新・数理/工学ライブラリ［情報工学 ＝2］

Rプログラミングマニュアル[第２版]
—Rバージョン３対応—

間瀬　茂　著

数理工学社

第 2 版まえがき

> As for elegance, R is refined, tasteful, and beautiful. When I grow up, I want to marry R.
> – Andy Bunn, R-help (May 2005)
>
> It might surprise many R-help posters, but R has manuals as well ...
> – Uwe Ligges, R-help (January 2006)

プログラミングマニュアルの初版が出版された 2007 年当時，R のバージョンは 2.4, 2.5 であった．2013 年 4 月[*1]に R は記念すべきバージョン 3 台に到達した．この間の R の進歩，普及，成熟は真に目覚しかった．今や統計において R はほとんど唯一無二の地位を占めている．R 開発者の当初の目標であった「全ての統計的手法のワークベンチ」は間違いなく既に達成済みである．日本をとってみても，いわゆる R 本の数は訳本を含めて 100 を超えており，従来の統計本のスタイルを一新している．学生自身のパソコンを用いた本格的なデータと高度な統計手法を用いたデータ解析の手軽な実践という点で，統計教育も大きく変えている．何よりも統計の専門家がこぞって使い始めており，若い研究者が最初に使うシステムの地位を獲得している．

一般企業での利用も[*2]，オープンソースのシステムであることが逆に利用の足かせになっていた当初とは異なり，珍しくなくなっているようである．Google 等の巨大データを所持する企業は，むしろ R の積極的利用を後押ししている．

こうした R の現況は，R が常に手本としてきた S システムの革新性に加え，統計家とユーザが使いやすい統計システムを追求してきた結果であり，誰でも使用でき開発にも参加できるというオープンソースシステムならではの利点の賜物である．逆に，日進月歩の R をフォローするのは容易ではなく，著者もバージョン 3.0 の登場を機に現況の精査を試み，システムが大きく変貌していることに気づいた．多くの新機能の追加と，既存の機能の様々な改良・変更[*3]は想像以上であった．CRAN に登録されているアドオンパッケー

[*1] ニュージーランドのオークランド大学の統計の講義用ソフトとして産声を上げた R が，わずか千行程度のプログラムとしてささやかにネットに公開された 1993 年から数えて，丁度 20 年が経過している．

[*2] 公式 FAQ *R FAQ* ([9])) では R の商用，コンサルティング利用は自由と明記されている．R の既存コードを組み込んだ商品の販売については GNU GPL の観点から原則ソースコードの公開が義務づけられるが，REvolution R のように付加機能を加えた商用版も幾つか登場している．

[*3] R の基本パッケージ中には総計 1,264(内部関数は 488) 個のオブジェクトがある．CRAN から入手

ジ数も 2013 年後半に 5,000 を超え，CRAN 未登録分を含めれば 6,000 を超えると思われる．

主な変更点としては以下が挙げられる：

- **国際化の徹底**　中間栄治氏の R の日本語化作業は，その後 R の国際化につながり，今や R は世界の数多くの言語で使用可能になっており，R 自体にもこうした国際化を支援する機能が加わっている．フリーなシステムであることに加え，言語の壁を取り払うことにより，統計処理の南北問題が二重の意味で解決された．
- **巨大データ**　従来のベクトルの長さの「先天的制限」である $2^{31}-1$ の壁 (正整数の最大値) を超えることを可能にする “long vectors” 機構の登場．
- **速度向上**　手軽な速度向上の工夫．切り札として，バイトコードコンパイル機能と，マルチコア CPU を持つ計算機の普及を背景にした並列処理機能が R の推奨パッケージとして新たに加わった．実際 R およびパッケージ中の関数はほとんどバイトコンパイル済みである．
- **バイト (raw) 型データ**　バイト型データが標準で使えるようになり，それを支援する多くの関数が加わっている．
- **S4 化の推進**　完全なオブジェクト指向機構を備えた S4 (S 言語仕様第 4 版) オブジェクトのサポートが一層充実してきた．
- **開発・サポート・デバッグ体制**　R Foundation を中心とする，全世界の多数の統計専門家と熱狂的ユーザからなる支援体制は，進歩と全体的な安定との絶妙なバランス[*4]をもたらしている．
- **多数の洗練された有益なアドオンパッケージの登場**　これらは個人のみならず，コミュニティーで開発されているものもある．そうしたパッケージは R を側面から補強している．一部もしくは全体が R の推奨パッケージに取り入れられたものもある．
- **多くの関数・機能の追加**　既存の関数にも新しいオプションが追加され．対応ヘルプ文章および公式マニュアルも相当改訂されている．

こうした変更にも関わらず，従来のような使い方でも支障が起きることはほとんどなく，この本の初版は依然手引きとして有効である．しかしながら，初版の目標であった中級程度のユーザに R のプログラミング機能をできる限り網羅的に紹介する，という趣旨からは，こうした新しい R に触れないまま放置しておくことは許されない，というのが

できるバージョン 3.0.0 の基本，標準そして推奨パッケージのヘルプ文章を網羅した 3,604 頁に及ぶ PDF ファイル R-fullrefman.pdf がある．こうした目録が必要なほど R システムは巨大になっている．R に関しては「群盲象を撫でる」,「車輪を二度発明する」という故事成語が決して冗談ではない．

[*4] R システムの成功はある意味 Linux システムの成功に比肩できるが，開発システムの観点からは相当異なる．Core チームやパッケージ作者を含む R の開発者はピラミッド化されておらず，多分に水平的な関係しか持たず，専業開発者はいない．

今回改訂を志した理由である．

今回の第 2 版も原則 R のプログラミング機能[*5]に的を絞っている[*6]．初版の読者の中には，プログラミングマニュアルと称しながら記述がいわゆる計算機言語のそれとは大きく違うことに戸惑いを覚えた方がいらっしゃるようである．R プログラミングの理想は「できれば 1 命令，さもなければせいぜい 1 行で」ということにあり，実際それが十分可能なように設計されている．この点ではいわゆるスクリプト言語にあい通ずるものがある．統計解析は試行錯誤の連続からなり，いちいち長いコードを書いて思考中断しているわけにはいかないという，統計解析システム固有の事情が背景にある．更に，多くの統計ユーザにとってプログラミングの手間はなければないほど良いのであろう．

今回の改訂版の体裁と章立ては初版のそれを基本的に踏襲し，多くの章は変更点の追加にとどめているが，章の構成を変更したものもある．バイト (raw) 型データ，バイトコードコンパイラ，マルチコア CPU 利用の並列処理については特に新しい章を設ける価値があると考えた．初版が目指した何でもかんでも紹介するというスタイルも踏襲しているが，完璧を期すのは困難であり，また一般ユーザが使うことを想定していない関数も多く無駄でもある．内容のソースは第一に各オブジェクトのヘルプ文章と公式マニュアルである．ここに書けなかった内容も多い．是非，必要に応じて参照して欲しい．

初版では R でのプログラミングスタイルに不慣れな読者がまだ多いという視点から Tips を多く載せたが，これらの中には既に常識化しているものも多く，またインターネット上には従来 R の公式メーリングリストでしか得られなかった詳しい情報が集められたサイトが幾つも存在しているという現状から，最小限度に留めることとした．

この本の初版は初心者向きでないと序文で特に断ったにも関わらず，多くの R ユーザから類書のない参考書として高い評価を得てきた．今回の改訂版も同様の評価が得られれば幸いです．

2014 年 3 月

間瀬　茂

[*5] 現在日本では「R 言語」という呼び方が定着しつつあるが，著者は二重の意味でこれは好ましくないと考えている．何よりも R は統計システム・環境であり，言語機能は即統計解析機能ではなく，単に計算機言語と捉えるのは矮小化していると思うからである．更に R の言語機能は S 言語のフリーな移植 (エンジン) であり，開発者達自身が R の言語機能は S 言語であると明記している．例えば，公式マニュアル *An Introduction to R* ([1]) には “a well developed, simple and effective programming language (called ‘S’) which includes conditionals, loops, user defined recursive functions and input and output facilities. (Indeed most of the system supplied functions are themselves written in the S language.)”，また公式 FAQ 集 *R FAQ* ([9]) には “We can regard S as a language with three current implementations or “engines”, the “old S engine” (S version 3; S-PLUS 3.x and 4.x), the “new S engine” (S version 4; S-PLUS 5.x and above), and R. Given this understanding, asking for “the differences between R and S” really amounts to asking for the specifics of the R implementation of the S language, i.e., the difference between the R and S engines.” と書かれている．

[*6] 統計機能については [20] 参照．

初版まえがき

 は統計解析言語・システムであり，統計解析用に開発された S 言語・環境のフリーな移植である．現在の R は，S-PLUS 等の S 言語の商用版と比較しても遜色のない水準に達している．遠からず R は統計的手法の共通インフラの地位を占めることは間違いない．実際，日本でも既に R を前提とした出版が 20 冊を越えようとしており，全世界でもブームといえるほど R 使用を前提とした出版が相次いでいる．

このマニュアルの目的は，R の計算機言語としての豊富な機能を，参考コードとその出力例を逐一加えて，紹介することである．全くの初心者というよりは，R を一通り使いこなせる方を読者として想定している．構成は諸機能の索引になることを目指しており，決して系統的ではない．R は通常の計算機言語では考えられないほど多様な関数を最初から備えている．この特徴は，R の本来の機能である対話的統計データ解析処理を容易にするためであり，簡潔なコードを書くことを可能にし，プログラム開発効率が極めて高い．逆に，一般ユーザにはその全貌を把握するのは至難の技で，しばしば「車輪を 2 度発明する」無駄を強いられることになる．著者の経験では，こうした機能があれば便利なのにと感じたことが，実際は既に存在していたことが頻繁にある．また，これまで気づかなかった便利な関数・機能を発見して驚くことが今でも多い．

このマニュアルの内容，特に Tips は，著者を含むしばしば匿名の R ユーザが，R 情報サイト RjpWiki に投稿した記事を基礎材料としている (一部勝手に引用・改編していることをお許し願いたい．無論，最終的な内容の正しさの責任は著者にある)．その意味で，「The R Book」([14]) 同様 RjpWiki の産物といえる．この場を借りて，管理者の岡田昌史氏を含め，RjpWiki を支えている質問者を含む皆さんにお礼を申し上げたい．これらは更に，全世界のユーザと R 開発チームによるメーリングリスト記事・マニュアル・書籍を参考にしていることも多い．R という素晴らしいシステムを作り，改良発展されている R 開発チームとともに，それを陰に陽に支えている全世界の R ユーザにも同時にお礼を申し上げたい．また，著者の研究室の修士院生の千田敏君からは，原稿に対する多くの意見と訂正の指摘を頂いたことを記して感謝します．

