2007 年日本行動計量学会 R チュートリアル・セミナー資料 ver0.31

石田 基広 *

2007/09/02

目次

* ishida-m@ias.tokushima-u.ac.jp

1 午前の部：初めての R

1.1 環境設定

始めに Windows 版の R での環境設定について説明する．インストール直後に R を起動すると，MDI のウィンドウが現れる．好みによるが，SDI の設定に変え，コマンド，グラフ，ヘルプがそれぞれ独立したウィンドウとして現れる方が使いやすいのではないか．そこでメニューから

[編集] → [GUI プリファレンス]

と操作し，ダイアログ上のラジオボタン，またフォントを好みで [MS Gothic] などに設定する．そしてダイアログ下の [保存] を選び，Rcosole というファイルを My Document フォルダに保存しておく．[OK] を押して，一度 R を終了させ，再び R を起動する．コンソールウィンドウが現れ，SDI ウィンドウ設定に変わっていることが確認できる．

なお R に関しては豊富な情報がインターネット上で公開されている．日本では http://www.okada.jp.org/RWiki/ が有名であるが，さらにメーリングリスト https://stat.ethz.ch/mailman/listinfo/r-help には多くの質問と答えが蓄積されている．メーリングリストの内容については，R から検索することができる．`?RSiteSearch` を参照されたい．

1.2 基本演算

R を使いこなす上で必要となるデータの構造について学ぶ前に，簡単な計算操作を実行してみる[*1]．

プログラミング環境としての R を使いこなす第 1 歩は「代入」あるいは「付値」に慣れることである．なお，R でコンソールに入力する命令のまとまりを「式」とも言う．詳細は『R の基礎とプログラミング技法』(ウーヴェ, 2006) を参照されたい．また R では「関数」を使って，各種の計算，処理を行う．関数は `fun()` という形式をしており，`fun` は関数名，また括弧内にはデータや，計算の条件などを指定する．関数の括弧内に指定する要素を関数の「引数」と言う．

以下，実例を示す．なおキーボードの矢印キーを使うと，直前に実行した式が自動的に補完されるので，いちいち入力を繰り返す手間がなく，非常に便利である．下の入力例で #記号より右は単なる解説であり，無視して構わない．仮に入力しても R は # から改行までを無視する．

```
  # 基本演算
1 * 2         # 掛け算．なお シャープ記号（#）以降の文字列は無視される
10 / 3        # 通常の割り算
10 %/% 3      # 整数割り算
10 %% 3       # 余りを求める

y <- sqrt(9) # 9 の平方根を求める「関数」の出力を y に「代入」
y            # 代入の結果を確認
             # この y <- sqrt(9) を代入「式」とも言う
```

[*1] なお，これから実行するコードは http://150.59.18.68/bsj2007.R として公開している．

```
(y <- log(2.718)) # 式を括弧で囲むと，計算と同時に代入結果が出力される

# 数値を丸める
floor(y) ; ceiling(y) ; round(y) # 複数の式をセミコロン（;）で区切って実行する

round(0.9541, digits = 2) ; round(0.9551, digits = 2)
# 表示する小数点位置を引数として指定した
# ただし R の「丸め」関数 round は IEEE 方式であり，五捨もありうる
```

R では，代入先の変数，ここでは y を「オブジェクト」ともいう．

2　データの型と構造

R を使いこなす上で必須の知識であるデータの構造について説明する．

2.1　データの型

データの構造とは別に，データには「型」がある．型は，大きくは数値かカテゴリ (文字など) である．R では次の五つの型が用意されている．表 (2–1) を参照されたい．型が互いに異なる要素を一つのベクトル (ベクトルについては後述) にまとめることはできない．異なる型を一つのベクトルにまとめようとすると，どれか

表 2–1　原子データ型

説明	例	データ型
空値	`NULL`	*NULL*
論理値	`FALSE`	*logical*
整数，実数	`3.14`	*numeric*
複素数	`2.13+1i`	*complex*
文字，文字列	`"Hello"`	*character*

一つの型に強制変換される．変換では表 (2–1) の下に位置する型が優先される．

```
  # データの型
y <- c(1, 3.14, "A")
y
```

この場合，文字の A が表 (2–1) の一番下にあるので，他の数値も文字として扱われる．出力から明らかなように，文字あるいは文字列には引用符が使われる．

2.1.1　factor

文字型と混同しやすいが，factor という概念がある．

```
  # Factor　はカテゴリ変数などを表すの使われる
y <- factor(c("A","B","C"))
y          # 文字として表示されるが
str(y)     #
mode(y)    # 実体は数値である
```

factor は文字として表示されるが，実体は数値である．この性質を応用して，例えばグラフを描く場合，factor の水準ごとに線種や色を変えることができる．

2.2　ベクトル

ベクトルは R においてもっとも基本的なデータ構造である．すなわち複数の数値，あるいは文字列を一つにまとめたオブジェクトである．

```
  # ベクトル を作成するには関数 c() を利用する
y <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
y
y <- 1:10           # コロンを使うと連続した整数を作成できる．c() は不要
y[1]                # ベクトルの一部の要素を抽出できる． [] を「添字」と呼ぶ
y[1:3]              # 添字そのものをベクトルで指定しても構わない
y[c(1, 3, 5, 7)]

z <- seq(0, 10, 2) # seq 関数は範囲と間隔を指定したベクトルを作成できる
                   # 範囲は引数で指定する
                   # 括弧内の最初のコンマまでを第一引数，二つ目を第二引数と呼ぶ
z <- rep(1:3, 3)   # rep 関数は第一引数で指定したベクトル 1:3
                   # つまり 1:3 =  c(1, 2, 3) を 3 回繰り返す

# ベクトルの各要素には名前を付けることができる
y <- c(1,2,3)
names(y) <- c("A", "B", "C")
y
mean(y)
```

R において演算はベクトル単位で行われる．プログラミング言語としての R は，オブジェクトの基本単位がベクトルである点に特徴である．言い方を変えると，R に精通するにはベクトルとその性質，操作法に習熟

する必要がある．以下，ベクトルを対象とした演算を紹介する．

```
  # ベクトルを使った演算
y <- 40:60    # 40, 41, ..., 59, 60 という数列が生成される

# ベクトルを対象とした関数類
length(y)     # 要素の数
sum(y)        # 合計
mean(y)       # 平均
var(y)        # 不偏分散
sd(y)         # 標準偏差
summary(y)    # 四分位点など

plot(y)       # 散布図．この場合 x 座標には自然数が設定される
hist(y)       # ヒストグラム．区間の設定については後述

# R 独特のベクトル演算の例
y + 1         # ベクトルの「すべて」の要素に 1 が足される
```

なお R では，スカラーは要素数が一つのベクトルと見なされている．

2.3 行列

行列はベクトルを矩形に並べたものである．R におけるデータ処理では，行列と次に取り上げるデータフレームの扱いが重要になる．行列オブジェクトを作成するには関数 matrix() を利用する．第 1 引数にデータを，第 2 引数で行数を，第 3 引数で列数，第 4 引数で要素を埋めていく方向 (縦あるいは横) を指定する．

なお関数の引数には名前が付いていることが多い．matrix() の場合，第 1 引数は data，第 2 引数が nrow，第 3 引数 ncol，第 4 引数 byrow である．引数名を指定するならば，その順番は任意である．なお引数名は簡略化できる．ncol の代わりに nc としても同じである．特に R に関する参考書やマニュアル類では，省略した引数名を利用していることがよくある．関数の引数名，あるいは関数の使い方を参照したければ ?matrix のように，関数名の頭に疑問符を付けて実行すればよい．この場合，関数名の最後の括弧は付けない．

```
  # 行列
(mat <- matrix(data = c(38, 14, 11, 51), ncol = 2))
(mat <- matrix(c(38, 14, 11, 51), nc = 2))    # 上と同じ

# 行列には行名，列名を指定できる．
rownames(mat) <- c("light", "dark")
colnames(mat) <- c("blue",  "brown")
mat
```

```
colSums(mat); rowSums(mat)      # 列あるいは行ごとの合計
colMeans(mat); rowMeans(mat)    # 列あるいは行ごとの平均
apply(mat, 1, sum)              # rowSums(mat) に同じ
apply(mat, 2, sum)              # colSums(mat) に同じ

# 行列は，例えばカイ二乗検定で使われる
chisq.test(mat)
```

2.4 データフレーム

データフレームは R でデータ解析を行う上で，もっとも頻繁に使われるデータ構造である．行列と似ているが，違いは，ある列には数値を，別の列には文字列を指定することができる．つまり数値やカテゴリを一つのデータオブジェクトとしてまとめることができる．

始めに簡単なデータフレームを作成してみよう．

```
  # データフレーム
# 6 人の生徒の数学の試験結果をデータフレームにまとめてみる
y <- data.frame(                # 適当に改行を行う．一行で入力しても構わない
        students = c("Michiko","Taro","Masako","Jiro","Aiko","Kenta"),
        math = c(50, 60, 70, 80, 90, 100))     # 閉じ括弧の数に注意
y                               # オブジェクト y の中身を確認する

# 同じことだが，先にデータを表すベクトルを作って，これを結合しても構わない
name <- c("Michiko", "Taro", "Masako", "Jiro","Aiko","Santa")
math <- c(50, 60, 70, 80, 90, 100)
name.math <- data.frame(students = name, math = math)
name.math                       # オブジェクト y と中身は同じ

# 先の行列をデータフレームで作成し，カイ二乗検定を行ってみる．
dat <- data.frame(blue = c(38, 14), brown = c(11, 51), # コンマを忘れずに
            row.names = c("light","dark"))             # 列名を同時に指定
chisq.test(dat)                                        # カイ二乗検定

fix(dat)              # GUI 画面を呼び出して修正などもできる
```

データフレームは非常に重要な構造であるので，後でもう一度データフレームの操作方法を確認していく．

2.5 リスト

リストは，行列やベクトル，データフレームをまとめたオブジェクトである．

```
  # リスト
x <- 1:10                          # ベクトル
y <- matrix(1:9, ncol = 3)         # 行列
# そしてデータフレーム
z <- data.frame(category = c("A","B","C"), data = c(1,2,3))
# これらを一つにまとめてしまう
xyz <- list(x, y, z)
xyz
# リスト作成時に各要素に名前を付けこともできる
xyz <- list(x = x, y = y, z = z)       # 同じリストを変数名付きで作成
xyz
```

ここではリストに三つのデータを統合している．リストのそれぞれの要素にアクセスするには，[[]] という二重括弧の添字を使う．

リストオブジェクトを自ら作成する機会は少ないかもしれないが，R パッケージの解析関数には，その出力がリストになっているものが少なくない．リストの処理は分かりにくい面もあるが，その要素へアクセスするには二重括弧 [[]] が必要だと覚えておいてほしい．

```
  # リストの要素にアクセスする
xyz[[1]]       # 名前付きならば xyz$x でも良い
xyz[[2]]       # 名前付きならば xyz$y でも良い
xyz[[3]]       # 名前付きならば xyz$z でも良い

# リストの要素の，そのまた要素にアクセスする
xyz[[3]][1]    # データフレームの一列目
xyz[[c(3,1)]]  #  上と同じ出力
xyz[[3]][[1]]  #  上と同じ出力

xyz[["z"]][[1]]#  上と同じ出力
xyz$z[[1]]     #  上と同じ出力
```

3 データフレームの操作方法

3.1 データフレームの一部を参照する，取り出す

データフレームの作成には前節で見たように data.frame() 関数を利用するが，ここでは作成したデータフレームオブジェクトの操作方法，さらには変更方法を確認していく．

