2009 年　日本行動計量学会　春の合宿セミナー（長崎大学） 資料

# R 入門

2009 年 3 月

大津起夫 [1] ・ 中島　晃[2]

[1] 独立行政法人 大学入試センター 研究開発部
[2] 北海道大学 文学研究科 人間システム科学専攻

# 目 次

# 第 I 部
# 基礎知識

## 1　はじめに

[3]

本稿では、データ解析のためのソフトウェアである R の利用法について、入門的な解説を行う。取り扱う範囲は、おもに基本的なデータ操作、２群データの比較および線形モデルを用いたデータ分析法、データのグラフィック表示についてである。

大規模なデータを扱うことも工夫により可能ではあるが（ODBC を用いたデータベースとの連携、および 64bit Linux 上で高速な数値計算機能を利用するための R システムの構築法など）ここでは触れない。

従来、社会科学・行動科学の関連分野において利用されるデータ分析用のソフトェアは、定型的な入力とそれに応じた定型の帳票形式の出力を備えたものが主流であった。現在では、数値計算やグラフィックス機能を強化したコンピュータ言語としての特徴をもつシステムが，特にデータ解析や技術計算の分野で広く利用されるようになった。

次のような傾向が、このようなシステムの利用が一般的になったことに影響している。

1. 伝統的な統計ソフトウェアで採用している関係表形式のデータ構造は、定型業務データを表現するのには向いているが、研究データの表現としては必ずしも使いやすくはない。

2. J.W.Tukey らによる探索的データ解析 (EDA) の方法論等の影響により、柔軟で対話的なデータ解析環境への要求が増えた。

3. 統計理論の急速な進展によって手法の多様化が著しいため、固定された機能しか持たないソフトウェアでは対応ができない。利用者がプログラムを自ら記述するなどによって機能拡張を実現する必要が多い。

4. 計量モデルの利用技術が向上するにつれて、利用者がカスタムメイドのモデルを利用することが多くなってきた。

5. 乱数シミュレーションの利用が一般的になった。

過去においては，これらの要求を満たすには、研究者が独自に FORTRAN や C などのコンパイラ言語を用いて、入出力まで含めたプログラムを自作していた。最近開発されている数学・統計ソフトウェアにおいては、独自の高水準プログラム言語を備えることにより、上記の要求を満たすことができる。これらを使いこなすためには、独自言語の利用方法を学ぶ必要があるが、プログラムを全て自作するのに比べれば大幅に作業効率が向上する。

本稿は R のすべての機能について説明しているわけではなく、主に入門的な話題に限定している。詳しい解説は、英文の R 利用者マニュアルに記述されている。

## 2　R の特徴

R はデータをメモリ上に展開して操作するため、小〜中規模の研究データの分析に向いている。対話的なデータ分析をグラフィカル機能を用いて柔軟に行える。各種の分析機能は「関数」として組み込まれて

[3] 本稿にあらわれるシステム名・製品名などは一般にそれらの開発元の商標または登録商標です。

おり、分析にかかわる簡単なサマリーを表示する関数が用意されているものもあるが、SAS や SPSS の様に常に分析結果が一覧表の形で表示される訳ではないので、利用者が自分で各種の指定を行わなければならない。特に、探索的データ解析 (Exploratory Data Analysis) [25] と呼ばれる各種の手法やノンパラメトリック回帰関連の機能が充実している。

日本語の教科書もここ数年出版されるようになったので、情報の入手は容易であるが、個別の分析手法については、英文のマニュアルしか存在しない場合がある。SAS などとは異なりマニュアル中には理論的な解説はほとんど記述されていないので、詳しい機能を知るためには関係する教科書など文献を参照する必要のある場合もある。

マイクロソフト社の Windows(32bit) 上では、利用者のアプリケーションプログラムが利用できるメモリは原則として 2GB が上限であり，極めて大規模のデータを扱うとこの上限のために処理を行えない場合がある。特に R においては数値データは全て 8 バイトの倍精度浮動小数点数として保持され、単精度数は利用できないため、メモリの利用量は多くなる傾向がある。また、残念ながら現状 (2009 年 1 月) では Windows 用の 64bit 版 R は開発されていないため[4] 、大規模データを R で取り扱うには 64bit の Linux や Unix 用のシステムを用いる必要がある。

R の原型となった S 言語はデータ解析用のソフトウェアとして AT&T の Bell 研究所のデータ解析研究グループによって開発された。R はこれと類似の言語仕様を持つよう開発されたオープンソースのソフトウェアである。オリジナルのディストリビューションは

`http://www.r-project.org/`

またはこのミラーサイトから入手できる。また、日本語への対応については、岡田昌史氏のサイト

`http://www.okada.jp/org/RWiki/`

から情報を入手できる。

R には多少の記号微分の機能はあるが、Matematica や Maple などの本格的な数式処理言語のような機能はない。また、インタプリタ言語であり、現在のところ最適化を行うコンパイラは存在しないため、繰り返し処理やテキストファイルの入出力などの速度は比較的遅い。

R の原型である S システムでは、作成されたベクトルや行列などのそれぞれのデータ（オブジェクトと呼ばれる）は、基本的に個別にディスク上に保存されるが、R では作業領域全体のイメージをひとつのファイルとして保存する。このため、S と比較して実行速度が向上する利点はあるが、実行中にシステム障害が発生すると、途中の計算経過がすべて失われる危険がある。

R は計算機統計の国際的な研究者のボランティア集団によってサポートされており、現在のところ商用サポートは存在しない。しかしながら、フリーにソースコードが公開されているため、各種の拡張機能が利用者によって作成されており、共用ライブラリとして提供されている。また、統計の専門的な教科書には、分析手法が S または R 言語によって記述されているものも増えており，英文では R による統計分析をテーマとしたモノグラフのシリーズも出版されている。(Springer の USE R!シリーズなど).

# 3 R の基本機能

## 3.1 起動

- Linux/Unix の場合：コマンドモードで`R` と入力する。先頭は大文字、それ以外は小文字である。

[4] ただし 32bit 版の R を 64bit 版の Windows で利用することは可能である。

- Windows の場合、スタートメニューから選択し、R システムを起動すると GUI 環境が利用できる。また，R の実行形式が含まれるディレクトリを環境変数 PATH に加えることにより、コマンドウィンドウ内で実行することができる。R-2.8.0 の Windows 版の場合には，C:\Program Files\R\R-2.8.0\bin を環境変数 PATH に加えると

  c:\... >Rterm --no-save < R スクリプト名

  ようにしてコマンドライン版の R を実行できる。ただし、日本語の処理方法が GUI 版と異なる場合が版によってある。ここで，--no-save は、実行結果得られた R の内部状態をファイルとしては残さないことの指定である。実行結果を保存する場合には--save と指定する。

  Windows 上の GUI 版のファイル名は Rgui.exe であり、アイコンをクリックするとこちらのプログラムが起動される。

Linux/Unix の場合には、ESS(Emacs Speaks Statistics) という名称の Emacs 用のマクロを導入することにより、Emacs 上で便利に使うことができる。(Windows 上でも Meadow あるいは Windows 用の Emacs を使えば利用は可能であるが、言語環境などの設定に注意を払う必要がある。)

次は ESS を Windows 上の GNU Emacs で利用する場合の.emacs での設定例である。Vista x64 環境で一応動いている。大津の環境では Emacs が LANG 環境変数を（現在では非標準的な）JPN と設定してしまうので、これを防ぐためにあらかじめja_JP.cp932 と指定している（正式な指定ではない)。コマンドモードで利用のためには、R の実行ファイルを含むディレクトリが、環境変数 PATH に含まれるように設定しておく必要がある。

```
;; ESS のインストール箇所を指定
(load-file "c:/usr/ess-5.3.11/lisp/ess-site.el")

;; shell(cmd) バッファで日本語エンコードディング cp932
;; (シフト JIS) を使うための指定
(add-hook 'shell-mode-hook
   (lambda ()
     (set-buffer-process-coding-system 'cp932 'cp932 )
     (ansi-color-for-comint-mode-on))
)

;; 次は Emacs のメニュー設定で自動的に設定されたもの。
(custom-set-variables
  ;; custom-set-variables was added by Custom.
  ;; If you edit it by hand, you could mess it up, so be careful.
  ;; Your init file should contain only one such instance.
  ;; If there is more than one, they won't work right.
 '(current-language-environment "Japanese"))

;; R を Esc-x R で起動する場合の日本語対応の指定。
(setq ess-pre-run-hook
      '((lambda () (setq S-directory default-directory)
```

```
  (setq default-process-coding-system '(cp932 .   cp932))
  )))
(setq inferior-R-args "LANG=")
```

[5]

Windows 版では GUI 環境から、メニューによってダウンロードサイトを指定し、機能拡張のためのパッケージの追加更新を行うことができる。追加パッケージがインストールされるディレクトリは版によって異なるが、最近の版では指定により、利用者個人用のパッケージディレクトリにインストールすることができる。g また J.Fox によって Rcmdr(R commander) という強化された GUI が開発されており、商用システム風の GUI メニューから各種の分析法を利用できるが、ここでは触れない。

Windows 上ではデスクトップまたはスタートメニューのアイコンをクリックすると GUI 版の R(Rgui.exe) が起動する。ただし、このままであるとしばしば作業ディレクトリが、R のインストールされたディレクトリになってしまうので、メニューの「ファイル」−>「ディレクトリの変更」によって、作業を行うディレクトリを指定するか、R アイコンのプロパティで作業フォルダを指定しておく。

## 3.2 終了方法

コマンドモードで

```
q()
```

と入力する。ここで丸括弧 () は必須である。通常は、ここで「作業を保存しますか？」とのポップアップ画面が表示される。「はい (Y)」をクリックすると、計算結果などの作業内容が、`.Rdata` という名のファイルに格納される。　GUI の場合には、メニューから「ファイル」「終了」を選択する。

Windows では次の起動時に、この `.Rdata` をクリックすると R が自動的に起動し、前回の作業内容が復元される。ただし、`library` コマンドによって取り込んだパッケージの関数やデータは、次回起動時には自動的には登録されないので、起動時に再び設定する必要がある。

## 3.3 入力行の編集

コマンドモードで、矢印や BS キーによる行編集が可能である。Emacs に類似のコントロールキーと文字キーとの組み合わせの利用も可能である。

一般に R で作業をするには関数の呼び出しの形式をとる。コンピュータ用語で「関数」とは、引数を与えると、何かしらの値を計算して返すものという意味である。ある関数を利用するには、

関数名 ( 引数１ , 引数２ , ... )

のように記述する。引数が存在しない場合にも () は必要である。これを記述せずに関数名のみを入力すると、関数の定義内容が画面に表示される。

[5] 現状で Linux で日本語を取扱う場合には UTF-8 が標準的である。Windows では R の版により扱いが異なるらしい。Windows 上でも Unicode での文字表現が次第に標準になるとおもわれるが、現状では日本語を含むデータの互換性が Linux 上のステムと Windows 上のシステム間にない。日本語のデータを保存する場合には、テキストファイルに dump するなどして保存したほうが、今後のシステムの変更に際して安全と思われる。また、RODBC を用いて MySQL と接続する場合には、Linux 版の R を文字コード UTF-8 で利用し、データベースも UTF-8 を利用するのが安全のようだ。

| キー | 機能 |
|---|---|
| Ctrl+F | 一文字前進 |
| Ctrl+B | 一文字後退 |
| Ctrl+P | 一つ前に入力されたコマンドを表示 |
| Ctrl+N | 表示されているコマンドのつぎに入力されたコマンドを表示 |
| Ctrl+D | カーソル位置にある一文字を削除 |
| 文字キー | カーソル位置の前に入力 (入力モードに移行) |
| Ctrl+A | 行の先頭に移動 |
| Ctrl+E | 行の最後に移動 |
| Ctrl+K | カーソル以降行の最後までを削除 |

## 3.4 データの代入

記号 <- が値の代入をあらわす。C や Fortran とは異なり、代入される変数を宣言する必要はなく、また代入時に変数の型が決定される。記号の左辺は、値が代入される変数を表し、右辺は代入する値である。R は数値と文字列を扱える。ダブルクオート"またはクオート' で囲まれた文字は、文字列として扱われる。文字列については計算は行なえない (異同判断と大小比較は可能である)。変数および関数の名前は英字、ピリオド (.) または数字 1 文字以上からなるもので、先頭文字は数字であってはならない。R が扱うことのできるデータ構造には、ベクトル、行列、リストなどがある。本稿ではベクトルと行列について主に説明する。また実数 (スカラー) は、長さ 1 のベクトルとして扱われる。

R のプログラム中では、#記号はその右側が注釈であることを意味する。この資料では、R との応答例においても#記号の右側は注釈であることにする。

```
\begin{verbatim}
> a = 123.45                    #  変数 a に 123.45 を代入。
> b <- 567                      #   b に 567 を代入。
> d <- "Sapporo"                #   d に 文字列 Sapporo を代入。
> ab1 <- c(1,2,3,4,5)           #   ab1 に ベクトル を代入。
> cd2 <- c("aaa","bbb","ccc")   #   cd2 に文字列のベクトルを代入。
```

上で用いているc という名前の関数は、値をまとめてベクトルにするものである。同名のデータを作成しても、多くの場合問題は生じないが、混乱しないよう注意する必要がある。作成したデータは R 終了時の指定により、ファイルとして保存される（通常は .RData というファイル名)。

また、複素数も取扱可能である。i が虚数単位をあらす。

```
>  1.0i * 1.0i
[1] -1+0i
>  (1.0-1.0i)^2
[1] 0-2i
```

Re(実数部),Im(虚数部),Mod(複素数の絶対値), Arg(偏角), Conj(共役) の関数が利用できる。

```
>  z <- 3+4i;
> Re(z)
[1] 3
> Im(z)
[1] 4
> Mod(z)
[1] 5
> Arg(z)
[1] 0.9272952
> Conj(z)
[1] 3-4i
```

R の機能は、基本的に各種の関数の参照と、それによって作られた値の代入から成り立っている。もし、代入を行なわず右辺の部分だけを入力すると、

```
> c(1,2,3,4,5)
[1] 1 2 3 4 5
```

のように、指定された内容の値を表示するだけである。先頭部分の[1] は、表示されている部分の先頭がベクトルの第 1 番目の要素であることを示している。また、R においてはデータと関数とが形式上区別されないため (両者ともにオブジェクトと呼ばれる)、システムが用意する関数と同名のデータを作成すると、関数の内容を表示しようとしても、データの内容が表示されてしまう。通常は、あらかじめ準備された関数が保存されているディレクトリと、利用者のデータが保存されるディレクトリは異なるので、関数の内容が消滅してしまうことはない。また、関数を利用すると、引数の指定があることからデータが参照されたのではないことが自動的に判断される。利用者が作成したデータは、Linux/Unix システムにおいては、通常利用者の作業ディレクトリの下の.RData という名のファイル保存される (ファイルの拡張子が表示される指定になっていないとファイル名が表示されない)。作業用のディレクトリを変更する場合には、Windows の GUI 版ではメニュー「ファイル」「ディレクトリの変更」によって行う。

現在の作業ディレクトリおよび関数とデータが参照されているオブジェクトの集合の一覧（R の用語で environment と呼ばれる）は search 関数によって表示される。先頭の.GlobalEnv が利用者の作業領域をあらわす。２番目以降の要素は参照されているパッケージ（関数とデータのライブラリ）をあらわす。

```
> search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

これらのオブジェクトが、コンピュータのどのディレクトリに保存されているかは、次の searchpaths 関数で表示される。

```
> searchpaths()
[1] ".GlobalEnv"
[2] "c:/PROGRA~2/R/R-28~1.1PA/library/stats"
[3] "c:/PROGRA~2/R/R-28~1.1PA/library/graphics"
[4] "c:/PROGRA~2/R/R-28~1.1PA/library/grDevices"
[5] "c:/PROGRA~2/R/R-28~1.1PA/library/utils"
[6] "c:/PROGRA~2/R/R-28~1.1PA/library/datasets"
[7] "c:/PROGRA~2/R/R-28~1.1PA/library/methods"
[8] "Autoloads"
[9] "c:/PROGRA~2/R/R-28~1.1PA/library/base"
```

## 3.5 オブジェクトの表示

オブジェクト (データまたは関数) の名前のみを入力すると、その内容が画面に表示される。変数名の後につづけて[自然数] と指定すると、指定された添字に対応する要素が表示される。ただし、数として 0 を指定すると、データの要素ではなく、データ型についての情報が得られる。添え字は 1 が先頭要素を示す。添え字 0 を指定すると、そのデータの型についての情報が示される。

```
> a
[1] 123.45
> ab1
[1] 1 2 3 4 5
> ab1[3]                    # 3 番目の要素を表示する。
[1] 3
> cd2
[1] "aaa" "bbb" "ccc"       # 文字列は"で囲まれて表示される。
> cd2[0]
character(0)                # 文字型データであることがわかる。
```

同様に関数名のみを入力すると、その関数の定義内容が表示される。次の例では、関数c の内容を表示している。

```
> c
function (..., recursive = FALSE)  .Primitive("c")
```

Linux などのコマンドモードを利用している場合、オブジェクトが多くの要素を含むと表示内容が 1 画面に収まらず流れてしまう場合がある。この際には、page 関数を用いて

```
> page(airquality)
```

のように指定すると、 1 ページ毎に表示が停止する。ここで airquality は、R が提供しているサンプルデータの名前である。

## 3.6 作成済オブジェクトの一覧

関数objects は、作成済みオブジェクトの名前の一覧を、文字列を要素とするベクトルとして与える。

```
> objects()
[1] "a"   "ab1" "b"   "cd2" "d"   "z"
```

## 3.7 オブジェクトの削除

関数 rm によって、不要なオブジェクト (データまたは自作の関数) を削除できる。削除したあとは復活できない。

```
> rm(a)       # オブジェクト a を削除する。
```

## 3.8 利用説明の表示

関数help は、関数や R が用意しているサンプルデータの使用法や内容を説明する。特殊記号についての説明を表示する場合には、記号を"で囲んで指定する。特定の文字を含む話題を検索するには help.search を用いる。

```
> help(help)
> help(c)
> help(ls)
> help("*")
> help.search(glm)
```

Windoes の GUI 版では、Windows Help による説明が表示される。Linux 版や ESS 利用時にはテキスト版が表示される。Linux 上でhelp 命令による説明画面が表示されたなら、次の操作が可能である (Linux の less コマンドと同じである)。

| キー | 機能 |
|---|---|
| スペース (空白) | 1 画面分先へ進む。 |
| b | 1 画面戻る。 |
| h | 操作方法の説明を表示する。 |
| q | 説明の表示を終える。 |

また、Linux 上では次のように指定することにより、X Window System 上のグラフィカル・インタフェースを用いた説明が利用可能である。 Windows 版では HTML ヘルプが表示される。

```
> help.start()
```

## 3.9 Emacs/Meadow 上での R の利用

R(および他のプログラミング言語) を利用するのに、Linux/Unix 上では Emacs エディタの環境を用いると便利である。計算を実行過程や表示結果がバッファに保存することができる。また Emacs の編集機能を用いて、結果の整理やレポートを作成できる。R や S,SAS など統計ソフトウェア用の Emacs インタフェースは ESS (Emacs Speaks Statistics) というプロジェクトで開発されている。http://ess.r-project.org/ から Emacs の拡張用コードを入手できる。

# 4 データの基本操作

## 4.1 ベクトルの算術演算

数、ベクトルおよび行列を対象とする各種の計算が簡単な記述で行なえる。特にベクトルの操作はRにおいて基本となる機能である。ベクトルの加減乗除および数学関数の適用は、要素毎に実行される。2つのベクトルの加減乗除を行なう場合、ベクトルの長さが一致しない場合には、短い方のベクトルの内容が最初の要素に戻って再び用いられる。特に、実数 (長さ1のベクトル) との演算では、他方のベクトルの全ての要素との計算が行なわれる。

```
> a <- 123.45
> a + b                 # 実数と実数の和を求める。
[1] 690.45
> a + ab1               # 実数をベクトルの各要素に加算する。
[1] 124.45 125.45 126.45 127.45 128.45
> ab1/a                 # 実数による各要素の除算。
[1] 0.008100446 0.016200891 0.024301337 0.032401782 0.040502228
> ab1 * ab1             # 要素毎の乗算。
[1]  1  4  9 16 25
> ab1 / ab1             # 要素毎の除算。
[1] 1 1 1 1 1
> ab1^2                 # 2乗
[1]  1  4  9 16 25
> sum(ab1)              # ベクトル要素の総和を求める。
[1] 15
> prod(ab1)             # 全ての要素の積を求める。
[1] 120
> length(ab1)           # 要素の数 (ベクトルの長さ) を求める。
[1] 5
```

2つのベクトルの演算を行なう場合、原則として長い方のベクトルの長さは、短いものの整数倍であることが仮定されている。もし、この条件が満たされていなければ、つぎのように警告 (warning) が表示される。

```
> p1 <- c(10,20,30,40,50)
> p2 <- c(1,2)
> p1 + p2
> p1+p2
[1] 11 22 31 42 51
Warning message:
In p1 + p2 :
   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
```

また、欠測値はNAとして表現される。ただし全ての関数が欠測値を持つベクトルの入力を許している訳ではない。また、無限大の大きさを持つ数値はInfで表される。 浮動小数点の演算結果が、表現可能なものでない場合にはNaNとなる。

```
> sqrt(c( 3,2,1,0,-1,-2))    # 負の数の平方根は求められない。
[1] 1.732051 1.414214 1.000000 0.000000      NaN      NaN
Warning message:
In sqrt(c(3, 2, 1, 0, -1, -2)) :  計算結果が NaN になりました
> ab3 <-  1/c( 3,2,1,0,-1)   # 0 の逆数は無限大。
> ab3
[1]  0.3333333  0.5000000  1.0000000        Inf -1.0000000
>
```

上の NaN は欠測値として扱われるが、Inf は欠測値とはみなされない。

表 1: 主な算術演算子一覧

| 演算子 | 機能 | 演算子 | 機能 | 演算子 | 機能 |
|---|---|---|---|---|---|
| + | 加算 | - | 減算 | * | 乗算 |
| / | 除算 | ^ | べき乗 | %/% | 整数の除算 |
| %% | 整数除算の剰余 | %*% 行列積 | | | |

## 4.2 数学関数と数列・並べ替え

R では多くの数学関数が利用可能であるが、これ以降で必要なものを中心に、いくつかを表 2 に示す。ベクトルを引数として数学関数を適用すると、ベクトルの各要素に関数が適用された値が新たなベクトルとして作られる。

また、2 つの数をコロンで区切ったもの (数 1 : 数 2) は数 1 からはじまり 1 ずつ増加して数 2 に至る数列を意味する。数 2 が数 1 より小さい場合には、1 ずつ減少する数列が得られる。数列を指定するには、関数seq も用いることができる。こちらは、増分の大きさを指定することができる。

関数sort は、データの要素を昇順に並べ換える。