2007 年 9 月

間瀬　茂

目　　次

第1章
序

 はかつてのベル研究所で開発されたS言語を基本とした対話的統計解析環境・システムである．単なる計算機言語システムではなく，対話的統計解析を効率的に行えるように，過剰とも思える豊富なシステム関数，様々な統計データの保存・編集機能や高品位のグラフィックス機能を持つ．また，豊富な組み込みデータセットを備えており，各関数毎に詳細なヘルプ文章と例示用コードを備えている．

1.1 このマニュアルについて

R は本体だけを取っても，全部で千数百あまりの関数・データセットおよび関連ドキュメント・コードの巨大な集合体である．どのような関数・データセットがあるか，それらの関連は容易には窺い知ることができない．関数は，実際のデータと解析目的の多様性に応じ，多数のオプション引数を持ち，同じ関数から微妙に，場合によれば全く異なった結果を得ることができる．R はその膨大な関数と本格的な組み込みデータセット毎に，詳細なヘルプ文章，更に例示用コードを持つ．R の使用に熟達する王道は，これらのヘルプ文章を繰り返し読み，参考コードを吟味[*1]することであろう．しかしながら，全て英語で書かれたヘルプ文章および例示コードの理解には，語学的な障壁に加え，広範囲に渡る統計手法と，計算機言語としての R の機構に関するかなりの知識を既に持っていることが前提となる．

著者は R の 1 ユーザであり，計算機言語一般にも必ずしも詳しくはない．また，R が持つ広範な機能の実際の実装方法にも決して詳しくない．結果として多くの誤解が含まれている可能性がある．日本語訳語だけを取っても，必ずしも各分野で慣用的なもの (それら自体が混乱していることは別としても) になっているとは限らないであろう．著者は，こうした欠陥を持つにせよ，このマニュアルが R のプログラミング機能を一般ユーザが理解する助けになると期待し，また確信している．

[*1] 実際，RjpWiki に寄せられる質問の過半数は，関連ヘルプ文章を良く読めば答えが得られることを強調しておきたい．

第 2 章

R オブジェクトのタイプ・保管モード・属性

 の全てのデータ型の基本は原子的 (atomic) なオブジェクトである．全てのオブジェクトは内部的表現法を表す保管モードを持ち，更に多くはその素性を表す属性を持つ．こうしたオブジェクトに対しては，そのタイプ・モード・属性を検査・取り出し・設定するための関数ファミリがある．

2.1 原子的なオブジェクトと再帰的なオブジェクト

R の全てのデータ型の基本は原子的なオブジェクトである．原子的なデータ型は，論理値 (logical)，整数 (integer)，倍精度実数 (numeric, double)，複素数 (complex)，文字列 (character)，バイト (raw)，そして NA (Not Available)，Inf，NaN と NULL である．それぞれの原子型については後の章で解説する．

原子的なデータ型に対し，リストやデータフレーム等は再帰的 (recursive) なデータ型と呼ばれる．R オブジェクトは原子的であるか再帰的であるかのいずれかである．R オブジェクトが原子的かどうかは関数 **is.atomic** で判定できる．原子的な要素を持つベクトル (従って行列・配列) も原子的なデータ型とされる．特殊値 NA, NaN, Inf, NULL も原子的とされる．R オブジェクトが再帰的かどうかは関数 **is.recursive** で判定できる．関数やモデル式等も再帰的オブジェクトとされる．

```
# 整数・実数・文字列ベクトルは原子的
> is.atomic(1:5,sqrt(1:5),c("a","b"))
[1]  TRUE TRUE TRUE
# 行列は原子的
> is.atomic(matrix(1:4,2,2))
[1]  TRUE
# 組み込み関数は再帰的
> is.recursive(sin)
[1] TRUE
# リストは再帰的
> is.recursive(list(1:3,runif(3)))
[1]  TRUE
# モデル式は再帰的
> is.recurxsive(y~x)
[1] TRUE
# 私製関数は再帰的
> is.recursive(function(x) x)
[1] TRUE
# 表現式は再帰的
> is.recuersive(expression(x+1))
[1] TRUE
```

第 3 章
整　　数

3.1　整数型オブジェクト


の原子オブジェクトの基本は整数 (integer) である．R では整数値は必要に応じて倍精度実数値に強制変換されるため，特に整数であることを意識する局面は少ない．コンソール表示においても両者の区別は困難である[*1]．但し，ベクトルの添字は整数でなければならず，また整数値ベクトルのサイズは対応する実数値ベクトルよりも保存サイズが少なくなる．1:3 等で作られる数列は整数値ベクトルになる．特に整数であることを指定するには表現 1L のように記号 L をつける．

3.1.1　整数の判定 is.integer

整数かどうかの判定には **is.integer(x)** 関数を用いる．x が全て整数からなるときのみ TRUE を返す．

```
# 整数値 1L
> is.integer(1L)
[1] TRUE
# 単なる 1 は実数 1.0
> is.integer(1)
[1] FALSE
> identical(1,1.0)
[1] TRUE
> identical(1L,1)
[1] FALSE
> is.integer(1:10)
[1] TRUE
# 整数値 1L
> is.integer(1L)
[1] TRUE
# 1L に強制変換
> is.integer(as.integer(1))
[1] TRUE
```

実数が実質整数値であるかどうかを判定するには以下のような関数を用いる．

```
> is.Integer <- function(x)
    abs(x-round(x))
      < .Machine$double.eps^0.5
> is.Integer(1)
[1] TRUE
> is.Integer(seq(1,3,by=1/2))
[1] TRUE FALSE TRUE FALSE TRUE
```

[*1] 因子も見かけ上整数として表示されることがあるので紛らわしい．56 頁参照．

第 4 章
倍精度実数と浮動小数点数演算

 には倍精度実数 (double)[*1]を表す原子オブジェクトがある．整数は文脈に応じて倍精度実数に変換される．倍精度実数を引数に取る数値計算関数については第 18 章 (196 頁) にまとめて紹介する．

4.1 実数値 (numeric, double)

4.1.1 実数への変換と判定 is.numeric, is.double, as.numeric, as.double

オブジェクトが実数型[*2]かどうかは関数 **is.double** で判定できる．関数 **as.numeric** と **as.bouble** は数値として解釈できるものを倍精度実数化する

```
# 整数値も Inf も numeric
> is.numeric(c(1,1L,Inf))
[1] TRUE
# 数値と解釈できないものは NA とされ警告が出る
> as.numeric(c(1,1L,"a",NA,NaN,Inf))
[1]   1   1  NA  NA NaN Inf
警告メッセージ:
強制変換により NA が生成されました
# 整数値も倍精度実数化される
> is.integer(as.numeric(1L))
[1] FALSE
# as.numeric と同じ
> as.double(c(1,1L,"a",NA,NaN,Inf))
[1] 1 1 NA NA NaN Inf
警告メッセージ:
強制変換により NA が生成されました
# 整数値も倍精度実数化される
> is.integer(as.numeric(1L))
[1] FALSE
警告メッセージ:
強制変換により NA が生成されました
# 長さ 3 の実数の零ベクトル
> double(3)
[1] 0 0 0
> x <-factor(sample((1:10)/10,200,
            replace=TRUE))
# 因子は見かけ上整数値ベクトルだが...
> str(x)
 Factor w/ 10 levels "0.1","0.2","0.3",..:
  9 4 7 3 4 1 6 4 1 6 ...
# 実際は数値とは見なされない
> is.numeric(x)
[1] FALSE
# 整数値も Inf も numeric
> is.numeric(c(1,1L,Inf))
[1] TRUE
```

[*1] 倍精度実数は実際は 10 進法で 15，16 桁の表現を持つが，コンソールへは既定で 7 桁だけが表示される．

[*2] 歴史的理由から R では用語 numeric, double, real が混在している．判定関数 **is.numeric** は整数値もしくは倍精度実数値かどうかの判定を行い，倍精度実数かどうかの判定を行う関数 **is.double** とは異なる．

第5章
複　素　数

 の複素数 (complex) は原子オブジェクトである．複素数は 2 つの倍精度実数の対で表現される．複素数に関する基本的演算とその三角関数や指数関数が提供されている．

5.1　複素数の生成

5.1.1　複素数表現 x+yi

1 つの複素数は 1+2i, 0+2i, 1+0i のような表現を持つ[*1]．

```
# 実数の 1
> str(1)
 num 1
# 複素数の 1
> str(1+0i)
 cplx 1+0i
# 実数 1 と複素数 1+0i
# は等しいとされる
> 1 == 1+0i
[1] TRUE
# 厳密な判断では異なる
# オブジェクト
```

```
> identical(1,1+0i)
[1] FALSE
# これは正しいが
> 2i
[1] 0+2i
# 次はエラー.
# (-2)*1iか-2iとする
> (-2)i
エラー: 予想外のシンボル
ルです in "(-2)i"
# 虚部が NA の複素数の
# つもり
```

```
> 1+NAi
エ ラ ー: オ ブ ジェク
ト 'NAi' がありません
# NA*(1i) とすれば良い
> NA*(1i)
[1] NA
# もしくは
> complex(1,1,NA)
[1] NA
# NA_complex_ である
> str(
    complex(1,NA,1))
```

```
 cplx NA
> is.na(
    complex(1,NA,1))
[1] TRUE
> complex(1,NaN,1)
[1] NaN+1i
> complex(1,1,NaN)
[1] 1+NaNi
> complex(1,Inf,1)
[1] Inf+1i
> complex(1,1,-Inf)
[1] 1-Infi
```

5.1.2　複素数ベクトルの生成 complex

関数 complex は複素数ベクトル生成関数である．

```
# 実部・虚部ベクトルを与えて
# 複素数ベクトルを作る
> complex(2,1:3,2:4)
[1] 1+2i 2+3i
```

```
# 絶対値・角度ベクトルを与える
> complex(2,mod=1:2,arg=pi*(2:3)/4)
[1] 0.000000+1.000000i -1.414214+1.414214i
```

[*1] 実数値変数 x に対し xi, x*i, x*(i) という表現はできない．x*(1i) または complex(1,0,x) とする必要がある．1 と 1+0i, 0 と 0i は R オブジェクトとしては異なる．実部と虚部のどちらか，もしくは双方が NA, NaN, Inf, -Inf であっても良い．但し直接 NAi, Infi のように記述することはできない．

