第 8 章

データ解析の実例

データ解析の一般的な流れは，まず一変量解析を行うことから始まる。データに層や群があるなら，単に各変数の度数分布（平均値，標準偏差など）を把握するだけでなく，それらを区分することでデータに違いがあるのかないのかを確認する必要もある。図として表示したり，検定によって偶然の誤差なのか本質的な差なのかを検討することが必要である。

本章では，R に用意されている iris データセットを取り上げ，第 7 章で紹介した関数を使いながらデータ解析を行う実例を示す。

iris データセットは以下に示すようなデータフレームである。

- 5 個の変数を含む。
- Sepal.Length，Sepal.Width，Petal.Length，Petal.Width は計測値である。
- Species は，setosa，versicolor，virginica の 3 群を識別する factor 変数である。
- 各群は 50 個体ずつのデータで，全部で 150 個体のデータである。

```
> iris # iris データセット（データフレーム）
    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
⋮              ⋮           ⋮            ⋮           ⋮          ⋮
149          6.2         3.4          5.4         2.3  virginica
150          5.9         3.0          5.1         1.8  virginica
```

8.1　各変数の度数分布

iris[1:4] の, Sepal.Length, Sepal.Width, Petal.Length, Petal.Width の分布を確認する。まずは，iris[5] の Species を考慮せず，全体を 1 つの群として基本統計量と度数分布を求める。

最初に summary 関数で要約統計量を求めよう。この段階での基本統計量が与える情報は多くはないが，欠損値はあるのか，外れ値はないか，どのような範囲の値をとっているのかなど，今後の分析の土台や参考になるものである。

```
> summary(iris) # 要約統計量を計算する
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
 setosa    :50  # 3種, 各50個体ずつ
 versicolor:50  # 欠損値はない
 virginica :50

> sd(iris[1:4]) # 標準偏差を計算する
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
   0.8280661    0.4358663    1.7652982    0.7622377
```

次に，7.2 節に示した frequency 関数で，各変数の度数分布表とヒストグラムを求める。

```
> frequency(1:4, iris, output="iris-frequency.tex", encoding="euc-jp",
+           plot="iris-frequency", width=400, height=300)
```

表 8.1〜8.4，図 8.1〜8.4 に出力結果をまとめる。

表 8.1 および図 8.1 を見ると，Sepal.Length は一峰性ではあるが平均値の周りがなだらかな丘状の分布である。これは後の分析（図 8.5）でわかることであるが，3 つの群の分布が部分的に重なっているためである。

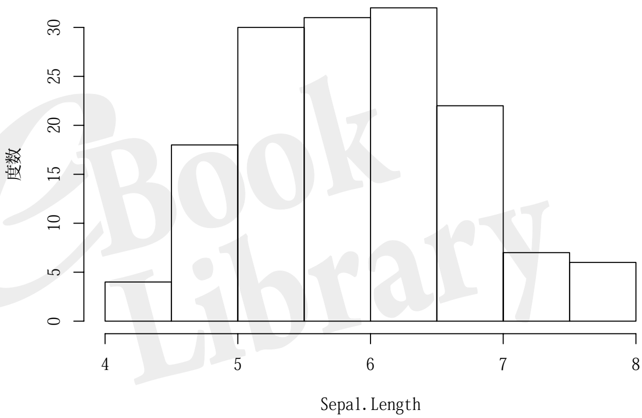
表 8.2 および図 8.2 を見ると，Sepal.width も一峰性ではあるが，Sepal.Length とは異なり，平均値付近に突出したピークを持つ分布である。また，右裾が長いことも特徴である。これも後の分析（図 8.6）でわかることであるが，setosa が他の 2 群よりやや大きい値を持つためである。

表 8.3 および図 8.3 を見ると，Petal.Length は二峰性の分布を示している。setosa はほかの 2 群から離れて存在していることがわかる。

表 8.4 および図 8.4 を見ると，Petal.Width は三峰性の分布を示している。setosa はほかの 2 群から離れて存在しており，versicolor と virginica も分布に違いがあることがわかる。

▶表 8.1　Sepal.Length の度数分布

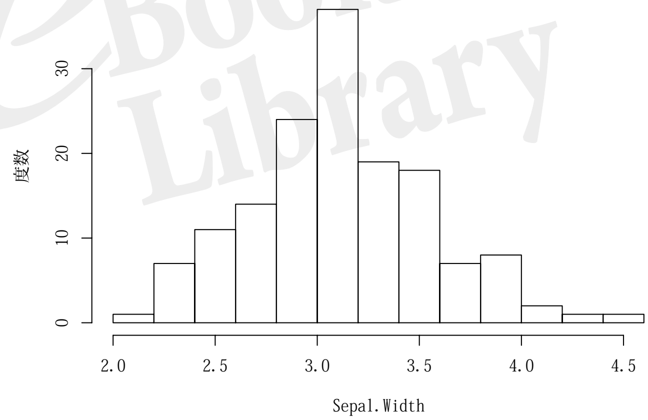
階級	度数	相対度数	累積度数	累積相対度数
4.0〜	4	2.7	4	2.7
4.5〜	18	12.0	22	14.7
5.0〜	30	20.0	52	34.7
5.5〜	31	20.7	83	55.3
6.0〜	32	21.3	115	76.7
6.5〜	22	14.7	137	91.3
7.0〜	7	4.7	144	96.0
7.5〜	6	4.0	150	100.0
合計	150	100.0		



▶図 8.1　Sepal.Length のヒストグラム

▶表 8.2　Sepal.Width の度数分布

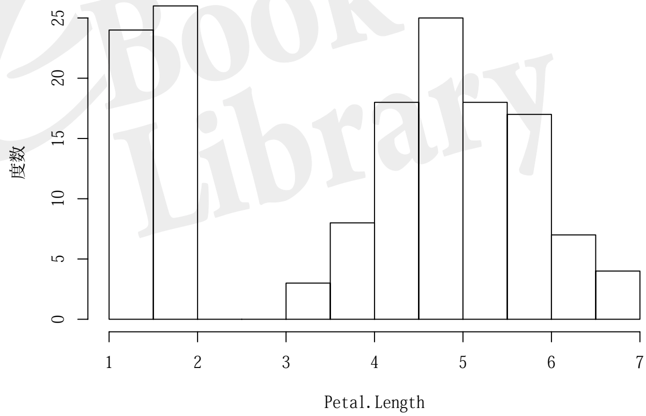
階級	度数	相対度数	累積度数	累積相対度数
2.0〜	1	0.7	1	0.7
2.2〜	7	4.7	8	5.3
2.4〜	11	7.3	19	12.7
2.6〜	14	9.3	33	22.0
2.8〜	24	16.0	57	38.0
3.0〜	37	24.7	94	62.7
3.2〜	19	12.7	113	75.3
3.4〜	18	12.0	131	87.3
3.6〜	7	4.7	138	92.0
3.8〜	8	5.3	146	97.3
4.0〜	2	1.3	148	98.7
4.2〜	1	0.7	149	99.3
4.4〜	1	0.7	150	100.0
合計	150	100.0		



▶図 8.2　Sepal.Width のヒストグラム

▶表 8.3　Petal.Length の度数分布

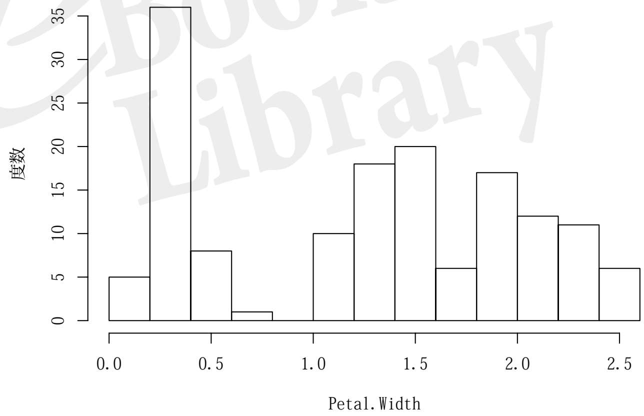
階級	度数	相対度数	累積度数	累積相対度数
1.0〜	24	16.0	24	16.0
1.5〜	26	17.3	50	33.3
2.0〜	0	0.0	50	33.3
2.5〜	0	0.0	50	33.3
3.0〜	3	2.0	53	35.3
3.5〜	8	5.3	61	40.7
4.0〜	18	12.0	79	52.7
4.5〜	25	16.7	104	69.3
5.0〜	18	12.0	122	81.3
5.5〜	17	11.3	139	92.7
6.0〜	7	4.7	146	97.3
6.5〜	4	2.7	150	100.0
合計	150	100.0		



▶図 8.3　Petal.Length のヒストグラム

▶表 8.4　Petal.Width の度数分布

階級	度数	相対度数	累積度数	累積相対度数
0.0〜	5	3.3	5	3.3
0.2〜	36	24.0	41	27.3
0.4〜	8	5.3	49	32.7
0.6〜	1	0.7	50	33.3
0.8〜	0	0.0	50	33.3
1.0〜	10	6.7	60	40.0
1.2〜	18	12.0	78	52.0
1.4〜	20	13.3	98	65.3
1.6〜	6	4.0	104	69.3
1.8〜	17	11.3	121	80.7
2.0〜	12	8.0	133	88.7
2.2〜	11	7.3	144	96.0
2.4〜	6	4.0	150	100.0
合計	150	100.0		



▶図 8.4　Petal.Width のヒストグラム

8.2 群による各変数の分布の違い

Petal.Length と Petal.Width が特に Species により違いがあることがわかったので，次はそれを明らかにするための分析を行う。

グラフとして表現する方法はいくつかあるが，ここでは箱ひげ図を用いて表すことにする。boxplot 関数を直接使うこともできるが，7.3 節の twodim.plot 関数を使う。

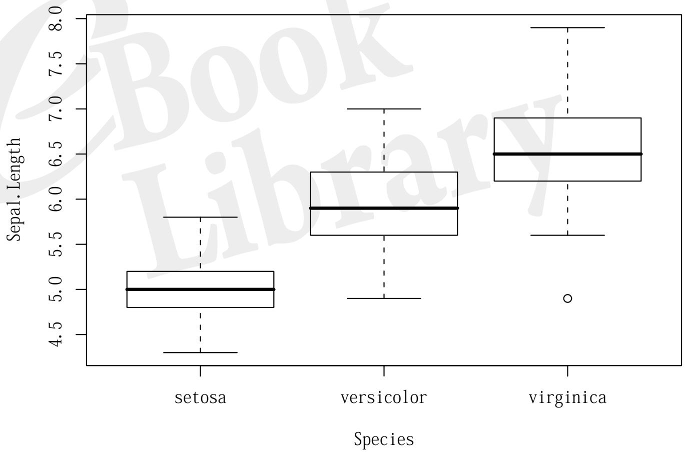
```
> twodim.plot(5, 1:4, iris, plot="iris-twodim.plot",
+             width=400, height=300)
```

結果は図 8.5〜8.8 のようになる。

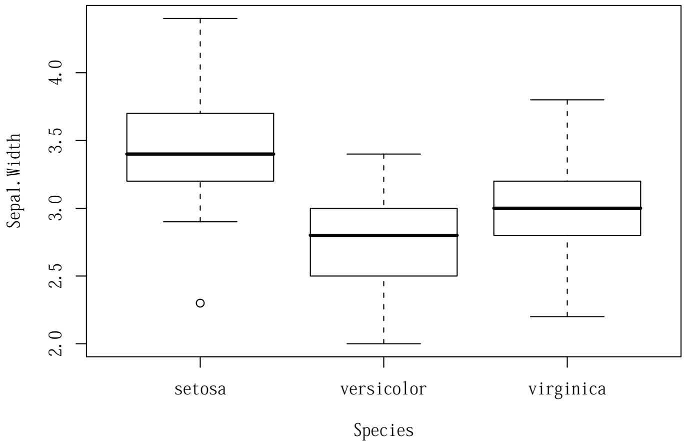
図 8.6 のように，Sepal.Width は setosa がいちばん大きく，ほかの 3 変数では setosa がいちばん小さい。

Petal.Length と Petal.Width では setosa がほかの 2 群とはきわだって小さい。2 番目に小さい versicolor とも，データ分布に重なりがない。

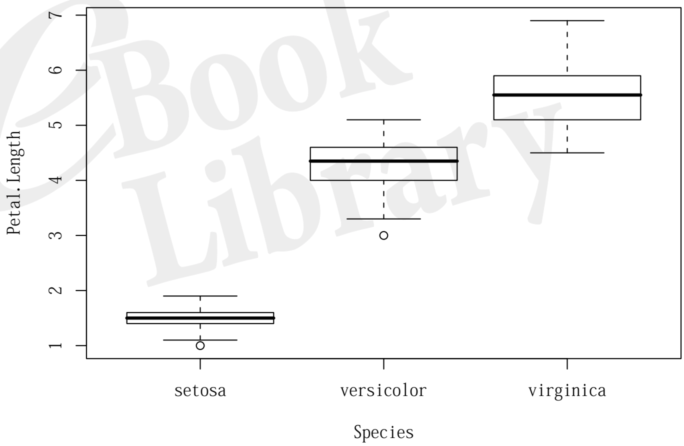
4 つの変数すべてにおいて，versicolor は virginica に比べて小さいが，データの分布は重なっている。