データフレームは添字を使って，要素を抽出することができる．ただしベクトルの場合と違って，行と列からなるので，行番号と列番号を二つ指定しなければならない．添字の [] 内をコンマ (,) で区切り，前に行番号，後に列番号を指定する．指定しない (空白の) 場合は，全行ないし全列を指定したのと同じことになる．

```
  #  データフレームの操作，確認方法．先ほど作成した name.math を利用
name.math[3, ]         # 3 行目を確認
name.math[3:5, ]       # 3 行目から 5 行目を確認
name.math[c(1,3,5), ] # 1，3，5 行を確認
              # 複数の数字を並べる場合は c() 関数の間に数字をコンマで区切る
name.math[1, 1]        # データの 1 行目の名前（1 列目）を確認
name.math[1, 2]        # データの 1 行目の得点（2 列目）を確認
name.math[ , 2]        # 特定の列にアクセスする
```

ここでデータフレームに，性別を表す列と，国語の得点を加えてみる．

```
  # データフレームの加工
(gender.data <- rep(c("female", "male"), 3)) # rep 関数を使って効率的に
(kokugo.data <- math - 5)                    # 国語の点を作成
name.math$kokugo <- kokugo.data              # これで kokugo 列が追加される
name.math$gender <- gender.data
name.math
name.math <- name.math[, c(1, 4, 2, 3)]      # 列を並び替えたければ
                                             # 列番号を指定して上書きする

fix(name.math)                               # GUI 画面で修正
scale(name.math$kokugo)                      # 正規化（標準化）
scale(name.math$kokugo)  * 10 + 50           # 偏差値
```

まず先ほど作成したデータフレームには行名 (生徒名) と列名 (数学) が指定されている．

```
  # このオブジェクトに平均を求める関数を適用してみると
mean(name.math)            # 数値列だけ，平均が計算される．好ましい操作ではない
mean(name.math$kokugo); mean(name.math$math)  # 数値列だけ指定
mean(name.math[, c(3, 4)])                    # 3 列目と 4 列目の平均
```

```
mean(name.math[, c("math", "kokugo")])        # 上の処理に同じ．列名で指定
```

4 外部データの利用

4.1 CSV ファイルの扱い

データを入力するには R ではなく，OpenOffice の Spreadsheet や Microsoft の Excel を利用し，作成したファイルを R に読み込んだ方が便利であろう．R には Excel 形式のデータを直接操作するパッケージも用意されているが，今回は csv 形式のデータを扱う方法を練習する．

ここでデータフレームを csv 形式のファイルとして書き込む方法を実践してみよう．csv 形式のファイルの書き込みには `write.csv()` 関数を利用する．

```
  # ファイルへの保存
setwd("E:/tmp")  # 保存先を指定したい場合．フォルダの区切りは "/" で
# setwd(tempdir()) # 保存先を一時フォルダと指定
getwd()          # ワーキングディレクトリを確認する

y <- data.frame(category = c("A","B","C"), data = c(1,2,3))

write.csv(y, file = "bsj.csv", quote = FALSE, row.names = FALSE)

# 上のコードでの引数の意味は
# y =  保存するオブジェクトを指定，file = 新規ファイル名を指定
# quote = FALSE : 文字列に引用符を付けない指定
# row.names = FALSE : 行番号を含めない指定
```

続いて，このファイルを読み込んでみる．csv 形式のファイルの読み込みには `read.csv()` 関数を利用する．

```
  # ファイル名を指定し，その最初の行が列名 header と指定
new.y <- read.csv(file = "bsj.csv", header = TRUE)
new.y
```

なお Excel 形式のファイルを直接読み書きするには **xlsReadWrite** や **RODBC** パッケージを利用されたい．

5 グラフィックス作成

ここからグラフィックスの作成方法について説明する．グラフィックス一般については Murrell (2005) が詳しい．

5.1 棒グラフ

始めに棒グラフを作成してみる．R で棒グラフを手軽に作成するには barplot() 関数を利用すれば良い．ここでは barplot() 関数のヘルプに基づき，R に組込みのデータセット VADeaths を例に作図を行う．VADeaths はヴァージニア州の 1940 年の死亡率を，地域別，性別，年令別にまとめた小さなデータセットである．詳細は ?VADeaths を実行されたい．

```
  # 棒グラフ．barplot のヘルプを参照のこと
VADeaths                                # VADeaths は R に組み込みのデータ
barplot(VADeaths)                       # これだけで単純な棒グラフが作成される
?barplot                                # barplot() 関数のヘルプ

# 必要があれば，色を指定できる
barplot(VADeaths,                       # 色は数値でも，英語名でも指定可能
         col = c("lightblue", "mistyrose", "lightcyan",
                 "lavender", "cornsilk") )
?colors                                 # colors についてヘルプを見る

barplot(VADeaths, beside = TRUE)        # 横に並べ直す
barplot(VADeaths, beside = TRUE,
        legend =  rownames(VADeaths))   # 凡例を付加する

# 続けてメインタイトルを付加する
title(main = "Death Rates in Virginia",
        font.main = 4,                  # 字体の指定
        cex = 1.2)                      # タイトルの文字サイズを変える
?title                                  # title のヘルプを見る
```

ここでメインタイトルにフォントを数値で指定している．4 種のフォントが指定でき，1 はプレーンテキスト，2 がボールド，3 がイタリック，そして 4 がボールドイタリックである．また基本的な色彩の番号を参照するには which(colors() == "yellow") などと実行されたい．

5.2 折れ線グラフ

ここでは折れ線グラフを描きながら，さらにグラフィックスのパラメーターの指定方法を学ぶ．利用するデータは R 組込みの Longley の経済回帰データであるが，詳細は ?longley を参照されたい．

```
  # Longley の経済回帰データから 2 列を抽出してグラフィックス化
matplot(longley[c("GNP", "Unemployed")],
        type = "l",                             # type で折れ線の種類
```

```
        xlab = "",  ylab = "", axes = FALSE)   # 軸目盛とラベルは後で調整
# x 軸の設定を R に任せると 1 から行数までの自然数となる

# 続いて軸を設定
axis(1, 1:nrow(longley["GNP"] ), row.names(longley["GNP"]))  # x 座標
axis(2)                                                      # y 座標はデフォルトと同じ
legend.xy = locator(1)                                       # マウスを使って座標を得る

#   この間にグラフィックス上の任意の場所をクリックする

legend(legend.xy$x, legend.xy$y,                             # 取得した座標を使って凡例を追加
       legend = c("GNP", "Unemployed"),                      # データの列名
       col = 1:2, lty = 1:2)                                 # 色と線種を凡例に加える
```

凡例を追加する locator() 関数や legend() 関数については，散布図のところで，もう一度説明する．

5.3 ヒストグラム

ヒストグラムは hist()，barplot()，truehist() などの関数を利用して作成することができる．hist() ではデータをそのまま渡せば，自動的に区間設定 (スタージェスの方法) を行った上で，頻度を求めてグラフィックスを作成してくれる．

なお rnorm()，rpois() は，それぞれ正規分布とポアソン分布に従う乱数を生成する関数である．語頭の r_はランダムという意味である．正規分布では，平均と標準偏差の二つのパラメータを指定する必要があるが，省略された場合は，自動的に平均が 0，標準偏差は 1 と設定される．ポアソン分布のラムダパラメータは明示的に指定しなければならない．R の基本パッケージに備わっている確率変数については『R の基礎とプログラミング技法』 pp. 140–142 を参照頂きたい．

```
  # 連続データの例
# 平均 50，標準偏差 10 に従う乱数を 1000 個生成
y <- rnorm(100, mean = 50, sd = 10)
hist(y)                          # まずデフォルトのヒストグラムを見てみる

dev.off()                        # デバイス（図のウィンドウ）を閉じる
```

barplot() を利用する場合は，データの頻度をあらかじめ求めておく必要がある．頻度は table() 関数で頻度を求めることができる．

```
  # 離散データの例
y <- rpois(100, lambda = 3)      # 平均 3 のポアソン分布に従う乱数を 100 個生成
barplot(table(y))                # table() で頻度を計ってグラフィックス化
```

```
# 同じデータに truehist() を適用する
library(MASS)                    # MASS パッケージをロードする
truehist(y)

dev.off()                        # デバイス（図のウィンドウ）を閉じる
```

5.3.1 区間の設定

R の頻度設定はデフォルトでは区間設定が右閉じである (right = TRUE). つまり指定した数値「以下」となる. これを「未満」に変えたければ明示的に right = FALSE とする必要がある. このパラメータの設定によってヒストグラムの形は当然異なってくるので注意されたい.

```
  # ヒストグラムの区間の問題点

# 作為的なデータを作成
y <- c(rep(0,2),rep(1,3),rep(2,5),rep(3,4),rep(4,2),rep(5,1),rep(6,1))
par(mfrow = c(1, 2))             # 画面を二分割する. 後述
hist(y)                          # 簡単にヒストグラムを作成する
barplot(table(y))                # table() で頻度をはかって
                                 # 棒グラフ（ここではヒストグラム）を作成

dev.off()

# 区間を自分で設定する. 区間設定には breaks 引数を利用する
par(mfrow = c(1, 2))
hist.y <- hist(y, breaks = seq(0, max(y), 2))
hist.y$breaks                    # 設定された区間を確認
hist.y$counts
hist.y <- hist(y, breaks = seq(0, max(y), 2), right = FALSE);
hist.y$breaks                    # 設定された区間を確認
hist.y$counts                    # 頻度を確認

dev.off()
```

5.3.2 練習

```
  # 軸の作成方法
# p.162
rpois.data <- rpois(1000, 1.7)
```

```
hist(rpois.data)        # 0 から 1 のバーの高さは何なのか
table(rpois.data)       # 0 の頻度らしい
hist(rpois.data, breaks = seq(-0.5, max(rpois.data)+ 0.5, 1))
hist(rpois.data, breaks = seq(-0.5, max(rpois.data)+ 0.5, 1),
     axes = FALSE)
axis(1, 0:max(rpois.data), col.axis = "blue")
axis(2, seq(0, max(table(rpois.data)), 50), col.axis = "red" )
axis(3, seq(-0.5, max(rpois.data)+ 0.5, 1), cex.axis = 0.8)
```

この他，データに区間を行う関数としては cut() がある．詳細は ?cut として確認されたい．

5.4 散布図

散布図はもっとも基本的な関数 plot() で描画する．この関数は，対象データにあわせて適切なグラフィクスが描画されるように設計されている．始めに plot() 関数を実際に利用してみる．続けてグラフにラベルを加えるなどの技法を紹介する．ここでの操作は R で各種のグラフィックスを作成する際の基本である．

5.4.1 基本

```
  # 散布図．以下の操作は plot 関数のヘルプに基づく
head(cars)      # cars は R に組み込まれている関数で，速度と停止までの時間を表す
plot(cars)      # もっとも単純な方法
title(main = "cars data")                          # 図を作成後，タイトルを加える

# 散布図を作成する際にラベル等を指定する
plot(cars, main = "How to plot", sub = "sample",          # タイトル類を指定
     xlab = "Speed (mph)", ylab = "Stopping distance (ft)", # ラベルを指定
     las = 1)                           # これは軸の目盛を軸に対して水平にする指定

# 以下，様々なオプションを試してみる
plot(cars, type = "b", col = "green") # 散布図のタイプ type を指定
plot(cars, type = "h", col = "purple", lwd = 5) # ヒストグラム，lwd は線の太さ
plot(cars, type = "h", col = "gray", lty = 3)   # 線種 lty を変えてみる
plot(cars, pch = 8, col = "blue")      # 点の記号 pch を変更し，色 col も指定
```

ここまで plot() 関数の引数として las，lty，lwd などを指定してきた．R にはグラフィックス作成用の豊富な引数が用意されており，柔軟なグラフィックスを作成することができる．指定されたグラフィックス関連の引数は par() 関数に引き継がれて処理される．?par を実行すると，利用可能な多数の引数を見ることができる．

