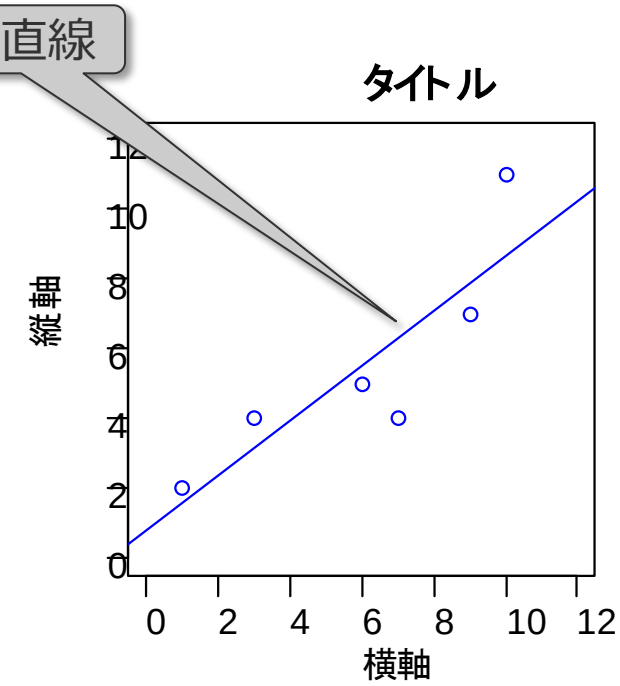
```
52 ## (b) 引数によるカスタマイズと低水準関数との組合せ
53 plot(                  # 高水準関数、散布図
54   x1, y1,              # 数値ベクトル(対応あり)
55   xlim = c(0, 12),     # x 軸の範囲の指定
56   ylim = c(0, 12),     # y 軸の範囲の指定
57   main = "タイトル",   # タイトルの追加
58   xlab = "横軸",       # x軸の軸ラベルの追加
59   ylab = "縦軸",       # y軸の軸ラベルの追加
60   pch = 21,            # マーカーの種類の指定
61   col = "blue",        # マーカーの色指定
62   las = 1)             # 目盛ラベルの方向
63
64 lm_out <- lm(y1~x1)    # 回帰分析,線形モデルオブジェクト
65
66 abline(lm_out,         # 低水準関数
67        col = "blue")   # 回帰直線の追加
```

（my_base_graphics2.R：52-67）

高水準関数 plot()

回帰分析

低水準関数 abline()



関数 plot() の使い方

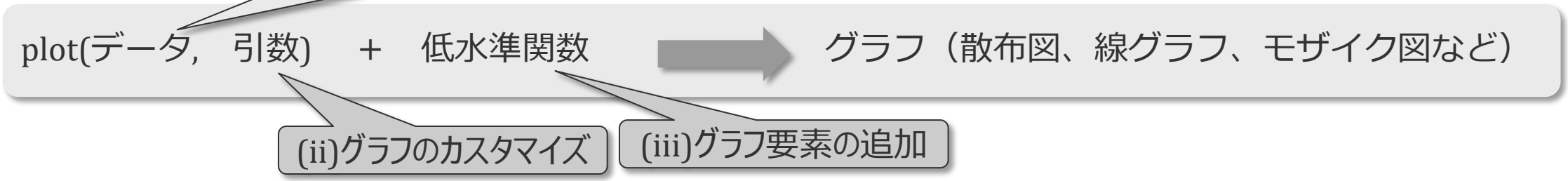
●関数 plot() の使い方：(3) 典型的な例

(i) 関数 plot() に渡すデータの種類に対応
　　データの種類（クラス）に応じてグラフを選択
　　（ジェネリックス関数、plot() の特徴）

(ii) 関数 plot() の引数に値を渡してカスタマイズ
　　col（色）、lty（線種）、xlab（x 軸ラベル）など

(iii) 関数 plot() と低水準関数の組合せ
　　回帰直線、凡例、注釈など
　　グラフ要素の追加

(i) 関数 plot() に渡すデータ
- 因子ベクトル、1 個、2 個
- 数値ベクトル、1 個、2 個
- 因子ベクトル 1 個 ＋ 数値ベクトル 1 個
- データフレーム
- マトリックス（行列）
- 時系列オブジェクト（ts 型オブジェクト）
- 線形モデルオブジェクト（lm 型オブジェクト）
- 関数オブジェクト（function 型オブジェクト）
- 分割表オブジェクト（table 型オブジェクト）



関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(4) 因子ベクトル１個（ 棒グラフ）

１つの因子ベクトル（因子変数）を集計した結果として棒グラフを得る

plot(f)・・・f：因子ベクトル

水準数２→ ２本の棒グラフ、水準数３ → ３本の棒グラフ

データフレームの１列（因子ベクトル）の場合

plot(df$f)

```
70 # (4) 因子ベクトル 1 個 ------------------------
71
72 plot(answer)                # 棒グラフ
73
74 tb2 <- table(answer)        # テーブルオブジェクト
75 print(tb2)
```

（my_base_graphics2.R：70-75）

度数表

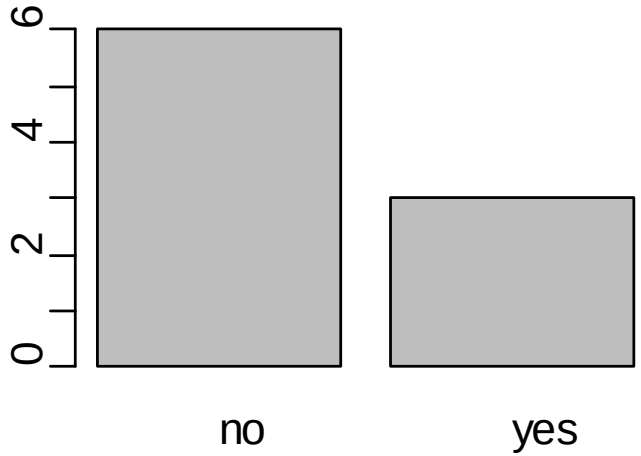
```
> print(tb2)
answer
 no yes
  6   3
```

（Console の表示）

answer
yes
no
no
yes
no
no
yes
no
no

集計

(4) 棒グラフ



関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(5) 数値ベクトル１個（インデックスプロット）

１つの数値ベクトル（数値変数）を渡すと、
行番号（インデックス）を x 軸に割当て、
数値変数を y 軸に割当てたグラフを得る
→インデックスプロット

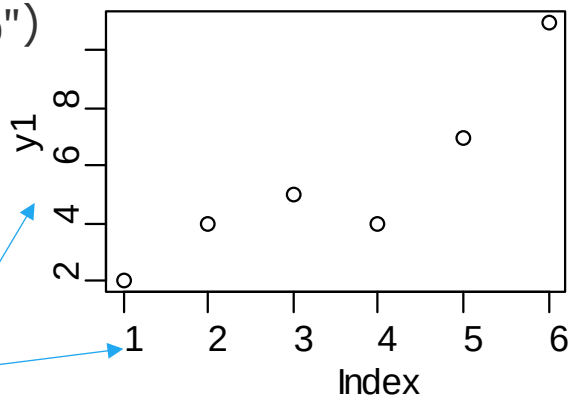
plot(y1) ・・・plot(y1, type = "p")

既定値
省略可

```
78  # (5) 数値ベクトル 1 個（インデックスプロット）
79
80  plot(y1)                  # 点グラフ("p",既定値)
81  plot(y1, type = "h")      # 垂線
82  plot(y1, type = "l")      # 線グラフ
83  plot(y1, type = "c")      # 線グラフ(線分)
84  plot(y1, type = "o")      # 線グラフ(点の上に線)
85  plot(y1, type = "b")      # 線グラフ(点と線分)
86  plot(y1, type = "s")      # 階段グラフ(水平→垂直)
87  plot(y1, type = "S")      # 階段グラフ(垂直→水平)
88  plot(y1, type = "n")      # プロットなし(枠のみ)
```

（my_base_graphics2.R：78-88）

インデックスプロット
（type = "p"）



行番号
（インデックス）

index	y1
1	2
2	4
3	5
4	4
5	7
6	11

インデックスとは
データを並べたときの
順番（行番号）

関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(5) 数値ベクトル１個（インデックスプロット）

１つの数値ベクトル（数値変数）を渡すと、
行番号（インデックス）を x 軸に割当て、
数値変数を y 軸に割当てたグラフを得る
→インデックスプロット

plot(y1) ・・・plot(y1, type = "p")

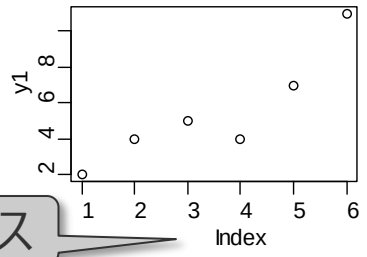
既定値
省略可

```
78  # (5) 数値ベクトル 1 個（インデックスプロット）
79
80  plot(y1)                # 点グラフ("p",既定値)
81  plot(y1, type = "h")    # 垂線
82  plot(y1, type = "l")    # 線グラフ
83  plot(y1, type = "c")    # 線グラフ(線分)
84  plot(y1, type = "o")    # 線グラフ(点の上に線)
85  plot(y1, type = "b")    # 線グラフ(点と線分)
86  plot(y1, type = "s")    # 階段グラフ(水平→垂直)
87  plot(y1, type = "S")    # 階段グラフ(垂直→水平)
88  plot(y1, type = "n")    # プロットなし(枠のみ)
```

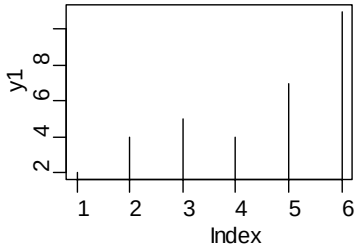
（my_base_graphics2.R：78-88）

インデックス

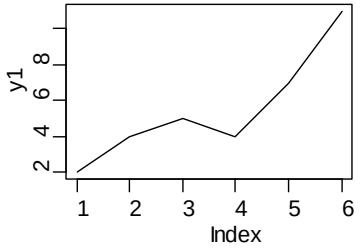
type = "p"（既定値）



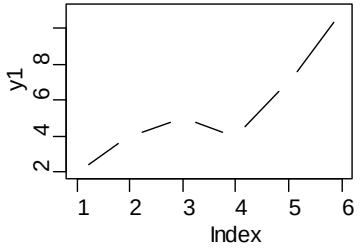
type = "h"



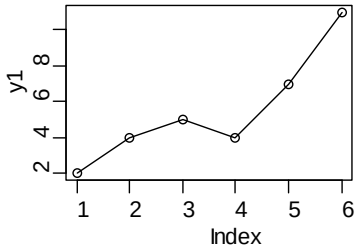
type = "l"



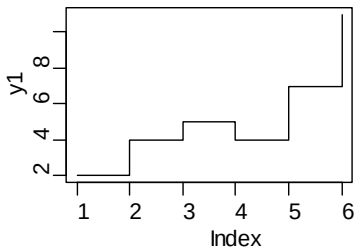
type = "c"



type = "o"



type = "s"



関数 plot() の使い方：データの種類

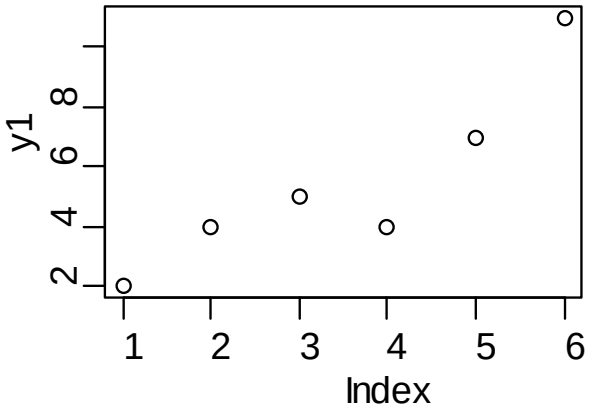
●関数 plot() に渡すデータ：(5) 数値ベクトル１個（インデックスプロット）

１つの数値ベクトル（数値変数）を渡すと、
行番号（インデックス）を x 軸、
数値変数を y 軸にプロットしたグラフを得る
→インデックスプロット

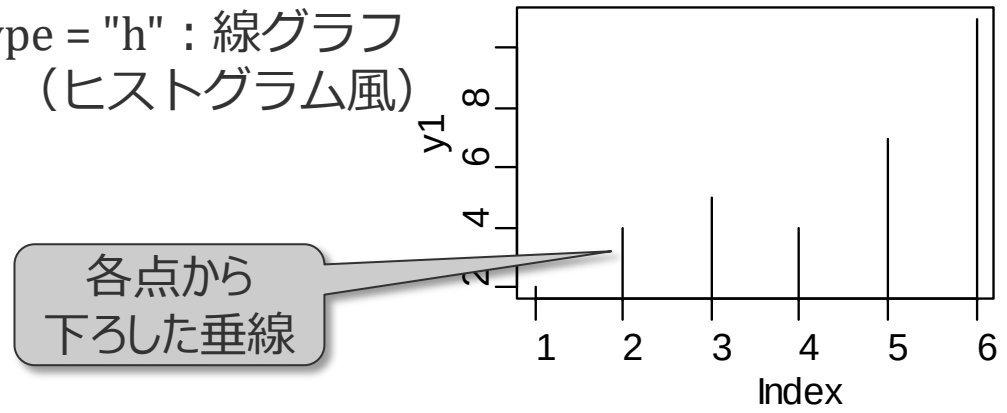
```
78  # (5) 数値ベクトル 1 個（インデックスプロット）
79
80  plot(y1)                  # 点グラフ("p",既定値)
81  plot(y1, type = "h")      # 垂線
82  plot(y1, type = "l")      # 線グラフ
83  plot(y1, type = "c")      # 線グラフ(線分)
84  plot(y1, type = "o")      # 線グラフ(点の上に線)
85  plot(y1, type = "b")      # 線グラフ(点と線分)
86  plot(y1, type = "s")      # 階段グラフ(水平→垂直)
87  plot(y1, type = "S")      # 階段グラフ(垂直→水平)
88  plot(y1, type = "n")      # プロットなし(枠のみ)
```

（my_base_graphics2.R：78-88）

type = "p"　散布図
（既定値、省略可）



type = "h"：線グラフ
（ヒストグラム風）



関数 plot() の使い方：データの種類

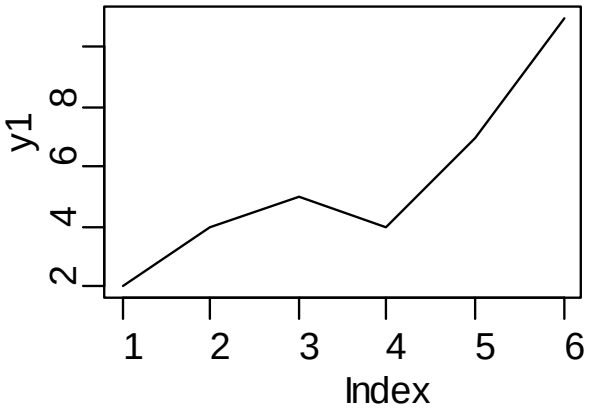
●関数 plot() に渡すデータ：(5) 数値ベクトル１個（インデックスプロット）

１つの数値ベクトル（数値変数）を渡すと、
行番号（インデックス）を x 軸、
数値変数を y 軸にプロットしたグラフを得る
→インデックスプロット

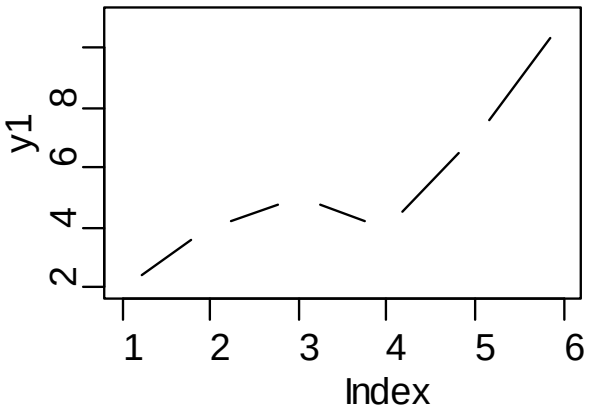
```
78  # (5) 数値ベクトル 1 個（インデックスプロット）
79
80  plot(y1)                # 点グラフ("p",既定値)
81  plot(y1, type = "h")    # 垂線
82  plot(y1, type = "l")    # 線グラフ
83  plot(y1, type = "c")    # 線グラフ(線分)
84  plot(y1, type = "o")    # 線グラフ(点の上に線)
85  plot(y1, type = "b")    # 線グラフ(点と線分)
86  plot(y1, type = "s")    # 階段グラフ(水平→垂直)
87  plot(y1, type = "S")    # 階段グラフ(垂直→水平)
88  plot(y1, type = "n")    # プロットなし(枠のみ)
```

（my_base_graphics2.R：78-88）

type = "l"　線グラフ



type = "c"：線グラフ（線分）



関数 plot() の使い方：データの種類

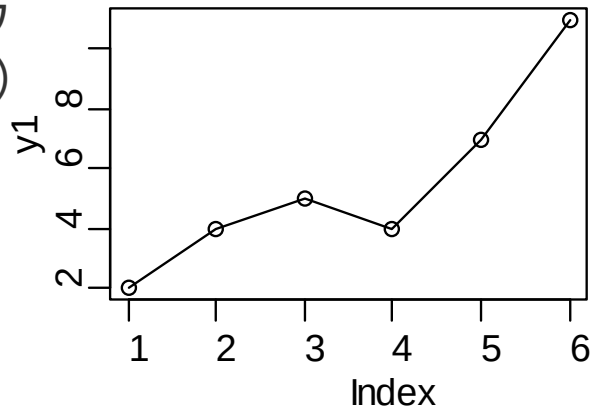
●関数 plot() に渡すデータ：(5) 数値ベクトル１個（インデックスプロット）

１つの数値ベクトル（数値変数）を渡すと、
行番号（インデックス）を x 軸、
数値変数を y 軸にプロットしたグラフを得る
→インデックスプロット

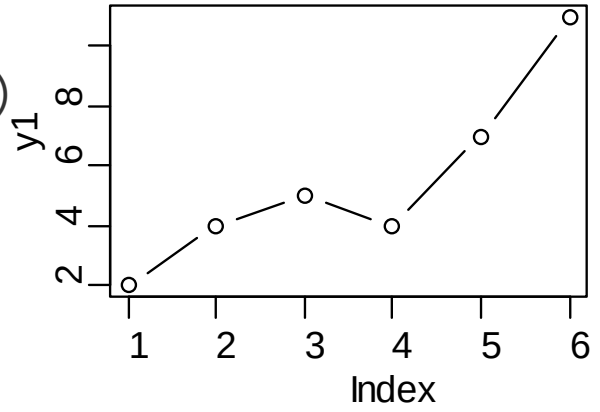
```
78  # (5) 数値ベクトル 1 個（インデックスプロット）
79
80  plot(y1)                # 点グラフ("p",既定値)
81  plot(y1, type = "h")    # 垂線
82  plot(y1, type = "l")    # 線グラフ
83  plot(y1, type = "c")    # 線グラフ(線分)
84  plot(y1, type = "o")    # 線グラフ(点の上に線)
85  plot(y1, type = "b")    # 線グラフ(点と線分)
86  plot(y1, type = "s")    # 階段グラフ(水平→垂直)
87  plot(y1, type = "S")    # 階段グラフ(垂直→水平)
88  plot(y1, type = "n")    # プロットなし(枠のみ)
```

（my_base_graphics2.R：78-88）

type = "o"　線グラフ
（点の上に線）



type = "b"：線グラフ
（点と線分）



関数 plot() の使い方：データの種類

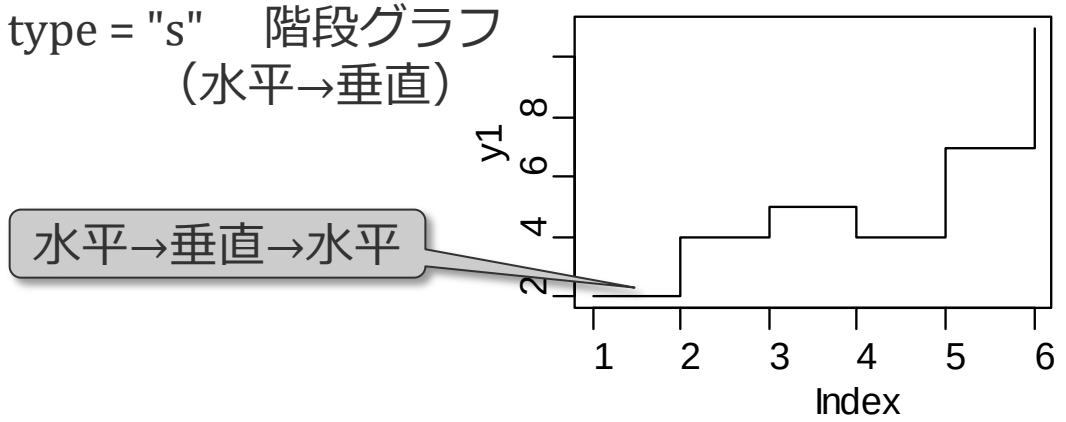
●関数 plot() に渡すデータ：(5) 数値ベクトル１個（インデックスプロット）

１つの数値ベクトル（数値変数）を渡すと、
行番号（インデックス）を x 軸、
数値変数を y 軸にプロットしたグラフを得る
→ インデックスプロット

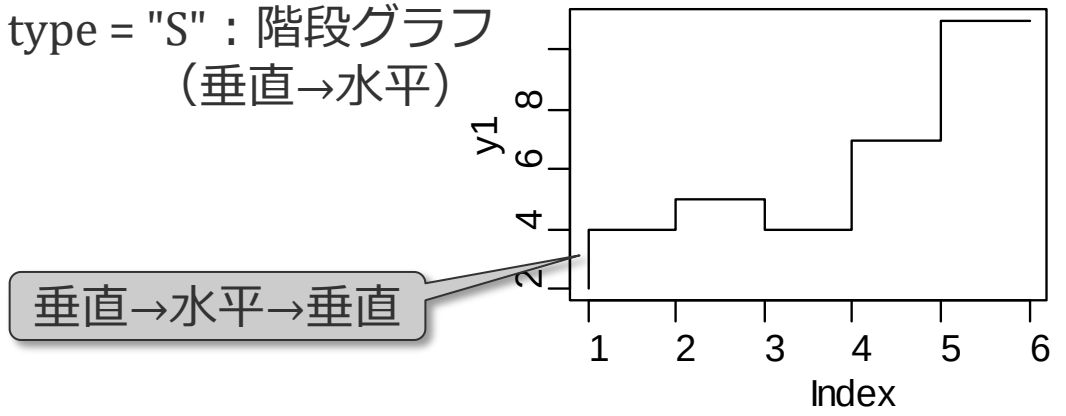
```
78  # (5) 数値ベクトル 1 個（インデックスプロット）
79
80  plot(y1)                # 点グラフ("p",既定値)
81  plot(y1, type = "h")    # 垂線
82  plot(y1, type = "l")    # 線グラフ
83  plot(y1, type = "c")    # 線グラフ(線分)
84  plot(y1, type = "o")    # 線グラフ(点の上に線)
85  plot(y1, type = "b")    # 線グラフ(点と線分)
86  plot(y1, type = "s")    # 階段グラフ(水平→垂直)
87  plot(y1, type = "S")    # 階段グラフ(垂直→水平)
88  plot(y1, type = "n")    # プロットなし(枠のみ)
```

（my_base_graphics2.R：78-88）

type = "s"　階段グラフ
（水平→垂直）



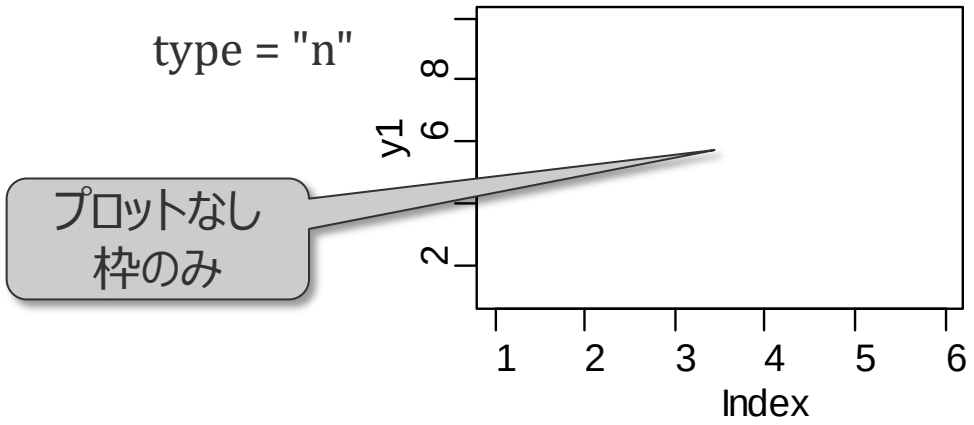
type = "S"：階段グラフ
（垂直→水平）



関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(5) 数値ベクトル１個（インデックスプロット）

１つの数値ベクトル（数値変数）を渡すと、
行番号（インデックス）を x 軸、
数値変数を y 軸にプロットしたグラフを得る
→インデックスプロット



type = "n" のグラフの場合、
この空のグラフにグラフ要素を追加して
グラフを描く

```
78  # (5) 数値ベクトル 1 個（インデックスプロット）
79
80  plot(y1)                # 点グラフ("p",既定値)
81  plot(y1, type = "h")    # 垂線
82  plot(y1, type = "l")    # 線グラフ
83  plot(y1, type = "c")    # 線グラフ(線分)
84  plot(y1, type = "o")    # 線グラフ(点の上に線)
85  plot(y1, type = "b")    # 線グラフ(点と線分)
86  plot(y1, type = "s")    # 階段グラフ(水平→垂直)
87  plot(y1, type = "S")    # 階段グラフ(垂直→水平)
88  plot(y1, type = "n")    # プロットなし(枠のみ)
```

（my_base_graphics2.R：78-88）

関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(6) 因子ベクトル２個（スパインプロット）

２つの因子ベクトルを渡すとスパインプロットを得る　→　spineplot() の項で説明

plot(gender, answer)、 plot(answer ~ gender)

```
23
24  ## (c) 因子ベクトルとテーブルオブジェクト
25  ##      原因：Male/Female → 結果：yes/no、
26  gender <- factor(
27    rep(c("male", "female"), times = c(4, 5)),
28    levels = c("male", "female")
29    )
30  answer <- factor(
31    c("yes", "no",  "no", "yes",
32      "no",  "no", "yes",  "no",  "no"),
33    levels = c("no", "yes")
34    )
```

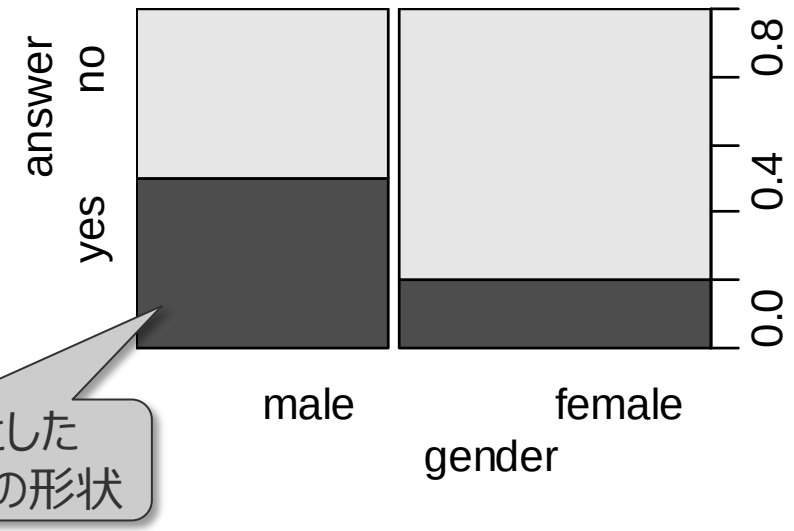
順序

順序

```
91  # (6) 因子ベクトル 2 個  ------------------------
92
93  plot(gender, answer)     # (a)  スパインプロット
94  plot(answer ~ gender)    # (a') スパインプロット
95              # gender：説明変数、answer：応答変数
```

（my_base_graphics2.R：91-95）

(6) スパインプロット



縦軸を割合とした
積上げ棒グラフの形状

関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(7) 数値ベクトル２個（散布図）

２つの数値ベクトル（数値変数）を渡すと、
それぞれ x 軸と y 軸にプロットした散布図を得る

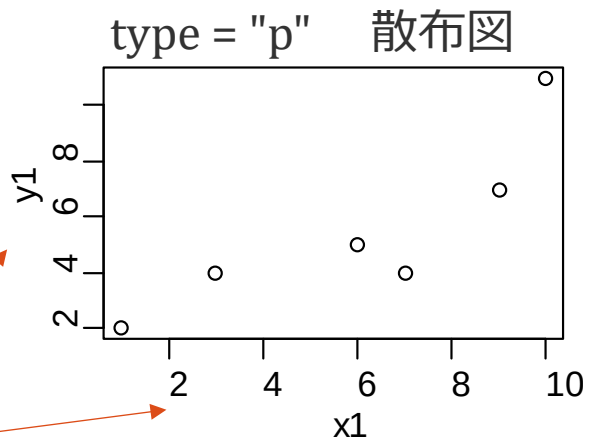
plot(x1, y1)　plot(y1 ～ x1)　type = "p" は省略可

データフレームの場合（後述）

plot(dfx, dfy)　plot(y ~ x, data = df)

```
 97 # (7) 数値ベクトル 2 個 ------------------------
 98
 99 plot(x1, y1)              # 散布図("p", 既定値)
100 plot(y1 ~ x1)             # 散布図
101
102 plot(x1, y1, type = "h") # 垂線
103 plot(x1, y1, type = "l") # 線グラフ
104 plot(x1, y1, type = "c") # 線グラフ(線分)
105 plot(x1, y1, type = "o") # 線グラフ(点の上に線)
106 plot(x1, y1, type = "b") # 線グラフ(点と線分)
107 plot(x1, y1, type = "s") # 階段グラフ(水平→垂直)
108 plot(x1, y1, type = "S") # 階段グラフ(垂直→水平)
109 plot(x1, y1, type = "n") # プロットなし
```

（my_base_graphics2.R：97-109）



x1	y1
1	2
3	4
6	5
7	4
9	7
10	11

対応あり

関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(7) 数値ベクトル２個 （棒グラフ、線グラフなど）

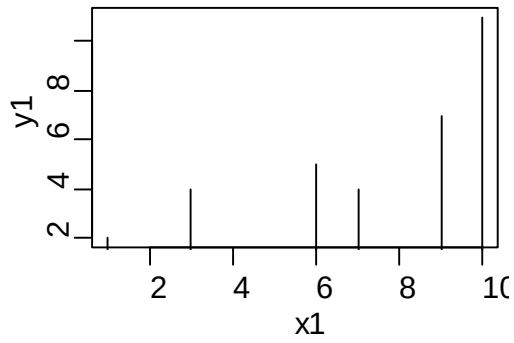
２つの数値ベクトル（数値変数）を渡すと、
それぞれ x 軸と y 軸にプロットした散布図を得る

plot(x1, y1, type = "h")　plot(y1 ～ x1, type = "h")

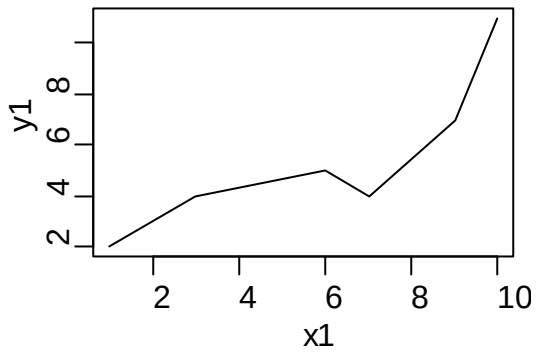
```
 97 # (7) 数値ベクトル 2 個 ------------------------
 98
 99 plot(x1, y1)              # 散布図("p", 既定値)
100 plot(y1 ~ x1)             # 散布図
101
102 plot(x1, y1, type = "h") # 垂線
103 plot(x1, y1, type = "l") # 線グラフ
104 plot(x1, y1, type = "c") # 線グラフ(線分)
105 plot(x1, y1, type = "o") # 線グラフ(点の上に線)
106 plot(x1, y1, type = "b") # 線グラフ(点と線分)
107 plot(x1, y1, type = "s") # 階段グラフ(水平→垂直)
108 plot(x1, y1, type = "S") # 階段グラフ(垂直→水平)
109 plot(x1, y1, type = "n") # プロットなし
```

（my_base_graphics2.R：97-109）

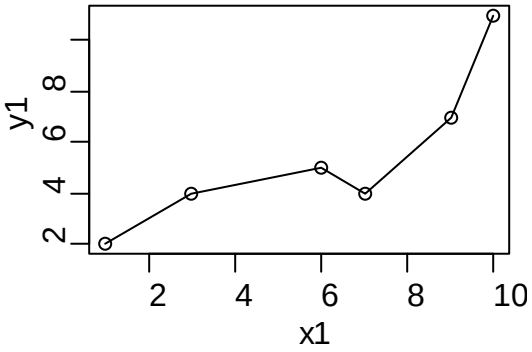
棒グラフ（"h"）



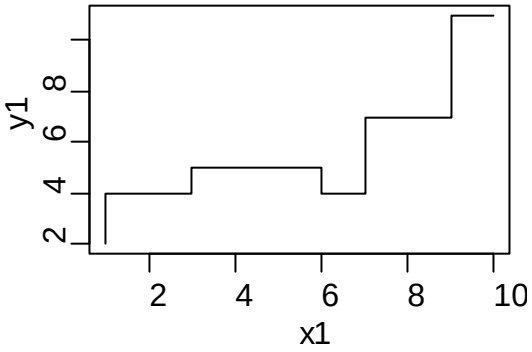
線グラフ（"l"）



線グラフ（ "o"）



階段グラフ("S")



関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(8) 因子ベクトル１個、数値ベクトル１個

(a) 数値ベクトル～因子ベクトル　→　箱ひげ図
(a') 因子ベクトル，　数値ベクトル　→　箱ひげ図
(b) 因子ベクトル～数値ベクトル　→　スピノグラム
(c) 数値ベクトル，　因子ベクトル　→　１次元散布図

(a) (a') データフレームの場合（右表）

plot(obs ～ grp, data = df)　　plot(dfgrp, dfobs)

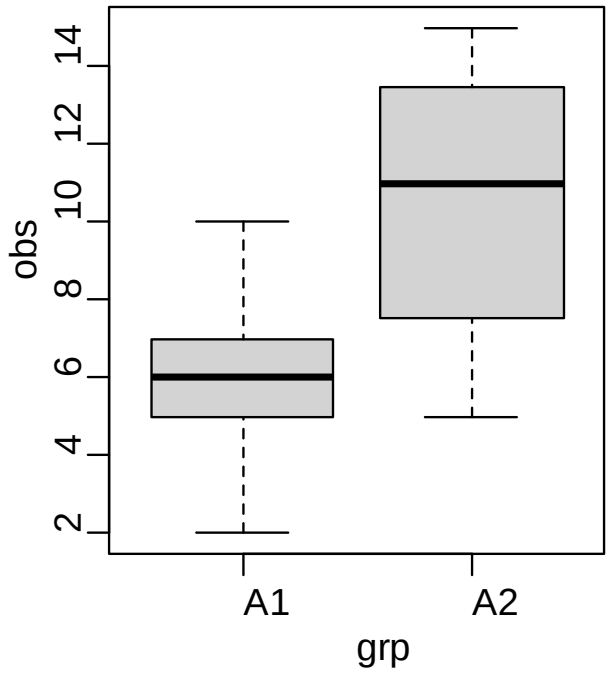
```
112 # (8) 因子ベクトル 1 個、数値ベクトル 1 個 ------
113
114 plot(obs ~ grp, data =df)     # (a) 箱ひげ図
115 plot(df$grp, df$obs)          # (a')箱ひげ図
116
117 plot(obs ~ grp, data =df)     # (b) スピノグラム
118
119 plot(df$obs, df$grp)          # (c) １次元散布図
```

（my_base_graphics2.R：112-119）

df

grp	obs
A1	7
A1	5
A1	6
A1	2
A1	10
A2	4
A2	15
A2	12
A2	10

(a) (a') 箱ひげ図



関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(8) 因子ベクトル１個、数値ベクトル１個

(a) 数値ベクトル～因子ベクトル　→　箱ひげ図

(a') 因子ベクトル，　数値ベクトル　→　箱ひげ図

(b) 因子ベクトル～数値ベクトル　→　スピノグラム

(c) 数値ベクトル，　因子ベクトル　→　１次元散布図

(b) データフレームの場合（右表）

plot(g ～ y, data = df)

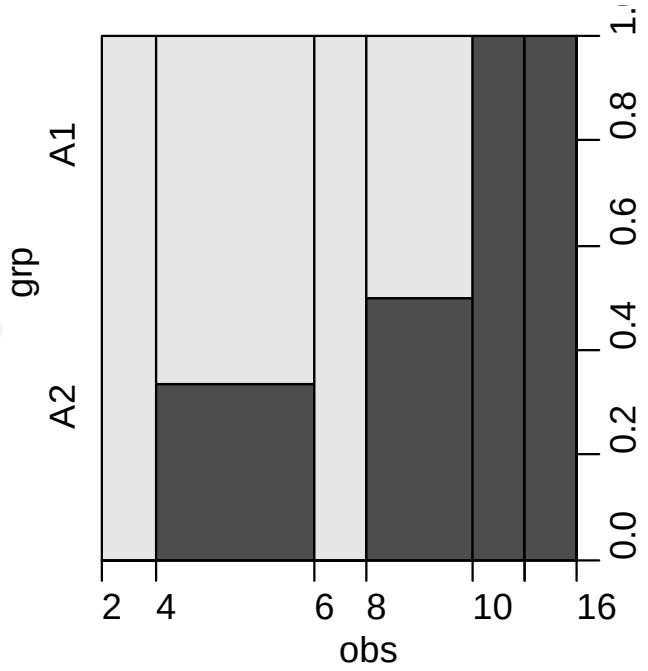
```
112 # (8) 因子ベクトル 1 個、数値ベクトル 1 個 ------
113
114 plot(obs ~ grp, data =df)     # (a) 箱ひげ図
115 plot(df$grp, df$obs)         # (a')箱ひげ図
116
117 plot(obs ~ grp, data =df)     # (b) スピノグラム
118
119 plot(df$obs, df$grp)         # (c) １次元散布図
```

（my_base_graphics2.R：112-119）

df

grp	obs
A1	7
A1	5
A1	6
A1	2
A1	10
A2	4
A2	15
A2	12
A2	10

(b) スピノグラム



関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(8) 因子ベクトル１個、数値ベクトル１個

(a) 数値ベクトル～因子ベクトル　→　箱ひげ図

(a') 因子ベクトル，　数値ベクトル　→　箱ひげ図

(b) 因子ベクトル～数値ベクトル　→　スピノグラム

(c) 数値ベクトル，　因子ベクトル　→　１次元散布図

(c) データフレームの場合（右表）

plot(dfy, dfg)

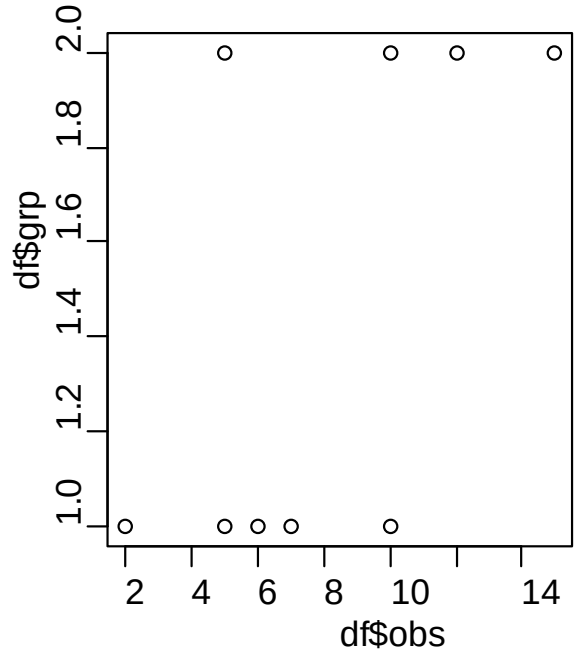
```
112 # (8) 因子ベクトル 1 個、数値ベクトル 1 個 ------
113
114 plot(obs ~ grp, data =df)     # (a) 箱ひげ図
115 plot(df$grp, df$obs)         # (a')箱ひげ図
116
117 plot(obs ~ grp, data =df)     # (b) スピノグラム
118
119 plot(df$obs, df$grp)         # (c) １次元散布図
```

（my_base_graphics2.R：112-119）

df

grp	obs
A1	7
A1	5
A1	6
A1	2
A1	10
A2	4
A2	15
A2	12
A2	10

(c) １次元散布図



関数 plot() の使い方：データの種類

- 関数 plot() に渡すデータ：(9) 複数の数値ベクトル（散布図行列）

  複数の数値ベクトルは、１つのデータフレームにまとめて plot() に渡す

  （３つの数値ベクトルをそれぞれ渡すことは不可）

  散布図行列（総当たりの散布図）を得る

plot(df3)

yy1, yy2, yy3
対応がある
数値データ

yy1	yy2	yy3
1	11	12
3	15	10
…	…	…
7	21	3
10	25	3

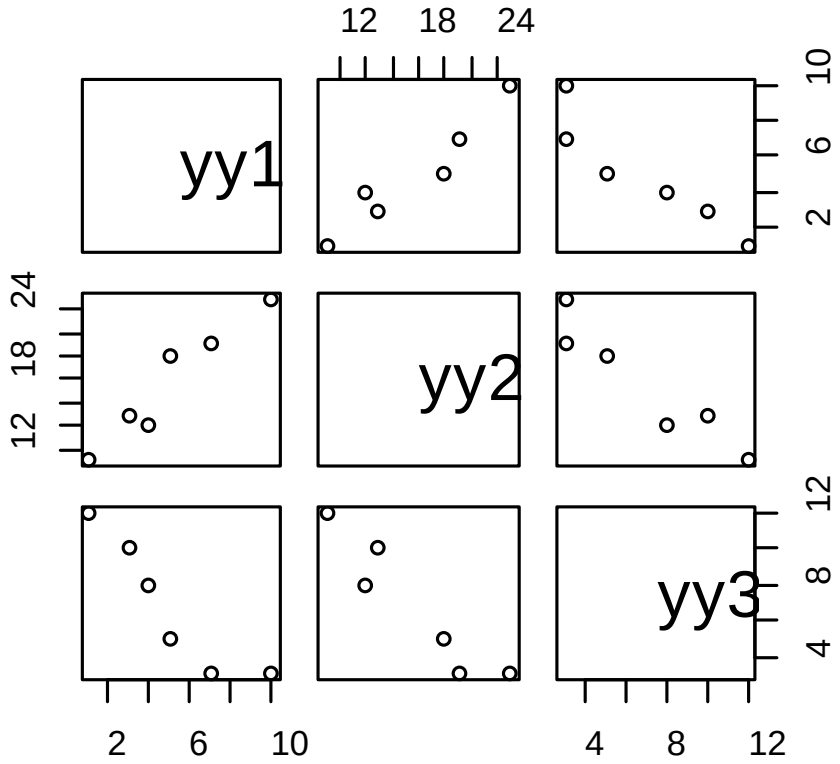
df3

yy1	yy2	yy3
1	11	12
3	15	10
…	…	…
7	21	3
10	25	3

```
122 # (9) 複数の数値ベクトル → データフレーム -------
123
124 yy1 <- c( 1, 3, 4, 5, 7,10)  # 対応のあるデータ
125 yy2 <- c(11,15,14,20,21,25)
126 yy3 <- c(12,10, 8, 5, 3, 3)
127 df3 <- data.frame(yy1, yy2, yy3)
128
129 plot(df3)                    # 散布図行列
```

（my_base_graphics2.R：122-129）

(9) 散布図行列



関数 plot() の使い方：データの種類

- 関数 plot() に渡すデータ：(10) データフレーム（記述方法）
  - (i) 「データフレーム名 $ 列名」の利用

    ```
    plot(df$x, df$y)
    ```

    最も明示的で、どのデータフレームの列かが一目で分かる
    コードが冗長になりやすい、オブジェクトを短い名称（df など）に変更
  - (ii) 関数 with() の利用　（推奨）

    ```
    with(df, plot(x, y))
    ```

    データフレームを一度だけ指定すれば、（）中で列名をそのまま使える
    一時的に環境を切り替える仕組み
  - (iii) 引数 data の利用　（推奨）

    ```
    plot(y ~ x, data = df)
    ```

    formula インターフェイスでデータを指定、簡便な記述方法
    使える関数は限られる（plot(), boxplot(), stripchart(), coplot(), mosaicplot() など）
  - (iv) 関数 attach()、detach()（非推奨）

    ```
    attach(df)
    plot(x, y)
    detach(df)
    ```

    データフレームの列を直接オブジェクト名として参照可能
    不具合の発生が懸念されるため非推奨（他人のスクリプトを読むための知識）

関数 plot() の使い方：データの種類

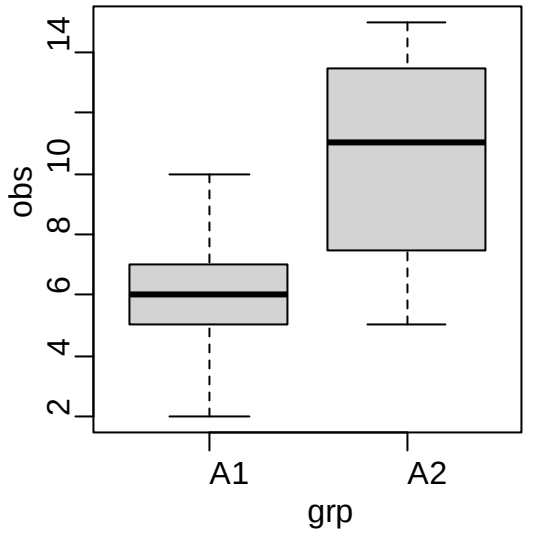
- 関数 plot() に渡すデータ：(10) データフレーム（記述方法）
  - (a) 「データフレーム名 $ 列名」の利用
  - (b) 関数 with() の利用
  - (c) 引数 data の利用

```
132 # (10) データフレーム --------------------------
133
134 ## (a) モザイク図、「データフレーム$列名」の利用
135 plot(df$grp, df$obs)
136
137 ## (b) モザイク図、関数 with() の利用
138 with(df, {
139   plot(grp, obs)
140   abline(h = mean(obs), col = "red")
141 })
142
143 with(df, plot(grp, obs))
144
145 ## (c) モザイク図、引数 data の利用
146 plot(obs ~ grp, data = df)
```

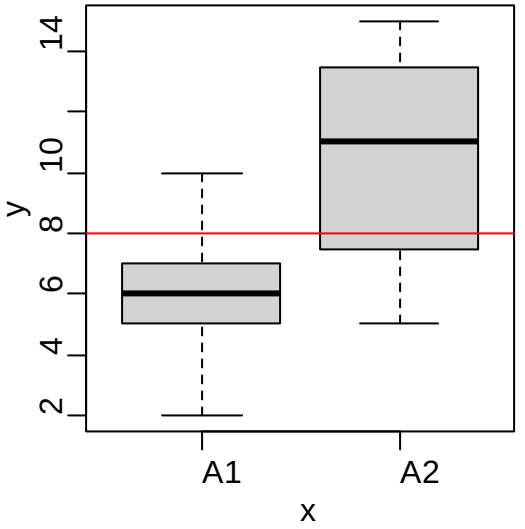
関数 with の { } 内に、
複数行のコードを記述

（my_base_graphics2.R：132-146）

(a)
(c)



(b)



関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(11) 時系列オブジェクト（ts 型オブジェクト）

時系列オブジェクト（ts：Time Series）：関数 ts() などで作成された時系列データを付値

関数 ts() の引数の使い方

data：数値ベクトルまたはマトリックス

start：開始時点（例：c(2020, 1) は 2020年1月）

end ：（任意）終了時点。必要な場合のみ指定

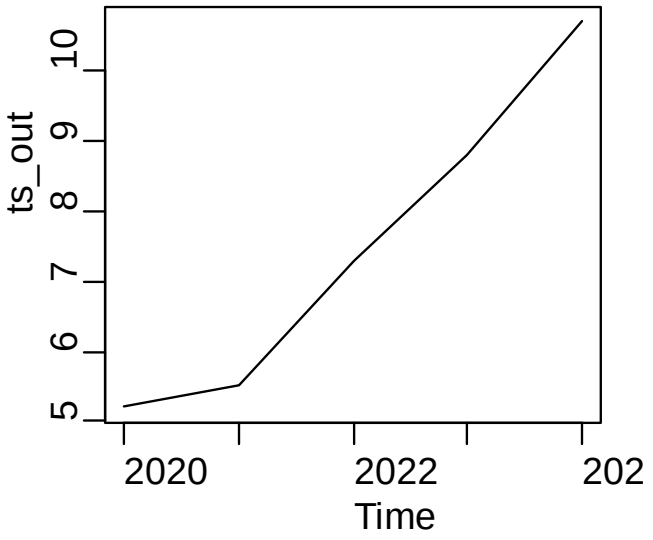
frequency ：年間の観測回数、12(毎月)、4(四半期)、1(年次) 等

横軸に時間、縦軸に値、時間経過に伴うデータの変動を可視化

```
149 # (11) 時系列オブジェクト(ts型オブジェクト) -----
150
151 ts_out <- ts(c(5.2, 5.5, 7.3, 8.8, 10.7),
152               start = c(2020, 1), frequency = 1)
153
154 plot(ts_out)                # 時系列グラフ
```

（my_base_graphics2.R：149-154）

(11) 時系列グラフ



関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(12) 線形モデルオブジェクト（lm 型オブジェクト）

線形モデルオブジェクト（lm：Linear Model）：関数 lm() による回帰分析の結果を付値

関数 lm() の使い方

lm_out <- lm(y ~ x)　y：目的変数、x：説明変数

残差診断図

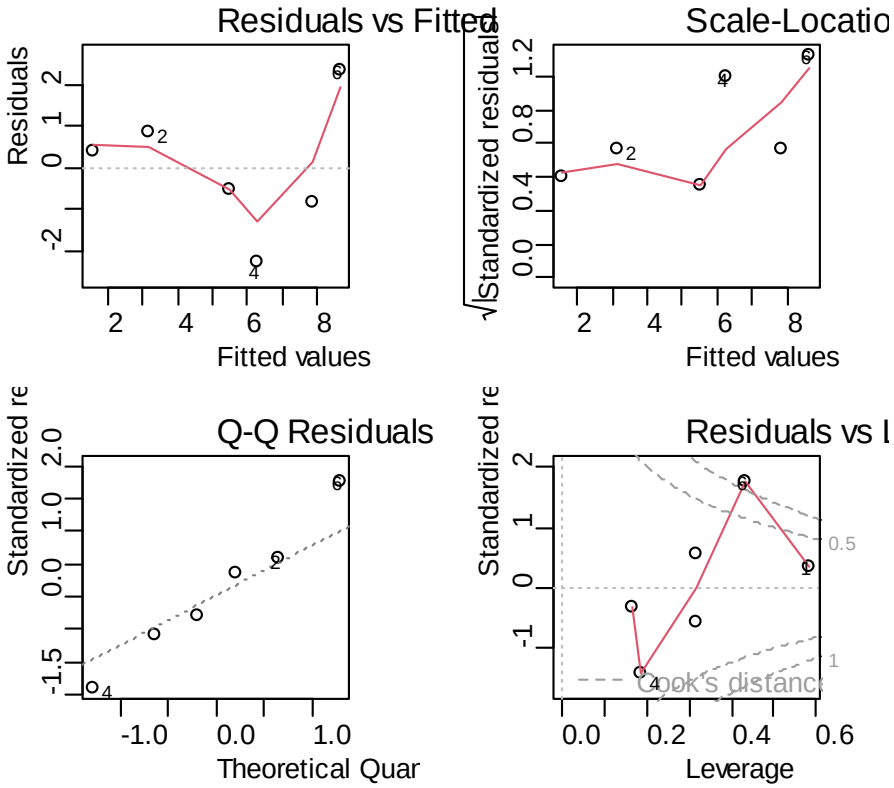
モデルの当てはまり、誤差の正規性や等分散性を可視化

「4.5　回帰分析適用上の諸問題　Rの補足」参照

```
157  # (12) 線形モデルオブジェクト(lm型オブジェクト)--
158  #        （マルチパネル、ファセット）
159
160  lm_out <- lm(y1 ~ x1)     # 回帰分析
161
162  par(mfcol = c(2, 2))      # 作図画面を４分割
163  plot(lm_out)              # 回帰分析の残差診断図
164  par(mfrow = c(1, 1))      # 画面分割を復元
```

（my_base_graphics2.R：157-164）

(12) 残差診断図



関数 plot() の使い方：データの種類

- 関数 plot() に渡すデータ：(13) 関数オブジェクト（function 型オブジェクト）

  関数オブジェクト：関数 function() でユーザーが定義した関数

  sin(), cos() などの関数

  関数 function() の使い方

  例 1：　1 次回帰式の値を返す関数（引数 1 個 x）

  ```
  mod1 <- function(x)  x * 5.2 + 1.0
  ```

  mod1(5) は 27.0 を返す

  例 2：1 次回帰式の値を返す関数（引数 2 個 x, a）

  ```
  mod2  <- function(x, a)  x * (-5.2) + a
  ```

  mod2(2, 1) は 2 * (-5.2) + 1 = -9.4 を返す

関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(13) 関数オブジェクト（function 型オブジェクト）

関数 function() で mod1、mod2 を定義

(a)　mod1 <- function(x) x * 5.2 + 1.0　・・・・引数 1 つ

(b)　mod2 <- function(x, a) x * (-5.2) + a・・・引数 2 つ

関数 plot() に mod1, mod2 を渡すと関数のグラフを得る

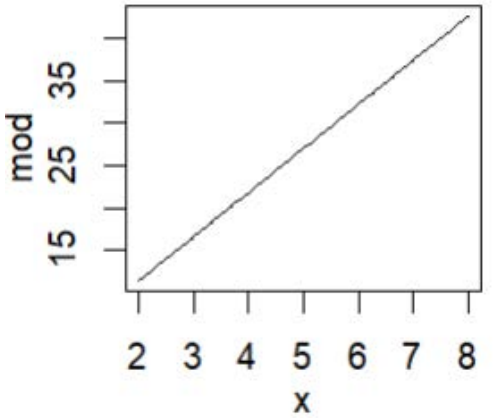
（関数 curve() を使う方法もある）

```
167  # (13) 関数オブジェクト(function型オブジェクト)--
168
169  ## (a) 関数のグラフ、引数が１つのユーザー定義関数
170  mod_1 <- function(x) x * 5.2 + 1
171
172  plot(mod_1 , from = 2, to= 8)
173
174  ## (b) 関数のグラフ、引数が２つのユーザー定義関数
175  mod_2 <- function(x, a) x * (-5.2) + a
176
177  plot(function(x) mod_2(x, a = 1),
178       from = 2, to = 8)
```

（my_base_graphics2.R：167-178）

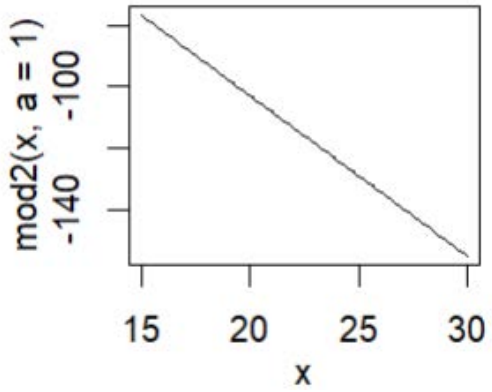
(a)引数が 1 つのユーザー定義関数

mod1



(b)引数が 2 つのユーザー定義関数

mod2



関数 plot() の使い方：データの種類

●関数 plot() に渡すデータ：(14) 分割表オブジェクト（table 型オブジェクト）

分割表オブジェクト：table() , xtabs() などで集計された
クロス集計表（分割表）

関数 table() による因子ベクトルの集計

(a) gender の集計→ tb3（度数表）→ 棒グラフ

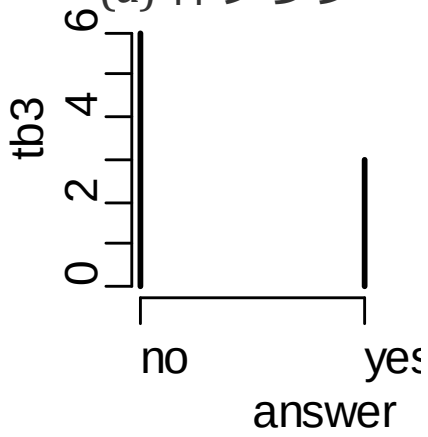
(b) gender と answer のクロス集計→tb4（分割表）→モザイク図

```
181  # (14) 分割表オブジェクト(table 型オブジェクト)--
182
183  ## (a) 1 つの因子ベクトルの集計(度数表)
184  tb3 <- table(answer)          # 分割表オブジェクト
185
186  print(tb3)    # 度数表
187  plot(tb3)     # 棒グラフ
188
189  ## (b) 2 つの因子ベクトルの集計(分割表)
190  tb4 <- table(gender, answer) # 分割表オブジェクト
191
192  print(tb4)    # 分割表
193  plot(tb4)     # モザイク図
```