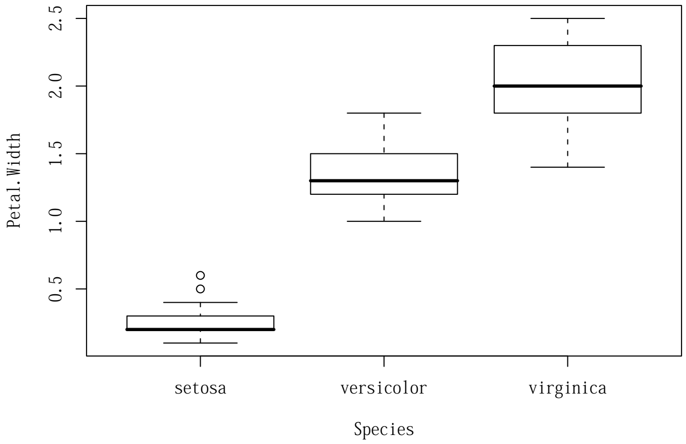
▶ 図 8.5　Sepal.Length の箱ひげ図



▶図 8.6　Sepal.Width の箱ひげ図



▶図 8.7　Petal.Length の箱ひげ図



▶図 8.8　Petal.Width の箱ひげ図

分布の違いを数値的に表すためには，by 関数によって，群別の要約統計量を求めることができる。検定も同時に行う場合には，次節（8.3 節）も参照のこと。

```
> for (i in 1:4) {
+  print(colnames(iris)[i])
+  print(by(iris[,i], iris[,5], summary))
+  print(by(iris[,i], iris[,5], sd))
+ }
[1] "Sepal.Length" # 3群別のSepal.Lengthの要約統計量
iris[, 5]: setosa
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   4.800   5.000   5.006   5.200   5.800
------------------------------------------------------
iris[, 5]: versicolor
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.900   5.600   5.900   5.936   6.300   7.000
------------------------------------------------------
iris[, 5]: virginica
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.900   6.225   6.500   6.588   6.900   7.900

iris[, 5]: setosa  # 3群別のSepal.Lengthの標準偏差
[1] 0.3524897
------------------------------------------------------
iris[, 5]: versicolor
[1] 0.5161711
------------------------------------------------------
iris[, 5]: virginica
[1] 0.6358796

[1] "Sepal.Width"  # 3群別のSepal.Widthの要約統計量
iris[, 5]: setosa
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.300   3.200   3.400   3.428   3.675   4.400
```

```
------------------------------------------------------
iris[, 5]: versicolor
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   2.525   2.800   2.770   3.000   3.400
------------------------------------------------------
iris[, 5]: virginica
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.200   2.800   3.000   2.974   3.175   3.800

iris[, 5]: setosa  # 3群別のSepal.Widthの標準偏差
[1] 0.3790644
------------------------------------------------------
iris[, 5]: versicolor
[1] 0.3137983
------------------------------------------------------
iris[, 5]: virginica
[1] 0.3224966

[1] "Petal.Length" # 3群別のPetal.Lengthの要約統計量
iris[, 5]: setosa
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.400   1.500   1.462   1.575   1.900
------------------------------------------------------
iris[, 5]: versicolor
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3.00    4.00    4.35    4.26    4.60    5.10
------------------------------------------------------
iris[, 5]: virginica
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.500   5.100   5.550   5.552   5.875   6.900

iris[, 5]: setosa  # 3群別のPetal.Lengthの標準偏差
[1] 0.173664
------------------------------------------------------
iris[, 5]: versicolor
[1] 0.469911
------------------------------------------------------
iris[, 5]: virginica
[1] 0.5518947

[1] "Petal.Width"  # 3群別のPetal.Widthの要約統計量
iris[, 5]: setosa
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.100   0.200   0.200   0.246   0.300   0.600
------------------------------------------------------
iris[, 5]: versicolor
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.200   1.300   1.326   1.500   1.800
------------------------------------------------------
iris[, 5]: virginica
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.400   1.800   2.000   2.026   2.300   2.500

iris[, 5]: setosa  # 3群別のPetal.Widthの標準偏差
[1] 0.1053856
------------------------------------------------------
iris[, 5]: versicolor
[1] 0.1977527
------------------------------------------------------
iris[, 5]: virginica
[1] 0.2746501
```

なお，複雑ではあるが，次のようにすると結果に分析対象の変数名，群の名前，結果の名前が付記されるので，見やすいかもしれない。

```
> lapply(iris[1:4], function(i) tapply(i, iris[5],
+        function(j) return(list(Summary=summary(j), SD=sd(j)))))
$Sepal.Length
$Sepal.Length$setosa
$Sepal.Length$setosa$Summary
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   4.800   5.000   5.006   5.200   5.800

$Sepal.Length$setosa$SD
[1] 0.3524897

$Sepal.Length$versicolor
$Sepal.Length$versicolor$Summary
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.900   5.600   5.900   5.936   6.300   7.000

$Sepal.Length$versicolor$SD
[1] 0.5161711
    ⋮
```

8.3 群による各変数の位置の母数の検定

各群により位置の母数（平均値や中央値）の違いがあるといえるかどうか検定を行う。本来は，理論分布を考えて，パラメトリック検定かノンパラメトリック検定のいずれかのみの検定を行うべきであるが，ここでは例示のために両方の検定について行う。

7.6 節（248 ページ）で示した breakdown 関数または 7.7 節（251 ページ）で示した indep.sample 関数を用いる。

● パラメトリック検定（一元配置分散分析）

breakdown 関数を使った一元配置分散分析の結果は表 8.5 のようになる。

setosa は，Sepal.Width の平均値はほかの 2 種より大きいが，Sepal.Length，Petal.Length，Petal.Width はほかの 2 種より小さい。特に，Petal.Length，Petal.Width では標準偏差も小さく，ほかの 2 種とは離れた位置にこぢんまりと分布しているようだ。一元配置分散分析によれば，どの変数も 3 つの群の平均値には差があるという結果であった。

```
> breakdown(1:4, 5, iris, test="parametric",
+           output="iris-breakdown.tex", encoding="euc-jp")
```

▶ 表 8.5　Species 別の基本統計量と一元配置分散分析

Species	Sepal.Length		
	データ数	平均値	標準偏差
setosa	50	5.006	0.352
versicolor	50	5.936	0.516
virginica	50	6.588	0.636
全体	150	5.843	0.828

F 値 = 138.908, 自由度 = (2, 92.211), P 値 = 0.000

Species	Sepal.Width		
	データ数	平均値	標準偏差
setosa	50	3.428	0.379
versicolor	50	2.770	0.314
virginica	50	2.974	0.322
全体	150	3.057	0.436

F 値 = 45.012, 自由度 = (2, 97.402), P 値 = 0.000

Species	Petal.Length		
	データ数	平均値	標準偏差
setosa	50	1.462	0.174
versicolor	50	4.260	0.470
virginica	50	5.552	0.552
全体	150	3.758	1.765

F 値 = 1828.092, 自由度 = (2, 78.073), P 値 = 0.000

Species	Petal.Width		
	データ数	平均値	標準偏差
setosa	50	0.246	0.105
versicolor	50	1.326	0.198
virginica	50	2.026	0.275
全体	150	1.199	0.762

F 値 = 1276.885, 自由度 = (2, 84.951), P 値 = 0.000

breakdown 関数や indep.sample 関数を使わない場合には，lapply 関数を用いて以下のようにすれば一括して検定を行うことができる。

```
> lapply(iris[1:4], function(x) oneway.test(x ~ iris[,5]))
$Sepal.Length

One-way analysis of means (not assuming equal variances)

data:  x and iris[, 5]
F = 138.9083, num df = 2.000, denom df = 92.211, p-value < 2.2e-16

$Sepal.Width

One-way analysis of means (not assuming equal variances)

data:  x and iris[, 5]
F = 45.012, num df = 2.000, denom df = 97.402, p-value = 1.433e-14

$Petal.Length

One-way analysis of means (not assuming equal variances)

data:  x and iris[, 5]
F = 1828.092, num df = 2.000, denom df = 78.073, p-value < 2.2e-16

$Petal.Width

One-way analysis of means (not assuming equal variances)

data:  x and iris[, 5]
F = 1276.885, num df = 2.000, denom df = 84.951, p-value < 2.2e-16
```

● ノンパラメトリック検定（クラスカル・ウォリス検定）

前項ではパラメトリック検定を行い，分析結果の表記にも平均値と標準偏差を使用した。

ここでは，位置母数の差のノンパラメトリック検定を行うために，breakdown 関数の引数として test="non-parametric"を使用する。統計量としては中央値，四分偏差（四分領域）のほうが好ましいので，statistics="median"を指定する。

```
> breakdown(1:4, 5, iris, test="non-parametric", statistics="median",
+           output="iris-kruskal.tex", encoding="euc-jp")
```

クラスカル・ウォリス検定の結果は表 8.6 のようになり，いずれの変数も群の中央値に有意な差が認められる。

▶ 表 8.6　Species 別の基本統計量とクラスカル・ウォリス検定

	Sepal.Length		
Species	データ数	中央値	四分偏差
setosa	50	5.000	0.400
versicolor	50	5.900	0.700
virginica	50	6.500	0.700
全体	150	5.800	1.300

χ^2_{kw} 値 = 96.937, 自由度 = 2, P 値 = 0.000

	Sepal.Width		
Species	データ数	中央値	四分偏差
setosa	50	3.400	0.500
versicolor	50	2.800	0.500
virginica	50	3.000	0.400
全体	150	3.000	0.500

χ^2_{kw} 値 = 63.571, 自由度 = 2, P 値 = 0.000

	Petal.Length		
Species	データ数	中央値	四分偏差
setosa	50	1.500	0.200
versicolor	50	4.350	0.600
virginica	50	5.550	0.800
全体	150	4.350	3.500

χ^2_{kw} 値 = 130.411, 自由度 = 2, P 値 = 0.000

	Petal.Width		
Species	データ数	中央値	四分偏差
setosa	50	0.200	0.100
versicolor	50	1.300	0.300
virginica	50	2.000	0.500
全体	150	1.300	1.500

χ^2_{kw} 値 = 131.185, 自由度 = 2, P 値 = 0.000

breakdown 関数や indep.sample 関数を使わない場合には，lapply 関数を用いて以下のようにすれば一括して検定を行うことができる。

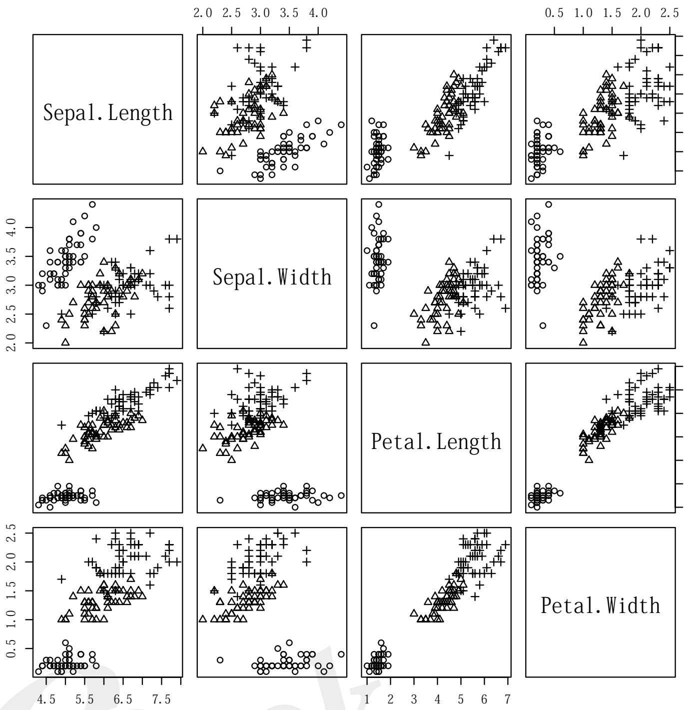
```
> lapply(iris[1:4], function(x) kruskal.test(x ~ iris[,5]))
$Sepal.Length

Kruskal-Wallis rank sum test

data:  x by iris[, 5]
Kruskal-Wallis chi-squared = 96.9374, df = 2, p-value < 2.2e-16

$Sepal.Width

Kruskal-Wallis rank sum test

data:  x by iris[, 5]
Kruskal-Wallis chi-squared = 63.5711, df = 2, p-value = 1.569e-14

$Petal.Length

Kruskal-Wallis rank sum test

data:  x by iris[, 5]
Kruskal-Wallis chi-squared = 130.411, df = 2, p-value < 2.2e-16

$Petal.Width

Kruskal-Wallis rank sum test

data:  x by iris[, 5]
Kruskal-Wallis chi-squared = 131.1854, df = 2, p-value < 2.2e-16
```