なお R の解説書では，図を作成する際に，引数に type ="n" を指定していることがある．この場合，図の

レイアウト部分だけが設定され，データの描画は行わない．これは例えば，実測値ではなく，理論曲線のみを散布図に描く場合などに使われるテクニックである．

```
  # type = "n" の効果を確認
par(mfrow = c(2,1))
plot(cars)                    # 一つ目のプロット作成
t(cars, type = "n")           # 二つ目のプロットの輪郭だけ作成
lines(lowess(cars))           # ノンパラメトリックな平滑曲線を追加

dev.off()

#  スピードの数値を散布図上に表示
plot(cars, type = "n")
text(cars$speed, cars$dist, cars$speed)   # text() 関数は指定のラベルを表示

# 車にアルファベットの名前を付ける
plot(cars, type = "n")
text(cars$speed,  cars$dist, c(letters, LETTERS)[1:length(cars$dist)])

dev.off()
```

5.4.2 練習

```
  # Crawley 2007, p.141．グラフィックスパラメータ pch を参照する
plot(0:10, 0:10, type = "n", xlab = "", ylab = "")
k <- 1

for(i in c(2, 5, 8)){
  for(j in 0:9) {
    k <- k+1
    points(i,j, pch = k, cex = 2)
  }
}
```

5.5 凡例の追加

凡例を追加するには，挿入位置を指定する必要がある．それには，大まかな位置を指定する方法，座標を指定する方法があり，さらに後者の場合には，グラフから座標を取得して，その座標点を利用する方法もある．

```
  plot(cars)
legend("topleft","cars data",pch = 1,col = "blue") # 凡例を挿入
dev.off()

# 次のようにして凡例を挿入しても良い
plot(cars)
xy <- locator(1)                              # R がマウス操作を待つ
# ここで図の上でクリックする

legend(xy$x, xy$y, "cars data", pch = 1)   # クリックした場所の座標を使う

#  ちなみに特定のプロット点にラベルを付与することもできる
identify(cars$speed,cars$dist,
         labels= c(letters,LETTERS)[1:length(cars$speed)])
# 実行後，プロット点の近くを左クリックすると，指定のラベルが表示される
# 右クリックで，ラベルの表示を中止する



# data(iris) # R にデフォルトで登録されている 3 種類のアヤメの計測値
head(iris)
levels(iris$Species)            # 種類を確認
attach(iris)                    # iris を現在のワークスペースに登録
plot(Sepal.Width, Sepal.Length, pch = as.numeric(Species),
     col = as.numeric(Species))
legend("topleft", levels(Species), pch = 1:3, col = 1:3)

detach(iris)                    # iris を現在のワークスペースから削除
```

plot() 関数の引数 col などは数値を指定することができる．ここでは Species が factor であるのを利用して，これを数値に変えている．factor の実態は数値であったのを思い出してほしい．1 から 3 まで 3 色を指定している．

locator(1) を実行すると，マウスがクリックされるのを R は待つ．クリックすると，その座標が返される．座標はリストオブジェクトとなっているが，これを利用して凡例を追加する．

iris はデータフレームだが，Species 列に品種名が factor として登録されている (is.factor(iris[,5]) とすると確認できる)．levels() 関数を利用すると，factor 名を「文字列」として取得することができる．ここでは品種の数を確認するために実行している．

なおデータフレームのように，複数の列からなるオブジェクトでは，個々の列にアクセスする必要に迫られる場合が多い．そのような場合オブジェクト名に添字 [] を利用するか，オブジェクト名と列名を $ でつないで指定することができる．

ただし，オブジェクト名をそのたびに入力するのは煩わしい．そこで，オブジェクトをワークスペース (作業台) に登録し，単に列名だけを指定すれば済む方法が用意されている．オブジェクトを登録するための関数が attach() である．なお一度 attach() したオブジェクトは，利用する必要がなくなれば detach() 関数でワークスペースから消去することができる．現在ワークスペースに登録されている内容は ls() で確認することができる．

ここで注意が必要なのは，attach() を実行後，データ本体に操作を加えても，その操作はデータのコピーを対象とすることである．すなわちデータ本体には変更は生じない．さらに，複数のオブジェクトを attach() すると，それらのオブジェクト内部で同名の変数が使われている場合があり，後から attach() された変数名が優先される．例えば最初に attach() した data1 に変数 x があり，後で attach() した data2 に同じく変数 x があった場合，x は data2$x を指す．

このように attach() 関数は混乱を招くことが多々あるので，R になれてくれば使わない方が良い．attach() 関数を使ってコピーが作成された様子，また with() 関数を使う方法を以下に示す．詳細は『R の基礎とプログラミング技法』 p. 45 を参照されたい．

```
  # attach() は時に混乱を招く
head(trees)         # アメリカ桜の測定値
mean(Height)        # attach されていない場合 trees$Height とする
attach(trees)       # attach する
mean(Height)        # $ 記号なしで直接アクセスできる

Height[1] <- 100    # １行目のデータを変更する
head(Height)        # 変更されたように見えるが
head(trees)         # もとのデータフレームに変更はない
detach(trees)       # detach するが
Height              # Height は残ったまま
rm(Height)          # コピーを削除

# attach() を使わずに要素にアクセスする方法
with(trees, mean(Height))
with(trees, plot(Volume ~ Height))

# ただし関数 plot() は data 引数の指定が可能なので
plot(Volume ~ Height, data = trees)
```

5.5.1 練習

```
  # グラフにデータを追加する. Crawley 2007, p.136
# 最初のデータをもとに散布図を描き,
data1 <- read.table(
  "http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/scatter1.txt",
   header = T)
attach(data1)
names(data1)

plot(ys ~ xv)
abline(lm(ys ~ xv), col = "gray", lwd = 2)

# 同じ散布図に別のデータを追加する
data2 <- read.table(
  "http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/scatter2.txt",
   header = T)
attach(data2)
names(data2)

# points() 関数を使って点を追加する
points(ys2 ~ xv2, col = "blue", pch = 2)
abline(lm(ys2 ~ xv2), col = "green", lty = 2, lwd = 2)

# ところが 2 番目のデータはすべてが表示されてない
range(ys)
range(ys2)

X11()       # 新たに描画ウィンドウを開いて

# こうした場合 c() 関数で最初に両方のデータを足して散布図の原型を作ってしまう
plot(c(xv,xv2),  c(ys, ys2), xlab = "x", ylab = "y", type = "n")
#
points(ys ~ xv)
abline(lm(ys ~ xv), col = "gray", lwd = 2)
points(ys2 ~ xv2, col = "blue", pch = 2)
abline(lm(ys2 ~ xv2), col = "green", lty = 2, lwd = 2)

# あるいは最初のプロットで軸の範囲を指定する
range(c(ys, ys2))               # y の範囲を調べる
```

```
range(c(xv, xv2))                # x の範囲を調べる

X11()       # 新たに描画ウィンドウを開いて
plot(xv, ys, xlab = "x", ylab = "y",
     xlim = range(c(xv, xv2)), ylim = range(c(ys, ys2)))
#
abline(lm(ys ~ xv), col = "gray", lwd = 2)
points(ys2 ~ xv2, col = "blue", pch = 2)
abline(lm(ys2 ~ xv2), col = "green", lty = 2, lwd = 2)
```

6 基礎的な統計解析

6.1 基本統計量

ここで R を使って各種の統計量，さらに基礎的な解析手法を取り上げる．

始めに平均，分散，標準偏差などを求める方法を取り上げる．これらは R に用意された関数を利用して求めることになる．まず R に標準で用意されているデータ women を例に取る．

```
  # ここまでの復習をかねて
women            # data(women) はアメリカ人女性の平均身長と平均体重
head(women)      # データの冒頭部分を見る．変数名などの確認になる
nrow(women)      # 行数（ここではサンプル数）
women$height <- women$height * 2.54   # ベクトル演算の例（センチに変換）
women$weight <- women$weight * 0.45   # ベクトル演算の例（キロに変換）
mean(women)      # それぞれの平均
attach(women)    # ここでデータをワークスペースに登録し，入力数を減らす
var(women)                  # 共分散行列
var(height); var(weight)    # 個々の変数の不偏分散
sd(height); sd(weight)      # 標準偏差
summary(women)              # 上から最小値，第 1 四分位点，中央値
                            # 算術平均，第 3 四分位点，最大値

plot(height ~ weight)       # 散布図．plot(height, weight) でも良い
(cor.women <- cor(height, weight)) # 相関係数を求めてオブジェクトに保存し

# paste 関数で，文字と数値を一つにあわせる
women.legend <- paste("Correlation = ", round(cor.women, 3))

# 相関係数を散布図に書き込む
legend("topleft", legend = women.legend)
```

```
# 同じことだが，やや高度な応用的方法
# legend("topleft", legend =
         substitute(Correlation == cw, list(cw = cor.women )))

dev.off()


library(lattice)            # lattice を使って同じグラフを作成
xyplot(height ~ weight,     # legend などに工夫を凝らすことができる
   groups = 1:15, type = c("p"),
   auto.key = list(title = "women", cex = 0.5,
   border = TRUE, lines = FALSE))

# 正規性の検定．コルモゴロフ・スミルノフ検定
ks.test(weight, "pnorm", mean = mean(weight), sd = sd(weight))
shapiro.test(weight)  # この他に car や nortest パッケージを参照のこと

detach(women)          # 利用が終ったオブジェクトはワークスペースから開放
```

ここで利用した paste() 関数は，文字列を結合するのに使われる．これは例えばグラフィックスのラベルとして，変数名とその統計的指標を組み合わせた文字列などを作成するのに重宝する関数である．

6.2 カイ二乗検定

R の組込みデータセットを使ってカイ二乗検定を実施してみる．ここで使うデータ HairEyeColor は array というデータ構造になっている．これは行列を多次元に拡張したものと考えられ，データとしては多次元分割表にあたろう．多次元分割表の解析としては「対数線形モデル」が考えられるが，詳細は ?loglm を参照して頂くとして，ここでは男女をまとめてしまおう．

```
  # R パッケージ組込みデータを利用する
HairEyeColor              # 統計学受講生の髪と目の色のデータ
is.array(HairEyeColor)    # 配列データ型
HairEyeColor[,,"Male"]    # 男子だけを確認．HairEyeColor[,,1] でも同じ
# 男女をまとめてしまう．apply() はオブジェクトに指定の関数を適用する
y <- apply(HairEyeColor, c(1, 2), sum)
y
# モザイクプロットを作成 各カテゴリの組み合わせごとの頻度情報が視覚的に分かる
mosaicplot(y, shade = TRUE,
           main = "Relation between hair and eye color")
```

```
chisq.test(y)                # カイ二乗検定を実施する
```

apply() 関数は，引数として指定されたオブジェクトに，同じく引数として指定された関数を適用する．ただし適用する方向として行 (1) と列 (2) が指定できる．ここでは両方指定している．どちらか片方だけを指定すれば，その周辺度数が得られる．

6.2.1 G 検定

分割表の検定方法．以下であたえられる G 値をカイ自乗分布で判断する．

(1) $G = 2 \sum O \ln \left(\frac{O}{E} \right)$

```
  # Crawley 2007, p.301 より
# カイ自乗検定
hair.eye <- matrix(c(38, 11, 14, 51), ncol = 2, byrow = TRUE)
chisq.test(hair.eye)

# G 検定．変数間の関係をより細かくチェックできる．Crawley 2007, p.552 より
hair.eye <- c(38, 11, 14, 51)
hair <- factor(c("fair","fair","dark", "dark"))
eye <- factor(c("blue","brown","blue", "brown"))
(h.e.glm.1 <- glm(hair.eye   ~ hair * eye, poisson))
(h.e.glm.2 <- glm(hair.eye   ~ hair + eye, poisson))
anova(h.e.glm.1, h.e.glm.2, test = "Chi")
```