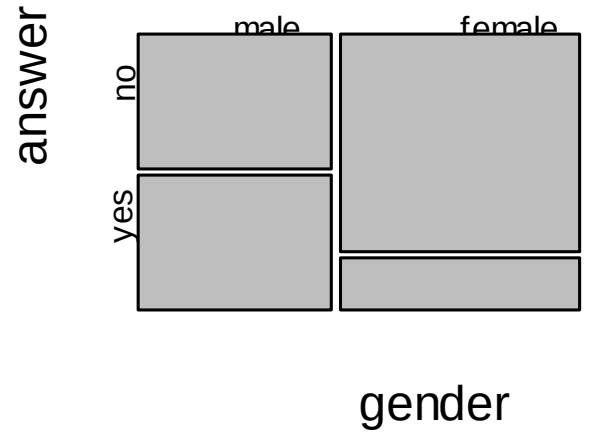
(my_base_graphics2.R：181-193)

```
> print(tb3)                  # 度数表
answer
 no yes
  6   3
> print(tb4)                  # 分割表
        answer
gender   no yes
  male    2   2
  female  4   1
```

(a) 棒グラフ



(b) モザイク図



関数 plot() の使い方：引数

- plot() 関数の引数

  関数 plot() には多くの引数がある（見るためのグラフを描画するのであれば使用は限定的）

  他の graphics 関数も同様の引数をもつ（matpliot、 barplot、 boxplot()、 stripchart() など）

主な引数	機能（引数に渡す値）	規定値
x, y	プロットするデータ	
type	プロットの種類、点、線など（"p", "l", "b", "c", "o", "h", "s", "S", "n"）	"p"
xlim, ylim	x軸・y軸の範囲、ベクトルで指定（例 xlim = c(0, 100)）	NULL
log	軸を対数目盛に変換（"", "x", "y", "xy"）	""
main, sub	メインタイトルの文字列、サブタイトルの文字列	NULL
xlab, ylab	x軸・y軸のラベルの文字列	NULL
axes	軸の表示の有無（TRUE, FALSE）	TRUE
xaxt, yaxt	軸の表示の有無（"n"を指定すると非表示、"s" は表示）	s
las	軸ラベルの方向（0：軸に平行、1：水平、2：軸に垂直、3：垂直）	0
frame.plot	プロット領域の周囲の枠線の表示の有無	axes
ann	軸のラベルとタイトルの表示（TRUE, FALSE）	par

関数 plot() の使い方：引数

- plot() 関数の引数（グラフィックスパラメータ）

  関数 plot() には多くの引数がある（見るためのグラフを描画するのであれば使用は限定的）

  他の graphics 関数も同様の引数をもつ（matpliot、barplot、boxplot()、stripchart() など）

●、○、■、□などの呼び方
ここでは「記号」を主として使用
シンボル、マーク、マーカー、

主な引数	機能（引数に渡す値）	規定値
pch	プロットする記号（シンボル、マーカー）の形、plot character（1～26）	1
col	プロットする記号の色（1～657、1："black"、2："red" など）	1
bg	pch = 21～25 の記号における塗りつぶし色（背景色）	
cex	記号や文字の大きさ（拡大率）。1 が標準、1.5 や 0.8 などで調整	
cex.axis	軸の目盛のフォントの拡大率、同様に cex.lab, cex.main, cex.sub	
lty	線種、line type（1～6）	1
lwd	線の太さ、line width（デフォルトは 1、太くしたければ 2 など）	1
font	文字のフォントスタイル	1
font.axis	軸の注釈のフォントスタイル、同様に font.lab、font.main、font.sub	
col.main	タイトルの文字列の色、同様に col.sub、col.xlab、col.ylab、col.axis	
asp	x, y の比率、aspect ratio（asp = 1 で正方形スケール）	NA

関数 plot() の使い方：引数

- 関数 plot() の引数：(15) xlim, ylim, xaxp, yaxp（軸）
  - (a) xlim, ylim：軸の範囲（最小値、最大値）を指定
  - (b) xaxp, yaxp：軸の範囲と目盛り線の位置を設定
    （xaxp と xlim、yaxp と ylim の併用を推奨、最小値と最大値を一致）

```
196 # (15) 引数：xlim, ylim(軸の最小値と最大値) -----
197 #             xaxp, yaxp(目盛線の位置)
198
199 ## (a) 軸の範囲(最小値,最大値)の設定,目盛線は自動
200 plot(
201   x1, y1,
202   xlim = c(0, 16),     # x軸 c(最小値、最大値)
203   ylim = c(0, 12))     # y軸 c(最小値、最大値)
204
205 ## (b) 軸の範囲と目盛線の設定
206 plot(
207   x1, y1,
208   xlim = c(0, 16),
209   xaxp = c(0, 16, 2), # c(最小値,最大値,区切り数)
210   ylim = c(0, 12),
211   yaxp = c(0, 12, 4)) # c(最小値,最大値,区切り数)
```

（my_base_graphics2.R：196-211）

xlim = c(0, 12)

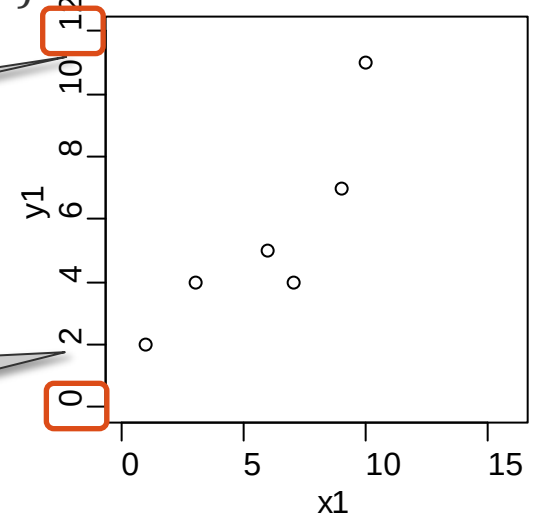
目盛線の位置は
自動設定

xlim と xaxp を併用
0, 16 を一致させる

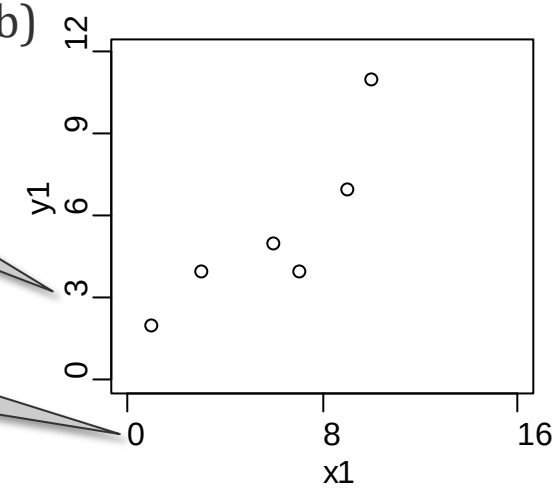
目盛線の位置を指定
区切り数：4

目盛線の位置を指定
区切り数：2

(a)



(b)



関数 plot() の使い方：引数

●関数 plot() の引数：(16) xlab, ylab, main, sub, ann（軸）

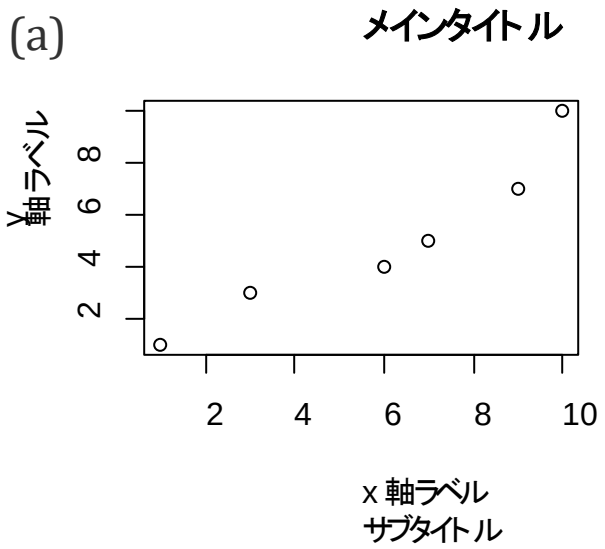
(a) xlab, ylab：x 軸ラベル、y 軸ラベルのテキストを渡す
　　main, sub：メインタイトル、サブタイトルのテキストを渡す

(b) ann：FALSE で xlab, ylab, main, sub の非表示（TRUE が規定値）

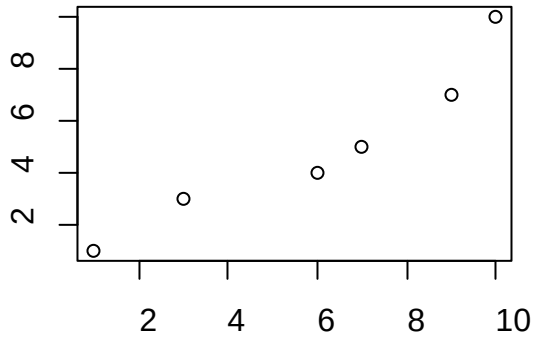
```
214 # (16) 引数：xlab, ylab, main, sub, ann----------
215
216 ## (a) xlab, ylab, main, sub の表示
217 plot(x1, y1,              # 散布図
218      xlab = "x軸ラベル",
219      ylab = "y軸ラベル",
220      main = "メインタイトル",
221      sub = "サブタイトル")
222
223 ## (b) annにより非表示、代わりに低水準関数で表示
224 plot(x1, y1,
225      ann = FALSE)      # main, xlab, ylab を非表示
226 title(
227   main = "カスタムタイトル",
228   xlab = "カスタムx軸ラベル",
229   ylab = "カスタムy軸ラベル")
```

（my_base_graphics2.R：214-229）

(a)



(b)



関数 plot() の使い方：引数

- 関数 plot() の引数：(17) log（軸）

  対数目盛の軸に変換

  plot(x, y, log = "x")　　：x 軸を 常用対数スケールに変換

  plot(x, y, log = "y")　　：y 軸を 常用対数スケールに変換

  plot(x, y, log = "xy")　　：x, y の両軸を 常用対数スケールに変換

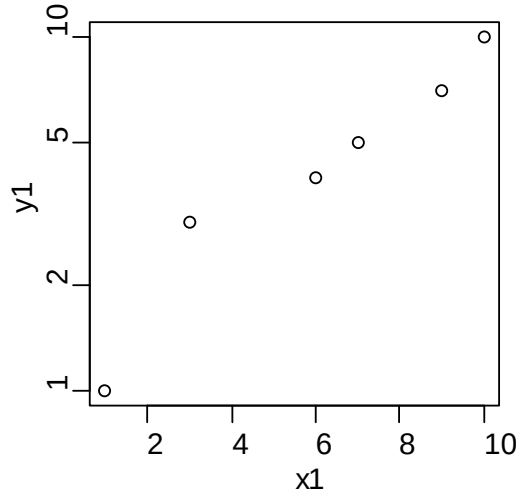
  自然対数スケールには対応しない

  表示のみ、データが対数変換されているのではない

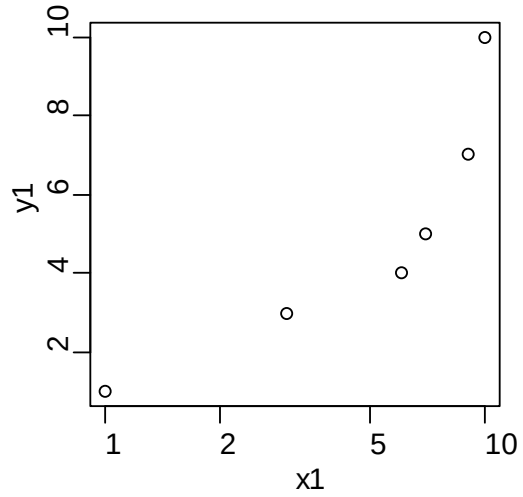
```
232 # (17) 引数：log(常用対数の軸、表示上) ----------
233
234 plot(x1, y1, log = "y")   # x軸を常用対数スケール
235 plot(x1, y1, log = "x")   # y軸を常用対数スケール
236 plot(x1, y1, log = "xy")  # 両軸を常用対数スケール
```

（my_base_graphics2.R：232-236）

log = "x"



log = "y"



関数 plot() の使い方：引数

●関数 plot() の引数：(18) axes, xaxt, yaxt, frame.plot（軸）

デフォルトの軸の表示を非表示に設定

(a) axes = FALSE：軸とプロット領域の枠線を非表示

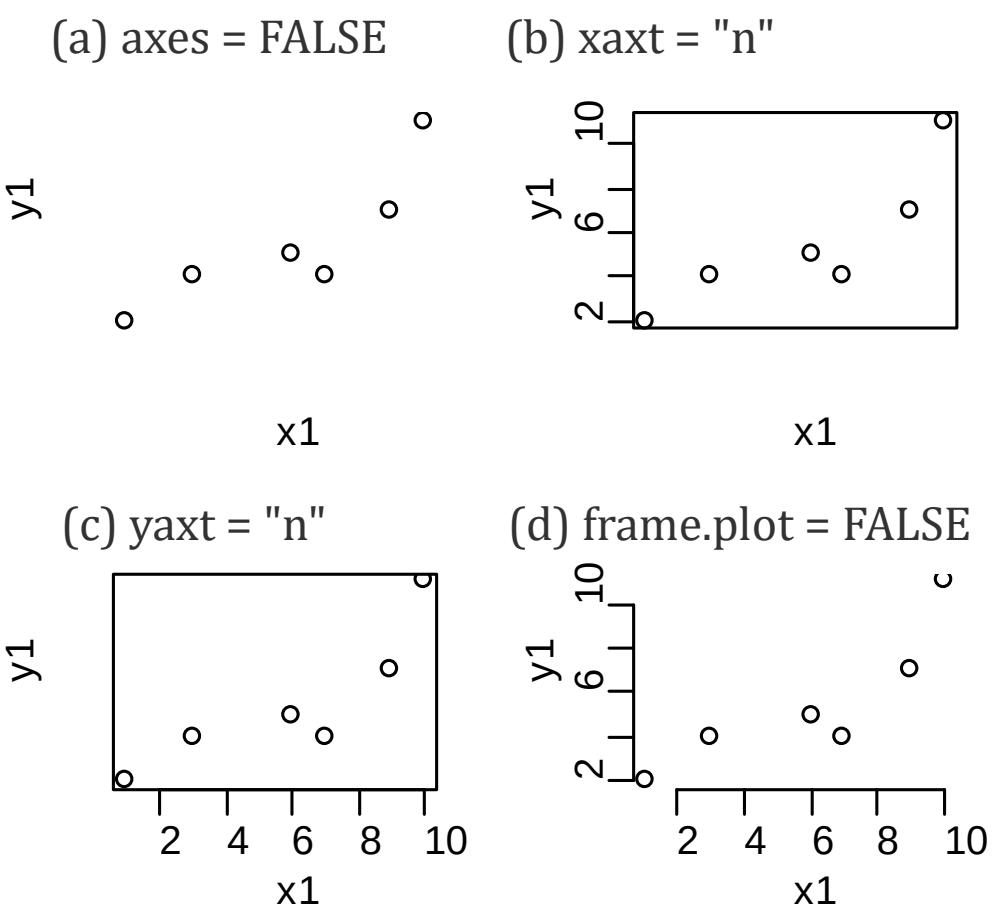
(b) xaxt = "n"：x 軸 の目盛線、目盛ラベルを非表示

(c) yaxt = "n"：y 軸の目盛線、目盛ラベルを非表示

(d) frame.plot = FALSE：プロット領域の枠線を非表示

```
239 # (18) 引数：axes, xaxt, yaxt, frame.plot -------
240
241 par(mfrow = c(2, 2))        # 作図画面を４分割
242
243 plot(x1, y1, axes = FALSE) # (a) 両軸と枠を非表示
244 plot(x1, y1, xaxt = "n")    # (b) x軸を非表示
245 plot(x1, y1, yaxt = "n")    # (c) y軸を非表示
246 plot(x1, y1,
247      frame.plot = FALSE)    # (d) 枠を非表示
248
249 par(mfrow = c(1, 1))        # 作図画面を復元
```

（my_base_graphics2.R：239-249）

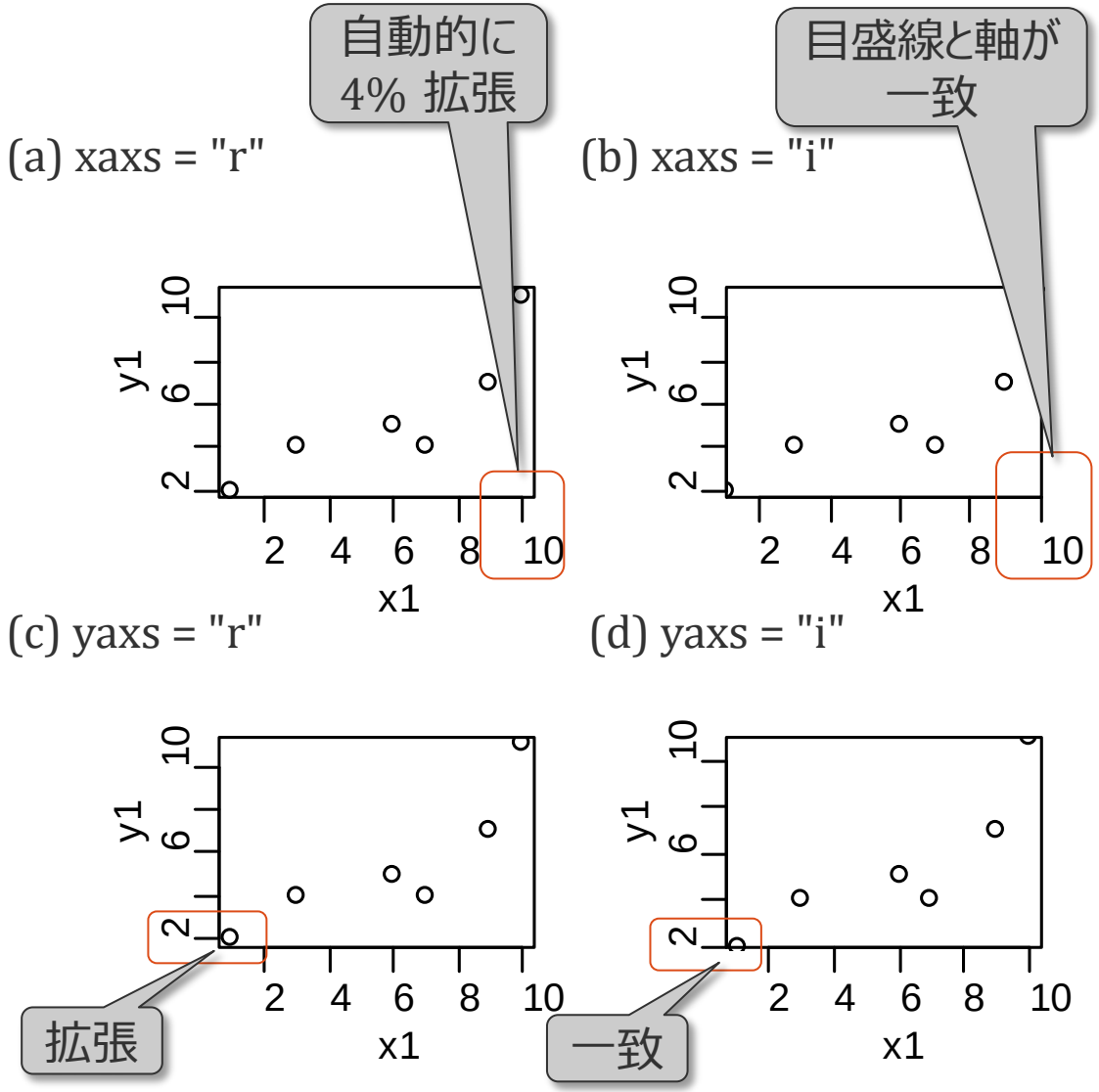


関数 plot() の使い方：引数

- 関数 plot() の引数：(19) xaxs, yaxs（軸）
  目盛線の位置と軸線の位置の関係を制御
  (a) xaxs = "r"：x 軸の範囲を 4% 自動拡張（規定値）
  (b) xaxs = "i"：x 軸の目盛線と軸が一致
  (c) yaxs = "r"：y 軸の範囲を 4% 自動拡張（既定値）
  (d) yaxs = "i"：y 軸の目盛線と軸が一致

```
252 # (19) 引数：xaxt, yaxt ------------------------
253
254 par(mfrow = c(2, 2))        # 作図画面を４分割
255
256 plot(x1, y1, xaxs = "r")    # (a) 自動(規定値)
257 plot(x1, y1, xaxs = "i")    # (b) 軸の拡張なし
258 plot(x1, y1, yaxs = "r")    # (c) 自動(規定値)
259 plot(x1, y1, yaxs = "i")    # (d) 軸の拡張なし
260
261 par(mfrow = c(1, 1))        # 作図画面を復元
```

（my_base_graphics2.R：252-261）



関数 plot() の使い方：引数

aspect ratio

●関数 plot() の引数：(20) asp（軸）

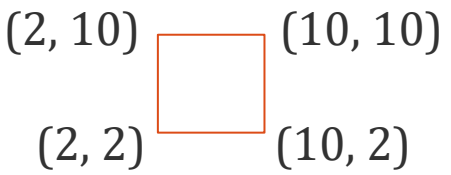
軸のアスペクト比（縦/横の比）を制御

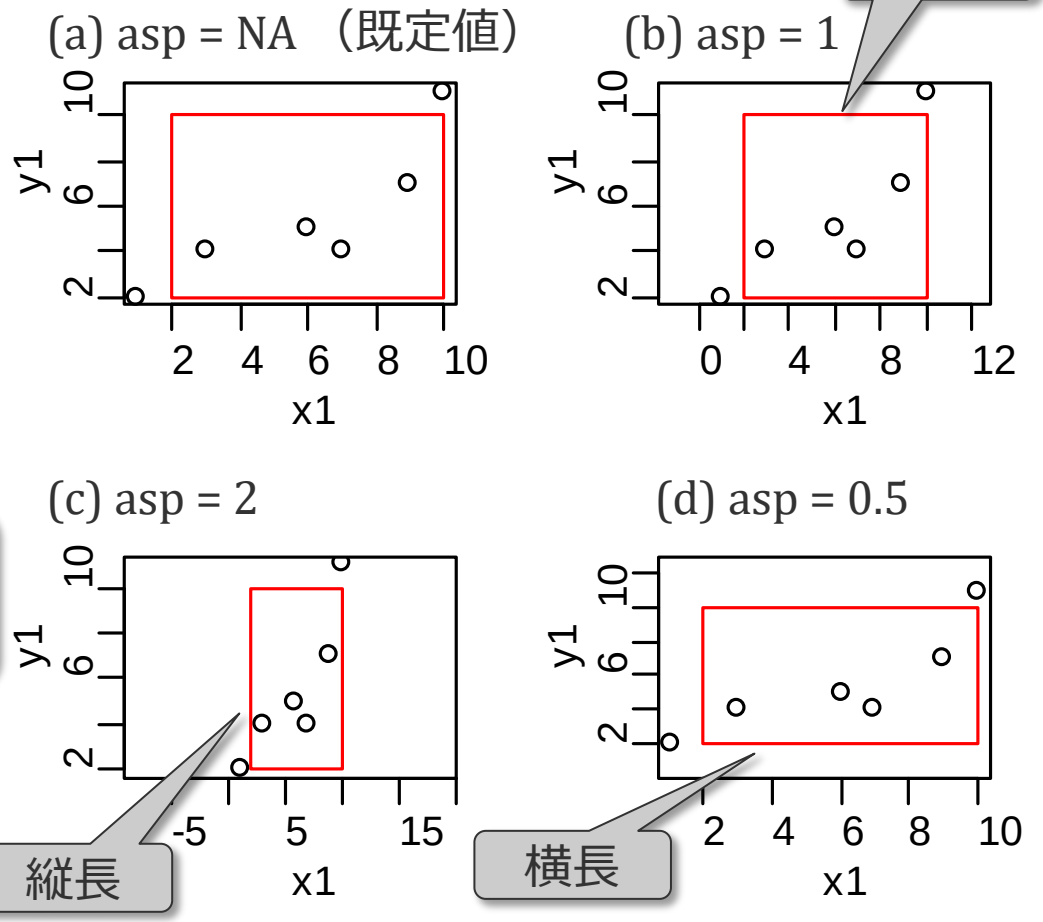
```
264 # (20) 引数：asp(アスペクト比) -----------------
265
266 par(mfrow = c(2, 2))    # 作図画面を４分割
267
268 x_coords <- c(2, 2, 10, 10, 2)
269 y_coords <- c(2, 10, 10, 2, 2)
270
271 plot(x1, y1, asp = NA)  # (a) 自動（規定値）
272 lines(x_coords, y_coords, col = "red")
273
274 plot(x1, y1, asp = 1)   # (b) x軸1単位＝y軸1単位
275 lines(x_coords, y_coords, col = "red")
276
277 plot(x1, y1, asp = 2)   # (c) y軸/x軸=2
278 lines(x_coords, y_coords, col = "red")
279
280 plot(x1, y1,asp = 0.5)  # (d) y軸/x軸=0.5
281 lines(x_coords, y_coords, col = "red")
282
283 par(mfrow = c(1, 1))    # 作図画面を復元
```

既定値
自動調整

両軸の１単位
が同じ長さ
地図などに適

（my_base_graphics2.R：264-283）





関数 plot() の使い方：引数

シンボル、マーク、マーカー

●関数 plot() の引数：(21) グラフィックスパラメータ（記号）

(a) 引数 pch：記号の番号（Point CHaracter）、記号 26 種類＋ASCII文字（番号 26～31 は未使用）
引数 lwd：記号の枠線の太さ（規定値 1）、引数 cex：記号のサイズ（規定値 1）

pch = 0～14：白抜き記号（中身は透明、背景を隠さない）、引数 col で枠線の色指定

pch = 15～20：塗りつぶし記号、引数 col で記号全体の色指定

pch = 21～25：塗りつぶし記号、引数 col で枠線の色指定（既定値は黒）
引数 bg で中身(背景色) の色指定（既定値は白）

pch = 32～127：ASCII 文字を表示（pch = "A"、"a"、"#"、"." なども可）

丸印の区別

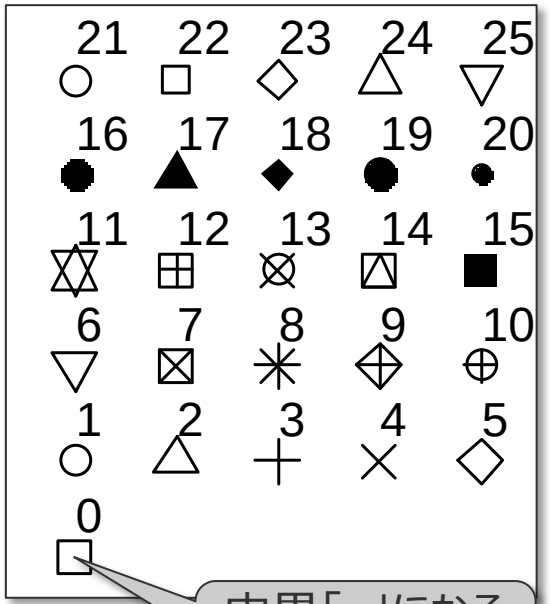
pch = 1 ：中抜き記号、標準的に使用、規定値（記号の後ろが見える）

pch = 16：塗りつぶし記号、標準的に使用、可視性が高い（枠線なし）

pch = 19：塗りつぶし記号、やや大きめ、強調用（枠線あり、lwd の影響）

pch = 20：塗りつぶし記号、やや小さめ、多数のデータの時に便利

pch = 21：塗りつぶし記号、枠線と中身を別々に色指定



pch = 46

中黒「・」になる場合がある

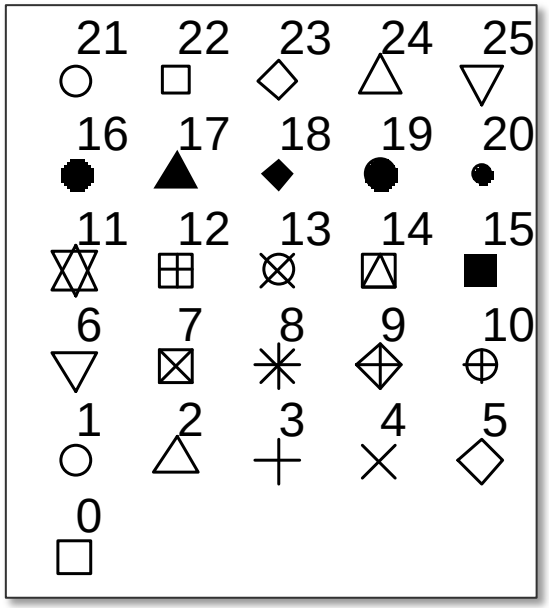
関数 plot() の使い方：引数

●関数 plot() の引数：(21) グラフィックスパラメータ（記号）

(a) 引数 pch：記号の番号（Point CHaracter）、記号 26 種類＋ASCII文字（番号 26～31 は未使用）
引数 lwd：記号の枠線の太さ（規定値 1）、引数 cex：記号のサイズ（規定値 1）

記号の選択

とりあえずグラフを書きたい（グループを色で区別）	pch = 1（既定値、指定不要）
白黒で 3 グループを区別（色覚特性に配慮）	pch = 1 (○), pch = 2 (△), pch = 0 (□)
発表用にグラフを綺麗に見せたい（プロジェクター投影で見やすい）	pch = 16（●）, 17（▲）, 15（■）
枠線と中身の色にこだわりたい	pch = 21（〇）, 24（□）, 22（□）



各パッケージ（graphics、lattice、ggplot2）で基本的に共通
パッケージごとに若干の違いあり

（my_base_graphics2.R：314-335）

関数 plot() の使い方：引数

●関数 plot() の引数：(21) グラフィックスパラメータ（色）

(b) 引数 col、bg：プロットの色、ぬりつぶしの色

引数 bg, fg：背景色、前景色

引数 col.axis：軸の色

引数 col.lab：ラベルの色

引数 col.main, col.sub：タイトルの色

指定方法

数値（標準関数でのみ有効）

col = 1

色の名前を指定

col = "red"、colors()

RGB 指定

col = rgb(1, 0, 0)

col = "#FF0000"

1: "black" （黒）
2: "red" （赤）
3: "green3" （緑）
4: "blue" （青）
5: "cyan" （シアン）
6: "magenta" （マゼンタ）
7: "yellow" （黄）
8: "gray" （グレー）

色の名前

```
315  points(1, 1, pch = 21, bg = "black")
316  points(2, 1, pch = 21, bg = "white")
317  points(3, 1, pch = 21, bg = "red")
318  points(4, 1, pch = 21, bg = "green")
319  points(1, 2, pch = 21, bg = "blue")
320  points(2, 2, pch = 21, bg = "yellow")
321  points(3, 2, pch = 21, bg = "cyan")
322  points(4, 2, pch = 21, bg = "magenta")
323  points(1, 3, pch = 21, bg = "gray")
324  points(2, 3, pch = 21, bg = "lightgray")
325  points(3, 3, pch = 21, bg = "darkgray")
326  points(4, 3, pch = 21, bg = "orange")
327  points(1, 4, pch = 21, bg = "pink")
328  points(2, 4, pch = 21, bg = "purple")
329  points(3, 4, pch = 21, bg = "brown")
330  points(4, 4, pch = 21, bg = "beige")
331  points(1, 5, pch = 21, bg = "darkred")
332  points(2, 5, pch = 21, bg = "darkgreen")
333  points(3, 5, pch = 21, bg = "lightblue")
334  points(4, 5, pch = 21, bg = "lightgreen")
```

（my_base_graphics2.R：314-335、一部省略）

関数 plot() の使い方：引数

●関数 plot() の引数：(21) グラフィックスパラメータ（線）

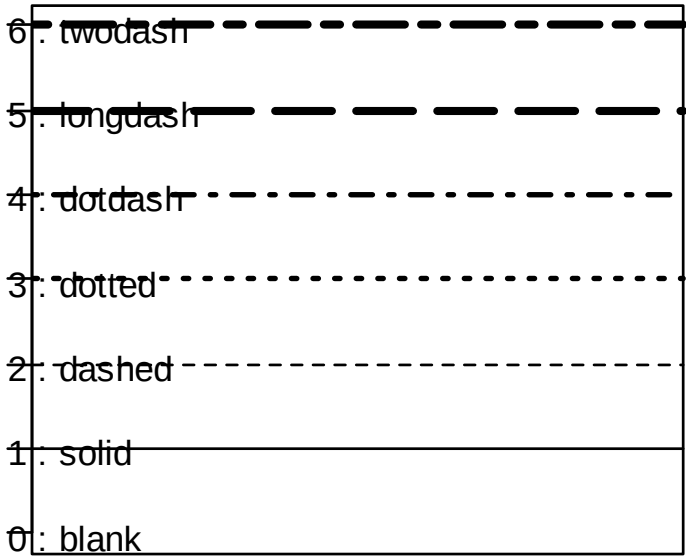
(c) 引数 lty、lwd ：線の種類と太さ

lty（Line Type）：線種、番号または線種の名前

lwd（Line Width）：線の太さ、規定値の 1

```
339  ## (c) 引数：lty(線種)、lwd(線の太さ)
340  plot(x1, y1,
341       xlab = "線の太さ(1.0,1.5,2.0,2.5,3.0,3.5)",
342       ylab = "", type = "n", yaxt = "n", xaxt = "n",
343       xlim = c(0, 6), ylim = c(0, 6))
344
345  abline(h = 1, lwd = 1.0, lty = "solid")
346  abline(h = 2, lwd = 1.5, lty = "dashed")
347  abline(h = 3, lwd = 2.0, lty = "dotted")
348  abline(h = 4, lwd = 2.5, lty = "dotdash")
349  abline(h = 5, lwd = 3.0, lty = "longdash")
350  abline(h = 6, lwd = 3.5, lty = "twodash")
351
352  axis(side = 2,              # 左(y軸)を指定
353       at = 0:6,              # x座標の位置
354       labels = c(
355         "0 : blank","1 : solid","2 : dashed",
356         "3 : dotted", "4 : dotdash",
357         "5 : longdash", "6 : twodash"),
358       las = 2, cex.axis = 0.8)
```

（my_base_graphics2.R：339-358）



線の太さ（1.0,1.5,2.0,

関数 plot() の使い方：引数

●関数 plot() の引数：(21) グラフィックスパラメータ（フォント）

(d) 引数 font：フォントの種類、番号 1～5

1: 普通のフォント（"plain"、標準のローマンフォント）

2: 太字（bold）

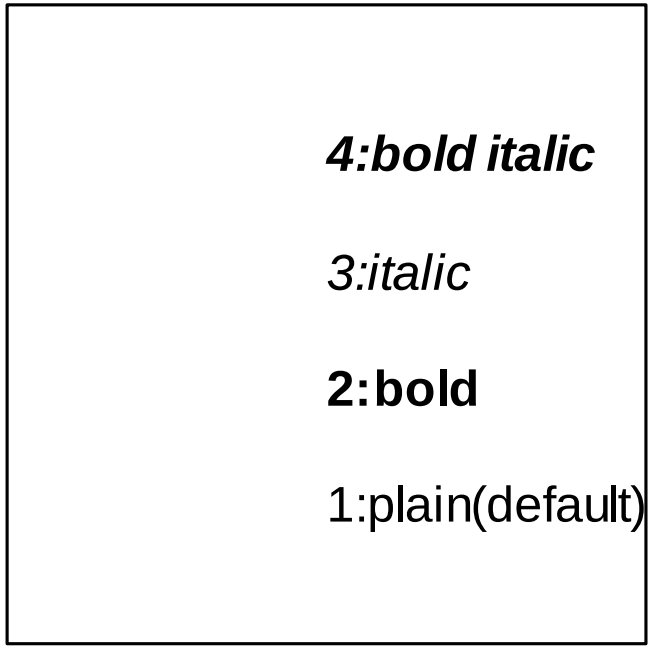
3: イタリック（斜体）

4: 太字かつイタリック

5: Adobe Symbol フォント（特殊記号や数学記号）

```
360  ## (d) 引数：font(テキストのフォント)
361  plot(x1, y1,
362       xlab = "フォントの種類と番号",
363       ylab = "",
364       xlim = c(0, 6), ylim = c(0, 5),
365       type = "n", yaxt = "n", xaxt = "n")
366
367  text(3, 1, label = "1:plain(default)", font = 1)
368  text(3, 2, label = "2:bold", font = 2)
369  text(3, 3, label = "3:italic", font = 3)
370  text(3, 4, label = "4:bold italic", font = 4)
```

（my_base_graphics2.R：360-370）



関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数

高水準と低水準の関数を組合せてグラフを作成

高水準グラフィックス関数
　グラフの骨組みを描画
　（図の全体構造を作成）

低水準グラフィックス関数
　高水準関数で作成したグラフに
　装飾やデータを追加

ジェネリック関数
　plot()
　points()、lines()、text()

高水準グラフィックス関数

主な関数名	パッケージ名	グラフ名（主な用途）
plot()	base, graphics stats	散布図、線グラフ、棒グラフ 箱ひげ図、関数のプロット
pairs()	graphics	散布図行列
boxplot()	graphics	箱ひげ図
stripchart()	graphics	１次元散布図
hist()	graphics	ヒストグラム
barplot()	graphics	棒グラフ
dotchart()	graphics	Cleveland ドットチャート
mosaicplot()	graphics	モザイク図
fourfoldplot()	stats	4分割プロット
qqplot()	stats	２標本 Q-Q プロット
qqnorm()	stats	１標本 Q-Q プロット
contour()	graphics	等高線プロット
persp()	graphics	３次元透視図
image()	graphics	ヒートマップ
curve()	graphics	関数のプロット

低水準グラフィックス関数

主な関数	追加する要素
points( )	点
lines( )	線
abline( )	直線
segments( )	線分
arrows( )	矢印
rect( )	矩形
polygon( )	多角形
text( )	文字列
mtext( )	余白の文字列
legend( )	凡例
title( )	タイトル、軸ラベル
rug( )	軸上のラグ
axis( )	軸
box( )	箱
grid( )	グリッド線

関数 plot() の使い方：低水準関数との組合せ

- 関数 plot() と低水準関数：(22) points(), matpoints()
  - (a) points()： 1 点の追加
  - (b) points()：複数点の追加、引数 type によるタイプの選択

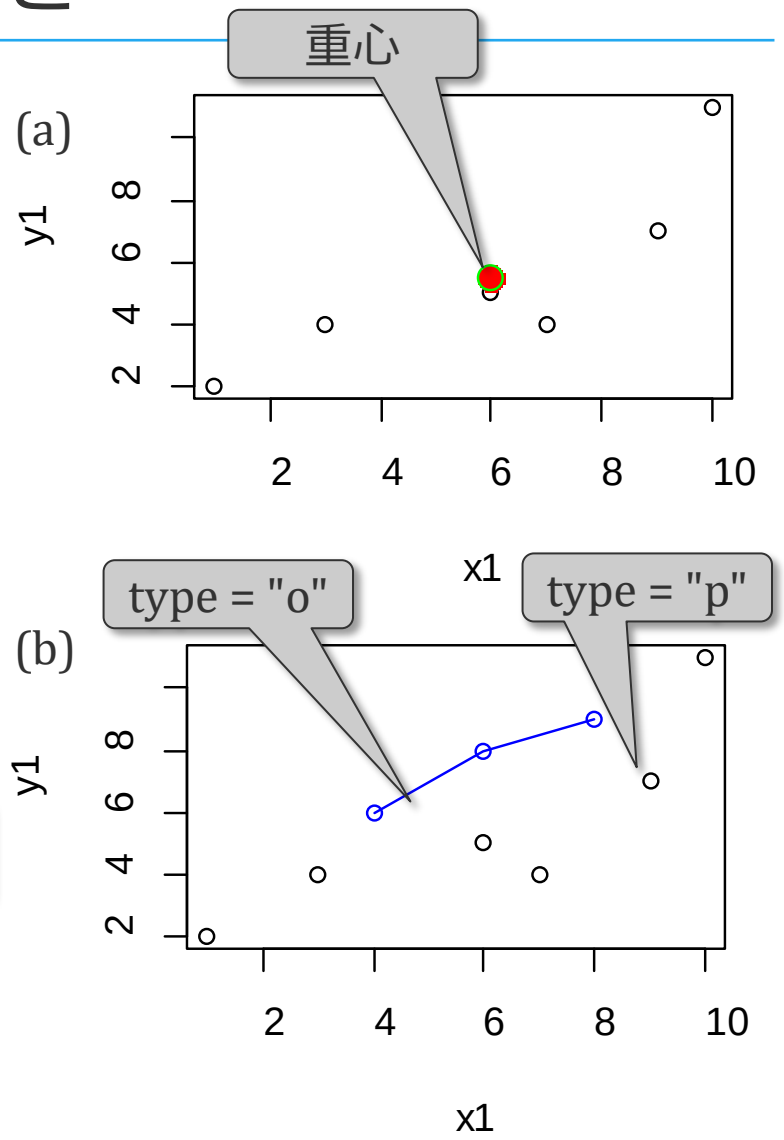
```
373 # (22) 低水準関数：points, matpoints(点) --------
374
375 ## (a) 1 点の追加
376 plot(x1, y1)                # 散布図
377 points(x = mean(x1),       # 重心の計算
378        y = mean(y1),       # 重心の計算
379        pch = 21,           # 記号の形状
380        bg = "red",         # 記号の塗りつぶし
381        col = "green",      # 記号の枠の色
382        cex = 1.5,          # 記号サイズ
383        lwd = 1.2)          # 記号の枠の太さ
384
385 ## (b)  複数点の追加(線の追加)
386 plot(x1, y1)                # 散布図
387 points(x = c(4, 6, 8),     # x 座標
388        y = c(6, 8, 9),     # y 座標
389        type = "o",         # 点の上に線
390        col = "blue")       # 点と線の色
```

（my_base_graphics2.R：373-390）

（plot() と同様）

関数 lines() と同じ機能

plot() と同様 type を指定可



関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(22) points(), matpoints()（点、線）

(c) matpoints()：複数系列の点（またはそれを結んだ線）を追加、行列の列をグループとして扱う

points() の複数回呼び出しと同等の機能、系列ごとに色と形状の指定が可能

```
392  ## (c) 複数系列の点の追加
393  xx  <- c(1, 2, 4, 5, 7)
394  yy1 <- c(2, 3, 4, 6, 8)
395  yy2 <- c(1, 2, 3, 5, 7)
396
397  ### (c-1) points() による点の追加
398  plot(NA, xlim = c(0, 7), ylim = c(1, 8),
399       xlab = "x", ylab = "y")
400  points(xx, yy1, col = "blue", pch = 1)
401  points(xx, yy2, col = "red", pch = 2)
402
403  ### (c-2) matpoints() による点の追加
404  yy <- cbind(yy1, yy2)
405  plot(NA, xlim = c(0, 7), ylim = c(1, 8),
406       xlab = "x", ylab = "y")
407  matpoints(xx, yy,
408            pch = 1:2, col = c("blue", "red"))
```

要素数は 5
対応のある
データ

枠のみのグラフ
type= "n" と同じ

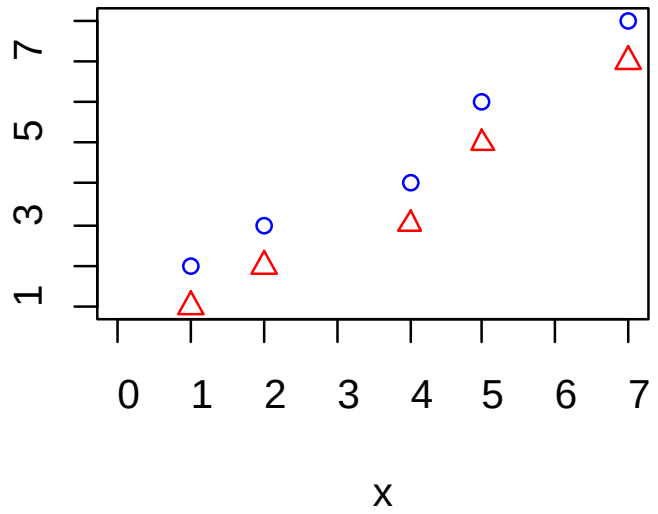
points() を
2 回呼び出し

5行×2列の
行列に変換

matpoints() を
1 回呼び出し

（my_base_graphics2.R：392-408）

points()、matpoints() は
ジェネリック関数
ベクトル、データフレーム、
マトリックスを渡すことも可



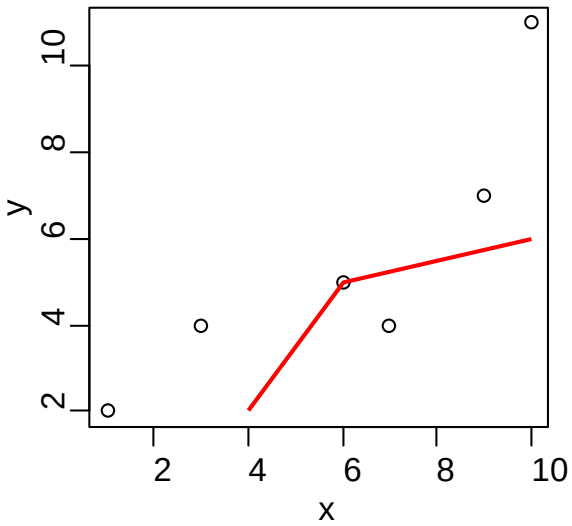
関数 plot() の使い方：低水準関数との組合せ

- 関数 plot() と低水準関数：(23) lines()、matlines()（直線）
  - (a) lines()　　：既存のグラフに、折れ線を 1 本追加、同じ長さのベクトルを x,y に渡す
  - (b) matlines()　：複数列のデータを一度にプロット、行列を利用、列ごとに別グループとして扱う
    lines() の複数回呼び出しと同等の機能、系列ごとに色と形状を指定可

```
411 # (23) 低水準関数：lines, matlines (直線)--------
412
413 ## (a) 折れ線の追加
414 x2 <- c(4, 6, 10)              # 追加するx座標
415 y2 <- c(2, 5,  6)              # 追加するy座標
416
417 plot(x1, y1,
418      xlab = "x", ylab= "y") # 散布図
419 lines(x2, y2,                  # 座標を線で連結
420       lty = "solid",          # 線種
421       lwd = 2,                # 線の太さ
422       col = "red",            # 線の色
423       lend = "butt",          # 線の端点の形状
424       ljoin = "bevel")        # 線の結合部の形状
```

3 点の座標データ

（my_base_graphics2.R：411-424）



関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(23) lines()、matlines()（直線）

(a) lines()　　：既存のグラフに、折れ線を 1 本追加、同じ長さのベクトルを x,y に渡す

(b) matlines()　：複数列のデータを一度にプロット、行列を利用、列ごとに別グループとして扱う

lines() の複数回呼び出しと同等の機能、系列ごとに色と形状を指定可

```
426  ## (b) 複数系列の折れ線の追加
427  xx1 <- c(1, 2, 3, 5, 6)      # 系列1 のx座標
428  yy1 <- c(3, 5, 6, 8, 7)      # 系列1 のy座標
429  xx2 <- c(0, 2, 3, 4, 6)      # 系列2 のx座標
430  yy2 <- c(1, 2, 4, 3, 5)      # 系列2 のy座標
431
432  ### (b-1) lines() による折れ線の追加
433  plot(x = NA, xlim = c(0,6), ylim= c(0, 10))
434  lines(xx1, yy1, col = "blue", lty = 1)
435  lines(xx2, yy2, col = "red", lty = 2)
436
437  ### (b-2) matlines() による折れ線の追加
438  xx <- cbind(xx1, xx2)
439  yy <- cbind(yy1, yy2)
440  plot(x = NA, xlim = c(0,6), ylim= c(0, 10))
441  matlines(xx, yy,
442           lty = 1:2, col = c("blue", "red"))
```

（my_base_graphics2.R：426-442）

要素数は全て 5 系統内で対応のあるデータ

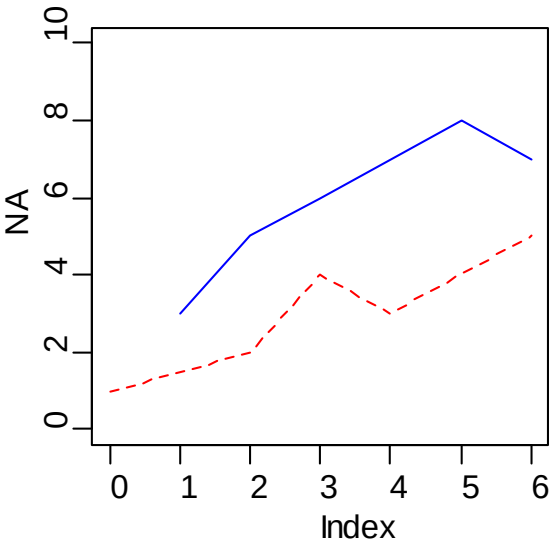
枠のみのグラフ type= "n" と同じ

lines() を 2 回呼び出し

行列に変換

matlines() を 1 回呼び出し

lines()、matlines() は plot() と同様にジェネリック関数



関数 plot() の使い方：低水準関数との組合せ

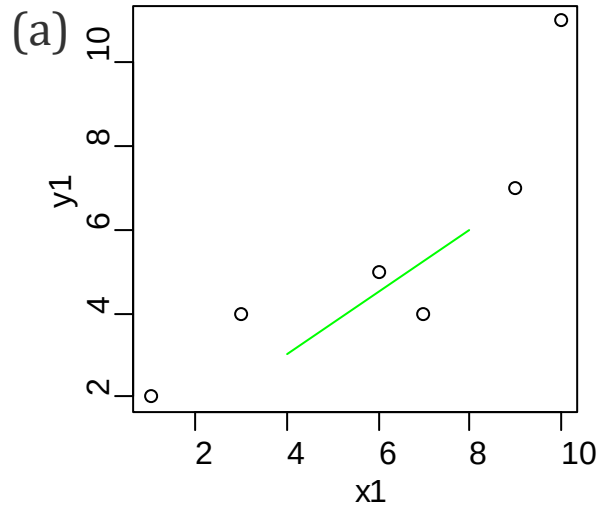
- 関数 plot() と低水準関数：(24) segments()（線分）
  - (a) 1 本の線分を追加　引数：線分の始点 (x0, y0)、終点 (x1. y1)
  - (b) 複数の線分をまとめて追加　引数：始点と終点の数値ベクトル

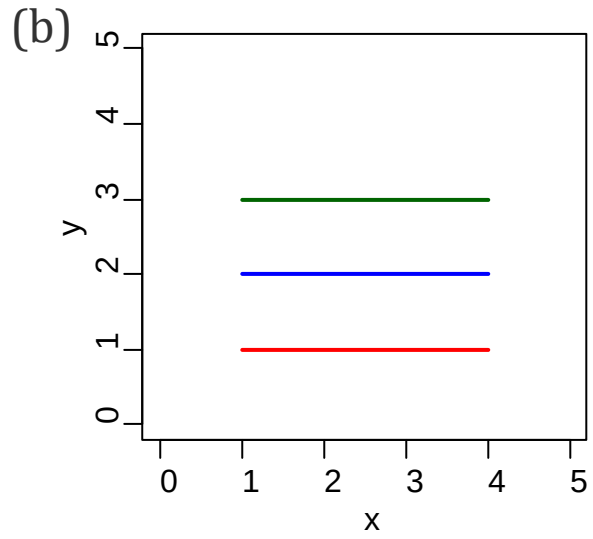
```
445 # (24) 低水準関数 segments(線分) ---------------
446
447 ## (a) 1 本の線分を表示
448 plot(x1, y1)                # 散布図
449 segments(x0 = 4, y0 = 3, # 線分の始点の座標
450          x1 = 8, y1 = 6, # 線分の終点の座標
451          col = "green")
452
453 ## (b) 複数の線分を同時に表示
454 xx0 <- c(1, 1, 1); yy0 <- c(1, 2, 3)
455 xx1 <- c(4, 4, 4); yy1 <- c(1, 2, 3)
456
457 plot(NA, xlab = "x", ylab = "y",
458      xlim = c(0, 5), ylim = c(0, 5))
459 segments(xx0, yy0, xx1, yy1,
460          col = c("red", "blue", "darkgreen"),
461          lwd = 2)
```

引数 x0, y0, x1, y1
線分の始点(x0, y0)と
終点(x1. y1)

線分の始点(x0, y0)と
終点(x1. y1) のベクトル

（my_base_graphics2.R：445-461）





関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(24) segments()（線分）

(c)エラーバーの表示

事前に標準偏差（SD）、標準誤差（SE）などを計算してブジェクトに付値

プロットした記号（シンボル）の上下に線分を描画

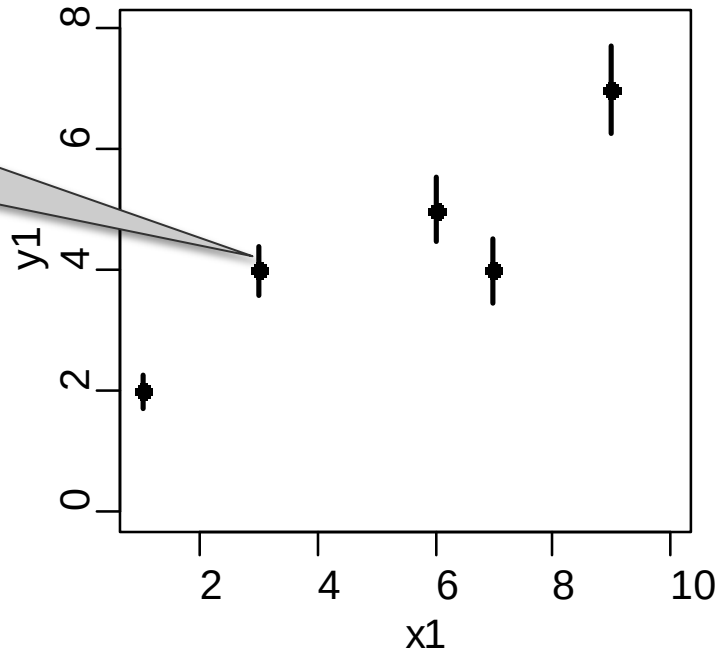
この他、目盛線の追加など適用場面は多い

```
463  ## (c) エラーバーを表示
464  ### 事前に標準偏差、標準誤差などを計算、err に付値
465  err <- c(0.28, 0.42, 0.53, 0.54, 0.71, 0.89)
466
467  ### エラーバー付の散布図
468  plot(x1, y1, pch = 16, ylim = c(0, 8))
469  segments(x1, (y1 + err), x1, (y1 - err), lwd = 2)
```