8.4　変数間の相関関係

変数間の相関関係を見るときに，まずはすべての変数の組み合わせによる散布図を描くとよい。これは，plot 関数を 1 回使うだけで描くことができる。plot 関数を使えば，群ごとに記号や色を変えて散布図を描くこともできる。

群により平均値や中央値に差が見られたことから，変数間の相関関係も群によって異なっているだろうと予想できたが，以下のようにして図 8.9 のような散布図を描けばそれが実際に確認できる。

```
> pdf("plot.pdf", width=800/72, height=600/72)
> plot(iris[,1:4], pch=as.integer(iris[,5]))
> dev.off()
```

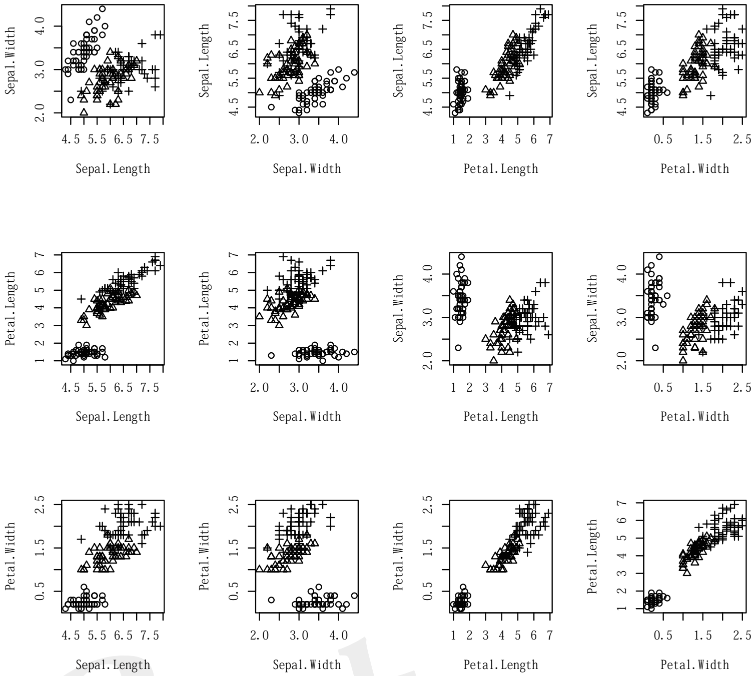


▶ 図 8.9　群ごとに記号を変えたすべての変数の組み合わせに対する散布図–1

特定の変数の組み合わせによる散布図の場合には，7.3 節（241 ページ）で示した twodim.plot 関数を用いればよい。

layout 関数を使えば，複数の散布図をまとめて描画することもできる。以下は，図 8.9 と同等の散布図（図 8.10）を描く例である。

```
> pdf("twodim-plot.pdf", width=800/72, height=600/72)
> layout(matrix(1:12, 3))        # 12枚のグラフを3×4に配置する
> par(mar=c(5, 5, 1, 1))         # グラフ間の余白を確保する
> twodim.plot(1:4, 1:4, iris, 5) # すべての組み合わせで散布図を描く
> dev.off()
```



▶図 8.10 群ごとに記号を変えたすべての変数の組み合わせに対する散布図–2

たくさんの変数の相関係数を求める場合であっても，全体の相関係数行列は cor 関数を 1 回使うだけで計算できる。

Sepal.Width はほかの 3 変数と負の相関を持っていることがわかる。だとすれば，Sepal.Width が大きい個体では，Petal.Length は小さくなるのだろうか。図 8.9 や図 8.10 をよく見ればそうではないことがわかる。

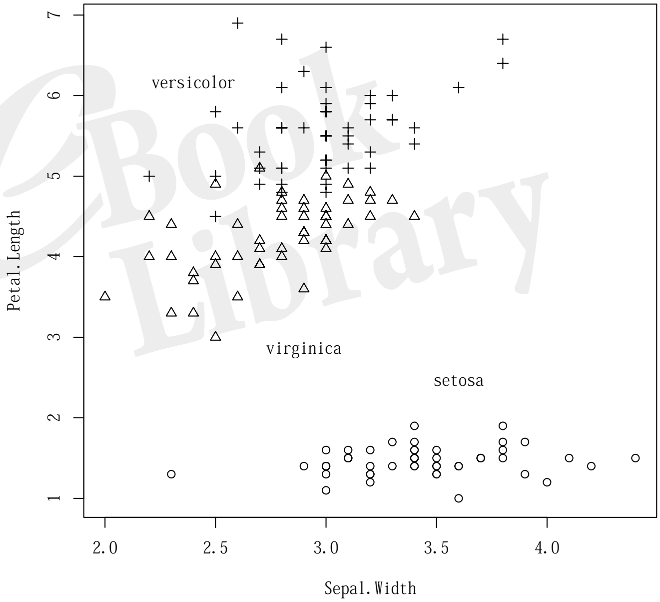
```
> cor(iris[1:4])                 # 相関係数行列を求める
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

問題に答えるために，群ごとに相関係数を求めてみよう。

```
> by(iris[,1:4], iris[,5], cor)  # 群変数の水準ごとにcorを適用
iris[, 5]: setosa                # setosa での相関係数行列
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   0.7425467    0.2671758   0.2780984
Sepal.Width     0.7425467   1.0000000    0.1777000   0.2327520
Petal.Length    0.2671758   0.1777000    1.0000000   0.3316300
Petal.Width     0.2780984   0.2327520    0.3316300   1.0000000
```

```
-----------------------------------------------------
iris[, 5]: versicolor            # versicolorでの相関係数行列
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   0.5259107    0.7540490   0.5464611
Sepal.Width     0.5259107   1.0000000    0.5605221   0.6639987
Petal.Length    0.7540490   0.5605221    1.0000000   0.7866681
Petal.Width     0.5464611   0.6639987    0.7866681   1.0000000
-----------------------------------------
iris[, 5]: virginica             # virginica での相関係数行列
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   0.4572278    0.8642247   0.2811077
Sepal.Width     0.4572278   1.0000000    0.4010446   0.5377280
Petal.Length    0.8642247   0.4010446    1.0000000   0.3221082
Petal.Width     0.2811077   0.5377280    0.3221082   1.0000000
```

群を考慮しない場合には Sepal.Width と Petal.Length の相関係数は -0.428 で負の相関を示している。しかし，3 つの群それぞれにおける Sepal.Width と Petal.Length の相関係数は 0.178, 0.561, 0.401 であり，いずれも正の相関を示している。このようなことがなぜ起きるかは，図 8.11 の散布図を見ればわかる。同じようなことが, Sepal.Width と Petal.Width, Sepal.Length と Sepal.Width の間にも起こっている。



►図 8.11　Sepal.Width と Petal.Length の Species 別の散布図

8.5 グループの判別

さて，iris データセットの解析例の最終目的として，3 つのグループを判別するにはどのようにすればよいかを考えてみよう。

iris データセットには各データがどの種のものであるかという情報があるので，実際には 3 群の判別分析をすればよい（191 ページに示したように lda で正準判別分析をする）。しかし，ここでは個々のデータがどの種に属するかわからないと仮定して分析してみよう。

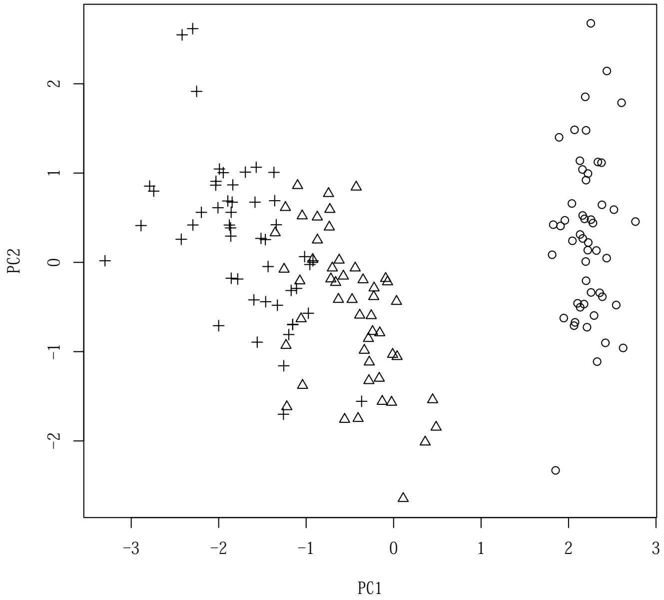
まず，図 8.9 や図 8.10 のような全変数分の二変数ごとの散布図を眺める代わりに，少数個の合成変数で全体のデータの分布状況を把握するために主成分分析を行い，主成分得点を散布図に描くことを考えてみよう。

iris データセットの最初の 4 変数に対して主成分分析を適用する（199 ページの prcomp3）。

```
> ans <- prcomp3(iris[1:4])
   ⋮
> ans$eigenvalues
[1] 2.91849782 0.91403047 0.14675688 0.02071484
```

第 1 主成分と第 2 主成分に対応する固有値は 2.918 と 0.914 であり，それらを合わせると全情報の 96% を説明することがわかる。

そこで，第 1 主成分得点と第 2 主成分得点のみを用いて Species ごとに記号を変えた散布図を描画すれば，全変数を使った散布図とほとんど同じ情報を持つといえる。その散布図が図 8.12 である。図 6.18（193 ページ）や図 8.11 とよく似ていることがわかる。



►図 8.12　第 1，第 2 主成分得点の Species 別の散布図

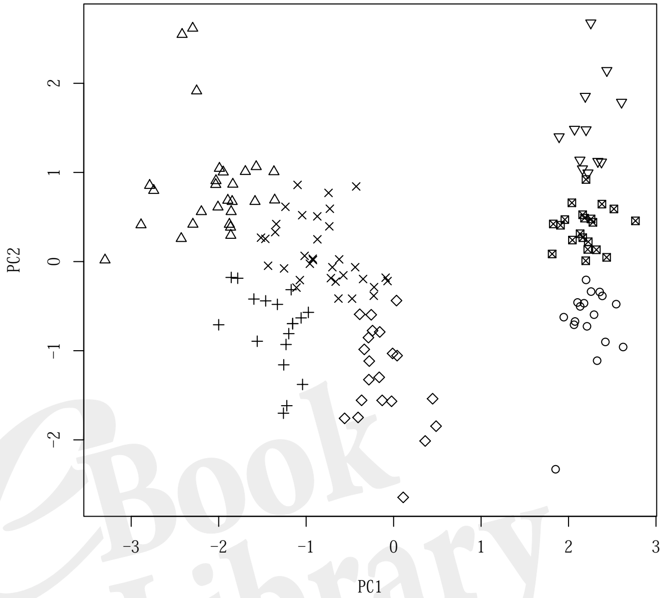
次に，第 1 主成分得点と第 2 主成分得点だけを用いて，k-means 法によるクラスター分析をしてみよう。

クラスターが 3 個であるとして分析してみると，以下のように versicolor と verginica の判別がよくない。やはり餅は餅屋というように，正準判別分析を行うほうがよい。

```
> ans2 <- kmeans(ans$x[,1:2], 3, nstart=100)
> xtabs(~iris[,5]+ans2$cluster)
            ans2$cluster
iris[, 5]     1  2  3
  setosa     50  0  0
  versicolor  0 39 11
  virginica   0 14 36
```

試みにクラスター数を 4，5 と順に変えて分析を続けてみると，クラスター数を 7 と仮定した分析では判別結果がよくなる。少しの違いに基づいてより細かく分けようというのであるから，当然の結果ではある。以下のようにして分類結果図（図 8.13）を描くことにより，例えば setosa は縦軸方向にかなり広い範囲に散らばっているが，これが 3 つのサブグループに分かれるということになれば，そのサブグループを特徴付けるデータの有無を検討するきっかけになるであろう。

