R ノート:
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

Bill Venables & Dave Smith

Department of Statistics
The University of Adelaide

Robert Gentleman & Ross Ihaka

Department of Statistics
University of Auckland

Martin Mächler

Statistics Seminar
ETH Zurich

前書き

この R の解説書は Bill Venables と Dave Smith によって書かれた S と S-PLUS 環境の解説書のオリジナル版を元に書かれている．R と S のプログラム機能で違う若干の点は修正している．R プロジェクトは現在進行形中であるため，未だその能力は S ほどではない．この解説書において，我々がインプリメントする予定の機能については，その機能を解説している章のはじめでその旨を断るという便法を採用している．ユーザーは，こうした未装備の機能のどれでも良いから自らインプリメントすることにより，このプロジェクトに貢献することができる．

このような改訂版を配布する許可を与え，背後から R のサポートを与えてくれている Bill Venables に厚く感謝したい．

コメントや訂正はいつでも歓迎される．下記アドレスにメールしてください．

`R@stat.auckland.ac.nz`.

読者への注意

ほとんどの R の初心者は Appendix A の入門的な章からスタートするだろう．こうすることにより R におけるセッションのスタイルにある程度なれることができるだけでなく，より重要な点は実際にどんなことが出来るのかを直ちに体験することができるであろう．

主にグラフィックス機能を目当てにR を使うユーザーが多いであろう．そうした場合には，グラフィックス機能を扱った章 9 をいつでも必要な時に参照すればよく，その前のすべての章の内容を理解するまで待つ必要はない．

Robert Gentleman and Ross Ihaka,
オークランド大学,
April, 1997.

訳者注この Rnotes の日本語訳は，英語原文と同様に，完全にフリーであり，英語原文と全く同じ条件の下で，自由に配布，利用，修正可能である．邦訳についての不備は，ひとえにボランティアの翻訳者の能力（そしてなによりも暇）の欠如によるものである．こういった表現をした方がいいだろうとか，ここは本当はこういう意味だよ，とかいったアドバイスはいつでも歓迎する．邦訳に関してのメールはメイリングリスト

`R-jp@epidemiology.md.tsukuba.ac.jp`

宛に送ってほしい．このメイリングリストは，すでに存在する R プロジェクト関連の文章の日本語化を目指して，とりあえず作られた．協力してくださる方は，このアドレスにメイルをください．

目 次

1 簡単な操作：数字とベクトル

1.1 ベクトルと付値

R は名前のついた「データ構造」を処理する．もっとも簡単なそうした構造は「ベクトル」であり，順序づけられた数字の集まりからなる単一の対象である．5 つの数字（たとえば 10.4, 5.6, 3.1, 6.4 and 21.7）から構成された x という名前のベクトルをつくるには，R 命令

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

を用いる．これは「関数」 c() を使った「付値」であり，この文脈では，任意の個数のベクトル「引数」取ることができ，その値はその引数を端から端まで連結して得られるベクトルである[1]

ある表現中に単独で現れた数値は，長さ 1 のベクトルと見なされる．

付値演算子は通常の = で「ない」ことに注意しよう．等号記号は別の目的のためにとっておかれている．付値演算子は，真横に並ぶ二つの文字 < （「より小さい」）と - （「マイナス」）からなり[2]，表現式の値を受け取るオブジェクトを「指して」いる．

付値は assign() 関数を使うことによっても可能である．上記の代入操作と同様のことを行うには

```
> assign("x", c(10.4, 5.6, 3.1, 6.4, 21.7))
```

とすればよい．通常の演算子 <- は，これの構文的な短縮形と考えることができる．

付値は同様に別向きで行うことも可能で，付値演算子の自明な変更を伴う．従って，同じことを

```
> c(10.4, 5.6, 3.1, 6.4, 21.7) -> x
```

としてもよい．もし表現が完全な命令として実行されるならば，その値が表示され，そして「失われる」．だから，もし命令

```
> 1/x
```

を実行すると，5 つの値の逆数がターミナルに表示される（そして，もちろん x の値は変化しない）．

更に次の付値

```
> y <- c(x, 0, x)
```

は，x の二つのコピーと，真ん中に零がある 11 個のエントリーからなるベクトル y を作るだろう．

1.2 ベクトル演算

ベクトルは算術表現中で使うことができ，その場合は，演算は要素毎に行われる．同じ表現中のベクトルは全て同じ長さである必要は無い．もし同じ長さでないならば，表現の値は，表現中の最も長い配列と同じ長さの配列になる．表現中のより短いベクトルは，それが最長のベクトルの長さに

[1] list モードの引数のような，ベクトル引数以外では，c() の行動はかなり異なる．§4.2.1 を参照せよ．
[2] 下線記号 _ は左側付値演算子 <- の代わりに使うことができるが，より可読性の低いコードにつながりやすく，勧められない．

一致するまで（おそらく部分的に）必要なだけ「リサイクル」される．特に定数は単純に繰り返される．したがって，上記の付値において，次の命令

```
> v <- 2*x + y + 1
```

は，要素毎の和からなる長さ 11 のベクトル v を作るが，2*x が 2.2 回繰り返され，y は丁度一度だけ繰り返され，1 は 11 回繰り返される．

初等的な演算子は通常の +, −, *, /，そして冪乗 ^ である．更に普通の全ての算術関数を使うことができる．log, exp, sin, cos, tan, sqrt 等は，全てそれらの通常の意味を持つ．max と min それぞれベクトル中の最も大きいものと最も小さいものを選択する．range() は長さ 2 のベクトル，つまり c(min(x), max(x)) を値に持つ関数である．length(x) は x の要素の数を返し，sum(x) は x の要素の総和を返し，prod(x) は全要素をかけた値を返す．

統計関数として，標本平均を計算する mean(x) があり，これは sum(x)/length(x) と同じである．var(x) は

```
sum((x-mean(x))^2)/(length(x)-1)
```

つまり標本分散を与える．もし var() の引数が $n \times p$ 行列なら，その値は，列を独立な p 変量標本ベクトルとみなした $p \times p$ の標本共分散行列となる．

sort(x) は要素を昇順に並べ変えた x と同じ長さの配列を返す．しかしながら，その他にも，もっと融通の効くソート機能がある（ソートのための置換を作る order() や sort.list() をみよ）．

rnorm(x) は x と同じ長さの，標準正規分布に従う疑似乱数からなるベクトル（より一般に配列）を作り出す．

1.3 規則的な数列の生成

R はよく使われる数列を生成する幾つかの機能を持つ．例えば 1:30 はベクトル c(1,2, ...,29,30) である．コロン演算子は一つの表現中で最も高い優先度を持つので，例えば 2*1:15 は c(2,4,6, ...,28,30) というベクトルになる．n <- 10 として，数列 1:n-1 と 1:(n-1) を比べてみよ．