第 6 章
文字列とその操作


の文字列 (character)[*1]は原子オブジェクトである．文字列 (ベクトル) はデータとして登場するばかりではなく，プログラムの一部としても頻繁に登場する．R には文字列を操作する豊富な関数ファミリが用意されている．

6.1 原子オブジェクト文字列 (character)

文字列は 2 重引用符で挟んで"abc" のように表される[*2]．1 重引用符 (右引用符) ’ で挟んで ’abc’ としても良い．左引用符 ‘ は使えない．2 重引用符自体を含む文字列を作るには 1 重引用符で囲めば良い．\" としても良い．

```
# "abc"と’abc’ は
# 同じオブジェクト
> identical("abc",’abc’)
[1] TRUE
# 2 重引用符自体を含む文字列
# 2 重引用符は\"で表示
> ’"ab"c"’
[1] "\"ab\"c\""
> is.character(’"ab"c"’)
[1] TRUE
> identical(’"ab"c"’,
            "\"ab\"c\"")
[1] TRUE
# 逆も真である
> is.character("’abc’")
[1] TRUE
# 一重引用符は
# そのまま表示される
> "’abc’"
[1] "’abc’"
```

6.1.1 文字列の生成と検査 character, is.character

関数 character は空文字列 "" からなる指定された長さの空文字列ベクトルを作る．関数 is.character は引数が文字列 (ベクトル) であるかどうかを検査する．

```
# 長さ 4 の空文字列ベクトル
> character(4)
[1] "" "" "" ""
# 空文字列
> character(0)
character(0)
> str(character(0))
chr(0)
> is.character(NA)
[1] FALSE
```

[*1] 定数値文字列というニュアンスのストリング (string) という呼び方もある．また，出力やコード上では文字列と区別が難しい名前 (name) という概念がある．これは名前ラベル・引数ラベル・オブジェクト名等に使われるもので，文字列と区別される．

[*2] データフレームの文字列変数はコンソールへの出力では 2 重引用符なしで表示される．

第 7 章
論理値と条件判断


でも条件判断はプログラム言語としての中核機能であるが，R らしいプログラミングスタイルはむしろ if 文等の条件判断文を表に出さず，論理値ベクトルを仲介とするベクトル操作や条件判断を内包する関数 (探せば結構ある) を用いて行うことにある．

7.1 論理値 TRUE, FALSE

論理値 TRUE, FALSE は R の原子オブジェクトである．NA 値も論理オブジェクトオブジェクトであり，論理値ベクトル中に含まれる可能性がある．

```
# 論理値は原子オブジェクト
> is.atomic(TRUE)
[1] TRUE
# モードは logical
> mode(TRUE)
[1] "logical"
# 型も logical
> typeof(TRUE)
[1] "logical"
# 32 ビットで表現
> object.size(TRUE)
[1] 32
```

7.1.1 論理値への変換と検査 as.logical, is.logical

数値 x は論理値が必要とされる文脈では，関数 as.logical を用いて論理値 TRUE, FALSE に変換される．実際には整数と実数の 0 が FALSE に，それ以外の数値は正負の無限大を含み TRUE に変換される．数値以外では文字列 "T", "TRUE", "True", "true" は TRUE に変換され，文字列 "F", "FALSE", "False", "false" は FALSE に変換される．それ以外は全て NA に変換される．関数 is.logical は論理値かどうか検査する．

```
# 数値 0 は FALSE に，1 は TRUE に変換される
> as.logical(c(0L,0,0.0,1L,1,1.0))
[1] FALSE FALSE FALSE TRUE TRUE TRUE
# NA は NA に変換される
> as.logical(c(2.3,-10,NA,NULL,Inf,-Inf))
[1] TRUE TRUE   NA TRUE TRUE
# TRUE に変換される 4 種類の文字列
> as.logical(c("T","TRUE","True","true"))
[1] TRUE TRUE TRUE TRUE
# FALSE に変換される 4 種類の文字列
> as.logical(
   c("F","FALSE","False","false"))
[1] FALSE FALSE FALSE FALSE
# NA 値はなぜか論理値とされる
> is.logical(NA)
[1] TRUE
# 整数としての論理値
> c(TRUE,FALSE) * pi
[1] 3.141593 0.000000
```

第8章
因　　子

 の因子 (factor) は整数値ベクトルの一種類と考えられるが，その真の値は対応する水準ベクトルにより間接的に表現される．同じ値を持つものをグループ化し 1 つの水準値で代表させることにより，操作が簡単になり，必要メモリも減らせる．データフレームの文字列変数は原則因子として扱われ，グループ化される．因子は統計モデル関数で特に重要[*1]になる．

8.1　因　　子

8.1.1　ベクトルを因子にする factor

`factor` はベクトルを因子にする．S との互換性のために同値な関数 `ordered` がある．データフレーム等を操作する関数の内部で，変数を因子化するのに使われる．検査・変換関数 `is.factor`, `is.ordered`, `as.factor` がある．もし `ordered=TRUE` ならば因子水準は順序づけられていると見なされる．これはクラス属性だけの違いであるが，モデル当てはめ関数等では全く異なった扱いがされる．基本的には，もし `x[i]` が `levels[j]` に等しければ `x[i]` の水準は `j` とされる．`exclude` で与えられた値は水準に含められない．水準値集合 `levels` は既定でデータ値から決まるが，もし独自に与えた `levels` 中にない値は `NA` とされる．数値 `x` に対し `exclude=NULL` とすると `NA` 値は特殊水準 `"NA"` とされ，水準の最後の値とされる．返り値はクラス属性 `"factor"` を持つオブジェクトで，属性 `"levels"`，モード `character` を持つ `x` と同じ長さの整数値コードの集合である．順序づけられた場合はクラス属性 `c("ordered","factor")` を持つ．

因子の実際の解釈はコードと水準集合双方に依存する．水準集合が同じ 2 つの因子の比較には `as.numeric(levels(f))[f]` で数値ベクトル化することが勧められる．因子水準は既定でソートされるが，ソートはロケールに依存する．例え少しでも同じ値が頻繁に繰り返されている文字列データは，因子として扱えば必要メモリ量が減る．32 ビット機では `n` バイトの文字列の保存には `28+8*ceiling((n+1)/8)` バイトが必要だが，そうした

[*1] `help(contrasts)` 参照.

第 9 章
バイト型データとビット操作


に比較的新しくつけ加わったデータ型がバイト (**raw**) 型[*1]である．計算機で処理されるデータは基本的にバイト列で表現されている．そのうちアスキー文字等可視化できるものだけからなり，テキストとして意味を持つものは限られる．ファイルも一般にバイト列として扱うことになる．

9.1 バイト (raw) 型オブジェクト

1 つのバイト型データは 16 進数 (`0,1,2,...,9,a,b,c,d,e,f`) の対[*2]で表現される．一方でバイト型データのビット列 (2 進法) 表現があり，`00,01` (それぞれ 2 進法の 0,1 を表す) の列で表現される．1 つのバイト型データは長さ 32 のビット列で表現される．

```
# 255 は 16 進法では ff
> x <- as.raw(255); x
[1] ff
```

```
> str(x)
 raw ff
```

9.1.1 バイト列を作る・変換する・検査する raw, as.raw, is.raw

関数 **raw** は指定された長さの 0 バイト `00` からなるベクトルを作る．**is.raw** はバイト列かどうかを検査する．**as.raw** は整数をバイト列に変換する．

```
# 長さ 2 の 0 バイトベクトル
> xx <- raw(2); xx
[1] 00 00
# 40 のバイト (16 進法) 表現
> xx[1] <- as.raw(40)
```

```
# 文字列"A"のバイト表現
> xx[2] <- charToRaw("A")
# 16 進数の対の並びで表現
> xx
[1] 28 41
```

[*1] そのまま RAW 型と書いたり，ロー型と訳されている例があるが，ここでは意訳してバイト型と訳す．

[*2] 21 頁参照．

第 10 章
特殊オブジェクト NA, NULL

に存在する特殊オブジェクト NA, NULL についてまとめておく[*1]．NA は「不明だが何かある」ことを意味するが，NULL は「そもそも何もない」ことを意味する．これらは便利でもあるが，反面 R の操作を困難にする主要な原因の 1 つになっている．特に一連のプログラム実行中に登場する際には，エラーに悩まされることもある．

10.1 NA 値

NA(Not Available) は欠損値を表現する原子 R オブジェクトである．オブジェクトの初期化の際は，各要素を良くある 0 ではなく NA と置くと後で確認しやすい．

```
# 行列の要素を NA 値で初期化
> matrix(NA,2,2)
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
```

10.1.1 NA 値は論理型，様々な NA 値

NA 値は，同時に論理型を持つ[*2]．R 2.5.0 より各種原子型オブジェクトの欠損値を表す専用オブジェクト，NA_integer_, NA_real_, NA_character_, NA_complex_ が導入された．それ以前は全ての型に対応する NA と as.character(NA) で得られる文字列型の欠損値だけが存在した．いずれも is.na 関数では TRUE となるが，NA との identical 関数による厳密な一致比較では FALSE となる．

```
# NA は論理型を持つ
> typeof(NA)
[1] "logical"
```

```
# モードは "logical"
> mode(NA)
[1] "logical"
```

```
# 保管モードは "logical"
> storage.mode(NA)
[1] "logical"
```

[*1] 浮動少数点数演算に関係する特殊オブジェクトの NaN (非数，Not a Number) と Inf (無限，Infinity, Infimum) については第 4 章 (20 頁) で扱う

[*2] 比較演算等では，比較不能な場合が生じると NA 値を返すことが多い．NA 値自体を論理値として扱うことで，全体としての演算を取りあえずエラーにしないようにできる．

第 11 章
繰り返しと条件判断

 は多くの計算機言語と同じような制御命令のセットを持つが，より多様で柔軟[*1]である．

11.1 繰り返し

11.1.1 範囲に渡って繰り返す for ループ

for(arg in range) expr 文[*2]はループ範囲 range の各要素 arg に対して式 expr を実行する．for と ( ) の間には空白を置いても置かなくても良い．R ではループ範囲 range にベクトル・行列・リスト・データフレーム・因子が可能である．行列・配列は実際はベクトル as.vector(range) としてアクセスされる．空の範囲では一度も実行されない．

```
# ループ範囲にベクトルを
# 取る
> for (i in 1:3) cat(i,"\n")
1
2
3
# ループ範囲に文字ベクトルを
# 取る
> for(i in c("a","b","1"))
    cat(i,"\n")
a
b
1
# 行列ループ範囲 (実際は
# ベクトルとしてアクセス)
> X <- matrix(1:4,c(2,2))
> for(i in X) cat(i,"\n")
1
2
3
4
# リストループ範囲
> X <- list("a",1:4,3)
> for (i in X) cat(i,"\n")
a
1 2 3 4
3
# ループ範囲に
# データフレームを取る
> x <- data.frame(
    a=3:5,b=letters[1:3],
    c=LETTERS[7:9])
> x
  a b c
1 3 a G
2 4 b H
3 5 c I
# 文字列は因子に変換
> for(i in x)
    cat(i,"\n")
3 4 5
1 2 3
1 2 3
> Y <- list(c=rnorm(5),
        d=letters[1:3])
# 入れ子リスト
> X <- list(a=1:3,b=X)
```