```
> ans7 <- kmeans(ans$x[,1:2], 7, nstart=100)
> print(sum(ans7$withinss))
[1] 47.3823
> xtabs(~iris[,5]+ans7$cluster)
            ans7$cluster
iris[, 5]     1  2  3  4  5  6  7
  setosa      0  0 12  0 17 21  0
  versicolor 26 20  0  4  0  0  0
  virginica   8  1  0 14  0  0 27
```



►図 8.13　第 1，第 2 主成分得点を用いた k-means 法によるクラスター分析の結果

Rの概要

付録として R の基本的な使用法について簡単にまとめる。関連図書もたくさん刊行されているので，より詳しくはそれらを参考にしていただきたい。

本書の実行例で，行頭にある「>」はプロンプトを示す。それに続く太字がキーボードからコンソールへの入力である。入力が複数行にわたる場合，2 行目以降には行頭に「+」が表示される。

「#」から行末までは注釈であり，何が書かれていても R は無視する。本書では，この注釈を使い，実行例に対する簡単な補足を行っている。

各行の入力の最後には［return］キーを押す。入力に対する結果があれば，次の行以降に表示される。結果の行頭に出力される「[1]」などは，結果の一部ではなく，その次に表示されるものが結果の何番目の要素であるかを示している。

A.1 データの種類

R で使えるデータの主な種類は，スカラー，ベクトル，行列，データフレーム，リストである[*1]。

データの内容としては，数値データ，factor データ（カテゴリーデータ），文字データなどがある。

A.1.1 スカラー

スカラーは 1 個のデータである。

```
> 1                          # スカラー（数値データ）
[1] 1
> "Japan"                    # スカラー（文字データ）
[1] "Japan"
```

スカラーを含め，本節で示すようなデータは，変数（オブジェクト）に付値（代入）して後から呼び出して使うことができる。変数名は，英字で始まり，英数字およびピリオドで構成される。以下の例では，x，foo.bar3 などが変数名である。変数名の直後の「<-」は付値を意味し，それに続くものを直前の変数に代入することを表す。

*1 R では，スカラーは長さ 1 のベクトル，行列は次元属性を持つベクトルとして扱われる。

変数への付値によって表示される結果は何もない。付値した後に，付値された変数を入力すれば，結果が表示される。付値すると同時に付値された結果を表示したい場合は，式の前後を「( )」（丸括弧）でくくるとよい。

```
> (foo.bar3 <- 5.234)          # 括弧でくくると，付値された結果が表示される
[1] 5.234
> x <- 6
> x                            # 付値された数値を引用できる
[1] 6
> x+foo.bar3                   # 付値された数値を使って計算する
[1] 11.234
```

A.1.2　ベクトル

ベクトルは複数個の同種のデータの集まりである。ベクトルは c 関数を用いて作る。

```
> c(1, 4, 8, 21)                # 4つの要素を持つベクトル
[1]  1  4  8 21
```

連続する整数の範囲の始まりと終わりを「:」でつなげることで，連続する整数からなるベクトルが指定できる。

```
> 11:15                         # 5つの要素を持つベクトル
[1] 11 12 13 14 15
```

必ずしも連続するわけではない等差数列のようなベクトルは，seq 関数を用いて作ることができる。最初の 2 つの引数は，数列の最初と最後の数値である。by 引数によって公差を指定するか，length.out 引数によって数列の長さを指定するという 2 通りの使用法がある。

```
> seq(3.5, 5.9, by=0.1)         # 3.5～5.9まで0.1刻みのベクトル
 [1] 3.5 3.6 3.7 3.8 3.9 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.0
[17] 5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9
> seq(3.5, 5.9, length.out=5) # 3.5～5.9の範囲で要素数が5のベクトル
[1] 3.5 4.1 4.7 5.3 5.9
```

ベクトルの要素を参照するときは，以下のように，「[ ]」内に何番目の要素であるかを表す添え字（数値）を指定する。

```
> x <- c(2, 4, 6, 1, 8)         # 5つの要素を持つベクトル
> x[3]                          # xの3番目の要素
[1] 6
```

添え字は，定数だけでなく，変数や式を使っても指定できる。ベクトルの要素の指定法には，このほかにも A.2 節に示すような様々な参照方法がある。

A.1.3 行列

行列は同じ種類のデータを行方向と列方向に並べたものである。行列は matrix 関数で作ることができる。

R の行列は，C や Java の行列と異なり，列優先で値が割り当てられる。1 から 6 の要素を 2 行 3 列の行列にすると，以下のように割り振られる。

```
> matrix(1:6, nrow=2, ncol=3) # デフォルトの2×3行列
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

C や Java のように行優先で割り振るには，以下のように byrow=TRUE を指定すればよい。

```
> matrix(1:6, nrow=2, ncol=3, byrow=TRUE) # 行優先の2×3行列
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

行列は，行ベクトルまたは列ベクトルをそれぞれ rbind 関数，cbind 関数で束ねることによっても作成できる。

```
> r1 <- c(1, 2, 3)             # r1 <- 1:3 と同じこと
> r2 <- c(4, 5, 6)             # r2 <- 4:6 と同じこと
> rbind(r1, r2)                # 2つのベクトルを行ベクトルに束ねる
   [,1] [,2] [,3]
r1    1    2    3
r2    4    5    6
> c1 <- c(1, 4)
> c2 <- c(2, 5)
> c3 <- c(3, 6)
> cbind(c1, c2, c3)            # 3つのベクトルを列ベクトルに束ねる
     c1 c2 c3
[1,]  1  2  3
[2,]  4  5  6
```

行列の要素は，以下のように参照する。

```
> m <- matrix(1:6, nrow=2, ncol=3)
> m                            # オブジェクト名では全体が参照される
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m[2, 3]                      # [行番号, 列番号] の形式で参照
[1] 6                          # 結果はスカラー
> m[2,]                        # [行番号 , ] の形式で行全体を参照
[1] 2 4 6                      # 結果はベクトル
> m[,3]                        # [ , 列番号] の形式で列全体を参照
[1] 5 6                        # 結果はベクトル
```

行列の要素の指定法には，このほかにも A.2 節に示すような様々な参照方法がある。

A.1.4　データフレーム

R で最もよく使うデータはデータフレームであろう。データフレームの列は変数を表し，名前が付けられる。

```
> data.frame(x=c(1, 4, 8, 21), y=c(3, 2, 1, 5))
   x y
1  1 3
2  4 2
3  8 1
4 21 5
```

行列は，行列の全部の要素が同じデータ型であるが（例えば数値データ行列），データフレームは列が違えば別のデータ型をとることができる。列のなかでは同じデータ型の値しかとれない。これは，統計データにおいては当たり前の性質だといえる。以下の例では，変数 x は数値データであるが，z に付値されている性別は factor データ（カテゴリーデータ）になっている。

```
> data.frame(x=c(1, 4, 8, 21),
+            z=c("male", "female", "male", "U.K."))
   x      z
1  1   male
2  4 female
3  8   male
4 21   U.K.
```

データフレームは上の例のようにベクトルから作ることもあるが，データファイルから read.table 関数で読み込むことによって作るのが普通である。

```
> df <- read.table("idol.dat", header=TRUE) # ファイルからデータを読む
> head(df)                                  # 先頭部分を表示するhead関数
  Height Weight BloodType
1    159     45         B
2    160     45         A
3    167     NA         B
4    160     45         O
5    155     45         O
6    162     43         O
```

場合によっては，data.frame 関数によって，行列からデータフレームに変換することもある。

```
> m <- matrix(1:6, nrow=2, ncol=3)
> m                             # 2×3行列
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> df <- data.frame(m)           # データフレームに変換する
> df                            # 3変数からなるデータフレーム
  X1 X2 X3
1  1  3  5
2  2  4  6
```

逆に，データフレームを行列に変換する data.matrix 関数もある。

データフレームが複数のデータ型を含む場合には注意が必要である。以下の例において，データフレーム df2 の 1 列目は文字列のように見えるが，factor を表す。これを行列に変換すると，m2 の 1 列目は数値データになる。

```
> df2 <- data.frame(a=c("foo", "bar", "baz"),
+                   b=c(3.2, 5.3, 6.5))
> df2                          # 2変数からなるデータフレーム
    a   b                      # 1列目はfactor
1 foo 3.2
2 bar 5.3
3 baz 6.5
> m2 <- data.matrix(df2)       # 行列に変換する
> m2                           # 1列目は数値データになる
  a   b
1 3 3.2
2 1 5.3
3 2 6.5
```

データフレームの要素は，行列の場合の要素の参照法に加えて，A.2 節に示すようなデータフレーム特有の方法でも参照できる。

A.1.5 リスト

リストは，いくつかのデータのまとまりを名前を付けて記憶しているデータ型である。R の関数が結果として返すオブジェクトの表現形として使われることが多い。リストは list 関数によって作られる。リストの各要素は変数名$要素名として参照する。

```
> l <- list(a=c(1, 4, 8, 21), b=c("red", "blue", "green"), c=1.3)
> l                            # リスト全体の表示
$a                             # aという名前の要素
[1]  1  4  8 21                # 内容は数値ベクトル

$b                             # bという名前の要素
[1] "red"    "blue"   "green"  # 内容は文字列ベクトル

$c                             # cという名前の要素
[1] 1.3                        # 内容はスカラー

> l$a                          # リストの要素の参照
[1]  1  4  8 21
> l$b[2]                       # リストの要素のベクトルの要素
[1] "blue"
```

リストがどのような名前の要素を持つかは names 関数で調べる。さらにどのような要素であるかの情報（内部構造）は str 関数で調べることができる。

```
> names(l)                     # どのような名前の要素を持つか
[1] "a" "b" "c"                # a, b, cという名前の3つの要素を持つ
```

```
> str(l)                               # リストの内部構造を調べる
List of 3                              # 3つの要素を持つ
 $ a: num [1:4] 1 4 8 21               # 数値ベクトル
 $ b: chr [1:3] "red" "blue" "green"  # 文字列ベクトル
 $ c: num 1.3                          # 数値（スカラー）
```

A.2 ベクトルや行列やデータフレームの要素の指定法

ベクトルも行列も，その要素を指定するには（A.1）のように添え字を使う。添え字は整数定数だけでなく，式を計算した結果が整数になるようなものでもよい（計算結果が整数でない実数になる場合は，整数部分が添え字として使われる）。

ベクトル：	変数名 [順番を表す添え字式]	
行列：	変数名 [行の添え字式，列の添え字式]	(A.1)
データフレーム：	変数名 [行の添え字式，列の添え字式]	

R の特徴として，負の値を持つ添え字は「該当する要素を除いた残りの要素全部」を意味する。また，要素数が同じ論理ベクトルが添え字として使われた場合には，論理ベクトルの要素が真（TRUE）に対応する要素が選択される。

A.2.1 ベクトルの要素の指定例

以下にベクトルの要素を指定する様々な方法の例を示す。

```
> x <- c(3, 2, 5, 4, 1)
> x                           # ベクトル全体
[1] 3 2 5 4 1
> x[2]                        # 2番目の要素
[1] 2
> x[3:5]                      # 3〜5番目の要素
[1] 5 4 1
> x[-c(1, 3, 5)]              # 1, 3, 5番目を除いた残りの要素
[1] 2 4
> x[c(2, 4)]                  # 2, 4番目の要素
[1] 2 4
> x[-1]                       # 1番目を除いた残りの要素
[1] 2 5 4 1
> x[c(TRUE, FALSE, FALSE, TRUE, TRUE)]
[1] 3 4 1
> x[x > 3]                    # 3より大きい要素
[1] 5 4
> x > 3                       # 上の例を説明するために
[1] FALSE FALSE  TRUE  TRUE FALSE
```

A.2.2 行列，データフレームの要素の指定例

行列とデータフレームの要素は，2 個の添え字（行の添え字と列の添え字）を使って指定する。それぞれの添え字の指定方法は，前述のベクトルの要素の指定方法と同じである。

```
> z <- matrix(1:24, 4, 6)
> z                              # 行列全体
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
> z[2, 4]                        # 2行4列目の要素
[1] 14
> z[3,]                          # 3行目（ベクトルになる）
[1]  3  7 11 15 19 23
> z[3, , drop=FALSE]             # 1行6列の行列
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    3    7   11   15   19   23
> z[1:3,]                        # 1～3行
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
> z[-4,]                         # 4行を除く残り
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
> z[c(1, 3),]                    # 1行と3行
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    3    7   11   15   19   23
> z[,4]                          # 4列（ベクトルになる）
[1] 13 14 15 16
> z[, 4, drop=FALSE]             # 4行1列の行列
     [,1]
[1,]   13
[2,]   14
[3,]   15
[4,]   16
> z[,3:5]                        # 3～5列
     [,1] [,2] [,3]
[1,]    9   13   17
[2,]   10   14   18
[3,]   11   15   19
[4,]   12   16   20
> z[,-c(1, 2, 6)]                # 1, 2, 6列を除く残り
     [,1] [,2] [,3]
[1,]    9   13   17
[2,]   10   14   18
[3,]   11   15   19
[4,]   12   16   20
> z[1:2, 2:5]                    # 1, 2行と2～5列
     [,1] [,2] [,3] [,4]
[1,]    5    9   13   17
[2,]    6   10   14   18
> z[2, -5]                       # 2行で5列を除いたもの
[1]  2  6 10 14 22
```