構成 30:1 は降順の数列を作るのに使うことができる．

関数 seq() は数列を生成するもっと一般的な機能である．これは 5 個の引数を持つが，特定の呼出しでは，その一部分だけを指定すればよい．最初の二つの引数は．もし存在すれば．数列の最初と最後を指定し，もしこれだけが引数なら，結果はコロン演算子と同じになる．つまり seq(2,10) は 2:10 と同じベクトルになる．

他の多くの R の関数と同様に，seq() へのパラメーターは名前付き形式で与えることができ，その場合にはそれらが現れる順序は勝手である．最初の二つのパラメータは from=*value* と to=*value* と名前を付けることができる．したがって，seq(1,30), seq(from=1, to=30), seq(to=30, from=1) はすべて 1:30 と同じ結果になる．次の二つのパラメーターは by=*value*, length=*value* という名前を付けることができ，それぞれ数列の増分と長さを指定する．もしどちらも与えられなければ，既定では by=1 が仮定される．

例を示すと

```
> seq(-5, 5, by=.2) -> s3
```

とすると s3 は ベクトル c(-5.0, -4.8, -4.6, ..., 4.6, 4.8, 5.0) になる．同じく

```
> s4 <- seq(length=51, from=-5, by=.2)
```

とすると s4 も同じ配列になる．

5 番目のパラメーターは along=*vector* という名前を付けることができ，もし使うのなら，このパラメーターだけを指定しなければならない．これは 1,2,...,length(*vector*) というベクトルを作るか，もし空のベクトルを指定（可能である）すると空の数列をつくる．

関連した関数として rep() があり，さまざまな複雑なやり方でオブジェクトを複製するのに使うことができる．最も簡単な例

```
> s5 <- rep(x, times=5)
```

は，s5 を x の 5 つのコピーを端から端へとつないだものにする．

1.4 論理ベクトル

数値ベクトルと同様に，R は論理量を扱うことができる．論理ベクトルの要素は丁度二つの可能な値だけを持つことができ，形式的に FALSE と TRUE と表される．これらは，普通それぞれ F と T と省略される．

論理ベクトルは *conditions* により生成される．例えば，

```
> temp <- x>13
```

は temp を x と同じ長さで，x の対応する要素が条件を「満足しなければ」F を,「満足すれば」 T にしたベクトルにする．

論理演算子には <, <=, >, >=，完全な一致を表す ==，不一致を表す!= がある．更に，もし c1 と c2 が論理表現なら，c1 & c2 はそれらの論理積，c1 | c2 は論理和，そして ! c1 は否定である．

論理ベクトルは通常の算術演算において使うことができ，そのときは数式ベクトルに「強制変換」され，F は 0 に， T は 1 になる．しかしながら，論理ベクトルとその強制変換された数式ベクトルが等価にならない場合がある．例は次の項を参照のこと．

1.5 欠損値

ベクトルの要素が完全には知られていないことがある．統計的な意味で，要素や値が「利用不能」や「欠損値 (Not Available)」であるとき，ベクトル中のその位置に NA という特別な値を付値して保存しておくことができる．一般的には NA に対する処理は NA となる．この規則を設けた理由は単純で，もしある演算に対する指示が不完全ならば，結果を知ることができず，したがって利用不可能だからである．

is.na(x) 関数は，x と同じ長さの論理ベクトルで，x 中の対応する要素の値が NA の時，そしてそのときのみ値 T を持つ物を与える．

```
> ind <- is.na(z)
```

NA は実際は値でなく，利用不可能な量に対する標識であるから，論理表現 x == NA と is.na(x) は全く別物であることを注意しよう．だから x == NA は x と同じ長さを持ち，その「全て」の値が NA であるベクトルである．論理表現自身が不完全でしたがって決定不能だからである．

1.6 文字ベクトル

例えばプロットのラベル等，R では文字や文字ベクトルを頻繁に使う．必要ならば，それらはダブルクォート文字で括られた文字列によって表現される．例えば，"x-values" や "New interatoin results".

複数の文字ベクトルを c() 関数によって，一つのベクトルに連結するすることができる．以下で多くの例が登場するであろう．

paste() 関数は任意の数の引数を取り，それらを一つの文字列に連結することができる．引数中に与えられた数値は自明の仕方で文字列に強制変換される，つまり，それらが印字された時にそうなるであろう形である．引数は，デフォルトでは一つの空白文字で区切られた形になるが，これは名前付きパラメーター sep=*string* で変更でき，区切り文字を *string* に変える．空文字（区切り文字無し）も可能である．

たとえば，例

```
> labs <- paste(c("X","Y"), 1:10, sep="")
```

は labs を文字ベクトル

```
("X1", "Y2", "X3", "Y4", "X5", "Y6", "X7", "Y8", "X9", "Y10")
```

にする．ここでも短いリストはリサイクルが行われるということを注意せよ；つまり，c("X","Y") は数列 1:10 にマッチするまで 5 回繰り返される．

1.7 添字ベクトル．データセットの部分集合の選択と修正

ベクトルの要素の部分集合は，ベクトル名に鍵括弧に入った「添字ベクトル」をあてがうことにより選択することができる．より一般に，ベクトルとして評価される任意の表現は，表現の直後に鍵括弧に入った添字ベクトルを加えることにより，同じようにその要素の部分集合を得ることができる．

このような添字ベクトルは，4 つの異なったタイプのいずれでも良い．

論理ベクトル． この場合，添字ベクトルは，要素を選び出すベクトルと同じ長さを持つ必要がある．添字ベクトル中の T に対応する値が選択され，F に対応するものは無視される．例えば，

```
> y <- x[!is.na(x)]
```

は， x の欠損値でない値を，同じ順序に並べたオブジェクト y を作る（または y を変える）．もし x が欠損値を持てば，y は x よりも短くなることを注意しよう．同様に

```
> (x+1)[(!is.na(x)) & x>0] -> z
```

は，オブジェクト z をつくり，それに，ベクトル x+1 の対応する値が，欠損値でもなく，正の値持つようなものを，要素として置く．

2. **正の整数値ベクトル．** この場合，添字ベクトル中の値は集合 {1，2，...,length(x)} 中になければならない．対応するベクトルの要素が選び出され，結果中に「その順序で」まとめられる．添字ベクトルは任意の長さで良く，その結果は添字ベクトルと同じ長さとなる．例えば x[6] は x の 6 番目の成分であり