```
> sin(ab1)              # ab1 の要素の sin 関数値を求める。
[1]  0.8414710  0.9092974  0.1411200 -0.7568025 -0.9589243
> 1:9                   # 1〜9 の数列。
[1] 1 2 3 4 5 6 7 8 9
> -3:3                  # -3〜3 の数列
[1] -3 -2 -1  0  1  2  3
> seq(0,10,1.5)
[1] 0.0 1.5 3.0 4.5 6.0 7.5 9.0  # 0 から 10 までの 1.5 間隔の数列
> sort(sin(ab1))        # ab1 の各要素について sin を求め、昇順に並べる。
[1] -0.9589243 -0.7568025  0.1411200  0.8414710  0.9092974
```

表 2: 数学関数一覧

| 演算子 | 機能 | 演算子 | 機能 | 演算子 | 機能 |
|---|---|---|---|---|---|
| abs | 絶対値 | exp | 指数関数 | log | 自然対数 |
| log10 | 常用対数 | sqrt | 平方根 | sin | 正弦関数 |
| cos | 余弦関数 | tan | 正接関数 | | |

| 演算子 | 機能 | 演算子 | 機能 | 演算子 | 機能 |
|---|---|---|---|---|---|
| ceiling | 整数への切上げ | floor | 整数への切捨て | trunc | 絶対値の切捨て |
| round | 四捨五入（指定精度への丸め） | | | | |

## 4.3 指数と対数

### 4.3.1 指数

数式の記法で $10^3$ のように他の数の肩の部分につく数のことを指数とよぶ。これは 10 の 3 乗、つまり 10 を 3 回掛け合わせることを意味する。指数は自然数だけでなく、分数や小数、負の数であってもよい。$10^{1/2}$ は 10 の平方根 ($\sqrt{10}$ つまり 2 回掛け合わせると 10 になる数) を表し、$10^{1/3}$ は 10 の立方根 (3 回掛け合わせると 10 になる数) を表す。$10^{2/3}$ は 10 の立方根の 2 乗を表す。また、指数が負の数であることは、逆数を表す。つまり $10^{-2}$ は $1/10^2 = 1/100$ である。

一般的に実数 $a, b$ と任意の正の数 $c$ とについて、次の公式がなりたつ。

$$
\begin{aligned}
c^0 &= 1 \\
c^1 &= c \\
c^{1/2} &= \sqrt{c} \\
c^{-a} &= 1/c^a \\
c^{a+b} &= c^a c^b \\
(c^a)^b &= c^{ab}
\end{aligned}
$$

上の公式は、指数 (冪乗をあらわす数) が整数の場合には、$c$ が負の数であっても成立する。

### 4.3.2 自然対数の底

年間 100 パーセントの利子がつく預金を考える (普通にはないが)。1 年間に利子が元金に加えられる階数が 1 回であれば、1 年後の預金は 2 倍になり、2 年後には 4 倍、$n$ 年後には $2^n$ 倍になる。また、半年に 1 回利子が元金に加えられるならば、半年あたりの利子は元金の 1/2 倍であるから、半年後には元金は 1.5 倍になり、1 年後には元金は $1.5^2$ 倍になる。同様にして、$n$ 年後には $(1+1/2)^{2n}$ 倍になる。もし、1 年間に利子が元金に加えられる回数が $k$ 回であるならば、1 年後には元金は $(1+1/k)^k$ 倍になり、$n$ 年後には $(1+1/k)^{kn}$ 倍になる。$k$ をどんどん大きくしてゆくと (つまり瞬間複利計算を考えることになる)、$(1+1/k)^k$ は発散せずにある数に収束する。これが**自然対数の底**と呼ばれるものであり、$e = 2.71828...$ として表される。理論的に $e$ は重要な数であり、多くの数学公式に現れる。$e$ を底とする指数関数 $e^x$ は $\exp(x)$ とも表記される。次の公式が知られている。

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

### 4.3.3 対数

記号 $\log_a b$ は $a$ を底（てい）とする $b$ 対数と呼ばれる。これは、$a$ を何乗すれば $b$ になるか、その回数を表す。つまり、

$$\log_2 1 = 0,\ \log_2 2 = 1,\ \log_2 4 = 2,\ \log_2 8 = 3,\ \log_2 \frac{1}{2} = -1 \tag{1}$$

などとなる。対数は指数関数の逆関数として定義されるものである。上の例では対数の値は整数になっているが、一般的には整数でない場合も考える。

対数の底が 10 のものを常用対数とよび、底を $e = 2.71828...$ とするとき自然対数と呼ぶ。工学系の文献では自然対数は ln で表し、log は常用対数を表すこともあるが、理論的な文献では log によって自然対数を表すことが多い。本稿では、log は自然対数を表すことにする。

一般に次の公式が成り立つ。但し、ここで $a$,$b$,$c$,$d$ はいずれも正の実数とし、$a \neq 1$、$d \neq 1$ とする。

$$\log_a(bc) = \log_a b + \log_a c \tag{2}$$

$$\log_a \frac{1}{b} = -\log_a b \tag{3}$$

$$\log_a b = \frac{\log_d b}{\log_d a} \tag{4}$$

$$\tag{5}$$

最初の式 (2) はつぎのようにして導ける。$\log_a b = x$ とし、また $\log_a c = y$ とすると、$a^x = b$ および $a^y = c$ である。両者をかけると、$a^{x+y} = bc$ であることから分かる。

2 番目の式 (3) は、$a^x = 1/b$ とすると、$a^{-x} = b$ より分かる。

3 番目の式 (4) については、まず $a^x = b$ とおく。更に $d^y = a$ とすると、$(d^y)^x = d^{yx} = b$ である。これより、$\log_d b = yx$、また $\log_d a = y$ であるので、2 項の商は $x$ である。

### 4.3.4 R での指数と対数

R では `log` は自然対数を表し、`log10` が常用対数を表す。ある数の冪 (べき) 乗は`^`で表す。また `exp` は自然対数の底の冪乗 (指数関数) を表す。

```
> 10^2
[1] 100
> 10^3
[1] 1000
> exp(1)
[1] 2.718282
> exp(2)
[1] 7.389056
```

**課題**

$y = 1/x$、$y = \sqrt{x}$、$y = \log x$、$y = \exp(x)$ などのグラフを適当な $x$ の範囲について表示しなさい。

例えば

```
> x <- seq(0,5,0.05)  # x はどのような数列になるか確認しなさい。
> plot(x,sqrt(x))
> plot(x,sqrt(x),type="l")
```

または

```
> x <- seq(0,5,0.05)
> y <- sqrt(x)
> plot(x,y)
> plot(x,y,type="l",col=2,main="赤い平方根")
```

## 4.4 比較と論理演算

算術演算と同様に、ベクトル要素についての比較判断および論理演算を行なうことができる。比較演算の結果はT(真) またはF(偽) となる。これらの名前は R において論理値をあらわす特別なものであり、データや関数の名前とすることができない。真偽値 (T とF) を持つベクトルは、数値ベクトルと同様に算術演算の対象とすることができる。T は 1、またF は 0 と見なされる。

```
> 1:5
[1] 1 2 3 4 5
> 5:1
[1] 5 4 3 2 1
> 1:5 > 5:1
[1] FALSE FALSE FALSE  TRUE  TRUE
                    # 要素毎の比較判断の真偽値がベクトルになる。
>  5>4               # 実数の比較の結果は長さ 1 のベクトル。
[1] TRUE
>  4>5
[1] FALSE
```

文字列の大小判断は、コード表の順による。

```
> "a" > "b"
[1] FALSE
> "a" < "b"
[1] TRUE
> kk <- 1:5 > 3     # kk に真偽値を値とするベクトルを代入。
> kk
[1] FALSE FALSE FALSE  TRUE  TRUE
> kk+0              # 算術演算の対象とすることができる。
[1] 0 0 0 1 1
```

論理演算もまたベクトルの要素毎に値が計算される。 &は論理値をとるベクトルについての論理積（かつ)、| はベクトルについての論理和（または）を求める。

```
> 1:5>2              # 数列のうち値が 2 より大きなもののある位置が T。
[1] FALSE FALSE  TRUE  TRUE  TRUE
> 1:5<4              # 4 より小さいもののある位置が T となる。
[1]  TRUE  TRUE  TRUE FALSE FALSE
>  1:5>2 & 1:5<4    # 2 つの真偽値ベクトルの論理積をとる。
[1] FALSE FALSE  TRUE FALSE FALSE
> 1:5>2 | 1:5<4     # 論理和をとる。
[1] TRUE TRUE TRUE TRUE TRUE
```

表 3: 比較演算子一覧

| 演算子 | 機能 | 演算子 | 機能 | 演算子 | 機能 |
|---|---|---|---|---|---|
| == | 等しい | != | 等しくない | <= | 以下 |
| >= | 以上 | < | より小 | > | より大 |

表 4: 論理演算子一覧（ベクトルを引数とするもの）

| 演算子 | 機能 | 演算子 | 機能 | 演算子 | 機能 |
|---|---|---|---|---|---|
| ! | 否定 | & | 論理積 (かつ) | \| | 論理和 (または) |

記号&&と||は、ベクトルではなく一つの値についての論理判断（それぞれ論理積と論理和）を意味する。

## 4.5 条件分岐

R では多くのデータ操作はベクトルの演算として実行するため、FORTRAN や C のような実行の流れを詳細に指定する必要は必ずしも多くはないが、時には処理の手順を指示する必要が生じる。条件による処理の分岐は if または if ... else ... を用いる。ここで、if の次にくるものは論理式であって、長さは 1 でなければならない (論理値または 0 – 1 を持つベクトルを指定することはできない)。最初の例 (if ) は X が 100 以上であるので、その次に指定された文字列Large を返す。そうでなければ NULL を値とする。NULL はシステムが用意している特別な値であり、定義されている値がないことを示すのに用いられる。次の例 if ... else ... では X が 100 以上の時には上と同様であるが、X が 100 未満のときには Small を返す。

```
> X <- 200                    # X を 200 とする
> if( X >=100) "Large"        # if を実行
[1] "Large"                   # 条件が満たされるので、Large を返す
> X <- 50
> if( X >=100) "Large"
NULL                          # 条件が満たされない場合には、NULL
>
> if( X>=100) "Large" else "Small"

[1] "Small"
```

条件判断に基づいて実行される部分には、複数の関数の呼出しを指定することができる。この場合にはその部分を中括弧 {、} で括って指定する。各々の関数の呼出しはセミコロン; で区切る。ただし、これらの場合の戻り値は、実行される最後の部分で指定される値になる。つぎの例中の関数 cat は、指定された内容を表示のための文字列に変換し空白を接続部分に挿入した上で表示する。また記号\n は改行を指示する。

```
> X <- 200
> if(X>100) {cat(X," is large.\n"); X-100;} else {cat(X," is small.\n"); X;}
200  is large.    # cat で指定された内容の表示。改行も行なわれる。
[1] 100           # X-100 が戻り値になる。
```

また R には次のような論理値をとるベクトルに対応した関数が用意されている。

```
> a0 <- ifelse(c(T,T,F,F,T,T),"x","y")
> a0
[1] "x" "x" "y" "y" "x" "x"
> ifelse(c(T,T,F,F,T,T),c(1,2,3,4,5,6),c(10,20,30,40,50,60))
[1]  1  2 30 40  5  6
```

関数 ifelse の最初の引数は論理値をとるベクトルであり。$i$ 番目の要素が真 (T) ならば 2 番目の引数を $i$ 番目要素の値として返し、偽 (F) ならば 3 番目の引数の値を $i$ 番目の要素とする。

## 4.6 繰り返し

繰り返しを行なうための機能として他のプログラミング言語と類似の機能を持つ for、while などが用意されている。

### 4.6.1 for

繰り返しに伴って添字 (必ずしも数値でなくともよい) を変更するには、for を用いる。使用の書式は次のようなものである。

for(変数名 1 in 表現 1) 表現 2

ここで、表現 1 は数列を表すベクトルや文字列を要素とするベクトルであり、それらの要素の値がつぎつぎと変数名 1 で指定される変数に代入される。各回の代入が行なわれる毎に表現 2 が実行される。変数名 1 で示されるものは (下の例題における i や City) は、for の繰り返しの中だけで定義され、繰り返しの終了以後は変数としては残らない (つぎの例では cat のデフォールト（暗黙の標準値）の指定により、要素を繋ぐ際に空白が挿入される)。例の中で用いている nchar は文字列の長さを求める関数である。また rep は第 1 引数で指定されたデータを、第 2 引数で指定された回数繰り返してベクトルを生成する。

```
> for(i in 1:10) cat(i,",")      # 1:10 は 1 から 10 までの数列
1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,10 >  # 最後に改行がないので、>が続いている。
>
for(i in 10:1) cat(i," ")        # 今度は 10 から 1 までの数列
10  9  8  7  6  5  4  3  2  1  >
>
> cities <- c("函館","札幌","旭川","帯広")
> for(City in cities){cat(City); cat("は北海道にある.\n");}
函館は北海道にある.
札幌は北海道にある.
旭川は北海道にある.
帯広は北海道にある.
```

#### 4.6.2 while

ある特定の条件が成立するまで繰り返しを行なうには while を用いる。指定方法は

while(条件 1) 表現 1

とする。条件 1 が成立する間は表現 1 を実行する。条件 1 が成立しなくなったら (偽となったら) 次の処理に移る。

下の例は、最初に h の値を 2 と設定し、その 2 乗を h につぎつぎと代入している。h の値が 1000 を超えたら、繰り返しを中止する。

```
> h <-2
> while(h<=1000) {cat(h," "); h<- h*h;}
2  4  16  256  >
>
> h               # h には while 終了時の値が残っている。
[1] 65536
```

### 4.7 数値演算上の注意

1. 整数演算では桁あふれが生じる可能性がある。特に大きな数の掛け算を繰り返すと、表現可能な範囲を超えることがある。ただし、R では通常は浮動小数点演算によって計算されるので、特別な場合（C、FORTRAN など他言語の呼び出しの利用など）を除いて、あまり心配する必要はない。

```
> as.integer(2000000000)
[1] 2000000000
> as.integer(2000000000)+as.integer(2000000000)
[1] NA
Warning message:
In as.integer(2e+09) + as.integer(2e+09) :
   整数の桁あふれにより NA が生成されました
```

2. 浮動小数点による小数の表現には、一般的には誤差が生じる。高精度計算を必要とするのでなければ、実用上それほど気にする必要はないが、等号による判断を計算結果について行う場合には、注意を要する。options(digits=17) などのように指定することにより、画面表示の桁数を変更できる。

```
> options(digits=17)
> 1/3
[1] 0.33333333333333333
> options(digits=7)
> 1/3
[1] 0.3333333
```

3. 浮動小数点の精度は絶対値について相対的である.

```
> 1 < 1.000000000000001
[1] TRUE
> 1 < 1.0000000000000001
[1] FALSE
> 10000000000 < 10000000000.000001
[1] TRUE
> 10000000000 < 10000000000.0000001
[1] FALSE
```

## 4.8 欠測値の扱い

R で欠測値は NA として記述する。通常は、値が欠測値であるか否かを判断するには is.na 関数を用いる。

また、数値演算の結果、不定な値 (0/0 など) が代入されると、NaN という特別な記号で表現される値が代入されている。これを比較で判断する場合には、'NaN' のように引用符で囲む必要がある。is.na 関数を用いると、どちらの場合も (内容が NA であっても NaN であっても) 真となる。

R でベクトルから欠測値を除去するには、次のようにする。

```
>  vec1 <- c(1,2,3,NA,5)    # vec1 は欠測値を含む。
>  vec1[!is.na(vec1)]       # 欠測値以外の部分を選択する。
[1] 1 2 3 5
```

下は、上に説明したものの例を参考までに示す。

```
> a <- NA        # 欠測値を代入
> a
[1] NA           # 表示は NA
>  a == NA       # 比較の結果が欠測となる
[1] NA
> is.na(a)       # is.na は機能する。
[1] TRUE
> b <- 0/0       # 非数値になる結果を代入
> b
[1] NaN           # 表示は NaN
> b == NaN       # これでも等式が真にならない。
[1] NA
> b == 'NaN'     # これで真。
[1] TRUE
> is.na(b)       # is.na は真となる
[1] TRUE
```

# 第 II 部
# 線形計算

## 5　ベクトルと行列の操作

### 5.1　行列の意味

行列 (matrix) は行 (row) と列 (column) の 2 次元的な構造をもつデータである。単純な見方をすれば、$n$ 行 $m$ 列の行列は $n \times m$ 個の数の集まりでしかないが、行列の計算が想定している意味はもう少し複雑である。

$m$ 次元のベクトルを定義域とし、$n$ 次元のベクトルを値域とする関数 $f$ を考えよう。$m$ 個の数 $\boldsymbol{x} = (x_1, ..., x_m)^T$ を一つ決めると、$n$ 個の数 $\boldsymbol{y} = (y_1, ..., y_n)^T$ が $f$ によって一つ定まる。ここで、記号 $T$ は横ベクトルを縦ベクトルに変換すること (転置,transpose) を意味する。このとき $f$ がつぎのような性質を持っているものとする。

1. $f(a\boldsymbol{x}) = af(\boldsymbol{x})$。ここで $a$ は実定数である。写像 $f$ が作用する前に $a$ 倍するのと、作用したあとで $a$ 倍したものが同じことを意味する。

2. $f(\boldsymbol{x}_1 + \boldsymbol{x}_2) = f(\boldsymbol{x}_1) + f(\boldsymbol{x}_2)$。 これは、写像 $f$ が作用する前にベクトルの和を求めるのと、各々のベクトルに写像を作用させた後に和を求めたものが同一であることを意味する。

これら 2 つの条件を満たすものは、線形写像と呼ばれる。もし、$f$ が 1 次元のベクトルから 1 次元のベクトルへの写像なら、これらの条件を満たす写像は定数倍に限られる。また、より一般の場合には、$i = 1, ..., n$ について

$$y_i = a_{i1}x_1 + \cdots + a_{im}x_m \tag{6}$$

という式で表されることが分かる。逆に、ここで現れる $n \times m$ 個の係数 $\{a_{ij}\}$ を 1 セット決めると、線形写像が一つ定まることになる。

行列の本質的な意味はこのような線形写像であり、その要素は上にあらわれる係数としての意味を持っている。また、行列積はその行列によって表される写像を合成することを意味する。

ある行列が与えられたとき、それを特徴づける様々な値があるが、その一つにランク (rank) がある。ランクは階数とも呼ばれる。$n$ 行 $m$ 列の行列 $\boldsymbol{A} = (a_{ij})$ について、その $m$ 個の列ベクトルを $\boldsymbol{a}_1, ..., \boldsymbol{a}_m$ と表記することにする。ある $m$ 次元ベクトル $\boldsymbol{x}$ に $\boldsymbol{A}$ を作用させると、

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} = x_1\boldsymbol{a}_1 + \cdots x_m\boldsymbol{a}_m$$

となる。$\boldsymbol{x}$ の値が様々に変化すると $\boldsymbol{y}$ の値もそれにつれて変わる。しかし、もし $\boldsymbol{A}$ が特別な値である場合には $\boldsymbol{y}$ の取り得る値は、ある特定の範囲に限定されたものであるかも知れない。極端な場合、$\boldsymbol{A}$ の要素が全てゼロであれば $\boldsymbol{y}$ はゼロベクトル以外にはなり得ない。このようにして作られる $\boldsymbol{y}$ の全体を線形写像 $\boldsymbol{A}$ の像 (イメージ) と呼び、 $\mathrm{Im}\boldsymbol{A}$ と表記する。$\boldsymbol{A}$ のランク ($\mathrm{rank}\boldsymbol{A}$) は、$\mathrm{Im}\boldsymbol{A}$ の次元 (1 次独立なベクトルを最大何個とれるか) によって定義される。これはまた、$\boldsymbol{a}_1, ..., \boldsymbol{a}_m$ の中から選び出される 1 次独立なベクトルの最大の個数に等しい。いささか込み入った証明が必要であるが、$\mathrm{rank}\boldsymbol{A} = \mathrm{rank}\boldsymbol{A}^T$ であることが示せる。また、$\mathrm{rank}\boldsymbol{A} = \mathrm{rank}\boldsymbol{A}\boldsymbol{A}^T = \mathrm{rank}\boldsymbol{A}^T\boldsymbol{A}$ も成立する。

## 5.2 行列の作成

R では、1 次元的なベクトルから行列を作る方法が幾つか提供されている。関数matrix は、ベクトルを指定された大きさに区切ることにより、行列を作成する。最初の引数はベクトルであり、2 番目の引数は行の数、3 番目の列の数を意味する。関数rbind とcbind は、ベクトルを束ねることにより行列を作成する。rbind はベクトルを行とみなして結合を行なう。また、cbind は列方向の結合を行なう。これら 2 つの関数の引数は、何個であってもよい。また、引数にはベクトルだけでなく行列も許される。関数 dim は行列の次元（大きさ）を長さ 2 の整数ベクトルとして返す。

```
> mat1 <- matrix(c(1,2,3,4,5,6),2,3)    # 2 行 3 列の行列をつくる。
> mat1                                  # 行列の表示。
     [,1] [,2] [,3]                     # 列ベクトルに分割される。
[1,]    1    3    5
[2,]    2    4    6
> mat2 <- rbind(c(1,2,3,4),c(5,6,7,8))  # 行方向に結合
> mat2
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
> mat3 <- cbind(c(1,2,3),c(4,5,6),c(7,8,9))  # 列方向に結合
> mat3
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> dim(mat1)
[1] 2 3
> dim(mat2)
[1] 2 4
> dim(mat3)
[1] 3 3
```

新規に行列を作成するだけでなく、既存のベクトルや行列を操作することによって、新たな行列を作成できる。