```
> z[2, -5, drop=FALSE]         # 2行で5列を除いたもの
     [,1] [,2] [,3] [,4] [,5]  # drop=FALSEに意味がある
[1,]    2    6   10   14   22
> z[-c(3, 4), -5]              # 3, 4行と5列を除いた残り
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   21
[2,]    2    6   10   14   22
```

A.2.3　データフレームならではの要素の指定例

データフレームには，列が変数を表し，列に変数の名前が付いているという特徴がある。例えば，R に用意されている iris という 5 列のデータフレームには，それぞれの列に"Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "Species"という名前が付いている。データフレームの要素はこの名前を使っても指定できる。

```
> iris[,"Species"]             # カンマがあるとベクトル
  [1] setosa     setosa     setosa     setosa     setosa     setosa
  [7] setosa     setosa     setosa     setosa     setosa     setosa
   ⋮

> iris["Sepal.Width"]          # カンマがないとデータフレーム
    Sepal.Width
1           3.5
2           3.0
3           3.2
4           3.1
5           3.6
   ⋮
```

実はデータフレームは各列を要素とするリストなので，オブジェクト名$要素名として列を取り出せる。

```
> iris$Sepal.Width             # Sepal.Widthという名前の列（変数）
  [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4
 [17] 3.9 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4
   ⋮
> iris[,2]                     # 2列目がSepal.Width（上と同じもの）
  [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4
 [17] 3.9 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4
   ⋮
> iris[2]                      # iris[,2]と似ているが, これはデータフレーム
    Sepal.Width
1           3.5
2           3.0
3           3.2
4           3.1
5           3.6
   ⋮
> class(iris[,2])              # 確かにこれはベクトル
[1] "numeric"
> class(iris[2])               # これはデータフレーム
[1] "data.frame"
```

A.3 演算

数値間では四則演算をはじめとする通常の演算ができる。ベクトルや行列やデータフレームやリストも，その要素が数値のものは演算の対象になる。

スカラーとベクトル・行列・データフレーム・リストとの間の演算，ベクトルと行列・データフレーム・リストとの間の演算，行列と行列との間の演算，データフレームとデータフレームとの間の演算もある。

A.3.1 四則演算など

```
> 4+5                          # 足し算
[1] 9
> 6-2                          # 引き算
[1] 4
> 5*3                          # 掛け算
[1] 15
> 5/3                          # 割り算
[1] 1.666667
> 5%/%3                        # 割り算の商の整数部分
[1] 1
> 5%%3                         # 割り算の余り
[1] 2
> 2^3                          # べき乗
[1] 8
> 27^(1/3)                     # べき乗（3乗根）
[1] 3
```

A.3.2 関数

統計学でよく使われる数学関数は以下のような sqrt，exp，log であろう。

```
> sqrt(23.456)                 # 平方根
[1] 4.843139
> exp(3.4567)                  # 指数関数
[1] 31.71215
> log(exp(3.4567))             # 対数はデフォルトでは自然対数
[1] 3.4567
> log(100, base=10)            # 底を指定できる
[1] 2
> log10(100)                   # 常用対数関数もある
[1] 2
> log(16, base=2)
[1] 4
> log2(16)                     # 底が2の対数関数
[1] 4
```

R は統計学のためのプログラムであるから，多くの統計学関数も含まれている。統計学関数とは何かという定義にもよるが，R には標準正規分布（norm），χ^2 分布（chisq），t 分布（t），F 分布（f）などがある。さらに，それらの密度，確率，分位点，乱数に関する関数群もあり，それぞれ関数名の先頭に d，p，q，r を付けた

ものが関数名になっている。

分布関数としては，そのほかにも，一様分布（*unif），指数分布（*exp），対数正規分布（*lnorm），二項分布（*binom），ポアソン分布（*pois）など，全部で 22 種類の分布関数が用意されている。特に，それぞれの分布に従う乱数が容易に発生できるので，シミュレーションも簡単に行うことができる。

```
> pnorm(1.96, lower.tail=FALSE)       # 標準正規分布の上側確率
[1] 0.02499790
> pchisq(3.8416, 1, lower.tail=FALSE) # χ2分布の上側確率
[1] 0.04999579
> pt(1.96, 2500, lower.tail=FALSE)    # t分布の上側確率
[1] 0.02505336
> pf(1, 5, 20, lower.tail=FALSE)      # F分布の上側確率
[1] 0.4430252
> dnorm(0)                            # 標準正規分布でZ=0のときの高さf(Z)
[1] 0.3989423
> qnorm(0.05)                         # 標準正規分布で下側確率が5%のZ値
[1] -1.644854
> rnorm(100)                          # 標準正規乱数を100個生成する
  [1]  0.11764660 -0.94747461 -0.49055744 -0.25609219  1.84386201
  [6] -0.65194990  0.23538657  0.07796085 -0.96185663 -0.07130809
    ⋮
```

A.3.3　2 つのデータの間の演算

異なるデータ型を含まない限り，データフレームは行列と同じように扱えるので，以下では，ベクトルと行列についてのみ例示する。

● スカラーとベクトル，行列，データフレーム間の演算

スカラーとベクトル，行列，データフレーム間の演算は，ベクトル，行列，データフレームの各要素とスカラーの間で演算が行われる。また，スカラーデータを対象にする sqrt や log などの関数も各要素に対して適用される。

```
> (x <- c(2, 5, 9))
[1] 2 5 9
> x+4                        # xの各要素に4を加える
[1]  6  9 13
> x-2                        # xの各要素から4を引く
[1] 0 3 7
> x/3                        # xの各要素を3で割る
[1] 0.6666667 1.6666667 3.0000000
> m <- matrix(c(2, 3, 5, 7, 4, 1), 2, 3)
> m                          # 2×3行列
     [,1] [,2] [,3]
[1,]    2    5    4
[2,]    3    7    1
> 3*m                        # mの各要素に3を掛ける
     [,1] [,2] [,3]
[1,]    6   15   12
[2,]    9   21    3
```

```
> m^2                          # mの各要素を二乗する
     [,1] [,2] [,3]
[1,]    4   25   16
[2,]    9   49    1
> sqrt(m)                      # mの各要素の平方根をとる
         [,1]     [,2] [,3]
[1,] 1.414214 2.236068    2
[2,] 1.732051 2.645751    1
```

● **ベクトルとベクトル，行列，データフレーム間の演算**

ベクトルとベクトルの演算もそれぞれ対応する要素間で演算が行われる。

```
> (x <- 1:4)
[1] 1 2 3 4
> (y <- c(2, 4, 1, 5))
[1] 2 4 1 5
> x+y                          # 各要素間の和
[1] 3 6 4 9
> x-y                          # 各要素間の差
[1] -1 -2  2 -1
> x*y                          # 各要素間の積
[1]  2  8  3 20
> x/y                          # 各要素間の商
[1] 0.5 0.5 3.0 0.8
> x^y                          # xの要素をyの要素のべき乗
[1]    1   16    3 1024
```

R で特徴的なのは，要素数の異なるベクトルどうしの演算である。このときは，短いほうのベクトルは，先頭に戻って再利用される（リサイクルされる）。以下の例では，x は要素数 6，y は要素数 3 なので，x[4] と足し算されるのは y の先頭に戻って y[1] になる。さらに x[5] は y[2]，x[6] は y[3] と足されることになる。このように，長いほうのベクトルの要素数が短いほうのベクトルの要素数の倍数であれば問題なく処理が行われる。しかし，倍数になっていない場合には問題が生じる。以下の例では z の要素数は 4 なので，x の長さに対してリサイクルしようとしても余ってしまう。通常このようなことは何らかの間違いである可能性が高いので，R は警告メッセージを出す。

```
> (x <- 1:6)
[1] 1 2 3 4 5 6
> (y <- c(2, 3, 5))
[1] 2 3 5
> x+y
[1]  3  5  8  6  8 11
> z <- c(1, 3, 9, 10)
> x+z
[1]  2  5 12 14  6  9
Warning message:
In x + z :
   長いオブジェクトの長さが短いオブジェクトの長さの倍数になっていません
```

リサイクルにより，以下のように行ごとに一定の数を加減乗除するような場合に，ベクトルと行列の演算で便利な記述ができる。前述のように R の行列は列優先なので，x との演算の対象となるのは m[1, 1]，m[2, 1]，m[3, 1]，m[1, 2]，m[2, 2] の順である。m[1, 2] と演算を行うときに x は使い果たされているので x の先頭に戻って x[1] が使われ，m[2, 2] は x[2] と演算される。このようなことが繰り返され，結果として m の 1 行目にはすべて x[1] が加えられ，2 行目には x[2]，3 行目には x[3] が加えられる。

```
> (m <- matrix(1:12, 3, 4))
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> x <- c(2, 4, 7)
> x+m
     [,1] [,2] [,3] [,4]
[1,]    3    6    9   12
[2,]    6    9   12   15
[3,]   10   13   16   19
```

先ほどの行列 m において，第 1 列に 3，第 2 列に 4，第 3 列に 6，第 4 列に 9 を掛けたい場合（つまり列ごとに一定の数を加減乗除したい場合）にはどのようにしたらよいだろうか。

まず，行列を転置する t 関数（A.4.1 項）を使い，ベクトルと転置行列の間で演算を行ってから，結果をもう一度転置すればよい。

```
> (m <- matrix(1:12, 3, 4))
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> t(m)                          # 転置する
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> y <- c(3, 4, 6, 9)
> y*t(m)                        # 中間結果
     [,1] [,2] [,3]
[1,]    3    6    9
[2,]   16   20   24
[3,]   42   48   54
[4,]   90   99  108
> t(y*t(m))                     # 結果をもう一度転置する
     [,1] [,2] [,3] [,4]
[1,]    3   16   42   90
[2,]    6   20   48   99
[3,]    9   24   54  108
```

● **行列と行列の間の演算**

サイズが同じ 2 つの行列の間での演算も，要素ごとに行われる。行列どうしの演算においては，行列のサイズが異なるときにはリサイクルされずエラーになる。

```
> (m <- matrix(1:12, nrow=3, ncol=4))
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> (n <- matrix(c(2, 1, 3, 2, 3, 4, 5, 1, 2, 3, 4, 5), nrow=3, ncol=4))
     [,1] [,2] [,3] [,4]
[1,]    2    2    5    3
[2,]    1    3    1    4
[3,]    3    4    2    5
> m*n                                # ともに3行4列の行列の要素ごとの演算
     [,1] [,2] [,3] [,4]
[1,]    2    8   35   30
[2,]    2   15    8   44
[3,]    9   24   18   60
```

A.4 行列ならではの操作

A.3 節のような行列の要素ごとの演算のほかに，行列特有の演算がある。

A.4.1 転置行列

行と列を入れ替えたものは，元の行列の転置行列と呼ぶ。本書では行列 $\boldsymbol{A}$ の転置行列を右肩にプライムを付けて $\boldsymbol{A}'$ と表す。

```
> (m <- matrix(1:12, nrow=3, ncol=4))
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> t(m)                               # 転置する
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

A.4.2 対角行列と単位行列

対角要素が 0 以外の値で，対角以外の要素が 0 である正方行列を，対角行列と呼ぶ。

対角要素が 1 の対角行列を単位行列 $\boldsymbol{I}$ と呼ぶ。

R ではいずれも `diag` 関数で作ることができる。

```
> diag(c(2, 4, 8))                 # 対角要素を指定すると対角行列ができる
     [,1] [,2] [,3]
[1,]    2    0    0
[2,]    0    4    0
[3,]    0    0    8
> diag(3)                          # サイズだけを指定すると単位行列ができる
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

A.4.3　三角行列

対角要素よりも下の要素が 0 である正方行列を上三角行列，対角要素よりも上の要素が 0 である正方行列を下三角行列と呼ぶ。

R では，それぞれ upper.tri，lower.tri 関数を使って作る。また，対角要素を含めるか含めないかを指定することができる。

```
> (x <- matrix(1:9, 3, 3))
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x[upper.tri(x)] <- 0             # 上三角行列を0にする
> x
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    2    5    0
[3,]    3    6    9
> (y <- matrix(c(1, 2, 4, 3, 6, 5, 4, 6, 8), 3, 3))
     [,1] [,2] [,3]
[1,]    1    3    4
[2,]    2    6    6
[3,]    4    5    8
> y[lower.tri(y, diag=TRUE)] <- 0 # 対角を含み下三角行列を0にする
> y
     [,1] [,2] [,3]
[1,]    0    3    4
[2,]    0    0    6
[3,]    0    0    0
```

A.4.4　行列式

正方行列の行列式を求める関数は det である。

正方行列の行列式が 0 であるときには「行列は特異である」という。行列式が 0 でないときには「行列は正則である」という。