（my_base_graphics2.R：463-469）

標準偏差 SD
標準誤差 SE
事前に用意

標準偏差 SD
標準誤差 SE
を線分で表示



関数 plot() の使い方：低水準関数との組合せ

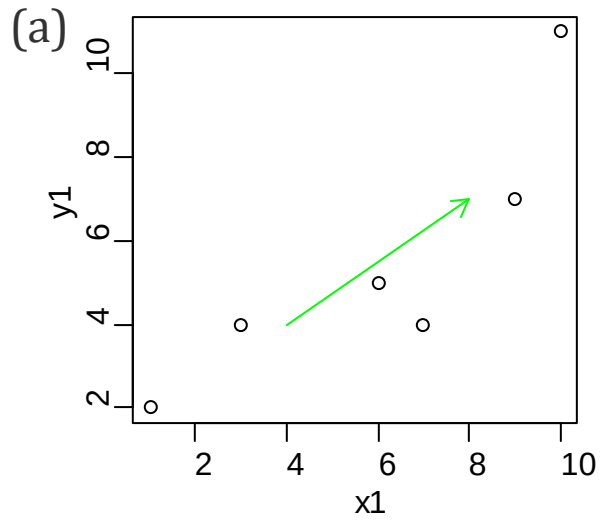
- 関数 plot() と低水準関数：(25) arrows()（矢印）
  - (a) arrows()：始点と終点をつなぐ線分を描き、先端に矢じりを付加
  - (b) エラーバーの表示　（segments() 参照）

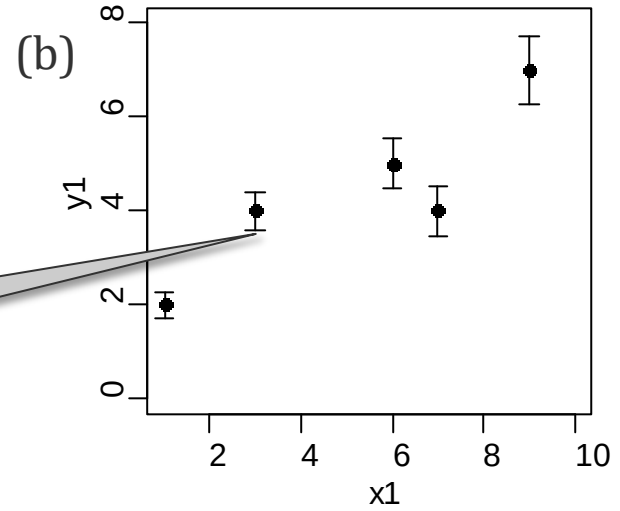
```
472 # (25) 低水準関数：arrows(矢印) ----------------
473
474 ## (a) 矢印を追加
475 plot(x1, y1)
476 arrows(x0 = 4, y0 = 4, # 線分の始点の座標
477        x1 = 8, y1 = 7, # 線分の終点の座標
478        length = 0.1,   # 矢印の先端の長さ
479        code = 2,       # 1,2:片矢印,3:両矢印
480        col = "green")
481
482 ## (b) エラーバーを表示
483 err <- c(0.28, 0.42, 0.53, 0.54, 0.71, 0.89)
484
485 plot(x1, y1, pch = 16, ylim = c(0, 8))
486 arrows(x1, (y1 - err), x1, (y1 + err),
487        angle = 90,     # 軸と先端の角度
488        length = 0.05,  # 矢印の先端の長さ
489        code = 3)       # 両矢印
```

（my_base_graphics2.R：472-489）

引数 code
1 ：(x0,y0) が矢じり
2 ：(x1,y1) が矢じり
3 ：両矢印

引数 angle = 90 で
「矢じり」が T 字形





関数 plot() の使い方：低水準関数との組合せ

●低水準関数：(26) abline()（直線）

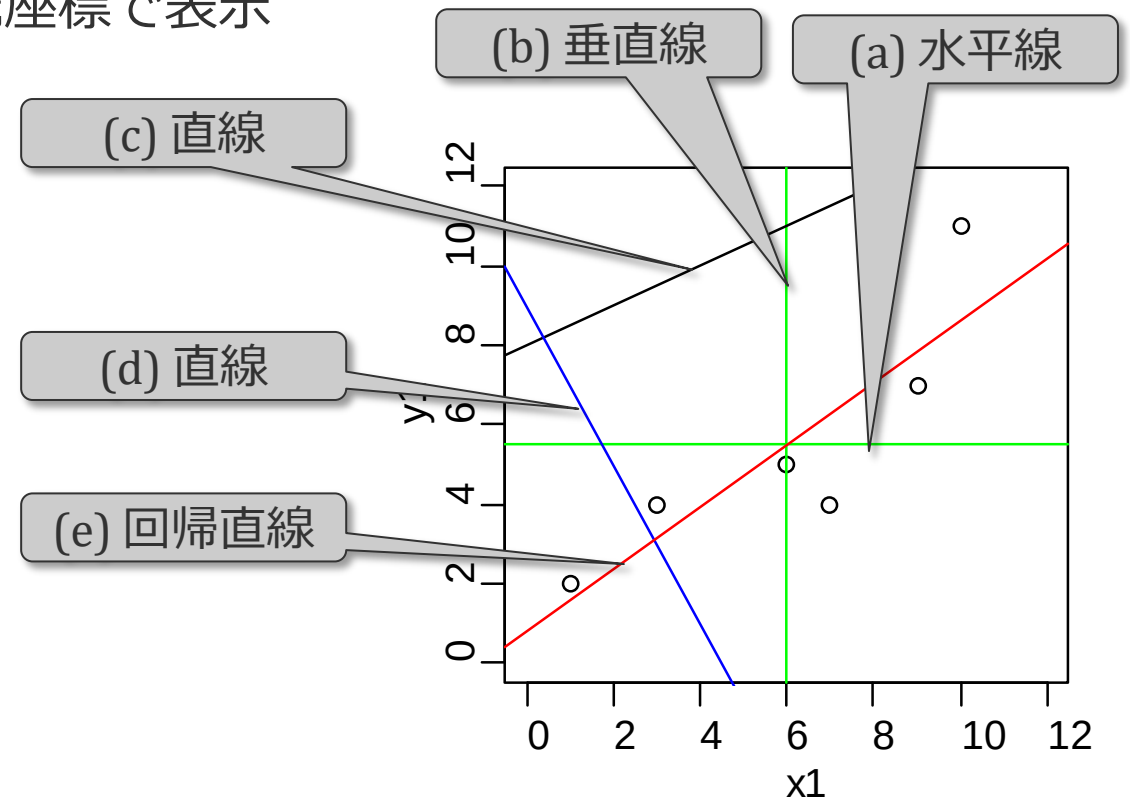
引数 h：水平線の y 値、v：垂直線の x 値、a,b： 1 次式の切片と傾き

coef： 1 次式の切片と傾きを表す長さ 2 のベクトル、reg：回帰オブジェクト、

untf：TRUE の場合に対数変換の軸において元座標で表示

```
492 # (26) 低水準関数：abline(直線) ----------------
493
494 plot(x1, y1, xlim = c(0,12), ylim = c(0, 12))
495
496 abline(h = mean(y1),      # (a) 水平線を追加
497        v = mean(x1),      # (b) 垂直線を追加
498        col = "green")
499
500 abline(a = 8, b = 0.5,    # (c) 直線を追加
501        col = "black")     #       y = a + bx
502
503 abline(coef = c(9, -2),   # (d) 直線を追加
504        col = "blue")      #       y = 9 - 2x
505
506 lm_out <- lm(y1 ~ x1)
507 abline(reg = lm_out,      # (e) 回帰直線を追加
508        col = "red")
```

（my_base_graphics2.R：492-508）



関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と高水準関数：(27) curve()（曲線、curve は高水準関数）

曲線を追加する低水準関数はないので、曲線を描く高水準関数 curve() を使用

（curve() は細かに連続させた x に対して y を計算して lines() を呼び出している）

curve()：数式に基づいた関数をプロット

引数 expr：x の関数名、数式、ユーザー定義関数を渡す　例 x^2 、sin(x)、"ifelse(x > 0, x, NA)"

add：TRUE を渡して既存のグラフに上書きする

n：描画する曲線を構成する点の数を指定（既定値 101）

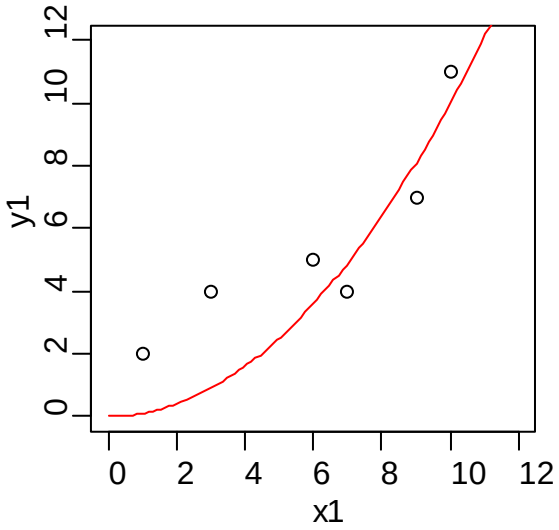
from, to：グラフを描画する x の下限と上限（xlim よりも優先的）

```
511 # (27) 高水準関数 curve(曲線) ------------------
512
513 plot(x1, y1,
514      xlim = c(0,12), ylim = c(0, 12))
515 curve(expr = 0.1 * x^2,   # 関数の数式
516       n = 101,             # xの範囲内での点数
517       col = "red",         #       (↑既定値101)
518       add = TRUE)          # 高水準関数で上書き
```

plot() の軸に従う

上書を許可

（my_base_graphics2.R：511-518）



関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(28) polygon()（多角形）

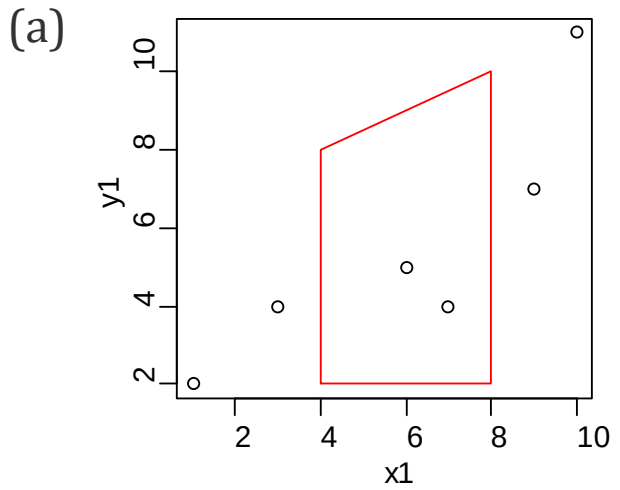
polygon()：任意の多角形（ポリゴン）をグラフ上に描く

引数 x, y：ポリゴンの頂点の x 座標と y 座標を数値ベクトルで渡す

座標が順序通りに結ばれ、最後の点と最初の点が結ばれる

col：ポリゴン内部を塗りつぶす色、NULLで無色（デフォルト）

border：ポリゴンの枠の色、通常は黒、NA で枠線なし

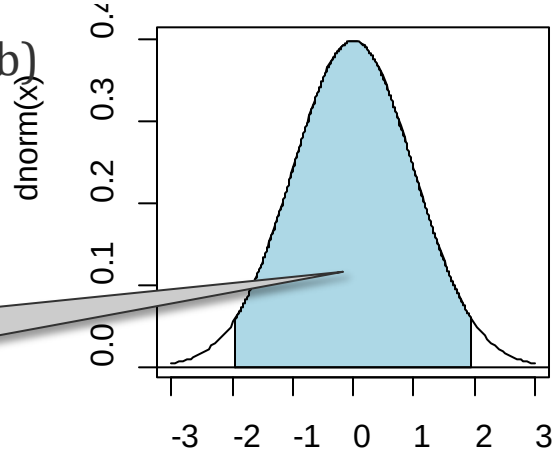


多角形の頂点を細かく設定して、曲線の一部の領域を表示

関数分布で説明

```
521 # (28) 低水準関数：polygon(多角形) --------------
522
523 plot(x1, y1)                    # 散布図
524 polygon(x = c(4, 4, 8, 8),     # x座標
525         y = c(2, 8, 8, 2),     # y座標
526         border = "red",
527         lty = "solid")
```

（my_base_graphics2.R：521-527）



polygon() で描画
関数分布で説明

関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(29) box()（領域の枠線）

プロット領域、作図領域、デバイス領域の枠線（［Plots］タブ）

```
530 # (29) 低水準関数：box(領域の枠線) --------------
531
532 current_par <-
533   par(no.readonly = TRUE)  # パラメータ保存
534
535 par(oma = c(1,1,1,1))      # 外側余白を行数で指定
536 par(mar = c(5,4,1,1))      # 余白を行数で指定
537 par(mfrow = c(2, 2))       # 作図領域を分割
538
539 for (i in 1:4) {
540   plot(x1, y1,
541        frame.plot = FALSE) # 散布図、枠を非表示
542 }
543
544 box(which = "plot")        # (a) プロット領域
545 box(which = "figure")      # (b) 作図領域
546 box(which = "inner")       # (c) 作図領域の集合
547 box(which = "outer")       # (d) デバイス領域
548
549 par(current_par)           # パラメータ復元
```

（my_base_graphics2.R：530-549）

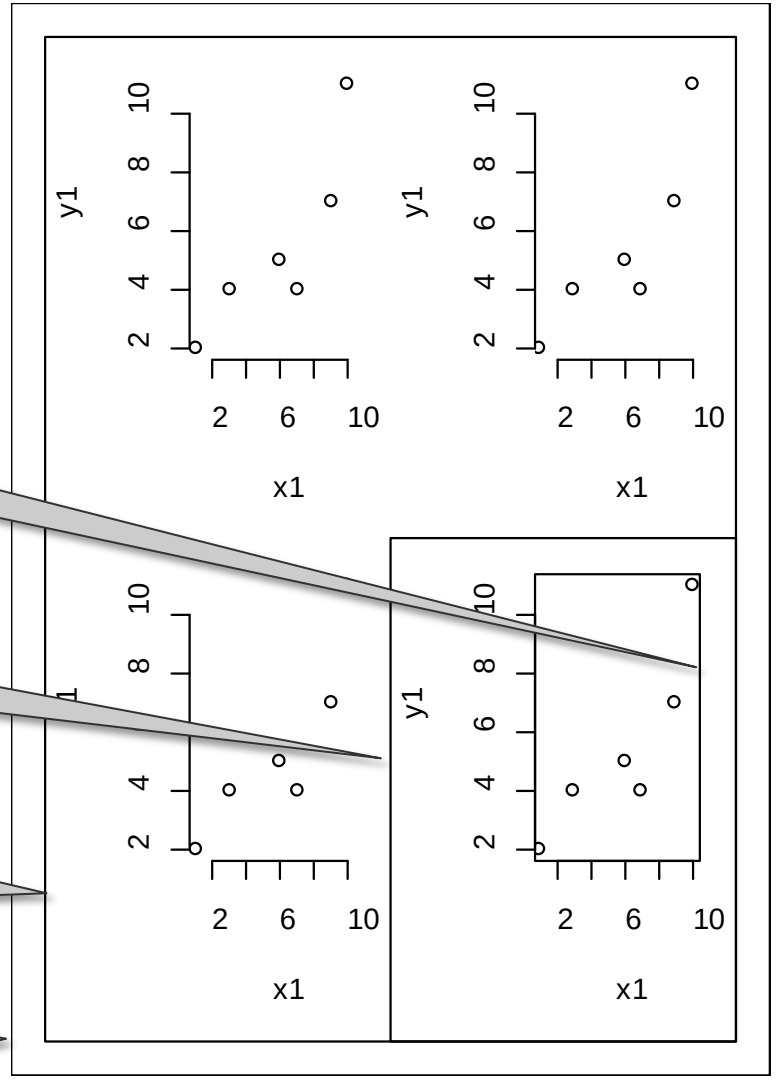
通常の設定では
(c) と (d) は一致

(a) which = "plot"

(b) which = "figure"

(c) which = "inner"

(d) which = "outer"



関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(30) text()（テキスト）

text()：プロット上の任意の座標 (x ,y) に文字列を表示

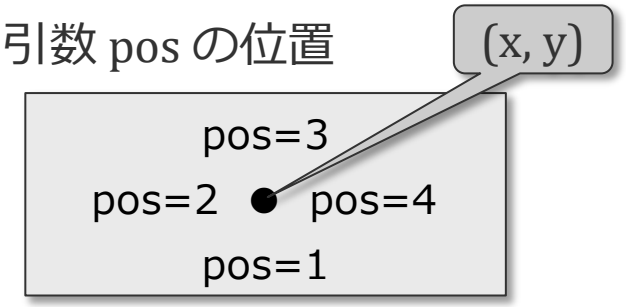
引数　label：表示する文字列、文字ベクトル

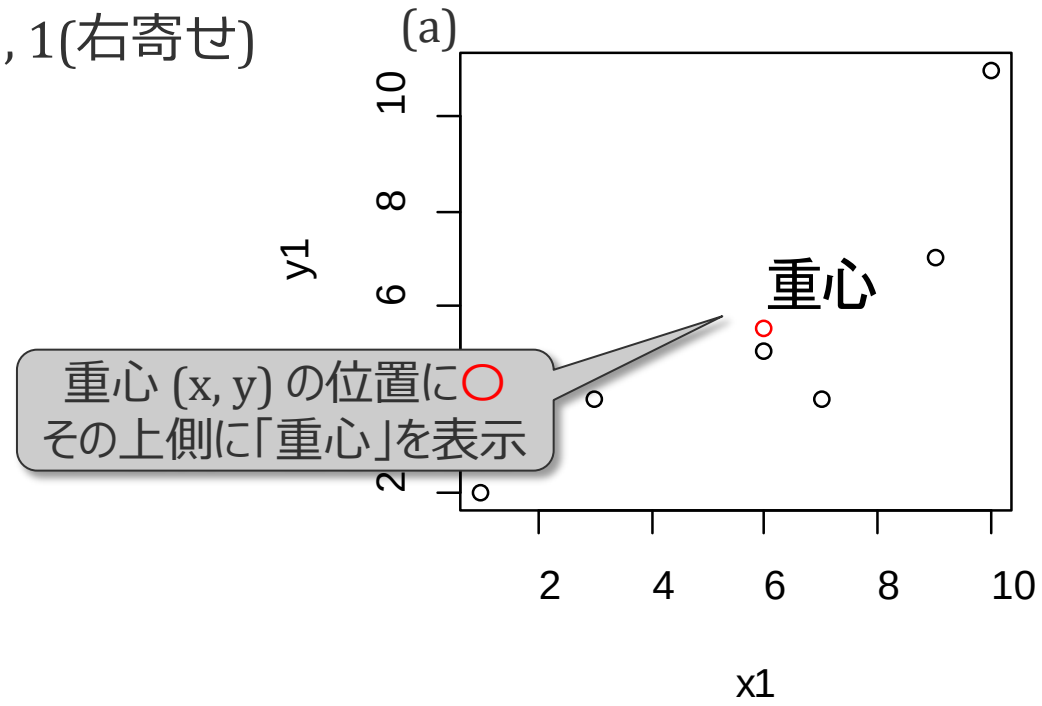
pos：座標の上下左右の位置（1～4）、引数 offset で微調整

adj：文字の位置、0～1、0(左寄せ), 0.5(中央), 1(右寄せ)

(a) 重心の位置の上側に「重心」の文字を表示

```
552 # (30) 低水準関数：text(テキスト) ---------------
553
554 ## (a) 文字の追加
555 m_x <- mean(x1)
556 m_y <- mean(y1)
557
558 plot(x1, y1)            # 散布図
559 points(m_x, m_y, col = "red")
560 text(
561   x = m_x,              # 重心の x 座標
562   y = m_y,              # 重心の y 座標
563   labels = "重心",      # 表示する文字
564   pos = 3,              # 下(1),左(2),上(3),右(4)
565   adj = 0.5,            # 中央に位置、0(左)～1(右)
566   cex = 1.5)            # テキストの表示倍率
```





(my_base_graphics2.R：552-566)

関数 plot() の使い方：低水準関数との組合せ

- 関数 plot() と低水準関数：(30) text()（テキスト）

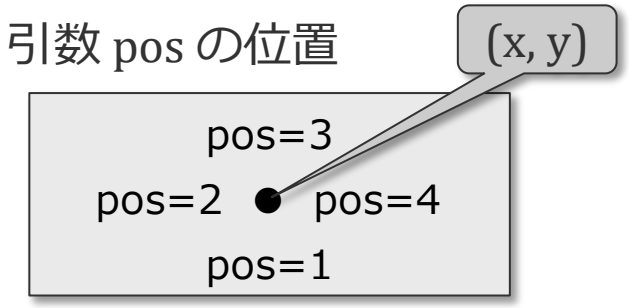
  text()：プロット上の任意の座標 (x ,y) に文字列を表示

  引数　label：表示する文字列、文字ベクトル

  　　pos：座標の上下左右の位置（1～4）、引数 offset で微調整

  　　adj：文字の位置、0～1、0(左寄せ), 0.5(中央), 1(右寄せ)

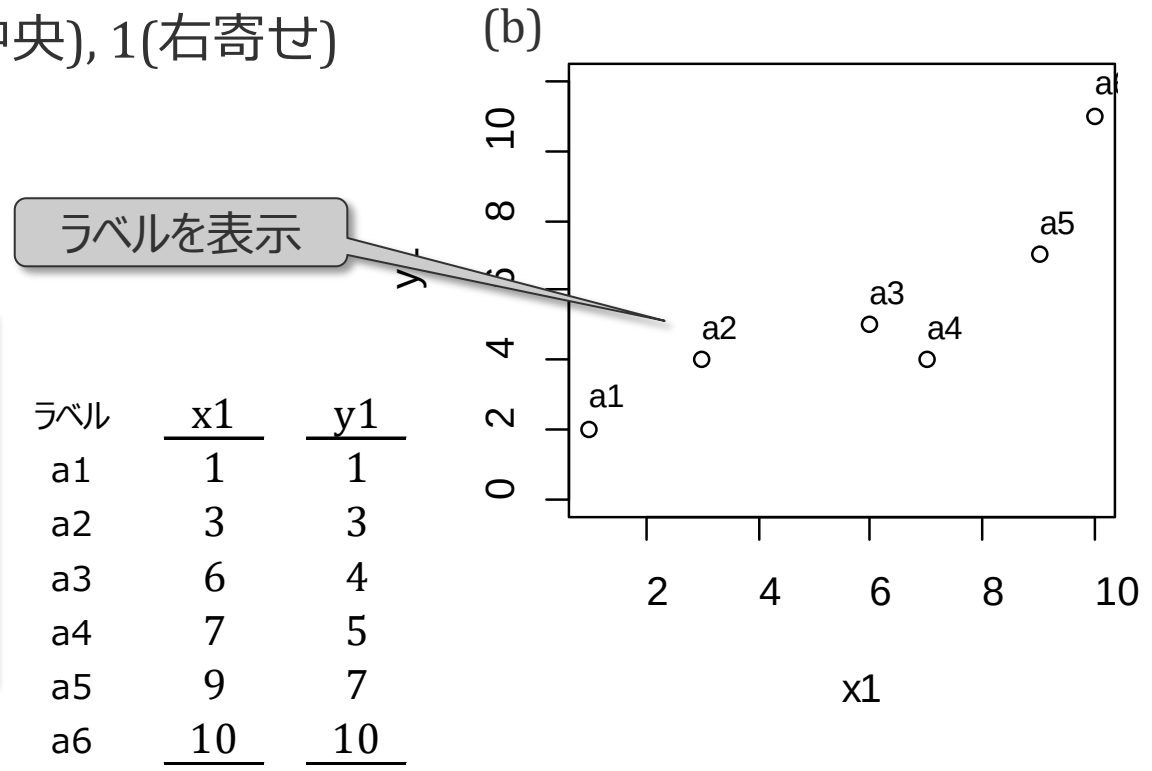
  (b) 散布図の各プロットの上側にラベルを表示




```
567  ## (b) 複数個所での文字の追加
568  plot(x1, y1, ylim = c(0,12))
569  text(
570    x = x1,
571    y = y1,
572    labels = c("a1","a2","a3","a4","a5","a6"),
573    pos = 3,
574    adj = 0.5,
575    cex = 0.8)
```

（my_base_graphics2.R：567-575）

ラベル	x1	y1
a1	1	1
a2	3	3
a3	6	4
a4	7	5
a5	9	7
a6	10	10



関数 plot() の使い方：低水準関数との組合せ

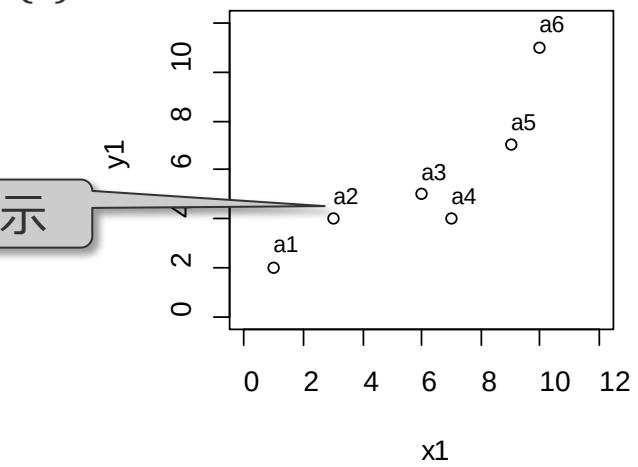
●関数 plot() と低水準関数：(30) text()（テキスト）
(c) プロットした記号に、関数で生成したラベルを表示
(d)プロットした記号に、関数で生成したデータの数値を表示

```
577  ## (c) プロットにラベルを自動生成して表示
578  plot(x1, y1, ylim = c(0,12), xlim = c(0, 12))
579  text(
580    x = x1,
581    y = y1,
582    labels = paste("a", 1:6, sep = ""),
583    pos = 3, cex = 0.8)
584
585  ## (d) プロットにデータの数値を表示
586  num <- round(y1, 1)    # 四捨五入
587  plot(x1, y1, ylim = c(0,12), xlim = c(0, 12))
588  text(
589    x = x1,
590    y = y1,
591    labels = format(num, nsmall = 1),
592    pos = 3, cex = 0.8)
```

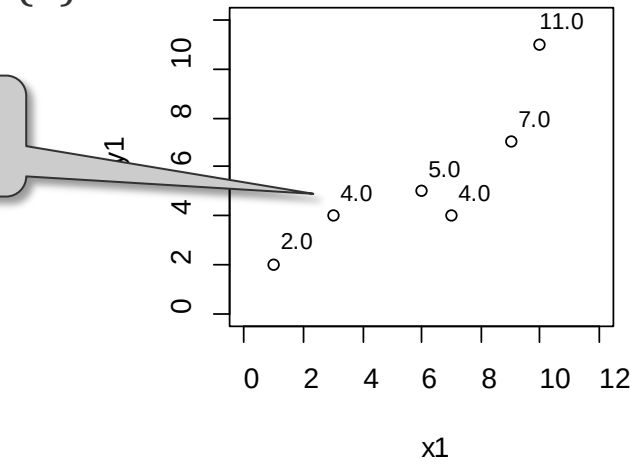
既定値 sep=" " を変更

（my_base_graphics2.R：577-592）

(c) ラベルを自動生成して表示



(d) 数値データを表示



関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(31) mtext()（余白のテキスト）

mtext()：グラフの「余白（margin）」にテキストを追加

引数 text：表示する文字列、文字ベクトルも可

side：1～4、1(下), 2(左), 3(上), 4(右)

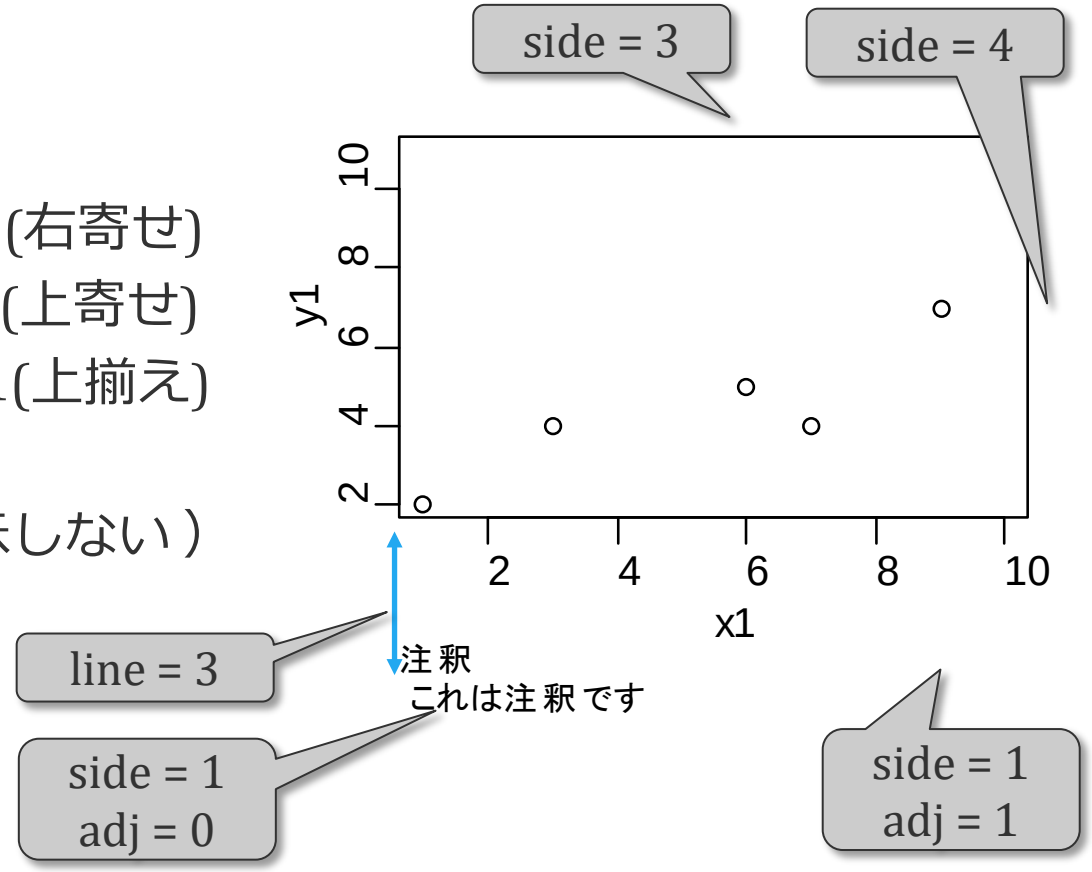
line：文字とグラフの距離、行単位

adi：文字の位置、0～1、0(左寄せ), 0.5(中央), 1(右寄せ)
0(下寄せ), 0.5(中央), 1(上寄せ)

padj：文字の位置、0～1、0(下揃え),0.5(中央),1(上揃え)

outer ：TRUE/FALSE、規定値はFALSE
（外側余白（outer margin）に表示／表示しない）

at：各文字列の位置をユーザー座標系で指定



関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(31) mtext()（余白のテキスト）

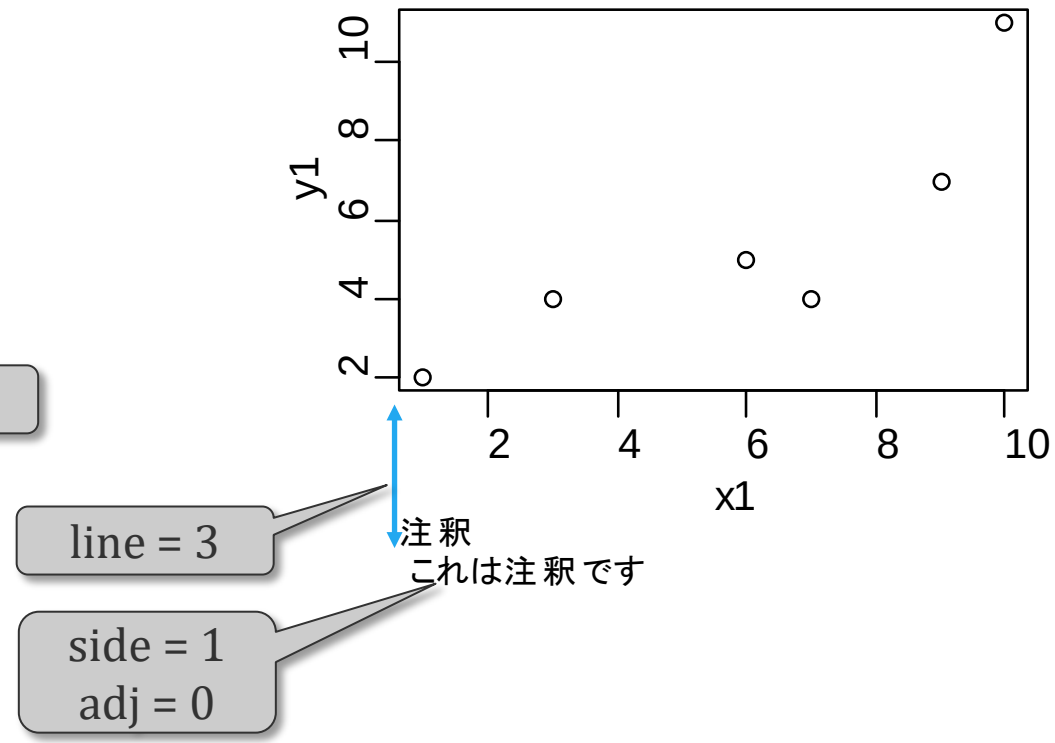
(a) グラフの下側の余白に注釈を追加（軸から 3 行下の左側の位置）

```
595 # (31) 低水準関数：mtext(余白のテキスト) --------
596
597 ## (a) グラフの下部余白に注釈を追加
598 current_par <-
599   par(no.readonly=TRUE)    # パラメータ保存
600 par(mar = c(5, 4, 1, 1))  # 余白を行数で指定
601
602 plot(x1, y1)             # 散布図
603 tx <- "注釈 \n これは注釈です"  # \n：改行
604 mtext(
605   text = tx,             # 表示する文字
606   side = 1,              # 下(1),左(2),上(3),右(4)
607   line = 3,              # プロット領域からの行数
608   adj = 0,               # 揃え位置、0:左or下,1:右or上
609   cex = 0.8)             # テキストの表示倍率
610
611 par(current_par)              # パラメータ復元
```

（my_base_graphics2.R：595-611）

改行

(a) 余白にテキストを表示



関数 plot() の使い方：低水準関数との組合せ

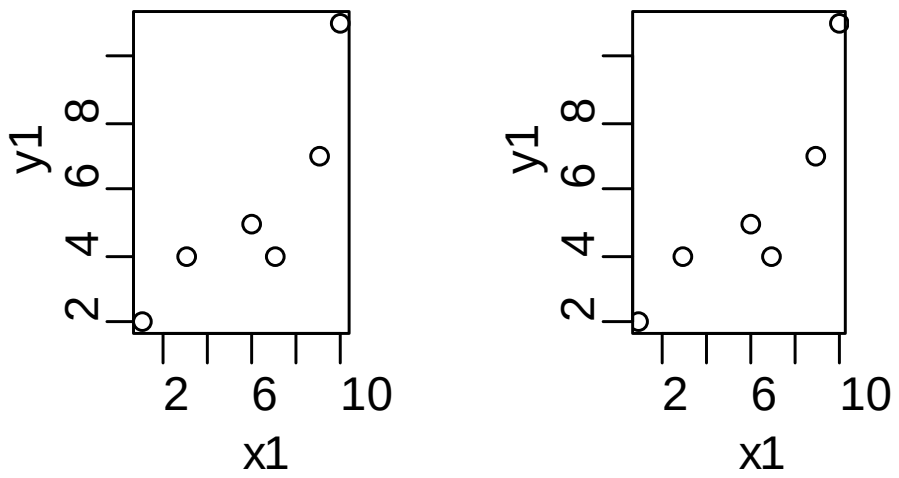
●関数 plot() と低水準関数：(31) mtext()（余白のテキスト）

(b) 複数のグラフに対して全体のタイトルを付ける

```
613  ## (b) 複数のグラフに対して全体のタイトルを付ける
614  ##      （マルチパネル、ファセット）
615  current_par <-
616    par(no.readonly=TRUE)  # パラメータ保存
617
618  par(mar = c(5,4,1,1))     # 全体のタイトル用の余白
619  par(mfrow = c(1, 2))      # 作図領域を分割
620
621  plot(x1, y1)              # 散布図-1
622  plot(x1, y1)              # 散布図-2
623
624  mtext(
625    text ="タイトル",       # 全体のタイトルの文字
626    side = 1,               # 表示位置
627    line = 3,               # プロット領域からの行数
628    at   = -4.5,                # 右の図からの位置
629    cex  = 1.3)              # 全体のタイトルの倍率
630
631  par(current_par)          # パラメータ復元
```

（my_base_graphics2.R：613-631）

(b) 複数のグラフに対して
全体のタイトルを付ける



全体の
タイトル

タイトル

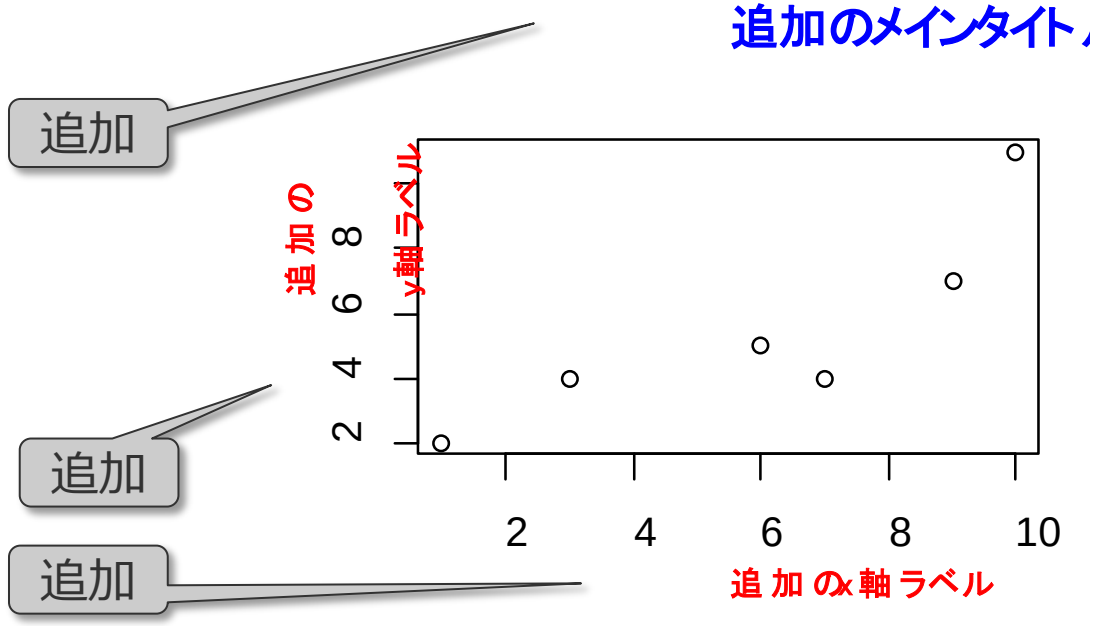
関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(32) title()（タイトル、軸ラベル）

title()：plot() での指定とは別に、タイトルと軸ラベルのカスタマイズを行う
　　作図後に追加や変更が容易、軸ラベルやタイトルを別々に調整可、複数のタイトルを追加

引数　main, sub, xlab, ylab：それぞれの文字列、col.lab, col.main, col.sub, cex.lab, font.lab　など

```
634 # (32) 低水準関数：title（タイトル）-------------
635
636 current_par <-
637   par(no.readonly = TRUE) # パラメータ保存
638 par(mar = c(4,4,3,1))
639
640 plot(x1, y1,                  # 散布図
641      main = "", xlab = "", ylab = "")
642
643 title(main = "追加のメインタイトル",
644       xlab = "追加のx軸ラベル ",
645       ylab = "追加のy軸ラベル",
646       font.lab = 2,          # ラベルのフォント指定
647       cex.lab = 0.8,         # ラベルのサイズ指定
648       col.lab = "red",       # ラベルの色指定
649       col.main = "blue",     # タイトルの色指定
650       line = 2)              # 軸からの距離(行数)
651
652 par(current_par)             # パラメータ復元
```



（my_base_graphics2.R：634-652）

関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(33) legend()（凡例）

legeng()：複数系列のデータを１つのグラフにプロット、その凡例をプロット領域に表示

引数　x, y：凡例を表示する座標（凡例の左上の角の座標）、または x に文字列を渡す

x ="topright", "topleft", "bottomright", "bottomleft", "top", "bottom", "right", "left", "center"

inset：凡例を表示する位置の微調整、例　inset = 0.05 など

legend：凡例のテキスト　例　legend = c("グループ A", "グループ B")

col, pch, lty, lwd, fill：グラフで使用したパラメータと同じものを指定して対応関係を表示

bty, bg, title：凡例ボックスの枠線の種類と背景色、凡例のタイトルの文字例

horiz, ncol：horiz = TRUE で凡例を横並び、ncol = 2 で凡例の列数を 2 列で表示

３系列（A, B, C）の事例

色で識別	legend(x, y, legend = c("A", "B", "C"), col = c("orange", "blue", green"))
記号の形状で識別	legend(x, y, legend = c("A", "B", "C"), pch = c(16, 17, 18))
線種で識別	legend(x, y, legend = c("A", "B", "C"), lty = c("solid", "dashed", "dotted"))
塗りつぶしの色	legend(x, y, legend = c("A", "B", "C"), fill = c("red","blue","green") ← 棒グラフ等

関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(33) legend()（凡例）

legeng()：複数系列のデータを１つのグラフに表示して、凡例で識別

引数　x, y：凡例を表示する座標（凡例の左上の角の座標）、または x に文字列を渡す

x ="topright", "topleft", "bottomright", "bottomleft", "top", "bottom", "right", "left", "center"

inset：凡例を表示する位置の微調整、例　inset = 0.05 など

legend：凡例のテキスト　例　legend = c("グループ A", "グループ B")

col, pch, lty, lwd, fill：グラフで使用したパラメータと同じものを指定して対応関係を表示

bty, bg, title：凡例ボックスの枠線の種類と背景色、凡例のタイトルの文字例

horiz, ncol：horiz = TRUE で凡例を横並び、ncol = 2 で凡例の列数を 2 列で表示

３系列（A, B, C）の事例

グラフと一致させる

色で識別　legend(x, y, legend = c("A", "B", "C"), col = c("orange", "blue", green"))

記号の形状で識別　legend(x, y, legend = c("A", "B", "C"), pch = c(16, 17, 18))

線種で識別　legend(x, y, legend = c("A", "B", "C"), lty = c("solid", "dashed", "dotted"))

塗りつぶしの色　legend(x, y, legend = c("A", "B", "C"), fill = c("red","blue","green") ← 棒グラフ等

関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(33) legend()（凡例）

(a) 2 系列のデータを 1 つのグラフに表示して、凡例で識別

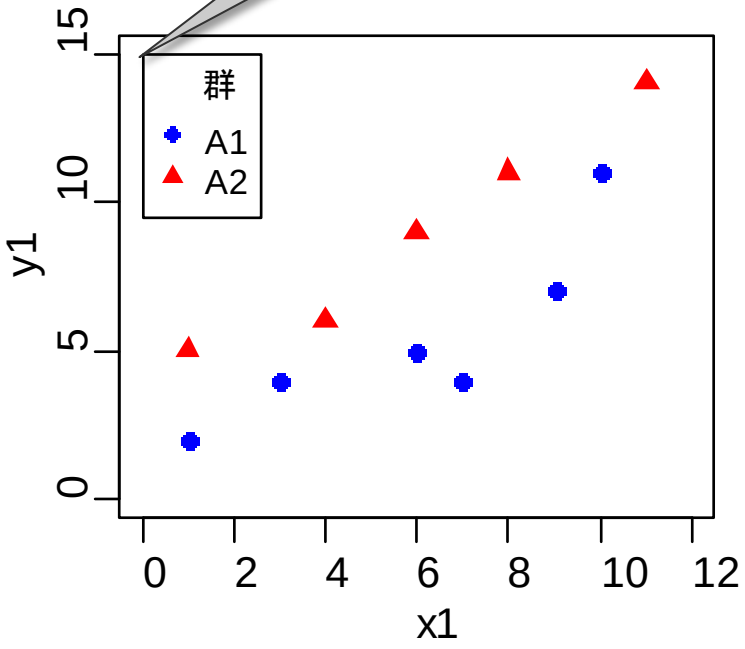
```
655 # (33) 低水準関数：legend(凡例) ----------------
656
657 ## (a) 2 系列(2グループ)を表示(1)
658 ###系統1： x1, y1、系統2：x2, y2
659 x2 <- c(1, 4, 6, 8, 11)
660 y2 <- c(5, 6, 9, 11, 14)
661
662 plot(x1, y1,                  # 散布図、系列 1
663      xlim = c(0, 12), ylim = c(0, 15),
664      pch = 16,
665      col = "blue")
666
667 points(x2, y2,                # 散布図、系列 2
668        pch = 17,
669        col = "red")
670
671 legend(x = 0, y= 15,          # 凡例を追加
672        title =  "群",
673        legend = c("A1", "A2"),
674        col = c("blue", "red"),
675        pch = c(16, 17),
676        cex = 0.8)
```

冒頭に設定　系統 x1, y1
新規に設定　系統 x2, y2

凡例のスペース確保のため
範囲を広くとる場合もある

系統 1	系統2
A1	A2
pch = 16	pch = 17
col = "blue"	col = "red"

左上の角
x = 0
y = 15



（my_base_graphics2.R：655-676）

関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(33) legend()（凡例）

(b) 凡例の位置を関数 locator() で指定

```
678  ## (b) 関数 locator() の利用
679  plot(x1, y1,           # 散布図、系列１
680       xlim = c(0, 12),
681       ylim = c(0, 20),
682       col = "blue")
683
684  points(x2, y2,         # 散布図、系列２
685         pch = 17,       # 659～660行目でデータを設定済
686         col = "red")
687
688  pos <- locator(1)      # 凡例の位置となる＋をクリック
689
690  legend(pos,            # 凡例を表示
691         title =  "群",
692         legend = c("A1", "A2"),
693         col = c("blue", "red"),
694         pch = c(16, 17),
695         cex = 0.8)
696
697  print(pos)  # 得られた値を690行目のposと書換
```

（my_base_graphics2.R：678-697）

(b)
＋ 印をクリック

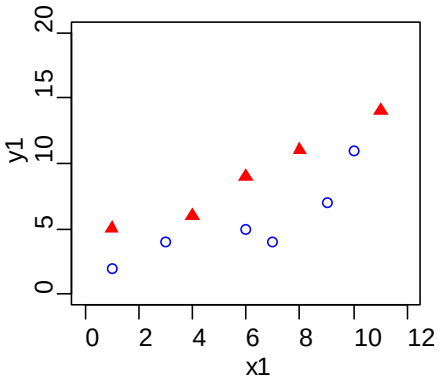
(d)
値の確認

再現性を確保するために
pos （2 要素のベクトル）の
値を確認・保存

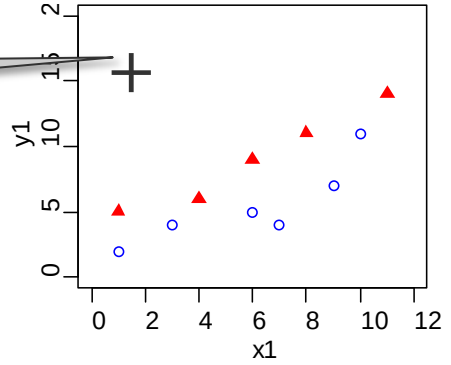
(b-4) 値の確認

```
> print(pos)
$x
[1] 0.9729784

$y
[1] 17.05515
```

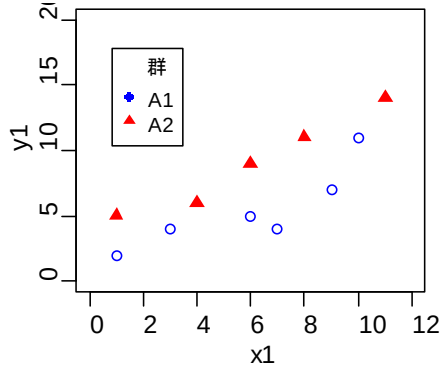
(b-1)



(b-2)



(b-3)



関数 plot() の使い方：低水準関数との組合せ

●関数 plot() と低水準関数：(34) axis()（軸）

axis()：既存のグラフに軸（Axis）を追加、既存の軸を細かくカスタマイズ

引数　side：軸の設定、必須、1（下, x 軸）、2（左, y 軸）、3（上, 横軸）、4（右, 縦軸）

at：目盛の座標、数値ベクトル、指定しないと自動的に設定される（既定値 NULL）

labels：目盛ラベル、TRUE/FALSE（atで指定した数値を使用(規定値)/表示なし）

at と同じ長さの文字ベクトルを渡すと文字の目盛ラベルを表示

tick：TRUE/FALSE（目盛線を表示(規定値)／非表示）

pos：軸線自体を描画する位置を、もう一方の軸の座標で指定

las：目盛ラベルの向き、0（軸と平行(既定値)）、1（水平）、2（軸に垂直）、3（垂直）

cex.axis：目盛ラベルのサイズを指定

col, col.tick：軸線の色、目盛線の色

既定値：axis(side, at = NULL, labels = TRUE, tick = TRUE, ...)