6.3 t 検定

```
  # chickwts は餌のタイプによる鶏の体重変化データ
levels(chickwts$feed) # 餌のタイプを確認
# 箱ヒゲ図を作成
boxplot(weight ~ feed, data = chickwts, col = "lightgray",
         varwidth = TRUE, notch = TRUE, main = "chickwt data",
         ylab = "Weight at six weeks (gm)")
```

ここで箱ヒゲ図を作成する際に引数として notch = TRUE を指定している．ノッチとは箱ヒゲ図の胴体のくびれを表し，二つの箱でこのくびれの先が重なっていなければ median に有意な差があると見なせることになる．また varwidth = TRUE は箱の横幅がデータ数で調整されるための指定である．この例の場合，見た目にはほとんど差はないが．

この図からカゼインとアマニには有意な差があるように思われるので，これを確かめてみる．

```
  # データをワークスペースに登録
attach(chickwts)
casein <- chickwts[feed == "casein",]   # カゼインのデータ
linseed <- chickwts[feed == "linseed",] # アマニ (亜麻仁) のデータ
# 等分散性の検定
var.test(casein$weight, linseed$weight)

# t 検定を実施
t.test(casein$weight, linseed$weight, var.equal = TRUE)  # t 検定
# 以下参考までに
t.test(casein$weight, linseed$weight)              # 等分散を仮定しない場合
t.test(casein$weight, linseed$weight, paired = TRUE)     # 対応がある場合
# 少し複雑になるが，次のように実行しても良い
t.test(weight ~ feed,
       data = chickwts[feed == "casein" | feed == "linseed" ,])
```

ノッチはデータ数が少ない，あるいは群内分散が大きい場合には有効な指標ではないことに注意されたい．

6.4 分散分析

```
  # chickwts データに分散分析を実行する
summary(chickwts)              # 念のため．unbalanced なデータ
chick.aov <- aov(weight ~ feed, data = chickwts)
# 分散分析表を作成
summary(chick.aov)
# 各水準の効果を見る．これはモデル選択などで役に立つ
summary.lm(chick.aov)

plot(TukeyHSD(chick.aov))

detach(chickwts)   # 利用が終ったオブジェクトはワークスペースから開放
```

6.5 単回帰分析とモデル式

ここでモデル式について簡単に触れる．前節では分散分析を実行しているが，その際 aov(weight ~ feed のような式を利用した．これは体重 (weight) を餌 (feed) で説明する (あるいは回帰する) ことを表す式であり，チルダ ~ で二つの変数間の関係を表している．「チルダの左の変数を，右の変数で説明する」と考える．このような式を「モデル式」と呼ぶ．

6.5.1 単回帰分析

ここでモデル式の例として，単回帰分析について簡単に触れる．

```
  # R 組込みの data(women)
attach(women)                    # データをワークスペースに登録
women.lm <- lm(weight ~ height) # 回帰分析を実行．体重を身長で説明するモデル式
summary(women.lm)                # 結果の表示
plot(weight ~ height)            # 散布図を描き
abline(women.lm )                # 回帰直線を追加
fitted(women.lm)                 # モデルによる予測値
resid(women.lm)                  # 残差を確認する
# モデルの適合度を調べる          # 残差のチェック
par(mfrow = c(2, 2))             # 四つのグラフが描かれるので，画面を分割しておく
plot(women.lm)

dev.off(); detach(women)         # 分析実行後の後片付け
```

最後に紹介したプロットは，モデルの適合度を確認する補助となるプロットを作成した例である．詳細は Faraway (2005, pp.14 – 17), Crawley (2007, pp.357 – 359) を参照されたい．ここでは左上のプロットは，残差と適合値を対応させたもので，データの変動に何ら規則性がなければ，この散布図上で各点はランダムに散っているはずである．この図の場合，体重の変化には何らかの系統的な要因が働いている，あるいは分散に変動があると思われる．左下の図は残差を標準化し (平均 0，標準偏差 1 とする)，その絶対値の平方根の取った図である (絶対値のままだと skew が生じるため)．後者の図では点が三角形の形になる場合，heteroscedasticity が疑われる．この種の図が得られた場合は，例えば変数の二次の項を導入するなどの変換を検討することになる．

右上の図は QQ プロットと呼ばれ，残差が正規分布に従っているかをチェックするのに使われる．最後の図はクックの距離で，モデルの当てはめに影響のある外れ値を検出するのに利用される．ここで点線がクックの距離を表し，0.5 を越えるデータは影響が少なくなく，1 を越える場合はかなり影響があると見なす．

それぞれのグラフで，問題となるデータ点にはデータ番号が振られる．

なお非線形回帰を行う場合については **nls** パッケージを参照されたい．

6.6 モデル比較

R がデータ解析環境として優れているのは，データの探索的な解析をスムーズに実行できる点にある．すなわちユーザーはまずデータから各種のグラフィックスを作成し，その直感的な印象から最初の仮説をモデル化して解析を実行する．その上で，最初の解析結果を検討し，モデルをさらにアップデートする．この一連の手順を数行のプログラミングで自動化できれば，解析効率は飛躍的にあがる．R はこのような場合に非常に強力なツールとなる．

ここで Crawley (2002, p. 170) に基づき，簡単なモデル比較の実例を示す．

```
    # growth.txt は Crawley (2002), p. 170 による．タブ形式のデータ
# 二つの要因 diet と supplement による動物の体重変化
# Crawley 教授の Internet 上のサイトから直接データを読み取る
growth.data <- read.table(
  "http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/growth.txt",
  header = TRUE)

# 万が一，上記サイトへのアクセスがうまくいかない場合
# growth.data<-read.table("http://150.59.18.68/growth.txt",header=T)

summary(growth.data)
attach(growth.data)
names(growth.data)  # ラベルを確認
tapply(gain, list(diet, supplement), mean)
# 上の操作で tapply () 関数は第 1 引数として与えられたデータに，
# 第 2 引数で与えたリストごとに
# 第 3 引数で指定した関数を適用する

barplot(tapply(gain, list(diet, supplement), mean),
        beside = TRUE, col = c("red","orange", "yellow"))
# 凡例を追加
labs <- c("Barley", "Oats", "Wheat")
cols <- c("red", "orange", "yellow")
legend(6.3, 26, labs, fill = cols)

model1 <- aov(gain ~ diet * supplement) # 交互作用を含むモデル
 # aov(gain ~ diet + supplement + diet:supplement)# に同じ
summary(model1)                          # 分散分析表
summary.lm(model1)                       # 推定された全パラメータの一覧

model2 <- update(model1,.~. -diet:supplement)  # モデルのアップデート
summary.lm (model2)

# 上の 2 行は下に同じ
model3 <- aov(gain ~ diet + supplement)# 交互作用を除いたモデル
summary.lm (model3)

# モデルを比較
anova(model1, model2)                  # より単純な model2 の方が好ましい
```

```
# 自動的にモデル選択を行う
step(model1)
```

なおデータに適用する関数 `aov()` と `lm()` の違いは，その結果の `summary()` が誤差分散と F 比を含む分散分析表になるか，各効果の大きさとその標準誤差の一覧になるかである．後者はモデル選択などを行うの役立つ．`anova()` によるモデル比較 (partial F test) の仕組みについては Verzani (2004, pp. 307–308) に詳しい．

7　午後の部 1 ：R によるデータ処理

R は統計解析に特化したソフトではなく，一つの独立したプログラミング環境である．そこでプログラミング言語としての R に触れていく．

7.1　データフレーム

始めにデータフレームを復習する．データフレームとは，一種の行列であるが，数学で一般的な行列とは異なり，列には数値やカテゴリを含めることができる．行数は同じでなければならない．データフレームは data.frame() 関数によって作成する．

```
  # データフレーム
# 6 人の生徒の数学の試験結果をデータフレームにまとめる
# 先にデータを表すベクトルを作って，これを結合
name <- c("Michiko", "Taro", "Masako", "Jiro","Aiko","Santa")
math <- c(50, 60, 70, 80, 90, 100)
y <- data.frame(students = name, math = math)
y
# さらに列を追加する
(gender <- rep(c("female", "male"), 3)) # rep 関数を使って効率的に
(kokugo <- math - 5)          # 国語の点を作成．5 点を減らしただけ
y$kokugo <- kokugo            # これで　y に kokugo 列が追加される
y$gender <- gender
y
y <- y[, c(1, 4, 2, 3)]       # 列を並び替えたければ
y
```

7.2　添字による操作

ここで作成したデータフレームを利用して,「条件抽出」を行ってみる．ここで「条件抽出」とは, 与えられたデータの中から, 条件を指定して, その一部を取り出すことである．条件の指定では <，>，&，|，==，!= などの演算子を利用する．<，>= は明瞭だと思うが，&，| は 論理積，論理和と呼ばれるもので，英語では and と or にあたる．== は「等しい」,!= は「異なる」を表す.

```
  # プログラミング
y[y$math > 70, ]                    # 数学が 70 点より上の生徒だけ．コンマは必須
y[y["math"] > 70, ]                 # 上に同じ

y[y$gender ==  "male", ]            # 男子だけ取り出す
```

```
y[y["gender"] ==  "male", ]          #  同上

attach(y)                            # y をデータフレームとして登録しておけば
y[math > 70, ]                       # y$ の部分を省略できる．
y[math > 70 & kokugo > 80, ]         # 数学と国語の両方で条件指定
subset(y, math > 70)                 # 同じことは subset 関数を使っても実行可能
subset(y, math > 70, students)       # 抽出結果の特定の列だけ第 3 引数として指定
subset(y, math >= 70 & kokugo >= 80) # 数学と国語の両方で条件指定
y[gender == "female", ]              # 女性のデータを取り出す．
y[gender != "female", ]              # 女性以外のデータを取り出す

mean(y[gender == "male", "kokugo"])  # 男子生徒の国語の平均点を計算する
# gender が male である全ての行の，kokugo 列の平均

cor(y[, "kokugo"], y[, "math"])      # 相関係数は cor() 関数で求める
```

7.3　条件制御

R はプログラミング言語として，処理の流れを条件制御することができる．制御のために頻繁に使われるのが if() 条件分岐と for() ループである．

```
  # if 制御の基本
x <- 1
if (x == 1) {       # 括弧を忘れずに
  print("x は 1 ")
} else {            # 括弧を忘れずに．また else の前で改行してはいけない
 print("x は 1 ではない")
}


  # for による制御
x <- 1:10
x
for(i in x){
  print(i)
}
# in x の部分は，ベクトル x の要素を先頭から一つずつ取り出す

x <- c(1, 5, 10)
```

```
for(i in x){ # i はループのたびに，1 -> 5 -> 10 と変化する
  print(i)
}
```

データ解析において条件制御には様々な応用場面がある．次は極めて単純な例である．ここで paste() は「文字列」を連結するための関数である．

```
  # for の応用． paste() 関数と組み合わせる
paste("this", "is", "a","pen", sep ="-") # 要素をハイフンでつなげる
paste("this", "is", "a","pen", sep ="")  # 要素を区切り記号なしでつなげる

# paste 関数を使うと，以下のような連番のついたカテゴリを自動生成できる
students.no <- numeric(6)                 # 要素数が 6 個の空のベクトルを作る

for(i in 1:6){
 students.no[i] <- paste("student", i,  sep = "")
}
students.no

y                                         # もとの y を確認した上で
y$students <- students.no                 # 実名を番号付きラベルで置き換える
y
```

7.4　apply() 関数の利用

R にはデータフレームや行列の柔軟な操作を可能にする apply() 関数群が用意されている．ここでは分割表などを作成するのに便利な tapply() を例に取るが，他に lapply(),mapply(),sapply() が用意されている．

基本的には apply() は配列や行列をベクトル単位で操作し，lapply() はベクトル，データフレーム，リストを対象とする．mapply(),sapply() は lapply() を拡張ないし単純化したものである．以下では Crawley (2007, p. 18) に基づき説明する．

```
  # Crawley 教授のサイトからデータを取得
daphnia <- read.table(
  "http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/daphnia.txt",
   header = T)
head(daphnia)
attach(daphnia)
# Growth.rate の平均を Detergent の種類ごとに求める
```

```
tapply(Growth.rate, Detergent, mean)

# 二次元分割表を作るには要因をリストとして指定
tapply(Growth.rate, list(Water, Detergent), median)

detach(daphnia)
```