```
> ab1 <-  c(1,2,3,4,5)
> ab2 <-  rev(10+ab1)      # 10 を加算して逆転
> mat4 <- cbind(ab1,ab2)
> mat4
     ab1 ab2                 # ベクトルの名前が見出しになる。
[1,]   1  15
[2,]   2  14
[3,]   3  13
[4,]   4  12
[5,]   5  11
> cbind(mat4,c(5,4,3,2,1))   # 行列に 1 列を加える。
     ab1 ab2
[1,]   1  15 5
[2,]   2  14 4
[3,]   3  13 3
[4,]   4  12 2
[5,]   5  11 1
```

上の操作とは逆に、行列をベクトルに戻すには関数c を用いる。これを用いると、列ベクトルを次々とつなげたベクトルが得られる。

```
> c(mat4)
 [1]  1  2  3  4  5 15 14 13 12 11
```

## 5.3 要素の参照

添字を指定することにより、ベクトルや行列の要素や一部分を n 参照することができる。添字が 1 個の整数である場合には、1 個の要素が取り出される。また、添字が整数値のベクトルである場合には、部分ベクトルまたは部分行列が得られる。添字はカギ括弧 ( [] ) でくくって指定する。

```
> ab2[1]              # ベクトル ab2 の第 1 番目の要素を得る。
[1] 15
> ab2[1:3]            # 第 1〜3 番目の要素を取り出す。
[1] 15 14 13
> mat1[1,1]           # mat1 の (1,1) 要素。
[1] 1
> mat1[1,]            # mat1 の 1 行目。
[1] 1 3 5
> mat1[,1]            # mat1 の 1 列目。
[1] 1 2
> mat1[,2:3]          # mat1 の 2 列目と 3 列目。
     [,1] [,2]
[1,]    3    5
[2,]    4    6
```

要素を参照するもう一つの方法は、添字に論理値を持つベクトルを指定することである。真値 (T) の位置に対応した部分のみが取り出される。

```
> ab1                 # ab1 の値を表示する。
[1] 1 2 3 4 5
> ab1[c(F,F,F,T,T)]   # 4 番目と 5 番目の要素が真値のベクトルを指定。
[1] 4 5
> ab1 > 2             # 比較判断により第 3〜5 番目が真値を持つ。
[1] FALSE FALSE  TRUE  TRUE  TRUE
> ab1[ab1>2]          # 添字部分に上のベクトルを指定するのと同 nn じ。
[1] 3 4 5             # 2 より大きな要素のみが得られる。
```

## 5.4 要素の変更

上に示した部分ベクトル・部分行列の参照を代入命令の左辺に置くと、指定された部分の値のみが変更される。

```
> mat1[1,1] <- 999     # (1,1) 要素を 999 に変更。
> mat1
     [,1] [,2] [,3]
[1,]  999    3    5
[2,]    2    4    6
> mat1[2,] <- c(22,44,66)  # 2 行目を変更する。
> mat1
     [,1] [,2] [,3]
[1,]  999    3    5
[2,]   22   44   66
> mat1[,3] <- c(555,666)   # 3 列目の変更。
> mat1
     [,1] [,2] [,3]
[1,]  999    3  555
[2,]   22   44  666
```

## 5.5 行列の計算

ベクトルの計算と同様に、行列の場合も要素毎の算術演算を行なえる。$(m, n)$ の大きさの行列は、長さ $m \times n$ のベクトルとみなされて計算される。

```
> mat3 <- matrix(seq(1:9),3,3)
> 10 * mat3               # 行列の定数倍
     [,1] [,2] [,3]
[1,]   10   40   70
[2,]   20   50   80
[3,]   30   60   90
> mat6 <- matrix(c(10,20,30,40,50,60,70,80,90),3,3)
> mat6
     [,1] [,2] [,3]
[1,]   10   40   70
[2,]   20   50   80
[3,]   30   60   90
```

```
> mat3 + mat6                # 要素毎の加算を行なう。
     [,1] [,2] [,3]
[1,]   11   44   77
[2,]   22   55   88
[3,]   33   66   99
> mat3 * mat6              # 要素毎の乗算 (行列積ではない)。
     [,1] [,2] [,3]
[1,]   10  160  490
[2,]   40  250  640
[3,]   90  360  810
> log(mat6)                  # 要素毎に自然対数を求める。
         [,1]     [,2]     [,3]
[1,] 2.302585 3.688879 4.248495
[2,] 2.995732 3.912023 4.382027
[3,] 3.401197 4.094345 4.499810
```

行列 $\boldsymbol{A} = (a_{ij})$ と $\boldsymbol{B} = (b_{ij})$ の行列積 $\boldsymbol{C} = (c_{ij})$ は

$$c_{ij} = \sum_k a_{ik} b_{kj}$$

で定義される。 R で行列積を計算するには `%*%` という演算子を指定すればよい。また、行列の転置 (行と列の入れ換え) は関数`t` で行なえる。関数`diag` は引数が行列の場合には、その対角要素を取り出してベクトルにする。もし引数がベクトルであれば、その要素を対角要素とする対角行列 (対角要素のみが非零の行列) を作成する。正方の対角行列で対角要素が全て 1 であるものは、単位行列とよばれる。単位行列と他の行列 $\boldsymbol{A}$ の行列積は $\boldsymbol{A}$ である。

**注意:** 後述の関数 `diag` で引数に長さ 1 のベクトルを指定すると、変則的にその数値の大きさの単位行列を表す (例えば `diag(5)` は $5 \times 5$ の単位行列)。

```
> mat3 %*% mat6              # 行列積を求める。
     [,1] [,2] [,3]
[1,]  300  660 1020
[2,]  360  810 1260
[3,]  420  960 1500
> t(mat6)                    # 行列の転置
     [,1] [,2] [,3]
[1,]   10   20   30
[2,]   40   50   60
[3,]   70   80   90
> diag(mat6)                 # 対角要素を取り出す。
[1] 10 50 90
> diag(c(77,88,99))          # 対角行列をつくる。
     [,1] [,2] [,3]
[1,]   77    0    0
[2,]    0   88    0
[3,]    0    0   99
```

ベクトルは本来は縦ベクトルとして解釈されるが、行列積%*%の引数ある場合には、計算がうまく定義されるように、縦ベクトルあるいは横ベクトルとして解釈される。２つの引数がいずれもベクトルの場合には、内積が計算される。

```
> c(1,2,3)
[1] 1 2 3

> t(c(1,2,3))
     [,1] [,2] [,3]
[1,]    1    2    3

> t(t(c(1,2,3)))
     [,1]
[1,]    1
[2,]    2
[3,]    3

> c(1,2,3) %*% c(1,2,3)
     [,1]
[1,]   14

> t(c(1,2,3)) %*% c(1,2,3)
     [,1]
[1,]   14

> c(1,2,3) %*% t(c(1,2,3))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    6
[3,]    3    6    9
```

```
> mat6 %*% c(10,100,1000)
      [,1]
[1,] 74100
[2,] 85200
[3,] 96300
> c(10,100,1000) %*% mat6
      [,1]  [,2]  [,3]
[1,] 32100 65400 98700
```

**課題**

1. 3 つの $3 \times 3$ 行列 $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$ を適当に設定せよ。
2. 2つの行列の積の転置は、各々の行列の転置の順番を逆にした行列積であることを確認せよ。( $(\boldsymbol{UV})^T = \boldsymbol{V}^T\boldsymbol{U}^T$ を確かめる。)
3. 行列積は計算の順序によらず一定であること、つまり $(\boldsymbol{UV})\boldsymbol{W} = \boldsymbol{U}(\boldsymbol{VW})$ であることを確認せよ。

## 5.6 行和・列和

行列の各行ごとの和、平均、積などを求めたり、また列ごとにこれらの値を計算するには、関数 apply を利用することができる。利用法は

apply(行列 1，次元 1，関数 1)

のように指定する。ここで、行列 1 は計算の対象となるものであり、次元 1 は整数値であり計算を適用する次元の指定である。次元 1 が 1 ならば各行に関数 1 を適用し、2 ならば列ごとに関数を適用する。

```
> a <- matrix(c(1,2,3,4),2,2)
> a
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> apply(a,1,sum)   # 各行ごとの和を求める
[1] 4 6            # 第 1 要素は 1+3，第 2 要素は 2+4
> apply(a,2,sum)   # 各列ごとの和を求める
[1] 3 7            # 1+2，3+4
> apply(a,1,prod)  # 各行ごとの積
[1] 3 8            # 1*3 , 2*4
> apply(a,2,prod)  # 各列ごとの積
[1]  2 12          # 1*2，3*4
```

apply と類似の機能を持つ関数として lapply（リスト要素への関数の適用)、sapply（lapply と類似の機能を持つが、1 次元のベクトルを値として返す)。list は要素からリスト構造を作る関数である。リストは複数の種類の異なる要素を束ねることができる。ベクトルや配列は同一種類の要素を並べたものであるが、リストの要素は同一である必要はなく、要素自体がリストであってもよい。リストの要素は[[ ]] の 2 重カギ括弧で番号を括って指定する。リストをベクトルに変換するには unlist を用いる。

```
> lapply( list(c(1,2),c(3,4,5),c(6,7,8,9)), sum)
[[1]]
[1] 3

[[2]]
[1] 12

[[3]]
[1] 30

> sapply( list(c(1,2),c(3,4,5),c(6,7,8,9)), sum)
[1]  3 12 30
```

また、指定された変数によって添え字の分類を行い、他のある変数の値を指定された分類によりグループに類別し、それぞれの群に総和や平均・分散などの関数を適用する tapply などの関数がある。n

```
> tapply(c(1,2,3,4,5,6,7,8,9),c(1,1,2,2,2,3,3,3,3),sum)
 1  2  3
 3 12 30
> tapply(c(1,2,3,4,5,6,7,8,9),c(1,1,2,2,2,3,3,3,3),max)
1 2 3
2 5 9
> tapply(c(1,2,3,4,5,6,7,8,9),c(1,1,2,2,2,3,3,3,3),mean)
  1   2   3
1.5 4.0 7.5
```

関数 apply は、より高次元の配列 (関数 array によって定義される) にも適用できる。次の例では、長さ 24 のベクトルを 3 つの添え字を持ち、$3 \times 4 \times 2$ の長さの多重配列に変換する。さらに、dimnames(配列) へリスト (list) によって表現された各次元の添え字に対応するラベルを代入することにより、添え字ラベルを付与できる。行列と同様に、添え字は行列ど同様に左から順に動く。これは Fortran の記法に準じている。C や Java の配列では、添え字は右から順に動く。ここで attributes は、データに付随する属性 (主たる値以外の補助的な情報) を示す関数である。

```
> ar1 <- array(seq(24),c(3,4,2))
> ar1
, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24

> attributes(ar1)
$dim
[1] 3 4 2
```

(+は継続行の入力中であることを示す R の表示であり、利用者が入力はしない。)

```
> dimnames(ar1) <-
+  list(c("A","B","C"),c("P","Q","R","S"),c("X","Y"))
> ar1
, , X

  P Q R  S
A 1 4 7 10
B 2 5 8 11
C 3 6 9 12

, , Y

   P  Q  R  S
A 13 16 19 22
B 14 17 20 23
C 15 18 21 24

> attributes(ar1)
$dim
[1] 3 4 2

$dimnames
$dimnames[[1]]
[1] "A" "B" "C"

$dimnames[[2]]
[1] "P" "Q" "R" "S"

$dimnames[[3]]
[1] "X" "Y"
```

関数 apply は array にも適用できる。

```
> apply(ar1,c(1,2),sum)
   P  Q  R  S
A 14 20 26 32
B 16 22 28 34
C 18 24 30 36

> apply(ar1,3,sum)
  X   Y
 78 222
```

# 6　ベクトルの計算をグラフで確認

以下では、ベクトルと行列の計算の意味を直観的に把握するために、ベクトルをグラフィック表示しながら計算を行なう。

## 6.1　平行 4 辺形を描く

2 つの 2 次元ベクトルv1,v2 とそれらの和をあらわすベクトルv3 によって作られる平行 4 辺形を描くことを試みる。

下の例で、X11() は X Window System において描画用のウィンドウを起動する命令である。Linux や Unix では描画を行う前に、描画用のデバイスを起動しておく必要がある。

既に描画用のウィンドウが画面上にある場合には、指定する必要はない。Microsoft の Windows の GUI 版では宣言を行わずとも、描画を行える。

関数plot はデータの散布図を表示する関数であり、第 1 引数のベクトルが横軸の座標、第 2 引数が縦軸の座標を与える。arrows(p1,p2,q1,q2) は、座標(p1,p2) から座標(q1,q2) に向かう矢印を描く。関数 text(x,y,z) は、各 $i$ について (x[$i$],y[$i$]) の座標で示される位置に、 文字列 z[$i$] を重ねて表示する。ただし、plot を実行せずに、この命令だけを用いても描画されない。図 1 に描画例を示す。また、複数の文字列や数値をつなげて新たな長い文字列を作るには paste を用いる。

```
> v0 <- c(0,0)                # 原点と 2 つのベクトル (座標) を定義する。
> v1 <- c(1,2)
> v2 <- c(2,0.5)
> v3 <- v1 + v2               # ベクトル和を求める。
> data1 <- cbind(v0,v1,v2,v3)  #  各列が点の座標となる行列を定義。
> data1
     v0 v1  v2  v3
[1,]  0  1 2.0 3.0
[2,]  0  2 0.5 2.5
> X11()                       # グラフィック画面の準備をする。
                              # Windows では不要。
> plot(data1[1,],data1[2,],xlim=c(-5,5),ylim=c(-5,5))
                              # 4 つの点をプロットする。
                              # xlim,ylim は描画領域の指定。
> arrows(0,0,v1[1],v1[2])     # 原点から v1 への矢印を描く。
> arrows(0,0,v2[1],v2[2])
> arrows(v1[1],v1[2],v3[1],v3[2])
> arrows(v2[1],v2[2],v3[1],v3[2])
> text(data1[1,],data1[2,]+0.5,c("0","v1","v2","v1+v2"))
```

## 6.2　関数の定義と編集

込み入った操作を何度も繰り返してタイプするのは面倒である。R には、幾つかの操作をひとまとめにする機能 (関数定義) がある。つぎのように入力すると、$2 \times 2$ の行列を指定して、その各列を 2 次元ベクトル

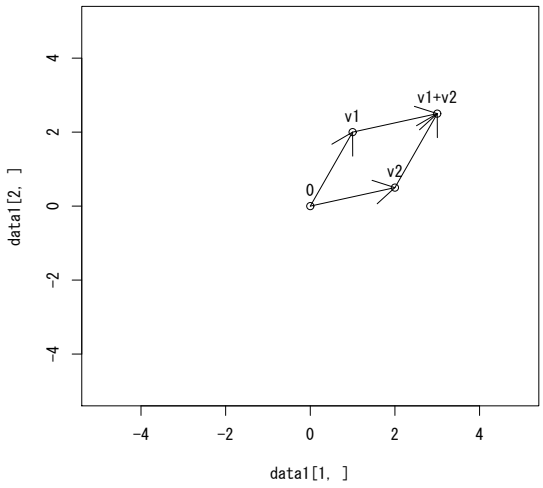


図 1: ベクトル和による平行四辺形の描画

とみなし、平行 4 辺形を描く関数を定義することができる。関数の中では、命令と命令の間は;(セミコロン) で区切る。行の先頭の+は指定が完結していないことを示すために R が自動的に表示する記号である。

```
> para1 <- function(mat1) {
+ v1 <- mat1[,1]; v2 <- mat1[,2]; v0 <- c(0,0); v3 <- v1+v2;
+ matwk <- cbind(v0,v1,v2,v3);
+ plot(matwk[1,],matwk[2,],xlim=c(-5,5),ylim=c(-5,5));
+ arrows(0,0,v1[1],v1[2]); arrows(0,0,v2[1],v2[2]);
+ arrows(v1[1],v1[2],v3[1],v3[2]); arrows(v2[1],v2[2],v3[1],v3[2]);
+ text(matwk[1,],matwk[2,]+0.5,c("0","v1","v2","v1+v2"),cex=5)
+ }
```

ここで命令text の中のcex は文字の大きさの指定である。また、オブジェクト名mat1,v1,v2,v0,v3,matwk などは、この関数の実行中に限って一時的に生成されるものであり、もし同名のオブジェクトが作成済みであったとしても、それらには影響を及ぼさない。

para1 と入力すると、指定した内容が表示される。ただし、このとき区切りのセミコロンは表示されない。どこか間違えたなら、もう一度最初から入力しなおすか、つぎに示す方法で内容を編集する。

Windows 上で notepad(メモ帳) を利用する場合には, 次のように指定する。

```
para1 <- edit(para1,editor="notepad")
```

Linux/Unix 上での利用の場合、vi エディタの使い方を知っているならば、次の命令によって、関数の内容を編集することができる。vi の説明は help(vi) と入力すればみることができる。この命令はオブジェクトpara1 の内容をvi エディタを用いて編集し、その結果を新たなpara1 の内容として代入することを意味する。

```
para1 <- vi(para1)
```

エディタとして vi の代わりに Emacs を使うためには、次のように指定する。

```
para1 <- emacs(para1)
```

Emacs は大きなプログラムであり、起動に時間がかかるが高機能で使いやすい。立ち上がってから、Ctrl+H T (T は大文字) と押すと、最下行でLangage: と尋ねてくるので、ここでJapanese と答えると、日本語の説明が現れる。

プログラムを編集するもう一つの方法は、あらかじめテキストファイルに Emacs などで R のプログラムを作成しておき、それを読み込むことである。プログラムの記述されているテキストファイル名をRprog1.R とし、これが R が作業中のディレクトリにあるとすると、R でsource("Rprog1.R") と実行することにより、ファイルに記述されたプログラムが R に読み込まれ実行される。記述される内容は、処理の手順やデータの定義であってもよいし、関数の定義であってもよい。

## 6.3 回転をあらわす行列の作成

つぎのようにして、平面上の 30 度の回転をあらわす行列をつくる。

```
> pi                          # 円周率 pi はあらかじめ準備されている。
[1] 3.141593
>  deg30 <- 2*pi * 30/360     # 30 度をラディアンであらわす。
> deg30
[1] 0.5235988
> cos(deg30)
[1] 0.8660254
> sin(deg30)                  # sin(30 °) = 0.5 を確認。
[1] 0.5
> sqrt(3/4)                   # cos(30 °) = √(3/4) を確認。
[1] 0.8660254
> data2 <- cbind(c(cos(deg30),sin(deg30)),c(-sin(deg30),cos(deg30)))
> data2                       # 30 °の回転を表す行列。
          [,1]       [,2]
[1,] 0.8660254 -0.5000000
[2,] 0.5000000  0.8660254
```

ここで定義されたdata2 は、30 °の反時計回りの回転をあらわす行列である。上で作成した関数para1 を利用すると、この行列によって、ベクトルがどのように、写像されるかがわかる。これを実行するには、para1(data2) と入力すればよい。

上と同様にして、60 °の回転をあらわす行列を作成することができる。

```
> deg60 <- 2*pi*60/360
> data3 <- cbind(c(cos(deg60),sin(deg60)),c(-sin(deg60),cos(deg60)))
> data3
          [,1]       [,2]
[1,] 0.5000000 -0.8660254
[2,] 0.8660254  0.5000000
> para1(data3)
```

次の行列data4 は、左右の反転をあらわす。

```
> data4 <- diag(c(-1,1))
> data4
     [,1] [,2]
[1,]   -1    0
[2,]    0    1
> para1(data4)
```

行列積は線形写像の合成を意味する。次のように回転をあらわす行列の積を求めると、回転の量が足し合わされるのが分かる。

```
> para1( data3 %*% data2 )
> para1( data2 %*% data3 %*% data2 )
```

回転をあらわす行列の場合には、行列の順番が違っても結果は変わらないが、一般的にはこれは成立しない。data2 %*% data4 とdata4 %*% data2 の値を比較すると、異なっているのが分かる。

**課題**

回転の角度を「度」で指定すると、その回転をあらわす $2 \times 2$ の行列を与える関数を作成しなさい。上のグラフ作成の例では、関数の値を利用してはいないが、R では関数定義の中の最後の式の値が、その関数自身の値となる。また、関数の中で作成されるベクトルや行列は、その関数の中でだけ有効なものであり、関数の実行後には残らない。また、それらのベクトルや行列は、既に作成済のものと名前が重複しても、別のものとみなされる。

例えば、次の関数test1 では、関数が返す値は4*a であり、それ以前に計算されるa や2*a は、関数が返す値には直接には影響しない。複数の値を返す場合には list を用いる。

```
> test1 <- function(a) { a; 2*a; 4*a }
> test1(3)
[1] 12       # 4*a = 4*3 を返す。

> test2 <- function(a) {list(a,2*a,4*4)}
> test2(3)
[[1]]
[1] 3

[[2]]
[1] 6

[[3]]
[1] 16
```

# 7　ベクトルの直交化

1 次独立な幾つかのベクトルが与えられたとき、それらの線形結合であってしかも互いに直交するベクトルをつくりだすことができる。その方法についてここで検討する。

## 7.1　ベクトルの長さ (ノルム)・内積・角度

ベクトル $\boldsymbol{x}$, $\boldsymbol{y}$がそれぞれ

$$\boldsymbol{x}^T = (x_1, x_2, ...x_p) \ , \ \boldsymbol{y}^T = (y_1, y_2, ..., y_p) \tag{7}$$

であるとする。$\boldsymbol{x}$と $\boldsymbol{y}$の内積は次式で表される。

$$(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y} = x_1 y_1 + x_2 y_2 + \ldots + x_p y_p \tag{8}$$

R で 変数x とy にそれぞれベクトルの値が代入されているとする。R では `sum(x * y)` とすると、2 つのベクトルの要素毎の積を求めたあと、それらの和を計算するので、内積が求まる。

ベクトル $\boldsymbol{x}$の長さ (ノルム)$\|\boldsymbol{x}\|$ はそれ自身との内積の平方根であるので、
`sqrt(sum(x*x))` とすればよい。

2 つのベクトルのなす角度 $\theta$ は次の公式を使って求めることができる。

$$\cos\theta \ = \ \frac{(\boldsymbol{x}, \boldsymbol{y})}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|} \ \ , \ \ \theta \ = \ \cos^{-1}\left(\frac{(\boldsymbol{x}, \boldsymbol{y})}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}\right) \tag{9}$$

この計算に対応する R の命令は、次のようになる。ここでacos は余弦関数の逆関数である。