   ```
   > x[1:10]
   ```

   は x の最初の 10 個の要素が選び出す（length(x) が $\geq$ 10 であることを仮定すれば）．同様に

   ```
   > c("x","y")[rep(c(1,2,2,1), times=4)]
   ```

   は，4 回繰り返された "x","y","y","x" からなる，長さ 16 の文字ベクトルを作る（あまりしないことだろうが）．

3. **負の整数値ベクトル．** このような添字ベクトルは，選択するよりも，除外されるべき値を特定する，だから

   ```
   > y <- x[-(1:5)]
   ```

   は，x の最初の 5 つの要素を除いた y を与える．

4. **文字列ベクトル．** これは，オブジェクトがその要素を特定するための名札属性を持つ場合のみに使われる．名札ベクトルのサブベクトルを，上記の 2. と同様に使うことができる．

   ```
   > fruit <- c(5, 10, 1, 20)
   > names(fruit) <- c("orange", "banana", "apple", "peach")
   > lunch <- fruit[c("apple","orange")]
   ```

   文字と数字を組み合わせた「名札」は，しばしば「数字の添字」よりも覚えやすいという利点がある．あとで分かるように，このオプションはデータフレームを扱う際にとりわけ役にたつ．

添字表現はまた，一つの付値の代入される側に現れることができ，そのときは付値演算は「ベクトルのそうして特定された要素に対してのみ」実行される．表現は vector[*index_vector*] の形を持たねばならない．ベクトル名の位置に勝手な表現を置くことは，この場合あまり意味がないからである．

付値されるベクトルは，添字ベクトルの長さと釣合が取れていなければならず，論理添字ベクトルの場合は，それは添字操作されるベクトルと同じ長さでなければならない．

例えば

```
> x[is.na(x)] <- 0
```

は，x 中の欠損値を零に置き換え，そして

```
> y[y<0] <- -y[y<0]
```

は次とと同じ効果を持つ

```
> y <- abs(y)^3
```

[3] abs() は複素数の場合には期待されるような行動をしない．複素数の絶対値に対する適当な関数は Mod() である．

2 オブジェクト．そのモードと属性

2.1 本質的属性： *mode* と *length*

R が処理する対象は，技術的には「オブジェクト」として知られているものである．例えば，数値（実数）ベクトルや複素数ベクトル，論理値ベクトルや文字列ベクトルである．これらは「アトミック（原始的）な」構造として知られている．なぜなら，それらが全て同じタイプ，もしくは「モード」，つまりそれぞれ「数値[4]」,「複素数」,「論理値」，そして「文字」，からなるからである．

ベクトルは必ず，全て同じモードからなる値をを持たなければならない．だから，与えられたどのベクトルも，曖昧さ無しに,「論理」,「数値」,「複素数」もしくは「文字」のどれかに属さなければならない．唯一の些細な例外が，利用できない量を示す `NA` として言及された特別な「値」である．ベクトルは空であっても，一つのモードを持つことを注意しよう．例えば，空の文字ベクトルは `character(0)` として，空の数値ベクトルは `numeric(0)` として言及できる．

R は「リスト」(*lists*) と呼ばれるオブジェクトも処理でき，それらはリストというモードを持っている．これらは，それぞれが任意のモードを持つことができる，オブジェクトの順序づけられた列である.「リスト」は，原始的ではなく,「再帰的な」構造を持つ．なぜなら，その構成要素がそれ自身リストになり得るからである．

他の再帰的な構造として「関数」(*function*) と「表現」(*expression*) がある.「関数」は R システムの一部分をなすものと，ユーザーが書いた関数とがあり，このノートの後のほうである程度詳しく議論されるであろう．オブジェクトとしての「表現」は R のより進んだ部分であり，このノートでは，R におけるモデリングとともに用いられる「公式」(*formulæ*) を議論する際に間接的にふれる以外には，議論されないであろう．

オブジェクトのモードとは，その本質的構成物の基本的な型を意味する．これはオブジェクトの「属性」(*attributes*) の特殊な例である．オブジェクトの属性は，オブジェクトそのもののに関する特別な情報を提供する．すべてのオブジェクトにも備わっているもう一つの特質はオブジェクトの「長さ」(*length*) である．関数 `mode(`*object*`)` と `length(`*object*`)` は，任意の既定義構造のモードと長さを見出すのに使うことができる．

より重要なことは，こうした関数は，あるオブジェクトの属性を変更するために付値の左側におくことできることであり，もしこの変更が唯一つのモードだけなら，この過程は「強制変換」(*coercion*) として知られているものになる．

例えば，`z` が長さ 100 の複素数ベクトルなら，表現 `mode(z)` は文字列 `complex` であり，`length(z)` は `100` になる．

同じく，付値

```
> mode(z) <- "numeric"
```

はベクトルを複素数部分を捨て，実数部分だけを残した数値ベクトルに強制変換する．さらに付値

```
> length(z) <- 10
```

は，最初の 10 個の値だけに縮小し，そして

[4]「数値」モードは実際には二つの異なったモード．つまり「整数」と「倍精度実数」の混ざったものである．

```
> length(z) <- 100
```

は，NA で埋めることにより，長さ 100 の構造に伸長する．最後の付値

```
> mode(z) <- "character"
```

は，それを数値ではなく，それらが印字されたときに取るであろうような文字列からなるベクトルに強制変換する．(値 NA は文字列 ‘‘NA’’ に変換される．)

R は，そうすることが意味があると判断されるようなほとんどあらゆるところで，モードの変換を提供する．例えば

```
> z <- 0:9
```

にたいし，

```
> digits <- as.character(z)
```

とすれば，digits は文字列ベクトル ("0", "1", "2", ..., "9") にになる．今一度の強制変換，つまりモードの変換，は数値ベクトルを再び復元する：

```
> d <- as.numeric(digits)
```

今や d と z は同じ[5]一つのモードから別のモードへの強制変換や，あるオブジェクトに，それがまだ持っていない他のモードを付与するために as.something() という形の多くの関数がある．それらになじみになるためには，ヘルプファイルを調べてみよ．