関数 plot() の使い方：低水準関数との組合せ

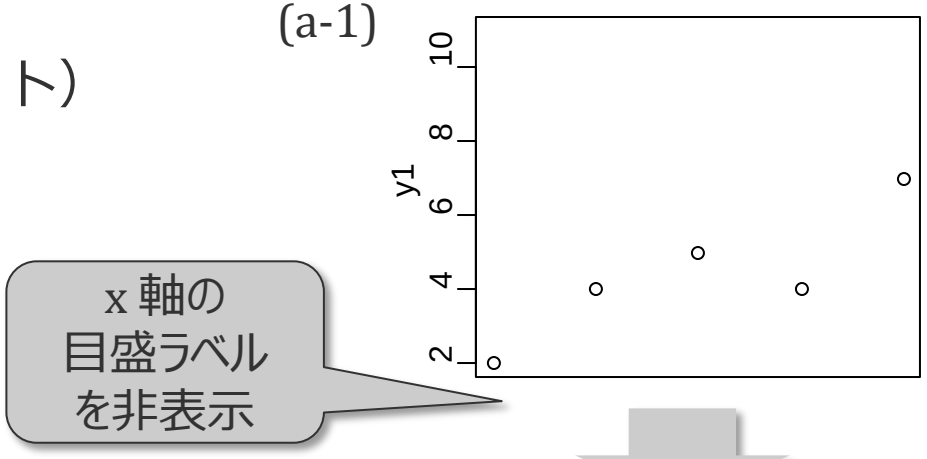
- 関数 plot() と低水準関数：(34) axis()（軸）

  (a) x 軸の座標 1～5 に A～E を表示（インデックスプロット）

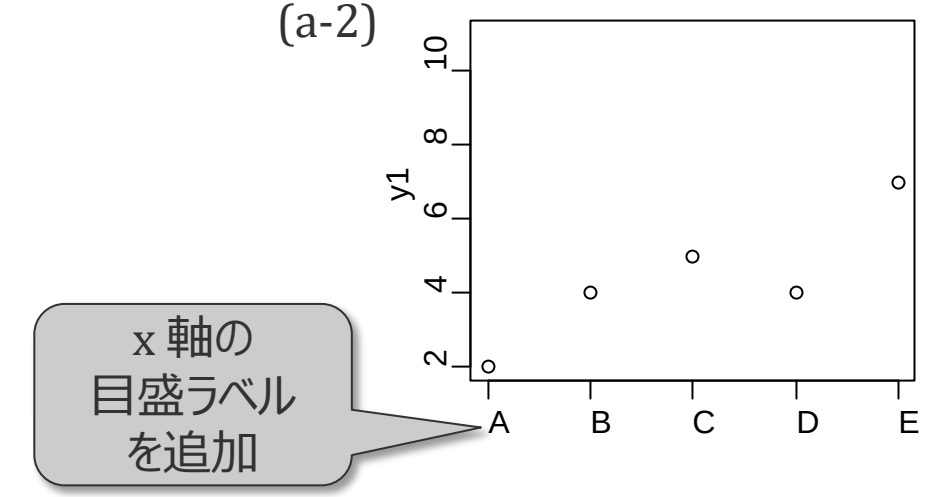
```
700 # (34) 低水準関数：axis(x軸, y軸の調整) ----------
701
702 ## (a) x軸の座標 1～5 に A～E を割付
703 plot(y1,                    # 散布図、x軸は行番号
704      xlim = c(1, 5),        # 座標：1,2,3,4,5
705      xaxt = "n",            # x軸の非表示
706      xlab = "")             # x軸ラベルの非表示
707
708 axis(side = 1,              # 1:下,2:左,3:上,4:右
709      at = 1:5,              # x座標の位置
710      labels = c("A", "B", "C", "D", "E"))
```

（my_base_graphics2.R：700-710）

(a-1)



(a-2)



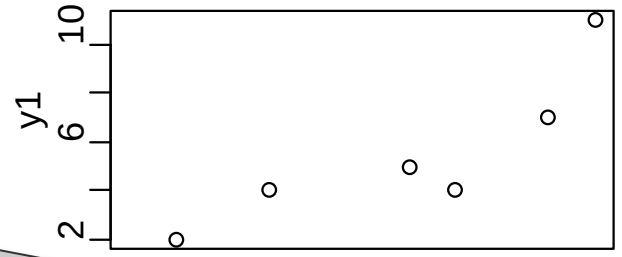
関数 plot() の使い方：低水準関数との組合せ

- 関数 plot() と低水準関数：(34) axis()（軸）
  - (b) x 軸の 0～10 の位置に 1 刻みで目盛線を追加
    x 軸の 0, 5, 10 の位置に目盛ラベルを追加

```
712  ## (b) x軸の位置1,3,5に0,2,4を割付、目盛線を追加
713  plot(x1, y1,                 # 散布図
714       xlim = c(0, 10),        # x座標の範囲
715       xaxt = "n",             # x軸の非表示
716       xlab = "")              # x軸ラベルの非表示
717
718  axis(side = 1,               # 下(x軸)を指定
719       at = 0:10,              # x座標の位置
720       labels = FALSE)         # 目盛線のみを表示
721
722  axis(side = 1,               # 下(x軸)を指定
723       at = c(0,5,10),         # x座標の位置
724       labels = c(0,5,10)) # x軸の目盛ラベルを表示
```

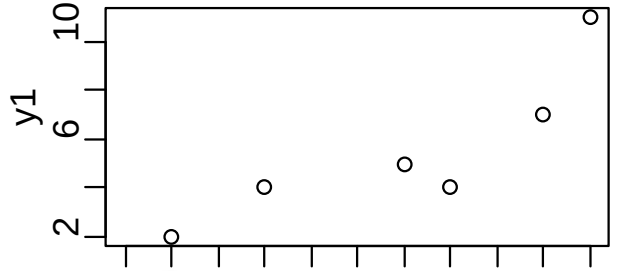
（my_base_graphics2.R：712-724）

(b-1)



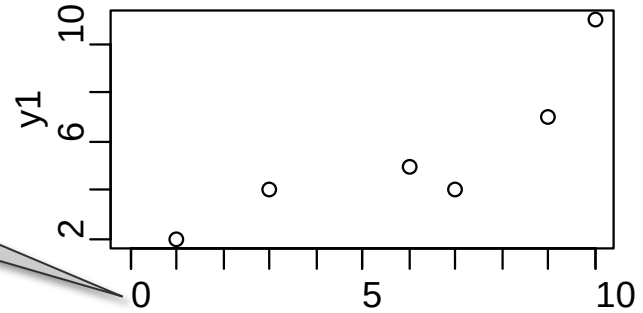
x 軸の
目盛ラベル
を非表示

(b-2)



目盛線の間隔を1

(b-3)



目盛ラベルの間隔を 5

関数 plot() の使い方：低水準関数との組合せ

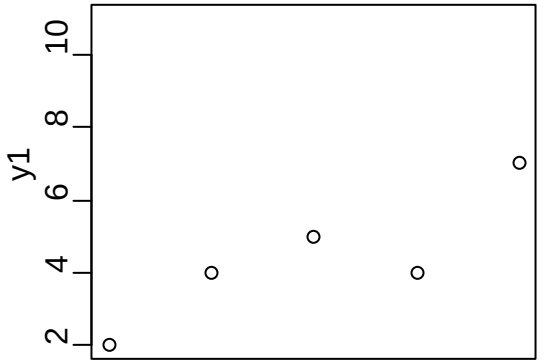
●関数 plot() と低水準関数：(34) axis()（軸）

(c) x 軸の座標 1～5 に x1～x9 を自動生成して表示
（インデックスプロット）

paste()　　：文字列を規則的に生成する関数

seq(1, 9, 2)　：数列を規則的に生成する関数
1 から 9 まで 2 刻みの数列を生成

(c-1)

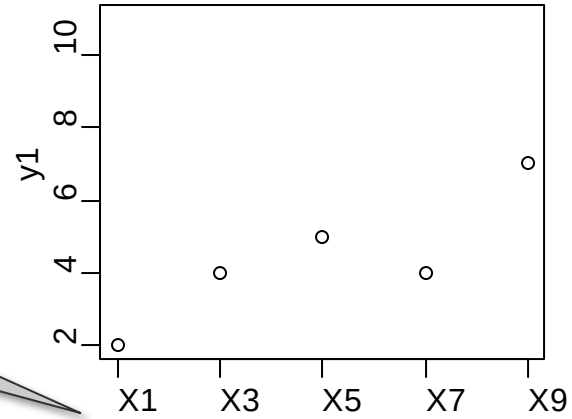



```
726  ## (c) x軸の位置 1～5 までに、x1～x9 を割付
727  plot(y1,                    # 散布図 x軸は行番号
728       xlim = c(1, 5),
729       xaxt = "n",            # x軸の非表示
730       xlab = "")             # x軸ラベルの非表示
731
732  axis(side = 1,              # 下(x軸)を指定
733       at = 1:5,
734       labels = paste0("X", seq(1, 9, 2)))
```

（my_base_graphics2.R：726-734）

x 軸の
目盛ラベル
自動的に生成

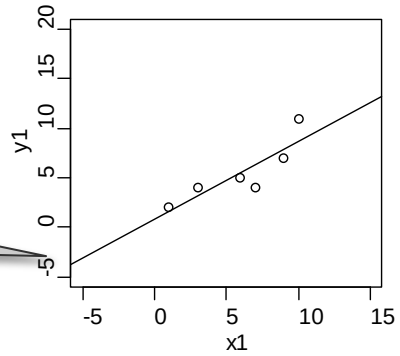
(c-2)



関数 plot() の使い方：低水準関数との組合せ

- 関数 plot() と低水準関数：(34) axis()（軸）

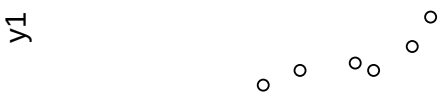
  (d) x 軸と y 軸の交わる位置を指定



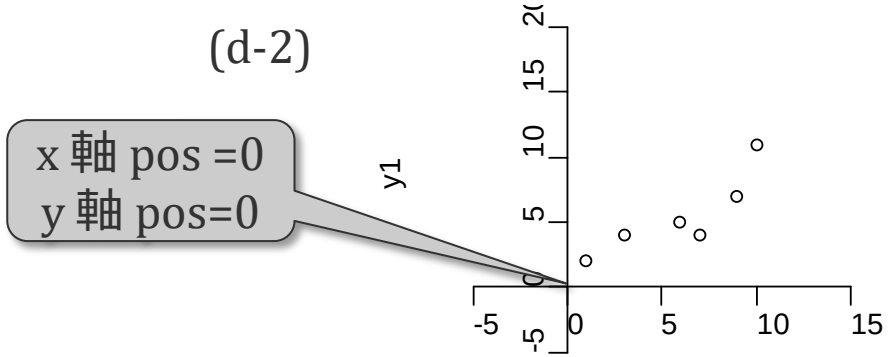

```
736  ## (d) 軸(x軸またはy軸)の位置を指定
737  lm_out <- lm(y1 ~ x1)     # 回帰分析
738
739  plot(x1, y1,              # 散布図
740       xlim = c(-5, 15),    # x軸の範囲を負まで拡大
741       ylim = c(-5, 20),    # y軸の範囲を負まで拡大
742       axes = FALSE)        # 軸を非表示
743
744  axis(side = 1, pos = 0)   # x軸を y=0 の位置に設置
745  axis(side = 2, pos = 0)   # y軸を x=0 の位置に設置
746
747  abline(lm_out)            # 回帰直線を追加
```

（my_base_graphics2.R：736-747）

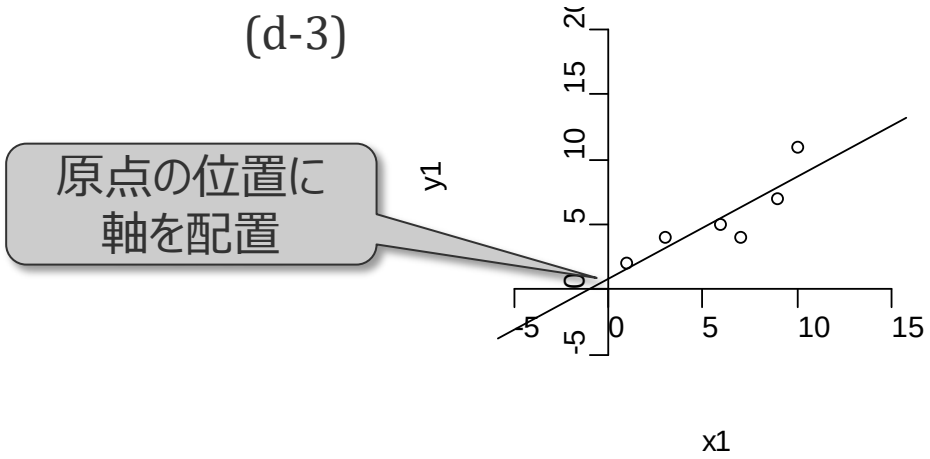
(d-1)



(d-2)



(d-3)



関数 plot() の使い方：低水準関数との組合せ

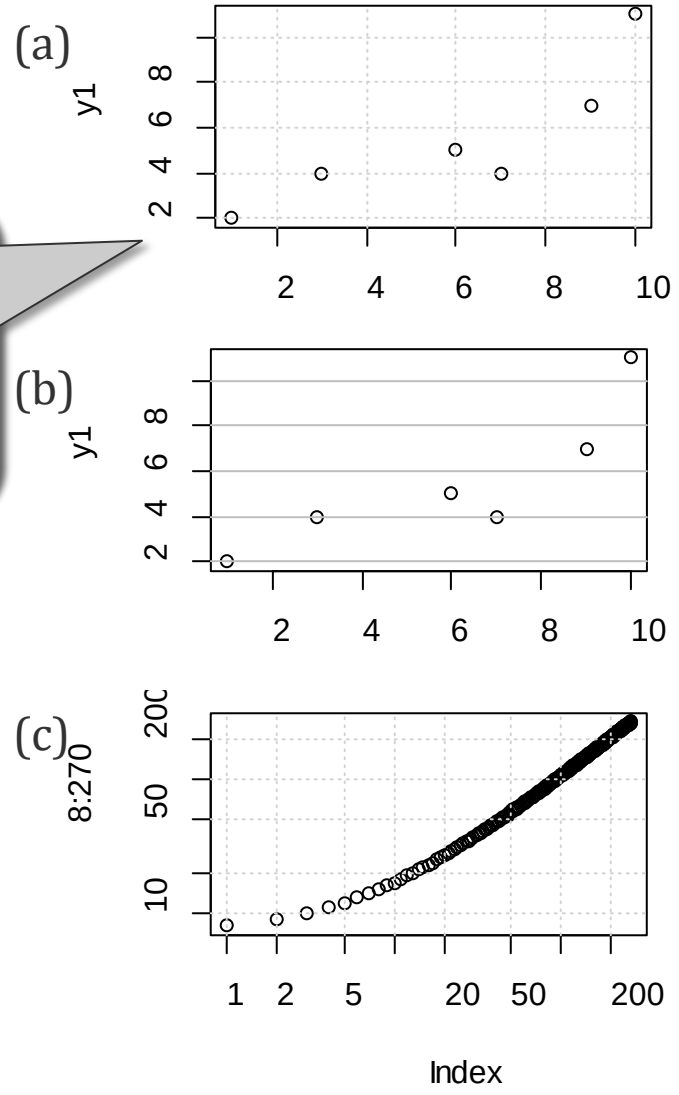
●関数 plot() と低水準関数：(35) grid()（グリッド線）

グリッド線（水平線、垂直線）の追加

```
750 # (35) 低水準関数：grid(グリッド線) -------------
751
752 ## (a) 自動設定
753 plot(x1, y1)                 # 散布図
754 grid()                       # 引数を省略、自動設定
755
756 ## (b) グリッドの水平線と垂直線の選択
757 plot(x1, y1)                 # 散布図
758 grid(nx = NA,                # 垂直線は非表示
759      ny = NULL,              # 水平線は自動設定
760      col = "gray",           # グリッド線の色
761      lty = "solid",          # グリッド線の形状
762      lwd = 1.5)              # グリッド線の太さ
763
764 ## (c) 対数目盛のグリッド線
765 plot(8:270, log="xy")
766 grid(equilogs = FALSE)
```

（my_base_graphics2.R：750-766）

自動設定（既定値）
nx = NULL：自動
ny = NULL：自動
col = "lightgray"
lty = "dotted"



関数 plot() の使い方：Help の見方

●関数 plot() のヘルプ

パッケージ base に属する plot() は、渡されたオブジェクトの種類（クラス）に基づいて、
パッケージ graphics とパッケージ stats に属する関数を自動的に呼び出す（概略）

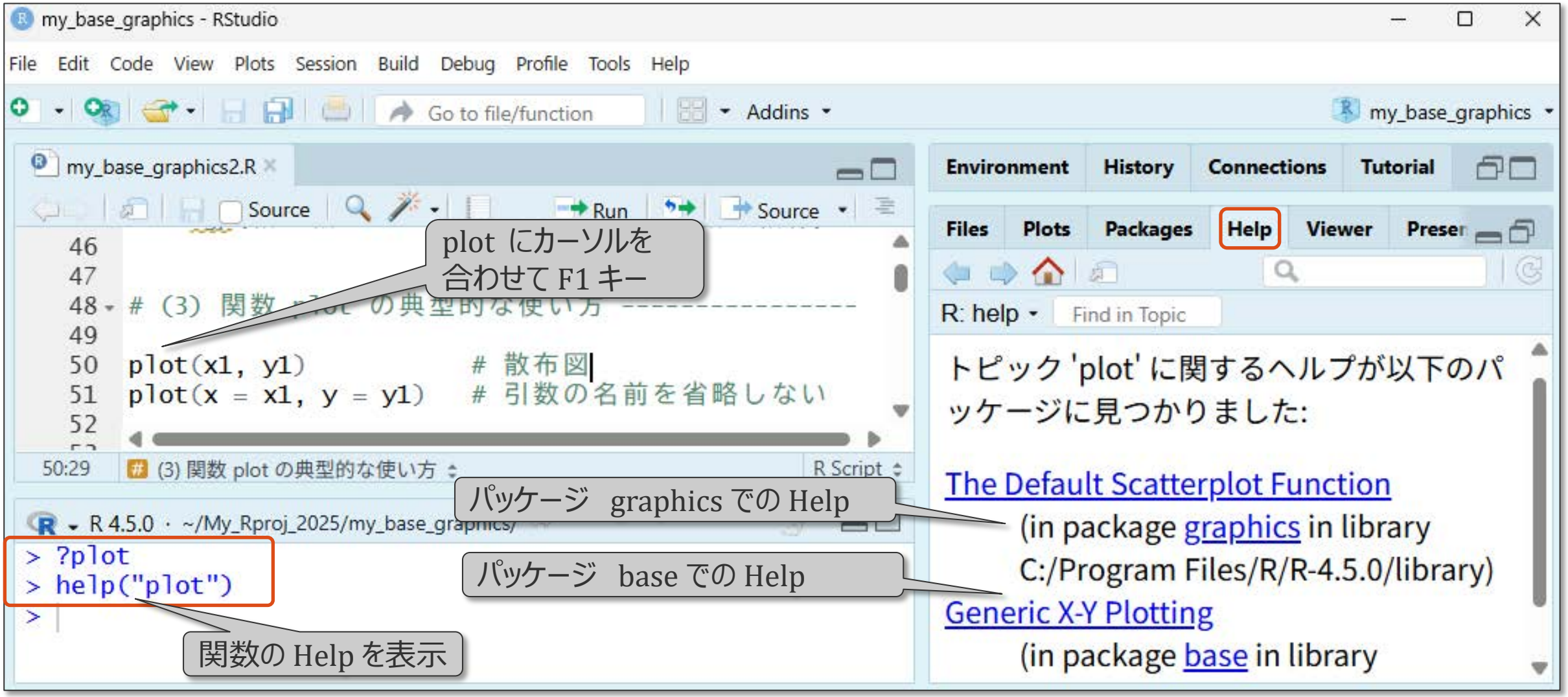
注）R 4.0.0 で、plot() の本体は graphics から base に移動した

関数 plot に関するヘルプの参照箇所

パッケージ	主な関数	役割、plot に渡すオブジェクト	Help の内容の一部
base	plot	plot の本体、以下の関数に振分	plot の概要
graphics	plot.default	基本的な描画	基本的なグラフィックスパラメータの説明
	plot.formula	formulaオブジェクト	式(y~x) を渡す場合の関数の対応
	plot.data.frame	データフレーム・オブジェクト	データフレーム全体を渡す場合の関数の対応
stats	plot.lm	lm, glmオブジェクト	線形診断プロットの種類と選択の方法
	plot.ts	ts(時系列)オブジェクト	時系列データ特有のパラメータ
	plot.density	deysityオブジェクト(密度推定)	密度曲線のパラメータ
graphics	par	グラフィックスパラメータの設定	関数に共通したグラフィックスパラメータの使い方

注）関数 par は関数 plot ではない。par で説明されるパラメータの多くは、plot の引数として直接指定することも可能
(例：col, pch, lty, cex等)。

関数 plot() の使い方：Help の見方

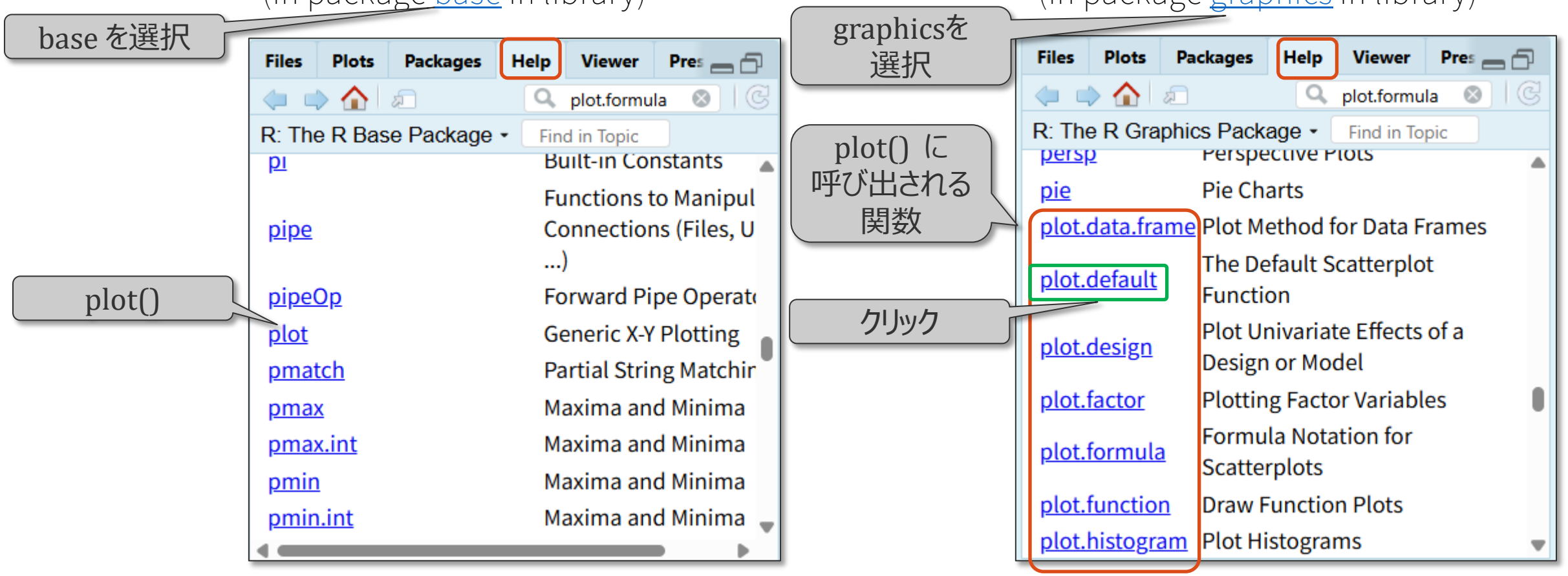


関数 plot() の使い方：Help の見方

●関数 plot() のヘルプ

Generic X-Y Plotting
(in package base in library)

The Default Scatterplot Function
(in package graphics in library)



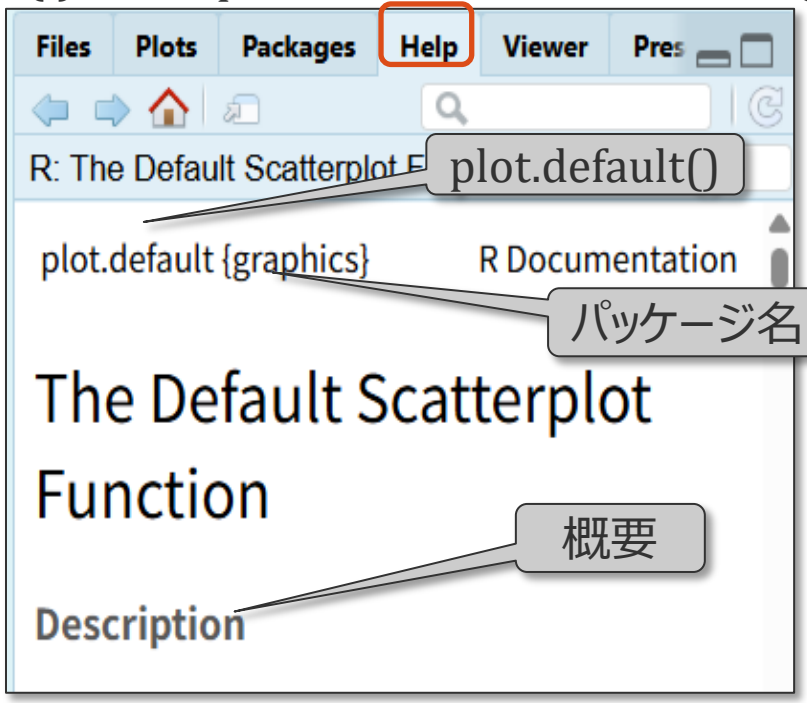
関数 plot() の使い方：Help の見方

●関数 plot() のヘルプ

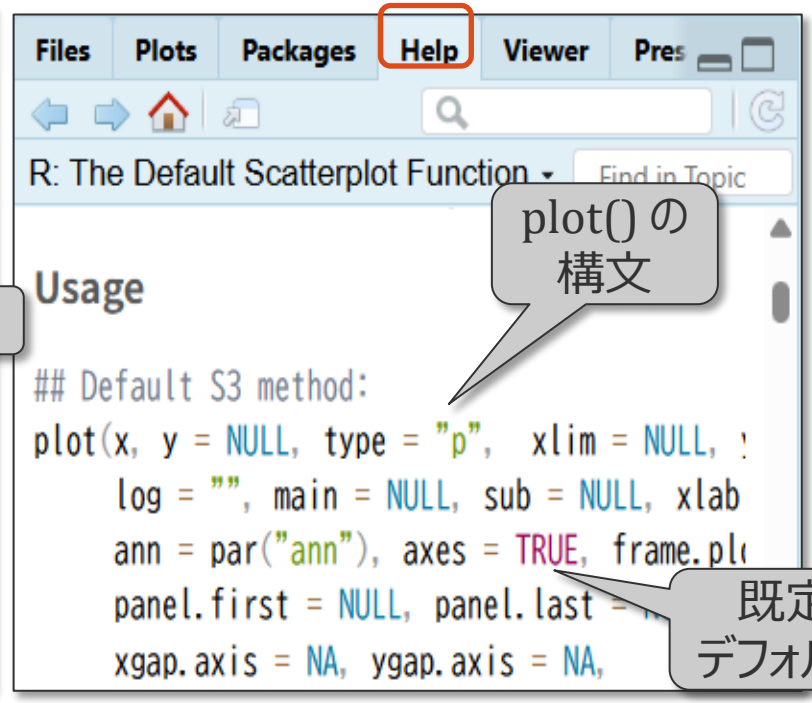
関数 plot.default() で、関数 plot() の Help を参照

Description, Usage, Arguments, Details, Value, Rferences, See Also, Example の項目に分けて表示

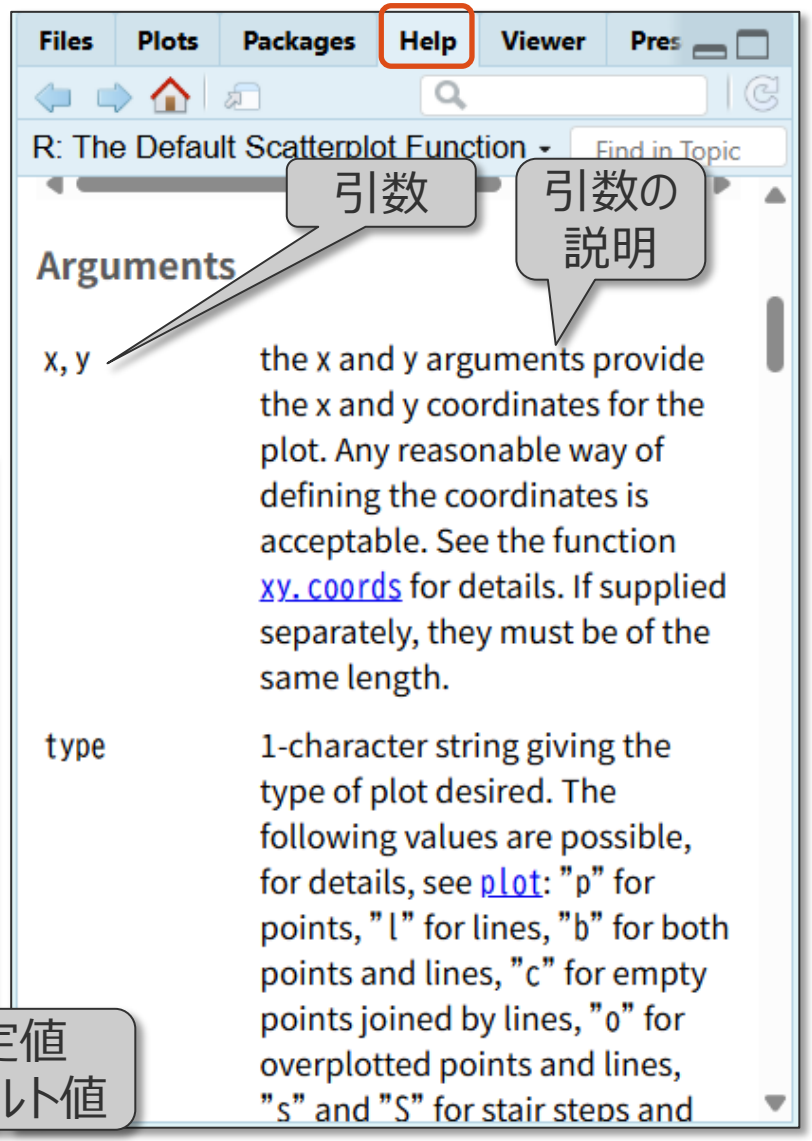
(i) Description（説明）



(ii) Usage（使用方法）

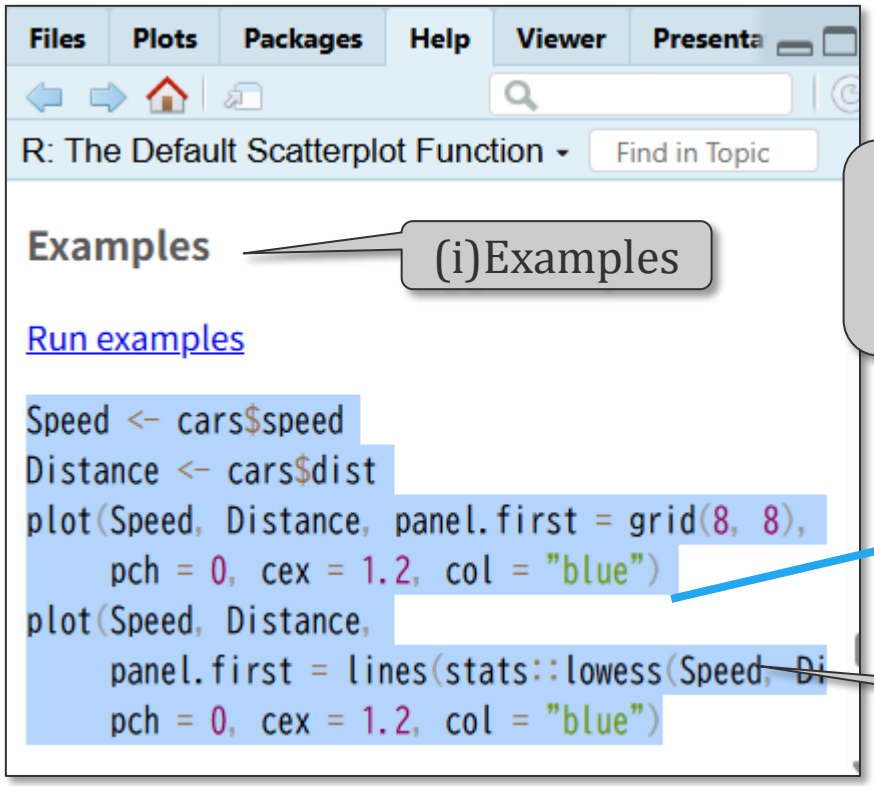


(iii) Arguments（引数）



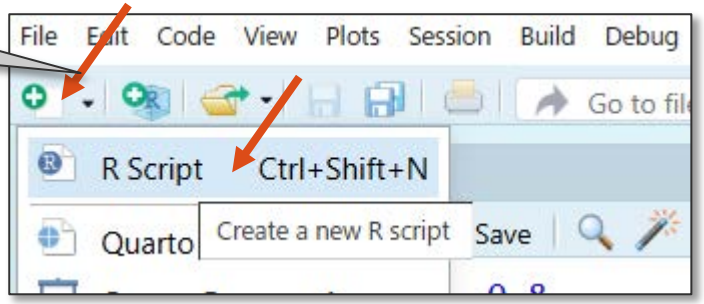
関数 plot() の使い方：Help の見方

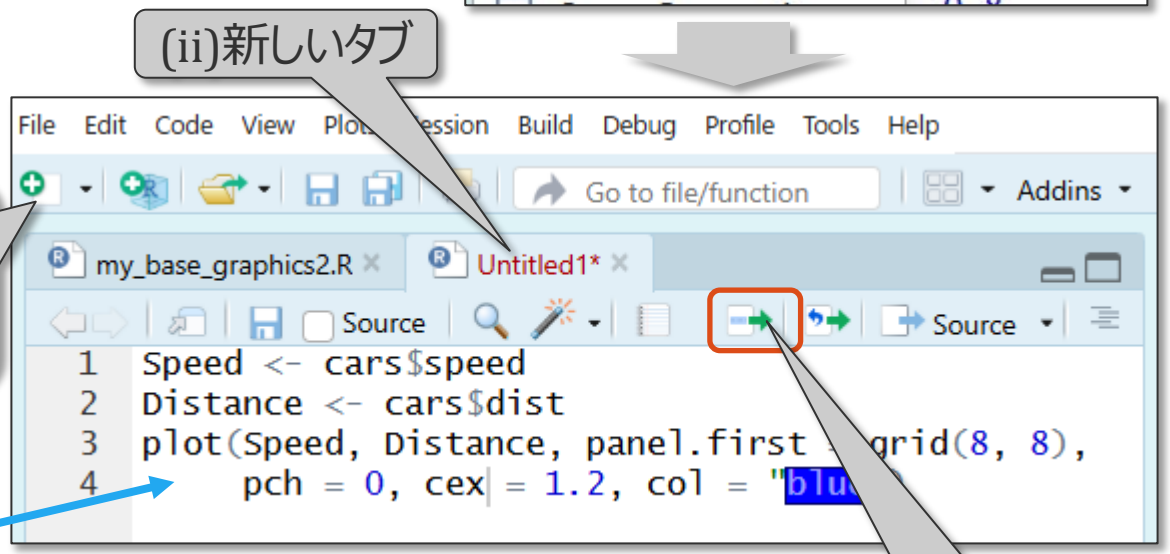
- 関数 plot() のヘルプ
  - Examples
    - サンプルコードを RStudio に貼付けて実行
    - コードと実行結果から、関数の使い方を確認



(i)Examples

(ii)クリック
[new file]
＞[R Script]





(iii)サンプルコードをコピー

(iii) 範囲選択

(iv) 実行

5　標準関数によるグラフ作成
（パッケージ graphics, stats のグラフィックス関数）

高水準グラフィックス関数の事例
使い方の詳細は省略（関数のヘルプを参照）
（グラフィックスパラメータはデフォルメした設定）

標準関数によるグラフ作成：準備

●グラフ作成用の標準の高水準関数

散布図、条件付き散布図、複数系列の散布図、散布図行列

サンフラワープロット　　記号プロット

箱ひげ図　　１次元散布図

ヒストグラム　　幹葉図

棒グラフ　　Cleveland ドットチャート

円グラフ　　モザイク図

スパインプロット　　スピノグラム

Cohen-Friendly の連関図　　4分割プロット

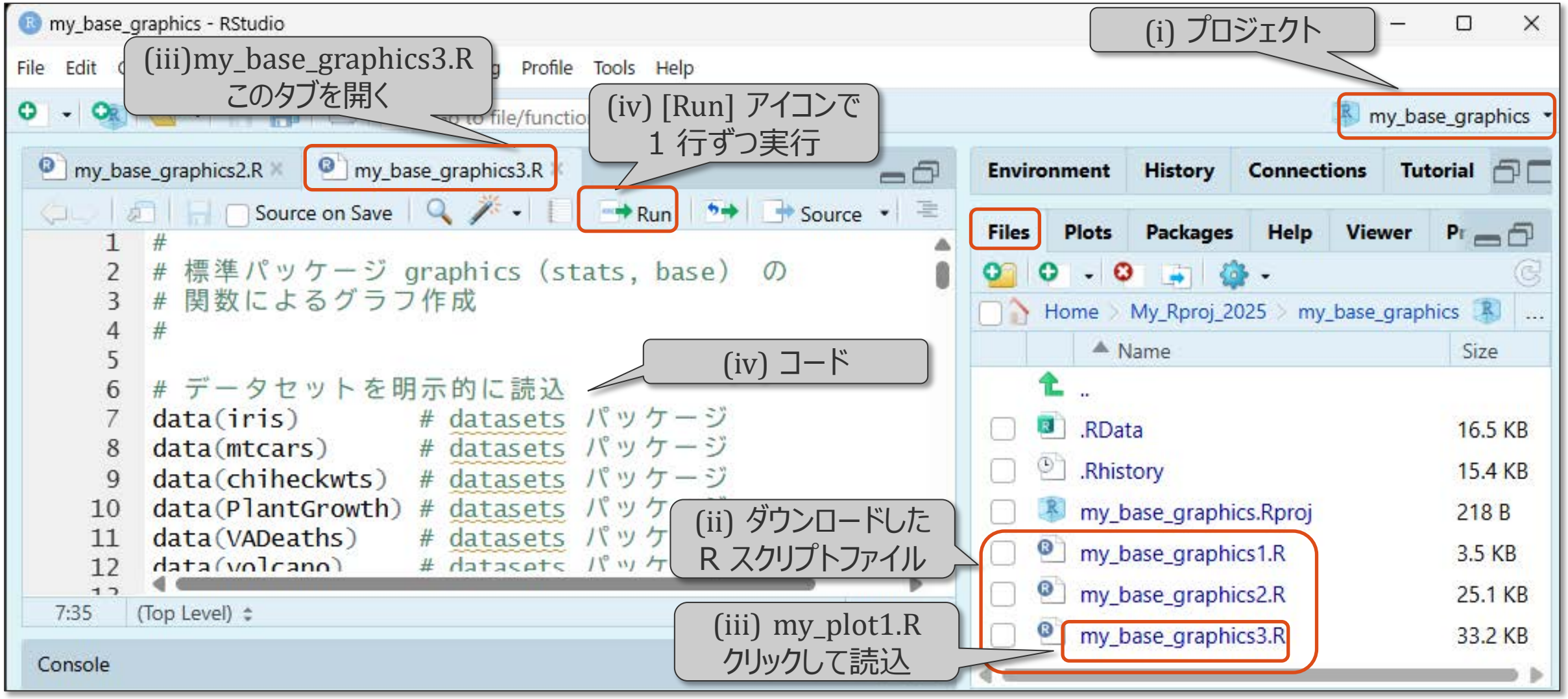
２標本 Q-Q プロット　　１標本 Q-Q プロット

等高線プロット　　3D曲面プロット

ヒートマップ　　関数のプロット

主な関数名	パッケージ名	グラフ名（主な用途）
plot	graphics 等	散布図、線グラフ、棒グラフ、箱ひげ図
coplot	graphics	条件付き散布図、線グラフ
matplot	graphics	複数系列の散布図、線グラフ
pairs	graphics	散布図行列
sunflowerplot	graphics	サンフラワープロット
symbols	graphics	シンボルプロット
boxplot	graphics	箱ひげ図
stripchart	graphics	１次元散布図
hist	graphics	ヒストグラム
stem	graphics	幹葉図
barplot	graphics	棒グラフ
dotchart	graphics	Cleveland ドットチャート
pie	graphics	円グラフ
mosaicplot	graphics	モザイク図
spineplot	graphics	スパインプロット、スピノグラム
assocplot	stats	Cohen-Friendly の連関図
fourfoldplot	stats	4分割プロット
qqplot	stats	２標本 Q-Q プロット
qqnorm	stats	１標本 Q-Q プロット
stars	graphics	スターチャート（レーダーチャート）
contour	graphics	等高線プロット
persp	graphics	3D曲面プロット（３次元透視図）
image	graphics	ヒートマップ
curve	graphics	関数のプロット

標準関数によるグラフ作成：準備



標準関数によるグラフ作成：準備

●事前の設定

(i) データとして利用するデータセットを、関数 data() により明示的に読込
　　このコードは必須ではないが、明示性と安全性のために使われる慣習

(ii) 関数 par() による余白と軸の設定

```
#
# 標準パッケージ graphics(stats, base) の
# 関数によるグラフ作成
#

# データセットを明示的に読込、パッケージ datasets
data(iris)              # データフレーム
data(mtcars)            # データフレーム
data(chickwts)          # データフレーム
data(PlantGrowth)       # データフレーム
data(VADeaths)          # マトリックス（行列）
data(volcano)           # マトリックス（行列）
data(HairEyeColor)      # テーブル・オブジェクト

# 余白の設定、軸の設定(余白を狭く設定)
par(mar = c(4, 5, 2, 2), mgp = c(2, 0.5, 0))
### mar(下余白, 左余白, 上余白, 右余白)、行単位
### mgp(軸～軸ラベル, 軸～目盛ラベル, 軸～軸線)
```

(i) データセットを明示的に読込
（必ず必要という訳ではない）

(ii) 余白と軸を設定

（my_base_graphics3.R：1-18）

標準関数によるグラフ作成：データ

●パッケージ datasets に含まれるデータセット

パッケージ datasets の内容（data() で一覧表示）

統計解析やグラフィックス用のサンプルデータ

分野は様々（植物学、経済学、気象学、機械工学など）

データセットの名称、内容、データ構造の例

iris	アヤメの花の形状データ（data.frame）
mtcars	自動車の性能データ（data.frame）
chickwts	鶏の飼料に関する飼育試験（data.frame）

パッケージ datasets の利用

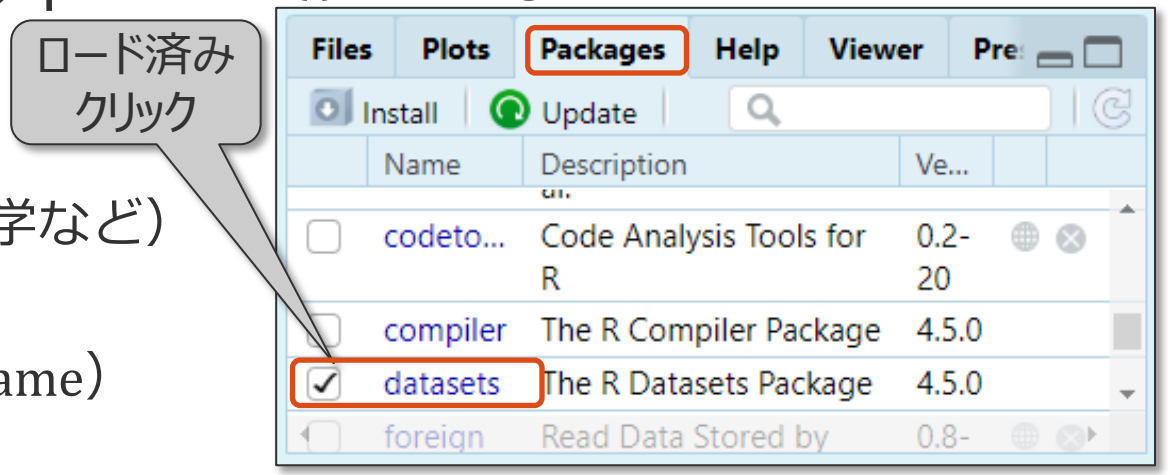
インストール済み、ロード済みである

データセットの名称（iris, mtcars など）を

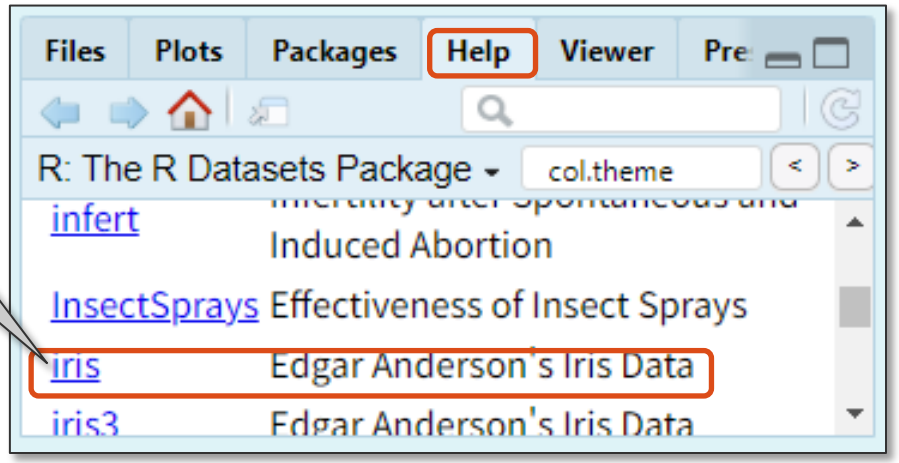
オブジェクト名として使用　　df <- iris、iris[,2]

［Packages］タブ、［Help］タブで確認

(i) ［Packages］タブ



(ii) ［Help］タブ

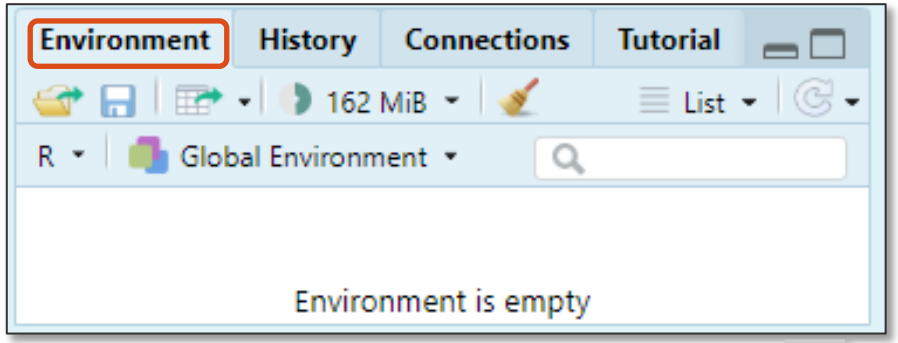


標準関数によるグラフ作成：データ

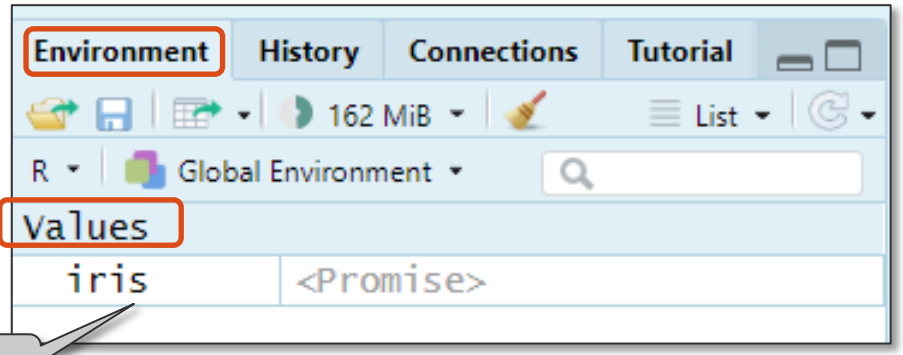
●パッケージ datasets に含まれるデータセット

関数 data で明示的にデータセットを読込

(i) ［Environment］タブ

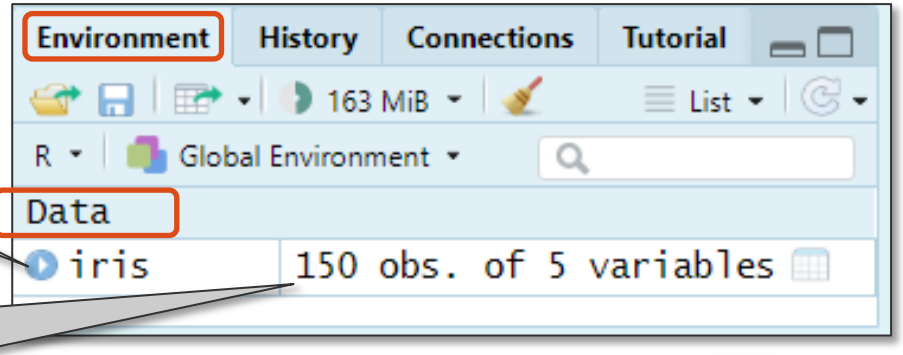


(ii) data(irix) を実行



クリック

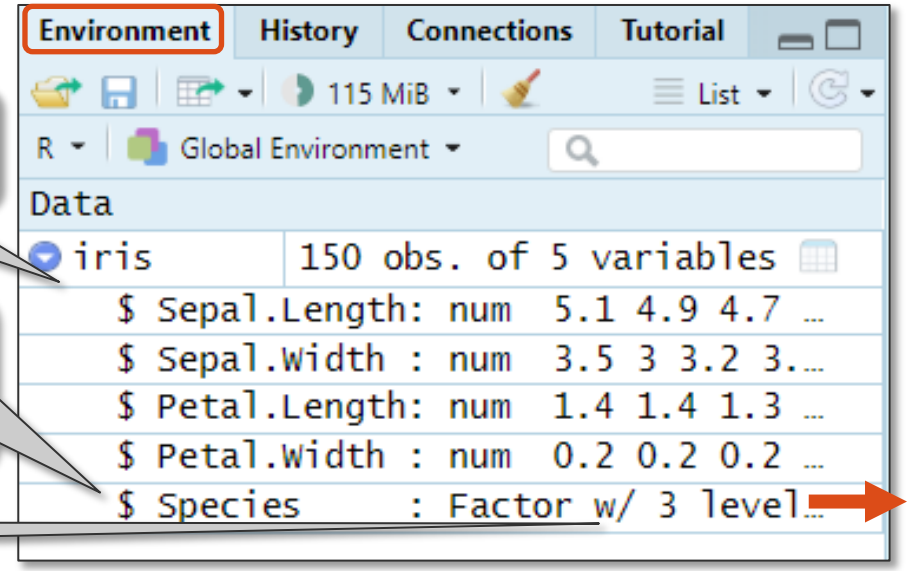
(iii) オブジェクト iris を開ける



クリック

5つの変数と
150 の観測値
から構成

(iv) オブジェクト iris の内容



数値ベクトル 4個
因子ベクトル 1 個

因子ベクトル
3 水準
内部コード 1, 2, 3

with

標準関数によるグラフ作成：データ

●データセット「iris」

アヤメの花の形態データ（Fisher, R. A., 1936）

「がく片」と「花弁」の形状のデータフレーム

長さ（縦）と幅（横）＋品種名 ・・・5 列

３品種につき 50 個体を測定・・・150 行

同一場所で、同じ日に、同一人物によって

同じ測定器で同時に記録（Anderson's Iris data set）

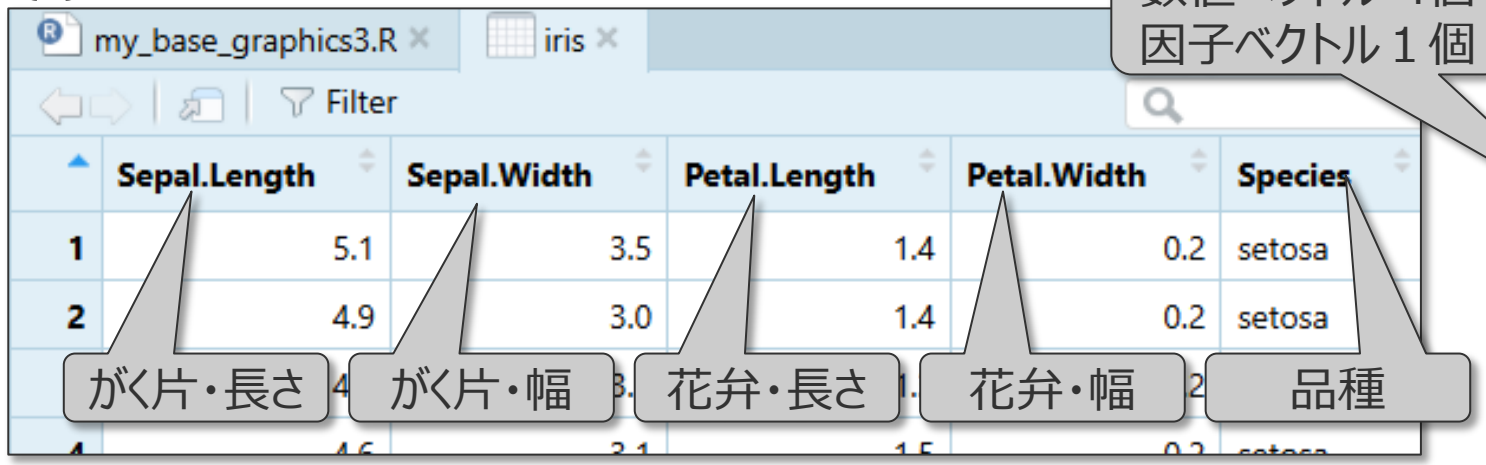
(i)関数 head() でiris の最初の部分を表示

Console | Terminal | Background Jobs

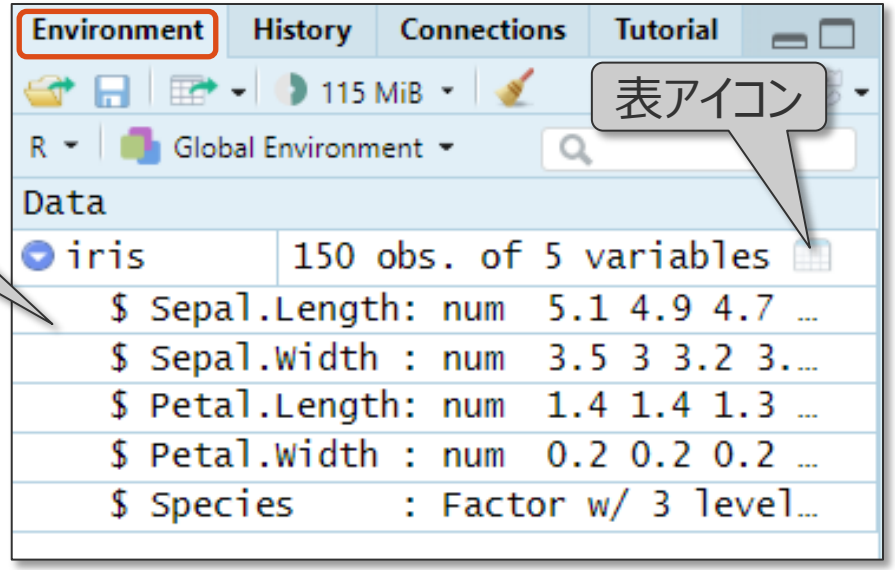
R 4.5.0 · ~/My_Rproj_2025/my_lattice/

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

(ii) 表型式の表示



(iii) オブジェクト iris の内容



標準関数によるグラフ作成：データ

●データセット「mtcars」

自動車の性能データ：米国版 Motor Trend 誌（1974）から抽出（Henderson and Velleman, 1981）

32 台の自動車 (1973 ～ 1974 年モデル) の燃費、自動車の設計および性能に関するデータ

車種の名称（行名）と 性能データ（10 列の数値ベクトル）のデータフレーム

mgp：燃費（miles/gallon）　cyl　：気筒数

disp：排気量(立方インチ)　hp　：総出力

drat：後車軸比　wt　：重量(1000 lbs)

vs　：エンジン(0 = V-shaped, 1 = straight)

am　：トランスミッション（0=automatic、1=manual）

gear：前進ギアの数　carb：キャブレター数

(ii) 表型式の表示

数値

行名
車種の名称

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160.0	110	3.90	2.6
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.8
Datsun 710	22.8	4	108.0	93	3.85	2.3

(i) ［Environment］タブの表示

表アイコン

白抜きの三角

```
Data
mtcars    32 obs. of 11 variables
  $ mpg : num  21 21 22.8 21.4 18.…
  $ cyl : num  6 6 4 6 8 6 8 4 4 6…
  $ disp: num  160 160 108 258 360…
  $ hp  : num  110 110 93 110 175 …
  $ drat: num  3.9 3.9 3.85 3.08 3…
  $ wt  : num  2.62 2.88 2.32 3.21…
  $ qsec: num  16.5 17 18.6 19.4 1…
  $ vs  : num  0 0 1 1 0 1 0 1 1 1…
  $ am  : num  1 1 1 0 0 0 0 0 0 0…
  $ gear: num  4 4 4 3 3 3 3 4 4 4…
  $ carb: num  4 4 1 1 2 1 4 2 2 4…
```

標準関数によるグラフ作成：データ

●データセット「chickwts」

鶏の飼育試験データ

孵化した直後、雛を無作為に6グループに分け、6 種類の異なる飼料を給与
6 週間後の体重（g）を測定、体重増加に対する飼料の種類の効果を解析

飼料の種類と雛の体重のデータフレーム

weight：鶏の体重（g）、数値ベクトル

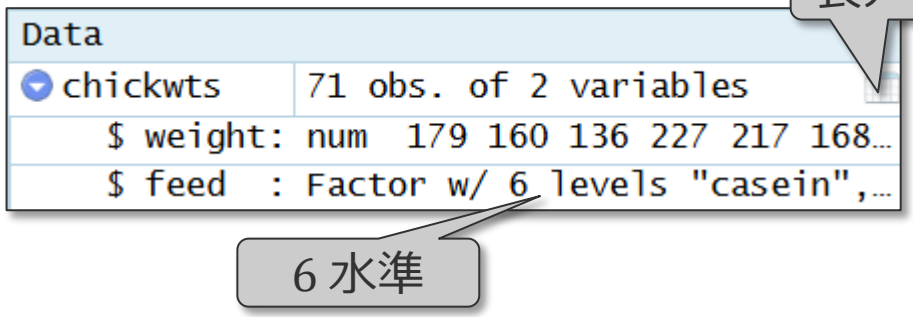
feed：飼料の種類、因子ベクトル（6 水準）

"sunflower" 12 羽,　"soybean" 14 羽,　"meatmeal" 11 羽,
"linseed" 12 羽,　"horsebean" 10 羽,　"casein" 12 羽

(iii)

```
> head(chickwts)
  weight      feed
1    179 horsebean
2    160 horsebean
3    136 horsebean
4    227 horsebean
```

(i) ［Environment］タブの表示



(ii) 表型式の表示

	weight	feed
1	179	horsebean
2	160	horsebean
3	136	horsebean
4	227	horsebean
5	217	horsebean
6	168	horsebean
7	108	horsebean
8	124	horsebean
9	143	horsebean
10	140	horsebean
11	309	linseed
12	229	linseed
13	181	linseed

標準関数によるグラフ作成：データ

●データセット「PlantGrowth」

植物の栽培試験データ

対照区（ctrl）と2つの異なる処理区（trt1, trt2）で植物を栽培

生長量を乾燥重量（g）で測定、各試験区とも繰返し数は 10

試験区と植物の生長量のデータフレーム

weitht：植物の生長量（g）

group：対照区（ctrl）、2つの異なる処理区（trt1, trt2）

各試験区とも 10 区、データフレーム

(ii)

	weight	group
1	4.17	ctrl
2	5.58	ctrl
3	5.18	ctrl
4	6.11	ctrl
5	4.50	ctrl
6	4.61	ctrl
7	5.17	ctrl
8	4.53	ctrl
9	5.33	ctrl
10	5.14	ctrl
11	4.81	trt1
12	4.17	trt1

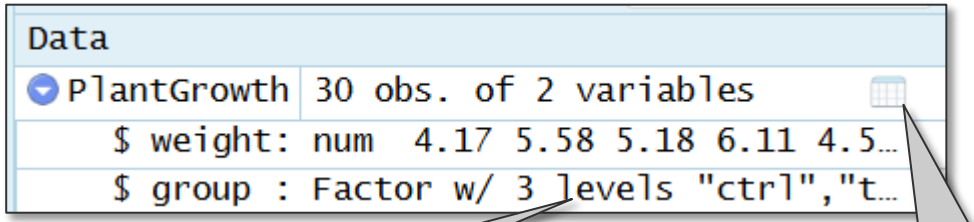
植物の生長量（g）

対照区（crtl）
処理九（trt1, trt2）

(iii)

```
> head(PlantGrowth)
  weight group
1   4.17  ctrl
2   5.58  ctrl
3   5.18  ctrl
4   6.11  ctrl
5   4.50  ctrl
6   4.61  ctrl
```

(i) ［Environment］タブの表示



3 水準

表アイコン

標準関数によるグラフ作成：データ

●データセット「VADeaths」

米バージニア州の1940 年における 1000 人あたりの死亡率（‰）

死亡率は人口 1000 人当たりの年間死亡率

5 段階の年齢層（行）と 4 つり地域・性別（列）で死亡率を計算

年齢層は「50～54歳」、「55～59歳」、「60～64歳」、「65～69歳」、「70～74歳」

地域・性別は「農村部の男性」、「農村部の女性」
「都市部の男性」、「都市部の女性」

５行×４列のマトリックス（行列）

行：年代

列：地域と性別

要素：死亡率（‰）

農村部の男性と女性
都市部の男性と女性

年齢層

1000 人あたりの死亡率 ‰

	Rural Male	Rural Female	Urban Male	Urban Female
50-54	11.7	8.7	15.4	8.4
55-59	18.1	11.7	24.3	13.6
60-64	26.9	20.3	37.0	19.3
65-69	41.0	30.9	54.6	35.1
70-74	66.0	54.3	71.1	50.0

標準関数によるグラフ作成：データ

●データセット「volcano」

火山（マウンガ・ワウ（マウント・イーデン）、ニュージーランド）の標高（m）のデータ

87 行 61 列のマトリックス（行列）

　行は東西の位置、列は南北の位置で 10 m 間隔

南北の位置 10 m 間隔

東西の位置 10 m 間隔

標高（m）

	V1	V2	V3	V4	V5	V6
1	100	100	101	101	101	101
2	101	101	102	102	102	102
3	102	102	103	103	103	103
4	103	103	104	104	104	104
5	104	104	105	105	105	105
6	105	105	105	106	106	106