8 データの要約と視覚化

8.1 データの要約

ここで，実際のデータを例に，データの要約，グラフィックスによる表現を見ていく．R の基本パッケージには幾つかのサンプルデータが含まれているので，それらのデータを例に，探索的なデータ解析を行ってみる．ここでは 3 種類のアヤメの品種ごとに，花びらの幅と長さ，また，がくの幅と長さを記録したデータである iris を取り上げる．

```
  # data(iris)
?iris           # データの説明を見る
# iris          # データ全体を見る．サンプル数，変数が多い場合，見通しが悪い
summary(iris)   # データの要約を見る．四つの連続量の変数と，一つのカテゴリ変数がある

cov(iris[, -5])                             # 共分散行列．var(iris[,-5] でも同じ
var(iris[, colnames(iris) != "Species"])    # 上に同じ．条件の指定方法を変更
cor(iris[sapply(iris, is.numeric)])         # 相関行列．条件の指定方法を変更
```

sapply() 関数は，第 1 引数で指定したオブジェクトに，第 2 引数で指定した関数を適用する．ここでは iris データの各列が数値であるかを確認し，数値データである列だけを対象に相関行列を計算している．

8.2 データの視覚化

ここで iris データをグラフで表現してみる．

8.2.1 散布図行列

単純にオブジェクト名 iris を関数 plot() の引数として実行すると，**散布図行列**が出力される．

```
  plot(iris)                          # 散布図行列の作成
pairs(iris, panel = panel.smooth)  # 同じだがノンパラメトリックな曲線を追加
```

多変量データの散布図行列は，それぞれの変数間の関係を見るのに便利だが，ここでもう少し工夫をする．iris はアヤメの 3 種類の品種ごとのデータなので，品種ごとにプロットの色を変えてみる．

```
  # 散布図行列の改良
attach(iris)                          # データをワークスペースに登録

is.factor(Species)                    # Species 列は factor なので数値化可能
(species.n <- as.numeric(Species)) # 品種名を数値に変える

plot(iris, col = species.n)           # 色は数値で指定可能
```

8.2.2 箱ヒゲ図の作成

各変数ごとに三つの品種の平均 (中央値) の違い，また分布幅を見てみる．箱ヒゲ図では，中央の箱の中にデータの 50 % が含まれ，中央の線は中央値を表す位置であり，データを 25% ずつに分ける位置である．箱の上 (上側ヒンジ) からは「ヒゲ」が，「第 1 四分位数 - 1.5 × 四分範囲」に含まれるデータの最大値まで伸び，その位置で水平に直線が引かれている．下側も同様である．なおここで四分範囲あるいはヒンジ散布度は，第 3 四分位数と第 1 四分位数の差である (あるいは上側ヒンジと下側ヒンジの差である)．

ここでは四つの測定値があるので，プロットを四分割し，同時に表示してみる．R で図を分割するには幾つかの方法がある．柔軟な分割を行うには layout() を利用するが，例えば 2 × 2 と対称に区切るので十分であれば，par() 関数を利用するのがもっとも簡単である．

```
  par(mfrow = c(2, 2)) # 画面を四つに分割
for(i in 1:4){       # for ループを使って，四つのグラフを順に描いていく
  boxplot(iris[, i] ~ Species, main = colnames(iris)[i])
}

dev.off()

# layout を使うなら
layout(matrix(c(1,2,3,4), 2, 2, byrow = TRUE), c(2,1), c(1,2), TRUE)
layout.show(4)           # 分割の結果を確認

for(i in 1:4){
  boxplot(iris[, i] ~ Species, main = colnames(iris)[i])
}

dev.off()
detach(iris)
```

8.2.3 coplot

ある変数と目的変数の関係が，別の説明変数に条件を付けた場合にどのように変化するかを探る．

```
  # 条件付きプロットの作成．library(lattice) による
ozone.data <- read.table(
  "http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/ozone.data.txt",
   header = T)
attach(ozone.data); names(ozone.data)
coplot(ozone ~ wind|temp, panel = panel.smooth)

dev.off()
detach(ozone.data)
```

作成された図で，下に並んだ散布図の左端下の図は，気温 (華氏) がもっとも低い場合の ozone 濃度と風速の関係であり，右上端の図は気温が一番高い場合である． 温度が低い場合には相関が内容に思えるが，気温が高くなるにつれて負の相関が生じているように見える．

8.2.4 sunflower プロット

離散値のデータなどで，そのまま散布図にしては，多くの点がかな去ってしまうような場合に役に立つグラフィックス．あるいは x, y の散布図に，x と y それぞれの組み合わせ場合の頻度を表すようなグラフを作成できる．

```
  numbers <- read.table(
 "http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/longdata.txt",
  header = TRUE)
attach(numbers); names(numbers)
length(xlong)  # データ数は 1275 個だが
# 普通にプロットすると点が重なってしまう
plot(ylong ~ xlong)
# そこで，重なっているデータの個数の多さを表現したグラフを作成
sunflowerplot(ylong ~ xlong)

dev.off()
detach(numbers)
```

8.3 グラフィックスのパラメータ指定

ここで少し複雑だが，『R の基礎とプログラミング技法』第 8 章で紹介されている図を作成してみる．

```
  # 始めに正規乱数を 100 個生成
x <- rnorm(100)
par(las = 1) # 軸の目盛を水平に
hist(x, main = "N(0, 1) に従う 100 個の乱数の分布", # ヒストグラムの作成
     freq = F, col = "green", ylab = "確率密度",
     xlim = c(-4, 4), ylim = c(0,.6)) # 軸の範囲を指定
curve(dnorm, from = -4, to = 4, add = TRUE,  # add はグラフへの上書きを許可
       lwd = 3, lty = 2, col = "red")
# 凡例の作成
legend(-4, .55, legend= c("経験分布", "理論分布"),
       col = c("green", "red"), lwd = 5)
# 図にキャプションを加える
text(-4, .3, adj=0, cex = 1.3,
     expression(f(x) ==  # この部分は数式にギリシャ文字を使うための書式
         frac(1, sigma * sqrt(2*pi))
            ~~ e^{frac(-(x - mu)^2, 2 * sigma^2)}))
# 図にキャプションを加える
text(4, .3, adj=1, cex = 1.2,
      expression(paste("パラメータ ", mu == 0, ", ", sigma == 1)))

dev.off()
```

expression() 関数や substitute() 関数をラベルを指定する引数として使い，数式などの記号をグラフに追加することができる．これらについては ?legend などを参照すると，幾らかの実例が掲載されている．

なお R の解説書では，図を作成する際に，引数に type ="n" を指定していることがある．この場合，図のレイアウト部分だけが設定され，データの描画は行わない．これは例えば，実測値ではなく，理論曲線のみを散布図に描く場合などに使われるテクニックである．

```
  # type = "n" の効果を確認
par(mfrow = c(2,1))       # 画面を 2 分割
plot(cars)                # 通常のプロット
plot(cars, type = "n")    # プロットの輪郭だけ作成し
lines(lowess(cars))       # ノンパラメトリックな平滑曲線を追加

dev.off()
```

8.4 図のマージンの設定

プロットのマージン，作画領域を理解するため，『R の基礎とプログラミング技法』 p.169 の図を一部改編の上，ここに再現する．かなり複雑なコードになるが，プロットのレイアウトの詳細を知る上で参考になろう．なおプロット全体はデバイス領域といわれ，この領域の (外部) マージンを除いた残りの領域内に作図が行われる．また作図領域はさらに (内部) マージンとプロット領域に分けて考えることができる．

```
  # 『R の基礎とプログラミング技法』 p.169
plot(c(0, 1), c(0, 1))                    # 基本となる図

X11()                                     # もう一枚ウィンドウを広げて
par(oma = rep(3, 4), bg = "gray")         # 四つの外部マージンを 3 行分の高さに
plot(c(0, 1), c(0, 1), type="n", ann = FALSE, axes = FALSE) # 先ほどの図
par(xpd = TRUE)                           # 作図領域に加工を加える
# 作図領域を黄色に塗る
rect(par()$usr[1] - par()$mai[2], par()$usr[3] - par()$mai[1],
     par()$usr[2]  +  par()$mai[4], par()$usr[4] + par()$mai[3],
     col = "yellow", border = NA)
box("figure")                             # 作図領域全体を黒枠で囲む
par(xpd = FALSE)                          # 描画の対象をプロット領域に戻す
# プロット領域を白く塗りつぶす
rect(par()$usr[1], par()$usr[3],
     par()$usr[2], par()$usr[4],
     col = "white", border = NA)
box("plot", lty = "dashed", col = "green") # プロット領域を囲む

text(.5, .5, "plot region", cex = 1.6)

# 四つの内部マージンに描画
mtext("figure region", side = 3, line = 2, adj = 1, cex = 1.4)
for (i in 1:4)
    mtext(paste("inner margin", i), side = i,
          las = 0, line = 1, outer = FALSE)
# 四つの外部マージンに描画 # outer = TRUE
for (i in 1:4)
    mtext(paste("outer margin", i), side = i,
          las = 0, line = 1, outer = TRUE)
# 外部マージンにラベルを付ける
mtext("device region", side=3, line=2, outer = TRUE, adj = 1, cex = 1.2)
```

```
# 最後にプロット全体を赤い枠で囲む
box("outer", col = "red", lwd = 3)

# axis(1)        # これまで作成した図と重なるが，x 軸の目盛を描く
# axis(2)        # y 軸の目盛を描く

dev.off()
```

ここでは oma を使って外部マージン (outer margin) を行の高さを基準にしているが，omi を使えばインチで指定もできる．同じように mar が (内部) マージンを行の高さで，mai がインチで指定するものである．

8.5 図の保存

図を保存する場合, まずフォーマットを選ばなければならない．ビットマップ画像で問題なければ, Windows の場合，単に図の上で右クリックし，クリップボードにコピーする．これによって ワープロソフトに直接ペーストすることができる．

R のコマンドから直接ファイル形式とファイル名を指定して保存することも可能である．ここでは eps 形式，png 形式と pdf 形式を紹介する．図に日本語が含まれている場合は，pdf 形式にしておくのが無難である．pdf 形式に変換した「図」もワープロソフトで画像として取り込むことができる．

各種形式で図をファイルに書き込む場合，これまでと同様に作図した後で次を実行する．

```
  # たったいま作成したマージン設定の図を保存
dev.copy2eps(file = "margin.eps")                # eps 形式．TeX などで使う
dev.print(device = png, file="margin.png",       # png 形式．ウェブ用に
          width = 480, height = 480 )            # サイズはピクセルで指定
dev.print(device = pdf, file = "margin.pdf")     # pdf 形式で画像を保存

dev.off()                                        # デバイスの終了

# キャプションに日本語を含むグラフィックスを pdf 化する場合
# ps.options(family= "Japan1") をあらかじめ実行しておく
```

この他，例えば pdf ファイルを作成する方法としては，始めに pdf(file = "new.pdf") を実行し，続けて作図を行う．この場合，新たにウィンドウが開いて図が表示されることはない．作図のコマンドをすべて終了した後，dev.off() を実行すると，ファイルへの書き込みが完了する．明示的にデバイスを閉じない場合，ファイルへ正しく書き込みが行われず，図も利用できない．png() の場合も同様に作成できる．

ただし pdf 化する図中に日本語を使う場合には，あらかじめ ps.options(family= "Japan1") を実行する必要がある．毎回入力するのが面倒であれば，R をインストールしたフォルダの etc/Rprofile.site に下記を追記する．あるいはホームフォルダの中に .Rprofile という名前のファイルを作成し，その中に以下を