[*1] 繰り返しや条件判断はプログラミング言語の中核を占める制御構造であり，R もいわゆる Algol 系言語の制御文を全て持つ．しかしながら，インタプリタ言語である R では，そうした制御文の使用は目立って実行速度を遅くしたり，コードが複雑になりやすい．R らしいコードは，むしろそれらを (あからさまには) 使わないことにある．

[*2] ループ処理は一般に実行速度を遅くするボトルネックになりやすい．またコードが長くなりがちである．apply 関数ファミリ (第 19 章) の使用や，ベクトル・行列・配列・データフレーム用に用意されている各種専用高速関数の使用を考えると良い．

第 12 章
ベクトル

のデータ構造ではベクトル (vector) は，リストと並んで最も基本的なもので，ベクトルを引数に取り，返り値もベクトルとなる膨大な関数ファミリ[*1]がある．ベクトルは，数値・文字列・バイト値等，同じ型を持つ要素を 1 次元的に並べたもので，各要素は括弧演算子 `[]` で取り出すことができる．単一の数値・文字列・論理値も長さ 1 のベクトルである．

12.1 ベクトルを作る

12.1.1 要素を結合してベクトルを作る c

ベクトルを作る基本関数は `c`[*2]で，任意個の引数を取り，それを連結したベクトルを返す．引数の型が異なるときは，全てを表現できる型に強制変換される．

```
# 数値とベクトルを結合
> c(1,2,runif(1))
[1] 1.0000 2.0000 0.8775
# 文字列ベクトル
> c("A","B","CDE")
[1] "A"  "B"  "CDE"
# 数値と文字列が混在すると
# 文字列ベクトルになる
> c(1:3,"A")
[1] "1" "2" "3" "A"
```

`recursive=TRUE` オプションはリストの成分を再帰的に並べたベクトルを作る．

```
> c(list(c(1,2),7),rec=TRUE)
[1] 1 2 7
# 名前ラベルを継承する
> c(list(A=c(B=1,C=2),B=c(E=7)),rec=TRUE)
A.B A.C B.E
  1   2   7
```

12.1.2 規則的なベクトルを作る numeric

関数 `numeric(n)` は 0 が `n` 個並んだ数値ベクトルを作る．同等な関数として `integer`, `character`, `logical` 等がある．初期値等を作るときに便利である．

[*1] 行列と配列は次元属性を持つベクトルに他ならず，行列・配列用の関数も適用できる可能性がある．
[*2] 結合 (combine, concatinate) 関数．

第 13 章
行　　列

 の行列[*1]は行列次元属性 (および次元名属性) を持つベクトルもしくはリスト[*2]であり，行列として添字操作ができる．従って R における行列の概念は，単なる数値行列に留まらず，文字列ベクトルやリストも行列化できる．

13.1　行列の生成・検査，行列への変換

13.1.1　1 つのベクトル・リストから行列を作る matrix

関数 **matrix** は行列を生成する R の基本関数であり，要素であるベクトル data と，それぞれ行数，列数を表す 2 つの正整数値 ncol, nrow を引数に取る．byrow=TRUE ならばベクトル要素は列順に，byrow=FALSE (既定) ならば行順に埋められる．dimnames 引数は次元名属性を与える長さ 2 のリストで，それぞれ行名と列名を与える．もし引数 nrow か ncol のいずれかが与えられないと，data の長さ等からもう一方が推測される．もし data の要素数が行列全体を埋めるだけないときは data の要素がリサイクル規則が適用される．もし data の長さが 0 ならば，その型に応じ NA や NULL で埋められる．

```
# 列順で行列にする
> matrix(1:8,nrow=2,ncol=4)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
# 行順で行列にする
> matrix(1:8 nrow=2,ncol=4,
          byrow=TRUE)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
# ベクトル 1:5 のリサイクル
# 使用．警告が出る
> matrix(1:5,nrow=2,ncol=4)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    2
[2,]    2    4    1    3
Warning message:
行列のデータ長 [5] が行数 [2] を
整数で割った，もしくは掛けた値
ではありません
```

次元が 0 の行列も構文的には許される．

[*1] 注意：R プログラミングのコツの 1 つは，ベクトル・行列・配列に対する操作を，その要素をいちいち取り出すループ操作・条件判断で行わないことにある．これはコードを冗長にし，しばしば極度の低速化を招く．専用の高速内部関数を用いベクトル・行列・配列そのものを操作するべきである．ベクトル・行列・配列を論理値添字で一括して操作することも R らしいスタイルである．

[*2] 実体はベクトルもしくはリストに過ぎないことは注意に値する．コンソールへの表示や各種操作では，この属性を参考に “行列風” に操作するだけである．データフレームは外見上行列と似ているが，両者は厳密に区別する必要がある．データフレームはリストであり，一見行列風に表示されるが，数値と文字列等が混在可能である．一方行列の要素は全て同じ型でなければならない (リストの行列化を除く)．

第 14 章
配　　列

 の配列 (array) は次元属性を持つ R オブジェクトであり，特別な場合として行列を含む．配列の操作は，多くの点で行列の操作に似ている．この章では，特に次元数が 3 以上の狭義の配列に関する関数を紹介する．

14.1　配列の生成と操作

14.1.1　配列の生成 array

関数 **array(data=NA, dim=length(data), dimnames=NULL)** はベクトルもしくはリスト data から配列を生成する．次元属性 dim は長さ 1 以上の正整数値ベクトルである．各次元の名前ラベルである次元名属性 dimnames を加えることができる．data は列順 (より左の添字が最も早く変化する) で埋められ，要素数が不足するならリサイクル規則が適用され，もし余分ならば切り捨てられる．1 次元配列はベクトルとは区別され，関数によっては異なった扱いを受ける可能性がある．

行列に関する操作・注意の多くが配列に対しても有効である．

```
> array()
[1] NA
> array(0)
[1] 0
# dim=c(1) の 1 次元配列
# ベクトルではない
> str(array(0))
 num [1(1d)] 0
# dim=c(1,3) とされる
> array(0,3)
[1] 0 0 0
# 2 次元配列 matrix(0,3,2) で
# ある
> array(0,c(3,2))
     [,1] [,2]
[1,]    0    0
[2,]    0    0
[3,]    0    0
# 3 次元配列
> array(0,c(2,2,2))
, , 1
     [,1] [,2]
[1,]    0    0
[2,]    0    0
, , 2
     [,1] [,2]
[1,]    0    0
[2,]    0    0
> x <- array(1:12,c(2,3,2))
# 次元は長さ 3
> dim(x)
[1] 2 3 2
# 総要素数
> length(x)
[1] 12
```

要素が文字列の配列．

```
# "y","z"は切り捨てられる
> array(letters[1:24],
        c(2,3,4))
, , 1
     [,1] [,2] [,3]
[1,] "a"  "c"  "e"
[2,] "b"  "d"  "f"
--途中省略--
, , 4
     [,1] [,2] [,3]
[1,] "s"  "u"  "w"
[2,] "t"  "v"  "x"
```

第 15 章
リ ス ト

 のリスト (list)[*1]は任意の相異なる型の R オブジェクトを一括して保存できる便利なデータ構造である.

15.1 リストを生成する

15.1.1 リストを生成する list, as.list, is.list

リストを作る基本関数は **list** であり，任意個数の R オブジェクトを引数に取る．各成分には名前ラベルを name=value の形で指定できる．リストの成分にはリスト自体も可能である．リスト x の各成分には，2 重括弧演算子と添字番号を用いて x[[i]] とするか，名前つき成分なら$ 演算子とその名前 (を表す文字列) を用いて x$"name" もしくは 2 重鈎括弧演算子を用いて x[["name"]] とする．検査・変換関数 is.list, as.list がある.

```
# 空リスト
> list(NULL)
NULL
# NA を用いた初期化
> list(NA)
[[1]]
[1] NA
# 0 を単一成分とする
# リスト
> list(0)
[[1]]
[1] 0
# 2 つの空成分を持つ
# リスト
> list(NULL,NULL)
[[1]]
NULL
[[2]]
NULL
# 2 つの空成分を持つ
# リスト
> vector("list",2)
[[1]]
NULL
[[2]]
NULL
# 名前のない成分を持つ
# リスト
> x <- list(1,2)); x
[[1]]
[1] 1
[[2]]
[1] 2
# 第 1 成分
> x[[1]]
[1] 1
# 数値と文字列のリスト
> x <- list(1,"2")
> x
[[1]]
[1] 1
[[2]]
[1] "2"
```

名前つき・なし成分を持つリスト.

```
> x <- list(a=1:3,b="abc",
            NA); x
$a
[1] 1 2 3
$b
[1] "abc"
[[3]]
[1] NA
# x[[1]],x[["a"]] でも良い
> x$a
[1] 1 2 3
> x[["b"]]
[1] "abc"
```

[*1] R には総称的ベクトルであるリストと，言語オブジェクト中で使われる Lisp 言語風の dotted-pair list, alist という特殊リストがある．また，R の統計データに対する中心的なデータ構造であるデータフレーム (第 16 章参照) は，行列風な外見を持つリストであり，数値・文字データを一括して保存できる.