```
> (w <- matrix(c(3, 2, 3, 4, 5, 4, 3, 1, 6), 3, 3))
     [,1] [,2] [,3]
[1,]    3    4    3
[2,]    2    5    1
[3,]    3    4    6
> det(w)
[1] 21
```

A.4.5 行列積

$a \times b$ 行列と $b \times c$ 行列をこの順で掛けるときのみ，行列の積が定義できる。行列積は（A.2）のように定義され，結果は $a \times c$ 行列になる。R での演算記号は %*% である。

$$\boldsymbol{X}\,\boldsymbol{Y} = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \end{pmatrix} \times \begin{pmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \\ y_{31} & y_{32} \end{pmatrix} \tag{A.2}$$

$$= \begin{pmatrix} x_{11}\,y_{11} + x_{12}\,y_{21} + x_{13}\,y_{31} & x_{11}\,y_{12} + x_{12}\,y_{22} + x_{13}\,y_{32} \\ x_{21}\,y_{11} + x_{22}\,y_{21} + x_{23}\,y_{31} & x_{21}\,y_{12} + x_{22}\,y_{22} + x_{23}\,y_{32} \\ x_{31}\,y_{11} + x_{32}\,y_{21} + x_{33}\,y_{31} & x_{31}\,y_{12} + x_{32}\,y_{22} + x_{33}\,y_{32} \\ x_{41}\,y_{11} + x_{42}\,y_{21} + x_{43}\,y_{31} & x_{41}\,y_{12} + x_{42}\,y_{22} + x_{43}\,y_{32} \end{pmatrix}$$

```
> (A <- matrix(c(1, 3, 2, 3, 2, 1), 3, 2))
     [,1] [,2]
[1,]    1    3
[2,]    3    2
[3,]    2    1
> (B <- matrix(c(3, 2, 1, 4, 3, 2, 5, 7), 2, 4))
     [,1] [,2] [,3] [,4]
[1,]    3    1    3    5
[2,]    2    4    2    7
> A%*%B                             # 行列の積
     [,1] [,2] [,3] [,4]
[1,]    9   13    9   26
[2,]   13   11   13   29
[3,]    8    6    8   17
```

A.4.6 逆行列

行列 $\boldsymbol{A}$ に行列 $\boldsymbol{B}$ を掛けた結果が単位行列になるとき，$\boldsymbol{B}$ を行列 $\boldsymbol{A}$ の逆行列 $\boldsymbol{B} = \boldsymbol{A}^{-1}$ という。特異行列に逆行列はない。

R で逆行列を求める関数は solve である。

```
> (w <- matrix(c(3, 2, 3, 4, 5, 4, 3, 1, 6), 3, 3))
     [,1] [,2] [,3]
[1,]    3    4    3
[2,]    2    5    1
[3,]    3    4    6

> (v <- solve(w))                   # 逆行列を求める
           [,1]       [,2]       [,3]
[1,]  1.2380952 -0.5714286 -0.5238095
[2,] -0.4285714  0.4285714  0.1428571
[3,] -0.3333333  0.0000000  0.3333333

> w%*%v                             # 元の行列と逆行列の行列積は単位行列になる
             [,1] [,2]         [,3]
[1,] 1.000000e+00    0 2.220446e-16
[2,] 1.665335e-16    1 5.551115e-17
[3,] 0.000000e+00    0 1.000000e+00
```

A.4.7　固有値と固有ベクトル

行列 $\boldsymbol{A}$ を実対称行列，列ベクトルを $\boldsymbol{x}$，λ を定数値とする。

$$\boldsymbol{A}\,\boldsymbol{x} = \lambda\,\boldsymbol{I}\,\boldsymbol{x}$$

の関係が成り立つとき，λ を固有値，$\boldsymbol{x}$ をその固有値に対応する固有ベクトルと呼ぶ。

R では固有値と固有ベクトルを eigen 関数で求めることができる。eigen 関数は，固有値（要素名 values）と固有ベクトル（要素名 vectors）の 2 つの要素からなるリストとして返す。固有ベクトルは，列ベクトルが個々の固有ベクトルであるような行列として返される。

```
> (w <- matrix(c(13, 4, 3, 4, 5, 1, 3, 1, 6), 3, 3))
     [,1] [,2] [,3]
[1,]   13    4    3
[2,]    4    5    1
[3,]    3    1    6
> a <- eigen(w)                # 固有値，固有ベクトルを求める
> str(a)                       # 結果の内部構造を見てみる
List of 2                      # 結果はリストで返される
 $ values : num [1:3] 15.68  5.00  3.32
 $ vectors: num [1:3, 1:3]  0.881  0.359  0.310 -0.236 ...
> a                            # 結果を表示する
$values                        # 固有値
[1] 15.684658  5.000000  3.315342

$vectors                       # 固有ベクトル（列ベクトル単位）
          [,1]       [,2]       [,3]
[1,] 0.8805633 -0.2357023  0.4111602
[2,] 0.3586504 -0.2357023 -0.9032244
[3,] 0.3098034  0.9428090 -0.1230161

> (w-a$values[2]*diag(3))%*%a$vectors[,2]
              [,1]
[1,] -8.881784e-16
[2,] -2.220446e-16
[3,]  2.664535e-15
```

A.4.8　特異値分解

任意の $n \times m$ 行列 $\boldsymbol{A}$ は，その階数を r とすると，(A.3) のように分解できる。

$$\underset{n\times m}{\boldsymbol{A}} = \underset{n\times r}{\boldsymbol{U}}\;\underset{r\times r}{\boldsymbol{D}}\;\underset{r\times m}{\boldsymbol{V}'} \tag{A.3}$$

ここで $\boldsymbol{U}$，$\boldsymbol{V}$ は正規直交ベクトルを列ベクトルとする行列（$\boldsymbol{U}'\boldsymbol{U} = \boldsymbol{V}'\boldsymbol{V} = \boldsymbol{I}$）であり，$\boldsymbol{D}$ は $\lambda_1, \lambda_2, \ldots, \lambda_r$ $(\lambda_1 \geqq \lambda_2 \geqq \cdots \geqq \lambda_r > 0)$ を対角要素とする対角行列である。

（A.3）より，以下のように表される。

$$\boldsymbol{A}'\boldsymbol{A} = (\boldsymbol{UDV}')'(\boldsymbol{UDV}') = \boldsymbol{VD}^2\boldsymbol{V}' \tag{A.4}$$

$$\boldsymbol{AA}' = (\boldsymbol{UDV}')(\boldsymbol{UDV}')' = \boldsymbol{UD}^2\boldsymbol{U}' \tag{A.5}$$

（A.4）は行列 $\boldsymbol{A}'\boldsymbol{A}$，（A.5）は行列 $\boldsymbol{AA}'$ のスペクトル分解（固有値，固有ベクトル分解）を表す。これにより，$\lambda_1^2, \lambda_2^2, \ldots, \lambda_r^2$ は $\boldsymbol{A}'\boldsymbol{A}$ と $\boldsymbol{AA}'$ の共通の固有値を表し，$\boldsymbol{V}$ の列ベクトルは $\boldsymbol{A}'\boldsymbol{A}$ の固有ベクトル，$\boldsymbol{U}$ の列ベクトルは $\boldsymbol{AA}'$ の固有ベクトルであることがわかる。

また，$\boldsymbol{U} = (\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_r)$，$\boldsymbol{V} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_r)$ とすると，（A.6）のように表すこともできる。

$$\boldsymbol{A} = \lambda_1\boldsymbol{u}_1\boldsymbol{v}_1' + \lambda_2\boldsymbol{u}_2\boldsymbol{v}_2' + \cdots + \lambda_r\boldsymbol{u}_r\boldsymbol{v}_r' \tag{A.6}$$

（A.3）は行列 $\boldsymbol{A}$ の特異値分解，$\lambda_1 \geqq \lambda_2 \geqq \cdots \geqq \lambda_r$ は特異値，$\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_r$ および $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_r$ は，左特異ベクトルおよび右特異ベクトルと呼ばれる。

（A.6）において，λ_{m+1} 以降の値が小さいときには，（A.7）のような近似式が成り立つ。

$$\boldsymbol{A}_m = \lambda_1\boldsymbol{u}_1\boldsymbol{v}_1' + \lambda_2\boldsymbol{u}_2\boldsymbol{v}_2' + \cdots + \lambda_m\boldsymbol{u}_m\boldsymbol{v}_m' = \boldsymbol{U}_m\boldsymbol{D}_m\boldsymbol{V}_m' \tag{A.7}$$

ここで，$\boldsymbol{U}_m$，$\boldsymbol{V}_m$ はそれぞれ $\boldsymbol{U}$，$\boldsymbol{V}$ のはじめの m 列からなる行列，$\boldsymbol{D}_m$ は $\boldsymbol{D}$ の左上の $q \times q$ 行列とする。

R で特異値分解を行う関数は svd である。

```
> A <- matrix(c(36, 64, 59, 18, 72,
+               57, 28, 54, 29, 82,
+               46, 48, 39, 30, 88), nrow=5, ncol=3)
> A
     [,1] [,2] [,3]
[1,]   36   57   46
[2,]   64   28   48
[3,]   59   54   39
[4,]   18   29   30
[5,]   72   82   88
> ans <- svd(A)                                  # 特異値分解
> (U <- ans$u)
           [,1]       [,2]       [,3]
[1,] -0.3910170 -0.4478403 -0.1883293
[2,] -0.3931531  0.7978237  0.2830433
[3,] -0.4266747  0.1922097 -0.8261713
[4,] -0.2168148 -0.2605565  0.2170691
[5,] -0.6807910 -0.2410023  0.3933707
> (V <- ans$v)
           [,1]       [,2]       [,3]
[1,] -0.5713251  0.7636848 -0.3006212
[2,] -0.5767658 -0.6341864 -0.5149261
[3,] -0.5838911 -0.1208021  0.8027939
```

```
> (D <- diag(ans$d))
         [,1]     [,2]     [,3]
[1,] 205.3679  0.00000  0.00000
[2,]   0.0000 31.73647  0.00000
[3,]   0.0000  0.00000 17.22851
> t(U) %*% U                                  # U′ U = I
             [,1]          [,2]          [,3]
[1,] 1.000000e+00  1.387779e-16  1.110223e-16
[2,] 1.387779e-16  1.000000e+00 -6.938894e-17
[3,] 1.110223e-16 -6.938894e-17  1.000000e+00
> t(V) %*% V                                  # V′ V = I
              [,1]          [,2]          [,3]
[1,]  1.000000e+00  5.551115e-17 -5.551115e-17
[2,]  5.551115e-17  1.000000e+00 -9.714451e-17
[3,] -5.551115e-17 -9.714451e-17  1.000000e+00
> t(A) %*% A                                  # (A.3)の確認
      [,1]  [,2]  [,3]
[1,] 14381 13456 13905
[2,] 13456 14514 14158
[3,] 13905 14158 14585
> t(U%*%D%*%t(V))%*%(U%*%D%*%t(V))
      [,1]  [,2]  [,3]
[1,] 14381 13456 13905
[2,] 13456 14514 14158
[3,] 13905 14158 14585
> V %*% D^2%*% t(V)
      [,1]  [,2]  [,3]
[1,] 14381 13456 13905
[2,] 13456 14514 14158
[3,] 13905 14158 14585
> A%*%t(A)                                    # (A.4)の確認
      [,1]  [,2]  [,3] [,4]  [,5]
[1,]  6661  6108  6996 3681 11314
[2,]  6108  7184  7160 3404 11128
[3,]  6996  7160  7918 3798 12108
[4,]  3681  3404  3798 2065  6314
[5,] 11314 11128 12108 6314 19652
> (U%*%D%*%t(V))%*%t(U%*%D%*%t(V))
      [,1]  [,2]  [,3] [,4]  [,5]
[1,]  6661  6108  6996 3681 11314
[2,]  6108  7184  7160 3404 11128
[3,]  6996  7160  7918 3798 12108
[4,]  3681  3404  3798 2065  6314
[5,] 11314 11128 12108 6314 19652
> U%*%D^2%*%t(U)
      [,1]  [,2]  [,3] [,4]  [,5]
[1,]  6661  6108  6996 3681 11314
[2,]  6108  7184  7160 3404 11128
[3,]  6996  7160  7918 3798 12108
[4,]  3681  3404  3798 2065  6314
[5,] 11314 11128 12108 6314 19652
> U[,1:2]%*%D[1:2, 1:2]%*%t(V[,1:2])          # (A.6)を使って，2次元で近似
         [,1]     [,2]     [,3]
[1,] 35.02459 55.32925 48.60477
[2,] 65.46595 30.51099 44.08524
[3,] 54.72105 46.67070 50.42673
[4,] 19.12426 30.92571 26.99773
[5,] 74.03737 85.48975 82.55931
```

A.5 apply 一族

行列，データフレームやリストなどの要素を対象にして一定の処理を行う関数群がある。繰り返し同じことを行うプログラムは for 文などを使って書くことができるが，R には apply，lapply，sapply，tapply，mapply があり，これらを使うほうがよい。また，tapply を使いやすくした by もある。これらは for 文を使って書くのと速度的にはあまり違わないことが多いが，簡潔にわかりやすく書くという点では優れている。

A.5.1 apply 関数

apply 関数は（A.8）のようにして使う。