標準関数によるグラフ作成：データ

●データセット「HairEyeColor」

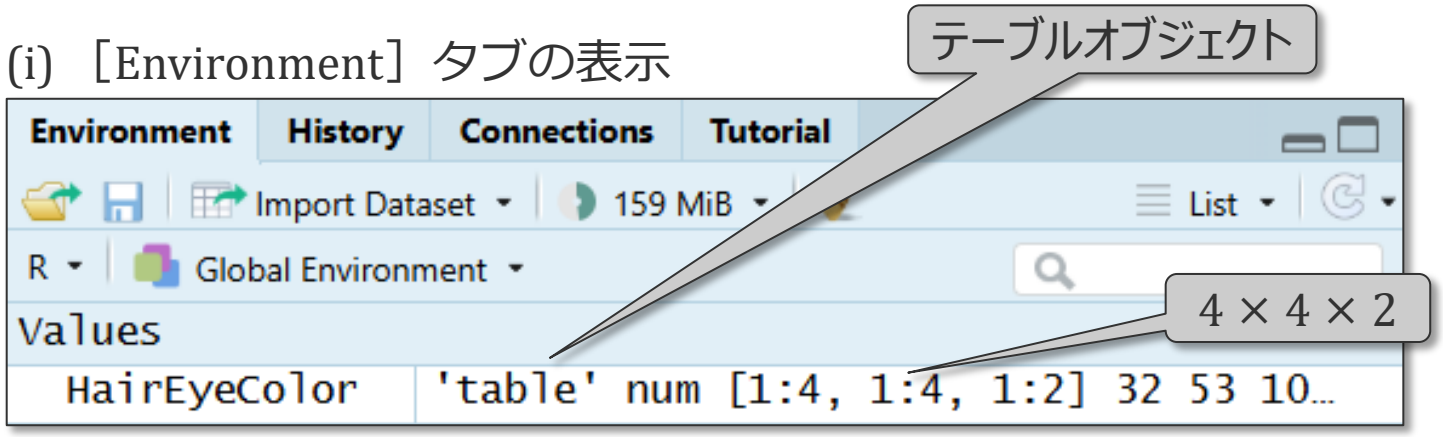
統計学を専攻する学生 592 人の、髪の色、目の色を性別で集計した分割表

テーブルオブジェクト（4×4 分割表が 2 つ）

因子と水準

因子	水準
Hair	Black, Brown, Red, Blond
Eye	Brown, Blue, Hazel, Green
Sex	Male, Female

(i) ［Environment］タブの表示



(ii)

Console に入力

```
> HairEyeColor
, , Sex = Male

       Eye
Hair    Brown Blue Hazel Green
  Black    32   11    10     3
  Brown    53   50    25    15
  Red      10   10     7     7
  Blond     3   30     5     8

, , Sex = Female

       Eye
Hair    Brown Blue Hazel Green
  Black    36    9     5     2
  Brown    66   34    29    14
  Red      16    7     7     7
  Blond     4   64     5     8
```

標準関数によるグラフ作成 (1)

●標準関数：(1) 散布図（Scatter Plot）、plot()、coplot()

(a) iris：アイリスのがく片の幅と長さの層別散布図（3 品種による層別）を plot() で作成
　因子ベクトルの内部コード（1, 2, 3）を条件変数として層別

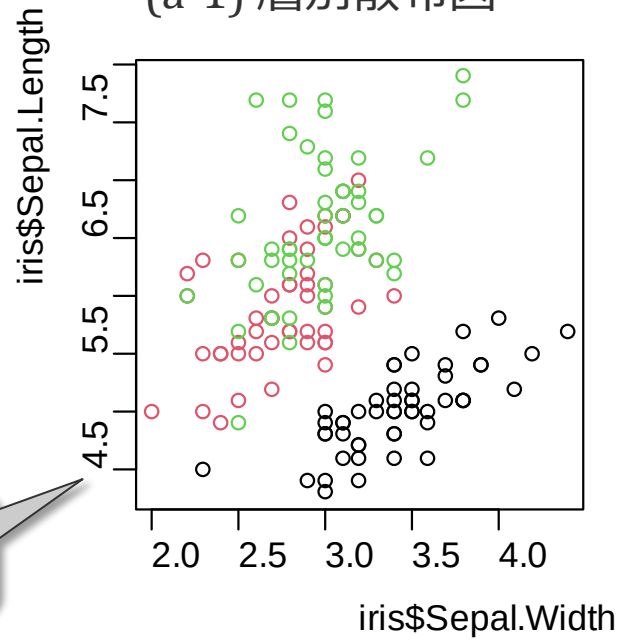
```
21  # (1) 散布図(Scatter Plot)、plot()  ------------
22
23  ## (a) iris：がく片の長さと幅の層別散布図
24  ### (a-1) シンプルなコード
25  plot(iris$Sepal.Width, iris$Sepal.Length,
26       col = iris$Species)
27
28  ### (a-2) 外観のカスタマイズ
29  plot(
30    Sepal.Length ~ Sepal.Width, data = iris,
31    col = iris$Species,     # 内部コード 1,2,3
32    xlab = "がく片の幅",    # x軸の軸ラベル
33    ylab = "がく片の長さ", # y軸の軸ラベル
34    las = 1)          # 目盛ラベルを軸と水平に表示
35
36  ### (a-3) データフレームから抽出
37  df <- iris[, c("Sepal.Width","Sepal.Length")]
38  plot(df, col = iris$Species)
```

（my_base_graphics3.R：21-38）

Species は因子ベクトル
内部コード(1,2,3)が col に渡される
別の識別方法
　pch = iris$Species

既定値は las=0
las = 1 により軸に水平

(a-1) 層別散布図



標準関数によるグラフ作成 (1)

●標準関数：(1) 散布図（Scatter Plot）、plot()、coplot()

(b) iris：アイリスのがく片の幅と長さの層別散布図（3 品種による層別）を plot() で作成

因子ベクトルの内部コード（1, 2, 3）を条件変数として層別

関数 legend() を用いて凡例を追加

```
40  ## (b) iris：がく片の長さと幅の層別散布図
41  ##      凡例を追加
42  plot(iris$Sepal.Width, iris$Sepal.Length,
43       col = iris$Species,
44       pch = 21,              # マーカー指定
45       xlim = c(2, 5.5))      # 凡例のスペース確保
46
47  legend("topright",          # 凡例を表示する位置
48         pch = 21,            # マーカー指定
49         cex = 0.7,           # 凡例の文字サイズ
50         col = unique(as.numeric(iris$Species)),
51         legend = levels(iris$Species),
52         title = "Species")
```

（my_base_graphics3.R：40-52）

内部コード（1, 2, 3）が渡される

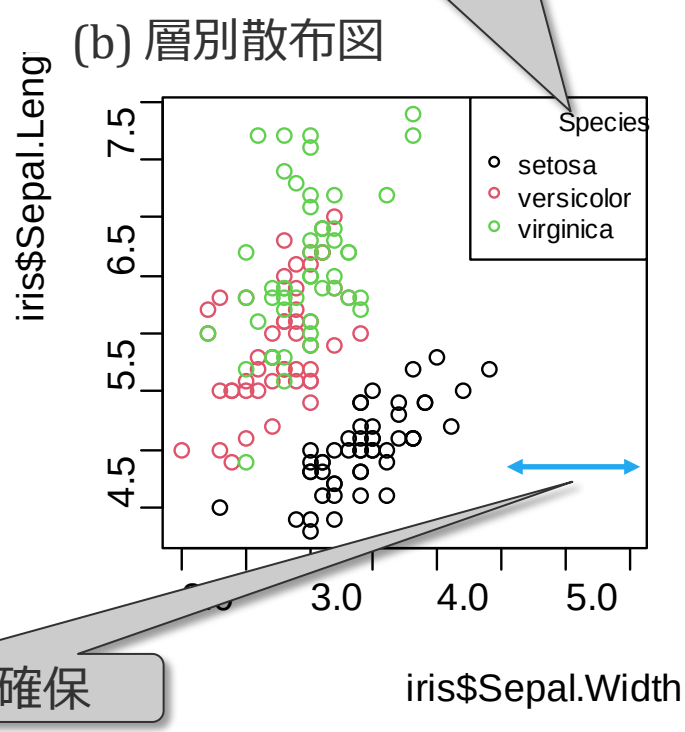
凡例のスペースを確保

内部コード（1, 2, 3）が渡される

品種名が渡される

凡例のスペースを確保

凡例の位置　topright



標準関数によるグラフ作成 (1)

●標準関数：(1) 散布図（Scatter Plot）、plot()、coplot()

(c) iris：アイリスのがく片の幅と長さの層別散布図（3 品種による層別）を plot() で作成

因子ベクトルの内部コード（1, 2, 3）を条件変数として層別

関数 dataEllipse() を用いて 50% 確率楕円を追加（パッケージ car をインストールしておく）

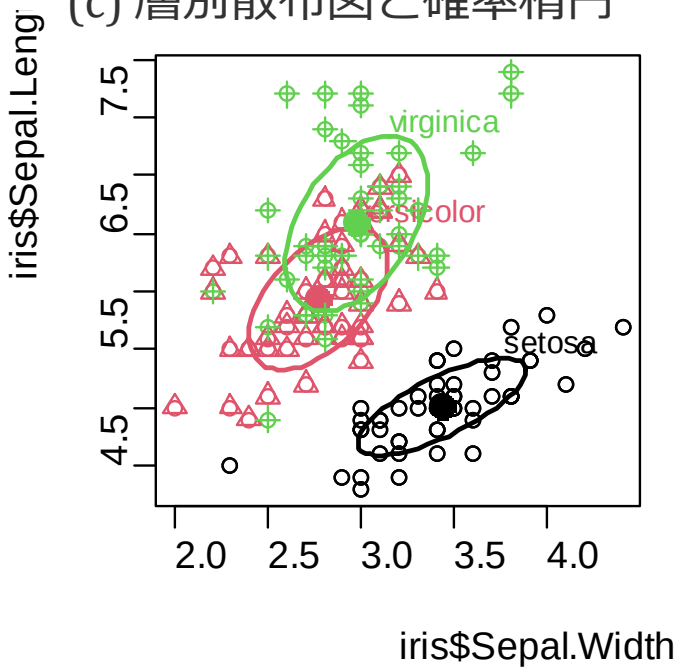
```
54  ## (c) iris：がく片の長さと幅の層別散布図
55  ##     50% 確率楕円を追加
56  library(car)                # 関数 dataEllipse
57
58  plot(iris$Sepal.Width, iris$Sepal.Length,
59       col = iris$Species,  # 因子の内部コード1,2,3
60       pch = 21)
61
62  dataEllipse(
63    iris$Sepal.Width, iris$Sepal.Length,
64    groups = iris$Species,  # グループ化する変数
65    level = 0.50,           # 確率水準 (50%)
66    col = unique(as.numeric(iris$Species)),
67    label.cex = 0.8,
68    add = TRUE)             # 既存のグラフに上書き
```

パッケージ car をロード

内部コード（1, 2, 3）が渡される

（my_base_graphics3.R：54-68）

(c) 層別散布図と確率楕円



標準関数によるグラフ作成 (1)

●標準関数：(1) 散布図（Scatter Plot）、plot()、coplot()

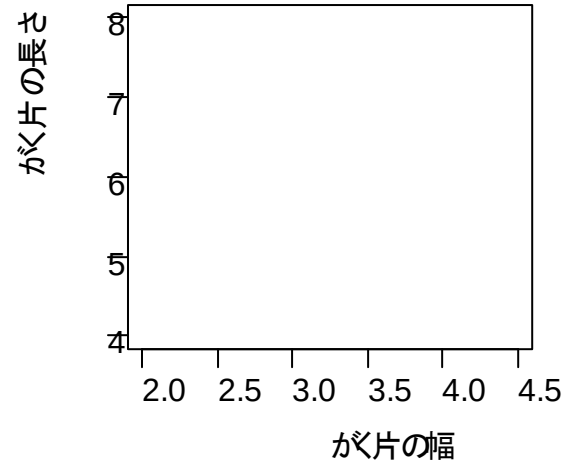
(d) iris：アイリスのがく片の幅と長さの層別散布図

関数 subset() と points() を使って抽出、points() で描画

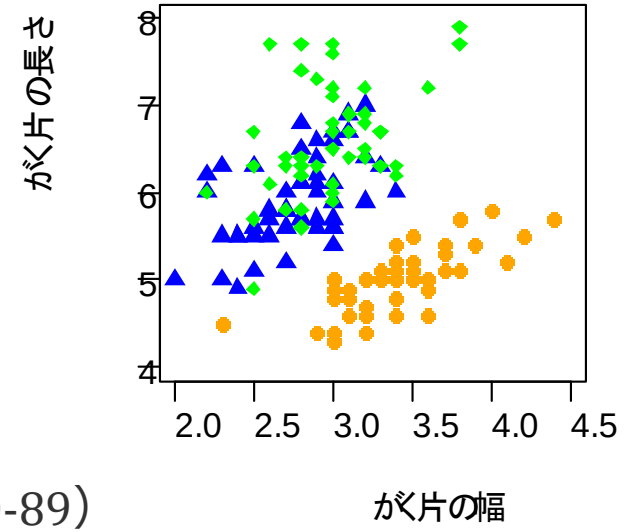
```
70  ## (d) iris：がく片の長さと幅の層別散布図
71  ##     plot()で枠を表示、points()でプロット
72  plot(x = NA,                  # 仮のデータ
73       xlim = c(2, 4.5),        # x軸の範囲指定
74       ylim = c(4, 8),          # y軸の範囲指定
75       xlab = "がく片の幅",
76       ylab = "がく片の長さ",
77       las = 1)                 # 目盛ラベルの表示方向
78
79  df <- subset(iris, Species == "setosa")
80  points(df$Sepal.Width, df$Sepal.Length,
81         col = "orange", pch = 16)
82
83  df <- subset(iris, Species == "versicolor")
84  points(df$Sepal.Width, df$Sepal.Length,
85         col = "blue", pch = 17)
86
87  df <- subset(iris, Species == "virginica")
88  points(df$Sepal.Width, df$Sepal.Length,
89         col = "green", pch = 18)
```

x = NA でプロットなし
散布図の枠を作成

for 文の利用も可



(d) 層別散布図



（my_base_graphics3.R：70-89）

標準関数によるグラフ作成 (1)

●標準関数：(1) 散布図（Scatter Plot）、plot()、coplot()

(e) iris：アイリスのがく片の幅と長さの層別散布図（3 品種による層別）

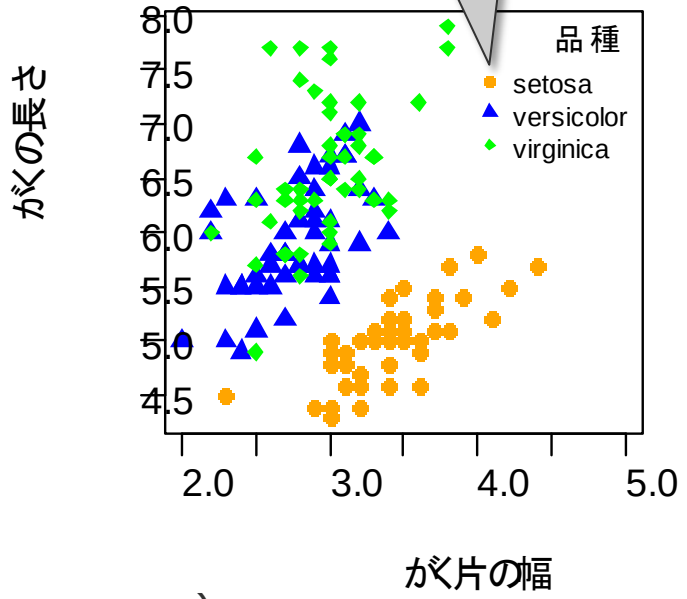
3 品種を区別する記号の形と色の指定

```
91  ## (e) iris：3品種のがく片の長さと幅の層別散布図
92  pch_num  <- c(16, 17, 18)       # 品種のマーカーの形
93  col_name <- c("orange", "blue", "green") # 品種の色
94
95  plot(
96    Sepal.Length ~ Sepal.Width,     # y軸 ~ x軸
97    data = iris,                    # データフレーム
98    pch = pch_num[iris$Species],    # 品種ごとの形状
99    col = col_name[iris$Species],   # 品種ごとの色
100   xlim = c(2, 5),                 # x軸の範囲指定
101   las = 1,                        # 目盛ラベルの方向
102   xlab = "がく片の幅", ylab = "がくの長さ")
103
104 legend(
105   x = "topright",           # 凡例の位置
106   bty = "n",                # 凡例の枠線非表示
107   cex = 0.7,                # 凡例の文字サイズ
108   title.cex = 0.8,          # タイトルの文字サイズ
109   legend = levels(iris$Species), # 凡例のテキスト
110   pch = pch_num,            # マーカーの形状
111   col = col_name,           # マーカーの色
112   title = "品種")           # 凡例のタイトル
```

3 品種ごとに記号の形と色を定義

3 品種ごとの記号の形と色

(e) 層別散布図



（my_base_graphics3.R：91-112）

標準関数によるグラフ作成 (1)

●標準関数：(1) 散布図（Scatter Plot）、plot()、coplot()

(f) iris：アイリスのがく片の幅と長さの層別散布図を coplot() で作成
3 品種ごとに別々のグラフ（パネル）で散布図を得る
（マルチパネル、ファセット）

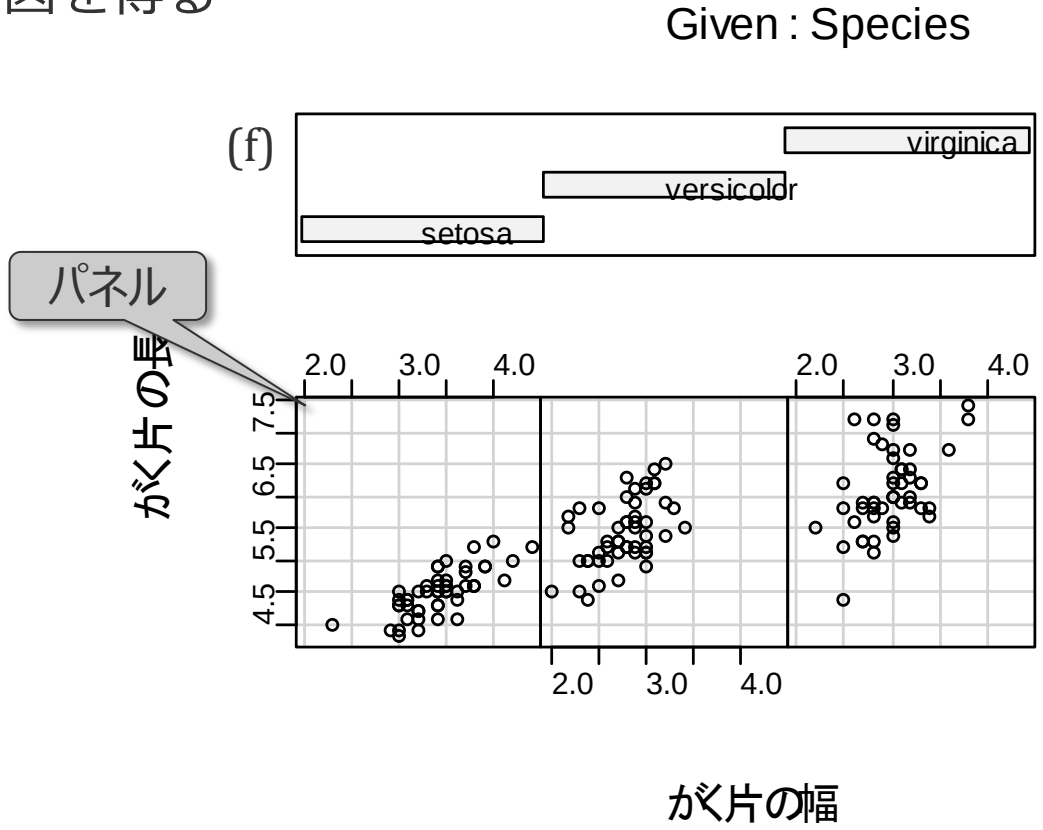
関数 coplot() に渡す数式

coplot(y ~ x | g)

パッケージ latticeの関数の利用を検討

```
114  ## (f) iris：3品種のがく片の長さと幅の層別散布図
115  ##     coplot()によるマルチパネル（ファセット）
116  ##     代わりにパッケージ lattice の関数を利用
117  coplot(
118    Sepal.Length ~ Sepal.Width | Species,
119    data = iris,
120    type = "p",              # プロットする形状
121    rows = 1,                # パネルを1列に並べる
122    xlab = "がく片の幅",
123    ylab = "がく片の長さ")
```

（my_base_graphics3.R：114-123）



標準関数によるグラフ作成 (1)

●標準関数：(1) 散布図（Scatter Plot）

関数のヘルプの利用方法

(i) 関数にカーソルを合わせて F1 キーを押す

(ii) Console 上で ?coplot を実行

(iii) Console 上で help(coplot) を実行

ヘルプの内容

Description, Usage, Arguments, Details,

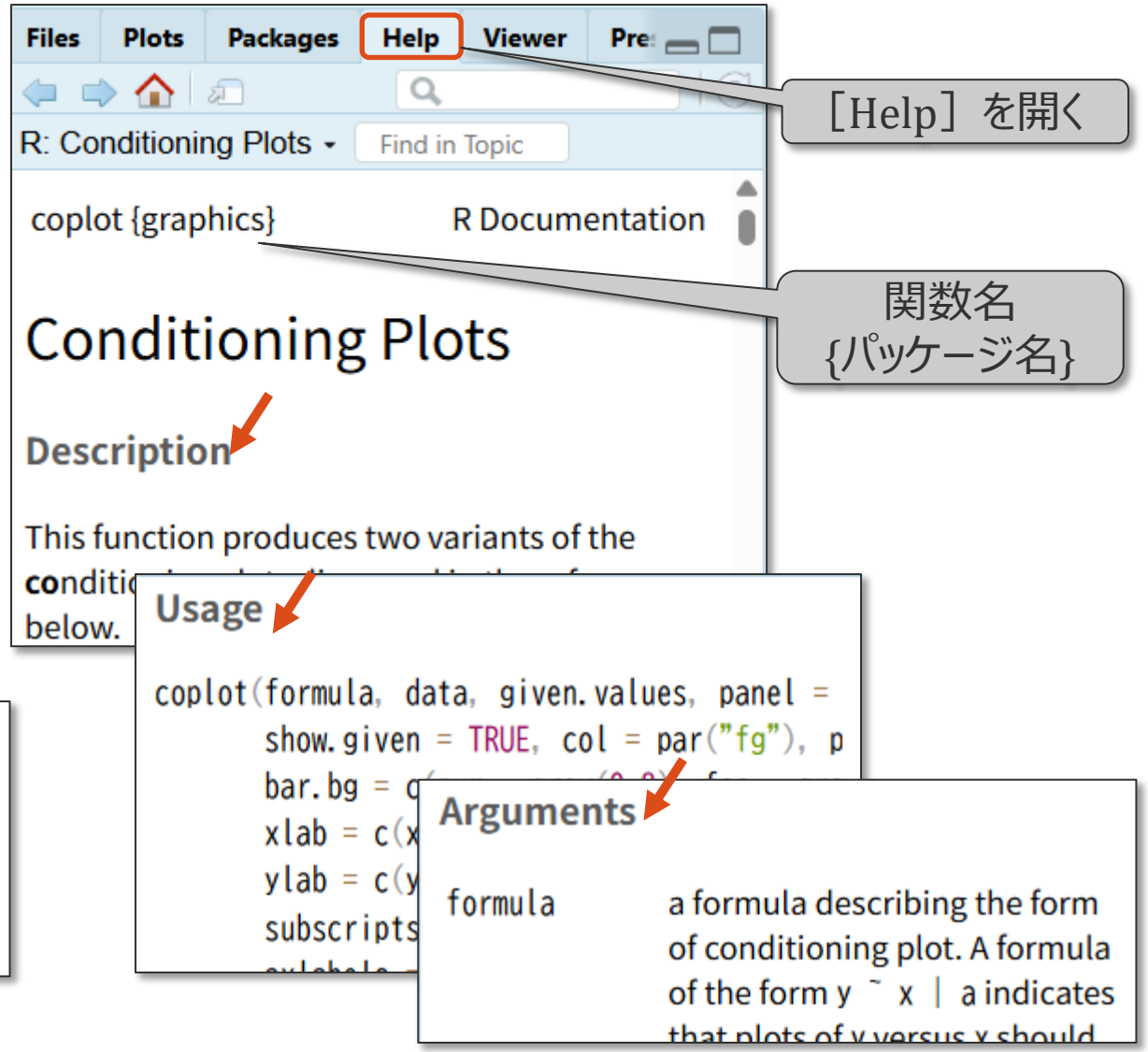
Value, Rferences, See Also, Example

AI の利用

```
109  ## (f) iris：3品種のがく片の長さ
110  ##     マルチパネルのデータセット
111  coplot(
112    Sepal.Length ~ Sepal.Width | Species, data = iris,
113    type = "p",
114    xlab = "がく片の幅",
115    ylab = "がく片の長さ")
```

カーソルを置いて
F1 キーを押す

（my_base_graphics3.R：109-115）



標準関数によるグラフ作成 (2)

●標準関数：(2) 散布図行列（Scatterplot Matrix）、pairs()

(a) mtcars：32 車種の燃費、馬力、重量の散布図行列を作成

列名：mpg（燃費）、hp（馬力）、wt（重量）

引数 labels：対角線上に表示する列名

引数 cex.labels：labels の文字サイズを指定

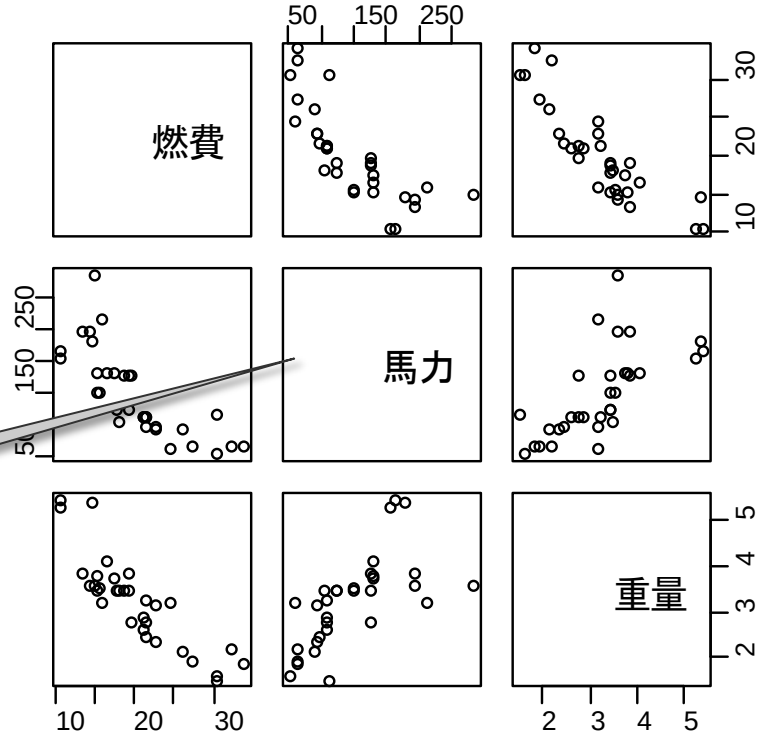
	mpg	cyl	disp	hp	drat	wt	qs
Mazda RX4	21.0	6	160.0	110	3.90	2.620	
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	

```
126  # (2) 散布図行列(Scatterplot Matrix)、pairs() --
127
128  ## (a) mtcars：燃費、馬力、重量の散布図行列
129  pairs(mtcars[, c("mpg", "hp", "wt")],
130        labels = c("燃費","馬力","重量"),
131        cex.labels = 1.5)
```

ラベル

(my_base_graphics3.R：126-131)

(a) 散布図行列



標準関数によるグラフ作成 (2)

●標準関数：(2) 散布図行列

(b) mtcars：燃費、馬力、重量の散布図行列
パネル関数によるヒストグラムと相関係数の追加

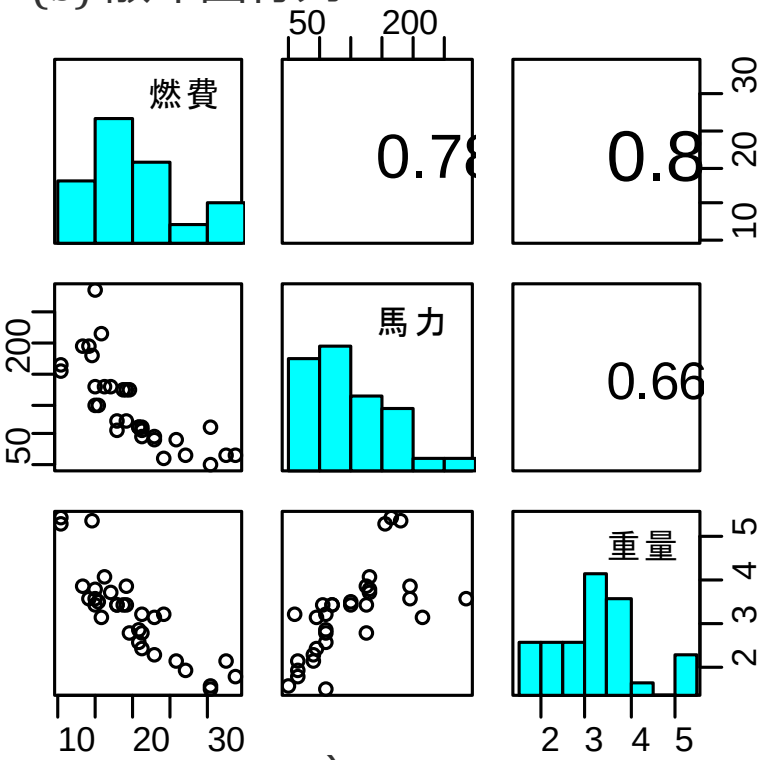
```
133 ## (b) mtcars：燃費、馬力、重量の散布図行列
134 ## 相関係数、ヒストグラムを表示(パネル関数の利用)
135 panel_cor <- function(
136     x, y, digits = 2, prefix = "", cex.cor, ...) {
137   par(usr = c(0, 1, 0, 1))
138   r <- abs(cor(x, y))
139   txt <- format(c(r,0.123456789), digits=digits)[1]
140   txt <- paste0(prefix, txt)
141   if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
142   text(0.5, 0.5, txt, cex = cex.cor * r)
143 }
144 panel_hist <- function(x, ...) {
145   usr <- par("usr")
146   par(usr = c(usr[1:2], 0, 1.5) )
147   hist_out <- hist(x, plot = FALSE)
148   breaks <- hist_out$breaks; nB <- length(breaks)
149   y <- hist_out$counts; y <- y/max(y)
150   rect(xleft = breaks[-nB], ybottom = 0,
151        xright = breaks[-1], ytop = y,
152        col = "cyan", ...)
153 }
```

相関係数を表示するパネル関数

ヒストグラムを表示するパネル関数

```
155 pairs(mtcars[, c("mpg", "hp", "wt")],
156       diag.panel = panel_hist,
157       upper.panel = panel_cor,
158       labels = c("燃費","馬力","重量"))
```

(b) 散布図行列



(my_base_graphics3.R：133-158)

標準関数によるグラフ作成 (3)

●標準関数：(3) 線グラフ（Line Graph）、plot()、points()、matplot()

(a) mtcars：32 車種のシリンダー数ごとの燃費（平均値）の折れ線グラフを plot() で作成

関数 aggregate() を用いて、シリンダー数（cyl）ごとの燃費（mpg）の平均値を算出

```
161  # (3) 線グラフ(Line Graph)
162  #     plot()、points()、matplot() --------------
163
164  ## (a) mtcars：mpg(燃費)とcyl(シリンダー数)の関係
165  ##     cylごとにmpgの平均を計算してプロット
166  agg_out <- aggregate(
167    mpg ~ cyl, data = mtcars,
168    FUN = mean)
169
170  print(agg_out)
171
172  plot(
173    mpg ~ cyl, data = agg_out, type = "l",
174    xlab = "シリンダー数",
175    ylab = "燃費",
176    las = 1)          # 目盛ラベルの表示の方向
```

type = "l"
線グラフを指定

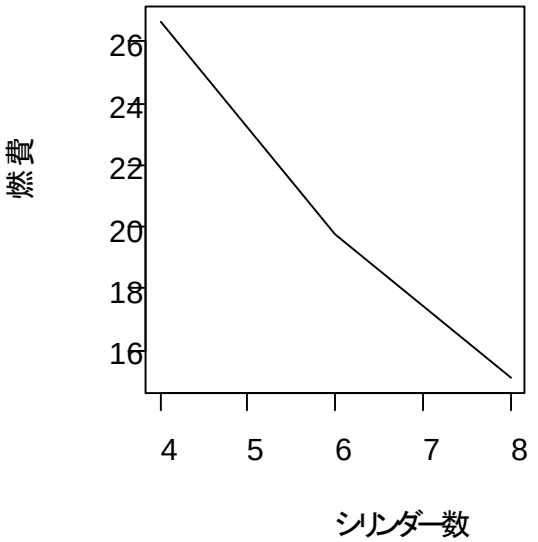
（my_base_graphics3.R：161-176）

データフレーム

agg_out

```
> print(agg_out)
  cyl      mpg
1   4 26.66364
2   6 19.74286
3   8 15.10000
```

(a) 折れ線グラフ



標準関数によるグラフ作成 (3)

●標準関数：(3) 線グラフ（Line Graph）、plot()、points()、matplot()

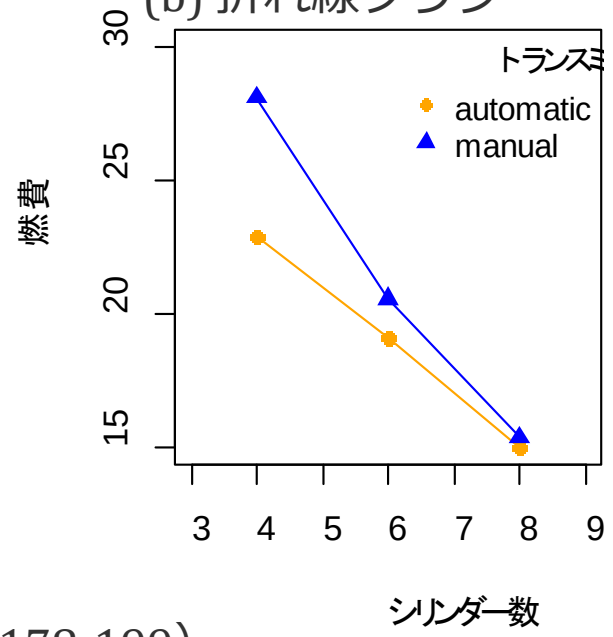
(b) mtcars：32 車種のシリンダー数（cyl）と燃費（mpg）の平均値との関係を、トランスミッション（am）ごとの折れ線グラフで可視化

```
178 ## (b) mtcars：cyl(シリンダー数)とmpg(燃費)の関係
179 ##      をam(トランスミッション)で層別して表示
180 ##      cylとamごとにmpgの平均を計算
181 agg_out <- aggregate(mpg ~ am + cyl,
182                       data = mtcars, FUN = mean)
183 print(agg_out)
184
185 plot(mpg ~ cyl, data = agg_out,
186      xlim = c(3, 9), ylim = c(15, 30), type = "o",
187      col = "orange", pch = 16,
188      subset = agg_out$am == 0,    # automaticで抽出
189      xlab = "シリンダー数", ylab = "燃費")
190
191 points(mpg ~ cyl, data = agg_out, type = "o",
192      col = "blue", pch = 17,
193      subset = agg_out$am ==1)     # manualで抽出
194
195 legend("topright",
196        legend = c("automatic", "manual"),
197        title = "トランスミッション",
198        col = c("orange", "blue"), pch = c(16, 17),
199        cex = 0.9, bty = "n", title.cex = 0.9)
```

データフレーム agg_out

```
> print(agg_out)
  am cyl      mpg
1  0   4 22.90000
2  1   4 28.07500
3  0   6 19.12500
4  1   6 20.56667
5  0   8 15.05000
6  1   8 15.40000
```

(b) 折れ線グラフ



(my_base_graphics3.R：178-199)

標準関数によるグラフ作成 (3)

●標準関数：(3) 線グラフ（Line Graph）、plot()、points()、matplot()

(c) ランダムウォーク（random walk、時系列データ）の線グラフ

matplot() で複数系列を一つのグラフ領域に重ねて表示

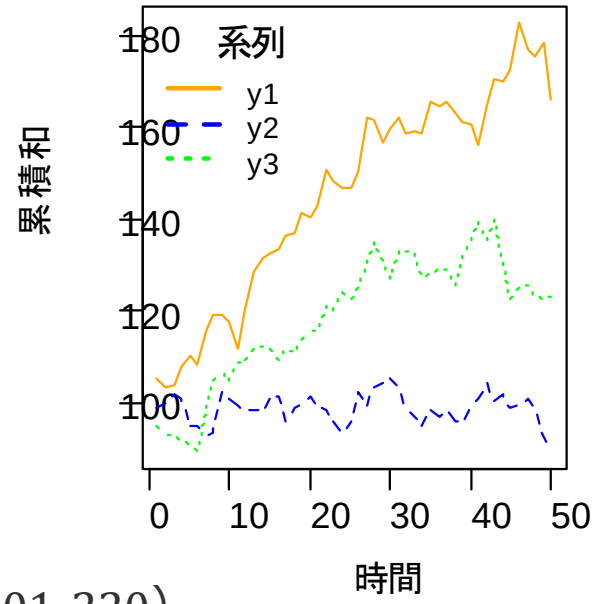
```
201  ## (c) ランダムウォーク(時系列データ)の線グラフ
202  ##      複数系列を１つのグラフに重ねて表示
203  t <- 1:50
204  y1 <- 100 + cumsum(rnorm(50, 0, 5))
205  y2 <- 100 + cumsum(rnorm(50, 0, 3))
206  y3 <- 100 + cumsum(rnorm(50, 0, 4))
207  (mx <- cbind(y1, y2, y3))
208
209  matplot(x = t, y = mx,
210          type = "l",  # "p" の指定により散布図
211          col = c("orange", "blue", "green"),
212          xlab = "時間", ylab = "累積和",
213          las = 1)
214
215  legend("topleft",
216         legend = c("y1", "y2", "y3"),
217         col = c("orange", "blue", "green"),
218         title = "系列",
219         lty = 1:3, lwd = 2, bty = "n",
220         title.cex = 1.1, cex = 0.8)
```

平均 0、標準偏差 5 の正規分布から50 個の乱数を生成、その累積和

( ) をつけると pirnt( ) と同じ機能

```
> (mx <- cbind(y1, y2, y3))
           y1       y2        y3
 [1,] 102.78061  98.29982 100.57365
 [2,] 108.29111  93.94338 102.26677
 [3,] 107.49168  95.13598 101.29266
 [4,] 107.34244  91.24018 106.19612
 [5,] 106.97271  91.28932 104.52213
```

(c)



（my_base_graphics3.R：201-220）

標準関数によるグラフ作成 (4)

●標準関数：(4) 散布図の特殊な可視化、sunflowerplot()、symbols()

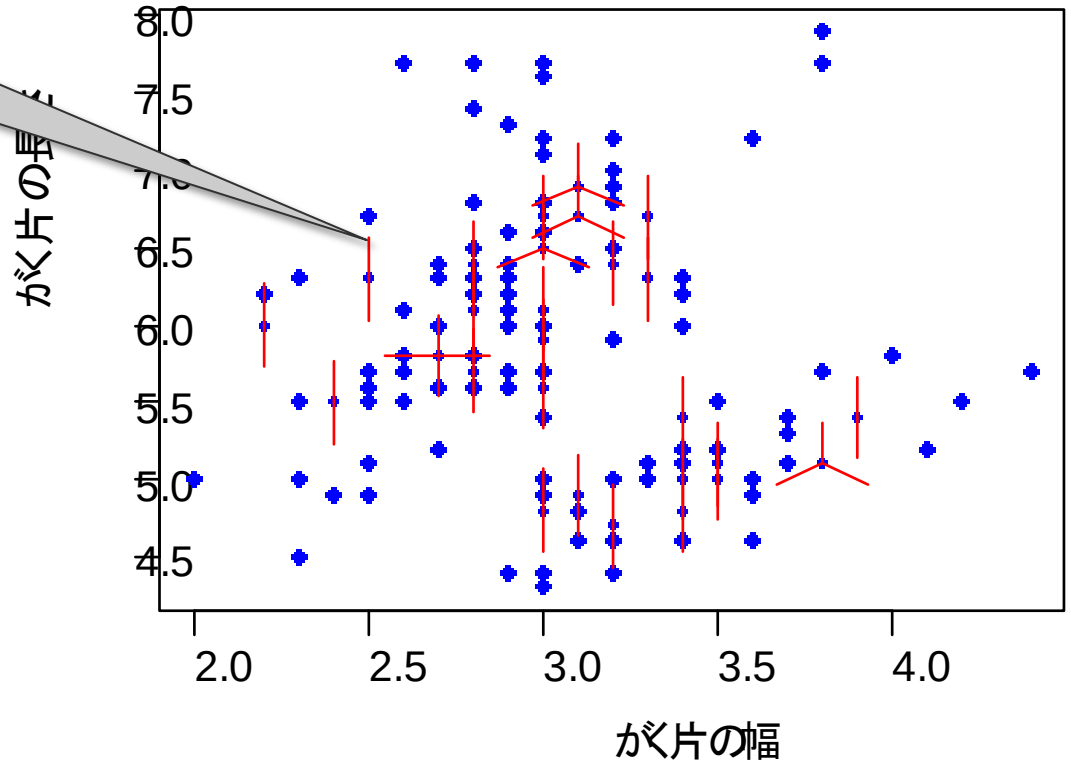
(a) iris：関数 sunflowerplot() を使い、アヤメのがく片の幅と長さの散布図として、
ひまわりプロット(Sunflower plot) を作成

同じ位置に複数の点がプロット
されていることを、赤い線分で表示
１ つの重複につき１本の線分

```
223  # (4) 散布図の特殊な可視化
224  #     sunflowerplot() 、symbols() --------------
225
226  ## (a) iris：ひまわりプロット(Sunflower plot)
227  ##     がく片の幅と長さのひまわりプロットを作成
228  sunflowerplot(
229    iris$Sepal.Width, iris$Sepal.Length,
230    xlab = "がく片の幅",
231    ylab = "がく片の長さ",
232    main = "Iris データの重複を表示",
233    col = "blue",       # 青い点をプロット
234    seg.col = "red",    # 重複に対して赤い線分で表示
235    las = 1)
```

(my_base_graphics3.R：223-235)

(a) ひまわりプロット



標準関数によるグラフ作成 (4)

●標準関数：(4)散布図の特殊な可視化、sunflowerplot()、symbols()

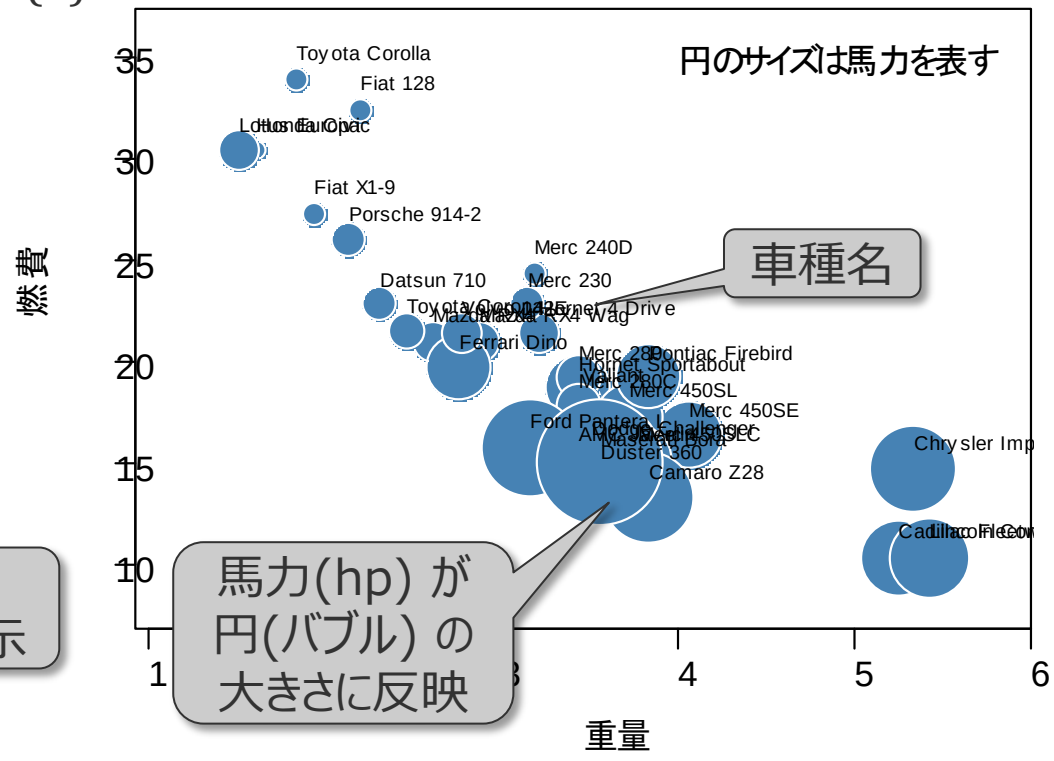
(b) mtcars：関数 symols() を使い、重量（wt）と燃費（mpg）の散布図に
馬力（hp）に対応した円を表示（バブルチャート、bubble chart)

```
237 ## (b) mtcars：シンボルプロット(Symbol plot)
238 ## wt(重量)とmpg(燃費)の散布図にhp(馬力)を表示
239 symbols(
240   mtcars$wt, mtcars$mpg,
241   circles = mtcars$hp,    # hpを円の大きさで表現
242   inches = 0.3,           # 円の最大寸法を指定
243   fg = "white", bg = "steelblue",
244   xlab = "重量", ylab = "燃費",
245   las = 1,
246   main = "車の重量・燃費・馬力の関係")
247
248 text(x = mtcars$wt, y = mtcars$mpg, pos = 3,
249      rownames(mtcars),
250      cex = 0.6)           # 車種名を表示
251
252 text(x = 4, y = 35,
253      labels = "円のサイズは馬力を表す")
```

各バブルに
車種名を表示

(my_base_graphics3.R：237-253)

(b) バブルチャート



標準関数によるグラフ作成 (5)

●標準関数：(5) 箱ひげ図（Box Plot）、boxplot()、plot()

連続変数の分布を箱ひげ図（ Box Plot , Box-and-Whisker Plot）で可視化

使い方　plot(y ~ grp)　・・・　y：数値ベクトル、grp：グループを表す因子ベクトル

boxplot(y ~ grp) ・・　y：数値ベクトル、grp：グループを表す因子ベクトル

boxplot(x)　　・・・　x：数値ベクトル（１変数の箱ひげ図）

boxplot(list(GroupA = x1, GroupB = x2, GroupC = x3)) ・・・x1, x2, x3：数値ベクトル

boxplot(mx) ・・・・　mx：数値マトリックス（列ごとに箱とヒゲを作成）

関数 boxplot() の主な引数

horizontal：TRUE/FALSE（箱が水平方向／垂直方向（既定値））

boxwex：すべてのボックスに適用されるスケール係数、箱の相対的な幅、規定値は 0.8

notch：TRUE/FALSE（箱にノッチ(くびれ)を付加／付加なし(既定値)）

range：ヒゲの長さと外れ値の判定に用いる係数(既定値 1.5)、names：箱の下に表示する名前

outline：TRUE/FALSE（外れ値表示(既定値)／非表示）、col, border：箱の背景色、箱枠の色

標準関数によるグラフ作成 (5)

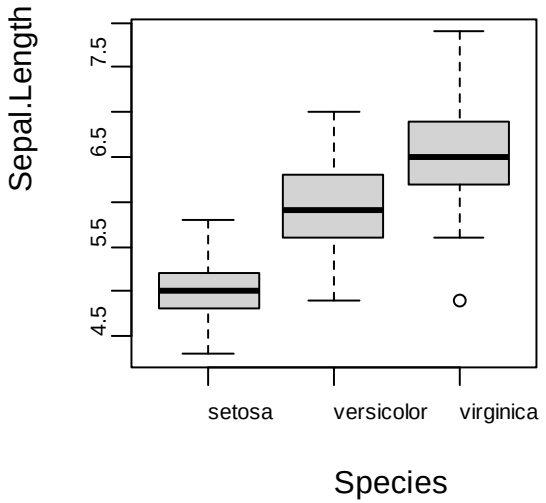
●標準関数：(5) 箱ひげ図（Box Plot）、boxplot()、plot()

(a) iris：がく片の長さの分布を表す箱ひげ図

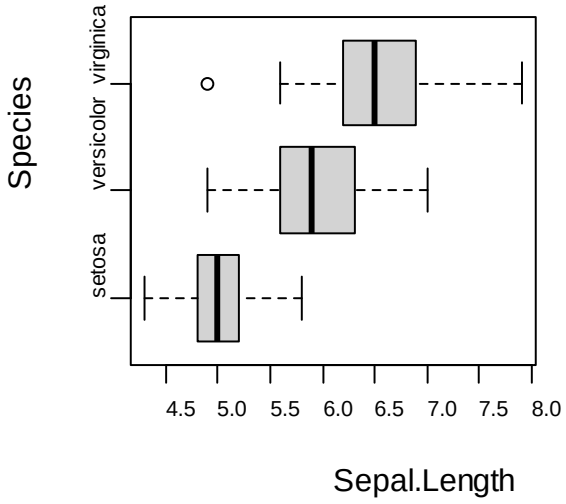
```
256 # (5) 箱ひげ図(Box Plot)、boxplot()、plot() ----
257
258 ## (a) iris：品種別のがく片の長さの分布
259 ### (a-1) plot() 、縦向き、品種別
260 plot(Sepal.Length ~ Species, data = iris,
261      cex.axis = 0.7)
262
263 ### (a-2) boxplot()、縦向き、品種別
264 boxplot(Sepal.Length ~ Species, data = iris,
265         cex.axis = 0.7)
266
267 ### (a-3) boxplot()、横向き、品種別
268 boxplot(Sepal.Length ~ Species, data = iris,
269         horizontal = TRUE, cex.axis = 0.7)
270
271 ### (a-4) boxplot()、縦向き、３品種込み
272 boxplot(iris$Sepal.Length)
273
274 ### (a-5) boxplot()、横向き、３品種込み
275 boxplot(iris$Sepal.Length, horizontal = TRUE)
```

（my_base_graphics3.R：256-275）

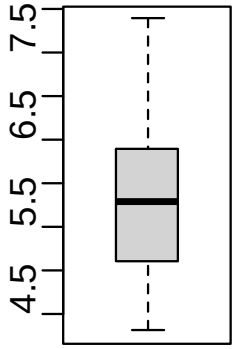
(a-1) (a-2) 箱ひげ図



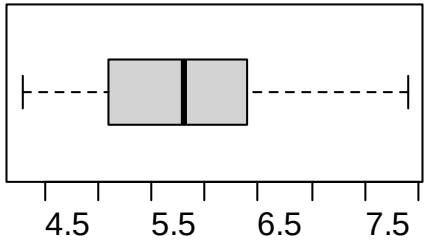
(a-3) 箱ひげ図



(a-4) 箱ひげ図



(a-5) 箱ひげ図



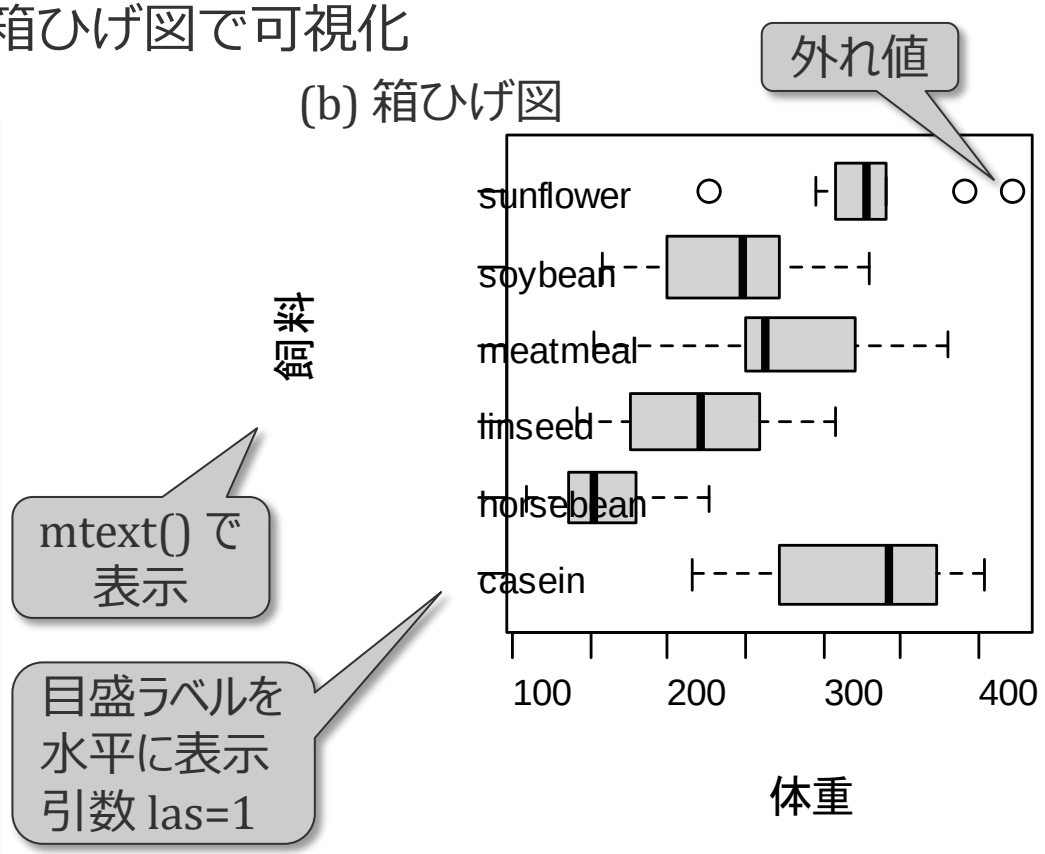
標準関数によるグラフ作成 (5)

●標準関数：(5) 箱ひげ図（Box Plot）、boxplot()、plot()

(b) chickwts：飼料の種類別に鶏の体重の分布を箱ひげ図で可視化

```r
## (b) chickwts：飼料の種類が鶏の体重に及ぼす効果
current_par <-
  par(no.readonly = TRUE)    # パラメータの保存
par(mar = c(4, 6, 1, 1))     # 余白の設定

boxplot(
  weight ~ feed, data = chickwts,
  las = 1,
  cex.axis = 0.8,        # 目盛ラベルの文字サイズ
  cex = 1.2,             # 外れ値のマーカーサイズ
  col = "lightgray",     # 箱の背景色
  border = "black",      # 箱枠の色
  coef = 1.5,            # 外れ値を検出する係数
  varwidth = TRUE,       # 箱の幅が n の平方根に比例
  notch = FALSE,         # ノッチの追加なし
  horizontal = TRUE,     # 横向きのグラフ
  xlab = "体重", ylab = "")

mtext("飼料", side = 2, line = 3.5) # y軸の軸ラベル

par(current_par)       #パラメータの復元
```

(b) 箱ひげ図



mtext() で表示

目盛ラベルを水平に表示 引数 las=1

（my_base_graphics3.R：277-297）

標準関数によるグラフ作成 (6)

●標準関数：(6) １次元散布図（Strip Chart）、stripchart()

連続変数の個々の値を点でプロットして分布を可視化（サイズが小さいサンプルに適す）

使い方　stripchart(y ~ g) ・・・y ：数値ベクトル、g：因子ベクトル

stripchart(x)・・・・・x：数値ベクトル（単一の１次元散布図）

stripchart(list(GroupA=x1, GroupB=x2, GropuC=x3) ・・・x1, x2, x3 ：数値ベクトル

stripchart(mx) ・・・・mx：数値マトリックス（列を１グループとして描画）

関数 stripchart() の主な引数

verctical：TRUE/FALSE（垂直方向にプロット／水平方向(規定値)）・・・boxplot() と逆

method："overplot"（重ねて表示、規定値）、"jitter"（散らす）、"stack"（並べる）

jitter：method="jitter" の場合、ばらつかせる量を指定、規定値は 0.1

group.names：因子ベクトルのグループ名

xlab, ylab：横軸（x軸）の軸ラベル、縦軸（y 軸）の軸ラベル

dlab, glab：数値データが表示されている軸のラベル、グループが表示されている軸のラベル

x 軸、y 軸に固定されない

標準関数によるグラフ作成 (6)

●標準関数：(6) １次元散布図（Strip Chart）、stripchart()