2.2 オブジェクトの長さの変更

「空」のオブジェクトもモードを持っているかも知れない．例えば，

```
> e <- numeric()
```

は e を数値モードの空のベクトルにする．同様に character() は空の文字ベクトルとなる等々．いったん任意の長さのオブジェクトが作られると，新しい要素を，単に以前の範囲の外部の添字値を与えることにより，追加することができる．だから

```
> e[3] <- 17
```

は e を長さ 3 のベクトル（この時点で，最初の二つの値はともに NA）にする．これは，もし追加される要素達のモードが，最初のオブジェクトのモードと一致するならば，任意の構造に適用できる．

この自動的なオブジェクトの長さの調整は，たとえば，入力のための scan() 関数において，しばしば使われる．(§5.2 を見よ．)

逆に，オブジェクトの長さを切り詰めるためには，単なる付値が必要になるだけである．したがって，もし alpha が長さ 10 のオブジェクトなら，

```
> alpha <- alpha[2 * 1:5]
```

[5]一般に，数値から文字への強制変換とその逆は，文字列への変換における丸め誤差のせいで，正確には可逆ではない．

により，それは最初に偶数の添字を持っていた要素だけからなる，長さ 5 のオブジェクトになる．もちろん，古い添字は保存されない．

もし，数値もしくは複素数値ベクトルに *dim* 属性を与えると，後で見るように，それを多次元配列に変換することができる．

2.3 attributes() と attr()

関数 attributes(*object*) は，そのオブジェクトにたいし現在定義されているすべての非本質的属性のリストを与える．関数 attr(*object*,*name*) は特定の属性を選ぶのに使うことができる．これらの関数が使われることは稀で，例外は，たとえばある R のオブジェクトにそれが作られた日時や演算子を関連づけるといった特別な目的のために，ある新しい属性が作られた場合である．しかし，この概念は極めて重要である．

属性は R のオブジェクトシステムの統合された一部分であり，それらを付加したり除去する際には注意しなければならない．

attr() 関数をある付値の左辺に用いた場合，それは *object* に新しい属性を付け加えたり，すでに存在している属性を変更することができる．例えば

```
> attr(z,"dim") <- c(10,10)
```

は， R が z を 10×10 行列であるかのように扱うことを可能にする．

2.4 オブジェクトの「クラス」(*class*)

オブジェクトの「クラス」(*class*) として知られている特別な属性は，R でオブジェクト指向スタイルのプログラミングを可能にするために使われる．

例えばあるオブジェクトが data.frame クラスを持っていると，それはある方式で表示され plot() 関数はそれをある方式でグラフィック表示し，summary() 関数といった他の総称的な関数は，引数のクラス属性を考慮した方式で作用する．

一時的にクラスの効果を除くには unclass() 関数を使う．例えば，winter がクラス data.frame を持つなら，

```
> winter
```

とすると，行列にかなり似たデータフレームの書式で表示するだろうし，他方で

```
> unclass(winter)
```

とすると，普通のリストとして表示するであろう．かなり特別な状況でのみこのような機能を使う必要が起こるが，その一つは読者がクラスと総称的な関数というアイデアを受け入れ始めた時である．

総称的な関数やクラスについては §7.8 で議論されるが，しかし単に概略のみである．

3 順序が付いた因子と順序の無い因子

「因数」とは，同じ長さを持つ別のベクトルの要素の離散的分類を指定するベクトルオブジェクトである．R は「順序が付いた」因子と「順序の無い」因子の双方を扱うことができる．

3.1 一つの例

例えば，全ての州と準州[6]からの 30 人の公認会計士の標本があるとし，それらの個々の本拠地の所在州が，州名の省略形の文字列によって

```
> state <- c("tas", "sa",  "qld", "nsw", "nsw", "nt",  "wa",  "wa",
             "qld", "vic", "nsw", "vic", "qld", "qld", "sa",  "tas",
             "sa",  "nt",  "wa",  "vic", "qld", "nsw", "nsw", "wa",
             "sa",  "act", "nsw", "vic", "vic", "act")
```

特定されているとする．文字ベクトルの場合「ソート」とはアルファベット順にソートすることを意味することを注意して欲しい．

「因子」は同様に `factor()` 関数によって作ることができる．

```
> statef <- factor(state)
```

`print()` 関数は因子を他のオブジェクトとは少々違った仕方で扱う：

```
> statef
 [1] tas sa  qld nsw nsw nt  wa  wa  qld vic nsw vic qld qld sa
[16] tas sa  nt  wa vic qld nsw nsw wa  sa  act nsw vic vic act
```

因子の水準を知るには，`levels()` 関数を使うことができる．

```
> levels(statef)
[1] "act" "nsw" "nt"  "qld" "sa"  "tas" "vic" "wa"
```

3.2 関数 tapply() と不揃いの配列

直前の例を続け，同じ会計士の収入が（適当な単位で）別のベクトルに与えられているとしよう

```
> incomes <- c(60, 49, 40, 61, 64, 60, 59, 54, 62, 69, 70, 42, 56,
               61, 61, 61, 58, 51, 48, 65, 49, 49, 41, 48, 52, 46,
               59, 46, 58, 43)
```

各州毎の収入の標本平均を計算するには，ここで特別な関数 `apply()` を使うことができる：

```
> incmeans <- tapply(incomes, statef, mean)
```

[6]外国の読者は，オーストラリアには八つの州と準州，つまり the Australian Capital Territory, New South Wales, the Northern Territory, Queensland, South Australia, Tasmania, Victoria そして Western Australia があることを注意して欲しい．

結果は水準をラベルに持つ要素からなる平均のベクトル

```
> incmeans
  act    nsw   nt  qld sa  tas vic    wa
 44.5 57.333 55.5 53.6 55 60.5  56 52.25
```

である．

関数 taaply() はある関数（ここでは mean()）を，最初の引数（ここでは incomes）の成分の，二番目の引数（ここでは statef）の水準で定義される各グループに，あたかもそれらが別個のベクトル構造であるかのように，適用する．結果は因子の水準属性と同じ長さを持つ構造である．詳細に付いてはヘルプ文章を参照されたい．

更に州ごとの平均収入に対する標準誤差を計算する必要があるとする．このためには，与えられたベクトルに対し，標準誤差を計算する R の関数を書く必要がある関数についてはこの文書のもっと後で詳しく述べるが，標本分散を求める組み込み関数 var() があるので，この関数はとても簡単に一行で書け，次のような付値になる：

```
> stderr <- function(x) sqrt(var(x)/length(x))
```

(関数の書き方に付いては後の §7 で考えられる.）この付値を行えば，標準誤差は次のように計算される

```
> incster <- tapply(incomes, statef, stderr)
```