```
vecnorm <- function(x) {sqrt(sum(x*x))}

acos( sum(x*y) / (vecnorm(x)* vecnorm(y))  )
```

ラディアンではなく度で表す場合には

```
acos( sum(x*y) / ( vecnorm(x)* vecnorm(y))  ) * 180/pi
```

とすればよい。

**課題**

適当なベクトルを設定し、それらの角度を求めなさい。

### 7.2 直交射影

2 つのベクトル $\boldsymbol{x}_1, \boldsymbol{x}_2$ があるとき、$\boldsymbol{y} = (\boldsymbol{x}_2, \boldsymbol{x}_1)\boldsymbol{x}_1/\|\boldsymbol{x}_1\|^2$ を $\boldsymbol{x}_2$ の $\boldsymbol{x}_1$ で張られる空間への直交射影という。$\boldsymbol{x}_2 - \boldsymbol{y}$ と $\boldsymbol{x}_1$ とは直交する。実際、

$$(\boldsymbol{x}_2 - \boldsymbol{y}, \boldsymbol{x}_1) = (\boldsymbol{x}_2, \boldsymbol{x}_1) - (\boldsymbol{y}, \boldsymbol{x}_1) = (\boldsymbol{x}_2, \boldsymbol{x}_1) - (\boldsymbol{x}_2, \boldsymbol{x}_1)(\boldsymbol{x}_1, \boldsymbol{x}_1)/\|\boldsymbol{x}_1\|^2 = 0 \tag{10}$$

である。

これと同様の考えに基づいて、1 次独立なベクトルから、次々と互いに直交し長さが 1 であるベクトルを求めることができる。その手順は次のようなものである。まず、1 次独立なベクトルの組を $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_p$ とする。

1. $\boldsymbol{z}_1 = \boldsymbol{x}_1/\|\boldsymbol{x}_1\|$ と置くと長さ 1 である。
2. $\boldsymbol{y}_2 = \boldsymbol{x}_2 - (\boldsymbol{x}_2, \boldsymbol{x}_1)\boldsymbol{x}_1/\|\boldsymbol{x}_1\|^2 = \boldsymbol{x}_2 - (\boldsymbol{x}_2, \boldsymbol{z}_1)\boldsymbol{z}_1$ とし、さらに $\boldsymbol{z}_2 = \boldsymbol{y}_2/\|\boldsymbol{y}_2\|$ とする。
3. $\boldsymbol{y}_3 = \boldsymbol{x}_3 - (\boldsymbol{x}_3, \boldsymbol{z}_1)\boldsymbol{z}_1 - (\boldsymbol{x}_3, \boldsymbol{z}_2)\boldsymbol{z}_2$ とする。$\boldsymbol{z}_2$ と同様に $\boldsymbol{z}_3 = \boldsymbol{y}_3/\|\boldsymbol{y}_3\|$ とする。
4. 以下同様に $\boldsymbol{y}_i = \boldsymbol{x}_i - \sum_{j=1}^{i-1}(\boldsymbol{x}_i, \boldsymbol{z}_j)\boldsymbol{z}_j$ とし、$\boldsymbol{z}_i = \boldsymbol{y}_i/\|\boldsymbol{y}_i\|$ とする。

**課題**

$\boldsymbol{x}_1^T = (1, 1, 2), \boldsymbol{x}_2^T = (1, -1, 2), \boldsymbol{x}_3^T = (-1, 2, 1)$ とおき、これらから上の手順に基づき、互いに直交する長さ 1 のベクトルの組を求めなさい。

## 8 行列式

$p \times p$ の正方行列 $\boldsymbol{A}$ について $\det \boldsymbol{A}$ または $|\boldsymbol{A}|$ という記法によって $\boldsymbol{A}$ の行列式をあらわす。$\boldsymbol{A} = (\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_p)$ としよう。$\boldsymbol{A}$ の行列式は $\boldsymbol{A}$ の $p$ 個の列ベクトルによって構成される平行 (超) 多面体の符号付きの体積である。$p = 2$ のときは、2 つのベクトル $\boldsymbol{a}_1, \boldsymbol{a}_2$ の 2 つのベクトルによって形づくられる平行四辺形の面積、すなわち $\boldsymbol{0}$,$\boldsymbol{a}_1$,$\boldsymbol{a}_1 + \boldsymbol{a}_2$, および $\boldsymbol{a}_2$ の 4 点で囲まれた領域の面積である。ただし、ベクトルの位置関係によって符号が変わる。$\boldsymbol{a}_1$ から $\boldsymbol{a}_2$ への方向の変化が逆時計回り (正の角度) のときには正符号であり、時計回り (負の角度) の時には負符号である。但し回転の角度は 180 度以下とする。$p = 3$ の時は 3 つのベクトルによって構成される平行 6 面体 (直方体を歪めたもの) の体積であり、ただしこの場合もベクトルの向きによって符号が変わる。厳密には、行列式は次の性質を満たす正方行列の関数として定義される。行列式の値は関数 `det` で求める。

1. $\det \boldsymbol{I} = 1$ ただし $\boldsymbol{I}$ は単位行列。
2. ある列を $c$ 倍すると、行列式の値も $c$ 倍になる。

3. $\boldsymbol{A}$ の第 $j$ 列 $\boldsymbol{a}_j$ を別のベクトル $\boldsymbol{b}_j$ に置き換えた行列を $B$ とする。また、$\boldsymbol{a}_j + \boldsymbol{b}_j$ に置き換えた行列を $C$ とする。このとき $\det \boldsymbol{C} = \det \boldsymbol{A} + \det \boldsymbol{B}$ となる。

4. 2 つの列を交換すると、行列式は絶対値が同じで符号が逆転する。

以上の性質を満たすものは、実は一通りしかない。上に述べた平行 4 辺形の符号つき面積や平行 6 面体の符号付き体積は、上の性質を満たすことが直観的にわかる。

また、行列式の定義から次の性質が導かれる。

1. 2 つの列が同じであれば行列式は 0 である。

2. 1 つの列が他の列の定数倍であれば行列式は 0 である。

3. ある列を定数倍したベクトルを別の列に加えても、行列式の値は変わらない。

4. $\boldsymbol{A}$ を $p \times p$ の行列とする。次の 4 つの条件は全て同値である。このとき $\boldsymbol{A}$ は正則 (regular) であるという。

   $\det \boldsymbol{A} \neq 0$ であること

   $\boldsymbol{A}$ の列が 1 次独立であること

   $\boldsymbol{A}_{p\times p}$ のランクが $p$ であること

   逆行列 $\boldsymbol{A}^{-1}$ が存在すること

5. $\det(\boldsymbol{AB}) = \det(\boldsymbol{A}) \det(\boldsymbol{B})$

6. $\det \boldsymbol{A}^{-1} = 1/\det \boldsymbol{A}$

7. $\det \boldsymbol{A} = \det \boldsymbol{A}^T$ ( $^T$ は転置を示す。)

8. 上の定義に示した性質は、列を行と読み換えてもすべて成立する。

9. $\boldsymbol{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ のとき、$\det \boldsymbol{A} = ad - bc$

10. $\boldsymbol{A}$ が対角行列または三角行列の場合には、$\det \boldsymbol{A} = a_{11} \times a_{22} \times \cdots \times a_{pp}$ である。

# 9 連立 1 次方程式と逆行列

$\boldsymbol{A}$ を $p \times p$ の正方行列、$\boldsymbol{b}$ と $\boldsymbol{x}$ を長さ $p$ の縦ベクトルとする。$\boldsymbol{b}$ は値がわかっているが、$\boldsymbol{x}$ は未知であるとする。このとき式 $\boldsymbol{Ax} = \boldsymbol{b}$ は、$p$ 元連立 1 次方程式をあらわす。これについて次のようなことがわかっている。

1. $\det \boldsymbol{A} \neq 0$ すなわち $\boldsymbol{A}$ が正則であれば、$\boldsymbol{x}$ の値は、常に一通りに定まる。

2. $\det \boldsymbol{A} = 0$ である場合には、解が存在しない場合 (不能) と、解が複数存在する場合 (不定) とがある。$\boldsymbol{A}$ が正則でないことを特異 (singular) であるという。解が複数存在するのは、$\mathrm{rank}\boldsymbol{A} = \mathrm{rank}(\boldsymbol{A}|\boldsymbol{b}) < p$ のときであり、解が存在しないのは $\mathrm{rank}\boldsymbol{A} < \mathrm{rank}(\boldsymbol{A}|\boldsymbol{b})$ の場合である。ここで $(\boldsymbol{A}|\boldsymbol{b})$ は $\boldsymbol{A}$ の横に縦ベクトル $\boldsymbol{b}$ をならべた $p \times (p+1)$ の行列である。

$\boldsymbol{A}$ が正則であるときには、$\boldsymbol{Ax} = \boldsymbol{0}$ となるベクトル $\boldsymbol{x}$ はゼロベクトルのみである。また、このとき逆行列 $\boldsymbol{A}^{-1}$ が存在する。$\boldsymbol{A}$ の逆行列とは、$\boldsymbol{AX} = \boldsymbol{I}_p$ となる行列 $X$ のことである。ここで $\boldsymbol{I}_p$ は、$p$ 次の単位行列を表す。$X$ の列ベクトルを $\boldsymbol{x}_1, ..., \boldsymbol{x}_p$ とし、また $\boldsymbol{I}_p$ の $j$ 列を $\boldsymbol{e}_j$ とする。方程式 $\boldsymbol{AX} = \boldsymbol{I}_p$ の両辺の $j$ 列をとると、$\boldsymbol{Ax}_j = \boldsymbol{e}_j$ である。$\boldsymbol{A}$ が正則であれば、これらの $p$ 個の連立 1 次方程式は必ず解を持つので、$\boldsymbol{x}_j$ が求まり、$X$ を定めることができる。$\boldsymbol{AX} = \boldsymbol{I}_p$ が成立すれば、つぎの議論から $\boldsymbol{XA} = \boldsymbol{I}_p$ であることも分かる。

$\boldsymbol{XA} = \boldsymbol{Y}$ とおいてみる。$\boldsymbol{AXA} = \boldsymbol{A} = \boldsymbol{AY}$ である。$\boldsymbol{A} = \boldsymbol{AI}_p$ なので、$\boldsymbol{A}(\boldsymbol{I}_p - \boldsymbol{Y})$ はゼロ行列である。$\boldsymbol{A}$ は正則なので $\boldsymbol{I}_p - \boldsymbol{Y}$ の各列がゼロベクトルであり、$\boldsymbol{Y} = \boldsymbol{I}_p$ となることがわかる。

R の関数solve によって連立 1 次方程式を解いたり、逆行列を求めることができる。

```
> A <- matrix(c(1,1,1,-1,0,1,1,0,1),3,3)  # 3 × 3 の行列を定義
> A
     [,1] [,2] [,3]
[1,]    1   -1    1
[2,]    1    0    0
[3,]    1    1    1

> solve(A,c(5,6,7))       # b^T=(5,6,7) とおいて x を求める。
[1] 6 1 0                 # b が横ベクトルであっても OK。
```

```
> a <- matrix(c(1,2,3,4,5,6,-1,1,-1),3,3)
> a
     [,1] [,2] [,3]
[1,]    1    4   -1
[2,]    2    5    1
[3,]    3    6   -1
> solve(a)                              # 引数が一つのみの場合は逆行列
                                        # を与える。
           [,1]       [,2]  [,3]
[1,] -0.9166667 -0.1666667  0.75
[2,]  0.4166667  0.1666667 -0.25
[3,] -0.2500000  0.5000000 -0.25
> a %*% solve(a)
             [,1]         [,2]          [,3]
[1,] 1.000000e+00 5.551115e-17 -2.775558e-17
[2,] 5.551115e-17 1.000000e+00  2.775558e-17
[3,] 2.775558e-16 5.551115e-17  1.000000e+00
> solve(a) %*% a
             [,1]          [,2]          [,3]
[1,] 1.000000e+00  4.996004e-16 -1.665335e-16
[2,] 0.000000e+00  1.000000e+00  5.551115e-17
[3,] 2.775558e-17 -5.551115e-17  1.000000e+00
```

e$xxx$ は、数値が $10^{xxx}$ 倍されることをあらわす。対角要素は 1 であり、非対角要素はほとんどゼロである

ことがわかる。

逆行列には次のような性質がある。

1. $(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$
2. $(\boldsymbol{A}^T)^{-1} = (\boldsymbol{A}^{-1})^T$
3. 上三角行列の逆行列は、上三角行列。
4. 下三角行列の逆行列は、下三角行列。

**課題**

適当な係数行列 (正方)$\boldsymbol{A}$ と右辺のベクトル $\boldsymbol{b}$ を指定し、`solve` を用いて、連立 1 次方程式を解け。また上に示した逆行列の性質を確認しなさい。R では関数`t` で行列の転置を行なえる。

# 10 答えのない連立 1 次方程式

## 10.1 最小 2 乗法 lsfit

前節で、連立 1 次方程式

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{11}$$

において、この式を満足する $\boldsymbol{x}$ が存在するのは、$\mathrm{rank}\boldsymbol{A} = \mathrm{rank}(\boldsymbol{A}|\boldsymbol{b})$ のときであると説明した。しかし、この条件が成立しない場合にも解に「近い」$\boldsymbol{x}$ がどれであるかを決めることはできる。ここで、$\boldsymbol{y} = \boldsymbol{Ax}$ とおく。$\boldsymbol{x}$ の解としての良さを $\|\boldsymbol{b} - \boldsymbol{y}\|^2$ によって定義する。これはベクトルの次元を $p$ とするとき、

$$\sum_{i=1}^{p}(b_i - y_i)^2 \tag{12}$$

となる。$\boldsymbol{x}$ の値を調節して、これを小さくするような $\boldsymbol{y}$ をつくり出すことについて考える。

ここで、$\boldsymbol{A}$ の次元は $n \times p(n \geq p)$ であり、$\boldsymbol{A}$ のランクは $p$ であるとする。つまり、$\boldsymbol{A}$ の各列は 1 次独立なベクトルである。簡単のために $n = 3, p = 2$ の場合を考えてみよう。$\boldsymbol{A} = (\boldsymbol{a}_1, \boldsymbol{a}_2)$ とすると $\boldsymbol{a}_1, \boldsymbol{a}_2$ の各々は次元 3 のベクトルであり、3 次元空間の中の同一直線上にはない。また、$x_1\boldsymbol{a}_1 + x_2\boldsymbol{a}_2$ の全体 $(x_1, x_2$ が様々に変化する場合のベクトル全体) は、原点を通る 1 つの平面をなす。

より具体的に $\boldsymbol{a}_1 = (1, 1, 1)^T$ , $\boldsymbol{a}_2 = (0, 1, 2)^T$ としよう。この場合 $\boldsymbol{b} = (0, 2, 3)^T$ とおくと、$\boldsymbol{Ax} = \boldsymbol{b}$ を満たす $\boldsymbol{x}$ は存在しない。多変数の微分を用いた計算から、実は

$$\hat{\boldsymbol{x}} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b} \tag{13}$$

が式 (12) を最小にすることがわかっている。これを使うと $\hat{\boldsymbol{y}} = \boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}$ である。計算すると $\boldsymbol{A}^T\boldsymbol{A} = \begin{pmatrix} 3 & 3 \\ 3 & 5 \end{pmatrix}$ であり、$(\boldsymbol{A}^T\boldsymbol{A})^{-1} = \begin{pmatrix} 5/6 & -1/2 \\ -1/2 & 1/2 \end{pmatrix}$ である。これらを使うと $x_1 = 1/6, x_2 = 3/2$ のとき $\hat{\boldsymbol{y}} = (1/6, 10/6, 19/6)^T$ であり、これが $\boldsymbol{b}$ に一番「近い」値であることがわかる。

**課題**

実際に $\boldsymbol{x}$ の値を様々に設定し、その場合の (12) の値を比較してみなさい。

これらの手順をひとまとめにした関数`lsfit` が R には用意されている。行列`a` とベクトル`b` が上の例の様に定義されているものとする。

```
> a <- matrix(c(1,1,1,0,1,2),3,2)
> a
     [,1] [,2]
[1,]    1    0
[2,]    1    1
[3,]    1    2
> b <- c(0,2,3)
```

次のように、関数 lsfit を使うと、$\boldsymbol{x}$ の推定を行なえる。

```
> ls1 <- lsfit(a,b,intercept=F)
> ls1
$coefficients
       X1        X2
0.1666667 1.5000000         # X の推定値

$residuals
[1] -0.1666667  0.3333333 -0.1666667

$intercept
[1] FALSE

$qr                # 以下は逆行列計算のための情報
$qt
[1] -2.8867513 -2.1213203 -0.4082483

$qr
             X1         X2
[1,] -1.7320508 -1.7320508
[2,]  0.5773503 -1.4142136
[3,]  0.5773503  0.9659258

$qraux
[1] 1.577350 1.258819

$rank
[1] 2

$pivot
[1] 1 2

$tol
[1] 1e-07

attr(,"class")
[1] "qr"
```

ls1 は、内部にいろいろな要素をもつリスト (list) データになっている。$に続く名前が、データの要素をしめしている。このうち、ls1$coefficients が推定された $\hat{\boldsymbol{x}}$ である。ls1$residuals は残差 $\boldsymbol{b}-\hat{\boldsymbol{y}}=\boldsymbol{b}-\boldsymbol{A}\hat{\boldsymbol{x}}$ を示している。次のように指定すると、$\hat{\boldsymbol{y}}=\boldsymbol{A}\hat{\boldsymbol{x}}$ を求めることができる。

関数 ls.print を用いると、最小 2 乗法の要約情報が得られる。

```
> ls.print(ls1)
Residual Standard Error=0.4082
R-Square=0.9872
F-statistic (df=2, 1)=38.5
p-value=0.1132

   Estimate Std.Err t-value Pr(>|t|)
X1   0.1667  0.3727  0.4472   0.7323
X2   1.5000  0.2887  5.1962   0.1210
```

ベクトル $\boldsymbol{b}$ の予測値は次のように計算できる。

```
> a %*% ls1$coefficients
          [,1]
[1,] 0.1666667
[2,] 1.6666667
[3,] 3.1666667
```

このベクトルは残差を $\boldsymbol{b}$ から引くことによっても求められる (ただし、この場合は行ベクトルになる)。

```
> b - ls1$residuals
[1] 0.1666667 1.6666667 3.1666667
```

ls1$intercept は定数ベクトルを係数行列の最初の列として付加するか否かを示すものである。この例ではlsfit の実行時にintercept=F(F は False の意味) を指定しているので、定数ベクトルは新たには付加されていない。つぎの様に指定すると、上の例と実質上同じ計算が行なわれる (T は True の意味)。

```
> a1 <- matrix(a[,2],3,1)
> a1
      [,1]
[1,]    0
[2,]    1
[3,]    2
 > ls2 <- lsfit(a1,b,intercept=T)
```

## 10.2 直交射影子

前項の最小 2 乗法で、$\boldsymbol{P} = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$ という形の行列がでてきた。これは直交射影子と呼ばれるものの例になっている。行列 $\boldsymbol{A}$ が $n \times p$ の大きさであり、列ベクトルが 1 次独立であるとすると、$\boldsymbol{P}$ は $n \times n$ の行列であり、その階数 (rank) は $p$ である。この行列が直交射影子と呼ばれるのは、$\boldsymbol{Px}$ が、$\boldsymbol{A}$ の列ベクトルによって張られる 3 次元空間の中の 2 次元の平面への $\boldsymbol{x}$ の垂直な射影になっているためである。

一般に、次の 2 つの性質を満たす正方行列は直交射影子になっている。

1. 対称である。すなわち $\boldsymbol{P}^T = \boldsymbol{P}$
2. 冪 (べき) 等である。すなわち $\boldsymbol{PP} = \boldsymbol{P}$

$\boldsymbol{P}$ が直交射影子ならば $\mathrm{rank}\boldsymbol{P} = \mathrm{trace}\boldsymbol{P}$ が成立し、この値は射影をとる部分空間の次元である。ここで trace(トレース) とは対角要素の和のことである。前項の最小 2 乗法によって求められる $\hat{\boldsymbol{y}}$ は、$\boldsymbol{A}$ の列ベクトルで張られる空間上への $\boldsymbol{b}$ の直交射影になっている。

**課題**

第 1 項で定義した $\boldsymbol{A}$ から、つぎのようにして直交射影子 $\boldsymbol{P} = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$ を作成せよ。

```
> p1 <- a %*% solve(t(a) %*% a) %*% t(a)
```

このようにして作成した直交射影子が、対称、冪等であり、またそのトレースが階数 (この場合は 2) に等しいことを確認せよ。

# 11 直交行列

$p \times p$ の行列で、各列ベクトルの長さが 1 で、しかも互いに直交している行列のことを直交行列 (orthogonal matrix) という。行列 $U$ が直交行列であるとすると、その定義より $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$ である。ここで、$\boldsymbol{I}$ は単位行列を示す。つまり、これは $U$ の逆行列が $U^T$ であることを示している。

一般に $\boldsymbol{A}^{-1}$ が $\boldsymbol{A}$ の逆行列であるとすると $\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}$ である。従って $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{U}'\boldsymbol{U}^T = \boldsymbol{I}$ がいえる。これより、$U$ の各行も長さ 1 で互いに直交していることがわかる。行列式の性質より、

$$\det\boldsymbol{U}\det\boldsymbol{U} = \det\boldsymbol{U}\det\boldsymbol{U}^T = \det(\boldsymbol{U}\boldsymbol{U}^T) = \det\boldsymbol{I} = 1 \tag{14}$$

である。これより、$\det U = \pm 1$ である。

**課題**

1. 次の行列が直交行列であることを、確認しなさい。

$$(1)\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \quad (2)\begin{pmatrix} -\sin\theta & \cos\theta \\ \cos\theta & \sin\theta \end{pmatrix}$$

$$(3) I - 2\boldsymbol{a}\boldsymbol{a}^T \text{ , ここで}\boldsymbol{a}\text{は長さ 1 のベクトル}$$

$$(4)\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V} \quad \text{ただしここで}\boldsymbol{U}\text{と}\boldsymbol{V}\text{はともに直交行列}$$

2. $\boldsymbol{U}_{p\times p}$ を直交行列とし、$\boldsymbol{x}$ を $p$ 次のベクトルとする。このとき $\boldsymbol{U}\boldsymbol{x}$ の長さは常に $\boldsymbol{x}$ の長さに等しいことを確認しなさい。

# 12 特異値分解

$\boldsymbol{A}$ を $p \times q$ の行列とする (特に制限は付けない)。ここで $r = \min(p, q)$ とする。実は、$\boldsymbol{A}$ のつぎのような分解が必ず存在する。

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$$

ここで $\boldsymbol{U}$ は $p \times r$ の行列であり各列は長さ 1 で互いに直交している。すなわち、$\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}_r$。 また $\boldsymbol{D}$ は $r \times r$ の対角行列で要素は非負の数である。さらに、$\boldsymbol{V}$ は $q \times r$ の行列で、$\boldsymbol{U}$ と同様に各列は長さ 1 で互いに直交している。つまり、$\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}_r$。

このような分解のことを、特異値分解 (singular value decomposition) と呼ぶ。行列 $\boldsymbol{A}$ のランク (階数) は、$\boldsymbol{D}$ の非零の対角成分の個数に等しい。$\boldsymbol{D}$ の対角要素のことを $\boldsymbol{A}$ の特異値と呼ぶ。R には特異値分解を行なう関数svd が用意されている。