(a) 因子ベクトルと数値ベクトルから１次元散布図を作成
（１次元散布図は、サンプルサイズが小さい場合に有効）

(a-1) 横向きの１次元散布図（引数 vertical の規定値）

(a-2) 縦向きの１次元散布図（引数 vertical = TRUE）

```
300  # (6) １次元散布図(Strip Chart)、stripchart() --
301
302  ## (a) ３試験区での観測値の分布(サンプルサイズ小)
303  group <- rep(c("A", "B", "C"), each = 3)
304  obs <- c(5, 7, 8, 6, 9, 10, 8, 10, 11)
305
306  ### (a-1) 横向きのグラフ(vertical=FALSE 規定値)
307  stripchart(obs ~ group, las = 1)
308
309  ### (a-2) 縦向きのグラフ
310  stripchart(obs ~ group, vertical = TRUE)
```

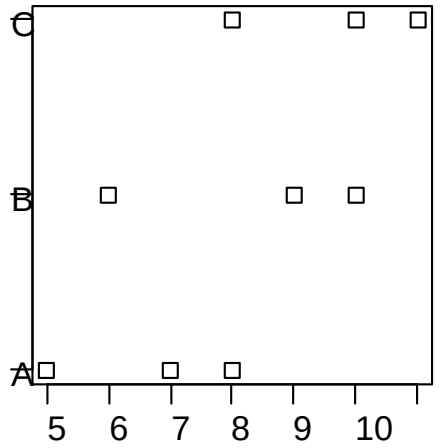
（my_base_graphics3.R：300-310）

人工データ

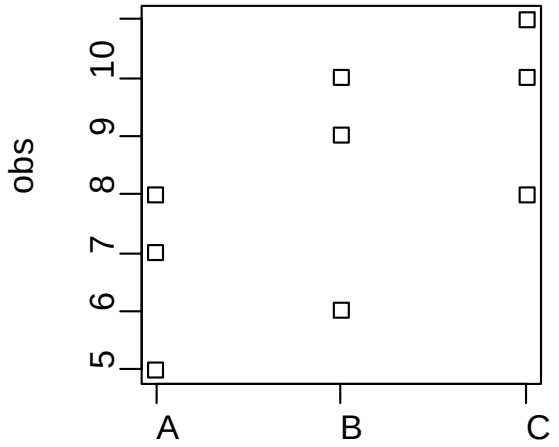
group	obs
A	5
A	7
A	8
B	6
B	9
B	10
C	8
C	10
C	11

目盛ラベルを
水平に表示
引数 las=1

(a-1) １次元散布図



(a-2) １次元散布図



標準関数によるグラフ作成 (6)

●標準関数：(6) １次元散布図（Strip Chart）、stripchart()

(b) PlantGrowth：３試験区の植物の生長量の分布を１次元散布図で可視化
（試験区：対照、処理１、処理２）

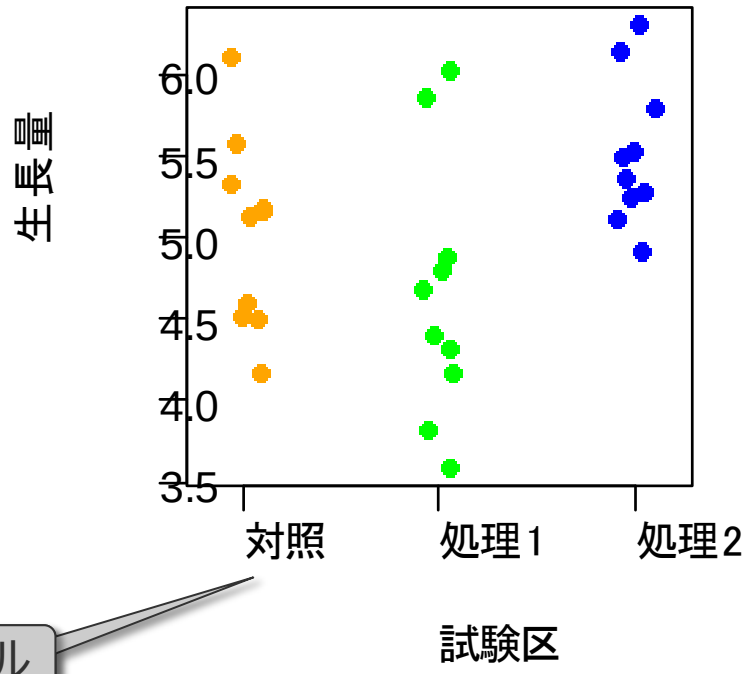
```
312  ## (b) PlantGrowth：3試験区の生長量の分布
313  stripchart(
314    weight ~ group, data = PlantGrowth,
315    vertical = TRUE,    # 縦方向のグラフ
316    method = "jitter", # 点をバラつかせる
317    pch = 16,           # マーカーの種類
318    col = c("orange","green","blue"),
319    xaxt = "n",         # x軸の目盛ラベルを非表示
320    las = 1,            # 目盛ラベルの方向
321    xlab = "試験区",
322    ylab = "生長量")
323
324  axis(side = 1, at = 1:3,
325       labels = c("対照", "処理1", "処理2"))
```

（my_base_graphics3.R：312-325）

目盛ラベルを非表示

目盛ラベル

(b) １次元散布図



標準関数によるグラフ作成 (6)

●標準関数：(6) １次元散布図（Strip Chart）、stripchart()

(c) chickwts：6 種類の飼料を鶏に与えて、6 週間後の体重を測定、測定値の分布を可視化

１次元散布図に箱ひげ図を追加（boxplot() は stripchat() で設定した x 軸とy 軸に従って描画）

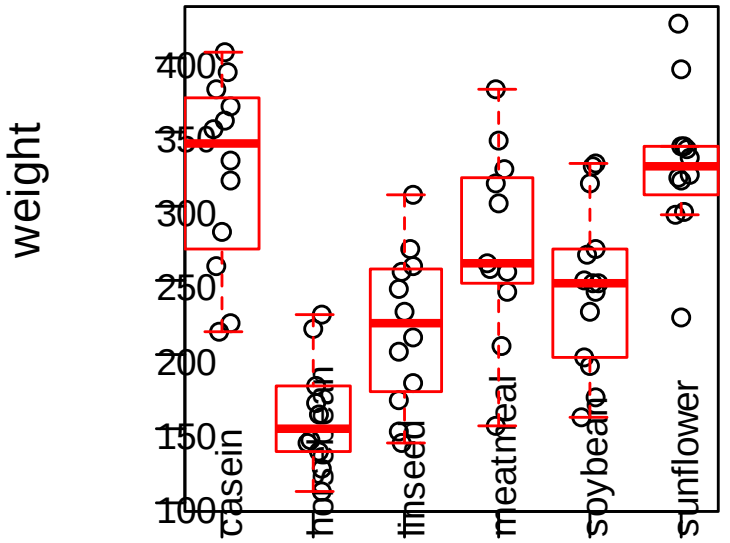
```
327  ## (C) chickwts：飼料の種類が鶏の体重に及ぼす効果
328  stripchart(
329    weight ~ feed, data = chickwts,
330    las = 2,              # 目盛ラベル：xは垂直,yは水平
331    method = "jitter",    # 点をバラつかせる
332    jitter = 0.1,         # ジッターの量 (0~1)
333    pch = 1,              # マーカーの種類
334    cex.axis = 0.8,       # 目盛ラベルの文字サイズ
335    vertical = TRUE)      # 縦方向のグラフ
336
337  boxplot(
338    weight ~ feed, data = chickwts,
339    add = TRUE,           # 既存のプロットに追加
340    outline = FALSE,      # 外れ値を非表示
341    col = NA,             # 箱の塗りつぶしなし
342    border = "red",       # 箱の枠線の色
343    boxwex = 0.8,         # 箱の相対的な大きさを指定
344    axes = FALSE)         # 軸を非表示
```

必須

上書きを指定

目盛ラベル
サイズ 0.8

(c) １次元散布図と箱ひげ図



（my_base_graphics3.R：327-344）

標準関数によるグラフ作成 (7)

●標準関数：(7) ヒストグラム（Histogram）、hist()

連続変数の分布を柱状グラフで可視化

使い方　hist(x) ・・・x：数値ベクトル（数値変数、ランダムに得たサンプル）

関数 hist() の主な引数

breaks：境界値を示す数値ベクトル、境界値を計算する関数、区間の数を示す数値（概数）、
区間を計算するアルゴリズム（"Sturges"(既定値), "Scott", "FD"(Freedman-Diaconis)）

freq：TRUE/FALSE（縦軸が度数／確率密度）、breaks が等距離の場合に既定値は TRUE
breaks が等距離でない場合に既定値は FALSE

probability：使用は非推奨、probability=TRUE は freq=FALSE と同じ意味

right：TRUE/FALSE（区間の境界値は右閉じ左開き（既定値）／右開き左閉じ）

col, border,：バーの背景色、バーの境界線の色

plot：TRUE/FALSE（描画（既定値）／描画せず区間と度数が返される）

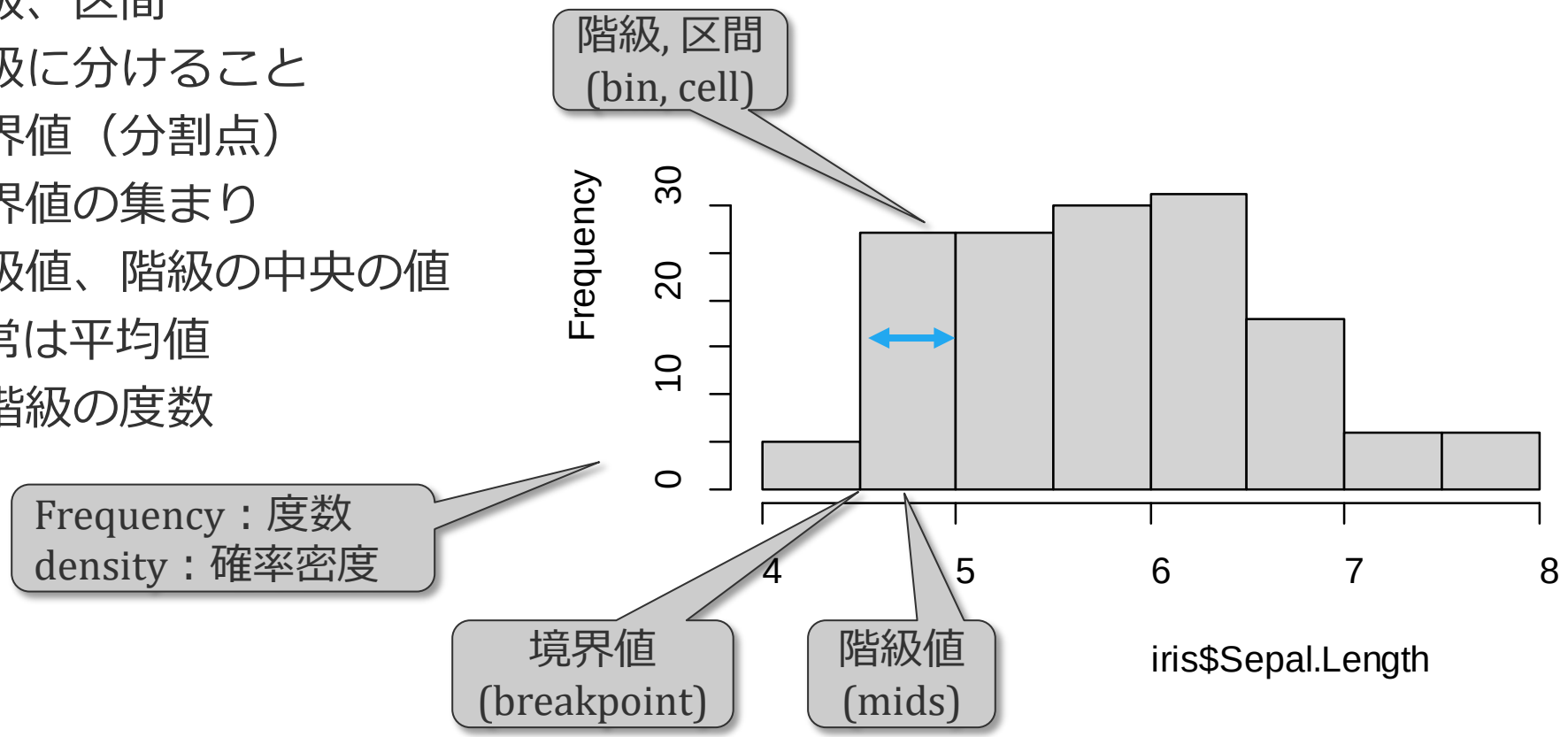
[a, b) の区間
b の値は
その区間に含まれる

標準関数によるグラフ作成 (7)

●標準関数：(7) ヒストグラム（Histogram）、hist()

ヒストグラムの各部分の名称（ヘルプを参照するときに利用）

bin、cell　：階級、区間
binning　：階級に分けること
breakpoint　：境界値（分割点）
breaks　：境界値の集まり
mids　：階級値、階級の中央の値
　通常は平均値
counts　：各階級の度数



標準関数によるグラフ作成 (7)

●標準関数：(7) ヒストグラム（Histogram）、hist()

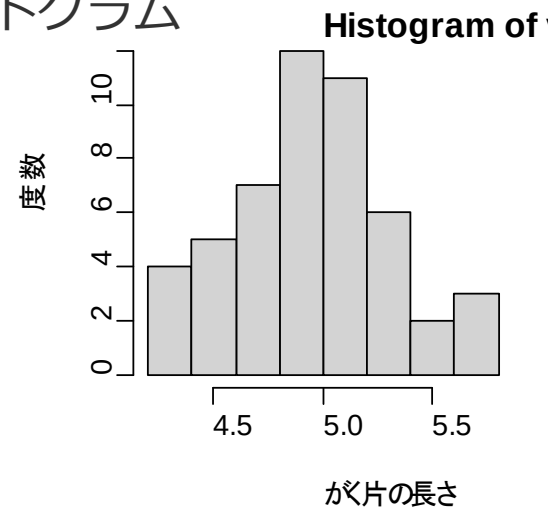
(a) iris：品種 "setosa" のがく片の長さの分布をヒストグラムで可視化

引数 breaks の使い方 (a-1)～(a-5)

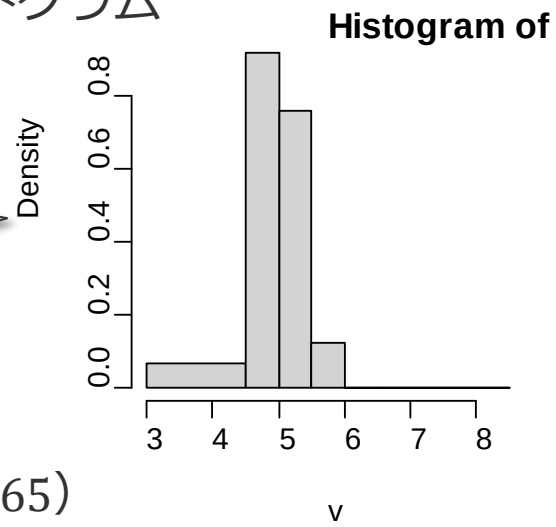
```
347 # (7) ヒストグラム(Histogram)、hist() ----------
348
349 ## (a) iris："setosa" のがく片の長さのヒストグラム
350 v <- iris$Sepal.Length[iris$Species == "setosa"]
351
352 ### (a-1) 既定値(引数 breaks = "Sturges")
353 hist(v, xlab = "がく片の長さ", ylab = "度数")
354
355 ### (a-2) 引数 breaks に区間の概数を渡す
356 hist(v, breaks = 8)
357
358 ### (a-3) 引数 breaks に等間隔の境界値を渡す
359 hist(v, breaks = seq(4, 6.5, by = 0.25))
360
361 ### (a-4) 引数 breaks に不等間隔の境界値を渡す
362 hist(v, breaks = c(3, 4.5, 5, 5.5, 6, 6.5, 8.5))
363
364 ### (a-5) 引数 breaks に計算方式を渡す
365 hist(v, breaks = "Scott")
```

品種 "setosa" のがく片の長さを抽出

(a-1) ヒストグラム



(a-4) ヒストグラム



不等間隔の境界値の場合自動的に確率密度に設定

(my_base_graphics3.R：347-365)

標準関数によるグラフ作成 (7)

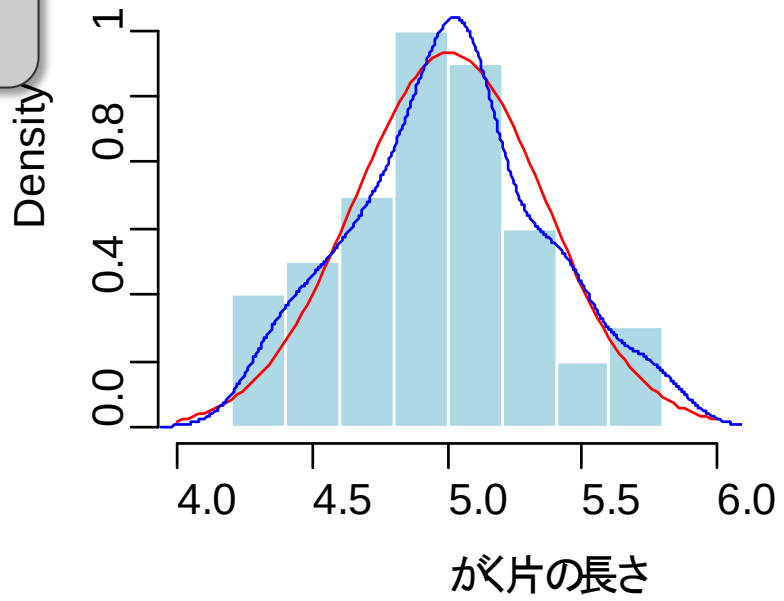
●標準関数：(7) ヒストグラム（Histogram）、hist()

(b) iris：品種 "setosa" のがく片の長さの分布をヒストグラムで可視化
ヒストグラムに正規分布、カーネル密度推定(KDE)による密度曲線を重ねる

```
367 ## (b) iris："setosa" のがく片の長さのヒストグラム
368 ##     ヒストグラムに正規分布、密度曲線を追加
369 y <- iris$Sepal.Length[iris$Species == "setosa"]
370
371 hist(y,
372      freq = FALSE,          # 縦軸を確率密度に指定
373      col = "lightblue",
374      border = "white",
375      xlim = c(4, 6),        # x 軸の範囲
376      main = "",             # タイトルの削除
377      xlab = "がく片の長さ")
378
379 m <- mean(y); s <- sd(y)  # 平均と標準偏差の計算
380 curve(expr = dnorm(x, mean = m, sd = s),
381       add = TRUE,          # 正規分布の曲線を上書
382       col = "red", lwd = 1.5)
383
384 lines(density(y),          # カーネル密度推定(KDE)
385       col = "blue", lwd = 1.5)
```

品種 "setosa" のがく片の長さを抽出

(b) ヒストグラムと正規分布



（my_base_graphics3.R：367-385）

標準関数によるグラフ作成 (7)

●標準関数：(7) ヒストグラム（Histogram）、hist()

(c) iris：がく片の長さの品種別ヒストグラム（マルチパネル）

```
387 ## (c) iris：３品種別のがく片の長さのヒストグラム
388 ##      品種ごとに別パネルで描画(マルチパネル)
389 par(mfrow = c(3, 1))  # 作画領域を 3行× 1列に分割
390
391 sp_names <- unique(iris$Species)    # 品種名を取得
392 x1 <- 3.5; x2 <- 8.5; y1 <- 0; y2 <- 25 # 軸の範囲
393
394 for (s in sp_names) {
395   y <-subset(iris, Species == s)$Sepal.Length
396   m <- mean(y)
397
398   hist(y, xlim = c(x1, x2), ylim = c(y1, y2),
399        main = paste("品種：", s),
400        xlab = "がく片の長さ", ylab = "度数",
401        col = "lightblue", border = "black",
402        breaks = seq(x1, x2, by = 0.5))
403
404   abline(v = m, col = "red")
405   text(x = m + 0.8, y = par("usr")[4] * 0.8,
406        paste("平均", round(m, 2)),col = "red")
407 }
408 par(mfrow = c(1, 1))  # 作業領域をリセット
```

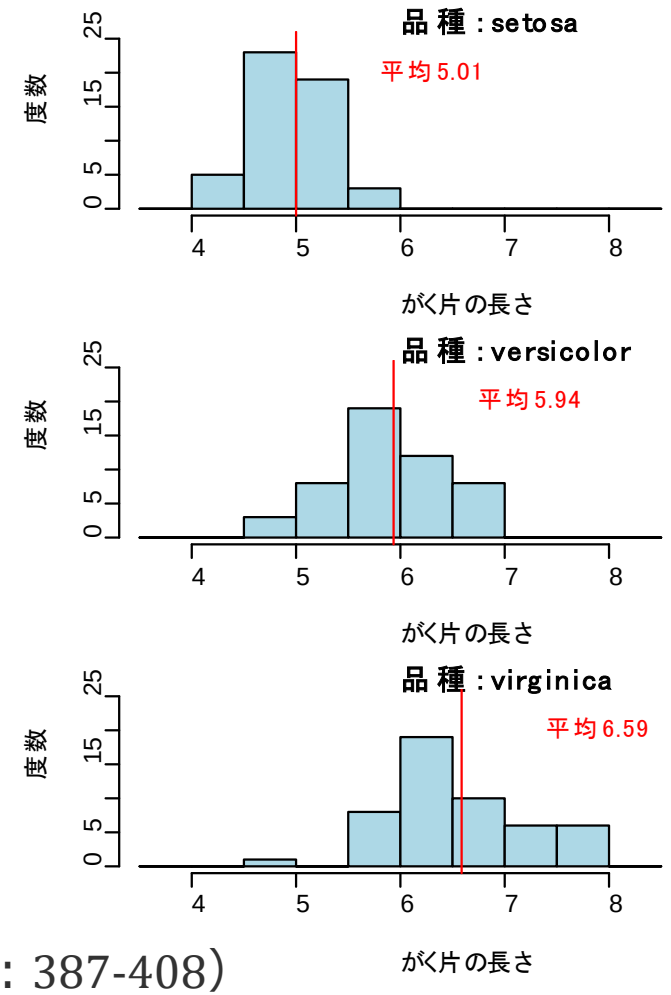
"setosa"
"versicolor"
"virginica"

3つの品種名を
順番に s に付値

par("usr") は
プロット領域の
x 軸と y 軸の座標

第 4 要素 [4] は
y2(上限、最大値)

(c) ヒストグラム



(my_base_graphics3.R：387-408)

標準関数によるグラフ作成 (8)

●標準関数：(8) 幹葉図（Stem-and-Leaf Plot）、stem()

Console にテキストベースで分布を素早く簡便に表示

「|」の左側：幹、実際の値、小数点の位置は１桁目

「|」の右側：葉、０でデータ数を表示

iris：品種 setosa のがく片の長さの幹葉図

(i) データ（昇順）

```
> sort(y)                 # y を昇順にソートして出力
 [1] 4.3 4.4 4.4 4.4 4.5 4.6 4.6 4.6 4.6 4.7 4.7 4.8 4.8
[14] 4.8 4.8 4.8 4.9 4.9 4.9 4.9 5.0 5.0 5.0 5.0 5.1 5.1
[27] 5.0 5.0 5.1 5.1 5.1 5.1 5.1 5.1 5.1 5.1 5.2 5.2 5.2
[40] 5.3 5.4 5.4 5.4 5.4 5.4 5.5 5.5 5.7 5.7 5.8
```

4.4～4.5 のデータ 4 つ

```
411 # (8) 幹葉図(Stem-and-Leaf Plots)、stem() --
412
413 ## iris："setosa" のがく片の長さの分布
414 y <- iris$Sepal.Length[iris$Species == "setosa"]
415 sort(y)                 # y を昇順にソートして出力
416
417 stem(y)                 # scale = 1 (既定値)
418 stem(y, scale = 2)      # scale = 1 の長さの２倍
```

（my_base_graphics3.R：411-418）

(ii)

```
> stem(y)                 # s
cale = 1 (既定値)

  The decimal point is 1
digit(s) to the left of
the |

  42 | 0
  44 | 0000
  46 | 000000
  48 | 000000000
  50 | 00000000000000000
  52 | 0000
  54 | 0000000
  56 | 00
  58 | 0
```

scale = 1

4.4～4.5 のデータ 4 つ

幹（Stem）

葉（Leaf）

(iii)

```
> stem(y, scale = 2)  #
scale = 1 の長さの２倍

  The decimal point is 1
digit(s) to the left of
the |

  43 | 0
  44 | 000
  45 | 0
  46 | 0000
  47 | 00
  48 | 00000
  49 | 0000
  50 | 00000000
  51 | 00000000
  52 | 000
  53 | 0
  54 | 00000
  55 | 00
  56 |
  57 | 00
  58 | 0
```

標準関数によるグラフ作成 (8)

●標準関数：(8) 幹葉図（Stem-and-Leaf Plot）、stem()

Console にテキストベースで分布を素早く簡便に表示

「|」の左側：幹、実際の値、小数点の位置は 1 桁目

「|」の右側：葉、 0 でデータ数を表示

iris：品種 setosa のがく片の長さの幹葉図

(i) データ（昇順）

```
> sort(y)                 # y を昇順にソートして出力
 [1] 4.3 4.4 4.4 4.4 4.5 4.6 4.6 4.6 4.6 4.7 4.7 4.8 4.8
[14] 4.8 4.8 4.8 4.9 4.9 4.9 4.9 5.0 5.0 5.0 5.0 5.1 5.1
[27] 5.0 5.0 5.1 5.1 5.1 5.1 5.1 5.1 5.1 5.1 5.2 5.2 5.2
[40] 5.3 5.4 5.4 5.4 5.4 5.4 5.5 5.5 5.7 5.7 5.8
```

4.4～4.5 のデータ 4 つ

```
411 # (8) 幹葉図(Stem-and-Leaf Plots)、stem() --
412
413 ## iris："setosa" のがく片の長さの分布
414 y <- iris$Sepal.Length[iris$Species == "setosa"]
415 sort(y)                 # y を昇順にソートして出力
416
417 stem(y)                 # scale = 1 (既定値)
418 stem(y, scale = 2)      # scale = 1 の長さの２倍
```

（my_base_graphics3.R：411-418）

(ii)

```
> stem(y)                 # s
cale = 1 (既定値)

  The decimal point is 1
digit(s) to the left of
the |

  42 | 0
  44 | 0000
  46 | 000000
  48 | 000000000
  50 | 00000000000000000
  52 | 0000
  54 | 0000000
  56 | 00
  58 | 0
```

scale = 1

4.4～4.5 のデータ 4 つ

幹（Stem）

葉（Leaf）

(iii)

```
> stem(y, scale = 2)   #
scale = 1 の長さの２倍

  The decimal point is 1
digit(s) to the left of
the |

  43 | 0
  44 | 000
  45 | 0
  46 | 0000
  47 | 00
  48 | 00000
  49 | 0000
  50 | 00000000
  51 | 00000000
  52 | 000
  53 | 0
  54 | 00000
  55 | 00
  56 |
  57 | 00
  58 | 0
```

scale = 2

2 倍

標準関数によるグラフ作成 (9)

●標準関数：(9) 棒グラフ（Bar Graph）、barplot()、plot()

数値ベクトルや行列などをもとに、量の大小を棒の長さで可視化

使い方

plot(x)	x：因子ベクトル（因子別に集計）
barplot(y)	y：数値ベクトル（各要素が 1 本の棒に対応） マトリックス（列ごとに棒（積上げ、並列）になる）
barplot(tb)	tb：テーブルオブジェクト（table() による集計）
barplot(y ~ x)	y：数値ベクトル、x：因子ベクトル
barplot(y ~ x, data = df)	y：データフレームの数値ベクトルと因子ベクトル
barplot(y ~ x1 + x2)	y：数値ベクトル、x1, x2：因子ベクトル
barplot(cbind(y1,y2) ~ x)	y1,y2：数値ベクトル、x：因子ベクトル、mx <- cbind(y1,y2) マトリックス（行列）にまとめる

特徴

関数 barplot は、凡例を表示する引数をもっている柔軟な関数（通常、関数 legend() を使用）

標準関数によるグラフ作成 (9)

●標準関数：(9) 棒グラフ（Bar Graph）、barplot()、plot()

関数 barplot() の主な引数

horiz：TRUE/FALSE（横向きの棒グラフ／縦向きの棒グラフ（既定値））

beside： TRUE/FALSE（並列棒グラフ／積み上げ棒グラフ（既定値））

width：棒の幅を表す数値ベクトル

space：棒と棒の間隔、beside=FALSE の場合、棒の間隔の数値を指定

beside=TRUE の場合、２要素の数値ベクトル、c(グループ内間隔、グループ間間隔)

names.arg：棒（またはグループ）のラベルの文字ベクトル、axisnames=FALSE で非表示

legend.text：TRUE（凡例に height のマトリックスの行名を利用）、文字ベクトル

args.legend：凡例の位置、枠線の有無を指定するリストを渡す、list(x = "topright", bty = "n")

x,y(凡例の位置)、bty(枠線の有無)、title(タイトル)、cex(文字サイズ)、horiz(横書き)

col, border：棒の塗つぶしの色、棒の枠線の色、ベクトル c("skyblue", "orange")

density：棒の塗りつぶしをハッチング（斜線模様）で描くときの線の密度、数値ベクトル

axes：TRUE/FALSE（軸の表示／非表示）、非表示にして関数 axis() で軸を追加

標準関数によるグラフ作成 (9)

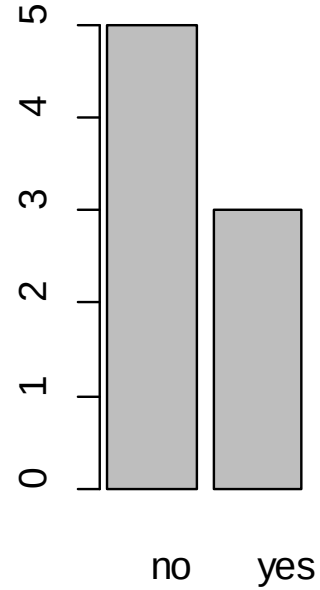
●標準関数：(9) 棒グラフ（Bar Graph）、barplot()、plot()

(a) 因子ベクトル、(b) 数値ベクトル、(c) 数値ベクトルと因子ベクトル

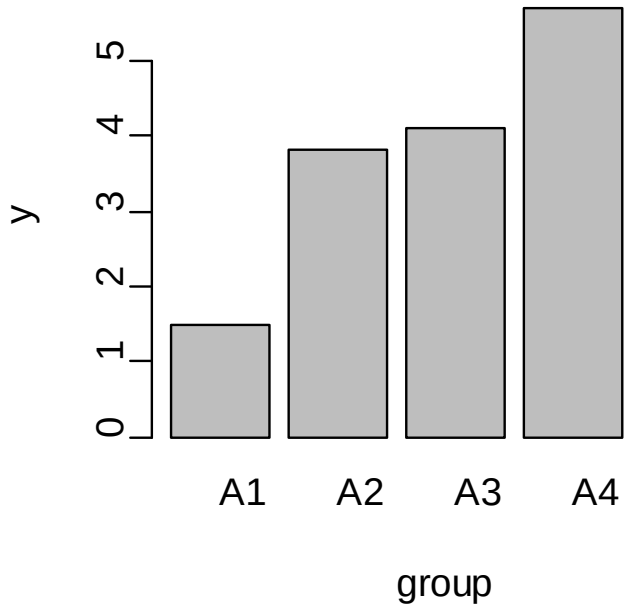
```
421  # (9) 棒グラフ(Bar Graph) 、barplot()、plot() --
422
423  ## (a) １つの因子ベクトルの集計結果
424  ans <- factor(
425    c("yes","no","yes","yes","no","no","no","no"),
426    levels = c("no","yes"))
427
428  plot(ans)        # levels の指定の順序で並ぶ
429
430  ## (b) １つの数値ベクトル（名前付き）
431  y <- c(1.5, 3.8, 4.1, 5.7)
432  names(y) <- c("A1","A2","A3","A4")
433
434  barplot(y)
435
436  ## (c) 数値ベクトル～因子ベクトル
437  y <- c(1.5, 3.8, 4.1, 5.7)
438  group <- factor(c("A1","A2","A3","A4"))
439
440  barplot(y ~ group)
```

（my_base_graphics3.R：421-440）

(a) 棒グラフ



(b)(c) 棒グラフ



標準関数によるグラフ作成 (9)

●標準関数：(9) 棒グラフ（Bar Graph）、barplot()、plot()

(d) 数値ベクトル～因子ベクトル＋因子ベクトル

積上げ棒グラフと並列棒グラフを作成

```
442  ## (d) 数値ベクトル～因子ベクトル+因子ベクトル
443  group <- factor(c("A","A","B","B","C","C"),
444                  levels = c("B", "C", "A"))
445  ans <- factor(c("yes","no","yes","no","yes","no"),
446                  levels = c("yes", "no"))
447  y <- c(11, 9, 15, 4, 19, 3)
448  data.frame(group, ans, y)     # 対応関係を表示
449
450  ### (d-1) 積上げ棒グラフ(既定値)
451  barplot(y ~ ans + group,
452          legend.text = TRUE,   # 凡例を表示
453          ylim = c(0,40))       # 凡例のスペース確保
454
455  ### (d-2) 並列棒グラフ
456  barplot(y ~ ans + group,
457          beside = TRUE,        # 並列棒グラフを指定
458          space = c(0.1, 0.5), # 棒と棒の間隔
459          legend.text = TRUE)  # 凡例を表示
```

因子の順序

beside = FALSE
省略（既定値）

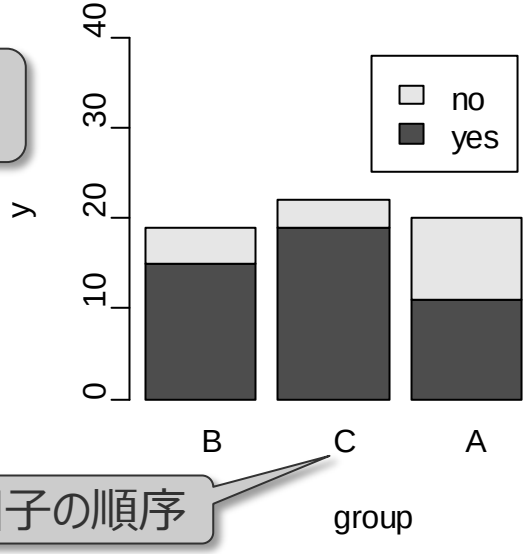
因子の順序

（my_base_graphics3.R：442-459）

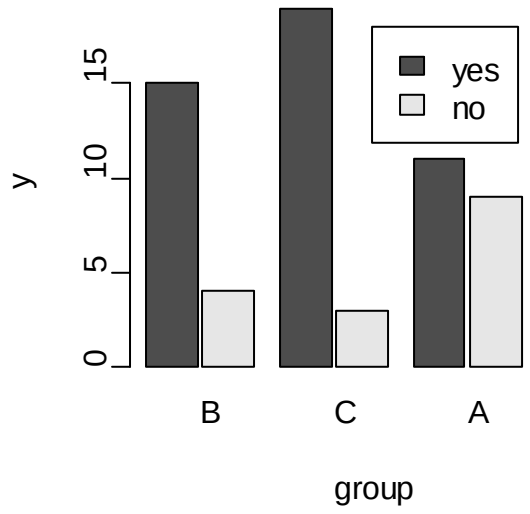
```
> data.frame(group, ans, y)
  group ans  y
1     A yes 11
2     A  no  9
3     B yes 15
4     B  no  4
5     C yes 19
6     C  no  3
```

回答数

(d-1) 積上げ棒グラフ



(d-2) 並列棒グラフ



標準関数によるグラフ作成 (9)

●標準関数：(9) 棒グラフ（Bar Graph）、barplot()、plot()

(e) 複数の数値ベクトル（行列）～因子ベクトル
積上げ棒グラフと並列棒グラフを作成

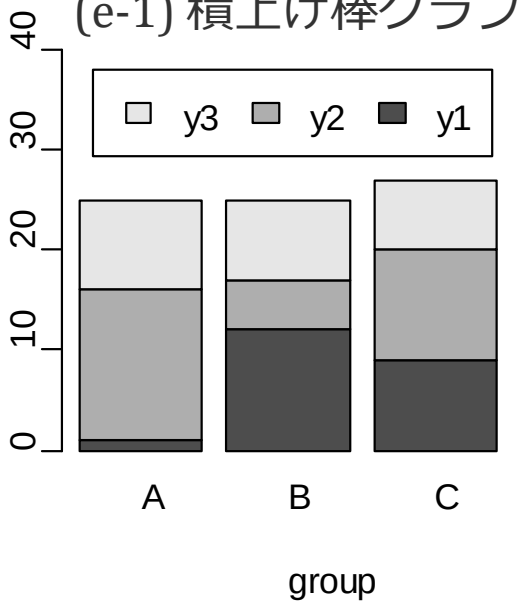
```
461  ## (e) 複数の数値ベクトル～因子ベクトル
462  group <- factor(c("A","B","C"))
463  y1 <- c( 1, 12,  9)
464  y2 <- c(15,  5, 11)
465  y3 <- c( 9,  8,  7)
466  mx <- cbind(y1, y2, y3) # マトリックス作成
467  print(mx)
468
469  ### (e-1) 積上げ棒グラフ(既定値)
470  barplot(mx ~ group,
471          legend.text = TRUE,
472          args.legend = list(horiz = TRUE),
473          ylim = c(0, 40))
474
475  ### (e-2) 並列棒グラフ
476  barplot(mx ~ group,
477          beside = TRUE,       # 並列棒グラフ
478          legend.text = TRUE,
479          args.legend = list(horiz = TRUE),
480          ylim = c(0, 25))
```

マトリックス（行列）

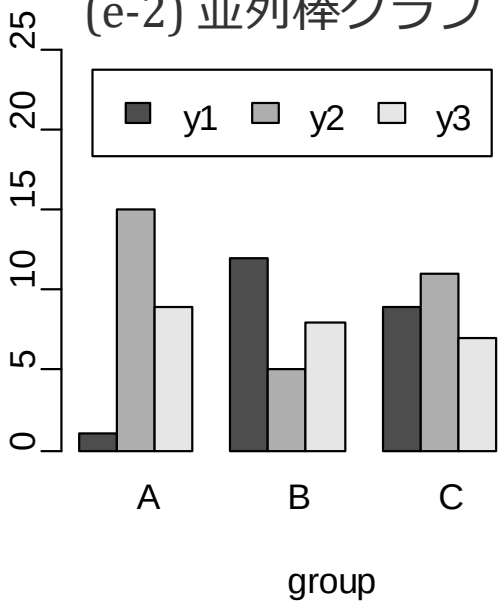
```
> print(mx)
     y1 y2 y3
[1,]  1 15  9
[2,] 12  5  8
[3,]  9 11  7
```

group に対応 A
B
C

(e-1) 積上げ棒グラフ



(e-2) 並列棒グラフ



（my_base_graphics3.R：461-480）

標準関数によるグラフ作成 (9)

●標準関数：(9) 棒グラフ（Bar Graph）、barplot()、plot()

(f) マトリックス（行列）

積上げ棒グラフ（実数、割合）

```
482 ## (f) マトリックス、%表示の積上げ棒グラフ
483 mx <- matrix(c(10, 12, 18, 21,
484                15, 18,  5, 10,
485                 8, 11, 10,  3),
486               nrow = 3, ncol = 4, byrow = TRUE)
487 dimnames(mx) <- list(
488   ans = c("−", "+", "++"),          # 行の項目名
489   group = c("G1", "G2", "G3","G4"))  # 列の項目名
490 print(mx)
491
492 ### (f-1)  積上げ棒グラフ(実数)
493 barplot(mx,
494         ylim = c(0, 50), las = 1,
495         legend.text = TRUE,  # 凡例を表示
496         args.legend = list(  # 凡例のカスタマイズ
497           x = "top",
498           horiz = TRUE,
499           bty = "n"),
500         beside = FALSE)       # 積上げ棒グラフ
```

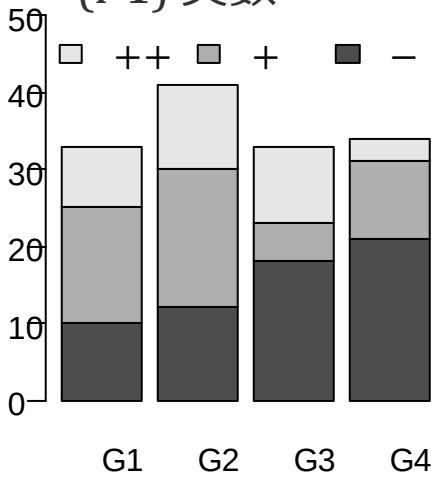
(my_base_graphics3.R：482-511, 省略 514-523)

1：行合計に対する割合
2：列合計に対する割合
規定値：全合計に対する割合

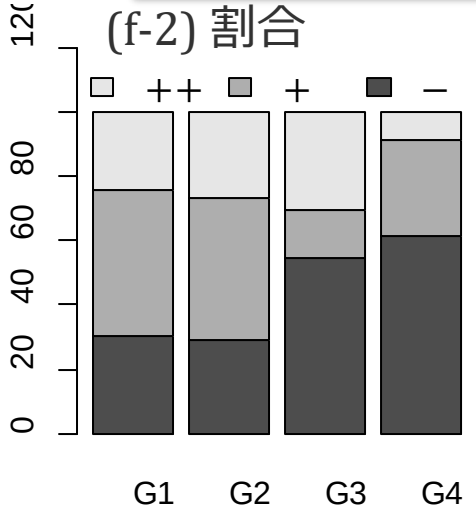
```
502 ### (f-2) 積上げ棒グラフ(割合)
503 mx_percent <- prop.table(mx, margin = 2) * 100
504 barplot(mx_percent,              # % データ
505         ylim = c(0, 120),
506         legend.text = TRUE,  # 凡
507         args.legend = list(  # 凡
508           x = "top",
509           horiz = TRUE,
510           bty = "n"),
511         beside = FALSE)       # 積
```

```
> print(mx)
      group
ans    G1 G2 G3 G4
  −    10 12 18 21
  +    15 18  5 10
  ++    8 11 10  3
```

(f-1) 実数



(f-2) 割合



標準関数によるグラフ作成 (9)

●標準関数：(9) 棒グラフ（Bar Graph）、barplot()、plot()

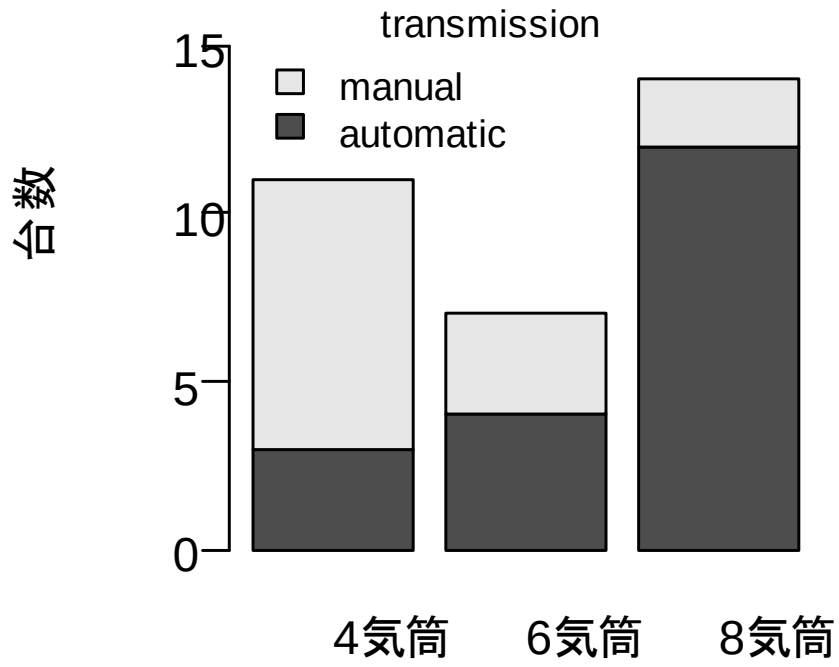
(g) mtcars：トランスミッションと気筒数でクロス集計を作成
作成したテーブル・オブジェクトを棒グラフで可視化

```
> print(tb)

   4  6  8
0  3  4 12
1  8  3  2
```

```
525 ## (g) mtcars：集計値(テーブル・オブジェクト)
526 tb <- table(mtcars$am, mtcars$cyl)
527 print(tb)
528
529 barplot(
530   tb,
531   ylim = c(0, 17), ylab = "台数",
532   las = 1,
533   names.arg = c("4気筒", "6気筒", "8気筒"),
534   legend.text = c("automatic", "manual"),
535   args.legend = list(          # 凡例の細部設定
536     x = "topleft",             # 左上に配置
537     bty = "n",                 # 枠なし
538     title = "transmission",    # 凡例のタイトル
539     ncol = 1,                  # 1 列
540     cex = 0.8,                 # 文字サイズ
541     inset = 0.02               # 内側に少し余白
542   )
543 )
```

（my_base_graphics3.R：525-543）

(g) 積上げ棒グラフ



標準関数によるグラフ作成 (10)

●標準関数：(10) ドットプロット（Cleveland Dot Plot）、dotchart()

(a) VADeaths：マトリックスを可視化

```
> print(VADeaths)
      Rural Male Rural Female Urban Male Urban Female
50-54       11.7          8.7       15.4          8.4
55-59       18.1         11.7       24.3         13.6
60-64       26.9         20.3       37.0         19.3
65-69       41.0         30.9       54.6         35.1
70-74       66.0         54.3       71.1         50.0
```

1000人当たりの死亡率（‰）

```
546 # (10) ドットプロット(Cleveland Dot Plot)
547 #       dotchart() ----------------------------
548
549 ## (a) VADeaths：死亡率を地域・性別と年齢層で比較
550 print(VADeaths)              # マトリックスを表示
551
552 dotchart(VADeaths,
553           main = "年齢層・グループ別死亡率",
554           xlab = "死亡率 (‰)",
555           color = "blue",     # 点の色
556           gcolor = "red",     # グループラベルの色
557           cex = 0.8,          # 文字サイズ
558           pch = 19)           # 点のシンボル
```

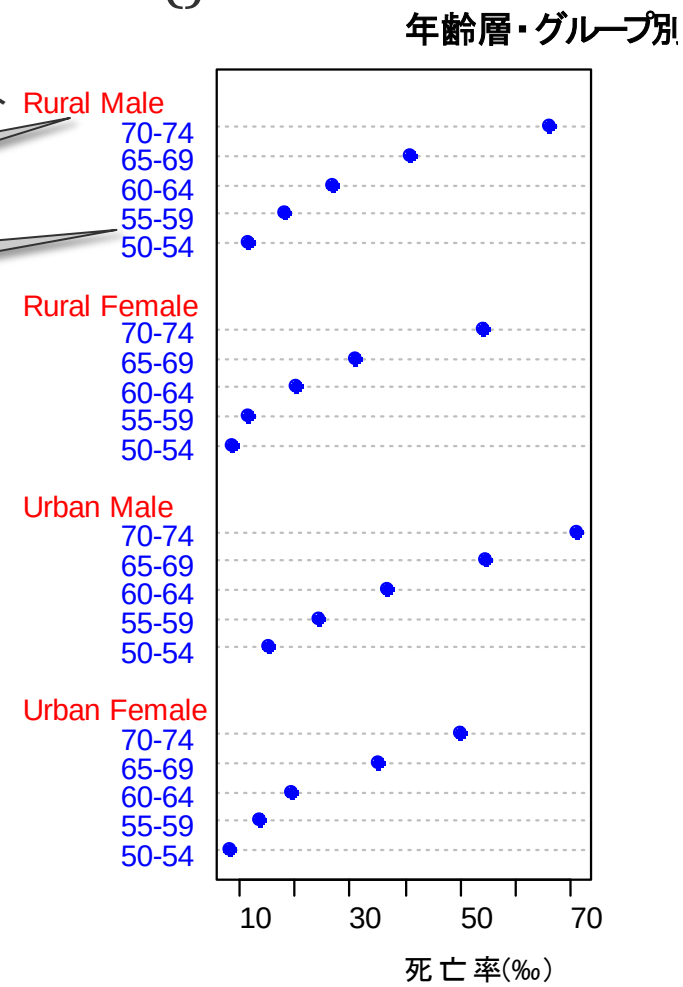
マトリックス

（my_base_graphics3.R：546-558）

(a) Cleveland ドットプロット

グループラベル(列)

複数のカテゴリ(行)



標準関数によるグラフ作成 (10)

●標準関数：(10) ドットプロット（Cleveland Dot Plot）、dotchart()

(a) VADeaths：マトリックスを可視化

```
> print(VADeaths)
      Rural Male Rural Female Urban Male Urban Female
50-54       11.7          8.7       15.4          8.4
55-59       18.1         11.7       24.3         13.6
60-64       26.9         20.3       37.0         19.3
65-69       41.0         30.9       54.6         35.1
70-74       66.0         54.3       71.1         50.0
```

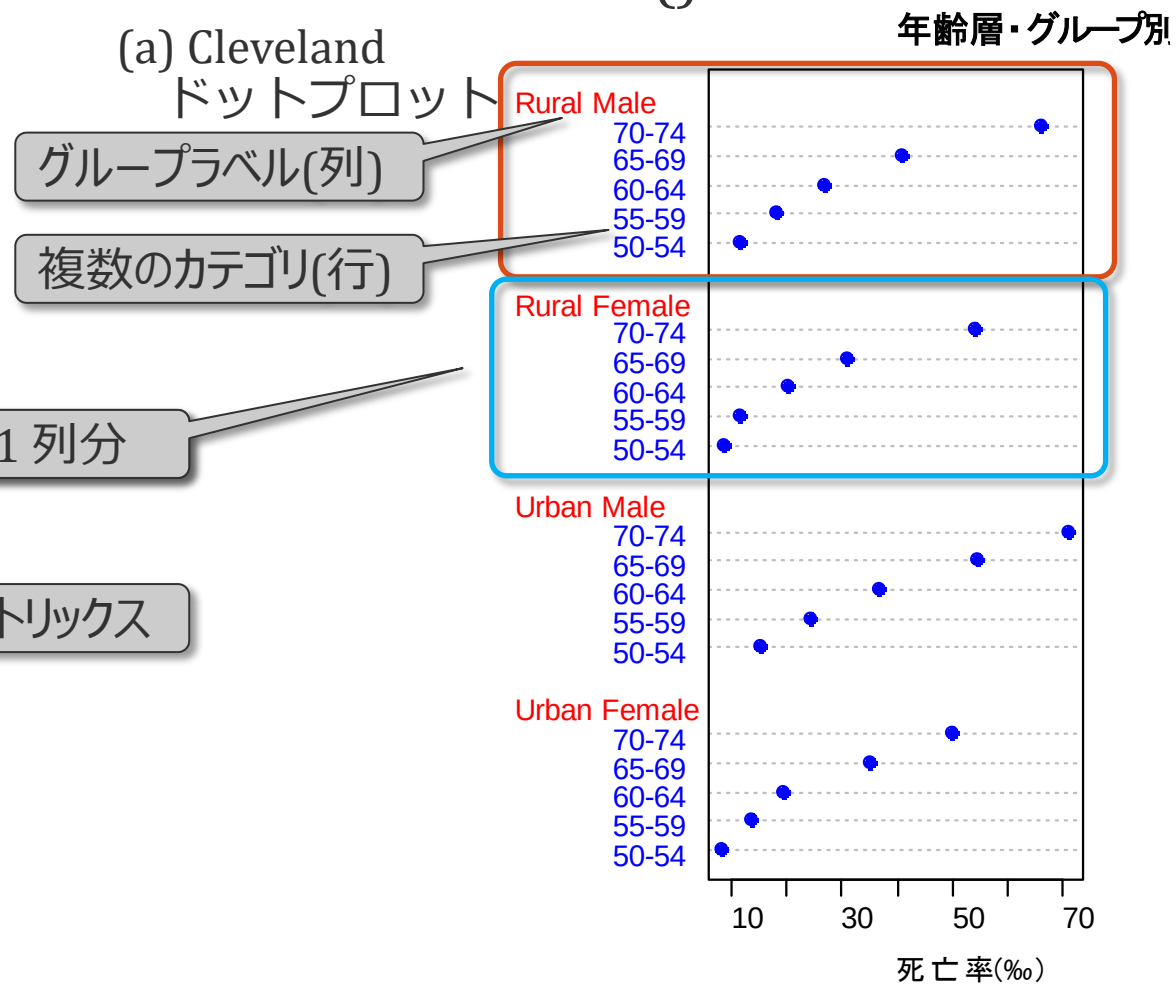
```
546 # (10) ドットプロット(Cleveland Dot Plot)
547 #       dotchart() ----------------------------
548
549 ## (a) VADeaths：死亡率を地域・性別と年齢層で比較
550 print(VADeaths)              # マトリックスを表示
551
552 dotchart(VADeaths,
553           main = "年齢層・グループ別死亡率",
554           xlab = "死亡率 (‰)",
555           color = "blue",     # 点の色
556           gcolor = "red",     # グループラベルの色
557           cex = 0.8,          # 文字サイズ
558           pch = 19)           # 点のシンボル
```

1 列分

マトリックス

（my_base_graphics3.R：546-558）

(a) Cleveland ドットプロット



標準関数によるグラフ作成 (10)

●標準関数：(10) ドットプロット（Cleveland Dot Plot）、dotchart()

(b) mtcars：データフレーム
気筒数でグループ分けした車種と燃費の関係

```
560 ## (b) mtcars：データフレーム
561 df <- mtcars[order(mtcars$mpg), ]  # 昇順に並び替え
562 g <- factor(df$cyl,
563              labels = c("4気筒", "6気筒", "8気筒"))
564 colors <- c("red", "blue", "green")[as.numeric(g)]
565
566 data.frame(df$mpg, g, colors)
567
568 dotchart(
569   df$mpg,
570   labels = rownames(df),
571   groups = g,        # グループ分けする因子ベクトル
572   main = "気筒数別・車種ごとの燃費",
573   xlab = "燃費(MPG)",
574   gcolor = "black", # グループ名の色
575   color = colors,   # グループごとの色
576   pch = 19)
```

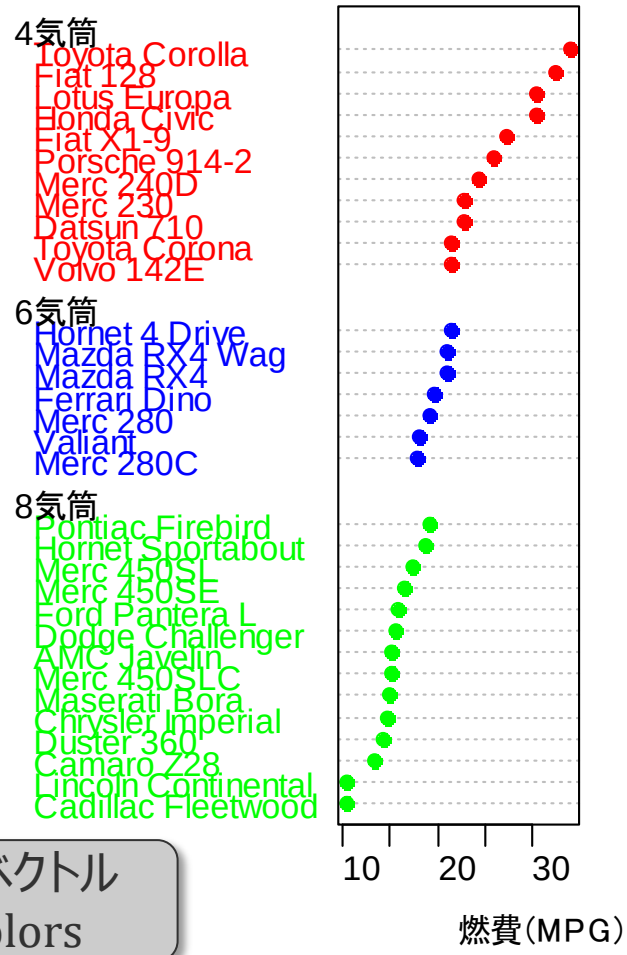
（my_base_graphics3.R：560-576）

```
> data.frame(df$mpg, g, colors)
   df.mpg      g colors
1    10.4 8気筒  green
2    10.4 8気筒  green
3    13.3 8気筒  green
4    14.3 8気筒  green
5    14.7 8気筒  green
6    15.0 8気筒  green
12   17.3 8気筒  green
13   17.8 6気筒   blue
14   18.1 6気筒   blue
15   18.7 8気筒  green
16   19.2 6気筒   blue
17   19.2 8気筒  green
```