```
apply(行列，計算方向の指定，関数，関数の追加引数)                (A.8)
```

apply 関数は，行列から行ベクトルまたは列ベクトルを順に取り出し，第 3 引数で指定された関数に順番に渡す。1 行ずつ取り出すときは計算方向の指定には 1，1 列ずつ取り出すときは 2 を指定する[*2]。第 4 引数以降には，第 3 引数の関数に渡す引数を指定する。

```
> (x <- matrix(1:12, 3, 4))    # 3×4行列
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> apply(x, 1, sum)             # 行ベクトルにsum関数
[1] 22 26 30                   # 結果は各行の和
> apply(x, 2, mean)            # 列ベクトルにmean関数
[1]  2  5  8 11                # 結果は各列の平均値
```

行の平均や列の平均のように，繰り返し処理が必要な操作のなかでもよく使うものについては，以下のような特別な関数が用意されている。

```
> x <- matrix(1:12, 3, 4)
> apply(x, 1, mean)            # 行の平均
[1] 5.5 6.5 7.5
> rowMeans(x)                  # こちらがお勧め
[1] 5.5 6.5 7.5
> apply(x, 1, sum)             # 行和
[1] 22 26 30
> rowSums(x)                   # こちらがお勧め
[1] 22 26 30
```

[*2] 本書の範囲を越えるが，第 1 引数は一般的には配列，第 2 引数も一般的にはベクトルである。

```
> apply(x, 2, mean)          # 列の平均
[1]  2  5  8 11
> colMeans(x)                # こちらがお勧め
[1]  2  5  8 11
> apply(x, 2, sum)           # 列和
[1]  6 15 24 33
> colSums(x)                 # こちらがお勧め
[1]  6 15 24 33
```

2 つ以上の要素を持つベクトルを返す関数の場合は，返された結果は列になり，apply 関数の結果は行列になる。

```
> x <- matrix(1:12, 3, 4)
> apply(x, 1, range) # 結果の第1列はmin(1,4,7,10)とmax(1,4,7,10)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]   10   11   12
> apply(x, 2, range) # 結果の第1列はmin(1, 2, 3)とmax(1, 2, 3)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    3    6    9   12
```

第 3 引数に指定する関数はユーザ定義の関数でよい。事前に定義した関数でもよいし，apply 関数のなかで名前を持たない関数（無名関数）として定義してもよい。関数記述の長さや行数にも制限はない。

```
> x <- matrix(1:12, 3, 4)
> apply(x, 1, mean)          # 基準とする使用法
[1] 5.5 6.5 7.5

> # 事前に定義したmy.meanを使う
> my.mean <- function(x) return(sum(x)/length(x))
> apply(x, 1, my.mean)
[1] 5.5 6.5 7.5

> # apply関数のなかで定義した無名関数を使う
> apply(x, 1, function(x) sum(x)/length(x))
[1] 5.5 6.5 7.5

> # apply関数のなかで定義する関数は長くてもよい
> apply(x, 1, function(x) {
+  s <- sum(x, na.rm=TRUE)
+  n <- sum(!is.na(x))
+  return(s/n)
+  })
[1] 5.5 6.5 7.5
```

A.5.2　lapply 関数と sapply 関数

lapply 関数と sapply 関数は（A.9）のようにして使う。

```
lapply(リスト, 関数, 関数の追加引数)
sapply(リスト, 関数, 関数の追加引数)
```
(A.9)

第 1 引数（データフレームのことが多い）の要素ごとに第 2 引数の関数を作用させて結果を返す。

lapply の場合にはリストとして返すが，sapply の場合には可能な限りベクトルや行列として結果を返す（結果の形が違うだけで，内容は lapply と同じである）。

```
> df <- data.frame(x=1:5,     # データフレーム
+                  y=c(2, 3, 1, NA, 4))
> df
  x  y
1 1  2
2 2  3
3 3  1
4 4 NA
5 5  4
```

1 個の数値を返す関数のとき，lapply では，データフレームの列ごとに関数を作用させた結果がリストで返される。

```
> lapply(df, mean, na.rm=TRUE)
$x                 # mean(1:5, na.rm=TRUE) の結果
[1] 3

$y                 # mean(c(2, 3, 1, NA, 4), na.rm=TRUE) の結果
[1] 2.5
```

sapply では，データフレームの列ごとに関数を作用させた結果がベクトルで返される。以下に述べる，2 個以上の結果を返す関数を使う場合を踏まえると，データフレームの列数を m とすれば，実はベクトルが返されるのではなく 1 行 m 列の行列が返されることがわかる。

```
> sapply(df, mean, na.rm=TRUE)
  x   y            # 第1要素はmean(1:5, na.rm=TRUE) の結果
3.0 2.5            # 第2要素はmean(c(2, 3, 1, NA, 4), na.rm=TRUE)の結果
```

2 個の数値を返す関数のとき，lapply では，データフレームの列ごとに関数を作用させた結果がリストで返される。

```
> lapply(df, range, na.rm=TRUE)
$x                 # 各要素が2個の数値を持つリストになる
[1] 1 5

$y
[1] 1 4
```

sapply では，データフレームの列ごとに関数を作用させた 2 個の結果が列になり，全体としては 2 行 m 列の行列になる。

```
> sapply(df, range, na.rm=TRUE)
     x y                        # 1列に2個の数値を持つ行列になる
[1,] 1 1
[2,] 5 4
```

3 個以上の結果を返す関数によって得られる結果は，2 個の結果を返す関数の場合から容易に推定できる。以下に，無名関数を使う場合の例を示しておこう。

```
> lapply(df, function(x) return(x^2))
$x
[1]  1  4  9 16 25

$y
[1]  4  9  1 NA 16

> sapply(df, function(x) return(x^2))
      x  y
[1,]  1  4
[2,]  4  9
[3,]  9  1
[4,] 16 NA
[5,] 25 16
```

第 1 引数はリストだけでなくて，ベクトルや行列でもよい。

```
> sapply(1:4, sin)            # sin(1), sin(2), sin(3), sin(4)
[1]  0.8414710  0.9092974  0.1411200 -0.7568025
> sin(1:4)                    # 普通はこう書く
[1]  0.8414710  0.9092974  0.1411200 -0.7568025
> sapply(1:4, "^", 2)         # 関数は"^"べき乗，2はべき乗の引数
[1]  1  4  9 16
> (1:4)^2                     # 普通はこう書く（1:4^2 とは違う）
[1]  1  4  9 16
> ( x <- matrix(1:6, 2, 3) )  # 行列
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> sapply(x, function(i) return(sum(1:i))) # 1からiまでの和を返す関数
[1]  1  3  6 10 15 21         # 要素ごとに関数に渡した結果
```

A.5.3　tapply 関数と by 関数

tapply 関数と by 関数は，(A.10) のようにして使う。

```
tapply(ベクトル，インデックス，関数，関数の追加引数)
by(データフレーム，インデックス，関数，関数の追加引数)                (A.10)
```

tapply の第 1 引数はベクトルである。第 2 引数は factor または factor にできるオブジェクトである。第 2 引数のとる値の種類別に第 1 引数を関数により処理する。

by の第 1 引数はデータフレームまたは行列である。第 1 引数の各列について tapply のときと同じように処理をする。

```
> x <- 1:6
> y <- c(2, 1, 2, 3, 5, 1)
> g <- c("a", "a", "b", "a", "b", "a")
> tapply(x, g, sum)
 a  b                         # 第1要素は1+2+4+6
13  8                         # 第2要素は3+5
> by(x, g, sum)
g: a                          # 1+2+4+6
[1] 13
------------------------------------------------------
g: b                          # 3+5
[1] 8

> (df <- data.frame(x=x, y=y, g=g))
  x y g
1 1 2 a
2 2 1 a
3 3 2 b
4 4 3 a
5 5 5 b
6 6 1 a
> tapply(df[,1:2], g, mean)   # エラーになる
 以下にエラー tapply(df[, 1:2], g, mean) :
   引数は同じ長さでなければなりません
> by(df[,1:2], g, mean)       # 各変数ごとに結果が出る
g: a
   x    y
3.25 1.75
------------------------------------------------------
g: b
  x   y
4.0 3.5
```

インデックスとする変数が複数個ある場合には（A.11）のようにそれらをリストにして使用する。

```
tapply(ベクトル, list(インデックス 1, ..., インデックス n),
       関数, 関数の追加引数)
by(データフレーム, list(インデックス 1, ..., インデックス n),
   関数, 関数の追加引数)                                      (A.11)
```

iris データセットにおいて，Sepal.Width と Sepal.Length の 2 変数それぞれが中央値以下とそれ以外の 2 区分されているとき，それぞれの組み合わせの 4 グループごとに Petal.Width の平均値を求めてみよう。

```
> sw <- iris$Sepal.Width > median(iris$Sepal.Width)
> sw <- factor(sw, levels=c(FALSE, TRUE),
+              label=c("sw.Lo", "sw.Hi"))
> sl <- iris$Sepal.Length > median(iris$Sepal.Length)
> sl <- factor(sl, levels=c(FALSE, TRUE),
+              label=c("sl.Lo", "sl.Hi"))
> tapply(iris$Petal.Width, list(sw, sl), length)
      sl.Lo sl.Hi
sw.Lo    38    45
sw.Hi    42    25
```

```
> by(iris$Petal.Width, list(sw, sl), mean)
: sw.Lo
: sl.Lo
[1] 1.115789
------------------------------------------------
: sw.Hi
: sl.Lo
[1] 0.2547619
------------------------------------------------
: sw.Lo
: sl.Hi
[1] 1.704444
------------------------------------------------
: sw.Hi
: sl.Hi
[1] 2.004
```

A.5.4　mapply 関数

mapply 関数は，sapply 関数の多変量版である。使用法は (A.12) のようになる。

```
mapply(関数, 引数 1, 引数 2, ..., 引数 n,*3
       MoreArgs=list(関数の追加引数))                    (A.12)
```

mapply 関数を使わなくても書ける場合もあるが，mapply 関数を使えばよりわかりやすく書ける場合がある。

```
> sapply(1:4, function(i) rep(i, 5-i)) # わかりにくい
[[1]]
[1] 1 1 1 1

[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

[[4]]
[1] 4

> mapply(rep, 1:4, 4:1)                # わかりやすい
[[1]]
[1] 1 1 1 1                            # rep(1, 4)

[[2]]
[1] 2 2 2                              # rep(2, 3)

[[3]]
[1] 3 3                                # rep(3, 2)

[[4]]
[1] 4                                  # rep(4, 1)
```

*3 引数 1，引数 2，...，引数 n はリストまたはベクトル。

以下のような行列を作る際は，mapply が最も簡単であろう。

```
> month <- list(a=month.name[1:4], b=month.abb[1:4],
+               c=c("睦月", "如月", "弥生", "卯月"))
> number <- list(n1=c(19, 32, 15, 36),
+                n2=c(125, 156, 324, 654),
+                n3=c(236, 348, 213, 359))
> mapply(paste, month, number, sep="-")
     a              b          c
[1,] "January-19"  "Jan-125" "睦月-236"
[2,] "February-32" "Feb-156" "如月-348"
[3,] "March-15"    "Mar-324" "弥生-213"
[4,] "April-36"    "Apr-654" "卯月-359"
```

A.6 制御構文

ここでは，R でプログラムを書くときに必要な制御構文についてまとめておく。

A.6.1 if, if-else, if-elseif-else

条件分岐には if 文を使う。（A.13）のような構文では，論理式が真の場合に式 1 が実行され，偽の場合には実行されない。式 1 は複数個あってもかまわない。式 1 が 1 つの場合には「{ }」で囲まなくてもよいが，囲むように習慣付けておくとよい。

```
if (論理式) {
    式 1
}
```
(A.13)

```
> a <- c(5, 2)              # 小さいほうを求める
> minimum <- a[1]           # 仮に，a[1]が最小値とする
> if (a[2] < minimum) {     # もしminimum < a[2]ならば
+     minimum <- a[2]       # a[2]を最小値とする
+ }
> minimum                   # 結果はどちらか？
[1] 2
```

（A.14）のような構文では，論理式が真の場合に式 1 が実行され，偽の場合に式 2 が実行される。式 1，式 2 は複数個あってもかまわない。なお，コンソールに直接記述しているような場合には if の後の{に対応する}と else は同じ行に書かなければならない。つまり，} else {のようにしなければならない。

```
if (論理式) {
    式 1
}
else {                                                    (A.14)
    式 2
}
```

```
> a <- 5                          # aを5にする
> if (a > 4) {                    # もしa > 4ならば
+     x <- TRUE                   # xをTRUEにする
+ } else {                        # さもなければ（a > 4でなければ）
+     x <- FALSE                  # xをFALSEにする
+ }
> x                               # xを表示する
[1] TRUE
```

（A.14）は式の一部として使われることもある。