書き込んでおく．

```
setHook(packageEvent("grDevices", "onLoad"),
      function(...) grDevices::ps.options(family="Japan1"))
```

なおこのファイルの中に書かれたコードは，R の起動時に実行される．起動と同時に処理しておきたいコマンド類があれば，ここに追記しておくと便利である．

8.6 拡張グラフィックス

R のグラフィックス機能をベースに，これを拡張したパッケージが R 本体に追加されている．使い方はベースのグラフィックスとそれほど変わらない．代表的な拡張が **lattice** パッケージである．ここでは **lattice** パッケージによるグラフィックス作成を紹介する．

```
  # 高度なグラフィックス作成 lattice パッケージ
library(lattice)

# iris データを使って，品種ごとに花びらとがくの散布図を描く
xyplot(Petal.Length ~ Sepal.Length | Species, data = iris)
#  この構文で | の右辺の変数名が条件となる

bwplot(Petal.Length ~ Species, data = iris) # lattice による箱ひげ図

# 3 D 風のグラフィックス
cloud(Sepal.Length ~ Petal.Length * Petal.Width | Species,
      data = iris)


  # xyplot の利用場面．Crawley 2007, p.314 より
rm(x, y)                # 念のためワークスペースを掃除しておく

productivity <- read.table(
  "http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/productivity.txt",
   header = TRUE)
attach(productivity)
names(productivity)    # f 森の区域，y 森の生産性，x そこに暮らす哺乳類の種類
plot(x, y, xlab = "Productivity", ylab = "Mammal species")

# 図から高い相関が認められるが，これは有意か？
cor.test(x, y, method = "spearman")  # 順位相関を検定する
library(lattice)
```

```
xyplot(y ~ x | f)        # 森の区域別にグラフにすると負の相関が見て取れる
```

この他，R のグラフィックスの可能性については，http://addictedtor.free.fr/graphiques/ などのウェブサイトを参照されたい.

9 午後の部 2 ：データの解析

ここで基本的な統計解析手法を R で実現してみる.

9.1 t 検定

t 検定は t.test() 関数で実現する．この際，回帰分析などで使われるモデル式を同じ構文 Y ~ X が利用できることに注目されたい.

```
  sleep # 学生の睡眠データ．二つの睡眠促進剤を 10 人の学生に投与

# 対応がある場合
t.test(extra ~ group, data = sleep, paired = TRUE)
# 対応がない場合
t.test(extra ~ group, data = sleep)
```

9.2 カイ二乗検定

カイ二乗検定は chisq.test() 関数を利用する.

```
  # ここでは  UCBAdmissions データを利用する
UCBAdmissions              # 多次元分割表だが Dept A のみ利用
UCBAdmissions[, , "A"]
chisq.test(UCBAdmissions[, , "A"])
fisher.test(UCBAdmissions[, , "A"]) # fisher の正確確率
```

なお，ここで利用したデータは多次元分割表であるが，これに対数線形モデルあるいはログリニア分析を行うには loglin(),loglm() を利用する．今回は取り上げないが，興味のある方は以下のコードを参考にされたい.

```
  # 参考：対数線形モデル，ログリニア解析
# 飽和モデル
loglin(UCBAdmissions, list(c(1,2,3)) )
# 三次の効果を外したモデル
loglin(UCBAdmissions, list(c(1,2), c(2,3), c(1,3)), param = TRUE)

library(MASS)
# 飽和モデル
USB.lglm <- loglm(~ Gender * Admit * Dept, data = UCBAdmissions)
```

```
USB.lglm2 <- update(USB.lglm,~.- Gender:Admit:Dept,
                    data = UCBAdmissions)
stepAIC(USB.lglm2, ~.^2,data = UCBAdmissions)
```

9.3 単回帰分析

R に組込みのデータを使って単回帰分析を実行してみよう．データオブジェクト cars は，車が停止するまでに必要とした距離 (フィート) と，制御をかけた時点のスピード (マイル) を記録したものである．

```
  # 単回帰
cars
cars.lm <- lm(dist ~ speed, data = cars) # モデル式
plot(dist ~ speed, data = cars)
abline(cars.lm)
summary(cars.lm)          # さほど当てはめは良くない
par(mfrow = c(2, 2))      # モデル診断プロットを描画する用意
plot(cars.lm)             # データにやや規則的な変動がある

dev.off()
```

9.4 一元配置の分散分析

ここで『分散分析のはなし』(石村, 1992, p. 77) 掲載のアフリカツメガエルの細胞分裂割合データを借用する．また始めにデータを横方向に長い形式で作成し，これを縦長の形式に変更する方法を確認する．特に時系列に従って記録されたデータでは，このようなフォーマットがしばしば利用されているので，変換方法を習得しておくのは有用だと思われる．

```
  # データフレームの作成
#
frogs <- read.table("http://150.59.18.68/frogs.csv", header = TRUE)
frogs

################     以下 8 行は参考     ######################
# 始めに横長形式のデータとして与えられていたとする
# frogs <- data.frame(stage = c("A1", "A2", "A3", "A4", "A5"),
#                     sample1 = c(12.2, 22.2, 20.8, 26.4, 24.5),
#                     sample2 = c(18.8, 20.5, 19.5, 32.6, 21.2),
#                     sample3 = c(18.2, 14.6, 26.3, 31.3, 22.4))
```

```
# write.table(frogs, file = "frogs.cvs", row.names = FALSE)
# rm(frogs)
# frogs <- read.table( file = "frogs.cvs", header = TRUE)
###############    以上は参考    ############################

# データを縦長形式に変更する．分散分析の準備
frogs.res <- reshape(frogs, idvar = "stage", # stage 変数がカテゴリを表す id
                     varying = list(names(frogs[2:4])), # 数値データ
                     v.names = "data", direction = "long") # 新しい変数名
frogs2 <-  frogs.res[order(frogs.res$stage),]     # カテゴリを優先して並び替え
frogs2
# ここで作成される time 変数は，各数値の順番を表し
# 時間ごとにデータを取っている場合には重要な情報である
frogs.aov <- aov(frogs2$data ~ frogs2$stage)  # 分散分析を実行し
summary(frogs.aov)
# anova(lm(frogs2$data ~ frogs2$stage))       # 上記と同じこと
# この段階で水準に差があることが分かる

# 多重比較を実施する
pairwise.t.test(frogs2$data , frogs2$stage)
# デフォルトは holm の方法．?p.adjust で可能な方法の説明が参照できる
plot(TukeyHSD(aov(frogs2$data ~ frogs2$stage)) ) # Tukey の方法の結果をグラフ化
```

R の分散分析の仕組み，また結果を正確に解釈するには「コントラスト」について理解しておく必要がある．ここでは詳しく述べないが，詳細は Crawley (2007, pp. 364–386) あるいは Crawley (2002, pp. 173–174, pp. 209–226, pp. 323–344)，Everitt and Hothorn (2006, p. 75)，Fox (2002, pp. 127–153) などを参照されたい．

なお正規分布を仮定しない場合の検定手法として kruskal.test() が用意されている．

9.4.1 練習：一元配置の分散分析

```
  # R 組込みの InsectSprays を対象に一元配置の分散分析を実行する
InsectSprays
bartlett.test(count ~ spray, data = InsectSprays) # 多群の等分散性
incSpr.aov <- aov(count ~ spray, data = InsectSprays)
summary(incSpr.aov)                               # spray に効果
summary.lm(incSpr.aov)

# treatment contrasts の場合の標準偏差．各水準の繰り返し数は 12
sqrt(15.38/12)              # Intercept の平均の標準誤差
sqrt(15.38/12 * 2)          # 平均の「差」の標準誤差
```

```
pairwise.t.test(InsectSprays$count, InsectSprays$spray)
plot(TukeyHSD(aov(count ~ spray, data = InsectSprays)))
```

ごく簡単にコントラスト treatment.contr に触れておくと R で分散分析を実行し，その結果から各水準の係数を summary.lm() で出力した場合，各水準の有意度は intercept と比較した場合を想定しており，前後の水準との比較では無い．さらにこの表で intercept は factor の (アルファベット順で) 最初の水準の平均に設定されており，全平均ということでは無い．

9.4.2 参考：コントラストについて

```
  # treatment contrasts を理解する
comp <- read.table(
  "http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/competition.txt",
  header = TRUE)
attach(comp)
comp.aov <- aov(biomass ~ clipping)
summary(comp.aov )
summary.lm(comp.aov)
# 上の出力の内容を確認
(means <- tapply(biomass, clipping, mean))
means - means[1] # control 群の平均を引く．これが「効果」

# 標準誤差 Std.Err の意味
# Intercept の誤差は，control 群の平均の誤差
# プールされた平均誤差分散を control の繰り返し数で割って平方根を取る
sqrt(4961/6)
# 2 行目以降の標準誤差は平均の「差」の誤差
# 水準が独立であれば，平均の差の分散は，二つの水準の分散の和
sqrt(2 * 4916/6)

detach(comp)
```

9.4.3 参考：コントラストをマニュアルで変更する

ここでは Fox に基づいて，次のような課題を分析してみる．66 人の幼児をランダムに三つのグループに分け，それぞれ異なった教授法で読み方を教えたとする．その上で，読み方の試験 (post.test.3) を実施し，標準的な教授法 (Basel) と，他の二つの新しい方法 (DRTA，Strat) とに効果の差があるか，また新しい二つの方法の間に差があるかを，一度に検定することにする．

```
  # Fox p.143 より
Baumann <- read.table("http://150.59.18.68/Baumann.txt")
attach(Baumann)
baumann.aov1 <- aov(post.test.3 ~ group)
summary.lm(baumann.aov1)      # 通常のコントラストによる解析
                              # これは Basel と他の水準の差
contrasts(group) <- matrix(c(1,-0.5,-0.5,  0,-1,1), 3,2)
contrasts(group)              # 直交するコントラストを作成
baumann.aov2 <- aov(post.test.3 ~ group)
summary.lm(baumann.aov2)
# 標準的方法と最新の方法の間には差があるが
# 最新の二つの方法の間には差は無い
```

9.5 二元配置の分散分析

ここでも『分散分析のはなし』から，反復のある二元配置の実験結果 p.166 の表を借用する．この場合，交互作用を調べることができるので，その手順を紹介する．

```
  # 二元配置の分散分析
frogs3 <- read.csv("http://150.59.18.68/frogs3.csv", header = FALSE)
frogs3                    # header = FALSE で，列名はファイルに未設定と指示
# なお列名が未定義の場合，自動的に V1, V2, V3 などの変数名が付加される
# 二つの要因がある場合，それらをチルダ記号の右側に + 記号で指定する
frogs3.aov <- aov(V1 ~ V2 + V3, data = frogs3)
summary(frogs3.aov)

# 交互作用を組み入れる
frogs3.aov2 <- aov(V1 ~ V2 + V3 +  V2:V3, data = frogs3)
# frog3.aov2 <- aov(V1 ~ V2*V3, data = frogs3)        # 上と同じ
# frog3.aov2 <- aov(V1 ~ (V2+V3)^2, data = frogs3)    # 上と同じ

summary(frogs3.aov2) # 交互作用が有意であることが分かる．
step(frogs3.aov2)    # 説明は省くが，モデルにどのような項を入れるかは
                     # R に判定させることもできる
```

9.5.1 練習：二元配置の分散分析

```
  # data(ToothGrowth) モルモットの歯の成長に対するビタミン C の効果
head(ToothGrowth)
TG.aov <- aov(len ~ supp * dose, data = ToothGrowth)
summary(TG.aov)  # 交互作用は有意である
interaction.plot( ToothGrowth$supp,  ToothGrowth$dose,  ToothGrowth$len)
```