因子ベクトル g

因子ベクトル colors

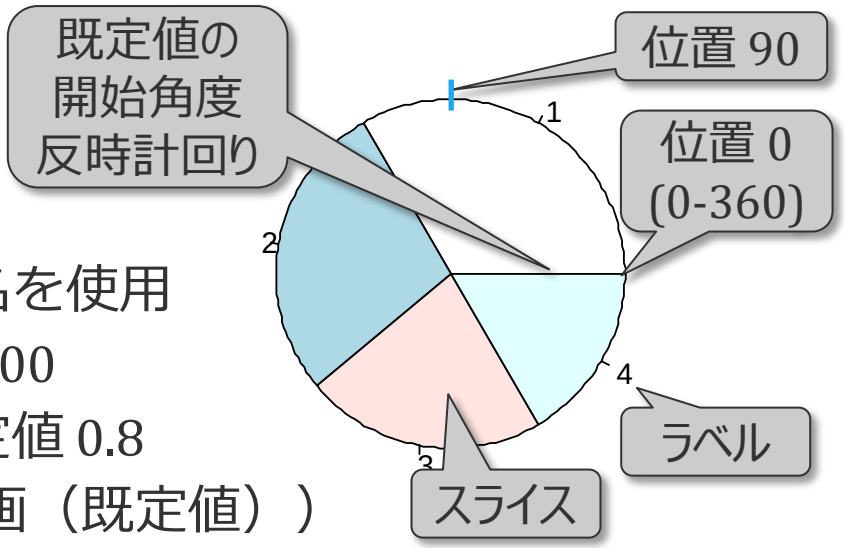
(b) Cleveland ドットプロット



標準関数によるグラフ作成 (11)

●標準関数：(11) 円グラフ（Pie Chart）、pie()

円全体を100%として、各項目が全体に占める割合を扇形の形（スライス）で表現したグラフ
（使用に対して否定的な意見あり、使用には注意が必要）
使い方　pie(x)　　x：数値ベクトル（負数がないこと）



pie() の主な引数

labels：ラベルの文字ベクトル、指定なしでベクトルの要素名を使用
edges：円グラフの輪郭を描画するのに使う区分数、既定値 200
radius：円グラフの大きさ（1 より大きいとはみ出す）、既定値 0.8
clockwise：TRUE/FALSE（時計回りの描画／反時計回りの描画（既定値））
init.angle：開始位置（0～360）、既定値は clockwise ≠ TRUE で0（3時）、TRUE で 90（12時）
　　多くの場合、init.angle = 90, clockwise = TRUE（12時から時計回り）が使われる
density：１インチ当たりの陰影線の密度を指定する数値、既定値 NULL
angle：陰影線の傾き（角度）を指定する数値、既定値 45
col, border, lty：スライスの色、境界線の色、線種の指定、いずれも既定値は NULL

標準関数によるグラフ作成 (11)

●標準関数：(11) 円グラフ（Pie Chart） 、 pie()

4 水準のデータ（要素数 4 の数値ベクトル）

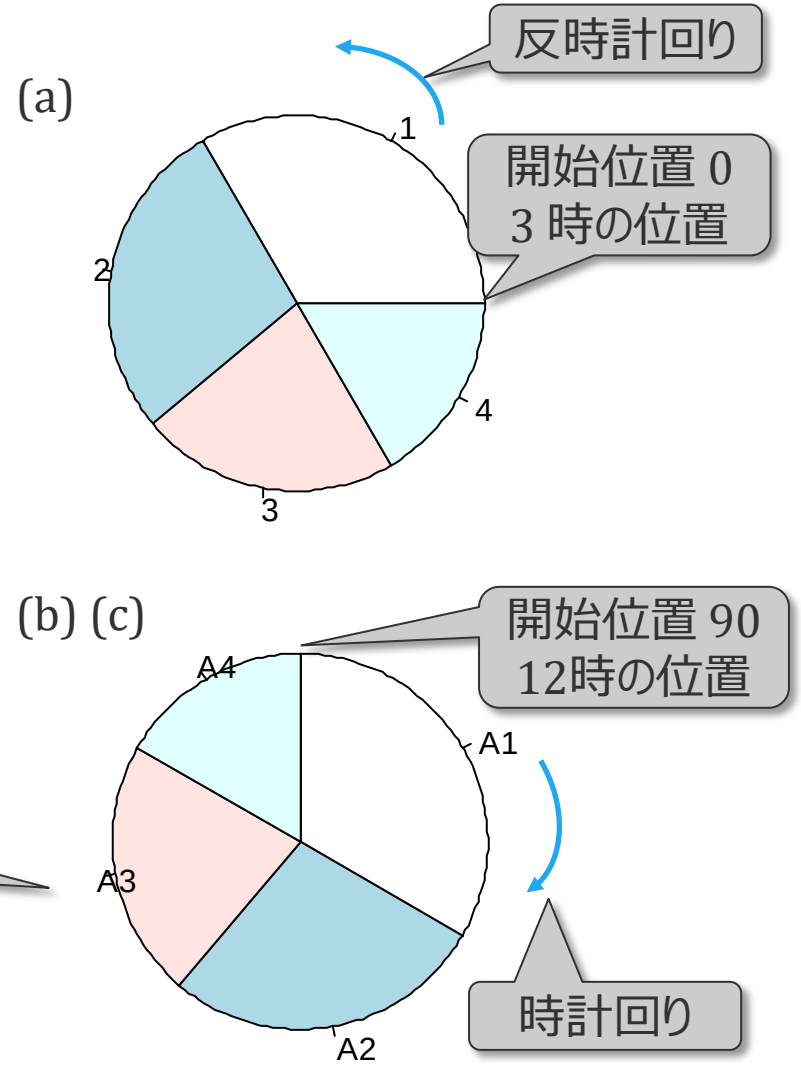
(a) 反時計回り、開始位置は 3 時（0,既定値）

(b) 時計回り（clockwise = TRUE）、開始位置は 12時（90,既定値）

(c) (b) と同じ設定、ラベル表示

```
579 # (11) 円グラフ(Pie Chart)、pie() --------------
580
581 ## (a) 4水準、反時計回り、開始位置は3時(規定値)
582 y1 <- c(30, 25, 20, 15)
583 pie(y1)    # 規定値での描画
584
585 ## (b) 4水準、時計回りで開始位置は12時（既定値）
586 y2 <- c(A1 = 30, A2 = 25, A3 = 20, A4 = 15)
587 pie(y2, clockwise = TRUE)  # 省略 init.angle=90
588
589 ## (c) 4水準、ラベル表示
590 y3 <- c(30, 25, 20, 15)
591 lab <- c("A1", "A2", "A3", "A4")
592 pie(y3, clockwise = TRUE, labels = lab)
```

（my_base_graphics3.R：579-592）



標準関数によるグラフ作成 (11)

●標準関数：(11) 円グラフ（Pie Chart） 、 pie()

(d) 割合で表示する円グラフ

関数 round()：四捨五入する

関数 paste()：文字列の連結

一般的な設定（12時の位置（90）から時計回り）

init.angle = 90, clockwise = TRUE

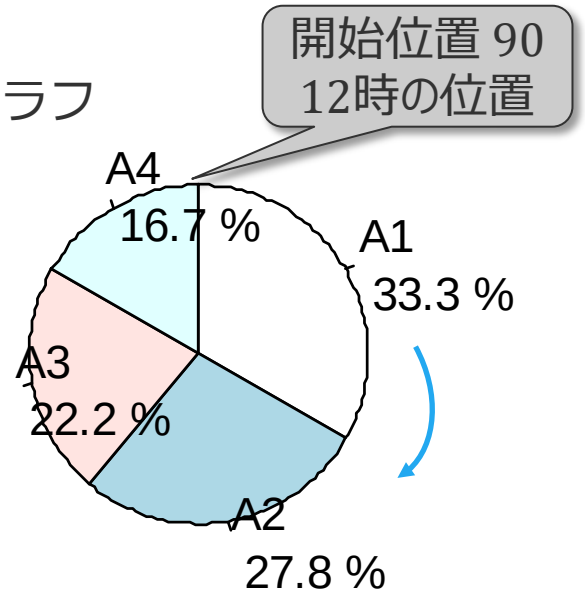
```
594  ## (d) 4水準、% 表示
595  y4 <- c(30, 25, 20, 15)
596  lab <- c("A1", "A2", "A3", "A4")
597  p <- round(y4/sum(y4) * 100, 1)  # 割合を四捨五入
598  tx <- paste(lab, "\n", p, "%")   # ラベルの文字列
599  pie(y4,
600      init.angle = 90, clockwise = TRUE, # 一般的
601      labels = tx,
602      cex = 1.1)
```

（my_base_graphics3.R：594-602）

文字の連結
"¥n" ：改行

一般的な設定

(d) 円グラフ



標準関数によるグラフ作成 (12)

●分割表の可視化

モザイク図（Mosaic Plot）

２次元～多次元の分割表の可視化

すべての水準の組合せたプロット

関連プロット（Cohen-Friendlyの連関図、Association Plot）

２次元の分割表の可視化

四分表示（四分割表示、Fourfold Display for 2 by 2）

２行２列の分割表の可視化

スパインプロット（Spine Plot）

２変量データに対するモザイク図の一種

スピノグラム（Spinogram）：スパインプロットの一種、説明変数が連続変数の場合

標準関数によるグラフ作成 (12)

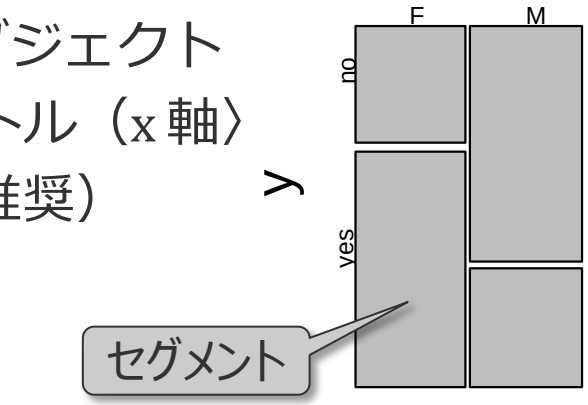
●標準関数：(12) モザイク図（Mosaic Plot）、mosaicplot()、plot()

mosaicplot()：クロス集計表の各セルを長方形の面積で表現し、データの大きさを可視化

使い方　mosaicplot(tb)　・・・tb：table()、xtabs() で作成した分割表オブジェクト

mosaicplot(~ x + y) ・・y：因子ベクトル（y 軸）、x：因子ベクトル（x 軸〉

mosaicplot(x ~ y)・・・plot() と逆の配置に注意（上の使い方を推奨）



関数 mosaicplot() の主な引数

subset：データを絞り込む条件を指定

dir：各変数がプロットのどの方向を分割するか指定、"v"（垂直方向）、"h"（水平方向）

規定値（NULL）では、垂直方向の分割→垂直方向の分割、交互に分割

off：各セグメント間の隙間の割合（0～20、特に 3～8 が適切）

color：セグメントの塗りつぶしの色、共通するセグメントを塗り分け、TRUE/FALSE

border：セグメントの境界線（枠線）の色

cex.axis：軸ラベルの文字サイズ、既定値 0.66

標準関数によるグラフ作成 (12)

●標準関数：(12) モザイク図（Mosaic Plot）、mosaicplot()、plot()

(a) 男女別（Male/Female）の回答数（yes/no）の比較

```
605 # (12) モザイク図(Mosaic Plot)
606 #      mosaicplot()、plot() -------------------
607
608 ## (a) 男女別の回答数の比較(対応のある因子ベクトル)
609 sex <- factor( c(rep("male", 4), rep("female", 5)),
610                levels = c("male", "female"))
611 answer <- factor(
612   c("no",  "no", "yes", "yes",
613     "no",  "no", "yes",  "no",  "no"),
614   levels = c("no", "yes"))
615 tb_out <- table(sex, answer)  #因子の順序に注意
616 print(tb_out)
617
618 ### (a-1) sex を横軸、answer を縦軸に配置
619 plot(sex, answer)       # スパインプロット(参考)
620 plot(answer ~ sex)      # スパインプロット(参考)
621 plot(tb_out)             # モザイク図
622
623 ### (a-2) sex を横軸、answer を縦軸に配置
624 mosaicplot(sex ~ answer)       # plotの設定と逆
625 mosaicplot(~ sex + answer,cex.axis=1.2)  # 推奨
626 mosaicplot(tb_out,cex.axis=1.2)
```

水準の順序

水準の順序

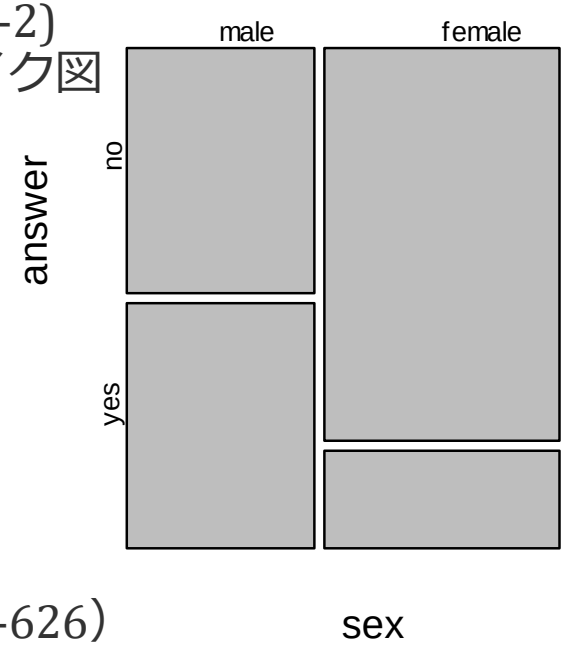
2×2 分割表

モザイク図の一種（後述）

2×2 分割表

```
> print(tb_out)
        answer
sex      no yes
  male    2   2
  female  4   1
```

(a-1) (a-2) モザイク図



（my_base_graphics3.R：605-626）

標準関数によるグラフ作成 (12)

●標準関数：(12) モザイク図（Mosaic Plot）、mosaicplot()、plot()

(b) HairEyeColor： 男性 279 人の髪の色と目の色の関係、相互関係や独立性を検討

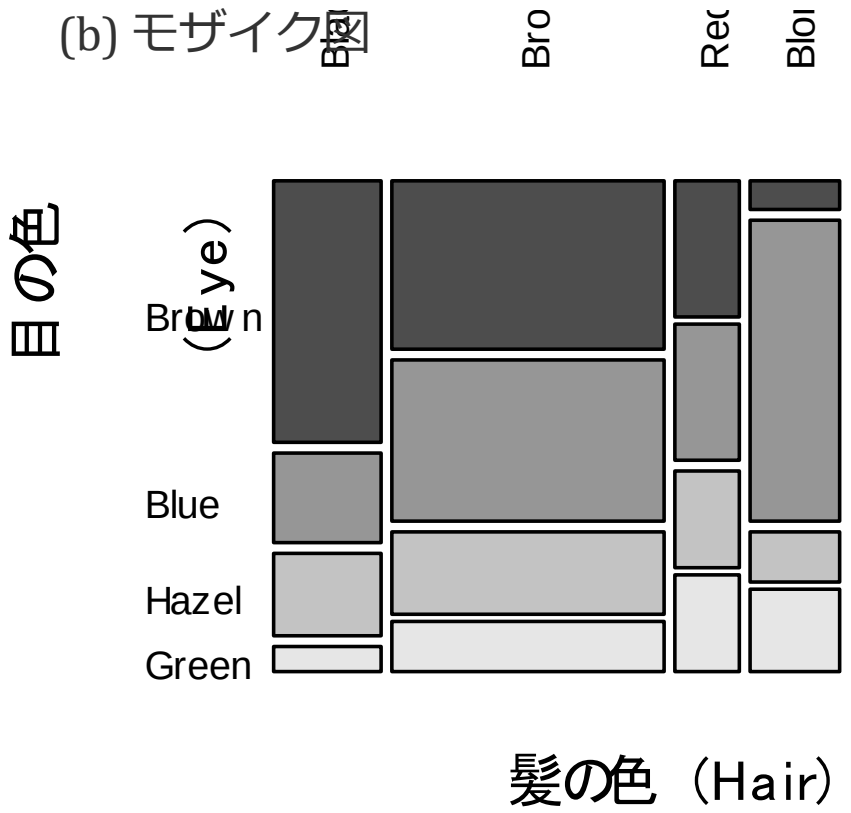
```
> print(mx)
       Eye
Hair    Brown Blue Hazel Green
  Black    32   11    10     3
  Brown    53   50    25    15
  Red      10   10     7     7
  Blond     3   30     5     8
```

4×4 分割表

```
628  # (b) HairEyeColor： 男性の髪と目の色の関係
629  mx <- HairEyeColor[, , 1]  # 男性を抽出
630  print(mx)
631
632  mosaicplot(mx,
633             main = "",
634             xlab = "髪の色 (Hair)",
635             ylab = "目の色 (Eye)",
636             las = 2,
637             color = TRUE)
```

（my_base_graphics3.R：628-637）

(b) モザイク図



標準関数によるグラフ作成 (13)(14)

●標準関数：(13) スパインプロット、(14)スピノグラム

モザイク図

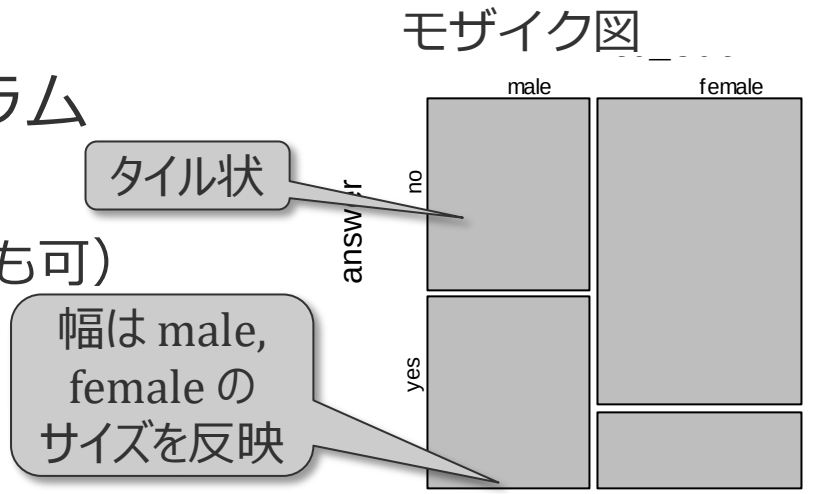
データを再帰的に分割したタイル状のグラフ（3 変数以上も可）

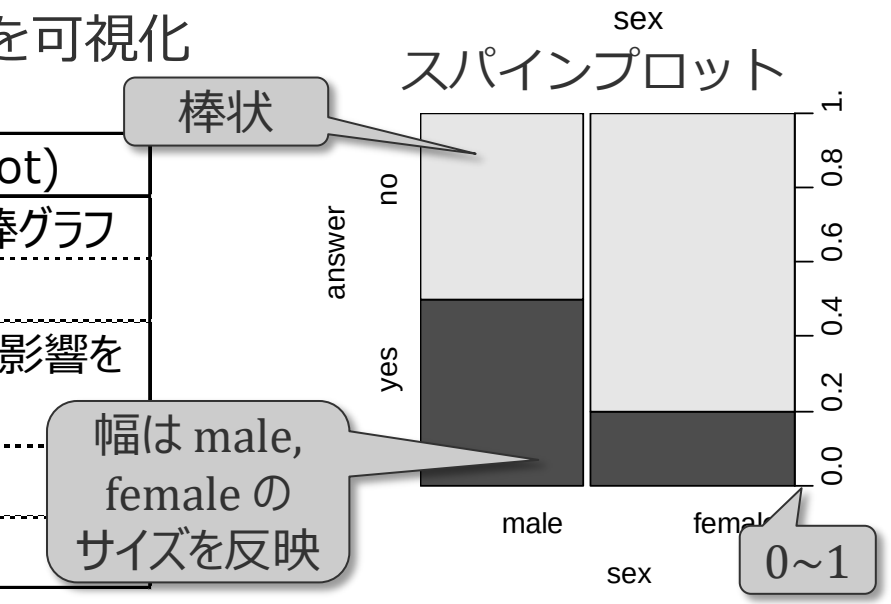
変数間の相互関係や独立性、全体的な分布構造を可視化

スパインスプロット

幅を変えた 100% 積み上げ棒グラフ（2 変数）

x のグループによって y の割合がどう変わるか、因果関係を可視化





	モザイク図 (mosaic plot)	スパインプロット (spine plot)
外観	データを再帰的に分割したタイル状	幅を変えた 100% 積み上げ棒グラフ
変数の数	3変数以上も表現可能	基本的に 2変数（y ~ x）
主な用途	変数間の相互関係や独立性を見たいとき（相関的）	説明変数が応答変数に与える影響を見たいとき（回帰的）
縦軸	カテゴリの分割、目盛りは必須ではない	条件付き確率 (0 ～ 1)
連続変数	事前にカテゴリ化が必要	対応可 (→ スピノグラム)

標準関数によるグラフ作成 (13)(14)

●標準関数：(13) スパインプロット、(14)スピノグラム

スパインプロット

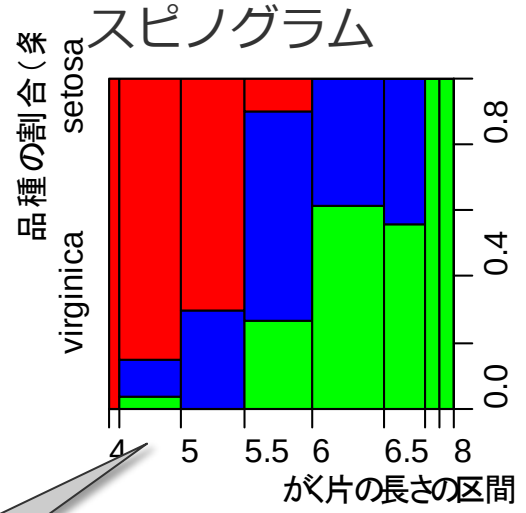
幅を変えた 100% 積み上げ棒グラフ

x のグループによって y の割合がどう変わるか、因果関係を可視化

スピノグラム

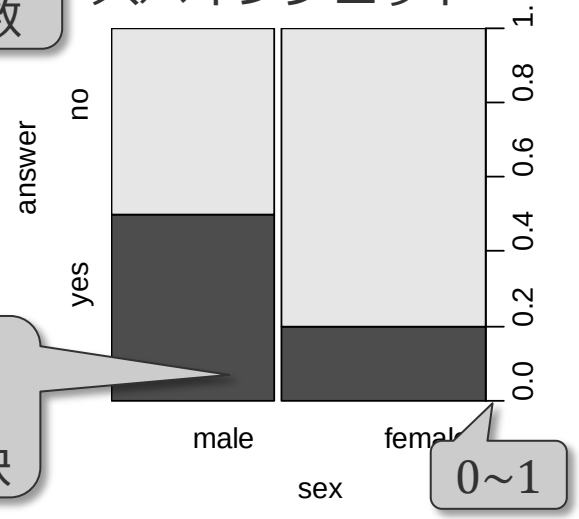
スパインプロットで、x 軸に連続的な数値変数をビニングして割付け

区分け



ビニングした数値変数

	スピノグラム（sinogram）	スパインプロット (spine plot)
目的	連続的な説明変数 (x) が 応答変数 (y) の割合に与える影響を 視覚的に把握	離散的な説明変数 (X) が 応答変数 (Y) の割合に与える影響を 視覚的に把握
横軸 (x)	連続的な数値変数を、 区間に区切って割当（ビニング）	離散的な因子変数を割当
縦軸 (y)	条件付き確率 (0 ～ 1)	条件付き確率 (0 ～ 1)
幅	各区間（ビン）に属するデータの度数に 比例する幅を持つ	各カテゴリに属するデータの度数に 比例する幅を持つ



幅は male, female のサイズを反映

0~1

標準関数によるグラフ作成 (13)

●標準関数：(13) スパインプロット（Spine Plot）、spineplot()、plot()

(a) 試験区（対照(ctrl)/処理(trt)）の反応（yes/no）の比較

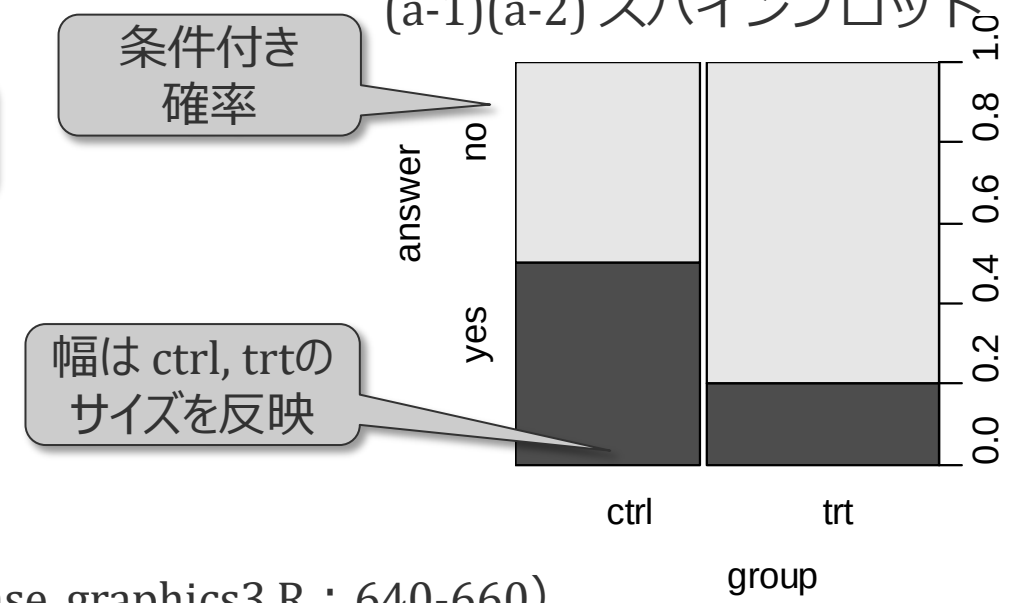
（因果関係の解析）

```
640 # (13) スパインプロット(Spine Plot)
641 #      spineplot()、plot() --------------------
642
643 ## (a) 処理群(説明変数)と反応(応答変数)の因果関係
644 group <- factor( c(rep("ctrl", 4), rep("trt", 5)),
645                  levels = c("ctrl", "trt"))
646 answer <- factor(
647   c("no",  "no", "yes", "yes",
648     "no",  "no", "yes",  "no",  "no"),
649   levels = c("no", "yes"))
650
651 tb_out <- table(group, answer)  #因子の順序に注意
652 print(tb_out)
653
654 ### (a-1) group を横軸、answer を縦軸に配置
655 plot(group, answer)
656 plot(answer ~ group)
657
658 ### (a-2) group を横軸、answer を縦軸に配置
659 spineplot(answer ~ group)
660 spineplot(tb_out)
```

テーブル
オブジェクト

```
> print(tb_out)
       answer
group   no yes
  ctrl   2   2
  trt    4   1
```

(a-1)(a-2) スパインプロット



（my_base_graphics3.R：640-660）

標準関数によるグラフ作成 (14)

●標準関数：(14) スピノグラム（Spirogram）、spineplot()

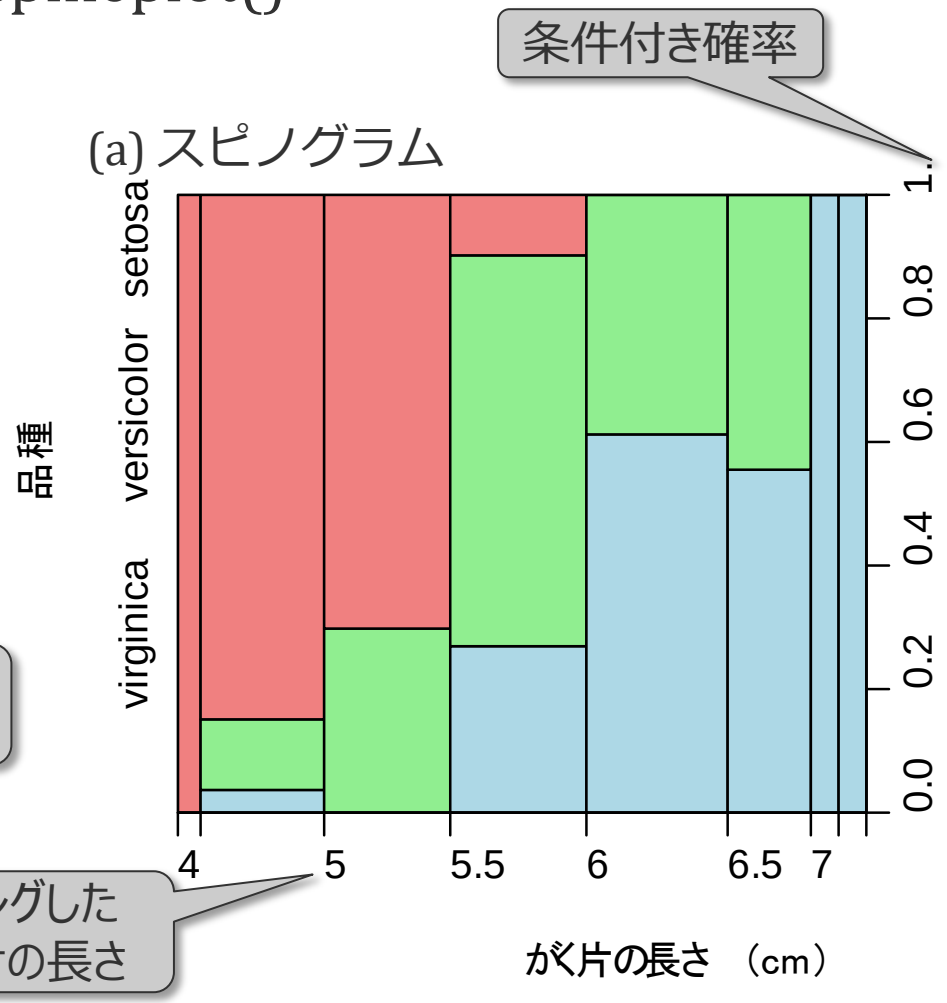
(a) iris：がく片の長さ（Sepal.Length, 連続変数）から
品種（Species, カテゴリ変数）をどの程度の
確率で推定できるか

がく片の長さを自動的にビニングしてプロット

```
663  # (14) スピノグラム(Spinogram) 、spineplot() ---
664
665  ## (a) iris：説明変数応答変数の因果関係
666  spineplot(
667    Species ~ Sepal.Length, data = iris,
668    xlab = "がく片の長さ (cm)",
669    ylab = "品種",
670    col = c("lightblue", "lightgreen", "lightcoral"))
```

（my_base_graphics3.R：663-670）

因子ベクトル
～数値ベクトル



ビニングした
がく片の長さ

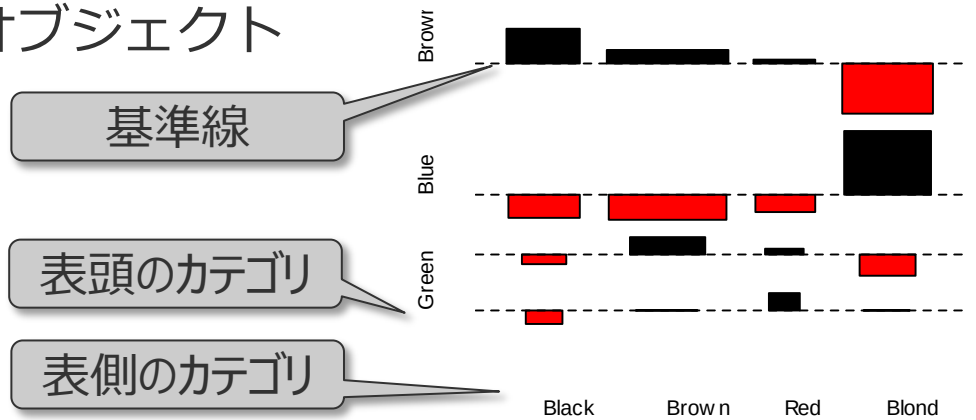
標準関数によるグラフ作成 (15)

●標準関数：(15) Cohen-Friendly の連関図（Association Plot）、assocplot()

２次元の分割表の可視化（２次元以上のデータは margin.table() などを使って２次元に集約）

分割表における観測度数と期待度数の乖離を視覚化、アソシエーション分析

使い方　assocplot(x) ・・・x：マトリックス、テーブルオブジェクト



assocplot() の主な引数

space：長方形の間の距離、既定値 0.3

（長方形の平均幅と高さの割合）

棒の高さ：ピアソン残差（Pearson residuals）

基準線より上に位置するほど観測度数が期待度数より多いことを示す（正の連関）

基準線に近いほど観測度数と期待度数が近いことを示す（独立性が高い）

基準線より下に位置するほど観測度数が期待度数より少ないことを示す（負の連関）

棒の幅　：期待度数の平方根に比例、幅が広い棒ほとその組み合わせの期待度数が大きい

棒の面積：観測度数と期待度数の単純な差（乖離の実数）

標準関数によるグラフ作成 (15)

●標準関数：(15) Cohen-Friendly の連関図（Association Plot）、assocplot()

(a) HairEyeColor： 女性 313 人の髪の色と目の色の関係、
相互関係や独立性を可視化

```
> print(mx15)
       Eye
Hair    Brown Blue Hazel Green
  Black    36    9     5     2
  Brown    66   34    29    14
  Red      16    7     7     7
  Blond     4   64     5     8
```

```
673  # (15) Cohen-Friendlyの連関図(Association Plot)
674  #      assocplot() ----------------------------
675
676  ## (a) HairEyeColor：女性の髪の色と目の色の関係
677  mx15 <- HairEyeColor[,, "Female"]   # 女性を抽出
678  print(mx15)
679
680  # par(cex.axis = 0.7, cex.lab = 0.7)
681  assocplot(mx15,
682            space = 0.3,
683            xlab = "髪の色", ylab = "目の色")
```

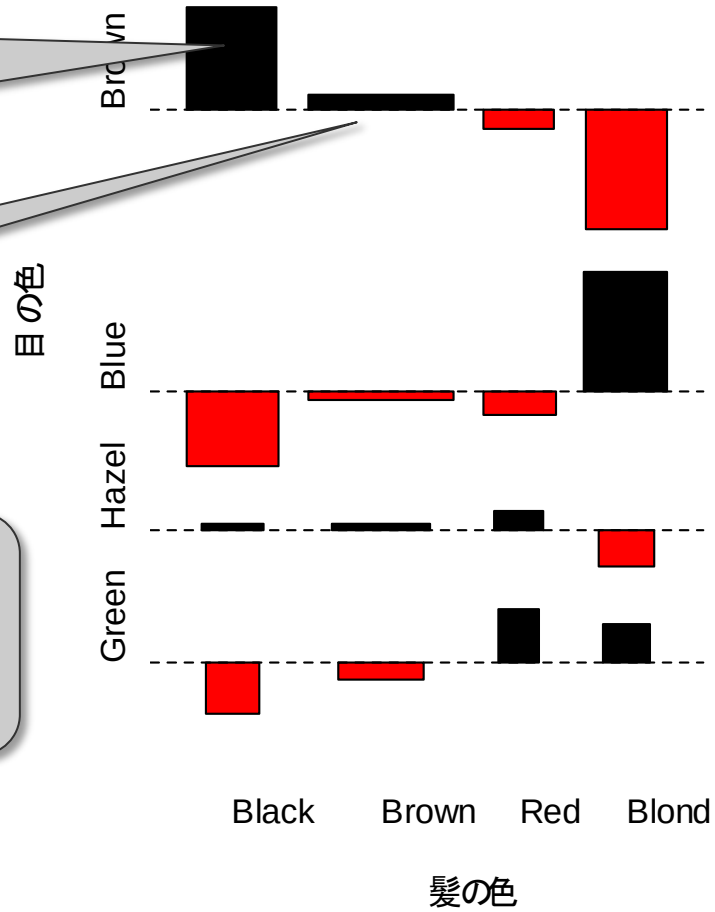
（my_base_graphics3.R：673-683）

黒髪の人は
茶色の目を持つ
傾向が強い

関連性が
小さい

文字を調整する
引数はない
その代わりに
par() を使う

(a) Cohen-Friendly の連関図



標準関数によるグラフ作成 (16)

仮称

●標準関数：(16) ４分表示（Fourfold Display）、fourfoldplot()

2×2の分割表（クロス集計表）を４分円の面積で視覚的に比較、オッズ比の視覚化

使い方 fourfoldplot(x)・・・x：2×2×k の配列(k 個の 2×2 分割表）、 tableオブジェクト（2×2）

関数 fourfoldplot() の主な引数

color：長さ 2 のベクトル、各 2 x 2 分割表の小さい方の対角線と大きい方の対角線に使用する色

conf.level：オッズ比の信頼区間に使用する信頼水準、既定値 0.95、 0 で区間推定なし

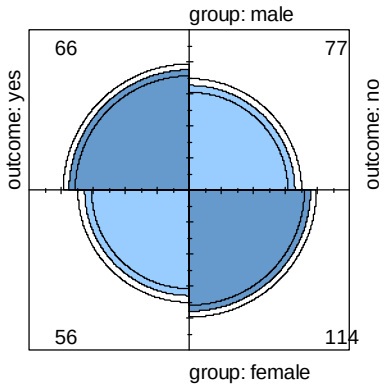
std：標準化の方法を指定、"margins"（既定値）、"ind.max"、"all.max"

margin：等しくする余白を表す数値ベクトル、1, 2, c(1, 2)

space： ４分円の空白、既定値 0.2（0 に近いほど４分円が接近）

mfrow, mfcolb：k 個のプロットを行優先で配置、列優先で配置

４分円



期待値からの偏りを４分円の形から視覚的に表示（k 個の４分円を得る）

４分円が完全な円に近い：２変数はほぼ独立（関連性がほぼない）、オッズ比は１に近い

４分円が非対称：２変数は独立ではない（関連性が強い）、オッズ比は１から離れる

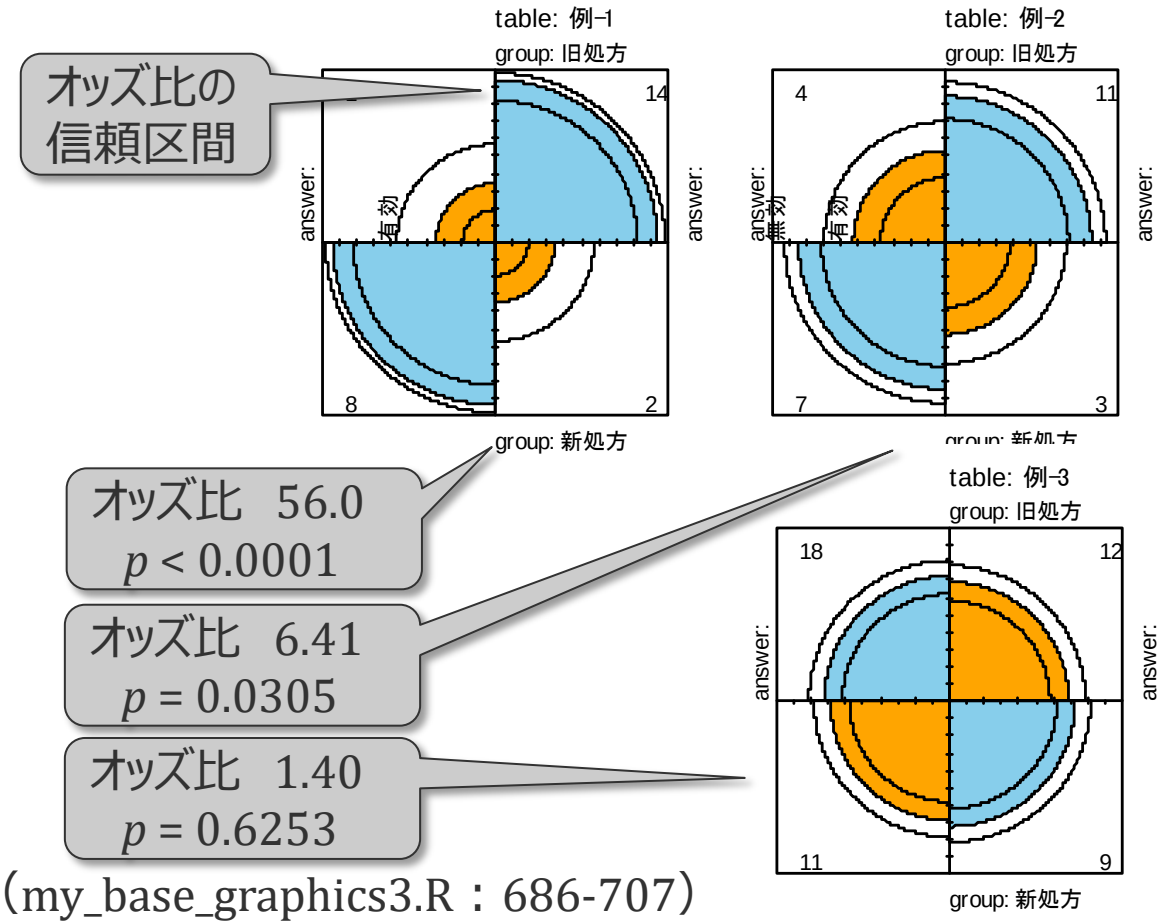
標準関数によるグラフ作成 (16)

●標準関数：(16) ４分表示（Fourfold Display）、fourfoldplot()

(a) ２×２分割表を３つ比較

```
686  # (16)フォーフォールド表示(Fourfold Display)、
687  #     (４分表示)、fourfoldplot() -------------
688
689  ## 3 組の 2x2 分割表を比較
690  mx1 <- matrix(c( 1, 14,
691                   8,  2), ncol = 2, byrow = TRUE)
692  mx2 <- matrix(c( 4, 11,
693                   7,  3), ncol = 2, byrow = TRUE)
694  mx3 <- matrix(c(18, 12,
695                  11,  9), ncol = 2, byrow = TRUE)
696  arr <- array(
697    c(mx1, mx2, mx3),
698    dim = c(2, 2, 3),     # 2行×2列×3個
699    dimnames = list(
700      group  = c("旧処方", "新処方"),      # 行名
701      answer = c("有効", "無効"),          # 列名
702      table  = c("例-1", "例-2", "例-3")  # グループ
703    ))
704
705  fourfoldplot(arr,
706               conf.level = 0.95, mfcol = c(2, 2),
707               color = c("orange", "skyblue"))
```

(a) ４分割プロット



(my_base_graphics3.R：686-707)

標準関数によるグラフ作成 (17)

●標準関数：(17) １標本 Q-Q プロット（Q-Q Plot）、qqnorm()、qqline()

１標本のサンプルデータ y が理論的な分布（正規分布）にどの程度従っているかを可視化

通常、正規分布と比較することから、正規 Q-Q プロット（Normal Q-Q plot）ともいわれる

Q：Quantile（分位数、分位点）、全体をある比率で分割する区切りの値、例えば 0.25 分位点

使い方　qqnorm(y)・・・y：数値ベクトル（無作為抽出された観測値）

qqline(y)　・・・参照線を引く低水準グラフィックス関数

0.25:0.75 に分割する値
パーセント点 percentile と
ほぼ同じ意味　下側 25% 点

関数 qqnorm() の主な引数

plot.it：TRUE/FALSE（グラフを表示／非表示）、分位点を取り出すとき qq_data <- qqplot(),

datax：TRUE/FALSE（x軸がサンプルデータ／x軸が理論値（既定値、標準的））

関数 qqline() の主な引数

第 1 四分位点
第 3 四分位点

distribution：論理分布の分位点関数、既定値 qnorm（正規分布）

probs：長さ２の数値ベクトル、対応する分位点を通る参照線を描画、規定値は c(0.25, 0.75)

qtype：分位点を計算する種類（1～9、規定値 7）　関数 quantile() を参照

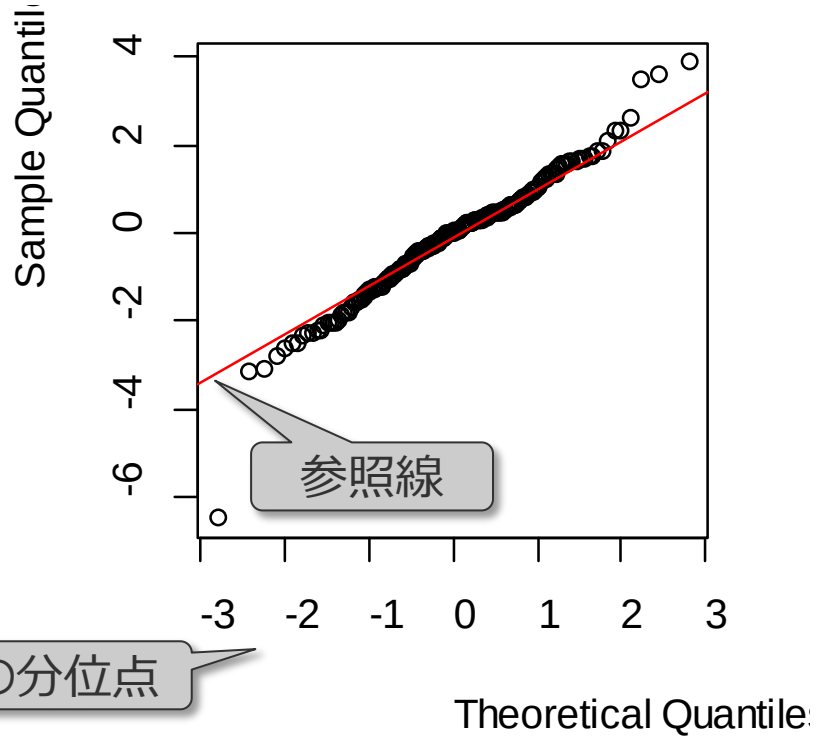
標準関数によるグラフ作成 (17)

●標準関数：(17) １標本 Q-Q プロット（Q-Q Plot）、qqnorm()、qqline()

(a) *t* 分布（自由度 5）に従う乱数と正規分布の比較

Q-Qプロットは、x軸に正規分布の分位点、
y軸にサンプルの分位点を取ったグラフ

(a) １標本 Q-Q プロット



y 軸：サンプルの分位点

x 軸：正規分布の分位点

```
710  # (17) １標本 Q-Q プロット(正規 Q-Q プロット)
711  #      (Q-Q Plot、Normal Q-Q Plot)、qqnorm() ----
712
713  ## (a) t分布の乱数と正規分布を比較するQ-Qプロット
714  y1 <- rt(200, df = 5) # t分布(自由度5)の乱数200個
715
716  qqnorm(y1)              # 正規 Q-Q プロット
717  qqline(y1,
718         col = "red")     # 赤い参照線を追加
```

（my_base_graphics3.R：710-718）

標準関数によるグラフ作成 (17)

●標準関数：(17) １標本 Q-Q プロット（Quantile-Quantile Plot）、qqnorm()

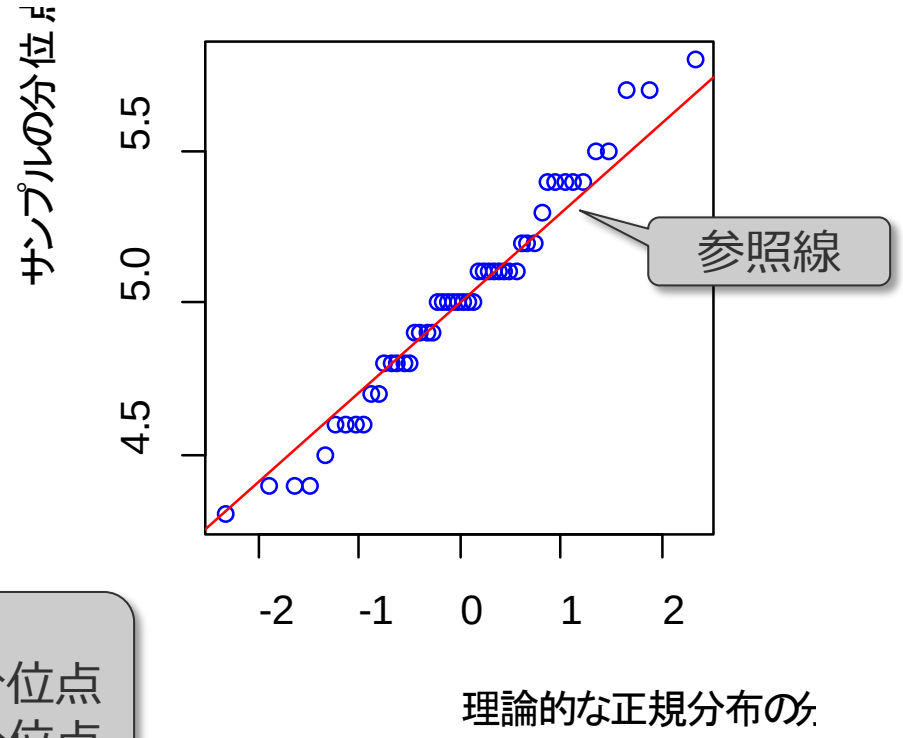
(b) iris：アヤメの品種 "setosa" のがく片の長さの分布と正規分布の比較

```
720  ## (b) iris：setosa のがく片の長さの分布と
721  ##      正規分布を比較する Q-Q プロット
722  y1 <- subset(
723    iris, Species == "setosa")$Sepal.Length
724
725  qqnorm(y1,
726         xlab = "理論的な正規分布の分位点",
727         ylab = "サンプルの分位点",
728         pch = 1,
729         datax = FALSE,    # データを y軸に割当て
730         col = "blue")
731
732  qqline(y1,                          # 参照線を追加
733         prob = c(0.25, 0.75), # 当てはめに使う分位点
734         distribution = qnorm, # 理論分布の関数
735         col = "red")          # ↑ 上記2行は規定値
```

（my_base_graphics3.R：720-735）

既定値
第１四分位点
第３四分位点

(b) １標本 Q-Q プロット



標準関数によるグラフ作成 (18)

●標準関数：(18) ２標本 Q-Q プロット（Quantile-Quantile Plot）、qqplot()

２つの標本分布の形状が似ているかどうかを視覚的に比較

使い方　qqplot(x, y)・・・x、y：数値ベクトル（比較する２群から無作為に抽出した観測値）

このグラフで「2つの分布が（ほぼ）同じ形である」ことを示す参照線は傾き１の直線

（正規分布の理論値ではない）

関数 qqplot() の主な引数

polt.it：TRUE/FALSE（グラフを表示／非表示）、分位点を取り出すとき qq_data <- qqplot()

conf.level：信頼区間の信頼度、たとえば 0.95、規定値は NULL で信頼区間は非表示

conf.args：信頼区間の計算と視覚化を定義する引数のリスト

exact： NULL(既定値)ではサンプルサイズの積が 10000 未満で正確な計算を実施

simulate.p.value：モンテカルロ法で p 値を計算するか否か（既定値 FALSE）

B：モンテカルロ検定で使用する反復回数（既定値 2000）

col、border：信頼区間の塗りつぶしの色と境界線の色（既定値 NA, NULL）

標準関数によるグラフ作成 (18)

●標準関数：(18) ２標本 Q-Q プロット（Quantile-Quantile Plot）、qqplot()

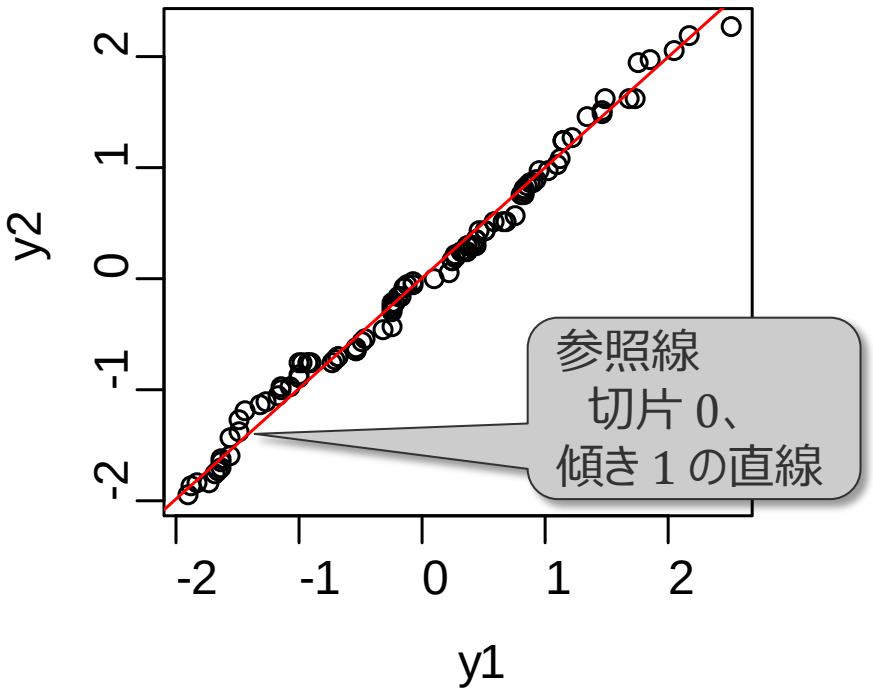
(a) ２つの標準正規乱数をそれぞれ 100個発生させて分布を比較

参照線：y=x の直線

```
738  # (18) 2標本 Q-Qプロット(Quantile-Quantile Plot)
739  #      qqplot() ------------------------------
740
741  ## (a) ２つの正規乱数の分布を比較するQ-Qプロット
742  y1 <- rnorm(100)
743  y2 <- rnorm(100)
744
745  qqplot(y1, y2)
746  abline(a = 0, b = 1,  # 参照線、a：切片、b：傾き
747         col = "red")
```

（my_base_graphics3.R：738-747）

(a) ２標本 Q-Q プロット



標準関数によるグラフ作成 (18)

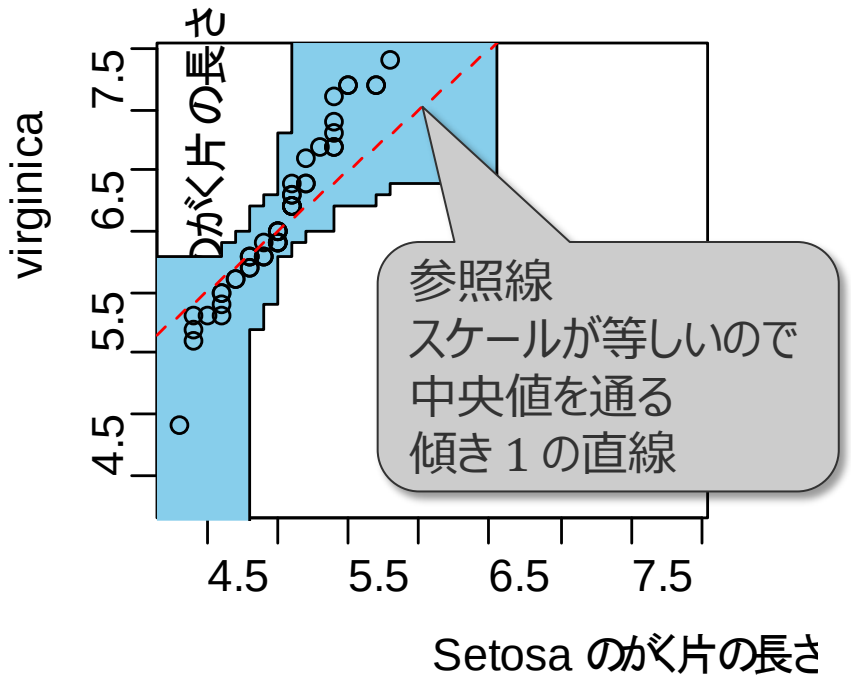
●標準関数：(18) ２標本 Q-Q プロット（Quantile-Quantile Plot）、qqplot()

(b) iris：２品種のがく片の長さの分布を Q-Q プロットで比較、参照線：中央値を通る傾き１の線

```
749  ## (b) iris：２品種のがく片の長さの分布を比較する
750  ##            Q-Q プロット
751  y1 <- iris$Sepal.Length[iris$Species=="setosa"]
752  y2 <- iris$Sepal.Length[iris$Species=="virginica"]
753  var_max <- max(max(y1), max(y2)) # 2変量の最大値
754  var_min <- min(min(y1), min(y2)) # 2変量の最小値
755
756  qqplot(
757    y1, y2,
758    xlim = c(var_min, var_max),
759    ylim = c(var_min, var_max),
760    xlab = "Setosa のがく片の長さ",
761    ylab = "virginica のがく片の長さ",
762    conf.level = 0.95,
763    conf.args = list(
764      exact = NULL,                  # 既定値
765      simulate.p.value = FALSE,   # 既定値
766      col = "skyblue", border = "black"))
767
768  abline(a = median(y2) - median(y1),
769         b = 1,
770         col = "red", lty = 2)    # 参照線
```