そして計算された値は

```
> incster
 act    nsw  nt    qld     sa tas   vic     wa
 1.5 4.3102 4.5 4.1061 2.7386 0.5 5.244 2.6575
```

となる．

演習問題として，州毎の平均収入に対する 95% 信頼区間を計算してみることができるだろう．このためには，標本サイズを見出すため，再び apply() 関数を length() 関数とともに使用し，適当な $t-$ 分布のパーセント点を見出すために，関数 qt() を使用することができるだろう．

tapply() 関数は，複数のカテゴリーによって分類されたベクトルの，より複雑な添字の操作を処理できる．例えば，会計士を州と性別によって分割したいとする．この簡単な例では，しかしながら，何が起こるかは次のように考えることができる．ベクトル中の値は，カテゴリー中の異なった項目に対応するグループに選別される．次に，これらのグループのそれぞれに対して個別に関数が適用される．結果の値は，関数値からなるベクトルであり，カテゴリーの水準属性によりラベルが付けられている．

ベクトルと，ラベル因子またはカテゴリーの組合せは，サブクラスのサイズが不規則になる可能性があるため「不揃いの配列」(*ragged array*) と呼ばれるものの例になる．もしサブクラスのサイズが全て同じならば，次の章で見るように，添字操作は暗黙のうちに，そしてより効率的に行うことができる．

4 リスト，データフレームとその用法

4.1 リスト

R の「リスト (*list*)」とはオブジェクトの順序付けられた集まりからなるオブジェクトのことである．個々の成分オブジェクトは，その「コンポネント (component)」と呼ばれる．

コンポネントは同じモード・型である必要は特に無く，例えば一つのリストは数値ベクトル，論理ベクトル，行列，複素数ベクトル，文字配列，関数，その他から成り立っていても良い．次はリストを作る簡単な例である：

```
> Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))
```

コンポネントは常に「順番が付けられ」，常にそれで参照出来る．したがって，もし `Lst` が 4 つのコンポネントを持つリストならば，これらは個別に `Lst[[1]]`, `Lst[[2]]`, `Lst[[3]]`, `Lst[[4]]` と参照することが出来る．もし更に `Lst[[4]]` がベクトルなら `Lst[[4]][1]` はその最初の項目である．

もし `Lst` がリストならば，関数 `length(Lst)` はそれが含む（トップレベルの）コンポネントの個数を与える．

リストのコンポネントには「名前」を付けることが出来，その場合はそのコンポネントは，二重鍵括弧の中に順番の代わりにコンポネントの名前を文字列で与えたり，より簡単には次の形の表現で参照することが可能である

```
> name$component_name
```

これはもし番号を忘れても正しいコンポネントを得ることの出来る非常に便利な記法である．

したがって上に挙げた簡単な例では

`Lst$name` は `Lst[[1]]` と同じであり，文字列 `"Fred"` となる，

`Lst$wife` は `Lst[[2]]` と同じであり，文字列 `"Mary"` となる，

`Lst$child.ages[1]` は `Lst[[4]][1]` と同じであり，数 4 になる．

`Lst[[1]]` と `Lst[1]` を区別することは非常に重要である．‘‘`[[...]]`’’ は一つの要素を抜き出す演算子であり，他方で ‘‘`[...]`’’ は一般的な添字指定演算子である．したがって，前者は「リスト `Lst` 中の最初のオブジェクト」であり，もし名前付のリストであってもその名前は含まれない．後者は「リスト `Lst` の最初の項目のみからなるサブリスト」である．もしそれが名前付のリストならば，該当する名前がサブリストに伝わる．

コンポネントの名前はそれらを一意的に特定するのに必要な最小の数の文字列に省略することが出来る．したがって `Lst$coefficients` は `Lst$coe` で，`Lst$covariance` は `Lst$cov` で最も短く特定することが出来る．

名前のベクトルは実際には他と同様にリストの属性であり，そうしたものとして扱うことが出来る．リスト以外の構造も勿論同様に「名前」属性を与えることが出来る．

4.2 リストの生成と変形

新しいリストは，すでに存在するオブジェクトから関数 `list()` で生成することが出来る．次の形式の付値

`> Lst <-` `list(`$name_1$`=`$object_1$`,` $name_2$`=`$object_2$`,` `...,` $name_m$`=`$object_m$`)`

は m 個のオブジェクト $object_1, \ldots, object_m$ をコンポネントに持つリスト `Lst` を生成し，各オブジェクトの名前を名前ベクトル（自由に選べる）として持つ．もしこれらの名前が無ければコンポネントは単に順序付けられるだけである．リストを形成するコンポネントは新しいリストを作る際に「コピー」され，オリジナルは変化しない．

添字を持つ他のすべてのオブジェクトと同様に，リストは付加的なコンポネントを特定することにより拡張できる．例をあげれば

```
> Lst[5] <- list(matrix=Mat)
```

4.2.1 リストの連結

連結関数 `c()` がリストの引数を与えられると，結果はリストモードのオブジェクトになり，そのコンポネントは引数リストのそれらが順につなげられたものになる．

```
> list.ABC <- c(list.A, list.B, list.C)
```

ベクトルオブジェクトを引数にとると，連結関数は同様にすべての引数を，単一のベクトル構造につなぎ合わせたことを思いだそう．この場合 dim 属性等，他のすべての属性は捨てられる．

4.3 リストを結果として返す関数

R の関数や表現は結果として単一のオブジェクトを返さなければならない．結果が複数のコンポネント部分からなるような場合，普通の形式は名前付のコンポネントを持つリストを返すことである．

4.3.1 固有値と固有ベクトル

関数 `eigen(Sm)` は対称行列 `Sm` の固有値と固有ベクトルを計算する．この関数の結果は `values` と `vectors` という名前のコンポネントを持つリストである．付値

```
> ev <- eigen(Sm)
```

はこのリストを `ev` に代入する．したがって `ev$val` は `Sm` の固有値ベクトル，`ev$vec` は対応する固有ベクトルの行列である．もし固有値だけが必要なら付値

```
> evals <- eigen(Sm)$values
```