第 16 章
データフレーム


のデータフレーム (data frame) は多変量データを表現する汎用的なオブジェクトで，行列風の外見を持つリスト[*1]である．各列は 1 つの変数の値を表す同じ長さの数値ベクトル，文字列ベクトル (引用符なしで表示される) 等である．行列と異なり，異種の型の変数を同時に格納できる．各行 (case) は 1 つの対象 (例えば 1 人の被験者) に関するひとかたまりのデータ項目を並べたものと考えられる．行と列には内容を示す名前ラベルがつくことが多い．R の多くの統計処理，グラフィックス関数は引数としてデータフレームを想定し，専用のメソッド関数を持つ．その際，モデル式を用いて変数間の関係を表現することが可能になる．処理データをデータフレームの形にまとめ，それを必要に応じて変形することは，R システムで統計解析を行う必須の基本技術となる．

16.1 データフレーム

16.1.1 データフレームを作る `data.frame`

関数 `data.frame` はデータフレームを作る．データフレームは同じ長さの変数のリストで，クラス属性 `"data.frame"` を持つ．文字列変数は関数 `I` で保護されない限り，因子 (整数値による内部的表現) に変換される．行列やリスト (各成分が同じ長さのベクトル) は関数 `as.data.frame` を用い，その列や成分が変数であるデータフレームに変換できる．`data.frame` に渡される各変数は，同じ長さでなければならない．しかし，原子的なベクトル・因子，`I` で保護された文字列ベクトルは，必要ならリサイクル規則が適用される．(列の) 長さが同じならデータフレームであっても良い．もし列 (変数) 名が与えられないと，適切な名前を持つ最初の成分から合成されたり，1 から始まる整数値とされる[*2]．

[*1] 残念ながら，この事実がデータフレームを対象とする操作を一般的に低速にする．157 頁参照．

[*2] データフレームの変数・ケース名の自動生成関数 `provideDimnames` については 124 頁を参照．

第 17 章
関　　数

 は対話的なシステムであり，試行錯誤をしながら命令を順次打ち込んでいくやり方が基本であるが，多少複雑な処理は，たとえ 1 回限りの作業でも関数としておくと，デバッグ・改良・実行が容易になる．R の真の能力は適切な関数を書く[*1]ことにより発揮される．R には自前で関数を書くための便利な機能が豊富に用意されており，コーディングの効率が高い．一度定義された関数は，システム関数と区別なく使用することができる．

17.1　関数の書き方の基本

R の関数定義の基本書式は

```
関数名 <- function(仮引数リスト) 関数本体
```

である．仮引数リスト[*2]はカンマで区切った仮引数列である．**仮引数名=省略時既定値**の形式で，仮引数の省略時既定値を与えることもできる．特別な仮引数として...(ellipsis, dot-dot-dot) 引数も利用できる．仮引数リストは空でも良い．関数本体は関数の仮引数を含む R の実行文である．実行文が 1 つの場合は括弧 { } で囲む必要はない．

```
# 仮引数 x,y を持ち名前が
# foo という関数の定義
> foo <- function(x,y) {
    s <- x+y;t <- x-y
    return(sqrt(s^2+t^2))}
> foo(1,2)
[1] 3.162278
# 括弧の前には任意個数の
# 半角空白をおいて良い
> foo (1,2)
[1] 3.162278
> foo   (1,2)
[1] 3.162278
# 仮引数 x に対する実引数はベク
# トル 1:3．返り値は自然にベク
# トル化されている
> foo(1:3,2)
[1] 3.1622 4.0000 5.0990
```

関数本体は仮引数・局所変数以外にシステム定数や既に定義されている変数・関数を含んでも良い．

```
> a <- pi/2
# 外部変数を含む関数例
> foo <- function(x,y)
    a*sqrt(x^2+y^2)
# 仮引数のない関数例
> foo <- function()
    sample(1:100,5)
> foo()
[1]  8 91 39 11 85
```

[*1] 実際，R の関数の多くは特に高速さを要求される基本関数や，他の言語で書かれた数値関数ライブラリを除けば，S 言語 (の R 版) で書かれている．

[*2] function と丸括弧の間には任意個数の半角空白を置いて良い．

第 18 章
R の数値関数


は通常の計算機言語ではあり得ないほどの数値関数を備えている．これらは R 同様，長年蓄積・改良されてきた信頼性の高いオープンソース数値ライブラリ等を取り込んだものである．多くは，Fortran や C, C++ 等の言語で書かれており，R はこれらを内部的に呼び出す．以下では，線形代数，最適化関連関数[*1]を除いた関数等を紹介する．関数は原則としてベクトル化されており，結果もベクトルになる．複素数引数を許し，結果も複素数になる関数もある．

18.1　2 項型数値演算子

```
x + y, x - y, x * y, x / y   加減乗除
x^y        巾乗
x**y       巾乗 (将来廃止される可能性がある)
```

```
x %% y    x を y で割った余り．非負で y 未満
x %/% y   いわゆる整数商で値は整数
```

```
> x
[1] -1  0  1  2  3  4  5  6  7  8
> x %% 5
[1] 4 0 1 2 3 4 0 1 2
> x %/% 5
[1] -1  0  0  0  0  0  1  1  1  1
# x を復元
> (x%/%5)*5 + (x%%5)
[1] -1  0  1  2  3  4  5  6  7  8
> x <- 1+0.5i; y <- 3+2i
# 複素数も問題なし
> c(x+y,x-y,x*y,x/y,x^y)
[1]  4.000000+2.500000i
[2] -2.000000-1.500000i
[3]  2.000000+3.500000i
[4]  0.307692-0.038461i
[5] -0.023927+0.552381i
# (-8)^(1/3)=-2 にはならない！
> (-8)^(1/3)
[1] NaN
# 複素数として計算
# x^3=1 の 3 つの複素数解の 1 つ
> (-8+0i)^(1/3)
[1] 1+1.732051i
```

`x %% y` は適当な整数 `n` で `0<= x - ny < y` となるときの値 `x - ny` を表す．そのときの整数 `n` が `x %\% y` になる．`1^y, y^0` は常に 1 になる．`x, y` が正負の `Inf` のときも，可能なら合理的な値が返る．これらの演算子は総称的関数であり，オブジェクトのクラスに応じたメソッド関数を定義できる．

[*1] 汎用的最適化関数 `optim`，`nlm`，線形制約つき最適化関数 `constrOptim`，1 変数最適化関数 `optimize`，1 変数関数の根を求める `uniroot`，実・複素多項式の零点を見つける `polyroot`，数式微分関数 `deriv` 等．

第 19 章
apply 関数ファミリ

 には apply 関数ファミリという，ベクトル・配列・データフーレム・リストにある処理を繰り返し適用した結果を，再びベクトル・配列・データフーレム・リストとして一括して返す関数ファミリがある．処理関数を適切に定義することで複雑な処理を行うことができる．R プログラミングの基本精神「ベクトル化」を効率的に実践でき，またループ処理を隠蔽[*1]するため簡潔なコードを書くことができ，独立して紹介する価値がある．R を使うメリットを感じさせる関数ファミリである．以下は R 3.0.1 にある，そうした関数[*2]の一覧である．

aggregate	dendrapply	mapply	sapply	vapply
apply	eapply	replicate	tapply	
by	lapply	rapply	Vectorize	

19.1 apply 関数ファミリ

19.1.1 配列のマージンに関数を適用 apply

関数 `apply(X, MARGIN, FUN, ...)` は配列 (ベクトル・行列を含む) `X` の次元 (ベクトルで複数指定できる) `MARGIN` に関数 `FUN` を適用して得られる値を，ベクトル・配列・またはリストとして返す．もし `FUN` の各適用が同じ長さ `n` のベクトルを返すならば，`apply` の返り値は次元 `c(n, dim(X)[MARGIN])` の配列になる．但し，`MARGIN` が長さ 1 ならベクトル，さもなければ次元 `dim(X)[MARGIN]` の配列を返す．結果は一旦 `as.vector` で型変換されるので，例えば因子は文字列配列になる．

[*1] `lapply`, `vapply`, `sapply`, `rapply` 関数は内部関数を用いて計算するため，処理内容によっても異なる可能性があるが，単なるループ処理よりは計算速度自体が向上すると思われる (但し，使用環境に依存する可能性がある)．それ以外もバイトコンパイルされている分計算速度は向上すると思われる．209 頁を参照．

[*2] 関連する関数 `Reduce`, `Filter`, `Map` については 182 頁を参照．またデンドログラムの各ノードに再帰的に関数を適用する `dendrapply` 関数がある．

第 20 章
作表関数

には，データを分割表 (contingency table) 形式に集計する関数が幾つかある．表オブジェクトは属性 "table", "xtabs", "ftable" 等を持つ整数値配列である．

20.1 作表関数

20.1.1 分割表 table, tabulate

table は因子水準の組み合わせ毎にオブジェクトを集計して，表にする．関数 tabulate はその実働関数で，正整数値ベクトルを表化する．as.table, is.table は表への変換・検査関数である．as.data.frame は属性 "table" を持つ整数値配列をデータフレームに変換する．もし dnn が与えられないと dimname names が内部的に計算される．... 中の引数が名前を持てば，それが使われる．

返り値は属性 "table" を持つ整数値配列である分割表 (contingency table) である．table と xtabs には要約関数 summary のメソッドがあり，χ^2 検定も行う．

```
> table(rpois(100,5))
0 1  2  3  4  5  6  7 8 9 10
1 1 10 22 19 12 11 13 6 4  1
# 組み込みデータフレームを
# 使った例．変数 wool, tension
# の組み合わせで 2 元分類
> with(warpbreaks,
       table(wool,tension))
    tension
wool L M H
   A 9 9 9
   B 9 9 9
# 組み込みデータフレームを
# 使った例．変数 Temp を
# クォンタイル値で分類
# Month とともに 2 元分類
> with(airquality,
    table(cut(Temp,
    quantile(Temp)),Month))
        Month
         5  6  7  8  9
(56,72] 24  3  0  1 10
(72,79]  5 15  2  9 10
(79,85]  1  7 19  7  5
(85,97]  0  5 10 14  5
```

オプション deparse.level による，次元名と次元名の名前の違い．

```
> a <- letters[1:3]
> b <- sample(a)
# 既定 deparse.level=1
> table(a,b)
a   a b c
  a 1 0 0
  b 0 1 0
  c 0 0 1
> table(a,b,deparse.level=0)
    a b c
  a 1 0 0
  b 0 1 0
  c 0 0 1
> table(a,b,deparse.level=2)
   sample(a)
a   a b c
  a 1 0 0
  b 0 1 0
  c 0 0 1
> b <- factor(
    rep(c("A","B","C"),10))
```

第 21 章
暦日・時間


には日付 (年月日，date) オブジェクトを表すクラス `"Date"` と，日付・時間 (date-time) オブジェクトを表すクラス `"POSIXlt"`, `"POSIXct"` がある．これらは使用ロケールやタイムゾーンにより表現が異なる可能性がある．日付は内部的にはある基準日付からの経過日数で表されている．