```
> a <- 5
> b <- if (a > 4) TRUE else FALSE # a > 4は真
> b
[1] TRUE
> a <- 1
> b <- if (a > 4) TRUE else FALSE # a > 4は偽
> b
[1] FALSE
```

（A.15）では，論理式 1 が真の場合に式 1 が実行され，論理式 2 が真の場合に式 2 が実行され，偽の場合に式 3 が実行される。式 1，式 2，式 3 は複数個あってもかまわない。`else if (論理式 i)` は複数個あってもよい。最後の `else { 式 i }`はなくてもよい。

```
if (論理式 1) {
    式 1
}
else if (論理式 2) {
    式 2                                                  (A.15)
}
else {
    式 3
}
```

点数（score）に応じて成績を表す文字を出力する例を以下に示す。

```
> score <- 65
> if (score >= 80) {
+     print("優")
+ } else if (score >= 70) { # コンソールへ入力するときは
+     print("良")            #  }とelse ifを同じ行に書く
+ } else if (score >= 60) {
+     print("可")
+ } else {                  # コンソールへ入力するときは
+     print("不可")          #  }とelseを同じ行に書く
+ }
[1] "可"
```

A.6.2 for

for 文は（A.16）のように使用する。変数が集合の個々の値をとりながら式が繰り返し実行される。式は複数個あってもよい。

```
for (変数 in 集合) {
    式                                        (A.16)
}
```

```
> a <- 0
> for (i in 1:10) {         # forループはi=1, i=2, ..., i=10で10回まわる
+     a <- a+i
+ }
> a                         # 結果は55になっている
[1] 55
```

A.6.3 while

while 文は（A.17）のように使用する。条件式が真である間，式が繰り返し実行される。式は複数個あってもよい。論理式は式の部分でいつかは偽になるようにプログラムしなければならない。そのようにしておかないと，永久に繰り返されプログラムが終わらなくなってしまう。

```
while (条件式) {
    式                                        (A.17)
}
```

```
> a <- 0
> while (a < 30) {          # whileループはaが30より小さい間まわる
+     a <- a+1
+ }
> a                         # 結果は30になっている
[1] 30
```

A.6.4　repeat

repeat 文は（A.18）のように使用する。式が繰り返し実行される。式 1 は複数個あってもよい。式 2 はなくてもよい。論理式を評価し，真になったら break で繰り返しを終えるようにプログラムする。

```
repeat {
    式 1
    if (論理式) {
        式 2
        break
    }
}
```
(A.18)

```
> a <- 0
> repeat{                     # repeatループは無限回まわるが
+     if (a == 30) {          # aが30になると，ループから抜ける
+         break
+     }
+     a <- a+1
+ }
> a                           # 結果は30になっている
[1] 30
```

A.6.5　break と next

break と next は以下のように使用する。break は，for や while や repeat で作られるループ（実行の繰り返し）を中断する（抜け出す）ためのものである。if と組み合わせて利用する。

```
> a <- 0
> for (i in 1:5) {            # forループは5回まわるが
+     if (i == 3) {           # iが3になると，ループから抜ける
+         break
+     }
+     a <- a+1
+ }
> a                    # a <- a+1 は2回しか実行されないので，結果は2である
[1] 2
```

next は，for や while や repeat で作られるループ（実行の繰り返し）のその時点以降の繰り返し部分をパスし，次の繰り返しを開始するためのものである。if と組み合わせて利用する。

```
> a <- 0
> for (i in 1:5) {             # forループは5回まわるが
+     if (i == 3) {            # iが3のときは, a <- a+1は実行されず,
+         next                 # iを4にしてループ部分の実行を始める
+     }
+     a <- a+1
+ }
> a             # a <- a+1 は4回しか実行されないので, 結果は4になっている
[1] 4
```

A.7　関数の作成

関数は（A.19）のような形式で作る。

```
関数名 <- function(引数1, 引数2, ..., 引数n) {
    関数の定義                                                    (A.19)
}
```

数値ベクトルに格納されているデータを受け取って，平均値を計算して返す関数は，以下のような定義になるであろう*4。

▶ 平均値を計算する関数の例

```
heikinchi <- function(x)
{
    result <- 0               # 合計値を計算するための変数
    n <- length(x)            # データの個数を数える
    for (i in 1:n) {          # n個のデータに対し以下を繰り返す
        result <- result+x[i] # i番目のデータを加える
    }
    return(result/n)          # 合計値をデータの個数で割った値を
}                             # 持って，呼び出したところへ帰る
```

このように定義された関数を heikinchi.R というファイルに保存する。heikinchi 関数を利用するためには，source 関数により R に読み込む。ファイルの文字コードと OS が仮定する文字コードが違う場合には，encoding 引数で文字コードを指定しなければならない。

heikinchi 関数は，様々なデータ（引数）を持って呼ばれる。

```
> source("heikinchi.R", encoding="euc-jp") # 関数の定義を読み込む
> heikinchi(1:10)                          # 1〜10の整数値の平均値
[1] 5.5
> heikinchi(iris[,1])                      # irisデータの1列目の平均値
[1] 5.843333
> sapply(iris[1:4], heikinchi)             # irisデータの1〜4列の平均値
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
```

*4 ここでは for を使って書いたが，実際には関数の本体は return(sum(x)/length(x)) だけで十分である。

R の参考図書など

B.1 参考図書

1. 中澤 港：R による統計解析の基礎，ピアソンエデュケーション (2003/10)
2. 間瀬 茂：工学のためのデータサイエンス入門 — フリーな統計環境 R を用いたデータ解析，数理工学社 (2004/04)
3. 岡田 昌史：The R Book — データ解析環境 R の活用事例集，九天社 (2004/05)
4. 舟尾 暢男：The R Tips — データ解析環境 R の基本技・グラフィックス活用集，九天社 (2005/02)
5. 牧 厚志他：経済・経営のための統計学，有斐閣 (2005/03)
6. 荒木 孝治：フリーソフトウェア R による統計的品質管理入門，日科技連出版社 (2005/06)
7. 渡辺 利夫：フレッシュマンから大学院生までのデータ解析・R 言語，ナカニシヤ出版 (2005/09)
8. 竹内 俊彦：はじめての S–PLUS/R 言語プログラミング — 例題で学ぶ S–PLUS/R 言語の基本，オーム社 (2005/11)
9. 舟尾 暢男：データ解析環境「R」 — 定番フリーソフトの基本操作からグラフィックス、統計解析まで，工学社 (2005/12)
10. 垂水 共之, 飯塚 誠也：R/S–PLUS による統計解析入門, 共立出版 (2006/04)
11. 赤間 世紀，山口 喜博：R による統計入門，技報堂出版 (2006/09)
12. U. リゲス著，石田 基広訳：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 (2006/10)
13. Peter Dalgaard 著，岡田 昌史監訳：R による医療統計学，丸善 (2007/01)
14. 熊谷 悦生，舟尾 暢男：R で学ぶデータマイニング I — データ解析編，オーム社 (2008/11)
    (R で学ぶデータマイニング — I データ解析の視点から，九天社 (2007/05))
15. B. エヴェリット著，石田 基広他訳：R と S–PLUS による多変量解析，シュプリンガー・ジャパン (2007/06)
16. 荒川 和晴他訳：R と Bioconductor を用いたバイオインフォマティクス，シュプリンガー・ジャパン (2007/07)

17. 舟尾 暢男：R Commander ハンドブック，オーム社 (2008/11)
(R Commander ハンドブック — A Basic–Statistics GUI for R，九天社 (2007/08))
18. 樋口 千洋，石井 一夫：統計解析環境 R によるバイオインフォマティクスデータ解析 — Bioconductor を用いたゲノムスケールのデータマイニング，共立出版 (2007/09)
19. 熊谷 悦生，舟尾 暢男：R で学ぶデータマイニング II — シミュレーション編，オーム社 (2008/11)
(R で学ぶデータマイニング — II シミュレーションの視点から，九天社 (2007/10))
20. 金 明哲：R によるデータサイエンス — データ解析の基礎から最新手法まで，森北出版 (2007/10)
21. 荒木 孝治:R と R コマンダーで始める多変量解析, 日科技連出版社 (2007/10)
22. 間瀬 茂：R プログラミングマニュアル，数理工学社 (2007/11)
23. 新納 浩幸：R で学ぶクラスタ解析，オーム社 (2007/11)
24. 中澤 港：R による保健医療データ解析演習 — An R Workbook for Health and Medical Data Analysis，ピアソンエデュケーション (2007/12)
25. 長畑 秀和，大橋 和正：R で学ぶ経営工学の手法，共立出版 (2008/01)
26. 山田 剛史，杉澤 武俊，村井 潤一郎：R によるやさしい統計学，オーム社 (2008/01)
27. Michael J. Crawley 著，野間口 謙太郎，菊池 泰樹訳：統計学：R を用いた入門書，共立出版 (2008/05)
28. 高階 知巳：プログラミング R — 基礎からグラフィックスまで，オーム社 (2008/11)
(R プログラミング&グラフィックス，九天社 (2008/04))
29. 朝野 熙彦：R によるマーケティング・シミュレーション，同友館 (2008/04)
30. 田中 孝文：R による時系列分析入門，シーエーピー出版 (2008/06)
31. 古谷 知之：ベイズ統計データ分析 — R & WinBUGS，朝倉書店 (2008/09)
32. P. スペクター著，石田 基広，石田 和枝訳：R データ自由自在，シュプリンガー・ジャパン (2008/10)
33. 石田 基広：R によるテキストマイニング入門，森北出版 (2008/12)
34. 秋山 裕：R による計量経済学，オーム社 (2008/12)
35. 豊田 秀樹：データマイニング入門 — R で学ぶ最新データ解析，東京図書 (2008/12)
36. 神田 範明 監修，石川 朋雄，小久保 雄介，池畑 政志：商品企画のための統計分析 — R によるヒット商品開発手法，オーム社 (2009/03)

B.2 Web サイト

1. 青木繁伸：R による統計処理
   http://aoki2.si.gunma-u.ac.jp/R/index.html
2. 石田 基広：R と Linux と...
   http://cms.ias.tokushima-u.ac.jp/index.php
3. 岡田 昌史：RjpWiki
   http://www.okada.jp.org/RWiki/index.php
4. 奥村 泰之：無料統計ソフト R で心理学 — Passepied —
   http://blue.zero.jp/yokumura/R.html
5. 加藤 悦史：R を利用する
   http://hosho.ees.hokudai.ac.jp/%7ekato/unix/R.html
6. 金 明哲：R 言語と WEKA など
   http://www1.doshisha.ac.jp/%7emjin/R/index.html
7. 久保 拓弥：生態学のデータ解析 — 統計学授業
   http://hosho.ees.hokudai.ac.jp/%7ekubo/ce/EesLecture.html
8. 里村 卓也：マーケティング・サイエンスの道具箱
   http://www.fbc.keio.ac.jp/%7esatomura/RdeMarketingScience/index.html
9. 下平 英寿：データ解析
   http://www.is.titech.ac.jp/%7eshimo/class/data2007/index.html
10. 竹内 昌平：R on Windows
    http://plaza.umin.ac.jp/%7etakeshou/R/
11. 竹澤 邦夫：〈R〉によるノンパラメトリック回帰
    http://cse.niaes.affrc.go.jp/minaka/R/R-NonparaRegression.pdf
12. 立川 察理：R 言語による医学統計
    http://akimichi.homeunix.net/hiki/biostat/
13. 田畑 智司：統計解析言語 R で多変量解析を行う
    http://www.lang.osaka-u.ac.jp/%7etabata/JAECS2004/multi.html
14. 中澤 港：統計処理ソフトウェア R についての Tips
    http://phi.med.gunma-u.ac.jp/swtips/R.html
15. 舟尾 暢男：続・わしの頁
    http://cwoweb2.bai.ne.jp/%7ejgb11101/
16. 間瀬 茂：オープンソース統計解析システム R について
    http://www.is.titech.ac.jp/%7emase/R.html

17. 松井 孝雄：言語 R による分散分析
http://mat.isc.chubu.ac.jp/R/tech.html
18. 三中 信宏：租界 R の門前にて — 統計言語「R」との極私的格闘記録
http://cse.niaes.affrc.go.jp/minaka/R/R-top.html
19. 森 厚：R の日本語文章
http://buran.u-gakugei.ac.jp/%7emori/LEARN/R
20. 山本 義郎：R — 統計解析とグラフィックスの環境
http://stat.sm.u-tokai.ac.jp/%7eyama/R/
21. 和田 康彦：統計パッケージ R
http://genome.ag.saga-u.ac.jp/R/
22. メディアラボ株式会社：Linux で科学しよう！ — R
http://www.mlb.co.jp/linux/science/R/

記号

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X