により `evals` には固有値が入り，固有ベクトルは捨てられる．もし表現

```
> eigen(Sm)
```

が命令自身として使われるなら，二つのコンポネントが名前とともに端末に表示されるであろう．

4.3.2 特異値分解と行列式

関数 svd(M) は任意の行列引数 M を取り，M の特異値分解を計算する．これは M と同じ列空間（列ベクトルの張る線形空間）を持ち列ベクトルが互いに直交する行列 U と，M と同じ行空間（行ベクトルの張る線形空間）を持ち行ベクトルが互いに直交する二番目の行列 V と，M = U %*% D %*% t(V) を満たす正の対角成分をもつ対角行列 D から成る．D は実際には対角成分からなるベクトルとして返される．svd(M) の結果は実際には（明らかな意味を持つ）名前 d, u, v を持つ三つのコンポネントのリストである．

もし M が実際は正方行列ならば

```
> absdetM <- prod(svd(M)$d)
```

は M の行列式の絶対になることは容易にわかる．もしこの計算がさまざまな行列に対し頻繁に必要になるならば R の関数

```
> absdet <- function(M) prod(svd(M)$d)
```

を定義すれば，absdet() を丁度別の R の関数であるかのように使うことが出来るであろう．更に自明であるがもしかすると役にたつかも知れない例として，正方行列のトレースを計算する関数，例えば tr() を定義したくなるかも知れない．[ヒント：あからさまなループを使う必要はない．diag() 関数をもう一度見よ．]

関数についてはこのノートの先で正式に議論される．

4.3.3 最小自乗法と QR 分解

関数 lsfit() は最小自乗法による当てはめの結果を与えるリストを返す．次のような付値

```
> ans <- lsfit(X, y)
```

は最小自乗法による当てはめの結果を与える，ここで y は観測値ベクトルであり，X は計画行列である．詳細についてはヘルプ機能，特に回帰診断に関しては何よりも支援関数である diag()，を見よ．回帰の定数項は自動的に補われるため X の列ベクトルとして明示的に入れる必要は無いことを注意せよ．

もう一つの密接に関係した関数は qr() とその同類である．次の付値を考えよう

```
> Xplus <- qr(X)

> b <- qr.coef(Xplus, y)

> fit <- qr.fitted(Xplus, y)

> res <- qr.resid(Xplus, y)
```

これらは y の X の値域の上への直交射影を fit 中に，直交補空間の上への射影を res 中に，係数ベクトルを b 中に計算する，つまり b は本質的に MATLAB の ‘backslash’ 演算子の結果となる．

X の列がフルランクであることは仮定されていない．冗長性はもし存在すれば検出され除去される．

このやり方は最小自乗法の計算を行なう旧式で低レベルの方法である．ある場合には依然として有効であるが，現在では一般的に §8 で議論される統計モデル機能が代わりに使用されるであろう．

4.4 データフレーム

「データフレーム」とは *data.frame* クラスを持つリストのことである．データフレームに入れることの出来るリストには幾つかの制限がある，つまり

- コンポネントはベクトル（数値，文字，もしくは論理値），因子，数値行列，リスト，もしくは他のデータフレームに限られる，
- 行列，リスト，そしてデータフレームは，それらがそれぞれ持つ列，要素，変数と同じ数の変数を新しいデータフレームに付け加える，
- 数値ベクトルと因子はそのままの状態で含まれ，非数値ベクトルは因子に強制変換される．その水準はベクトルに現れるユニークな値である，
- データフレームに変数として現れるベクトル構造はすべて同じ「長さ」を，行列構造は同じ「サイズ」を持たなければならない，

データフレームは多くの点で，異なったモードや属性を持ち得る列を持つ行列と見做すことが出来る．それは行列形式で表示できるし，その行や列は行列のインデックス形式で抜き出すことが出来る．

4.4.1 データフレームの作成

データフレームの列（コンポネント）に対する要請を満たすオブジェクトは，関数 data.frame を用いてデータフレームの作成に用いることが出来る．

```
> accountants <- data.frame(home=statef, loot=income, shot=incomef)
```

リストのコンポネントがデータフレームの要請を満たせば，関数 as.data.frame() を用いてデータフレームに強制変換出来る．

データフレームを新規に作る最も簡単な方法は read.table() 関数を用いて外部のファイルからデータフレームの全体を読み込むことである．

4.4.2 関数 attach() と detach()

リストのコンポーネントに対する accountants$statef といった $-記法はいつでも便利というわけではない．有用な機能はリストやデータフレームのコンポネントを，毎回リスト名を明示的に引用する必要なしに，コンポネント名の下に変数として一時的に目に見えるようにすることであろう・

関数 attach() は，その引数と同じ名前を持つディレクトリーを用意することにより，データフレームを作ることが出来る．例えば lentils が三つの変量 lentils$u. lentils$v, lentils$w を持つデータフレームとしよう.（検索リストへの）追加

```
> attach(lentils)
```

はこのデータフレームを探索リストの第二番目の位置に置き，探索リストの第一番目の位置に変数 u, v, w が無いという条件の下で，u, v, w をデータフレーム lentils の三つの変量名として使うこと使うことができるようになる．ここに於いて次の付値

```
> u <- v+w
```

はデータフレームのコンポネント u を置き換えずに，むしろ探索リストの第一位置の作業領域にある別の変数 u からそれを隠蔽する．データフレーム自身に永続的な変更を加えたければ，最も単純な方法はもう一度 $-記法に戻ることである

```
> lentils$u <- v+w
```

しかしながらコンポネント u の新しい値は（検索リストから）外し，もう一度追加するまで見えるようにはならない．データフレームを（検索リストから）外すには関数

```
> detach()
```

を用いる．

より正確には，この文は現在第二位置にあるものを検索リストから外す．従って，現在の例では，lentils$u といったリスト記法を用いない限り，u, v, w は最早見ることは出来なくなる．

ノート：現在の R のリリースでは検索リストは最大で 20 の項目を含むことが出来る．同じデータフレームを重ねて追加しないようにしよう．変量の使用が終ったらすぐにデータフレームを外そう．

ノート：現在の R のリリースではリストやデータフレームは（検索リストの）第二位置かそれ以上にしか追加出来ない．追加されたリストやデータフレームに直接付値することは出来ない（このように，ある程度それらは静的である）．この点は来年かそこらに恐らく変更されるであろう．

4.4.3 データフレームを用いた作業

同じ作業ディレクトリーで多くの異なった問題を快適に処理すること出来るようにする有用な指針は

- 適切に定義され分離された問題に対するすべての変量を，一つのデータフレームに集積し，それに適当な意味が明瞭な名前をつけよ；
- ある問題を処理する時は，適当なデータを検索リストの第二位置に追加し，第一位置を作業用や一時的な変数のために使え；
- 問題を終了する前に，将来の参考用に保存しておきたいすべての変量を $-記法を用いてデータフレームに追加し，それから detach() せよ；
- 最後にすべての不要な変量を作業ディレクトリーから消し去り，忘れ残しの一時的変量がないように可能な限りきれいにしておけ.