21.1 暦日・時間用のクラス

21.1.1 date クラス `"Date"`

クラス `"Date"` は暦日 (年月日, date) オブジェクトを表し，`1970-01-01` 以来の経過日数 (負の値は過去に遡る) で表現される．内部的には実数で表現されるが，表示の際は整数値とされる．日数との加減算，比較演算が可能である．`format` や `plot` 関数は `"Date"` クラス用のメソッド関数を持つ．`as.Date` は数値を年月日オブジェクトに変換する．`weekdays`, `months`, `quarters` はそれぞれ曜日・月・四半期名を返す．

```
# 今日の日付
# 表示の際は使用ロケールに合わせ整形される
> today <- Sys.Date(); today
[1] "2013-06-11"
> str(today)
Date[1:1], format: "2013-06-11"
# "Date"クラス
> class(x)
[1] "Date"
# クラス属性"Date"を与える
> start <- 0; class(start) <- "Date"
> start
[1] "1970-01-01"
# 等差数列
> weeks <- seq(today,len=3,by="1 weeks")
> weeks
[1] "2013-06-11" "2013-06-18" "2013-06-25"
# 曜日・月・四半期
> c(weekdays(today),months(today),
    quarters(today))
[1] "火曜日" "6 月"    "Q2"
# 年月日オブジェクトに対する数値演算
# 1000 日後
> z <- today+1000; z
[1] "2016-03-07"
# 年月日オブジェクトに対する比較演算
> z < c("2010-08-21","2017-1-1")
[1] FALSE  TRUE
# 文字列を年月日オブジェクトに変換
> x <- as.Date("2007-1-1"); x
[1] "2007-01-01"
```

```
# 過去の閏秒のあった日付
> as.Date(.leap.seconds)
[1] "1972-07-01" "1973-01-01"
[3] "1974-01-01" "1975-01-01"
--途中省略--
[25] "2012-07-01"
```

第 22 章
入　出　力


のデータ・プログラムの入力・出力用の機能を解説する．出力には，コンソールへの出力 (標準出力)，ファイルへの出力，そして他の命令へのリダイレクトがある．入力には，コンソールからの直接の出力 (標準入力)，コード・データファイルの読み込み，がある．R およびそのアドオンパッケージには特殊な形式のファイル (Excel ファイル，SAS 等の統計システム出力，等) への入出力を可能にする関数がある[*1] がここでは触れない．

22.1　標準入出力

22.1.1　標準出力 cat

関数 `cat(...,file="",sep=" ",fill=FALSE,labels=NULL,append=FALSE)` は引数のオブジェクトを連結して標準出力やファイルに出力する．現在，原子的なベクトル (リストは駄目) と名前だけが処理できる．数値等は文字列に変換された後連結されて出力される．既定 (`fill=FALSE`) では，改行文字 `"/n"` を置いたときだけ改行することを注意しよう．`print` 関数等と比べると，`cat` は最小限の整形 (システムオプション `width`, `digits`, `scipen` 等に依存) しかしない．246 頁を参照．

```
# 分離記号"/"，改行されない
> cat(letters[1:10],sep="/")
a/b/c/d/e/f/g/h/i/j>
# 改行するが行末にも"/"が入る
> cat(letters[1:10],"\n",sep="/")
a/b/c/d/e/f/g/h/i/j/
# 改行オプション fill=TRUE を使う
> cat(letters[1:10],sep="/",fill=TRUE)
a/b/c/d/e/f/g/h/i/j
# ファイル"test.txt"に出力
> cat(letters[1:10],file="test.txt",
      sep="/",fill=TRUE)
# 同じことを sink を使うと以下の 3 行と同値
> sink("test.txt")
> cat(letters[1:10],sep="/",fill=TRUE)
> sink()
# 幅 width 文字で改行，行頭ラベルつき
> cat(paste(letters[1:3],1:10),fill=TRUE,
      labels=paste("{",1:3,"}:",sep=""))
{1}: a 1 b 2 c 3 a 4 b 5 c 6 a 7
{8}: b 8 c 9 a 10
```

[*1] 例えばパッケージ `foreign` 参照．

第 23 章
R の起動・終了，バッチ処理，環境変数


の起動と終了の仕組みは普通意識することもないが，結構複雑である．この章では R の起動と終了，そして関連環境変数に関する話題を紹介する．

23.1 R の起動と終了

23.1.1 R の起動と終了のメカニズム

R の起動と終了の仕組みは次のようなステップで行われる (プラットフォーム依存の可能性がある)．

- `--no-environ` がコマンドラインオプションとして与えられない限り，R は設定すべき環境変数をサイトとユーザファイルから探す．サイト用ファイルの存在位置は，環境変数 `R_ENVIRON` が設定されていればそれが，さもなければ `R_HOME/etc/Renviron.site` (工場出荷値) が使われる．ユーザ用ファイルは，環境変数 `R_ENVIRON_USER` が設定されていればそれが，さもなければ現在のディレクトリかユーザのホームディレクトリにあるファイル `.Renviron` が (もし存在すれば) 使われる．
- `--no-site-file` がコマンドラインオプションとして与えられない限り，R はサイト用起動時プロファイルを記述したファイルを探す．このファイルは環境変数 `R_PROFILE` が指示する．もしこれがなければ既定値は (もし存在すれば) `R_HOME/etc/Rprofile.site` とされる．これに書かれたコードは R の基本パッケージに読み込まれる．基本パッケージ中のオブジェクトをうっかり上書きしないように注意すべきであり，以下の例が示すように実行可能オブジェクトに対しては `local` 関数の使用が勧められる．
- `--no-init-file` がコマンドラインオプションとして与えられない限り，R はユーザ用起動時プロファイルを記述したファイルを探す．このファイルは環境変数 `R_PROFILE_USER` が指示する．もしこれがなければ既定値は (もし存在すれば) 現在のディレクトリかユーザのホームディレクトリにある `.Rprofile` とされる．こ

第 24 章
パッケージ

 は無数にあるパッケージを読み込むことにより様々な機能をつけ加えることができる．パッケージには R 起動時に必ず読み込まれる必須パッケージ，R 本体に必ず付属する推奨パッケージ，そして全世界のユーザが作成した 6 千余りのアドオンパッケージ[*1]がある．この章ではパッケージに関する話題[*2]を紹介する．

24.1　パッケージ

パッケージのインストールや確認を行う関数を紹介する．パッケージの名前空間を直接操作する関数ファミリについては 280 頁を，パッケージ中の変数に直接アクセスする，2 重・3 重コロン演算子 ::, ::: については 281 頁を参照されたい．

24.1.1　パッケージのインストール・更新・削除

以下はパッケージ (もしくはパッケージバンドル) のインストール，更新を R 内部からインターネット経由で行う[*3]関数ファミリである．

update.packages	new.packages	contrib.url	.packages
available.packages	download.packages	installed.packages	
old.packages	install.packages	remove.packages	

```
> install.packages(c("XML_0.99-5.tar.gz",
                   "../../Interfaces/Perl/RSPerl_0.8-0.tar.gz"),
                   repos=NULL,
                   configure.args=c(XML='--with-xml-config=xml-config',
                   RSPerl="--with-modules='IO Fcntl'"))
--インストール情報省略--
# インストール済みパッケージ情報
> installed.packages()
         Package     LibPath                          Version  Priority
```

[*1] R 自体と同じくほとんどがオープンソースであるが，一部商用使用禁止のものもある．

[*2] R の起動と終了時におけるパッケージの処理については 247 頁を参照．

[*3] Linux 系 OS ではソースからその場でコンパイルする，Windows や MacOSX の場合はコンパイル済みのバイナリ版が使われる．現在のパッケージはバイトコンパイル済みのものが多いようである．

第 25 章
R のメモリ管理用関数


は実行時のメモリを管理・監視するための幾つかの関数を持つ．

25.1　ガベージコレクション

作業中に新規にオブジェクトを作ったり，消去したりを繰り返すと空きメモリは細断され，作業効率が落ちる．ガベージコレクション機能はメモリ上のオブジェクトを必要なら一続きに移動・整理し，使いやすくする．

25.1.1　ガベージコレクション関数 gc

大きなオブジェクト，多数のオブジェクトを消去した場合等，空きメモリは細断され効率が悪くなる．R は必要に応じメモリ中のオブジェクトを自動的に再配置するが，これを手動で強制実行する関数が **gc**[*1]である．関数 **gcinfo** は自動ガベージコレクションの際に，出力を行うかどうかを選択する．関数 **gctorture** は開発者向けであり，更に詳細なガベージコレクションを行い，メモリリークの有無のチェックが可能になる．

関数の実行時間は，有効メモリ量と空きメモリの配置にも依存する．不要なオブジェクトがあれば削除し，ガベージコレクションを行うべきである．**system.time** 関数は，既定では時間計測の開始前にガベージコレクションを行う．関数 **gc.time** (266 頁参照) はガベージコレクションに必要だった時間を計測する．

```
> gc()
Garbage collection 24 = 19+1+4 (level 0) ...
5.9 Mbytes of cons cells used (54%)
2.3 Mbytes of vectors used (37%)
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 218151  5.9     467875 12.5   350000  9.4
Vcells 250879  2.0     786432  6.0   786432  6.0
```

reg.finalizer はガベージコレクション時や，オプションで R 終了時に実行される関数を登録する．

[*1] gc 関数は続けて 2 度実行することが勧められる．292 頁参照．

第 26 章
R の内部時計に関連する関数


には計算機の内部時計を利用した幾つかの関数がある．

26.1 内部時計を利用した関数

26.1.1 プログラムの実行時間を計測 system.time

関数 **system.time(expr,gcFirst=TRUE)** は proc.time を用いて，表現式 expr の実行時間を計測する．コードの高速化のためには不可欠である．既定では計測前にガベージコレクションを行う．計測される時間は順にユーザ時間・システム時間，そして経過時間 (単位秒) である．ユーザ時間は計算に要した CPU 専有時間，システム時間はプロセスの呼び出し時間，そして経過時間は両者の和で計算全体の実時間である．実際には副プロセスに関するユーザ・システム時間[*1]も計測されている．

```
# ベクトルを逐次拡大するまずいコード
> system.time(for(i in seq(1e5))
    x <- c(x,rnorm(1)))
   ユーザ    システム        経過
    19.668       7.196      27.010
```

```
# まとめて生成すれば一瞬
> system.time(x <- rnorm(1e5))
   ユーザ    システム        経過
    0.012       0.000       0.012
```

```
# ある関数 exT の実行時間．100 回の計測値を 5x100 行列にまとめる
> x <- sapply(1:100, function(i) system.time(exT(1000)))
# ユーザ時間の要約統計量
> c(summary(x[1,]),Sd.=sd(x[1,]))
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.      Sd.
0.264000 0.268000 0.268000 0.268200 0.268000 0.280000 0.002261
```