```
> mat1 <- matrix(c(1,2,0,1,1,1),2,3)  # 行列を定義
> mat1
     [,1] [,2] [,3]
[1,]    1    0    1
[2,]    2    1    1

> mat1.svd <- svd(mat1)              # 特異値分解の実行し結果を代入
> mat1.svd                           # 特異値分解の中身
$d
[1] 2.7578164 0.6280515            # 対角要素（特異値）

$u                                   # U
           [,1]       [,2]
[1,] -0.4718579 -0.8816746
[2,] -0.8816746  0.4718579

$v                                   # V
           [,1]       [,2]
[1,] -0.8104989  0.0987837
[2,] -0.3197003  0.7513045
[3,] -0.4907986 -0.6525208

> mat1.svd$u %*% diag(mat1.svd$d) %*% t(mat1.svd$v) # 復元する。
     [,1]         [,2] [,3]
[1,]    1 1.519238e-16    1
[2,]    2 1.000000e+00    1
```

特異値分解は最小 2 乗法の計算と密接な関連を持っている。特異値分解を使うと、行列 $\boldsymbol{A}$ をより低いランクの行列で最小 2 乗基準の意味で近似することができる。このため、多くの変数を少数の変数で近似するために多変量解析でしばしば利用される。

**課題**

$\boldsymbol{A}$ の特異値分解の対角行列を $\boldsymbol{D}$ とする。$\boldsymbol{A}^T\boldsymbol{A}$ および $\boldsymbol{A}\boldsymbol{A}^T$ を特異値分解して得られる対角行列は、ともに $\boldsymbol{D}^2$ であることを確認せよ。また、$\boldsymbol{A}$ が逆行列を持つ場合には、その特異値はどのようになるだろうか。検討せよ。

# 13　多変数の 2 次関数

## 13.1　関数の最大・最小と対称行列の固有値

1 変数の 2 次関数は

$$f(x) = ax^2 + bx + c$$

のように書き表される (ただし、$a \neq 0$)。 この関数のグラフは $a$ が正の場合には、上に開いた放物線であり、$a$ が負の場合には、下向きに開いた放物線になる。このような 2 次関数を多変数に拡張してみよう。ここでは、単純のために 2 変数の場合を考える。

まず、2 変数の 2 次関数は一般的につぎのような形式で書きあらわされる。

$$f(x_1, x_2) = a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2 + b_1x_1 + b_2x_2 + c$$

これを行列とベクトルを用いて書き表すとつぎのようになる。

$$f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c$$

ただしここで

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},\ \boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix},\ \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

である。ここで $\boldsymbol{A}$ は対称 ($\boldsymbol{A} = \boldsymbol{A}^T$) であることに注意する。

この関数を $\boldsymbol{x}$ の要素毎に微分してみよう。まず、$x_1$ で微分すると

$$\frac{\partial f}{\partial x_1} = 2(a_{11}x_1 + a_{12}x_2) + b_1$$

である。また、$x_2$ で微分すると、

$$\frac{\partial f}{\partial x_2} = 2(a_{12}x_1 + a_{22}x_2) + b_2$$

である。

$$\begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{pmatrix} = \frac{\partial f}{\partial \boldsymbol{x}}$$

と表記することにすると、

$$\frac{\partial f}{\partial \boldsymbol{x}} = 2\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$

である。1 次微分ベクトル $\frac{\partial f}{\partial \boldsymbol{x}}$ は、関数 $f$ のグラディエントベクトルとも呼ばれる。これは関数を曲面として表示すると、各点における山側の方向 (水の流れと逆方向) を示すものである。もし $2\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} = \boldsymbol{0}$ の解 $\boldsymbol{x}_0$ が存在するならば、そこでは $\frac{\partial f}{\partial \boldsymbol{x}_0} = \boldsymbol{0}$ が成り立つ。これは関数をあらわす曲面の接平面が水平であることをあらわしている。この $\boldsymbol{x}_0$ は $f(\boldsymbol{x})$ にとって特別な意味を持つが、さらに関数の性質を詳しく調べるためには、行列 $\boldsymbol{A}$ の固有値と呼ばれる量について検討しなければならない。

一般に、$p \times p$ の対称行列 $\boldsymbol{A}$ はつぎのように直交行列 $\boldsymbol{U}$ と対角行列 $\Lambda$ の積に分解できる。

$$\boldsymbol{A} = \boldsymbol{U}\Lambda\boldsymbol{U}^T$$

これは、特異値分解によく似ているが、左右の直交行列が同一のものであるところが違う。また $\Lambda$ の対角要素は非負とは限らない。ある。この場合の $\Lambda$ の対角要素のことを、$\boldsymbol{A}$ の固有値 (eigenvalue) と呼び、また $\boldsymbol{U}$ の列ベクトルのことを、固有ベクトルとよぶ。

$\boldsymbol{U}$ の第 $j$ 列を $\boldsymbol{u}_j$ とすると、

$$\boldsymbol{A}\boldsymbol{u}_j = \lambda_j\boldsymbol{u}_j$$

が成立する。ここで $\lambda_j$ は $\Lambda$ の第 $(j,j)$ 要素である。

先の 2 次関数 $f(\boldsymbol{x})$ に戻ると、

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c \\ &= (\boldsymbol{x}-\boldsymbol{x}_0)^T\boldsymbol{A}(\boldsymbol{x}-\boldsymbol{x}_0) - \boldsymbol{x}_0^T\boldsymbol{A}\boldsymbol{x}_0 + c \end{aligned}$$

と変形される。また、$\boldsymbol{A} = \boldsymbol{U}\Lambda\boldsymbol{U}^T$ とし、$\boldsymbol{y} = \boldsymbol{U}^T(\boldsymbol{x}-\boldsymbol{x}_0)$ とすると、

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{y}^T\Lambda\boldsymbol{y} + \text{constant} \\ &= \lambda_1 y_1^2 + \lambda_2 y_2^2 + \text{constant} \end{aligned}$$

となる。

$\lambda_1, \lambda_2$ がともに正の場合は、関数曲面の形はお椀型であり、またともに負の場合は、お椀を伏せた形になる。一方が正で他方が負の場合には、馬の鞍のような形になる。また、片方の固有値が零の場合には、放物線を横に移動した痕跡の形になる。これから、固有値が全て正の場合には、$\boldsymbol{x}_0$ は $f(\boldsymbol{x})$ の最小値であり、また、固有値が全て負の場合には、最大値になることが分かる。

R では関数eigen を用いて、対称行列の固有値と固有ベクトルを求めることができる。

```
> A <- matrix(c(2,1,1,2),2,2)        # 対称行列を定義
> A
     [,1] [,2]
[1,]    2    1
[2,]    1    2
> eigen(A)
$values:                             # 固有値
[1] 3 1

$vectors:                            # 各列が固有ベクトル
          [,1]       [,2]
[1,] 0.7071068  0.7071068
[2,] 0.7071068 -0.7071068

> aeigen <- eigen(A)
> aeigen$vectors %*% diag(aeigen$values) %*% t(aeigen$vectors)
     [,1] [,2]                       #  A の復元
[1,]    2    1
[2,]    1    2
```

## 13.2 一般行列の固有値と固有ベクトル

実数を要素とする $p$ 次の正方行列 (必ずしも対称ではない)$\boldsymbol{A}$ について、$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$ が成立するものとする。ここで、$\lambda$ は実数または複素数であり、$\boldsymbol{x}$ は実数または複素数からなる $\boldsymbol{0}$ ではないベクトルである。この条件を満たす $\lambda$ のことを $\boldsymbol{A}$ の固有値とよび、$\boldsymbol{x}$ を固有ベクトルとするのが本来の定義である。

$\boldsymbol{A}$ が対称行列である場合には、前節に示した行列分解によって固有値と固有ベクトルを求めることが可能であり、$\boldsymbol{A}$ の要素が実数ならば、固有値も固有ベクトルも全て実数である。しかし、$\boldsymbol{A}$ が対称では

ない場合には、$\boldsymbol{A}$ の要素が実数であっても、固有値が実数である保証はない。$\lambda$ が $\boldsymbol{A}$ の固有値であれば、$(\boldsymbol{A}-\lambda\boldsymbol{I}_p)\boldsymbol{x}=\boldsymbol{0}$ であるので、$\boldsymbol{A}-\lambda\boldsymbol{I}_p$ の行列式は零でなければならない。$p$ 次正方行列の行列式は行列要素の $p$ 次式であるので、$\lambda$ はこの $p$ 次式の解になっている。

# 第 III 部
# データ解析

## 14　テキストファイルからの入力

### 14.1　ベクトルの入力

テキストファイルに記述されたデータをベクトルとして入力するには scan を用いる。例えば、sample1.dat というファイルに数値が順に記入されており、これを x という名のベクトルとして読み込む場合には、つぎのように scan 関数の引数にデータを含むファイル名を指定する。

ファイル sample2.dat に記入されているデータが文字型である場合には、scan の第 2 引数に文字型の定数（"a"など）を指定する。ここで、scan の 2 番目の引数"a"は、データの型を例示するものであり、この場合文字型のデータ (他のもの、例えば"B"でも良い) を指定する。

sample1.dat の内容

```
1.2  3.4  5.67
8.90 9.1 10.12
3.456
```

sample2.dat の内容

```
"Hokkaido" "Aomori" "Akita"
"Iwate" "Yamagata" "Miyagi" "Fukushima"
```

```
> x <- scan("sample1.dat")  # sample1.dat よりデータを入力
Read 7 items
> x
[1]  1.200  3.400  5.670  8.900  9.100 10.120  3.456
> y <- scan("sample2.dat","a") # sample2.dat より文字型で入力
Read 7 items
> y
[1] "Hokkaido"  "Aomori"    "Akita"     "Iwate"     "Yamagata"  "Miyagi"
[7] "Fukushima"
```

オプションの指定により、複数の（長さの同じ）ベクトルを同時に入力することもできる。

Microsoft Windows のテキストファイルを、ftp コマンドなどによって転送する場合、バイナリモードを用いると Unix/Linux では余分な復改コード^M がファイルに含まれる。つぎのように nkf[6] という UNIX ユティリティを用いると file1 から余分なコードを除いたものが、file2 となる。

```
nkf -d  file1 > file2
```

[6] 開発サイトは nkf.sourceforge.jp

## 14.2 ファイル名の指定

ファイル名を指定する場合には、Windows では文字列中にディレクトリの区切り記号を 2 重に記入する必要がある。たとえばC:\\Users\\All Users\\work\\sample1.dat のように記述する。日本語 Windows 上ではバックスラッシュが円通貨記号に見える。これはバックスラッシュが、文字列中でエスケープ記号としての役割をもつために必要な措置である。R では Unix 風に、ディレクトリの区切り記号をスラッシュ/で書くことができる。この場合には、2 重に記号を重ねる必要がないので、こちらのほうが使いやすい。

## 14.3 データフレームの作成

複数の変数 (数値型、文字型が混在する場合もある) から構成される表の形式を持つデータを入力するには、read.table を用いる。次のようなデータがファイル seisekia1.dat に保存されているとする。ここで最初の変数が、学生の名前であり、２番目が英語、３番目が数学、４番目が国語の成績であるとする。

seisekia1.dat

```
FUJIO    94  78  99
HAURO    52  49  20
HIROSHI  59  49  20
JUNKO    69  72  53
MIDORI   51  63  59
MINORU   85  90  77
SUMIO    26  71  36
TOHRU    18  35   9
TOYOKO   91  88  79
YAYOI    87  91  66
```

これを読み込むためには、次のように実行する。

```
seisekia1 <- read.table("seisekia1.dat",row.names=NULL,
   col.names=c("name","eng","math","jpn") )
```

この命令を実行すると、seisekia1 という名のオブジェクトにデータが保存される。これは**データフレーム**と呼ばれる型のオブジェクトであり、いくつかのベクトルや行列を要素として含む。この場合には、name、eng、math、jpn の 4 つのベクトルが要素である。

col.names には列 (変数) の名前を指定する。また、引数に header=T を指定すると、データの先頭行を列の名前として入力し、2 行目以降をデータベクトルの要素とする。

row.names=NULL は、行の名前を数字 (行番号) にする指定である。文字型のベクトルを指定すると (重複がなく、各要素が一意的であるもの)、それが行の名前になる。列の番号あるいは列の変数名を指定すると、その列が行の名前になる。また特に指定しないと (NULL 指定の場合)、header=F の場合には行の番号が行の名前になる。header=T の場合には、ヘッダーの要素数がデータの列数より 1 少ないと、最初の列が行名になり、特に指定がなければ行番号になる。

次ようにヘッダつきのデータがファイル seisekia1Head.dat に記述されているとする。

seisekia1Head.dat 一部省略

```
name    eng math jpn
FUJIO    94  78  99
HAURO    52  49  20
...........
YAYOI    87  91  66
```

この場合には次のように header=T を指定して入力する。

```
seisekia1h <- read.table("seisekia1Head.dat",row.names=NULL,
       header=T)
```

関数 read.table で読み込まれた変数のうち、name のように数値でない文字型の変数などは、factor と呼ばれる、個別の識別番号を示す整数と、それらに対応するラベルの組として表現される。この構造を用いず、入力した文字列をそのままデータとして保持したい場合には、as.is オプションを用いて指定する。

入力された各変数を参照するには seisekia1$name などのように、オブジェクト名と要素変数名を$で繋いで指定する。または、seisekia1[[1]] の様に指定することもできる。ここで [[$i$]] はリスト構造の $i$ 番目の要素を指定するものである。データフレームは、ベクトルや行列をリスト構造にまとめて一つのオブジェクトとしている。リスト構造についての詳細は後述する。各変数の個別の要素を参照するには、seisekia1$name[1] などのように [ ] によって要素の番号を指定する。また、行列の場合と同様に seisekia1[,1] とすると 1 番目の変数 name が参照され、また seisekia1[1,] とすると 1 行目のレコードが参照される。R に含まれる多変量を参照する分析方法の多くは、データフレームを引数とするよう設計されている。

既に作成済みのベクトルなどからデータフレームを作成するには、次のように関数 data.frame を用いる。ここで各引数の等号=の左側が要素名の指定、右側が参照するオブジェクト (ベクトルなど) の名前である。この際、各変数の長さは同じでなければならない。

```
nameeng <- data.frame( v1=seisekia1$name, v2=seisekia1$eng )
```

変数の値によってデータフレームの一部分を選択する場合には、次のように指定する。

```
> seisekia1[seisekia1$eng>60,]
     name eng math jpn
 1  FUJIO  94   78  99
 4  JUNKO  69   72  53
 6 MINORU  85   90  77
 9 TOYOKO  91   88  79
10  YAYOI  87   91  66
```

データフレームは、行がレコード (SAS でのオブザベーション、SPSS のケース)、列が変数である行列のような取り扱いが可能である。上の指定では英語の成績が 60 点を超えるレコードを選択している。

# 15 基礎統計機能

R には極めて多くの統計分析機能があるが、ここではおもに基本的な記述統計のための関数と、確率分布および乱数の生成法について紹介する。

## 15.1 1 変数の統計量

次に示すような、1 変数統計量を求める関数が用意されている。関数var によって求められる値は、分散の不偏推定量

$$\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\bar{x})^2$$

である。ここで、$\bar{x}$ は $\{x_i\}$ の標本平均である。分位点を求める関数quantile は、引数が 1 個の場合には、(最小値、第 1 四分位点、中央値、第 3 四分位点、最大値) からなる長さ 5 のベクトルを返す。下の例中の quantile の出力 1 行目は要素の見出しであり、2 行目がベクトルの値である。

```
>  dt1 <- c(163,165,158,170,177,168,172)  # データの代入
> mean(dt1)                               # 平均を求める。
[1] 167.5714
> var(dt1)                                # 分散
[1] 38.95238
> median(dt1)                             # 中位数
[1] 168
> max(dt1)                                # 最大値
[1] 177
> min(dt1)                                # 最小値
[1] 158
> quantile(dt1)                           # 4 分位点と最大・最小を求める。
   0%  25%  50%  75% 100%                 # 見出し
  158  164  168  171  177                 # 分位点の値
> quantile(dt1,0.2)                       # 20 %点を求める。
   20%
 163.4
```

欠測値を除いて計算を行う場合には、上に示した関数では na.rm=T を指定する。

## 15.2 分布の比較

2 群のデータにおける統計量 (平均、分散など) を比較することは、最も基本的な統計的な分析の 1 つである。ここでは、Cleveland(1993) にとりあげられている。原典は Frisby & Clatworthy (1975) によるランダムドットステレオグラムの反応時間の測定実験である。Statlib の Data and story library (DASL) より入手できる。

データの所在はhttp://lib.stat.cmu.edu/DASL/Datafiles/FusionTime.html

### 15.2.1 データフレームの作成

このデータは各行が 2 つの値 (1 列目が反応時間、2 列目が教示の種類) から構成されている。教示の種類は NV または VV のいずれかである。教示 NV はどのような図形が両眼の画像の融合によって生じるかを、言語的にあらかじめ説明するかまたは全く説明なしであることを意味し、VV は言語的な説明と実際に図形による説明の両者を行なったことを意味している。反応時間は、刺激図形 (ランダムドットステレオグラム) が提示されてから、被験者が立体画像を認識するまでの時間である。[7] データの最初の 5 行を次に示す。

```
 47.20001    NV
 21.99998    NV
 20.39999    NV
 19.70001    NV
 17.40000    NV
   ...
```

このデータを入力するには、R このデイレクトリにおいて起動し次のように指示する。

```
> fusion <- read.table("fusiontime.dat", row.name=NULL,
    col.name=c("time","inst"))
```

ここで、fusion という名称のデータフレームが作成される。各々の実験条件に対応するデータの件数を調べるには、table を使う。以後で入力を簡単にするために、各々の実験条件の反応時間データを fusionNV、fusionVV という名前のベクトルに保存する (データフレームのままの方が簡単な場合もあるので、必ずしも必要な訳ではない)。ここで、fusion$inst=="NV"は教示が NV であるレコード番号に対応する要素が TRUE それ以外が FALSE であるベクトルとなる。これを添字指定に用いることにより、NV に対応する反応時間が取り出せる。

```
> table(fusion$inst)
 NV VV
 43 35
>
> fusionNV <- fusion$time[fusion$inst=="NV"]
> fusionVV <- fusion$time[fusion$inst=="VV"]
```

### 15.2.2 枝葉図

分布の特徴を簡単に把握するための関数として stem が用意されている。これは Tukey(1977) において枝葉図 (stem-and-leaf plot) と名付けられているグラフである。 図 2 では、教示が"NV"であるものについて枝葉図を描いている。ここで、出力されているのは簡単なヒストグラムである。反応時間の整数部分が図の左側の見出しに描かれており、範囲に該当するデータの個数分の文字が、その行に並べて表示される。表示されるのは小数点 1 桁目の数値である。ほとんどの反応時間は 10 秒以下であるが、中には 47.2 秒かかるものもあり、正方向に裾が重いことが分かる。教示が"VV"であるものについても同様に、stem(fusionVV)、または直接stem(fusion$time[fusion$inst == "VV"]) とすれば枝葉図が表示される。

[7] M.Friendly による DASL のページにはデータ件数は 81 となっているが、実際に表示されているデータは 78 件であった。

```
> stem(fusionNV)

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

  0 | 2222222223333444
  0 | 5566678888999
  1 | 00002233
  1 | 57
  2 | 002
  2 |
  3 |
  3 |
  4 |
  4 | 7
```

図 2: 枝葉図

### 15.2.3 ヒストグラム

グラフィックウィンドウを用いてヒストグラムを描くには、関数 hist を用いる。この関数を用いる前にまず、X11() または motif() と入力し、描画用のウィンドウを表示する。既に描画用のウィンドウが表示されている場合には、これは不要である。

次に 2 つの分布を比較するため、1 画面に 2 つのグラフが表示されるよう、指定を行なう。関数 par は画面表示の各種のオプションを設定するための関数である。mfrow=c(1,2) は、画面を縦 1 列、横 2 列に分割し、左上から右に順次描画することの指定である。左上から下方向に描画させる場合には、mfcol と指定する。

次の 2 行がヒストグラムの表示である。ここでば、横座標 (xlim)、縦座標 (ylim) の範囲指定と、集計のための区分点の指定 (breaks) を明示的に指定しているが、これらは省略することも可能である。指定で breaks=5*(0:10) とあるのは境界点を表す。5*(0:10) は R の指定で 0,5,10,...,50 の数列を表す。