参照線
中央値を通る
傾き１の直線

(b) ２標本 Q-Q プロット



（my_base_graphics3.R：749-770）

標準関数によるグラフ作成 (18)

●標準関数：(18) ２標本 Q-Q プロット（Quantile-Quantile Plot）、qqplot()

(c) 正規乱数と標準正規乱数の分布を Q-Q プロットで比較

正規乱数：平均 2、標準偏差 3 の正規分布に従う乱数

標準正規乱数：平均 0、標準偏差 1 の正規分布に従う乱数

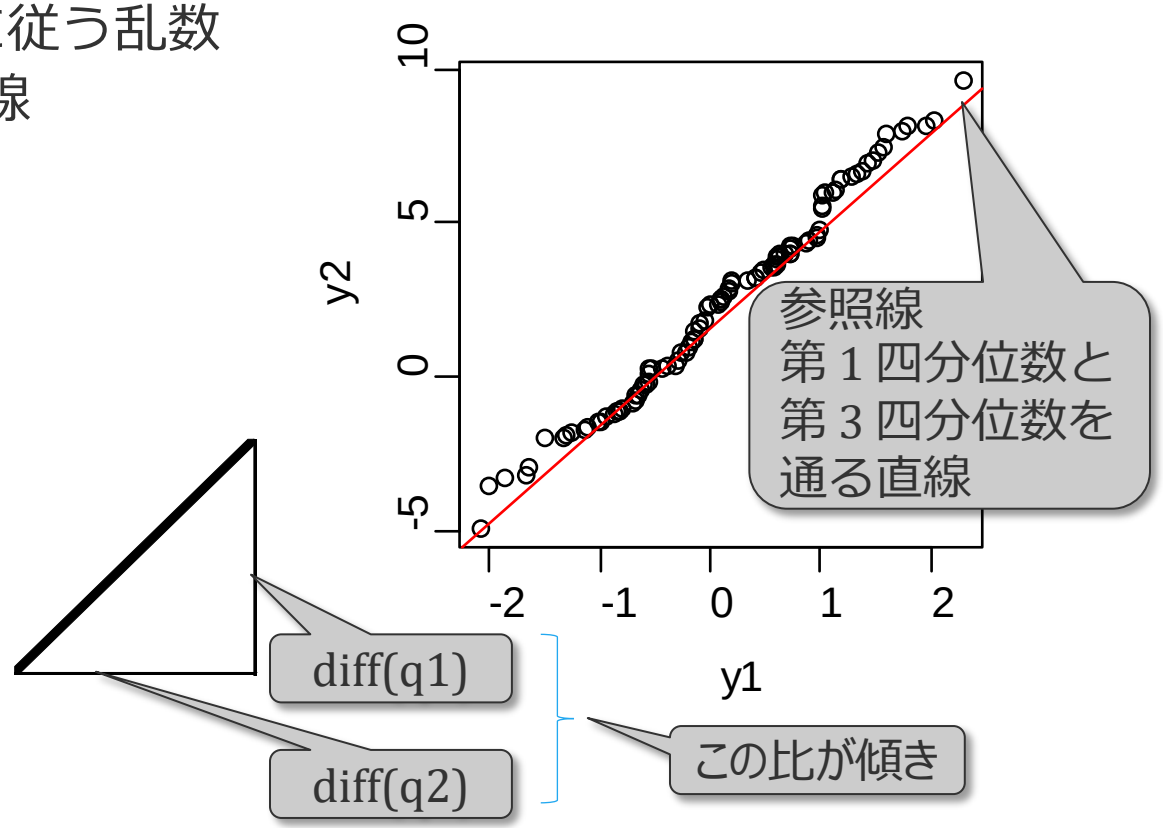
参照線：第 1 四分位数と 第 3 四分位数を通る直線

（一般的な参照線）

```
772  ## (c) 正規乱数と標準正規乱数を比較するQ-Qプロット
773  y1 <- rnorm(100)
774  y2 <- rnorm(100, mean = 2, sd = 3)
775
776  qqplot(y1, y2)
777
778  q1 <- quantile(y1, probs = c(0.25, 0.75))
779  q2 <- quantile(y2, probs = c(0.25, 0.75))
780  slope <- diff(q2) / diff(q1)
781  intercept <- q2[1] - slope * q1[1]
782  abline(a = intercept,
783         b = slope,      # 第1四分位数と第3四分位数
784         col = "red")    # を通る参照線
```

（my_base_graphics3.R：772-784）

(c) ２標本 Q-Q プロット



標準関数によるグラフ作成 (19)

●標準関数：(19) 等高線プロット（Contour Plot）、contour()

３次元データ（x, y, z）を２次元（x, y）の平面に投影し、同じ z の値を線で結んで可視化
関数の変化や地形、気象データなどを視覚化、空間データの変化を直感的に把握
z = f(x, y) の格子状データ（地形や密度など）を直感的に把握

使い方 contour(x, y, z) ・・・ x 軸とy 軸の座標（数値ベクトル、昇順）、
z はマトリックス（行列）、x, y を省略して z を x として渡すことも可

関数 contour() の主な引数

levels：等高線を描くzの値（既定値は最小値から最大値まで nlevesls の数だけ均等に分割）、
c(0.01, 0.1, 0.5, 1) と指定することで、特定の z の値の等高線を描画
nlevels：levels が指定されていない場合、描く等高線の数、既定値は 10
col：等高線の色、例えば topo.colors(10) を渡すと 10 段階のグラデーションの指定が可能
asp：x 軸と y 軸の比を指定、asp = 1 で x 軸と y 軸のスケールを同じにする
（asp 引数を省略すると、自動的にプロット領域に合わせて軸の長さを調整）

標準関数によるグラフ作成 (19)

●標準関数：(19) 等高線プロット（Contour Plot）

(a) volcano：火山の標高データを等高線で可視化

マトリックスの各要素は標高値

行と列は東西南北の座標（10 m 間隔）

行と列のインデックスを 10 倍して10 m 間隔に変換

列インデックス

行インデックス

要素

```
> head(volcano)
     [,1] [,2] [,3] [,4] [,5] [,6
[1,]  100  100  101  101  101  10
[2,]  101  101  102  102  102  10
[3,]  102  102  103  103  103  10
[4,]  103  103  104  104  104  10
[5,]  104  104  105  105  105  10
```

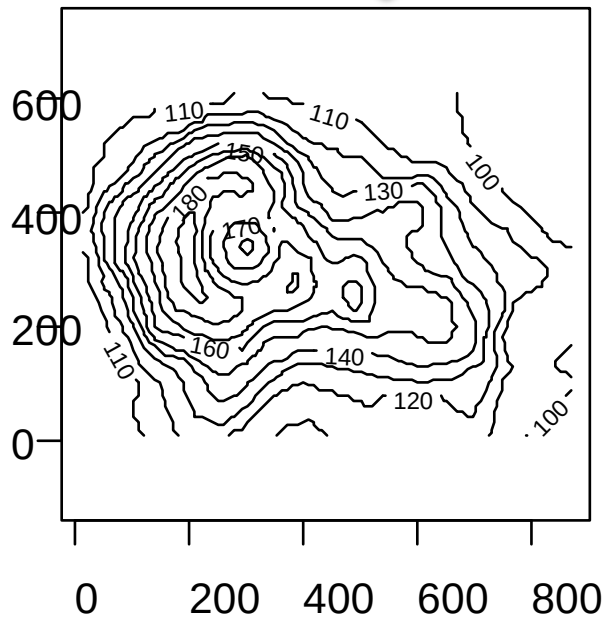
```
786  # (19) 等高線プロット(Contour Plot)、contour() -
787
788  ## (a) volcano：火山の 10m×10m の標高値から作成
789  head(volcano)          # データを一部表示
790
791  z <- volcano           # 等高線のマトリックス(行列)
792  x <- 10 * (1:nrow(z)) # 行番号を10 mに変換 (S～N)
793  y <- 10 * (1:ncol(z)) # 列番号を10 mに変換 (E～W)
794
795  contour(x, y, z,       # 位置引数
796          asp = 1,       # x軸とy軸を同じスケール
797          las = 1)       # 目盛ラベルを水平に表示
798
799  contour(volcano)       # マトリックスをそのまま描画
```

（my_base_graphics3.R：786-799）

引数 las =1で
水平に表示

x, y を省略
x に volcano を
渡す

(a) 等高線プロット



標準関数によるグラフ作成 (19)

●標準関数：(19) 等高線プロット（Contour Plot）、contour()

(b) ２次元正規分布（$\mu_x = 0,\ \mu_y = 0,\ \rho = 0.5$）の等高線

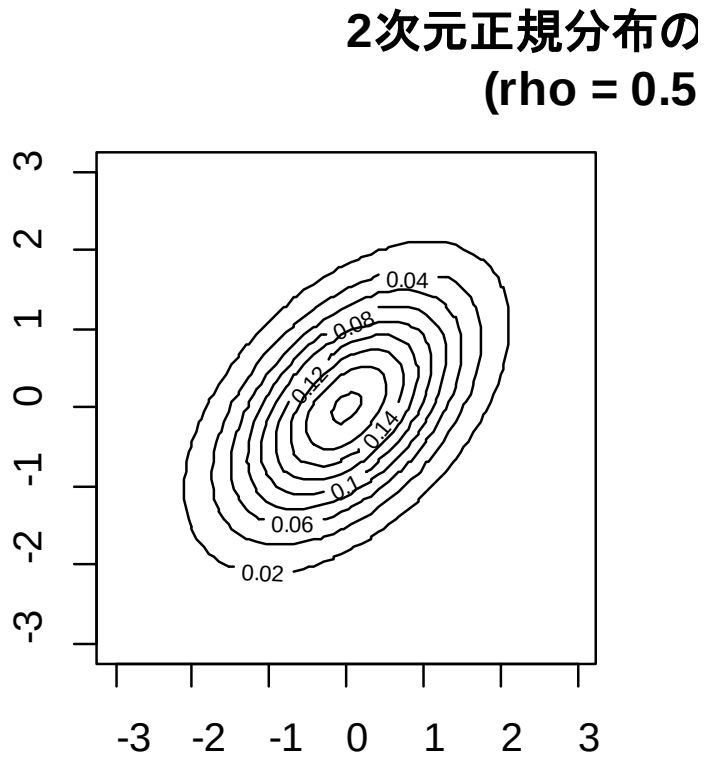
```
801  ## (b) ２次元正規分布の等高線
802  r  <- 0.5                              # 母相関係数
803  xx <- seq(-3, 3, length.out = 50)  # x軸の値 50個
804  yy <- seq(-3, 3, length.out = 50)  # y軸の値 50個
805
806  ### 2次元正規分布の確率密度関数 (PDF) を定義
807  f_pdf <- function(x, y, rho = 0.5) {
808    exponent <- -(x^2 - 2 * rho * x * y + y^2) /
809                  (2 * (1 - rho^2))
810    constant <- 1 / (2 * pi * sqrt(1 - rho^2))
811    return(constant * exp(exponent))
812  }
813
814  ### z の計算と等高線の描画、z軸の値(50×50 個)
815  zz <- outer(
816    xx, yy, FUN = f_pdf, rho = r)
817
818  contour(xx, yy, zz,
819          asp = 1,
820          main = paste0("2次元正規分布の等高線 \n
821                        (rho = ", r,")"))
```

既定値
rho = 0.5

２つの数値ベクトルから
指定した 関数により計算

改行

(b) 等高線プロット



（my_base_graphics3.R：801-821）

標準関数によるグラフ作成 (20)

●標準関数：(20) ３次元透視図（3D Perspective Diagram）、persp()

３次元データ（x, y, z）を立体的に可視化、3D 曲面プロットとほぼ同じ

z = f(x, y) の格子状データ（地形や密度など）を直感的に把握（3D perspective view）

使い方　pesp(x, y, z)・・・x 軸とy 軸の座標（数値ベクトル、昇順）、z はマトリックス（行列）

関数 persp() の主な引数

theta：視点の方位角（横方向の回転角度、単位は度）、規定値は 0

phi：視点の仰角（縦方向の角度、単位は度）、規定値 15

expand：z 軸方向の拡大・縮小率を指定。1 より小さい値で高さを抑えられる

col：面の色を指定。1 色または z の値に応じてベクトルで与えることも可

border：面の境界線の色を指定（NA で線を描かない）

ticktype：目盛りの種類を "simple"（簡易）または "detailed"（詳細）から選択

shade：陰影の濃さを 0～1 の範囲で指定。陰影によって立体感を強調

ltheta, lphi：光源の方向を方位角・仰角で指定、shade と組み合わせて使用

標準関数によるグラフ作成 (20)

●標準関数：(20) ３次元透視図（3D Perspective Diagram）、persp()

(a) volcano：火山の標高データ（マトリックス）を立体的に可視化
　行と列のインデックスを 10 倍して 10 m 間隔に変換

```
824  # (20) ３次元透視図(3D Perspective Diagram)
825  #     persp() ---------------------------------
826
827  ## (a) volcano：火山の10m×10mのグリッド上の地形情報
828  head(volcano)
829  z <- volcano              # 標高値を付値
830  x <- 10 * (1:nrow(z))     # 行番号を10 mに補正(S～N)
831  y <- 10 * (1:ncol(z))     # 列番号を10 mに補正(E～W)
832
833  persp(
834    x, y, z,
835    theta = 135,        # 横方向の回転角度(方位角)
836    phi = 30,           # 縦方向の回転角度(仰角)
837    shade = 0.4,        # 陰影の強さ(3D感を出す)
838    ticktype = "simple",  # 目盛ラベル非表示
839    border = NA,          # 面を囲む線の色
840    box = FALSE,          # 枠線を非表示
841    col = "lightgreen")
```

（my_base_graphics3.R：824-841）

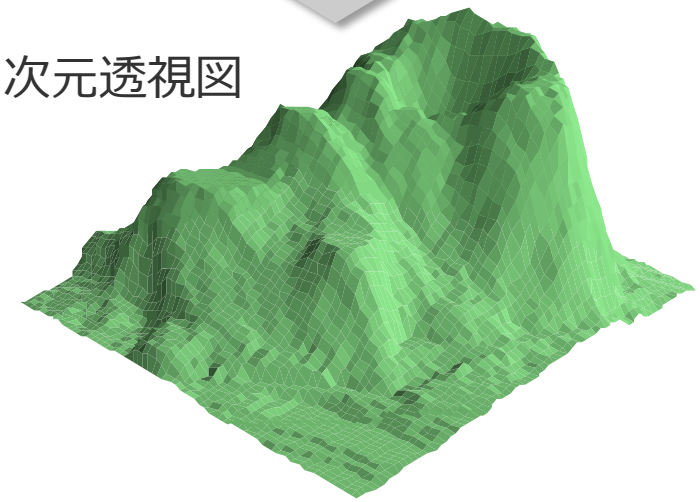
```
> head(volcano)
     [,1] [,2] [,3] [,4] [,5] [,6
[1,]  100  100  101  101  101  10
[2,]  101  101  102  102  102  10
[3,]  102  102  103  103  103  10
[4,]  103  103  104  104  104  10
[5,]  104  104  105  105  105  10
```

列インデックス

行インデックス

要素

(a) ３次元透視図



標準関数によるグラフ作成 (20)

●標準関数：(20) ３次元透視図（3D Perspective Diagram）、persp()

(b) ２次元正規分布を立体的に表現

$\mu_x = 0,\ \mu_y = 0,\ \rho = 0.5$ の２次元正規分布

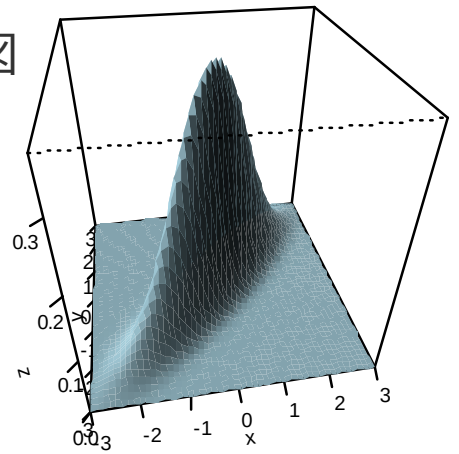
```r
## (b) ２次元正規分布の3D曲面プロット
r  <- 0.9                              # 母相関係数
xx <- seq(-3, 3, length.out = 50)  # x軸の範囲
yy <- seq(-3, 3, length.out = 50)  # y軸の範囲

### 2次元正規分布の確率密度関数 (PDF) を定義
f_pdf <- function(x, y, rho = 0.5) {
  z1 <- x; z2 <- y    # 標準化変数
  exponent <- -(z1^2 - 2 * rho * z1 * z2 + z2^2) /
              (2 * (1 - rho^2))
  constant <- 1 / (2 * pi * sqrt(1 - rho^2))
  return(constant * exp(exponent))
}

zz <- outer(xx, yy, FUN = f_pdf, rho = r)
```

（my_base_graphics3.R：843-870）

既定値
rho = 0.5

```r
persp(
  xx, yy, zz,
  main = paste0("2次元正規分布 \n (rho = ", r,")"),
  theta = -10,              # z軸周りの回転角度(方位角)
  phi = 35,                 # x軸周りの回転角度(仰角)
  expand = 1,               # z軸方向の伸縮率
  col = "lightblue",        # 表面の色
  shade = 0.4,              # 陰影の強さ（3D感を出す）
  ticktype = "detailed", # 軸の目盛りを非表示、矢印
  border = NA,              # メッシュの線を消す
  cex.axis = 0.5,           # 目盛ラベルの文字サイズ
  cex.lab = 0.5)            # 軸ラベルの文字サイズ
```

(b) ３次元透視図



標準関数によるグラフ作成 (21)

●標準関数：(21) ヒートマップ（Heat Map）、image()

数値マトリックス（行列）を 色の濃淡やグラデーションで可視化する二次元プロット

z = f(x, y) のような格子状データの z のパターンを直感的に把握

使い方　image(x, y, z, ...)・・・x 軸とy 軸の座標（数値ベクトル、昇順）、z は数値マトリックス

関数 image() の主な引数

col：色指定、規定値 hcl.colors(12, "YlOrRd", rev = TRUE)、errain.colors(20)、heat.colors(10) 等

breaks：z の値を分類する区切り点、col よりも 1 つ多い数の値が必要

length(breaks) = length(col) + 1

xlim, ylim, zlim：x軸、y軸、z軸の範囲

axes：TRUE/FALSE（軸の表示(規定値)／非表示）

asp：アスペクト比、asp = 1 で、x 軸と y 軸の単位当たりの長さ（スケール）が等しくなる

標準関数によるグラフ作成 (21)

●標準関数：(21) ヒートマップ（Heat Map）、image()

(a) volcano：火山の標高データをヒートマップで可視化

マトリックスの各要素は標高値

行と列のインデックスを 10 倍して 10 m 間隔に変換

```
873 # (21) ヒートマップ(Heat Map)、image() ---------
874
875 ## (a) volcano：火山の等高線情報のヒートマップ
876 head(volcano)
877
878 z <- volcano          # 等高線のマトリックス(行列)
879 x <- 10 * (1:nrow(z)) # 行番号を10 mに変換
880 y <- 10 * (1:ncol(z)) # 列番号を10 mに変換
881
882 image(x, y, z,
883       col = terrain.colors(100),
884       asp = 1,         # 地形を正しく表示するため
885       las = 1,
886       xlab = "X (m)", ylab = "Y (m)")
887
888 contour(x, y, z, add = TRUE) # 等高線を追加
```

(my_base_graphics3.R：873-888)

列インデックス

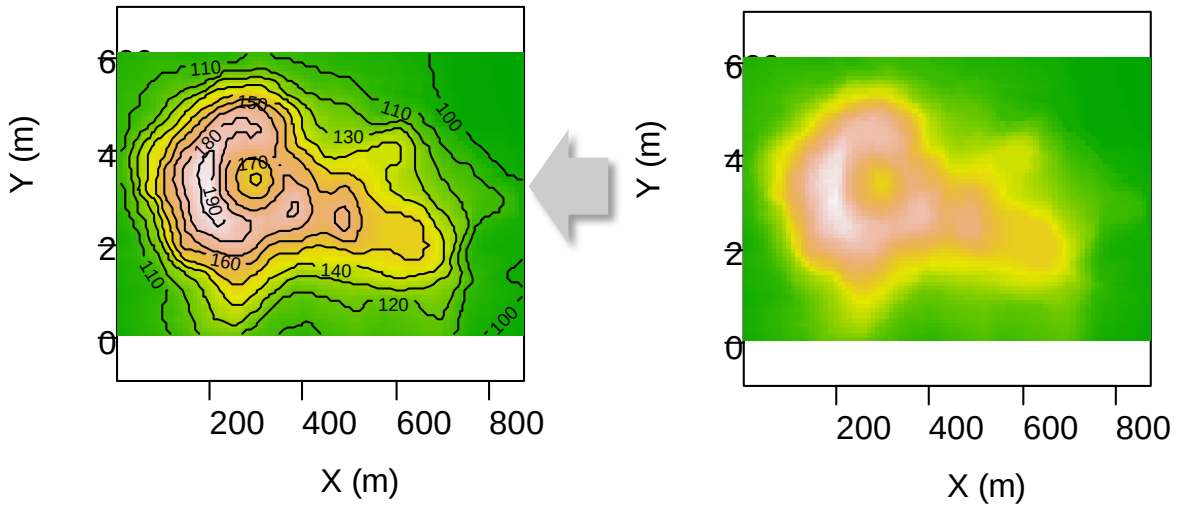
```
> head(volcano)
     [,1] [,2] [,3] [,4] [,5] [,6
[1,]  100  100  101  101  101  10
[2,]  101  101  102  102  102  10
[3,]  102  102  103  103  103  10
[4,]  103  103  104  104  104  10
[5,]  104  104  105  105  105  10
```

行インデックス

要素

(a) ヒートマップ

(a) volcano のヒートマップ



標準関数によるグラフ作成 (21)

●標準関数：(21) ヒートマップ（Heat Map）

(b) mtcars：自動車性能の相関行列を色で可視化

```
890  ## (b) mtcars：相関係数行列をヒートマップで可視化
891  mx <- cor(mtcars)                  # 相関係数行列を付値
892  head(mx)                           # マトリックスを表示
893  x <- 1:ncol(mx); y <- 1:nrow(mx)
894
895  image(
896    x, y, mx,
897    xlab = "", ylab = "",            # axisで設定
898    axes = FALSE,                    # 標準の軸を非表示
899    col = colorRampPalette(          # グラデーション
900      c("blue", "white", "red"))(100),
901    zlim = c(-1, 1))                 # z値の範囲指定
902
903  axis(side = 1, at = x,             # x軸目盛ラベル表示
904       labels = colnames(mx),        # 車種を表示
905       cex.axis = 0.8, las = 2)
906  axis(side = 2, at = y,             # y軸目盛ラベル表示
907       labels = rownames(mx),        # 車種を表示
908       cex.axis = 0.8, las = 2)
909  text(expand.grid(x, y),            # x,y の全ての組合せ
910       labels = round(as.vector(mx), 2),
911       cex = 0.4)                    # mxをベクトルに変換
```

11 行×11 列

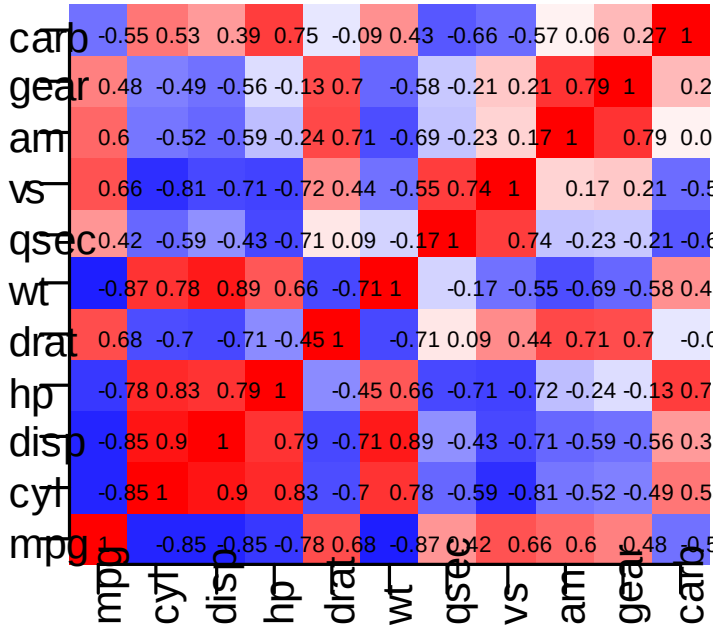
x = 1. 2. ・・・ 10, 11
y = 1, 2, ・・・ 10, 11

x, y
1, 1
2, 1
・・・・
10, 11
11, 11
121 組

相関係数行列
11 行×11 列

```
> head(mx)
            mpg        cyl       disp          h
mpg   1.0000000 -0.8521620 -0.8475514 -0.776168
cyl  -0.8521620  1.0000000  0.9020329  0.832447
disp -0.8475514  0.9020329  1.0000000  0.790948
hp   -0.7761684  0.8324475  0.7909486  1.000000
```

(b) ヒートマップ



(my_base_graphics3.R：890-911)

標準関数によるグラフ作成 (22)

●標準関数：(22) 関数プロット（Function Plot）、plot()、curve()

関数 curve() に式、関数オブジェクトなどの情報を渡して描画

(i) 関数オブジェクト（関数名）を渡す（x の表記がない）

curve(dnorm) ・・・関数名のみを渡す、「dnorm()」ではない

内部では第 1 引数に x を代入して実行

(ii) 式を渡す（式の中に x がある）

curve(x^2)

curve(dnorm(x))

curve(dnorm(x, mean = 1, sd = 2) +1)

curve() は渡された式の中の x を横軸の値（変数）であると認識（必ず「x」を使う）

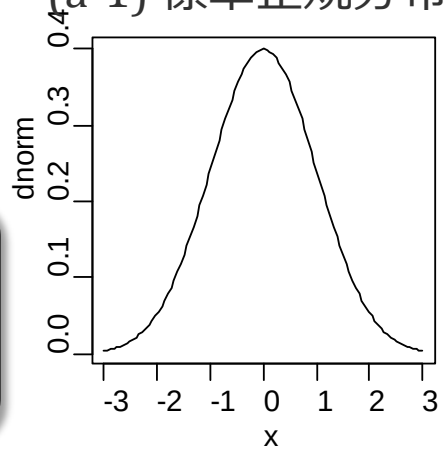
内部での処理（概念的な説明）

```
x <- seq(from, to, length.out = 101)
y <- dnorm(x)
plot(x, y, type = "l")
```

内部での処理（概念的な説明）

```
x <- seq(from, to, length.out = 101)
y <-x^2
plot(x, y, type = "l")
```

(a-1) 標準正規分布



関数 function() で定義した関数、無名関数（ラムダ式）を、オブジェクトまたは関数として使用可

標準関数によるグラフ作成 (22)

●標準関数：(22) 関数プロット（Function Plot）、plot()、curve()

(a) 正規分布の確率密度曲線

(a-1) 標準正規分布を関数 plot() と curve() で作成（関数オブジェクトを渡す）

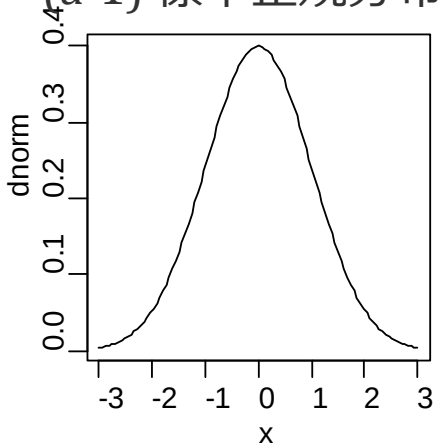
(a-2) 標準正規分布を関数 curve() で作成（式を渡す）

(a-3) 正規分布を関数 curve() で作成（式を渡す）

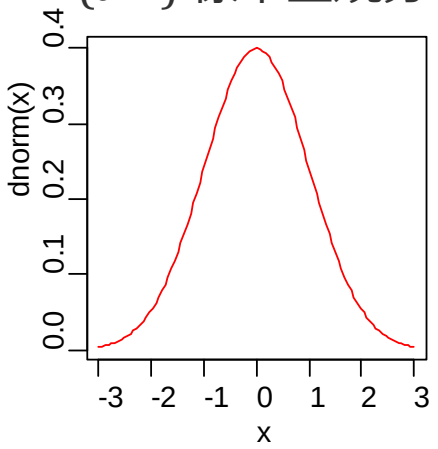
```
914  # (22) 関数プロット(Function plot)、plot()、curve()
915
916  ## (a) 正規分布(Normal Distribution)の確率密度曲線
917  ### (a-1) 標準正規分布を描画、関数オブジェクトを渡す
918  plot(dnorm, xlim = c(-3, 3))
919
920  curve(dnorm, from = -3, to = 3)
921
922  ### (a-2) 標準正規分布を描画、式を渡す
923  curve(dnorm(x), from = -3, to = 3, col = "red")
924
925  ### (a-3) 正規分布を描画、式を渡す
926  curve(dnorm(x, mean = 0, sd = 1), -3, 3,
927        col = "blue", ylab = "確率密度")
```

(my_base_graphics3.R：914-927)

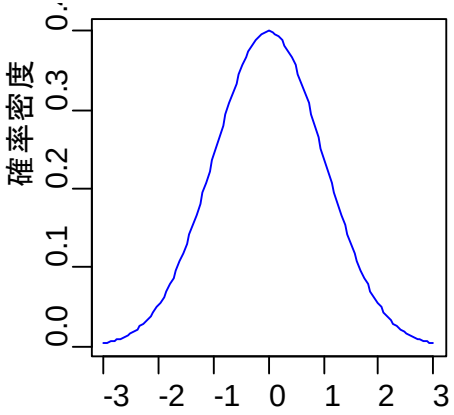
(a-1) 標準正規分布



(a-2) 標準正規分布



(a-3) 正規分布



標準関数によるグラフ作成 (22)

●標準関数：(22) 関数プロット（Function Plot）、plot()、curve()

(a-4) 標準正規乱数のヒストグラムと確率密度曲線（式を渡す）

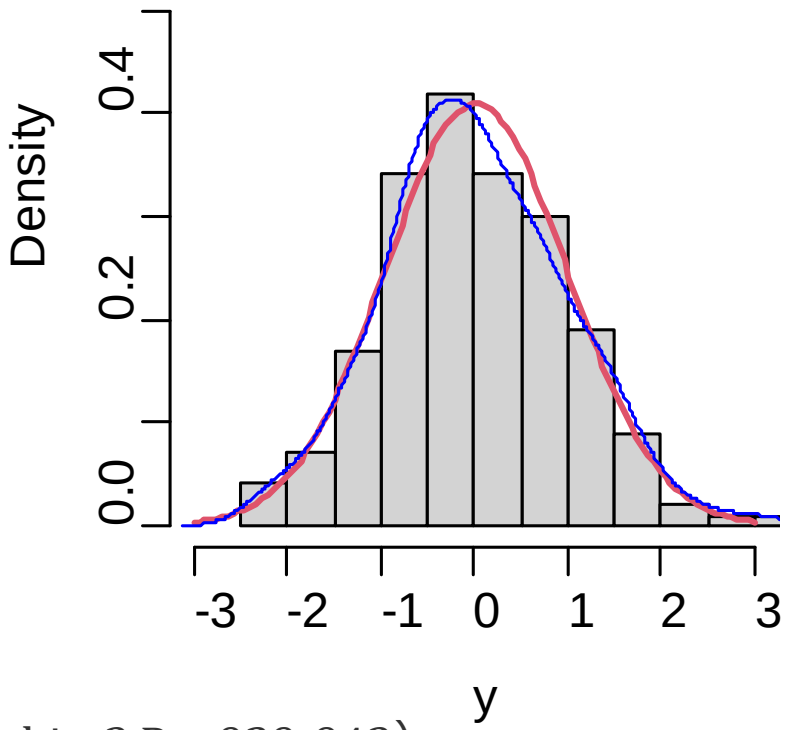
200個の標準正規乱数を発生させて、
そのヒストグラムと平均と標準偏差から推定した
正規分布の確率密度曲線を描画

```
929  ### (a-4) 標準正規乱数のヒストグラムと確率密度曲線
930  y <- rnorm(200)             # 標準正規乱数 200個
931  hist(y,
932       xlim = c(-3, 3),       # x軸の範囲
933       ylim = c(0, 0.5),      # y軸の範囲
934       freq = FALSE)          # 縦軸を確率密度に指定
935
936  m <- mean(y); s <- sd(y)
937
938  curve(dnorm(x, mean = m, sd = s),
939        col = "red",     # 正規分布による近似曲線
940        add = TRUE)
941
942  lines(density(y),      # カーネル密度推定による
943        col = "blue")    # 近似曲線
```

高水準関数で重ね書き

(my_base_graphics3.R：929-943)

(a-4) 標準正規乱数のヒストグラムと正規分布の確率密度曲線



標準関数によるグラフ作成 (22)

●標準関数：(22) 関数プロット（Function Plot）、plot()、curve()

(a-5) 標準正規分布と 95% 信頼区間

95% の領域を囲む座標 (x, y)

	xx					
x	-1.96	-1.96	-1.95	・・・	1.95	1.96	1.96
y	0	0.058	0.060	・・・	0.060	0.058	0
	yy					

縦の線

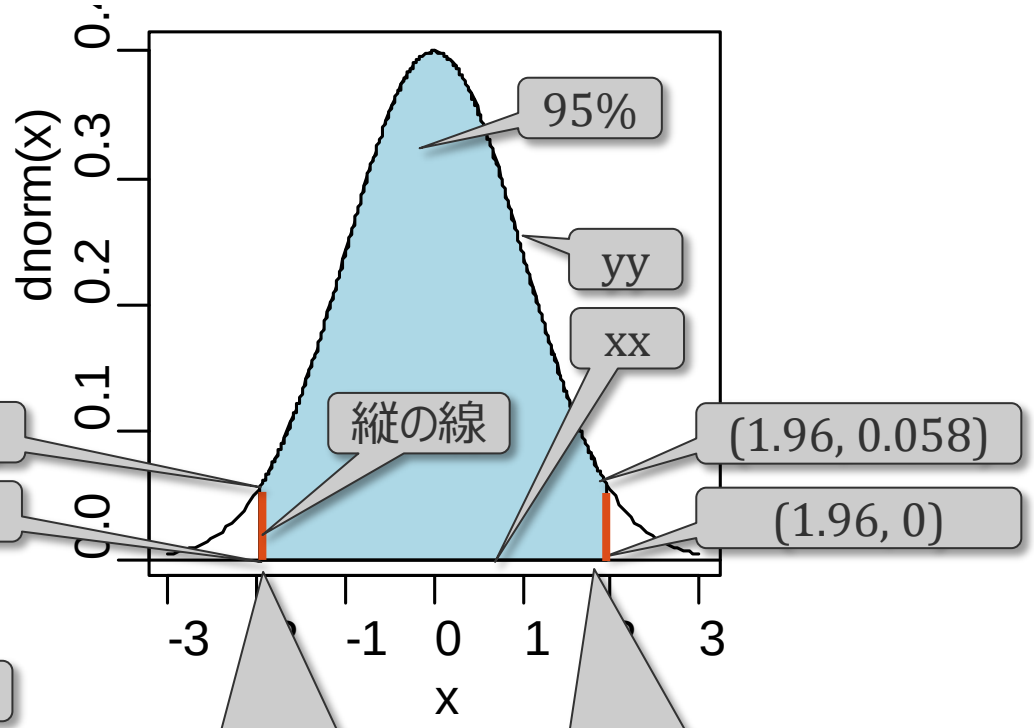
縦の線

```
945  ### (a-5) 標準正規分布と95%信頼区間
946  curve(dnorm, from = -3, to = 3)
947
948  xx <- seq(qnorm(0.025), qnorm(0.975), by = 0.01)
949  yy <- dnorm(xx)
950  x <- c(qnorm(0.025), xx, qnorm(0.975))
951  y <- c(0, yy, 0)
952
953  polygon(x, y, col = "lightblue")  # 95% 部分を着色
954
955  abline(h = 0)                     # 水平線の追加
```

392 個

(my_base_graphics3.R：945-955)

(a-5) 標準正規分布と95% 信頼区間



(-1.96, 0.058)

(-1.96, 0)

(1.96, 0.058)

(1.96, 0)

y = 0

下側 2.5% 点
qnorm(0.025) = −1.96

上側 2.5% 点
qnorm(0.975) =1.96

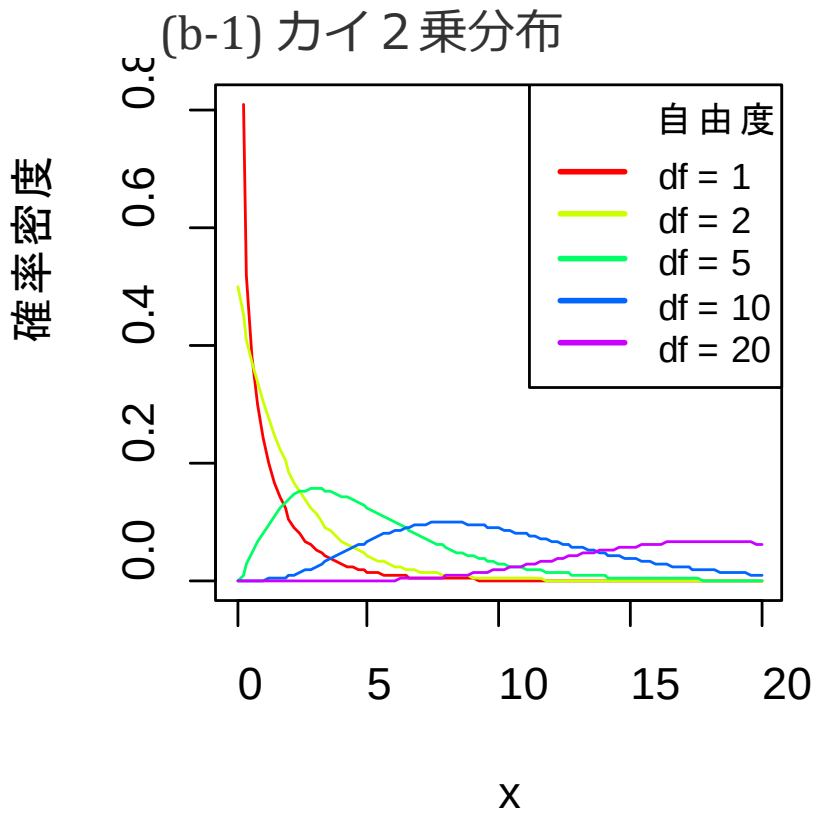
標準関数によるグラフ作成 (22)

●標準関数：(22) 関数プロット（Function Plot）、plot()、curve()

(b-1) カイ２乗分布を関数 curve() で作成（自由度 1～20）

```
957  ## (b) カイ２乗分布(Chi-square Distribution)
958
959  ### (b-1) 自由度を変えたカイ２乗分布
960  df_list  <- c(1, 2, 5, 10, 20)        # 自由度の指定
961  col_list <- rainbow(length(df_list)) # 色の指定
962
963  curve(dchisq(x, df = df_list[1], ncp = 0),
964        from =0, to=20,
965        ylab = "確率密度",
966        col = col_list[1])  # 1番目の自由度の分布
967
968  for (i in 2:5){
969    curve(dchisq(x, df = df_list[i], ncp = 0),
970          col = col_list[i],
971          add = TRUE)}      # 2番目以降の自由度の分布
972
973  legend("topright", title = "自由度",
974         legend = paste("df =", df_list),
975         col = col_list,
976         lwd = 2, cex = 0.8)
```

（my_base_graphics3.R：957-976）



標準関数によるグラフ作成 (22)

●標準関数：(22) 関数プロット（Function Plot）

(b-2) カイ２乗分布における両側確率 5% （片側 2.5%）の範囲を表示（自由度 8 の場合）

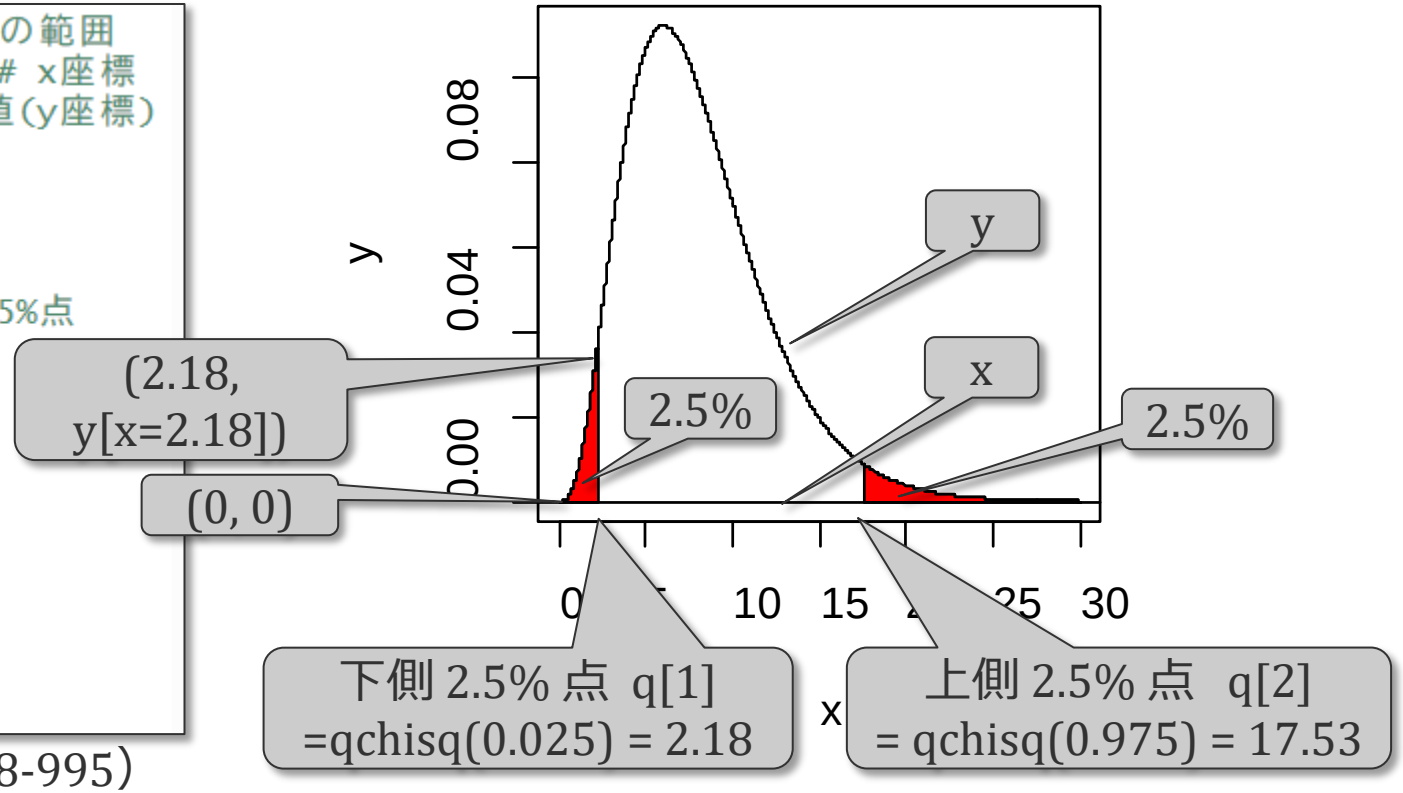
```
978  ## (b-2) カイ２乗分布の両側確率5%(片側2.5%)の範囲
979  x <- seq(from = 0, to = 30, by = 0.01)      # x座標
980  y <- dchisq(x, df = 8) # xに対するカイ２乗値(y座標)
981
982  plot(x, y, type = "l")  # 確率密度曲線
983
984  q <- qchisq(p = c(0.025, 0.975), df = 8)
985  print(q)   # q[1]：下側2.5%点、q[2]：上側2.5%点
986
987  polygon(x = c(0, x[x <= q[1]], q[1]),
988          y = c(0, y[x <= q[1]], 0),
989          col = "red")     # 下側2.5% の範囲
990
991  polygon(x = c(q[2], x[x >= q[2]],max(x)),
992          y = c(0, y[x >= q[2]], 0),
993          col = "red")     # 上側2.5%の範囲
994
995  abline(h = 0)            # 水平線の追加
```

（my_base_graphics3.R：978-995）

```
> print(q)
[1]  2.179731 17.534546
```

q[1], q[2]

(b-2) カイ２乗分布の確率密度曲線



標準関数によるグラフ作成 (22)

●標準関数：(22) 関数プロット（Function Plot） 、plot()、curve()

(c) *t* 分布の確率密度曲線

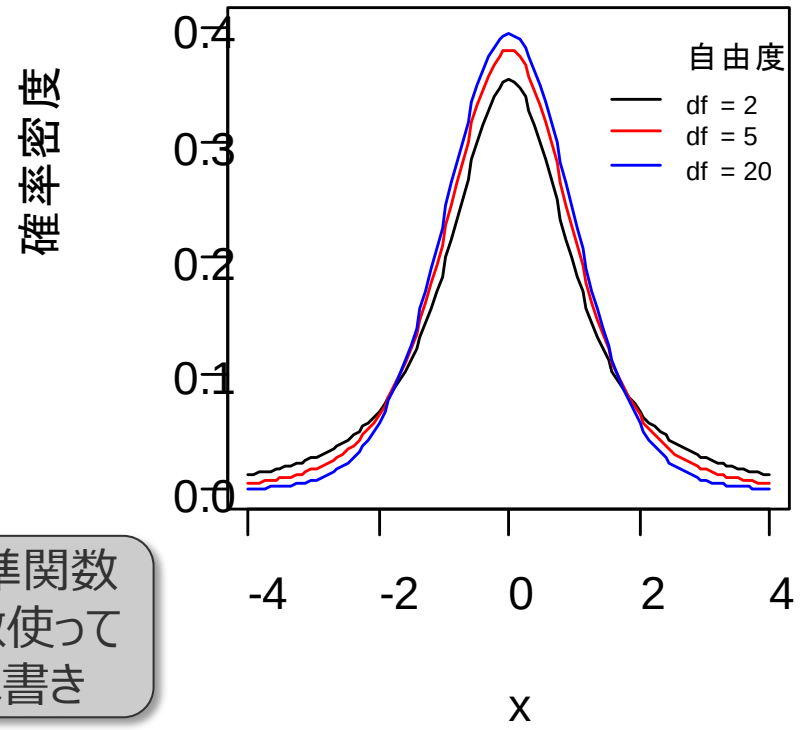
```
997  ## (c) t 分布(t distribution)
998  curve(dt(x, df = 2, ncp = 0),
999        from = -4, to = 4,
1000       ylim = c(0,0.4),
1001       ylab = "確率密度", main = "t 分布",
1002       col = "black",
1003       las = 1)
1004
1005 curve(dt(x, df = 5), col = "red", add = TRUE)
1006
1007 curve(dt(x, df = 20), col = "blue", add = TRUE)
1008
1009 legend(x = 1.2, y = 0.4,  # 位置を座標軸で指示
1010       legend = c("df = 2", "df = 5", "df = 20"),
1011       lty = 1,
1012       col = c("black", "red", "blue"),
1013       title ="自由度",
1014       cex = 0.6, title.cex = 0.7, bty = "n")
```

非心度省略可

高水準関数を複数使って重ね書き

（my_base_graphics3.R：997-1014）

(c) *t* 分布の確率密度曲線



標準関数によるグラフ作成 (22)

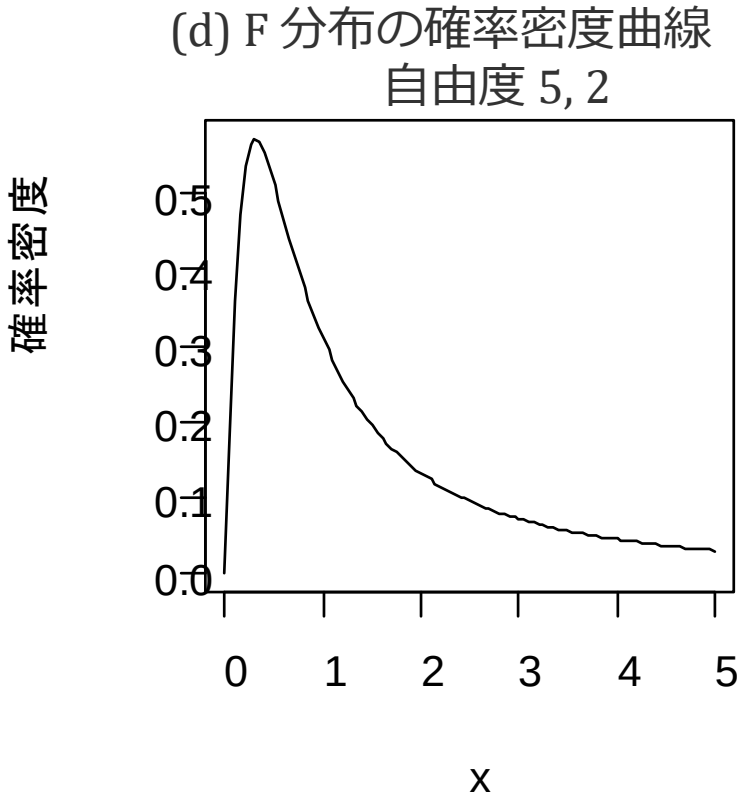
●標準関数：(22) 関数プロット（Function Plot） 、plot()、curve()

(d) F 分布の確率密度曲線（自由度 5, 2）

```
1016  ## (d) F 分布(F Distribution)
1017  curve(expr = df(x, df1 = 5, df2 = 2),
1018        from = 0,
1019        to = 5,
1020        ylab = "確率密度",
1021        main = "F 分布(自由度 5, 2)",
1022        las = 1)
```

式を渡す

（my_base_graphics3.R：1016-1022）



標準関数によるグラフ作成 (22)

●標準関数：(22) 関数プロット（Function Plot） 、plot()、curve()

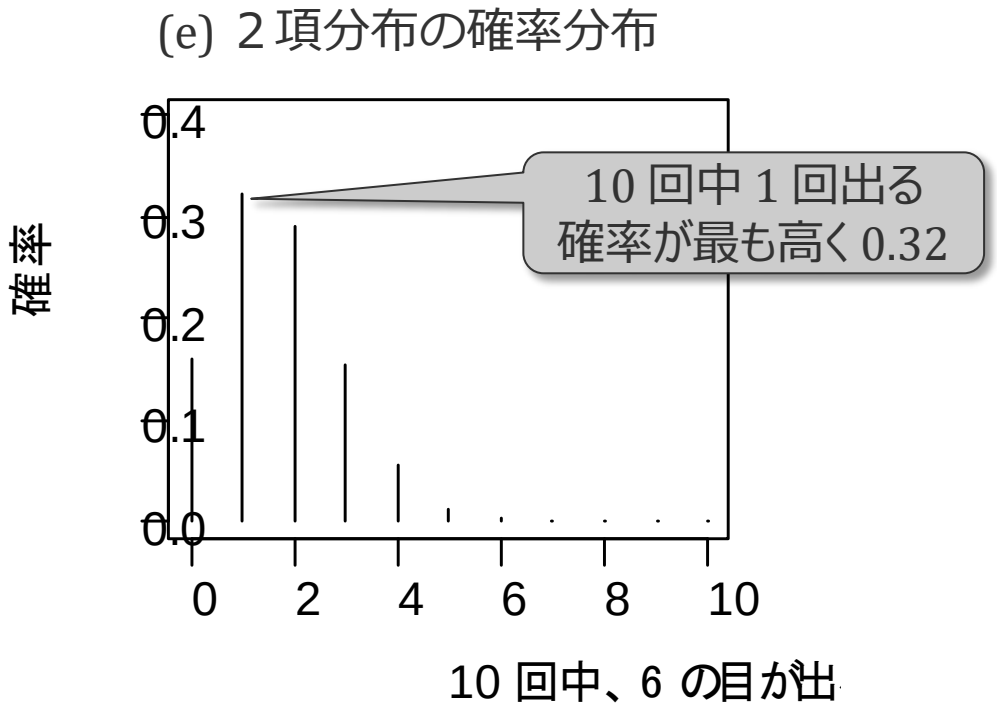
(e) ２項分布の確率分布

生起確率 π の事象があるとき、
独立した試行回数 n 回中、
その事象が起こる確率変数の分布

事例：公正なサイコロ（π=1/6）を 10 回投げて、
6 の目が出る回数とその確率の分布

```
1024  ## (e) ２項分布(Binomial Distribution)
1025  ##     公正なサイコロを10回投げたとき
1026  ##     6 の目が出る回数とその確率
1027  plot(x = 0:10,
1028       y = dbinom(x = 0:10, size = 10, prob = 1/6),
1029       type = "h",
1030       xlab = "10 回中、6 の目が出る回数",
1031       ylab = "確率",
1032       ylim = c(0, 0.4),
1033       las = 1)
```

（my_base_graphics3.R：1024-1033）



標準関数によるグラフ作成 (22)

●標準関数：(22) 関数プロット（Function Plot） 、plot()、curve()

(f-1) ２次関数のプロット（plot() を使用）

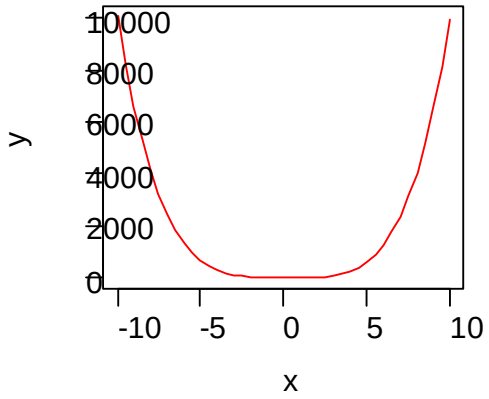
(f-2) ２次関数のプロット（curve() を使用）

```
1035  ## (f) ２次関数(Quadratic Function)
1036  ### (f-1)
1037  x <- seq(from = -10, to = 10, by = 0.5)
1038  y <- x ^ 4 - 5 * x
1039
1040  plot(x, y,
1041       type = "l",
1042       col = "red",
1043       las = 1)
1044
1045  ### (f-2)
1046  curve(expr = x ^ 4 - 5 * x,
1047        from = -10,
1048        to   = 10,
1049        col = "blue",
1050        las = 1)
```

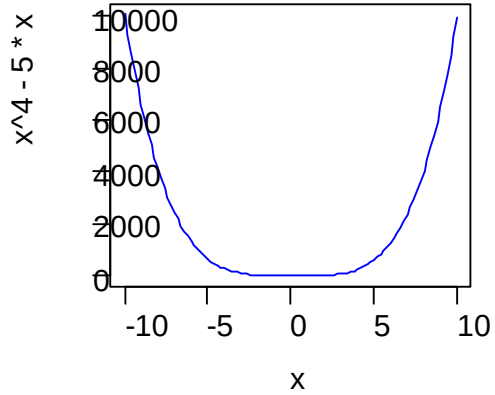
名前付き引数

（my_base_graphics3.R：1035-1050）

(f-1) ２次関数のプロット



(f-2) ２次関数のプロット



おわりに

●パッケージ graphics を学んだ次は、パッケージ lattice へ進む

R の graphics を主体とした標準関数は、基本的な描画操作を習得するための出発点

しかし、多変量データや条件付きプロットの場合、標準関数では作業が煩雑になることもある

パッケージ lattice を利用するメリット

(1) 多変量データを簡単に可視化できる

y ~ x | group のような式を指定するだけで、条件別にパネル分割された図を自動生成

(2) 見た目が一貫して見やすい

軸設定やレイアウトが統一的に管理され、論文・レポート向けのグラフを容易に生成

(3) 宣言型の書き方でコードが読みやすい

描画内容を一度の関数呼び出しで定義できる

後から追加描画をする必要がなく、再現性も高い

- 参考文献

Murrell, P (2006) R Graphics, Chapman & Hall

(久保拓弥(訳) (2009) R グラフィックス、共立出版

山本義郎ら(2013) 統計データの視覚化、共立出版

- 作成　片瀬雅彦
- 作成時期　2026年1月18日