このようにすれば，例えば同じディレクトリーで，どれもが x, y, z という変数名を持つ多くの問題を処理することは極めて簡単になる

4.4.4 任意のリストの追加

attach() は単にディレクトリーやデータフレームを検索リストに登録するだけでなく，他のオブジェクトのクラスの追加も出来る一般的な関数である．特にリストのモードの任意のオブジェクトは同じように追加出来る：

```
> attach(any.old.list)
```

同様に，クラス prframe のオブジェクトを，非線形回帰やその他で必要になる，いわゆる *parametrized data frame* に追加することも出来る．

一般的な関数であるがゆえに，必要があれば，更に多くのオブジェクトのクラスを追加するメソッドを加えることも可能である．

5 ファイルからのデータの読み込み

大規模なデータオブジェクトは，普通 R のセッション中にキーボードから入力されるよりは，外部ファイルから値として読み込まれる．R の入力機能は単純で，その機能はかなり厳格で融通がきかない．R の設計者は，使用者が入力ファイルをエディターや perl を用いて， R の要求を満たすように入力ファイルを変形[7]できるであろうことを当然の前提としている．普通これは非常に単純である，

しかしながら，固定長で分離されていない入力ファイルを，分離された欄を持つフアィルに変換するために関数 make.filed() を用いることが出来る．又，そのようなファイルの各行にある欄の数を数える count.fields() という道具がある．非常に単純な変換や問題をチェックするためには，しばしばこれらが目的に対し適当であるが，多くの場合には，R のセッションが始まる前に予備的な土均しをすることが好ましい．

もし変量が（我々が強く勧めるように）主にデータフレームに収められるなら，それらはデータフレームの全体が read.table() 関数を用いて直接読み込めるようになっているべきである．又よりプリミティブな入力関数 scan() があり，直接呼び出すことが出来る．

5.1 read.table() 関数

データフレームの全体を直接読み込むためには，外部ファイルは普通特別な形を持つ．

- ファイルの第一行はデータフレーム中の各変量に対する「名前」を含むべきである，
- ファイルの各追加行は最初の項目である「行ラベル」と各変量に対する値を持つ．

もしファイルが最初の行に，第二行よりも一つ少ない項目を持つなら，この配置は意味のあるものと見なされる．データフレームとして読み込むべきファイルの最初の数行は，例えば Figure 1 のようになるかもしれない．

```
     Price    Floor     Area   Rooms     Age  Cent.heat
01   52.00    111.0      830     5       6.2      no
02   54.75    128.0      710     5       7.5      no
03   57.50    101.0     1000     5       4.2      no
04   57.50    131.0      690     6       8.8      no
05   59.75     93.0      900     5       1.9     yes
...
```

図 1: 名前と行ラベルを持つ入力ファイルの書式

デフォールトとして（行ラベルを除くと）数値項目は数値変量として読み込まれ，上の例の Cent.heat のような非数値変量は因子として読み込まれる．必要ならこれは変更できる．

関数 read.table() はデータフレームを直接読み込むために使うことが出来る．

```
> HousePrice <- read.table("houses.data")
```

[7] Unix では sed や tt awk を用いることが出来る．

しばしば行ラベルを含めることをやめ，既定のラベルを使いたいかも知れない．こうした場合，ファイルは行ラベルの列を次の Figure 2 のように省略出来る：

```
   Price    Floor     Area   Rooms     Age  Cent.heat
   52.00    111.0      830     5       6.2      no
   54.75    128.0      710     5       7.5      no
   57.50    101.0     1000     5       4.2      no
   57.50    131.0      690     6       8.8      no
   59.75     93.0      900     5       1.9     yes
...
```

図 2: 行ラベルを持たない入力フォーム

そうするとデータフレームは次のように読み込むことが出来る：

```
> HousePrice <- read.table("houses.data", header=T)
```

ここでオプション heading=T は最初の行はヘディングの行であること，従って，ファイルの書式の含意から，明白な行ラベルは与えられていないことを指定している．

5.2 scan() 関数

データベクトルが同じ長さで，並列的に読み込まれるべきであると仮定しよう．更に，三つのベクトルがあり，最初はモードを表す文字で残りの二つはモードの数値とし，ファイルは input.dat とする．最初のステップは scan() を用い，この三つのベクトルをリストとして次のように読み込むことである：

```
> in <- scan("input.dat", list("",0,0))
```

二つ目の引数は，読み込むべき三つのベクトルのモードを確立するための，ダミーのリストである．in に収められた結果は，読み込まれた三つのベクトルをコンポネントに持つリストである．データ項目を三つの分離されたベクトルに分離するためには，次のような付値を使う：

```
> label <- in[[1]]; x <- in[[2]]; y <- in[[3]]
```

より簡便に，ダミーリストは名前付のコンポネントを持つことが出来，その時はその名前を読み込まれるベクトルにアクセスするために使うことが出来る：

```
> in <- scan("input.dat", list(id="", x=0, y=0))
```

もし変量に個々にアクセスしたければ，それらをワーキングフレーム中で変量に再付値するか，

```
> label <- in$id; x <- in$x; y <- in$y
```

リストを探索リストの第二位置に置く（§4.4.4 を見よ）．

もし第二引数が単一の変数でリストでなければ，単独のベクトルが読み込まれ，そのすべてのコンポネントはダミー変数と同じモードでなければならない．

```
> X <- matrix(scan("light.dat", 0), ncol=5, byrow=T)
```

より洗練された読み込み機能があり，このマニュアルの中で詳しく述べられるであろう．

5.3 他の機能，データの編集

データセットが一旦読み込まれれば，R には多少の修正を行なうウィンドウ形式の機能がある．命令

```
> xnew <- data.entry(xold)
```

はあなたのデータセット xold を，独立したウィンドウ上の表計算形式の環境を用いて編集することを可能にし，終了時には修正されたオブジェクトは xnew に付値される．xold，そして xnew は行列，ベクトル，データフレーム，もしくは原始的なデータオブジェクトで良い．

data.entry() を引数無しで呼び出すと

```
> xnew <- data.entry()
```

新しいデータを表計算形式のインターフェイスを用いて入力することが出来る．

6 その他の言語仕様．ループと条件実行

6.1 グループ表現

すべての命令が，値を返す関数や式であるという点において R は式 (expression) 言語といえる．変数への付値ですら，付値された値を返すような式であり，式が使えるところならどこにでも記述することができる．特に多重の付値も可能である．