9.6 分散分析での変量モデル，ネストモデル

ここでは『違いを見抜く統計学』(豊田, 1992) から，p.120, p.127 のデータを借用する．

```
  # csv ではなく，R のコードとしてデータが与えられている場合もある
p120 <- source("http://150.59.18.68/toyoda.p120.R")$value
#################  toyoda.p120.R ファイルの中身 ####################
## data.frame(type = rep(c("sea","mount"), each = 8),
##            develop = rep(rep(c("old","new"), c(4,4)),2),
##            region = rep(c("izu","kuju","karui","yatsu"), each=4),
##            price = c(18.0, 15.0, 12.0, 20.7, 7.6, 12.3, 15.0, 15.2,
##              11.7, 11.6, 14.5, 15.4, 7.8, 7.5, 4.2, 6.4))
#######  別荘地のブランド，開発時期とタイプ（海か山）を母数因子として考える

# 交互作用を含むモデル
p120.aov <- aov(price ~ type * develop, data = p120)
summary(p120.aov) # 交互作用は有意ではない

# 交互作用を含まないモデル
p120.aov2 <- aov(price ~ type + develop, data = p120)
summary(p120.aov2)

# 交互作用を変量としたモデル．豊田 p.121
p120.aov3 <- aov(price ~ type + develop + Error(type:develop),
                 data = p120)    # Error() 項に警告が出るがここでは無視
summary(p120.aov3)               # type も develop も有意ではない

8.556 / 8.270                    # 交互作用の F 値
1 - pf(8.556 / 8.270, 1, 12)     # p 値を求める
# pf(8.556 / 8.270, 1, 12, lower = FALSE) でも良い
```

```
# ネストを仮定したモデル．すなわち地域はタイプにネストしている．豊田 p.127
p120.aov4 <- aov(price ~ type + Error(region %in% type),
                 data = p120)   # 警告が出るがここでは無視してよい
## p120.aov4 <- aov(price ~ type + Error(region / type),
##                  data = p120) # でも良い
summary(p120.aov4)              # タイプは有意ではない

# ではネストした地域は有意か．以下参考までに
region.ratio <- summary(p120.aov4)[[1]][[1]][2, "Mean Sq"] /
       summary(p120.aov4)[[2]][[1]]$"Mean Sq"
# つまり 61.791 /  8.270 を計算して region.ratio に代入
1 - pf(region.ratio , 2, 12)
```

9.7 共分散分析

説明変数にカテゴリ変数と連続値の変数を含むデータの解析を扱う．例えば体重を性別 (カテゴリ) と年齢 (連続量とする)．この場合，男女それぞれに切片と傾きの四つのパラメーターを推定することになる．各パラメータの計算手順，またその標準誤差の計算方法については (Crawley, 2007, pp. 492–497) に詳細に説明されている．

```
  # Crawley 2007, p.489 より
# 植物の果実の生産力．説明変数は grazing の有無．実験開始前の植物のサイズ
regrowth <- read.table(
  "http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/ipomopsis.txt",
  header = T)
attach(regrowth )
names(regrowth )

plot(Fruit ~ Root, pch = 16 + as.numeric(Grazing),
      col = c("blue", "red")[as.numeric(Grazing)])
abline(lm(Fruit[Grazing == "Grazed"] ~ Root[Grazing == "Grazed"]),
       lty = 2, col = "blue")
abline(lm(Fruit[Grazing == "Ungrazed"] ~ Root[Grazing == "Ungrazed"]),
        lty = 2, col = "red")

tapply(Fruit, Grazing, mean)   # Grazed の方が生産力が高い？
t.test(Fruit ~ Grazing)

ancova1 <- lm(Fruit ~ Grazing * Root)
```

```
summary(ancova1)                # パラメータを見る
anova(ancova1)                  # 要因の効果は？

# 交互作用は有意でないので省く
ancova2 <- update(ancova1, ~. -Grazing:Root)
anova(ancova1, ancova2)         # 交差項を削除して差し支えない
ancova3 <- update(ancova2, ~. -Grazing)
anova(ancova3, ancova2)         # Grazing は削除できない
summary(ancova2)                # Grazing は約 36 mg 生産力を「下げる」
                                # (Intercept が Ungrazed の場合なので)
```

10　多変量データ解析

ここからは『R と S-PLUS による多変量解析』(ブライアン, 2007) を参考に，各種解析技法を実際に実行する．

10.1　重回帰分析

ここでは「通常の」線形回帰，すなわち目的変数の誤差は正規分布に従い，その分散は一定である場合の重回帰分析を検討する．これ以外の場合，例えば誤差が正規以外の分布に従う，分散が一定でない，そもそも分布が不明などの場合には，一般化線形モデルや一般化加法モデル，混合効果モデルなどが検討されるべきである．これらの分析のために R では `glm()`，`gam()`，`lme()` などの関数群が用意されているが，詳細は Faraway (2005)，Wood (2006),『R と S-PLUS による多変量解析』(ブライアン, 2007) を参照されたい．

実際のデータを使って分析を行ってみる．データは『R と S-PLUS による多変量解析』 p. 167 で取り上げられている大気汚染データである．これは再び米国における大気汚染に関するデータで，41 の都市について次の七つの変数が記録されている．データの変数名は以下の通りである．

SO2　:大気中の二酸化硫黄の含有量（マイクログラム/立方メートル）
Temp　: 年間平均気温（華氏）
Manuf　: 20 人以上を雇用する製造業者の数
Pop　: 住民数（1970 年の国勢調査に基づく）（千人単位）
Wind　: 年間平均風速（マイル/時間）
Precip　: 年間平均降水量（インチ）
Days　: 降水のあった日数の年間平均

```
  # 『R と S-PLUS による多変量解析』 からデータを借用
usair.dat  <- source("http://150.59.18.68/chap3usair.dat")$value
plot(usair.dat[, -1]) # 目的変数を除いた残りの六つの変数の散布図行列
attach(usair.dat)
# SO2 を，他の変数で説明する
usair.fit <- lm(SO2 ~ Neg.Temp + Manuf + Pop + Wind + Precip + Days)
summary(usair.fit)           # 結果を見る

# update を使って再分析する
usair.fit2 <- update(usair.fit, ~. -Days)  # Days を削って分析し直す
summary(usair.fit2)

# さらに Wind を削って分析
usair.fit3 <- update(usair.fit2, ~. - Wind)
summary(usair.fit3)
```

```
# さらに Precip を削って分析
usair.fit4 <- update(usair.fit3, ~. - Precip)
summary(usair.fit4)

# 最後に Neg.Temp　を削って分析
usair.fit5 <- update(usair.fit4, ~. - Neg.Temp)
summary(usair.fit5)
```

10.1.1　練習

Crawley (2007, p. 434) から，やはり大気汚染データを使って重回帰分析の練習を行う．目的変数はオゾン濃度，説明変数の候補は風速，気温，放射熱とする．なおこのデータには非線形性が認められるが，ここでは変数の 2 次の項を導入することで重回帰分析を行う．モデル式で 2 次項 x^2 をふくめる場合，その式は `I(x^2)` として表さなければならない．モデル式では `^2` は 2 次の交互作用を意味するからである．

```
  # Crawley 教授のサイトよりデータを借用
ozone.data <- read.table(
 "http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/ozone.data.txt",
  header = T)
names(ozone.data)                          # 変数名を確認
pairs(ozone.data, panel = panel.smooth)  # 非線形性を認める

# 説明変数の 2 次の項を導入することで重回帰モデルを当てはめてみる
ozone.model1 <- lm(ozone ~ temp * wind * rad
                   + I(rad^2) + I(temp^2) + I(wind^2),
                   data = ozone.data)
summary(ozone.model1)                      # 3 次の交互作用は有意でない
ozone.model2 <- update(ozone.model1, ~. -temp:wind:rad)
summary(ozone.model2)                      # temp:rad が有意でない

# このまま分析者が適宜判断を行いながら解析を続ける，もしくは R に任せる
ozone.step <- step(ozone.model1)                  # R に任せる場合
summary(ozone.step)                               # ただし R の判断は「甘い」
```

10.2　ロジスティック回帰分析

ここでは 2 値データを目的変数とする回帰分析を紹介する．この節の内容またデータは Faraway (2005, p. 32) に基づく．

データは幼児の呼吸器系疾患の発生をモデル化したもので，要因は性別とミルク (母乳，粉末ミルク，母乳とサプリメント) である．

```
  # Faraway 2006, p.32 よりデータを借用
babyfood <- read.table(file = "http://150.59.18.68/babyfood.txt")
babyfood
# データから要因別に罹患比率を求めて分割表にする．xtabs() 関数を利用
xtabs(disease/(disease+nondisease) ~ sex + food, data = babyfood)

# ロジスティック回帰分析を実行する
model1 <- glm(cbind(disease, nondisease) ~
           sex + food, family = binomial, data = babyfood)
# glm は一般化線形モデルを実行する関数．family は分布を指定する
summary(model1)              # 要約を見る
drop1(model1, test = "Chi")  # 各項は有意か
exp(-.669)                   # 母乳の効果を確認する
```

summary() 関数の出力で Residual Deviance は現在のモデルの deviance を表しており，ここではフルモデル (要因の交互作用を含む場合) との差を表しており，目的変数が binomial で観測数が十分に多ければ，カイ二乗分布で漸近的に近似される．大雑把な見方としては，Residual Deviance が自由度よりもかなり大きな数値でなければフィットしていると考えて構わない．つまり，この例では (上の解析操作では明示的には含めていない) 交互作用は有意ではない．

drop1() 関数は，モデルから項を一つずつ取り去り，フルモデルと比較する．すなわち，その項の説明力をチェックする．この場合，二つとも必要と見なされる．

ここで母乳の効果を見てみる．summary(mdl) の係数は対数オッズであることに注意する．ここでは母乳は，呼吸器疾患のリスクを粉末ミルクの場合に比べて 51 % に減らすことを意味する．

10.3　主成分分析

R で主成分分析を行うには二つの関数が利用できる．princomp() と prcomp() である．前者は固有値を利用し，後者は特異値分解を利用する．ここでは princomp() を使った分析を紹介する．

```
  # 重回帰分析で取り上げた大気汚染データを使う
cor(usair.dat[,-1])              # SO2 の要因となる六つの変数の相関行列
# 共分散行列ではなく，相関行列をもとに主成分分析を行う
usair.pc <- princomp(usair.dat[,-1], cor = TRUE)
# 因子負荷量を含めて結果を出力する
summary(usair.pc, loadings = TRUE)
plot(usair.pc, type = "l")       # スクリープロットで主成分の寄与率を確認
```

結果を見ると，最初の三つの主成分の分散 (固有値) で，データが持つ分散 (情報量) の 85% を説明しているが分かる．この三つの主成分について，Everitt の解釈をそえておく．第 1 主成分では Manuf と Pop の負荷量が高いことから「生活の質」を，また第 2 主成分は Precip と Days の負荷量が高いので「雨天」，さらに第 3 主成分は Neg.Temp と Precip の対比が著しいので「気候タイプ」とラベル付けできなくもない．

主成分分析の主な長所は，もとのデータをより低い次元で表現できることである．第 2 主成分と第 3 主成分を使って散布図を描いてみる．

```
  # 第 2 主成分得点と第 3 主成分得点を使って散布図を描く
plot(usair.pc$scores[,2],usair.pc$scores[, 3],
     xlab = "PC1", ylab = "PC2", type = "n")    # まだプロットしない
text(usair.pc$scores[,2],usair.pc$scores[, 3],
     labels = abbreviate(row.names(usair.dat)), # 都市名の略語をプロット
     cex = 0.7, lwd = 2)

# バイプロットを描く
biplot(usair.pc)

detach(usair.pc)
```

最初のプロットでは，左上に位置するバッファローは，降水量と気候タイプの両方に負荷が高く，「乾燥した気候を好むのであれば避けた方が良い」．バイプロットでは変数は赤い矢印で描かれている．方向と長さは，それぞれ対応する x 軸と y 軸，つまり第 1 主成分得点と第 2 主成分得点に対する相対負荷量を表している．