26.1.2 R の起動以来の経過時間 proc.time

関数 **proc.time** は現在稼働中の R の起動以来の経過秒数を与える．5 種類の時間の意味は前節を参照．

[*1] 普通は 0 となる．Windows では計測できず常に NA とされる．

第 27 章
環　　境


の基本概念である環境 (environment)・スコープ規則 (scoping rule)・検索パス (searching path) について簡単に紹介[*1]する．

27.1　環境・検索パス・スコープ規則

環境[*2]はオブジェクトのシンボル名とその値の対の集合であるフレーム (frame) と，その囲み (親) 環境 (enclosing environment) へのポインタであるエンクロージャ(enclosure) からなる．ある環境の囲み環境は関数 `parent.env` でアクセスできる．

27.1.1　環　　境

環境はツリー状になっており，囲み環境が親ノードになる．ツリーのルートノードは `emptyenv` と呼ばれる空の環境であり，R の基本パッケージ (base package) の親ノード (囲み環境) である．

```
# 大局的環境
> globalenv()
<environment: R_GlobalEnv>
# 大局的環境の親環境はパッケージ stats
> parent.env(.GlobalEnv)
attr(,"name")
[1] "package:stats"
attr(,"path")
[1] "/usr/lib/R/library/stats"
# stats の親環境はパッケージ graphics
> parent.env(parent.env(.GlobalEnv))
<environment: package:graphics>
attr(,"name")
[1] "package:graphics"
attr(,"path")
[1] "/usr/lib/R/library/graphics"
# 基本 (base) 環境
> baseenv()
<environment: base>
# 基本環境の親環境は空環境
> parent.env(baseenv())
<environment: R_EmptyEnv>
# 空環境は親環境を持たない
> parent.env(parent.env(baseenv()))
以下にエラー parent.env(env) :  空の環境は親を持ちません
```

[*1] 主に公式マニュアル「*The R Language Definition*」([2]) に基づく．

[*2] 複数の環境を持つことができるという R の特徴は R. Ihaka と R. Gentleman が R のプロトタイプを創った際，Lisp 言語の一種である Scheme 言語のアイデアを採り入れた (但し一部で誤解されているように，R 自体が Lisp 言語で書かれているわけではない) もので，本来の S 言語にはない独自の特徴である．S-PLUS では環境はフレーム (frame) と呼ばれ，大局的環境と，関数の評価環境しかない．

第 28 章
言語オブジェクト

 には関数型計算機言語としての R の機構の基礎である言語オブジェクト (language object) の概念[*1]がある．以下の解説は主として公式マニュアル「*The R language definition* (文献 [2])」に基づく．

28.1　言語オブジェクト

28.1.1　言語オブジェクト

計算機言語としての R を構成する 3 つの言語オブジェクト型，呼び出し (`call`)，表現式 (`expression`)，そして名前 (`name`)[*2]がある．これらはそれぞれ `"call"`, `"expression"` そして `"name"` というモードを持つ．関数 `is.language` は引数が言語オブジェクト型であるかどうかを検査する．

呼び出しオブジェクト (未評価表現式とも呼ばれるが，必ずしも適切ではない) は関数名と引数を与えて `call` 関数から作ることができる．

```
# call 関数で
# 呼び出しオブジェクトを作る
> cl <- call("round",10.4)
```

```
> cl
round(10.5)
# eval 関数で解釈実行できる
```

```
> eval(cl)
[1] 10
```

表現式は呼び出しオブジェクトを並べたリスト風構造であり，リスト風の成分操作ができる．

```
# 表現式を作る
> ex1 <- expression(1+0:5)
> str(ex1)
  expression(1 + 0:5)
```

```
# ex1 は呼び出しオブジェクトで
# はない
> is.call(ex1)
[1] FALSE
```

```
# eval 関数で解釈実行
> eval(ex1)
[1]  1  2  3  4  5
```

名前オブジェクトは `as.name` 関数を用いて文字列から作ることができる．

[*1] 一般ユーザにとって特に興味のあるこの章の内容は，`substitute` 関数で実行式から作図関数等における数式ラベル・注釈を作り出す機構であろう．

[*2] 「シンボル」(`symbol`) とほぼ区別なく使われる．シンボルは Lisp 言語での用語で，「名前」は S 言語での同等物である．

第 29 章
クラスとメソッド


のオブジェクト指向言語としての核となる機構がクラス (class) とメソッド (method) である．現在の R システムとそのパッケージには，S 言語第 3 版の仕様を実装した S3 クラス・メソッドと，S 言語第 4 版の仕様を実装した S4 クラス・メソッド[*1]が混在している．簡易で効果的ではあるが多分に見かけだけのオブジェクト指向機構である S3 クラス・メソッドに対して，S4 クラス・メソッドは本格的なオブジェクト指向機構を実現している．但し，S4 クラス・メソッドは以下で見るように多分に煩雑であり，S3 クラス・メソッドは将来もなくなることはないであろう．

クラスとメソッドに関する詳細はパッケージ等の汎用的ツールを作る際には必須の知識[*2]となる．一般ユーザにとっても，`str`, `plot`, `print`, `summary` 関数等，同じ関数の適用がオブジェクトに応じて全く異なる挙動を示す仕組みを知ることは R システムを理解する上で欠かせない．この章では R のクラスとメソッドについて，基本常識レベルの紹介を行う．

注意：この章で紹介する以外に R には無数のクラス・総称的関数・メソッドを対象にした関数があるが，多くは一般ユーザには無縁と思われる．`help.search("class", package="base")` 等の命令で検索してみよ．

29.1 オブジェクトとその属性

R の多くのオブジェクトは，自分がどのようなオブジェクトなのかを示す様々な属性 (attribute) を持つ．新しい属性を付加したり，既存の属性を変更することにより新しいオブジェクトを作ることができる．

[*1] 例えば，R 本体に付属するパッケージ `stats4` は S4 クラス・メソッドに基づく統計処理関数からなる．

[*2] S 言語仕様については S 言語の創始者たちによる [17, 18, 21] が原典である．R システムにおけるクラス・メソッドについては公式マニュアル [1, 2, 3] に解説がある．R News 4/1, 2004, pp. 33 – 36: `http://cran.r-project.org/doc/Rnews/Rnews_2004-1.pdf` に Thomas Lumley による解説 “*Programmer's Niche: A Simple Class, in S3 and S4*” がある．

第 30 章
情報を得る


に関する情報を得る方法についてまとめる．パッケージに関する情報を得る方法は 257 頁を参照されたい．

30.1 R のオブジェクトに関する情報

30.1.1 オブジェクトの構造を見る str

関数 **str** は引数で与えられたオブジェクトの構造を簡潔に示す．様々なクラス用のメソッドが用意されており，それぞれにふさわしい表示が自動的に選択される．豊富なオプションを設定するための専用関数 **strOptions** がある．作業中にあるオブジェクトの正体に疑問を感じたら，すかさず **str** 関数で調べることが勧められる．

```
# str 関数自体の構造，単に関数と表示
> str(str)
function (object,...)
# lm 関数の構造，引数も表示
> str(lm)
--出力総略--
# 整数ベクトルの構造
> str(1:5)
int [1:5] 1 2 3 4 5
# 実数ベクトルの構造
> str(as.numeric(1:5))
num [1:5] 1 2 3 4 5
# 整数行列の構造
> str(matrix(1:6,2,3))
int [1:2,1:3] 1 2 3 4 5 6
# 整数配列の構造
> str(array(1:8,c(2,2,2)))
int [1:2,1:2,1:2] 1 2 3 4 5 6 7 8
```

```
# リストの構造
> str(list(a=1:3,b=rnorm(3)))
List of 2
 $ a: int [1:3] 1 2 3
 $ b: num [1:3] 0.753 0.424 0.839
# 入れ子リストの構造
> str(list(a=1:3,b=list(c=runif(4),d=sin)))
List of 2
 $ a: int [1:3] 1 2 3
 $ b:List of 2
  ..$ c: num [1:4] 0.796 0.206 0.538 0.395
  ..$ d:function (x)
# データフレームの構造
> str(data.frame(a=1:3,b=letters[1:3]))
'data.frame':  3 obs. of  2 variables:
 $ a: int  1 2 3
 $ b: Factor w/ 3 levels "a","b","c":1 2 3
```

以下は線形モデル当てはめ結果の構造を幾つかのオプション引数で示したものである．

```
> str(lsfit(1:9,1:9))
--結果省略--
# リストの入れ子レベルを制限
> str(lsfit(1:9,1:9),max.level=1)
List of 4
```

第 31 章
バイトコードコンパイラパッケージ compiler


の推奨パッケージに比較的新しくつけ加わったバイトコードコンパイラパッケージ compiler[*1]を紹介する．バイトコードコンパイラは R コードをバイトコード[*2]に変換し，実行時間が数割程度早くなる可能性がある． 利用には事前に library(compiler) を実行しておく．

31.1 バイトコードコンパイラ

31.1.1 バイトコンパイル関数 cmpfun, compile

パッケージ compiler に含まれる関数は以下の通りである．

cmpfun	loadcmp	compilePKGS
compile	disassemble	getCompilerOption
cmpfile	enableJIT	setCompilerOptions

関数 cmpfun は関数クロージャの本体をコンパイルし，本体をそれで置き換えた新しいクロージャを返す．compile は表現式をバイトコードオブジェクトにコンパイルする．cmpfile は入力ファイル infile をコンパイルし，出力ファイル outfile に書き出す．出力ファイル名を与えないと，入力ファイル名に拡張子 .Rc をつけたものとされる．loadcmp はコンパイル済みファイルの読み込み[*3]をする．disassemble はコードの可読表現を与える．

コンパイル時に様々な警告が cat 関数を用いて出力される．options 引数はコンパイル動作を制御する．現在の 3 つのオプションは，optimize (最適化レベル, 0,1,2,3), suppressAll (TRUE ならメッセージは出力されない), suppressUndefined (TRUE なら未

[*1] かつては独立パッケージとして提供されていた．

[*2] バイトコードコンパイラはインタプリター型言語の実行速度を早めるために使われる．本来のコードを R が実行する際，まずそれを構文解析した中間コードに変換し，最後にそれを実行する．バイトコードコンパイラはこの中間コードを直接生成する．これは人間が解釈しやすい形式にはなっていない．

[*3] sys.source に似るが，既定の読み込み環境はベース環境ではなく，大局的環境になる．

第 32 章
並列処理パッケージ parallel

 の推奨パッケージに比較的新しくつけ加わった並列処理パッケージ **parallel**[*1]を紹介する．このパッケージはマルチコア CPU を持つ 1 台の計算機で並列処理を行うことを想定している．