```
> par(mfrow=c(1,2))
> hist(fusionNV,xlim=c(0,50),ylim=c(0,30),breaks=5*(0:10))
> hist(fusionVV,xlim=c(0,50),ylim=c(0,30),breaks=5*(0:10))
```

### 15.2.4 箱ヒゲ図

ヒストグラムより単純で、分布を比較するために効果的な方法として箱ヒゲ図 (boxplot) がある (Tukey,1977; Hoaglin et al.1983)。これらの図 4 の中心部分の箱は、データの 25 パーセント点 (第 1 四分位点) と 75 パーセント点 (第 3 四分位点) の範囲を示し、その中の横線はメディアンを表す。また、箱の両端から点線が伸びた先の線分は、ヒゲ (whisker) と呼ばれる。これはデフォールトでは、四分位点から四分位点間距離の 1.5 倍の長さをとり、その範囲内で一番外側のデータの位置を示す。ヒゲより外側については、通常は全てのデータを表示する。これらの表示を検討することにより、データの大小、分布の広がり、歪み、最大・最小などを視覚的に確認できる。これらの図は次のようにして表示した。

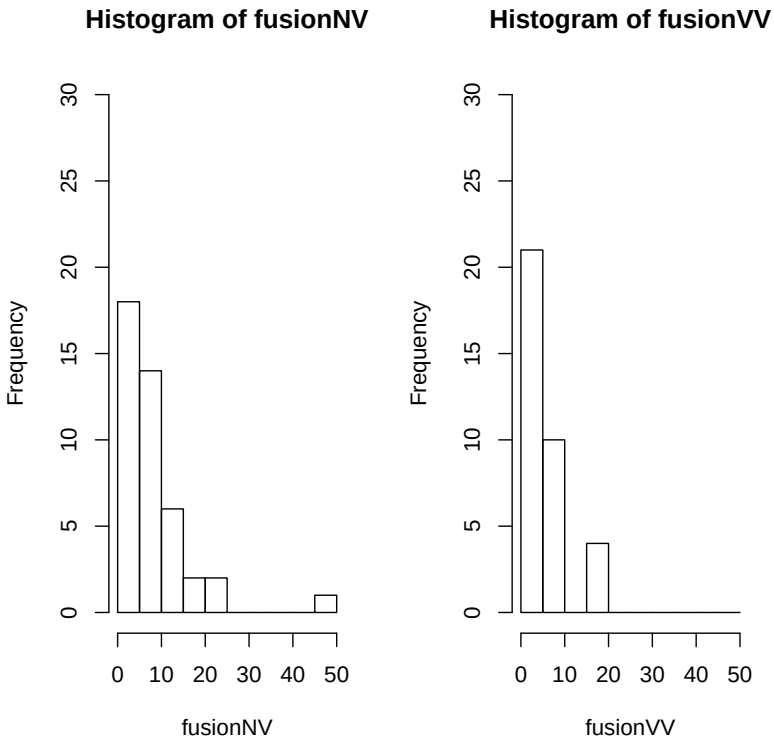


図 3: ヒストグラム

```
> boxplot(fusionNV,fusionVV,names=c("NV","VV"))
> boxplot(log(fusionNV),log(fusionVV),names=c("log NV","log VV"))
```

これらは、次のような命令によっても表示することができる。plot は引数として指定されたデータの型によって、適応的に機能を選択する関数であり、通常 2 つの変数の散布図を表示するために用いられる。ここでは、第 1 引数に指定された fusion$inst が離散値をとる変数 (R や S/S-PLUS の用語で**要因 (factor)** と呼ばれる) であるため、自動的に箱ヒゲ図が出力される。

```
> plot(fusion$inst,fusion$time)
> plot(fusion$inst,log(fusion$time))
```

標準正規分布では、第 3 四分位点の値は、0.6745 であり、これに四分位点間の距離の 1.5 倍を加えたヒゲの位置は 2.6980 となる。この上側確率は 0.0035(0.35 パーセント) であり、かなり小さい。より裾の重い $t$ 分布では、対応する確率はより大きくなる。例えば、自由度 4 の $t$ 分布では、第 3 四分位点は 0.7407、ヒゲの位置は 2.9628 であり、上側確率は 0.0207 である。更に自由度 1 の $t$ 分布 (Chaucy 分布) では、第 3 四分位点は 1、ヒゲの位置は 4、上側確率は 0.0780 であり、8 パーセントに近い。

### 15.2.5 分位点プロット

グラフによって分布の形を検討するための、もう一つの道具が**分位点プロット (quantile plot)** である。分位点プロットのなかでしばしば利用されるものには、分布の正規性を検討するための**正規確率プロッ**

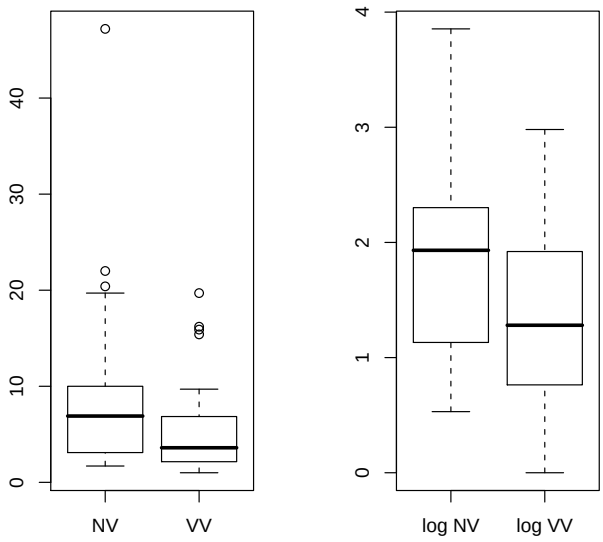


図 4: 両眼融合データの箱ヒゲ図 (左側：無変換、右側：対数)

**ト (normal-probability plot)** と、2 つの分布の関係をみるための**分位点 – 分位点プロット (quantile-quantile plot/ Q-Q plot)** とがある。

図 5 の上 2 つは、fusion データ (NV 条件) の反応時間を正規確率プロットによって表示したものである。左は無変換のデータであり、右は対数変換後のデータを示している。正規確率プロットは、データの各点について対応する下側確率を求め、その確率に対応する正規分布の分位点を横軸の値とし、データの値を縦軸にプロットする。もし、データの分布が正規分布に従っているならば、プロットされた点はほぼ直線上に並ぶはずである。左右の端で傾きが急になっているならば、これは正規分布に比べて裾の重い分布 (自由度の小さい $t$ 分布がこれにあたる) であることを意味する。逆に傾きが緩やかなら、裾のつまった分布 (一様分布など) であることを示す。正規確率プロットに示した直線は関数 qqline で表示したものであり、これは第 1 四分位点と第 3 四分位点とを結んでいる。変換前のデータはかなり正の方向に裾が重くなっているが、対数変換すると正規分布に近くなっている。ただし、小さい方の分布の裾が短い。

図 5 の下 2 つのグラフは、NV 条件のデータと VV 条件のデータを比較するための Q-Q プロットである。縦軸は先の正規確率プロットと同様にデータの分位点であるが、横軸も別のデータから求められた分位点の値である。もし、2 つのデータの分布が同一のものであるならば、表示される点はほぼ $X = Y$ の直線上にあるはずである。また、一方の分布が他方の分布を 1 次式で変換したもの (定数倍 + 定数) であれば、ほぼ直線上に点が表示されるはずである。左側は無変換のデータであり、右側は対数変換したもの同士を比較している。表示してある直線は $X = Y$ を示す。対応する分位点同士を比較すると一貫して NV データが VV データより大きな値を取っていることが分かる。Q-Q プロットにおいては、2 群のデータの個数が異なる場合には、件数の少ない方に表示点を合わせ、もう 1 群の分位点は補間により推定した値を用いている。

正規確率プロットを表示するには、関数 qqnorm を用いる。また、25 パーセント点と 75 パーセント点を結ぶ直線をプロットに追加するには、qqline を用いる。

分位点 – 分位点プロットを表示するには、qqplot を用いる。また、図の下段に表示してある直線は $Y = X$ であるが、これを表示するには abline(0,1) とする。abline($a$,$b$) を実行すると直線 $Y = a + bX$

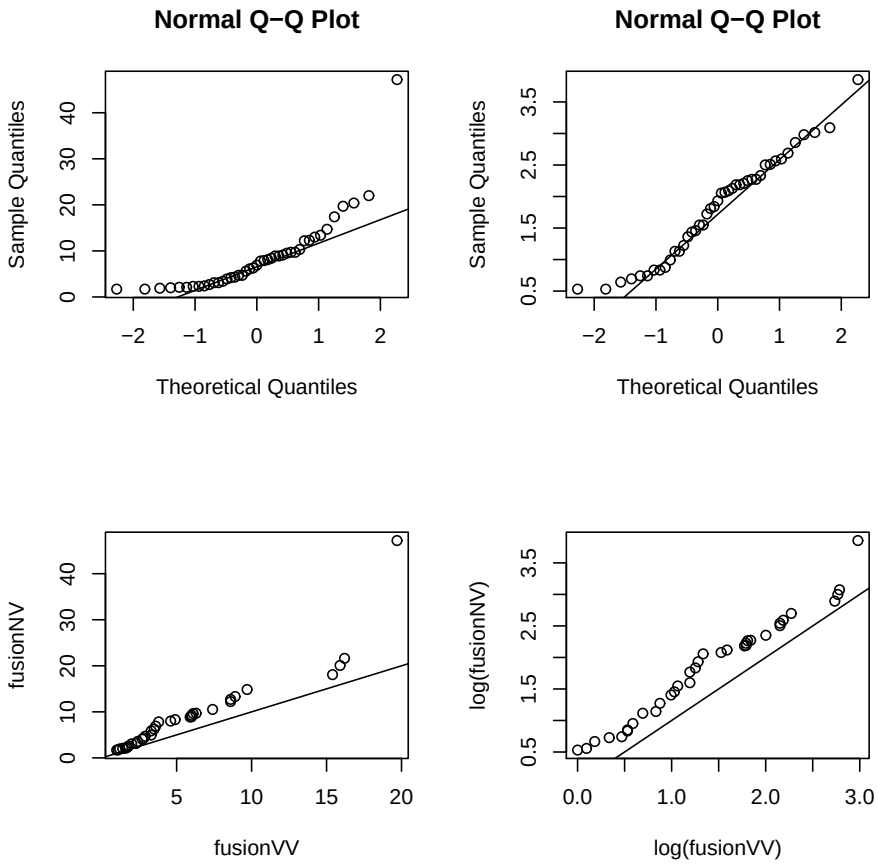


図 5: 正規確率プロットと分位点・分位点プロット

を画面に追加される。

```
> par(mfrow=c(2,2))          # 2 × 2 の表示を行なう。
> qqnorm(fusionNV)           # 正規確率プロット
> qqline(fusionNV)           # 25%点と 75%点を結ぶ
NULL                         # qqline の戻り値 (無視してよい)
> qqnorm(log(fusionNV))
> qqline(log(fusionNV))
NULL
> qqplot(fusionVV,fusionNV) #分位点-分位点プロット
> abline(0,1)                # Y=X の直線を描き込む
> qqplot(log(fusionVV),log(fusionNV)) # 対数変換して表示
> abline(0,1)
```

### 15.2.6 位置の差の検定

1 群および 2 群についての $t$ 検定および等分散を仮定しない Welch の検定は、関数 t.test で行なうことができる。オプションの指定により、対になったデータの検定もできる。デフォールトの設定では、両側の検定を行ない、差の 95%信頼区間を表示する。

これまでの検討からほぼ明らかだが、2 つのデータは無変換では分散がかなり異なる。しかし、対数をとることにより、正規性と等分散性の仮定は妥当なものとして扱える。Q-Q プロットから明らかなように、NV 条件の反応時間が明らかに大きい。無変換のデータに直接 $t$ 検定を実行しても、有意とはならないが、対数変換すると帰無仮説が棄却される。

```
> t.test(fusionNV,fusionVV,var.equal=T)

Two Sample t-test

data:  fusionNV and fusionVV
t = 1.9395, df = 76, p-value = 0.05615  # 5 %水準では有意にならない。
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.0809383  6.0990099
sample estimates:
mean of x mean of y
 8.560465  5.551429
```

```
> t.test(fusionNV,fusionVV,var.equal=F)

Welch Two Sample t-test

data:  fusionNV and fusionVV
t = 2.0384, df = 70.039, p-value = 0.04529
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.06493122 5.95314037
sample estimates:
mean of x mean of y
 8.560465  5.551429
```

```
> t.test(log(fusionNV),log(fusionVV),var.equal=T) # 対数をとる。

Two Sample t-test

data:  log(fusionNV) and log(fusionVV)
t = 2.319, df = 76, p-value = 0.02308  #等分散の仮定のもとでも有意
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.06077189 0.80034137
sample estimates:
mean of x mean of y
 1.820011  1.389454
```

2 群のデータの差を検出するためのもう一つの方法としては、ノンパラメトリック検定の一つである Mann-Wilcoxon-Whitney 検定 (Wilcoxon 順位和検定、Wilcoxon rank sum test) がある。この検定は、2 群のデータを一緒にして小さい方から順位をふり、各群に割り当てられた順位の和を用いて検定を行なう。順位の平均が 2 群で大きく異なれば差があるとみなす。関数 `wicox.test` によって、$t$ 検定とほぼ同様に検定を行なえる。デフォールトでは両側検定を行なう。この例では、$p$ 値は 0.0271 となっており、対数変換した場合の $t$ 検定による値と近い。この他に、比率の差の検定を行なうための `prop.test` などが利用できる。

```
>  wilcox.test(fusionNV,fusionVV)

Wilcoxon rank sum test with continuity correction

data:  fusionNV and fusionVV
W = 973, p-value = 0.02706
alternative hypothesis: true mu is not equal to 0

Warning message:
In wilcox.test.default(fusionNV, fusionVV) :
   タイがあるため、正確な p 値を計算することができません
```

## 15.3 2 変数の関連性

共分散および相関係数も求めることができる。関数var は、2 つのベクトルを引数に与えると、それらの共分散を計算する。計算される値は、1 変数の分散の場合と同じく、不偏推定量

$$\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\bar{x})(y_i-\bar{y})$$

である。また、var の引数に行列を与えると、各列を変数とみなして分散共分散行列を求める。

```
> ab2 <- c(15,14,13,12,11)
> ab1
[1] 1 2 3 4 5
> ab2
[1] 15 14 13 12 11
> var(ab1)                  # 分散を求める。
[1] 2.5
> var(ab2)
[1] 2.5
> var(ab1,ab2)              # 共分散を求める。
[1] -2.5
> cor(ab1,ab2)              # 相関係数を求める。
[1] -1                      # この場合、完全に線形の関係にあるので -1 になる。
```

2 つの連続変数の関連を検討するための、一番基本的な方法は散布図 (scatter plot) を描くことであるが、これは関数 plot によって可能である。

## 15.4 データセット longley

R には各種の関数の他に、サンプルデータが始めから用意されている。longley という名のオブジェクトは、サンプルデータの一つであり、J.W.Longley [14] による被雇用者数と GNP 等のデータである。内容についての解説は help(longley) でみることができる。

まず、

```
data(longley)
```

と入力すると、サンプルデータが作業領域にコピーされ利用可能となる。

names(longley) と入力すると、データフレームに含まれる変数名が表示される。longley$GNP または longley$Year などと R のなかで指定するとデータの内容をみることができる。names(longley) とすると，含まれる変数名が表示される。またattributes(longley) と入力すると、データフレームに付与された属性（変数名、行名など）が表示される。

## 15.5 相関係数の検定

関数cor は相関係数 (ピアソン積率相関係数)

$$\frac{\mathrm{cov}(x,y)}{\sqrt{\mathrm{var}(x)\mathrm{var}(y)}} = \frac{\sum_{i=1}^{N}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i-\bar{x})^2\sum_{i=1}^{N}(y_i-\bar{y})^2}}$$

を与える。またcor の引数が行列であると、各列を変数とみなし、それらの間の相関係数からなる行列 (相関行列) をつくり出す。上に述べた関数 var も、引数が行列の場合には分散共分散行列が結果となる。cov と cor で欠測値を含まないレコードのみを使う場合には use="na.or.complete" または use="complete.obs" オプションを指定する。

また、var,cor ともに、一つの引数がベクトルであり、他方が $m$ 列の行列であるとすると、前者と後者の各列の間の共分散または相関行列を求める。結果は長さ $m$ のベクトルになる。

```
> cor(longley$Employed, longley$GNP)
[1] 0.9835516
> round(cor(longley),2)   # 相関行列を求める。少数以下 2 桁で丸める。
             GNP.deflator  GNP Unemployed Armed.Forces Population Year Employed
GNP.deflator         1.00 0.99       0.62         0.46       0.98 0.99     0.97
GNP                  0.99 1.00       0.60         0.45       0.99 1.00     0.98
Unemployed           0.62 0.60       1.00        -0.18       0.69 0.67     0.50
Armed.Forces         0.46 0.45      -0.18         1.00       0.36 0.42     0.46
Population           0.98 0.99       0.69         0.36       1.00 0.99     0.96
Year                 0.99 1.00       0.67         0.42       0.99 1.00     0.97
Employed             0.97 0.98       0.50         0.46       0.96 0.97     1.00
> round(var(longley),2)
             GNP.deflator     GNP Unemployed Armed.Forces Population   Year
GNP.deflator       116.46 1063.60     625.87       349.03      73.50  50.92
GNP               1063.60 9879.35    5612.44      3088.04     685.24 470.98
Unemployed         625.87 5612.44    8732.23     -1153.79     446.27 297.30
Armed.Forces       349.03 3088.04   -1153.79      4843.04     176.41 138.24
Population          73.50  685.24     446.27       176.41      48.39  32.92
Year                50.92  470.98     297.30       138.24      32.92  22.67
Employed            36.80  343.33     164.91       111.77      23.46  16.24
             Employed
GNP.deflator    36.80
GNP            343.33
Unemployed     164.91
Armed.Forces   111.77
Population      23.46
Year            16.24
Employed        12.33
```

ピアソン積率相関係数は cor.test によっても求められる。こちらは統計的検定のための、各種の情報が出力される。

```
> cor.test(longley[,1],longley[,2])

Pearson's product-moment correlation

data:  longley[, 1] and longley[, 2]
t = 28.6666, df = 14, p-value = 7.816e-14
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9752586 0.9971563
sample estimates:
      cor
0.9915892
```

## 15.6 Spearman の順位相関係数と Kendall の $\tau$(タウ)

Spearman の順位相関係数と Kendall の $\tau$ を求めるには、`cor.test` を用いる。`method="s"`と指定すると Spearman の順位相関係数を求め、`method="k"`と指定すると Kendall の順位相関係数 ($\tau$) を求める。

Spearman の順位相関係数は、観測された 2 変量 $x_i, y_i, (i = 1, ..., N)$ をそれぞれ変数ごとに順位に変換した上で Pearson 積率相関係数を計算する。また、Kendall の $\tau$ は同一変数でのサンプル間の $N(N-1)/2$ 個の対,$(x_i, x_j), (y_i, y_j), (i < j)$ を考え、これらの 2 つの対が同順であるものと逆順であるものの個数を数える。順対の個数を $C$ とし、逆対の個数を $D$ とおく。ここで、もし同順位があればそれらはいずれにも数えない。$\{x_i\}$ における同順位の対の個数を $T_x$、$\{y_i\}$ における同順位の対の個数を $T_y$ とする。Kendall の $\tau$(より正確には $\tau_b$) はつぎの式で定義される。

$$\tau_b = \frac{C - D}{\sqrt{\{N(N-1)/2 - T_x\}\{N(N-1)/2 - T_y\}}} \tag{15}$$

Spearman の順位相関係数も Kendall の $\tau$ も、2 つの変数においてデータが完全に単調増加の関係にあれば $+1$ であり、逆に単調減少の関係であれば $-1$ となる。いずれも値は $[-1, +1]$ の範囲にあり、独立であれば 0 について対称に分布する。以下に各々の利用例を示す。

```
> cor.test(longley[,1],longley[,2],method="s")

Spearman's rank correlation rho

data:  longley[, 1] and longley[, 2]
S = 2, p-value = 6.396e-06
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.9970588   # Spearman の 順位相関係数

> cor.test(longley[,1],longley[,2],method="k")

Kendall's rank correlation tau

data:  longley[, 1] and longley[, 2]
T = 119, p-value = 1.529e-12
alternative hypothesis: true tau is not equal to 0
sample estimates:
      tau
0.9833333  # Kendall のタウ
```

## 15.7 2 値変数の関連性

連続変数間の相関係数の検定と同様に、2 値変数間の関連を調べるための関数も、幾つか用意されている。2 次元分割表の独立性の検定に多用される**カイ 2 乗検定 ($\chi^2$ test)** は、関数 `chisq.test` で行なえる。また、データの頻度が少ない場合に用いられる **Fisher の正確検定 (Fisher's exact test)** は `fisher.test` によって計算できる。

関数 `chisq.test` を用いて 2 元分割表の独立性の検定を行なうには、つぎのように分割表を行列として、引数に直接指定すればよい。$2 \times 2$ 表の検定については、Yates の連続補正と呼ばれる修正をデフォールト指定で行なっている。

```
> chisq.test(rbind(c(189,10845),c(104,10933)))

Pearson's Chi-squared test with Yates' continuity correction

data:  rbind(c(189, 10845), c(104, 10933))
X-squared = 24.4291, df = 1, p-value = 7.71e-07
```

また、x と y をそれぞれ離散値を持つ同じ長さの変数 (`factor` ) とすると、次のように集計する以前のデータを直接引数とすることもできる。

```
> chisq.test(x,y)
```

Fisher の正確検定は、$2 \times 2$ 表の独立性の検定に、$\chi^2$ と同様の目的で利用される。但し、$\chi^2$ 検定が漸近的な統計量の性質を用いているため、セルのデータの頻度が少ないとき (最小のセルの頻度が 5 以下では問題があるといわれている) には適用できない。周辺度数が固定された $2 \times 2$ 表において、行と列とが独立ならば $n_{11}$ の値は、**超幾何分布 (hypergeometric distribution)** という確率分布に正確に従う。この性質を用いた検定が、Fisher の正確検定と呼ばれるものである。この方法は関数 `fisher.test` で行なえる。データの指定方法は `chisq.test` と同様である。また、結果は両側検定で与えられる。

```
> fisher.test(rbind(c(3,1),c(1,3)))

Fisher's Exact Test for Count Data

data:  rbind(c(3, 1), c(1, 3))
p-value = 0.4857
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
   0.2117329 621.9337505
sample estimates:
odds ratio
  6.408309
```

# 16 確率分布と乱数

## 16.1 2 項分布

`dbinom` という関数で 2 項分布の確率を求めることができる。2 項分布 $X \sim Bin(n,p)$ とするとき、$X$ が値 $m$ をとる確率は

$$\Pr(X = m) \ = \ \left( \begin{array}{c} n \\ m \end{array} \right) p^m (1-p)^{n-m}$$

で与えられる。この値は `dbinom(`$m$`,`$n$`,`$p$`)` で得られる。以下は $n = 3, p = 0.5$ の例である。

```
> dbinom(0,3,0.5)
[1] 0.125
> dbinom(1,3,0.5)
[1] 0.375
> dbinom(2,3,0.5)
[1] 0.375
> dbinom(3,3,0.5)
[1] 0.125
```

`pbinom` は累積確率 $\Pr(X \leq m)$ を与える。