なお prcomp() を利用した場合の負荷量は $rotation で抽出できる．

10.4　因子分析

R で探索的 (記述的) 因子分析を実行するのに使われる関数は factanal() である．この関数はデータ行列あるいは共分散行列を対象に，最尤推定法によるパラメータ推定を行う手法である．なお回転手法としては直行回転の varimax 法と，斜交回転の promax 法を実現できる．これ以外の回転技法を必要とする場合には **GPArotation** パッケージを利用すれば良い．

ここでは『R と S-PLUS による多変量解析』第 4 章の解析例を紹介する．データは，国，年齢，性別ごとの 1960 年代の平均余命に関するものである．

```
  # 『R と S-PLUS による多変量解析』第 4 章
life <- source("http://150.59.18.68/chap4lifeexp.dat")$value
attach(life)
# 3 因子モデル
life.fa3 <- factanal(life, factors = 3, scores = "regression")
life.fa3                    # 結果を確認する
life.fa3$scores             # 因子得点を確認
```

出力の最後に，因子がさらに必要であるとする対立仮説に対する検定結果が得られる．この場合，因子の数 3 は十分であるとの結果を得る．この結果をもとに 3 次元のグラフを描いてみる．始めに **lattice** ライブラリを使ったプロットを紹介し，次に **rgl** ライブラリを使う方法を紹介する．**rgl** ライブラリはダウンロードとインストールが必要になるが，利用のコンピュータにその権限がない場合は，ユーザーのホームフォルダにインストールする．

```
  library(lattice)
# 3D 風のグラフを作成
cloud( life.fa3$scores[,"Factor3"] ~  # Z 軸に対して x 軸と y 軸を指定
       life.fa3$scores[,"Factor1"] * life.fa3$scores[,"Factor2"],
       pch = names(life.fa3$scores[,"Factor3"])) # 国名の頭文字がプロットされる
```

もう一つは，マウス操作で画像を回転させることができるよう工夫を凝らした 3D プロットである．

```
  # もしも E ドライブの users フォルダに書き込み権限があるならば
# .libPath("E:/users")     # ライブラリのインストール先を追加
# .libPath()               # 変更を確認
###### # .libPath(tempdir()) を使っても良い

# メニューから rgl  ライブラリをインストールして
library(rgl)
rgl.open()                          # デバイスを開く
rgl.bg(color=c("white","black")) # 背景色を指定

rgl.spheres( life.fa3$scores[,"Factor1"], # x 軸
             life.fa3$scores[,"Factor2"], # y 軸
             life.fa3$scores[,"Factor3"], # z 軸
             radius=0.1,                  # 球の大きさ
             color = 1: length(life.fa3$scores[,"Factor1"])) # 国ごとに色を変える

rgl.bbox(color="#112233", emission="#90ee90",
         specular="#556677", shinines=8,alpha=0.8) # 座標系の設定

rgl.texts( life.fa3$scores[,"Factor1"],   # 球の横に国名を略語で表示
           life.fa3$scores[,"Factor2"],
           life.fa3$scores[,"Factor3"],
           abbreviate( names(life.fa3$scores[,"Factor1"])),
           color =  1: length(life.fa3$scores[,"Factor1"]))
```

```
rgl.close()                                    # デバイスを閉じる
```

なお varimax 法と promax 法以外の回転技法を利用したい場合は **GPArotation** パッケージを導入する．そして，例えばオブリミン法を利用したい場合，次のように実行する．

```
  # 各種の回転技法
library(GPArotation)                           # このライブラリを導入すれば
life.fa3.ob <- factanal(life, factors = 3,
                    rotation = "oblimin",      # オブリミン法を指定できる
                    scores = "regression")
x11()                                          # 新たに描画ウィンドウを作成し
cloud( life.fa3.ob$scores[,"Factor3"] ~        # 先ほどの画像と比べる
       life.fa3.ob$scores[,"Factor1"] * life.fa3.ob$scores[,"Factor2"],
       pch = names(life.fa3.ob$scores[,"Factor3"]))
```

なお確証的因子分析用のパッケージとして CRAN に **sem** パッケージがある．

10.5　クラスター分析

ここでも平均余命データを例にクラスター分析を概観する．クラスターは「群」を意味し，この手法はデータの各個体を，幾つかの基準から複数の群に分類する手法である．ここでは次の三つの方法について論じる．

- 凝集型階層法
- k -平均法
- 分類最尤法

10.5.1　階層的方法

凝集型階層的クラスター分析は n 個の個体を次々と併合し，幾つかの群に分ける手法である．併合は，個体の非類似度を基準に行われる．非類似度を表す方法としては**最短距離法**あるいは**単連結法** (single linkage) や，**最長距離法**あるいは**完全連結法** (complete linkage)，**群平均法** (group average, average linkage) が知られている．以下に三つの方法による分類を示す．

```
  # ウィンドウを次々と開きながらプロットを作成する．
plclust(hclust(dist(life), method = "single"),
             labels = row.names(life), ylab = "Distance")
title("(a) Single linkage")
#
x11()
plclust(hclust(dist(life), method = "complete"),
```

```
                labels = row.names(life), ylab = "Distance")
title("(b) Complete linkage")
#
x11()
plclust(hclust(dist(life), method = "average"),
                labels = row.names(life), ylab = "Distance")
title("(c) Average linkage")
```

ここでは複数の関数を組み合わせて利用している．まず dist() は対象間のユークリッド距離を求め，これを hclust() 関数が指定された非類似度を基準としてクラスターに分類し，最後に plclust() 関数で「デンドログラム」を描いてる．

さてクラスターの分類は，個々の個体を次々と併合していく過程であるが，そのどこかで，研究者にとって意味のある「群分け」を選ぶ必要がある．デンドログラムで言えば，縦軸の「距離」の目盛のどこかで横に切って，「適切な」群の数を選ぶ必要がある．「適切な群の数」を選ぶ基準は後述するとして，ここでは最長距離法による分類を例に，距離位置として 21 を選び，ここで群に分けることを考える．

```
  # 21 で切った場合の各国の位置を調べる
cut.point <- cutree(hclust(dist(life), method = "complete"), h = 21)
# 結果をもとに国別に分類する
country.clus <- lapply(1:max(cut.point),
                       function(nc) row.names(life)[cut.point == nc])
# lapply はリストに指定された関数を適用する．
country.clus
```

10.5.2　非階層的方法

k 平均法では，個体を既知の k 個の群に分類する．この場合，各群の群内平方和が最小になるような組み合わせが選ばれる．ここでは『R と S-PLUS による多変量解析』pp. 132–138 から，ローマ軍がイギリスに駐屯していた時代の陶器の化学成分に関するデータを例に説明する．

```
  # ローマ時代のイギリスの陶器
pottery <- source("http://150.59.18.68/chap6pottery.dat")$value
# 各変数の尺度が異なるので基準化する．まず，それぞれについてレンジを求める
rge <- apply(pottery,2,max) - apply(pottery,2,min)
# 元データをレンジ (rge) で割る ( "/")．数値の 2 は列ごとに計算するよう指定
pottery.dat <- sweep(pottery, 2, rge, FUN="/")
n <- nrow(pottery.dat)              # 行数を取得
# 群分けしない場合の群内平方和を求める
wss1 <- (n-1)*sum(apply(pottery.dat, 2, var))
```

```
# 次に群数を 2 から 6 まで変化させながら，それぞれの群内平方和を求める
wss<-numeric(0)           # 群内平方和を保存するオブジェクトを準備し
for(i in 2:6) {           # ループで，kmeans 法を使って
                          # それぞれの場合の群内平方和を保存
   W <- sum(kmeans(pottery.dat, i)$withinss)
   wss <- c(wss, W)
}

# 群数 1 の場合 (wss1) と，2-6 の場合 (wss) を結合する
wss<-c(wss1,wss)
# 結果をプロットしてみる
plot(1:6, wss, type="l", xlab = "Number of groups",
     ylab = "Within groups sum of squares", lwd = 2)
```

このグラフのように，群内平方和が急激に減少するポイントを，適切な群数と考えることができる．そこで群数として 3 を指定して，詳細な解析結果を見る．

```
(pottery.kmean <- kmeans(pottery.dat, 3))
```

この調査では，陶器の窯の場所も記録されていたのだが，これらの窯を 1 から 5 の数値で表したベクトルを作成し，これをクラスター分析の結果と照合してみる．

```
  # 窯の場所を表すベクトル
# pottery の 1 行目から 21 行目までが窯番号 1 で，以下同様
kiln <- c(rep(1,21),rep(2,12),rep(3,2),rep(4,5),rep(5,5))
# 表形式でまとめる
table(kiln, pottery.kmean$cluster)
```

実は，この五つの窯は，それぞれ三つの地域に散らばっており，クラスター分析の結果は，その違いを十分に表している．

10.5.3　モデルに基づくクラスター分析

これについては **mlcust** パッケージを参照されたい．

10.6　対応分析

MASS ライブラリの corresp() 関数によって実現できる．ここでは簡単な実行例を挙げるにとどめる．

```
  data(caith)
corresp(caith)
```

```
dimnames(caith)[[2]] <- c("F", "R", "M", "D", "B")
par(mfcol = c(1, 3))
# library(mva)
plot(corresp(caith, nf = 2)); title("symmetric")
plot(corresp(caith, nf = 2), type = "rows"); title("rows")
plot(corresp(caith, nf = 2), type = "col"); title("columns")
par(mfrow = c(1, 1))
```

10.7 正準相関分析

正準相関分析は，目的変数を複数に拡張した重回帰分析だと考えれば良い．R では cancor() が備えられている．またこの関数を拡張した **CCA** パッケージも公開されている．ここでは簡単な実行例を挙げるにとどめる．

```
  data(LifeCycleSavings)
pop <- LifeCycleSavings[, 2:3]
oec <- LifeCycleSavings[, -(2:3)]
cancor(pop, oec)

x <- matrix(rnorm(150), 50, 3)
y <- matrix(rnorm(250), 50, 5)
(cxy <- cancor(x, y))
all(abs(cor(x %*% cxy$xcoef,
            y %*% cxy$ycoef)[,1:3] - diag(cxy $ cor)) < 1e-15)
all(abs(cor(x %*% cxy$xcoef) - diag(3)) < 1e-15)
all(abs(cor(y %*% cxy$ycoef) - diag(5)) < 1e-15)
```

参考文献

Crawley, Michael (2002) *Statistical Computing: An Introduction to Data Analysis Using S-Plus*: John Wiley & Sons.

——— (2007) *The R Book*: John Wiley & Sons.

Everitt, Brian S. and Torsten Hothorn (2006) *A Handbook of Statistical Analyses Using R*: Chapman & Hall/CRC.

Faraway, Julian J. (2005) *Extending the Linear Model with R*, Texts in Statistical Science: Chapman & Hall/CRC.

Fox, John (2002) *An R and S-Plus Companion to Applied Regression*: Sage Pubns.

石村貞夫 (1992) 『分散分析のはなし』，東京書籍.

Murrell, Paul (2005) *R Graphics (Computer Science and Data Analysis)*: Chapman & Hall/CRC.

豊田秀樹 (1992) 『違いを見抜く統計学―実験計画と分散分析入門』，講談社.

Verzani, John (2004) *Using R for Introductory Statistics*: Chapman & Hall/CRC.

Wood, Simon N. (2006) *Generalized Additive Models: An Introduction with R*: Chapman & Hall/CRC.

ウーヴェリゲス (2006) 『R の基礎とプログラミング技法』，シュプリンガー・ジャパン，第 3 版．石田 基広訳.

ブライアンエヴェリット (2007) 『R と S-PLUS による多変量解析』，シュプリンガー・ジャパン，第 1 版．石田 基広，石田 和枝訳.