32.1　クラスタの作成と操作

最近の CPU はマルチコア化されており，各コアが独立した CPU として動作するため，1 台の計算機を複数の計算機のように使うことができ，並列処理を手軽に実行できるようになった．但し，Windows 系 OS では使える機能に制限がある．

以下は `library(help=parallel)` で得られるパッケージ **parallel** の目次である．それぞれが関連する複数の関数を含む．詳細は各ヘルプ文章を参照されたい．

children	フォークされたプロセスを管理する低水準関数
clusterApply	クラスタを用いた Apply 操作
detectCores	CPU のコア数を検出する
makeCluster	並列ソケットクラスタを生成する
mcaffinity	現在のプロセスの CPU Affinity Mask を得る・設定する
mcfork	現在の R をプロセスのコピーをフォークする
mclapply	フォークを用いた lapply と mapply の並列版
mcparallel	R 表現式を別個のプロセスで非同期的に評価する
nextRNGStream	L'Ecuyer の RngStream（並列版擬似乱数生成器）
pvec	フォークを用いたベクトル Map 関数の並列化
splitIndices	クラスタに作業を分割配分する

32.1.1　R における並列処理機能

互いに情報をやり取りする親・子プロセスの集まりをクラスタ (cluster) と呼ぶ．この章ではマルチコア CPU を持つ計算機でのクラスタを用いた並列処理の概観を与える．並列計算では

1. R を実行している親プロセス (master process) が，それ自体独立した R セッショ

[*1] 2 つのパッケージ **multicore** と **snow** の内容を受け継ぐ．両者は子プロセスの実装の仕方に違い (fork と socket) がある．パッケージ **parallel** には解説ビニエットが存在する (`vignette("parallel")` を実行)．

付録
R を用いた講義デモ用の関数

 を用いた統計学・データ解析の講義の例は現在では珍しくないようである．自分のパソコンに R をインストールすることで学生が手軽に高度な統計手法を使用できること，本格的な大規模データをインターネット経由で自由に入手できることがこうした風潮を後押ししている．

以下では著者が実際にデータ解析の講義で使用しているプレゼンテーション用の関数 `Demo`[*1] を紹介する．この関数はアドオンパッケージ `evaluate` に含まれる関数 `evaluate` と `replay` 関数を使い，ソースファイルを予め命令単位に解釈実行した結果をリストにまとめ，それを改めて擬似的に実行しているように見せかけているため，実際に実行する過程をほぼ忠実に再現できる．擬似実行中に作られたオブジェクトは保存されない．途中でアドリブ的に追加実行はできない．コメントおよびコメント行もそのまま表示されるため，説明をコメントとして書き込むことができ，ソースファイル自体をテキスト代わりに使うこともできる．

```
# ソースファイルを一旦実行した結果を逐一リストにまとめ，後から擬似的に実行するデモ用の関数
# リターンキーを押すたびに 1 命令ずつ実行する
Demo <- function(x) {     # x はソースファイル名
  require(evaluate)       # アドオンパッケージ evaluate を読み込む
  xx <- evaluate(file(x))
  tmp <- function(y) {
    yy <- y
    z <- substr(yy, 1, nchar(yy)-1)
    if(class(yy)[1] == "source")
      yy <- paste("> ", z, sep="")
    else if(class(yy)[1] == "character")
      yy <- z
    replay(yy)
    invisible(readline())
  }
  invisible(lapply(xx, tmp))
}
```

使い方は，R のソースファイルを `foo.R` とすると `Demo("foo.R")` を実行 (取りあえず

[*1] 関数中の補助関数 `tmp` はステップ毎に必ず入る空白行を取り除くために使われるが，作図関数等ではやはり余分な空白行が入る．

参考文献

[1] *An Introduction to R*, R の公式マニュアル (R 入門)
CRAN の Documentation – Contributed 頁に和訳がある.

[2] *The R language definition*, R の公式マニュアル (R の言語仕様)
CRAN の Documentation – Contributed 頁に和訳がある.

[3] *Writing R Extensions*, R の公式マニュアル (R のパッケージ作成法)
CRAN の Documentation – Contributed 頁に和訳がある.

[4] *R Data Import/Export*, R の公式マニュアル (データ入出力)
CRAN の Documentation – Contributed 頁に和訳がある.

[5] *R Installation and Administration*, R の公式マニュアル (R のインストール法)
CRAN の Documentation – Contributed 頁に和訳がある.

[6] *R Internals*, R の公式マニュアル (R の内部仕様とコーディング標準).

[7] *The R Reference Index*, R の公式マニュアル (R 本体と推奨パッケージ中の全てのヘルプ文章を含む).

[8] *R Newsletter*, R の公式ニュースレター. 1 年に数回オンラインで発行される. CRAN から入手可能.

[9] *R FAQ*, R の公式マニュアル FAQ (しばしば質問される項目に対する回答集). RjpWiki に和訳がある.

[10] *R MACOS X FAQ*, R の公式マニュアル (MacOS X 専用の FAQ 集). RjpWiki に和訳がある.

[11] *R FAQ*, R の公式マニュアル (Windows 専用の FAQ 集). RjpWiki に和訳がある.

[12] *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*, R. Gentleman et al., Springer (2005). 著者 Gentleman は R 開発チームのメンバーで, R 創始者の 1 人. Bioconductor というバイオインフォーマティックス関連の膨大な R パッケージ群開発のリーダー. 和訳「R と Biocunductor を用いたバイオインフォーマティックス」, 荒川和春他訳, シュプリンガー・ジャパン (2007).

[13] *R Graphics*, P. Murrell, Chapman & Hall (2005). R のグラフィックスに関する基本的文献. 著者は R 開発メンバーの 1 人.

[14]「The R Book」, 岡田昌史編, 九天社 (2004).

[15]「R による医療統計学」, P. Dalgaard 著, 岡田昌史監訳, 丸善出版 (2007). *Intro-*

ductory Statistics with R, P. Dalgaard, Springer-Verlag (2002) の訳．R そのものをターゲットにした初めての本．著者は R の開発メンバーの 1 人．この本で取り上げられたデータとコードは `ISwR` という R のパッケージになって公開されている．

[16]「R の基礎とプログラミング技法」，U. リゲス著，石田基広訳，シュプリンガー・ジャパン (2006)．著者は R 開発メンバーの 1 人．

[17] J.M. Chambers & J.T. Hastie (eds.), *Statistical Models in S*, Wadsworth & BrooksCole (1992).「S と統計モデル – データ科学の新しい波 –」，柴田里程訳，共立出版 (1994).

[18] Becker, R. A., Chambers, J. M. and Wilks, A. R., *The New S Language* . Wadsworth & Brooks/Cole (1988).「S 言語 データ解析とグラフィックスのためのプログラミング環境 I，II」，渋谷政昭・柴田里程訳，共立出版 (1991).

[19]「S-PLUS による統計解析」，W.N. ヴェナブルズ，B.D. リプリー著，伊藤幹夫他訳，シュプリンガー・フェアラーク東京 (2001)．原著 *Modern Applied Statistics with S-PLUS*, 4th ed., W.N. Venables & B.D. Ripley, Springer (2002). S-PLUS のみならず R に関しても基本文献．特に統計手法の解説が詳しい．この本で取り上げられたデータとコードは `VR` という R のパッケージになって公開されている．著者リプリーは R 開発メンバーの中心人物．

[20]「R 基本統計関数マニュアル」, 著者が R 本体 (バージョン 2.6,2.7) に含まれる全統計関数とそのヘルプ文章をカテゴリー別に分類して翻訳したマニュアル．CRAN の非英語貢献マニュアル頁にある．404 頁の PDF ファイル．

[21] Chambers, J.M., *Programming With Data: Guide to the S Language*, Springer-Verlag (1998).「データによるプログラミング」，垂水共之他訳，森北出版 (2002).

[22] Chambers, J.M. *Software for Data Analysis: Programming with R*, Springer-Verlag (2008). 上記の本の R 版.

[23] RjpWiki．日本の R ユーザがボランティアで運営している情報サイト．
`URL http://www.okada.jp.org/RWiki/index.php?RjpWiki`.

[24] CRAN(Complete R Archive Network)．R の本拠サイト
`URL http://cran.md.tsukuba.ac.jp/`.
日本の代表的ミラーサイト `URL http://cran.md.tsukuba.ac.jp/`.

[25] R Wiki．R の公式 Wiki．様々な情報が得られる．
`URL http://wiki.r-project.org/rwiki/`.

[26] R とパッケージの参考コードの出力画像を一同に集めたサイト「R graphical Manual」．URL はキーワード `R Grapical Manual` でネット検索せよ．

[27] R 関連の検索エンジン「R Site Search」，
`URL http://finzi.psych.upenn.edu/search.html`.
Firefox 用のプラグインがある．URL `http://wiki.r-project.org/rwiki/`.

[28] R 関連のもう 1 つの検索エンジン「Rseek」, URL http://www.rseek.org/.

[29] S-Plus/R の関数のカテゴリー化されたリスト「S-Plus/R Function Finder」, http://biostat.mc.vanderbilt.edu/s/finder/finder.html.

索　　引

概念索引

関数索引

著者略歴

間瀬　茂（ませ　しげる）

1976年　東京工業大学大学院数学専攻博士課程修了
　　理学博士
現　在　東京工業大学　名誉教授
　　専門：空間統計学，地球統計学

主要著書

空間データモデリング（共著，共立出版，2001）
工学のためのデータサイエンス入門（共著，数理工学社，2004）
地球統計学とクリギング法（オーム社，2010）
ベイズ法の基礎と応用（日本評論社，2016）

新・数理/工学ライブラリ［情報工学＝2］

Rプログラミングマニュアル [第2版]
—Rバージョン3対応—　　（電子版）

2018年 2月 25日 ©　　第2版発行

この電子書籍は2014年5月10日第2版発行の同タイトルを底本としています.

著者　間瀬　茂　　発行者　矢沢和俊

【発行】　株式会社　数理工学社
〒151–0051　東京都渋谷区千駄ヶ谷1丁目3番25号
編集☎ (03) 5474–8661 (代)　サイエンスビル

【発売】　株式会社　サイエンス社
〒151–0051　東京都渋谷区千駄ヶ谷1丁目3番25号
営業☎ (03) 5474–8500 (代)　振替 00170–7–2387
FAX☎ (03) 5474–8900

組版　三美印刷
《検印省略》

ISBN978-4-86481-900-8