```
> pbinom(0,3,0.5)
[1] 0.125
> pbinom(1,3,0.5)
[1] 0.5
> pbinom(2,3,0.5)
[1] 0.875
> pbinom(3,3,0.5)
[1] 1
```

a に 0 から 10 の数列を用意する。つぎの様に指定すると、b は dbinom(0,10,0.5), dbinom(1,10,0.5), ..., dbinom(10,10,0.5) の値を含む長さ 11 のベクトルになる。この 2 項分布の形をグラフで確認する。plot 命令でtype="h"を指定すると、アスタリスクではなく、垂直方向の直線でデータを表示する。

```
> a <- 0:10              # 0〜10 の数列をつくる。
> b <- dbinom(a,10,0.5)  # 2 項分布 Bin(10,0.5) の値を x=0〜10 について
>                        # 得る。
> b
 [1] 0.0009765625 0.0097656250 0.0439453125 0.1171875000 0.2050781250
 [6] 0.2460937500 0.2050781250 0.1171875000 0.0439453125 0.0097656250
[11] 0.0009765625
> X11()
> plot(a,b)
> plot(a,b,type="h")
```

qbinom($r$,$n$,$p$) は 2 項分布 $Bin(n,p)$ の確率 $r$ に対応する分位点 (quantile) を与える。すなわち $\Pr(X \leq x_0) \geq r$ となる最小の $x_0$ を与える。

**課題**

$n$ と $p$ をいろいろに変化させて、確率分布の形を確認する。

## 16.2 2 項分布乱数

R の関数 rbinom($k$,$n$,$p$) を指定すると $Bin(n,p)$ に従う乱数が $k$ 個得られる。

```
> rbinom(10,3,0.5)
 [1] 2 2 2 1 2 2 1 1 2 2
> rbinom(10,3,0.5)
 [1] 1 2 1 2 3 1 1 1 0 3
```

乱数なので、1 回目と 2 回目で値が異なる。同一の乱数系列が必要な場合には、乱数を生成する前に set.seed($i$) によって乱数系列の初期値を指定する。ここで $i$ は 0 から 1000 までの間の整数である。

**課題**

2 項分布 $Bin(n,p)$ は、理論的には平均 $np$、分散 $np(1-p)$ を持つ。実際に乱数を 200 個程度生成し、それらの平均と分散を計算してみなさい。

## 16.3 ポアソン (Poisson) 分布

2 項分布において $n \times p$ の値を一定に保ちながら、$n$ の値を次第に大きくしてみると、分布の形が次第に一定に近づいてゆくのがわかる。

```
> a <- 0:10                          # a に 0〜10 の数列を代入
> plot(a,dbinom(a,10,0.3),type="h") # Bin(10,0.3) i.e. n=10,p=0.3 の 2 項分布
> plot(a,dbinom(a,100,0.03),type="h") # Bin(100,0.03) n × p は同じ
> plot(a,dbinom(a,200,0.015),type="h") # Bin(200,0.015)
```

この極限分布 (分布の形が近付く先) をポアソン (Poisson) 分布と呼び、$Po(\lambda)$ と表記する。ここで $\lambda = n \times p$ にあたる。ポアソン分布の平均は $\lambda$ であり、分散も $\lambda$ である。また、確率は

$$\Pr(X = x) = \frac{\lambda^x}{x!} e^{-\lambda}$$

であらわされる。ここで $e$ は自然対数の底 2.718282 である。R の関数 `dpois(`$x$`,`$\lambda$`)` は母数 $\lambda$ を持つポアソン分布の $x$ における確率を与える。`ppois(`$x$`,`$\lambda$`)` は累積確率をあたえ、`qpois(`$p$`,`$\lambda$`)` は確率 $p$ に対応する分位点を与える。また、`rpois(`$n$`,`$\lambda$`)` はポアソン分布に従う乱数を $n$ 個生成する。

**課題**

$Bin(n,p)$ と $Po(n \times p)$ の確率の違いを、$x = 0, ..., n$ について求めなさい。その差の最大はどの程度か? つぎのようにすると、$Bin(100, 0.03)$ と $Po(3)$ の確率値の違いがベクトル`diff1` として求められる。

```
diff1 <- dbinom(0:100,100,0.03) - dpois(0:100,3)
```

## 16.4 正規分布

2 項分布において平均を 0, 分散を 1 と基準化してみる。$Bin(n,p)$ の平均は、$n \times p$ である。分散は $np(1-p)$ なので、標準偏差は $\sqrt{np(1-p)}$ となる。$X \sim Bin(n,p)$ とし、

$$Z \sim \frac{X - np}{\sqrt{np(1-p)}}$$

とおくと、これは平均 0, 分散 1 である。$n$ が大きくなると $Z$ の分布は一定の形に近付いていく。この分布を標準正規分布 $N(0,1)$ と呼ぶ。標準というのは、平均 0、分散 1 という意味である。$Z$ が標準正規分布に従うとする。$Y = \sigma Z + \mu$ のとき、$Y$ は平均 $\mu$, 分散 $\sigma^2$ の正規分布 $N(\mu, \sigma^2)$ に従うという。

正規分布 $\mathrm{N}(\mu, \sigma^2)$ の確率密度関数は

$$f(y) \;=\; \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-\mu)^2/2\sigma^2}$$

で与えられる。この式を区間 $(y_1, y_2]$ で積分すると、$\Pr(y_1 < Y \leq y_2)$ が得られる。

R の関数 `dnorm(`$y$`,`$\mu$`,`$\sigma$`)` は平均 $\mu$, 標準偏差 $\sigma$(分散ではないことに注意) の正規分布 $N(\mu, \sigma^2)$ の確率密度関数の値である。また、`pnorm(`$y$`,`$\mu$`,`$\sigma$`)` は累積確率 $\Pr(Y \leq y)$ を与える。`qnorm(`$p$`,`$\mu$`,`$\sigma$`)` は確率 $p$ に対応する分位点を与え、また `rnorm(`$n$`,`$\mu$`,`$\sigma$`)` は $N(\mu, \sigma^2)$ に従う乱数を $n$ 個生成する。

**課題**

$X$ を $Bin(100, 0.3)$ に基づく確率変数とする。$X$ の確率 0.95 に対応する分位点を求める (`qbinom` を使う)。 平均 $30 = 100 \times 0.3$, 標準偏差 $\sqrt{21} = \sqrt{100 \times 0.3 \times 0.7}$ の正規分布の確率 0.95 に対応する分位点と比較してみなさい。平方根を求めるには関数 `sqrt` を使う。また、他の確率に対応する分位点の比較も試みよ。

## 16.5 一様分布

一番単純な連続分布である一様分布 (uniform distribution) についても、上記の分布と同様に各種の関数が利用できる。一様乱数の生成は `runif(`$n$`)` で行なう。ここで $n$ は生成すべき乱数の個数である。乱数の範囲は標準では $(0, 1)$ 区間である。

**課題**

$0-1$ 区間の値をとる一様乱数 2 個からなる対を $n$ 組生成する。各乱数の対を $(u_i, v_i)$ とし、このうち $u_i^2 + v_i^2 < 1$ である対の個数を $m$ とする。$n$ が十分大きければ $4m/n$ は円周率 $\pi$ を近似するはずである (なぜか？)。実際に一様乱数を生成して確認しなさい。円周率は `pi` という名前 R で利用できる。

# 17 統計モデルの推定

## 17.1 StatLabs データ: 母親の喫煙と新生児体重

回帰分析 (共分散分析) の例として StatLabs 教科書 (Nolan and Speed,2000) で扱われているデータの分析を行う。

**データの所在**

`http://stat-www.berkeley.edu/users/statlabs/labs.html#babies`

にある。

データはブラウザで取得できる。

WEB ページ上で Birth weight II と表示されている箇所でマウスの右ボタンをクリックし、「名前をつけて保存」を選択する。

SAS でのプログラミング例を参考に示す。

### 17.1.1 SAS による分析例

```
data babies;
   infile 'babies.data' firstobs=2;
   input  bwt gestat parity age height weight smoke ;
   if bwt = 999 then bwt = . ;
   if gestat = 999 then gestat = . ;
   if parity = 9 then parity = .;
   if age = 99 then age = .;
   if height = 99 then height = . ;
   if weight = 999 then weight = . ;
   if smoke = 9 then smoke = . ;
run;
* proc print;    /* 先頭の星をとると全件表示 */
proc means;
proc sort  data=babies;
   by smoke;
proc means; by smoke;
proc glm data=babies;
  class parity smoke;
  model bwt = gestat parity age height weight smoke
            / ss1 ss3  solutions ;
run;
```

### 17.1.2 R によるデータ分析例

以下の内容は、対話的に入力することも可能だが、notepad、emacs などでファイルにあらかじめ作成しておいたほうが容易である。ファイルに記述された内容を R で実行するにはsource("ファイル名") と入力する。

```
# 見出しがついているので、変数名が自動的につけられる。
  babies <- read.table("babies.data",header=T)

  dim(babies)
[1] 1236    7

#  babies データフレームの変数を、変数名だけで参照可能にする。
  attach(babies)

# 欠測値の処理を行う。
  babies$bwt <- replace(bwt, bwt == 999, NA);
  babies$gestation <- replace(gestation, gestation == 999, NA);
  babies$parity <- replace(parity, parity == 9, NA);
  babies$age <- replace(age, age == 99 ,NA);
  babies$height <- replace(height, height== 99,NA);
  babies$weight <- replace(weight, weight== 999,NA);
  babies$smoke <- replace(smoke, smoke == 9, NA);

# 離散値をとる変数を factor として宣言し、カテゴリーに名前をつける。
  babies$parity <- factor(parity, levels=c(0,1),
                            labels=c("First","Other"));
  babies$smoke <- factor(smoke,levels=c(0,1),
                           labels=c("Nosmoke","Smoke") );

  detach("babies")   # R の場合、修正を反映させるには detach してから
  attach(babies)   # 再 attach が必要のようだ。
```

```
# babies データフレームを attach しておくと、変数名だけで参照できる。
#  下の例では data=baies を指定しなくても OK。
  babies.lm <- lm(bwt ~ gestation + parity + age + height + weight + smoke,
                  na.action=na.omit, data=babies)

  summary(babies.lm)  # 分析結果の概要
  anova(babies.lm)    # 分散分析表 SS は SAS の TypeI に相当。

# factor 宣言された変数の回帰係数については注意が必要。
# どのようなデザイン行列 X が用いられているかは、
# つぎのようにして調べる。
 contrasts(parity)
 contrasts(smoke)

 plot(babies.lm)   #各種のモデル診断用の図を表示
```

```
> summary(babies.lm)

Call:
lm(formula = bwt ~ gestation + parity + age + height + weight +
    smoke, data = babies, na.action = na.omit)

Residuals:
     Min       1Q   Median       3Q      Max
-57.6128 -10.1887  -0.1346   9.6826  51.7131

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -80.41085   14.34657  -5.605 2.60e-08 ***
gestation     0.44398    0.02910  15.258  < 2e-16 ***
parityOther  -3.32720    1.12895  -2.947  0.00327 **
age          -0.00895    0.08582  -0.104  0.91696
height        1.15402    0.20502   5.629 2.27e-08 ***
weight        0.05017    0.02524   1.987  0.04711 *
smokeSmoke   -8.40073    0.95382  -8.807  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 15.83 on 1167 degrees of freedom
  (62 observations deleted due to missingness)
Multiple R-squared: 0.258, Adjusted R-squared: 0.2541
F-statistic: 67.61 on 6 and 1167 DF,  p-value: < 2.2e-16
```

```
> anova(babies.lm)
Analysis of Variance Table

Response: bwt
            Df Sum Sq Mean Sq  F value    Pr(>F)
gestation    1  65450   65450 261.2078 < 2.2e-16 ***
parity       1   2345    2345   9.3578  0.002271 **
age          1    216     216   0.8627  0.353179
height       1  12518   12518  49.9594 2.695e-12 ***
weight       1   1683    1683   6.7187  0.009660 **
smoke        1  19437   19437  77.5713 < 2.2e-16 ***
Residuals 1167 292409     251
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## 17.2 ロジスティック回帰

ロジスティック回帰は 2 値をとる従属変数の平均構造を、幾つかの説明変数の線形和によって説明しようとするものである。

線形の回帰分析は、説明変数 $\boldsymbol{X} = (X_1, ..., X_p)$ を固定した場合の $Y$ の平均を次のような式で表す。

$$E(Y|\boldsymbol{X}) = \mu_X = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p \tag{16}$$

さらに、$Y$ は $\mu_x$ を平均とする正規分布 $N(\mu_x, \sigma^2)$ に独立に従うと仮定している。

ロジスティック回帰は、上の式における次の 2 点を変更したものである。まず、説明変数 $\boldsymbol{X}$ を固定した際、被説明変数 $Y$ は 2 項分布 $Bin(m_x, \pi_X)$ に独立に従うとする。さらに $\pi_X = E(Y|X)$ が、つぎのよう

な式で表されると仮定する。

$$\mathrm{logit}(\pi_X) = \log \frac{\pi_X}{1-\pi_X} = \eta_X = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p \tag{17}$$

上の式は、ロジット (logit) 変換と呼ばれる。一方、$\log \frac{\pi_X}{1-\pi_X} = \eta_X$ の逆関数は、

$$\pi_X = \frac{\exp(\eta_X)}{1+\exp(\eta_X)} \tag{18}$$

となる。これは**ロジスティック (logistic) 関数**と呼ばれる。$\eta_X$ の値が $-\infty$ から $+\infty$ まで変化するとき、$\pi_X$ は単調に 0 から 1 まで変化する。

次に問題になるのは、どのような方法で $\beta_0, \beta_1, ..., \beta_p$ の値を推定するかである。現在、最も一般的な推定方法は**最尤法 (maximum likelihood method)** と呼ばれるものである。これは、観測されたデータを最も生成する可能性の大きいと思われるモデルを、ある種の確率論的な基準によって推定しようとするものである。

ここでは、ロジスティック回帰による簡単な分析例を示す。

## 17.3 多重分割表の入力

表 5 に示す多重分割表データの分析を考える。これは洗剤の利用についての選好データであり、Cox & Snell (1981) の表 L.1 より引用した。オリジナルの分析は Reis & Smith (1963) による。各被験者は、新製品 X を標準品 M と比較し、いずれを好むかを回答する。表中 $Y_j$ は、該当する条件において X を選好した人数であり、$n_j - Y_j$ 人が M を選好している。

表 5: 洗剤の選好データ ( Cox & Snell,1981,Table L.1 )

| 水の硬度 | | M 使用未経験 | | M 使用経験 | |
|---|---|---|---|---|---|
| | | 水温 | | 水温 | |
| | | 低 | 高 | 低 | 高 |
| 硬 | $Y_j$ | 68 | 42 | 37 | 24 |
| | $n_j$ | 110 | 72 | 89 | 67 |
| 中 | $Y_j$ | 66 | 33 | 47 | 23 |
| | $n_j$ | 116 | 56 | 102 | 70 |
| 軟 | $Y_j$ | 63 | 29 | 57 | 19 |
| | $n_j$ | 116 | 56 | 106 | 48 |

このデータを入力するには、テキストファイルに条件のコードと数値を記入して、`read.table` などの機能を用いることもできるが、ここではベクトルを直接指定する方法を示す。

```
CoxSnellL <- data.frame(
  yes=c(  68,   42,   37,   24,
          66,   33,   47,   23,
          63,   29,   57,   19 ),
  total=c(110,  72,   89,   67,
          116,  56,  102,   70,
          116,  56,  106,   48 ),
  water=c(rep("Hard",4),rep("Medium",4),rep("Soft",4)),
  temp=rep(c("Low","High"),6),
  use=rep(c("NonUser","NonUser","User","User"),3)
)
```

ここで用いられている関数 rep は、第 1 引数に指定されたオブジェクトを、第 2 引数に指定された回数繰り返してベクトルを生成する。

この命令を実行すると、データフレーム CoxSnellL の内容はつぎのようになる。

```
> CoxSnellL
   yes total  water temp     use
 1  68   110   Hard  Low NonUser
 2  42    72   Hard High NonUser
 3  37    89   Hard  Low    User
 4  24    67   Hard High    User
 5  66   116 Medium  Low NonUser
 6  33    56 Medium High NonUser
 7  47   102 Medium  Low    User
 8  23    70 Medium High    User
 9  63   116   Soft  Low NonUser
10  29    56   Soft High NonUser
11  57   106   Soft  Low    User
12  19    48   Soft High    User
```

上の例では water のカテゴリーは入力順に Hard,Medium,Soft となっており問題ないが、一般には R での内部表現での順番とカテゴリーが本来持つべき順番が等しいとは限らない。カテゴリーの順を明示的に指定するには、例えば

```
> CoxSnellL$water <-
  ordered(CoxSnellL$water,c("Soft","Medium","Hard"))
```

のように、関数 ordered を用いて指定できる (上の例では、カテゴリーの内部表現が逆順になる)。

## 17.4 ロジスティック回帰の推定

次のように関数 glm を用いてロジスティック回帰を行なえる。glm は一般化線形モデルとよばれるより一般的な統計モデル ( 線形の重回帰、分散分析、対数線形モデルを特殊ケースとして含む) の推測のための関数である。分布族の指定 (family) を 2 項分布とし、リンク関数と呼ばれる指定をロジット関数にする

と (以下の分析では、分布族を 2 項分布に指定すると、リンク関数は特に指定しない限りロジット関数に自動的に設定されている) ロジスティック回帰になる。

```
> CSL.log1 <- glm(yes/total ~ water + temp + use,
  family = binomial, data = CoxSnellL, weights = total)
```

関数 glm の中で最初の引数はモデル式の定義である。ここでは、yes の回答比率を被説明変数とし、それを水質 (water), 水温 (temp)、利用経験 (use) の主効果によって説明しようとしている。family=binomial は、確率モデルの分布族の指定であり、ここでは 2 項分布を意味している。data は分析対象となるデータフレームの指定、また weight は重みの指定である。ここでは、yes または no の 2 値をとる回答が total に示される回数繰り返されているので、このように指定する。

線形の回帰モデルを指定するには、glm において family=gaussian と指定するか、または関数 lm を用いる。また、説明変数に交互作用項を含める場合には x1:x2 などのように:によって離散値をとる変数をモデル式に含める。また x1*x2 のように表記すると、主効果と交互作用項の両者を同時に含める。R での関数 glm は SAS の GENMOD プロシジャにほぼ相当するものであり、SAS の GLM は S-PLUS の lm に相当する。

実行結果の概略は summary という関数で表示される。R ではカテゴリーの表示が SAS と比べて親切ではなく、若干結果を読みとりづらい。water.L とあるのは、water によって生成された第 1 対比 (線形成分) を意味する。これは

```
> round(contrasts(CoxSnellL$water),2)
          .L    .Q
Soft   -0.71  0.41
Medium  0.00 -0.82
Hard    0.71  0.41
```

の様に確認することができる。

つまり、water.L は 0.71(Hard-Soft) に相当し、また water.Q は 0.41(Hard-2Medium+Soft) に相当する。temp と use についても同様に確認すると、

```
> contrasts(CoxSnellL$temp)
     Low
High   0
Low    1
> contrasts(CoxSnellL$use)
        User
NonUser    0
User       1
```

となっていることが分かる。

```
> summary(CSL.log1)

Call:
glm(formula = yes/total ~ water + temp + use, family = binomial,
    data = CoxSnellL, weights = total)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.2785  -0.6382  -0.1765   0.5601   1.5738

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.12500    0.12295   1.017   0.3093
water.L     -0.01493    0.11146  -0.134   0.8935
water.Q      0.04905    0.10989   0.446   0.6554
tempLow      0.25665    0.13286   1.932   0.0534 .
useUser     -0.56702    0.12775  -4.439 9.05e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 32.826  on 11  degrees of freedom
Residual deviance:  8.228  on  7  degrees of freedom
AIC: 75.926

Number of Fisher Scoring iterations: 3
```

また、各変数のモデルへの寄与を調べたい場合には、関数 anova を用いて表示を行なえる。表示されている Deviance は、モデルに逐次的に変数 (要因) を加えた場合の尤度の変化量（より正確には対数尤度の 2 倍の変化量）である。SAS の GENMOD プロシジャにおける Type1 の統計量と同等のものである。

```
>  anova(CSL.log1)
Analysis of Deviance Table

Model: binomial, link: logit

Response: yes/total

Terms added sequentially (first to last)


      Df Deviance Resid. Df Resid. Dev
NULL                    11     32.826
water  2    0.395         9     32.430
temp   1    4.310         8     28.120
use    1   19.892         7      8.228
```

モデルによって得られた推定値 $\eta_X$ と $\hat{\pi}_X$ とは、それぞれ predict および fitted によって得られる。

```
> round(predict(CSL.log1),4)
      1       2       3       4       5       6       7       8       9      10
 0.3911  0.1345 -0.1759 -0.4325  0.3416  0.0850 -0.2254 -0.4821  0.4122  0.1556
     11      12
-0.1548 -0.4114
> round( fitted(CSL.log1),4)
     1      2      3      4      5      6      7      8      9     10     11
0.5966 0.5336 0.4561 0.3935 0.5846 0.5212 0.4439 0.3818 0.6016 0.5388 0.4614
    12
0.3986
```

# 第 IV 部
# 作図と印刷

## 18　R による作図

R でグラフィックを出力する場合には、デフォールトの画面出力を除いて，あらかじめ作図デバイスを指定しなければならない。どのようなデバイスに出力するか (画面に表示する、出力の描画フォーマット等) によって画面の大きさ、縦横の指定など各種の指定方法がある。これらの指定は、R を終了するか、`graphic.off()` 関数によってグラフィックデバイスを終了させるか，あるいは別の機器を指定しなおすまで有効である。

### 18.1　画面に表示する

WindowsPC を使用している場合，デフォールトで Windows の GUI への出力が可能である．明示的に `windows()` 関数を実行すると，現在表示されている描画ウィンドウのほかにウィンドウが表示され，これがグラフィック出力先となる。また Linux 上では X11 デバイスがデフォールトで使用できる。Linux 上で X11 のウィンドウに日本語を表示するには、つぎのように日本語に対応したフォントを指定する。フォント指定の文字列は xfontsel ユティリティで探す。

```
fixed <- X11Font("-*-fixed-%s-%s-*-*-%d-*-*-*-*-*-*-*")
X11Fonts(fixed=fixed)
```

Windows での画面出力のウィンドウでは「File」メニューから，プリンタへの出力，および「Save as」メニュー選択により各種の描画フォーマット (metafile,postscript,pdf,png,bmp,jpeg) を選択できる。ただし R-2.8.1(日本語パッチなしのもの）においては postscript と PDF への日本語表示ができない。

### 18.2　描画ファイルへの出力

描画命令の実行前にデバイスを明示的に指定することにより，上述の各種の描画フォーマットでファイルへの出力を直接行うことも可能である。

```
> png("pngfile1.png")
> plot(rnorm(10),rnorm(10),main="プロットのタイトル")
> dev.off()
```

のように指定することにより描画ファイルが出力される。ここで `plot` が実行されても `graphics.off()` か `dev.off()` が実行されるか R が終了するまではファイルへの出力は実際には行われない。

postscript と PDF で直接保存ファイルを指定する場合には、フォントグループを `family="Japan1"`と指定することにより日本語が表示できる。

### 18.3　作図機器を変えると図も異なる

作図機器の指定を変更すると、出力するグラフの細かなところが異なることがある。Windows 上でグラフを作成して確認し、満足できる結果が得られたので PostScript を指定して印刷してみると、少し違っ

た出力になる。もっとも、グラフの内容 (分析の結果) が異なるわけではなく、図や文字の大きさ、点をプロットするシンボル等が異なるだけであるが。

### 18.4 計算経過の保存

R を実行中に

```
> sink("ファイル名")
```

と入力すると、これ以降は R のテキスト出力は画面の代わりに、指定されたファイルに書き込まれる。ただし、利用者が入力した部分はファイルには記録されない。

```
> sink()
```

と指定すると、画面に表示されるように戻る。ベクトルの内容のみを出力するには

```
> cat(オブジェクト名,file="ファイル名")
```

とすればよい。指定されたオブジェクトの内容がファイルに書き出される。ただし、見出しなどは省略される。

他には、

1. Windows GUI の画面で「ファイルを保存」メニューを選択して、応答経過のテキストを保存する。
2. Emacs/Meadow で UNIX のシェルプログラム（または Windows 上の CMD）を起動するか ESS を利用して R を起動する。(ただし、Windows 上の CMD で R(Rterm.exe) を直接起動すると日本語がうまく扱えない。ただし、ファイルからの入出力では日本語が使える。また、ESS や Emacs 中のシェルでは日本語が使える。)
3. X Window System のカット・アンド・ペースト機能を用いて、画面に表示されている内容をファイルに保存する。

などの方法がある。

## 19 2 次元プロット

描画の方法については既に説明したが，各種のオプションなどについて追加の説明をする。

### 19.1 データの表示

2 次元の散布図を描くには `plot` を使用する。

```
plot(x, y)
```

のように使用する。ここで、`x` と `y` はそれぞれ x 座標、y 座標を与える数値ベクトルである。

次を実行せよ。

```
> x1 <- c(1,2,3,4)
> y1  <- c(1,3,2,5)
> plot(x1, y1)
```

4 個の点がプロットされる。ここで、plot(x1, y1) のかわりに、

```
> plot(x1, y1, type="l")
```

とすれば、4 点が線で結ばれる。type="b" なら点と線、type="o" なら点と線の重ね書きとなる。

### 19.2 複数の折れ線の表示

一枚のグラフに、複数のデータの折れ線 (など) を重ねて表示するには、以下のいずれかの方法をとる。

1. まず最初に plot で 1 つのグラフをプロットする。つぎに関数 lines または points を用いて、グラフ上に線分または点を重ね書きする。

2. 関数 matplot を利用する。この関数は引数にベクトルだけでなく、行列を指定することができる。x が長さ $n$ のベクトルであり、y が $n \times m$ の行列であるとする。 matplot(x,y,type="l",lty=1:m) と指定すると、$m$ 回 plot を繰り返し、重ね書きするのと同様の描画を行なう。x にも行列を指定することが可能であり、この場合には x の列が順次描画に用いられる。また、描画の線種も各データ毎に変化する (lty の指定による)。

lines、points と同様の機能を持つ matlines、matpoints も利用できる。

描画の詳細を指定のためのオプションである type、lty、pch、col などについては、lines のオンラインマニュアルに説明がある。

### 19.3 棒グラフと円グラフ

棒グラフを表示するには barplot、円グラフを表示するには pie を用いる。

## 20 矢印や文字を書き込む

plot で描いた図に、矢印を書き込むには、arrows を使用する。使い方は既に述べたが、arrows(p1, p2, q1, q2) のようにする。この時、座標 (p1, p2) から座標 (q1, q2) に向かう矢印が描かれる。

プロットに文字を書き込むには、text を用いる。text(x, y, z) のように使用する。x, y は文字を描く x, y 座標を与える数値ベクトル、z はその場所に描く文字列を与える文字列ベクトルである。z を省略すると、1 から順に番号が書き込まれる。

なお、arrows や text は、あらかじめ plot で図を描いておかなければ使用できない。

## 21 3 次元以上のプロット

### 21.1 persp

persp は、3 次元の鳥瞰図 (perspective plot) を描く。

```
> persp(x,y,z)
```

ここで、x, y は数値ベクトル、z は行列である。x, y で定められる等間隔のグリッド上に z がプロットされる。

表 6: plot のためのオプション（Windows 上、抜粋）

| 機能 | オプション | 値 | 指定内容 |
|---|---|---|---|
| 描画タイプ | type | "p" | 点を表示 |
| | | "l" | 線分で繋ぐ |
| | | "b" | 点を表示し線分で繋ぐ |
| | | "o" | 点と線分の重ね描き |
| | | "n" | 表示しない |
| | | "h" | 垂直線（ニードル） |
| 線種 | lty | 0 | 表示なし |
| | | 1 | 実線 |
| | | 2 | 破線 |
| | | 3 | 点線 |
| | | 4 | 一点破線 |
| | | 5 | 長破線 |
| | | 6 | 二点破線 |
| 色 | col | 0 | 表示なし |
| | | 1 | 黒 |
| | | 2 | 赤 |
| | | 3 | 緑 |
| | | 4 | 青 |
| | | 5 | シアン |
| | | 6 | マゼンタ |
| | | 7 | 黄 |
| | | 8 | 灰色 |
| 描点記号 | pch | 0 | 正方形 |
| | | 1 | 円 |
| | | 2 | 三角 |
| | | 3 | プラス |
| | | 4 | X |
| | | 5 | 菱形 |
| | | 6 | 逆三角 |
| | | 7 | 対角線入り正方形 |
| | | 8 | 星 |
| | | 9 | 対角線入りひし形 |
| | | 10 | プラスと円 |
| | | 11 | 三角と逆三角 |
| | | 12 | 正方形とプラス |
| | | 13 | 円と X |
| | | .. | .. |
| | | 文字（１文字） | 指定された文字を表示 |

例えば、次のように指定すると平面が表示される。

```
> x2 <- c(1,2,3,4)
> y2 <- c(1,2,3,4)
> z2 <- matrix(c(1,2,3,4,2,3,4,5,3,4,5,6,4,5,6,7),4,4)
> persp(x2,y2,z2)
```

また、

```
> x3 <- 0:50/50*2*pi
> y3 <- 0:50/50*2*pi
> z3 <- matrix(x3,51,51) + matrix(y3,51,51,byrow=T)
> persp(x3, y3, sin(z3))
> persp(x3, y3, cos(z3))
```

で、$z = \sin(x + y)$ と $z = \cos(x + y)$ (図 6) のグラフが描かれる。

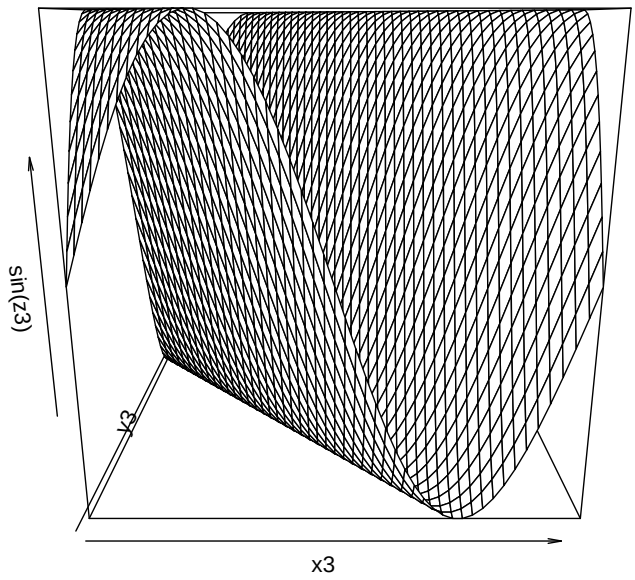


図 6: cos(x+y) の鳥瞰図

## 21.2 pairs

関数 paris は多変量のデータを、2 変量のペア毎にプロットする。

```
pairs(x)
```

x は行列で、各列が変量に対応する。

```
> m1 <- matrix(0:99,100,4)
> m1
> m1[,2] <- 100 - m1[,2]
> m1[,3] <- sin(m1[,3]/100*2*pi)
> m1[,4] <- cos(m1[,4]/100*2*pi)
> m1
> pairs(m1)
```

また、R のサンプルデータを用いて、つぎのように表示することもできる (図 7)。

```
> pairs(longley)
```

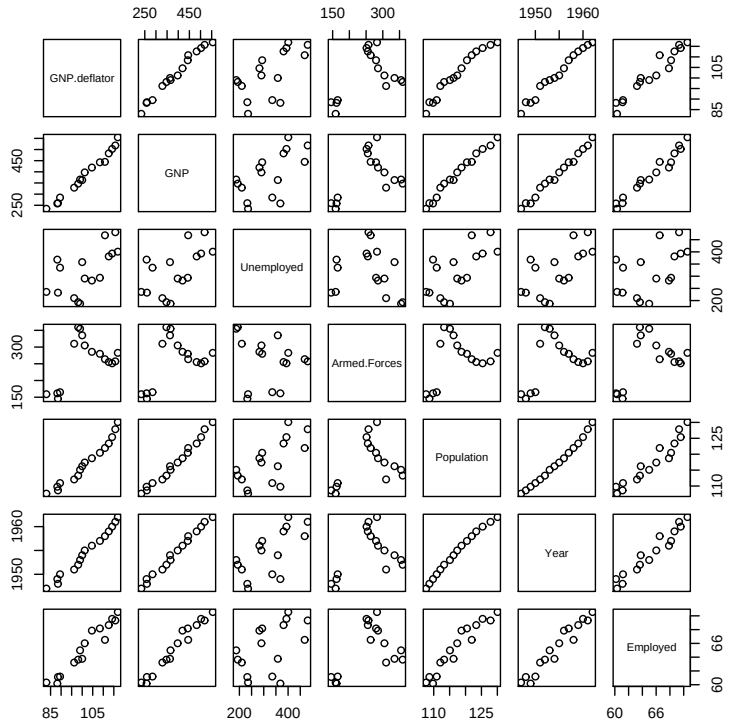


図 7: longley.x の多変量散布図

## 21.3 条件付きプロット

関数 coplot は層別の散布図を表示するためのものである。次の例は、quakes という名の R に付属のサンプルデータフレームに coplot を適用した例である (図 8)。

データフレーム quakes はフィジー近くの立方領域における 1964 年以降の 1000 件の地震（Richter マグニチュード 4.0 より大きいもの）の記録であり，緯度、経度、深さ、規模（Richter Magnitude)、観測所番号からなる。

```
> names(quakes)
[1] "lat"      "long"     "depth"    "mag"      "stations"
> coplot(lat ~ long | depth, data = quakes)
```

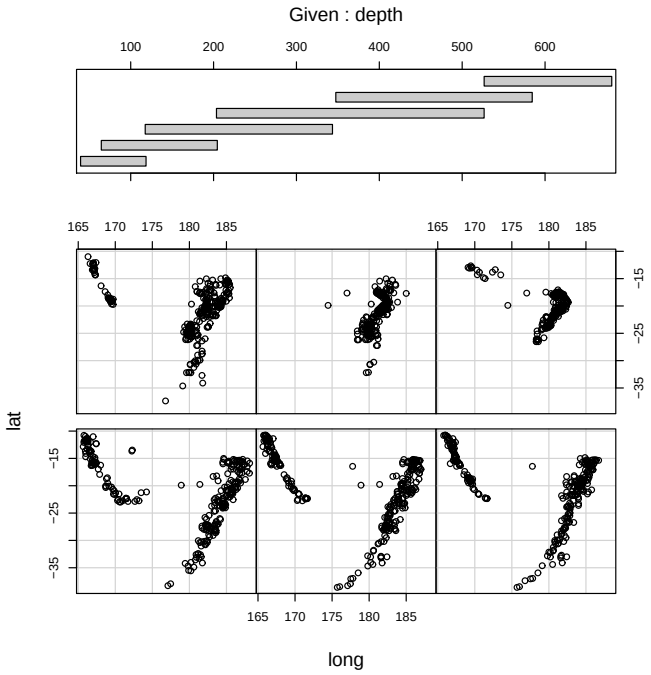


図 8: 条件付きプロット)

分割された画面の各部分は、深さについて層別した (サンプルを限定した) データの、緯度と経度を示す散布図である。サンプルがどの範囲に限定されているかは、画面の上端のグラフが示している。coplot の最初の引数lat ~ long | depth は、lat を縦軸に、long を横軸にした散布図を、depth について条件づけて表示することを意味する。

縦棒|の右に変数を一つだけ指定すると、条件は 1 つの変数についてのみとなる。このとき、グラフの表示順は、条件づける変数について一番小さな値の範囲にあたるものが、左下に描かれる。順に右側に進行し、その次は下から 2 段目に移り左側から順に割り当てられる。

図 9 は，次の指定によって層別に 2 つの変数を用いた。最初に引数である ll.dm は表示形式を指定するものである。縦棒の右側に指定してある depth*mag は、これら 2 つの変数を用いて層別を行うことを意味する。また、coplot 関数の 2 番目の引数は縦方向と横方向のパネルの個数を指定している。

```
> ll.dm <- lat ~ long | depth * mag
> coplot(ll.dm, number=c(4,3),data = quakes)
```

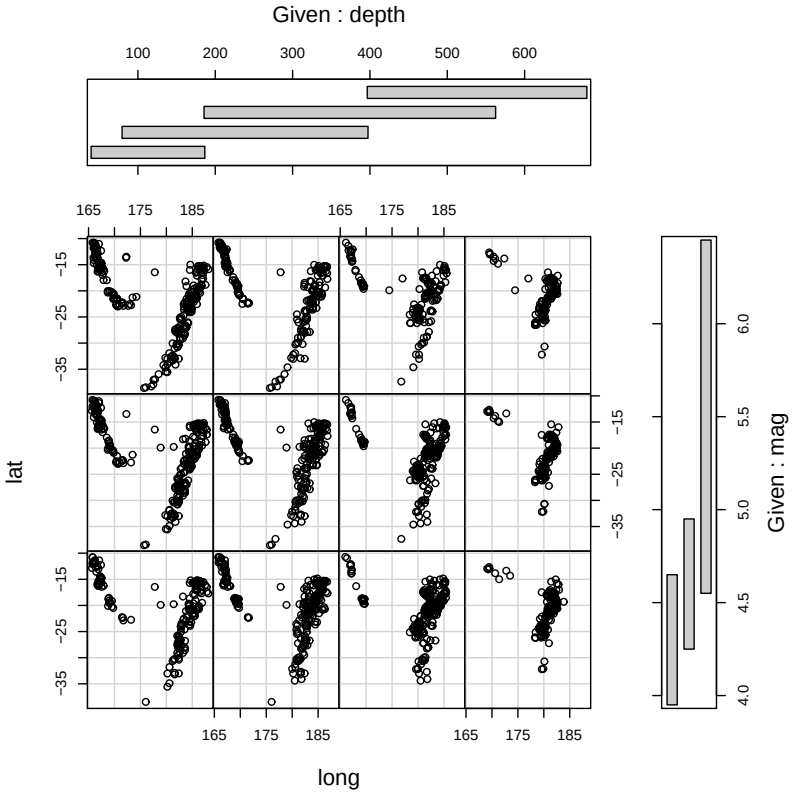


図 9: 条件付きプロット ( 2 変数による層別)

# 第Ⅴ部
# 文献案内

（この文献案内は改稿前の S-PLUS についての解説に対応している。最近では、R についての和書の解説がいくつか出版されており、使い方についての情報は容易に入手可能となった。詳細な情報は、R のマニュアルに直接あたるのが確実である。）

R を用いたデータ分析の解説書はすでに日本語の書籍が多く出版されており、検索可能である。線形モデル（共分散分析など）とその拡張についての R の利用にもとづく教科書としては [12] が詳しい。ロジスティック回帰や階層モデルを含む分析法について、最尤法とベイズ法による推定方法を解説している。話題は社会科学からとられている。ここで多用されている関数 `lmer`(混合モデルの推定関数）はパーッケージ `lme4` に含まれている（出版時からパッケージ名が変わっている)。

線形代数学の教科書は多く出版されている。ここでは標準的な教科書を 3 冊あげる [21],[22],[23]。(特に、[21] は簡潔で分かりやすい教科書。[23] はかなり専門的な内容を含んでいる。）また、線形代数の数値計算的側面については [17] をあげておく。原著は 1992 年に第 2 版が出版され、かなり内容が増えている。行列計算だけでなく、数値最適化、スペクトル解析、乱数生成などについての解説と、C 言語によるプログラム例がある。ただし、掲載されている線形計算のプログラムコードの頑健性 (ランク落ちへの対処など) は必ずしも高くない。

文献 [7] は S 言語が内部で利用している線形計算ソフトウェアの (専門的な) 利用解説書である。これの後継として開発されたものが [1] である。R は現在このルーチンを利用している。こららのプログラムは信頼性が高い。ソースコードは FORTRAN で記述されており公開されている。

文献 [2] は S-PLUS の原型になった S 言語の最新マニュアルである。旧版については翻訳 [3] があるが、拡張・変更された部分についての記述はない。

S-PLUS の入門書としては、[24],[8] がある。S および S-PLUS（および R) で利用可能な各種の統計分析法についての解説書としては、[4],[19] がある。S-PLUS/R の分析法の特徴ともいえる、データの平滑化と非線形な変換を利用する分析法についての教科書には、[11] がある。

また、米国カーネギー・メロン大学の統計学科の Mike Meyer 氏によって構築された StatLib アーカイブ `http://lib.stat.cmu.edu/`には S/R 言語で記述された各種の関数が登録されており、オンラインでこれらをコピーして利用することができる。StatLib には S, S-PLUS 用以外にも各種の統計分析用の関数が数多く集められている。

R のグラフィックスについては、開発者自身による解説 [15] が豊富な例を示している。

R については、`http://www.r-project.org` より、UNIX(Linux) 用、および Windows 用,MacOS 用のソフトウェアをダウンロードできる。また、Stalib および CRAN(The Comprehensive R Archive Network) と呼ばれるサービスサイトからダウンロードできる。国内では会津大学、東京大学、筑波大学がミラーサイトを提供している。

## 参考文献


[1] Anderson,E. et al. (1995). *LAPACK Users' Guide, 2nd. ed.*. Philadelphia: S.I.A.M.

[2] Becker,R.A.(1998) *The New S Language: A Programming Environment for Data Analysis and Graphics*, CRC.Pr.

[3] Becker, Chambers & Wilks (1988). (渋谷・柴田 訳, 1991), **S 言語 I,II**. 東京：共立出版.

[4] Chambers,J.M. & Hastie,T.J. eds. (1992).(柴田里程訳, 1994). **S と統計モデル:データ科学の新しい波**. 東京：共立出版.

[5] Cleveland,W.S. (1993). *Visualizing Data*. Murray Hill, New Jersey: AT&T Bell Laboratories.

[6] Cox,D.R. & Snell,E.J. (1981). *Applied Statistics: Principles and Examples*, London: Chapman& Hall. 医学統計研究会訳 (1985). 応用統計実践教本. 東京:MPC.

[7] Dongarra,J.J. et al. (1979). *LINPACK : Users' Guide*. Philadelphia: S.I.A.M.

[8] Everitt,B.S. (1994) *A Handbook of Statistical Analyses using S-PLUS* . London: Chapman and Hall.

[9] Frisby,J.P. & Clatworthy,J.L., (1975). Learning to see complex random-dot steregrams,*Perception*,**4**, 173-178.

[10] Gnanadesikan,R. (1997). *Method for Statistical Data Analysis of Multivariate Observations 2nd ed.*, Wiley. 邦訳 (初版) 丘本正・磯貝恭史訳. (1979) 統計的多変量データ解析, 東京:日科技連.

[11] Hastie,T.J. & Tibshirani, R.J. (1990). *Generalized Additive Models*, London : Chapman and Hall.

[12] Gelman,A. & Hill,J. (2007). *Data analysis using regression and multilevel/hierarchical models*, Cambridge University Press.

[13] Hoaglin,D.C., Mosteller,F. & Tukey,J.W. (1983). *Understanging Robust and Exploratory Data Analysis*. New Yourk: Wiley.

[14] Longley,J.W.(1967). An appraisal of least-squares programs from the point of view of the user,*Journal of the American Statistical Association*, **62**, 819-841.

[15] Murrell,P. (2005). *R graphics*, CRC Press.

[16] Nolan,D.A. & Speed,T. (2000). *Stat Labs: Mathematical statistics through application, Corr. 2nd ed*, Springer.

[17] Press,W.H. et al. (1988). (丹慶勝市他訳, 1993) *Numerical Recipes in C:***C 言語による数値計算のレシピ**. 東京：技術評論社.

[18] Ries,P.N. & Smith,H. (1963). The use of chi-square for preference testing in multidimensional problems. *Chem. Eng. Progress*,**59**, 39-43.

[19] Venables,W.N. & Ripley,B.D. (1999) *Modern Applied Statistics with S-Plus, 3rd ed.*, New York: Springer-Verlag.

[20] Venables,W.N. & Ripley,B.D. (2000) *S Programming*, New York: Springer-Verlag.

[21] 三宅敏恒 (1991). **入門線形代数学**. 東京:培風館.

[22] 齋藤正彦 (1966) **線型代数入門**. 東京:東京大学出版会.

[23] 佐武一郎 (1974) **線型代数学 (増補改題)**. 東京:裳華房.

[24] 渋谷政昭・柴田里程 (1992), **S によるデータ解析**. 東京：共立出版.

[25] Tukey,J.W. (1977) *Exploratory Data Analysis*. Reading,MA:Addison-Wesley.

## 謝辞

本稿のもとになったSplus用のテキストの作成にあたり、行廣隆次氏（現京都学園大学人間文化学部）の助力を得ました。

# 索 引