命令は括弧でグループ化することができる．たとえば，{expr$_1$; expr$_2$; ...; expr$_m$} という場合には，このグループ式の値は，グループの中で最後に評価された式の値となる．こういったグループもやっぱり式だから，たとえばグループ自体を括弧でくくって，もっと大きな式の一部として使うこともできる等々．

6.2 制御文

6.2.1 条件実行: if 文

次のような条件構造が可能である．

> if（$expr_1$）$expr_2$ else $expr_3$

ここで $expr_1$ は，必ず論理値として評価できねばならず，その時，式全体の結果は明白である．

6.2.2 繰り返し実行: for ループ、repeat 文と while 文

また次の形を持つ for–ループ構造がある．

> for（$name$）in $expr_1$）$expr_2$

ここで $name$ はループ変数である．expr$_1$ はベクトル式（1:20 のような数列が使われることが多い），expr$_2$ は普通ダミー引数 $name$ を使って書かれた副式を持つ，グループ化された式である．$name$ が expr$_1$ の結果であるベクトル中の値を一つずつ取るたびに expr$_2$ が繰り返し評価される．

たとえば ind がクラスを指定するベクトルとし，各クラス内での y の x に対するプロットを，個別に作りたいとしよう．一つの可能性は，あとで説明する coplot() 関数を使うことであり，これは因子の各水準に対応するプロットからなる配列を作成する．もう一つの方法は，すべてのプロットを 1 つの画面に表示するもので，次のようになる：

```
> yc <- split(y, ind); xc <- split(x, ind)

> for (i in 1:length(yc)){plot(xc[[i]], yc[[i]]);

    abline(lsfit(xc[[i]], yc[[i]]))}
```

（split() 関数は，あるカテゴリーによって指定されたクラスに基づいて，より大きなベクトルを分割してできるベクトルからなるリストを作成する．これは便利な関数で，ほとんどの場合箱型図と組み合わせて用いられる．より詳しくは help 機能を参照してほしい.）

警告： `for()` ループを使うと，式の評価が比較的遅くなってしまう．R の `for()` ループは S のものよりも速い傾向があるが，それでも可能ならば使用を避けるほうが良い．`apply()`，`tapply()`，`sapply()`，そしてその他の多くの関数は，何よりも `for()` ループのあからさまな使用を避けるために用意されている．

この他の繰り返し機能としては

`> repeat` *expr*

や

`> while` (*condition*) *expr*

がある．

`break` 文は，どんなループであってもそれを中断することができるが，異常終了になる可能性がある．これは `repeat` ループを中断する唯一の手段である．

`next` は特定のサイクルを中断し「次の」のサイクルへスキップさせるのに使うことができる．

制御文はほとんどの場合 §7 の章で扱う「関数」と組み合わせて用いられ，そこでより多くの例が登場するだろう．

7 自分自身の関数を書く

これまでに非公式に見てきたように，R 言語はユーザーが「関数（function）」モードのオブジェクトを作ることを許す。これらは特別な内部形式で保管され，他の表現で使うことが出来る．こうすることにより，R 言語は著しく力強さ，便利さ，優美さを増し，有用な関数を書くことを学ぶことは R の使用を快適かつ生産的にする主要な方法の一つである．

R システムの一部として提供される `mean()`, `var()`, `postscript()` といった関数のほとんどは，それ自身 R で書かれており，従ってユーザーが定義した関数と実態としては差がないことを強調しておこう．

一つの関数は次のような付値で定義される

`> name <- function(`arg_1`,` arg_2`, ...)` *expression*

ここで *expression* は引数 arg_i を用い，ある値を計算する R の表現（普通グループ化された表現）である．この表現の値は関数の返り値になる．

関数の呼び出しは普通 `name(`$expr_1$`,` $expr_2$`, ...)` の形を取り，関数呼び出しが許されるあらゆる所に登場することが出来る．

7.1 簡単な例

最初の例として二標本 t-統計量を計算する（細部がすべて見える）関数を考えよう．勿論，同じことをもっと簡単に行なう別の方法もあるから，これは人工的な例である．

関数は次のように定義される：

```
> twosam <- function(y1, y2) {
    n1  <- length(y1); n2  <- length(y2)
    yb1 <- mean(y1);   yb2 <- mean(y2)
    s1  <- var(y1);    s2  <- var(y2)
    s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
    tst <- (yb1 - yb2)/sqrt(s2*(1/n1 + 1/n2))
    tst
  }
```

もしこの関数を定義すれば，二標本 t-検定は次のような呼び出しで実行できるであろう

```
> tstat <- twosam(data$male, data$female); tstat
```

二番目の例として，Matlab の backslash 命令を直接エミュレートする関数を考えてみよう．これは，行列 X の列ベクトルの張る空間への，ベクトル y の直交射影の係数を返すものである．（これは回帰係数の最小自乗推定と普通呼ばれる．）これは通常 `qr()` 関数を用いて行なわれるのだろうが，これは直接使うのはしばしば少々技巧的に過ぎ，安全に使うために次のような簡単な関数を用意するのが割にあうであろう．

ベクトル $y^{n\times 1}$ と行列 $X^{n\times p}$ を与えた時,

$$X\backslash y =^{\text{def.}} (X'X)^{-}X'y$$

と定義される．ここで $(X'X)^{-}$ は $X'X$ の一般化逆行列である．

このオブジェクトが作られれば，それは永続的であり，すべてのオブジェクトと同様に，次のような文等で使うことが出来る

```
> regcoeff <- bslash(Xmat, yvar)
```

R の古典的な関数 `lsfit()` はこの作業，及びそれ以上のこと[8]を，極めてうまく行なう．それは逆に関数 `qr()` と `qr.coef()` を，上述した部分の計算を行なうために，上のような少々直観に反する仕方で使っている．従って，こうしたことが頻繁に使われるようならば，この部分を使いやすい単純な形に独立させることには，恐らく多少の価値があるであろう．もしそうならば，いっそう使いやすくするために，行列に対する二項演算にしたくなるかも知れない．

7.2 新しい二項演算の定義

もし `bslash()` 関数が別の名前，つまり次のような形

$$\%anything\%$$

を持てば，それを表現の中で関数形で無く「二項演算子」として使うことも出来る．例えば，もし真中の文字として `!` を使ったとする．関数の定義は

```
> "%!%" <- function(X, y) {... }
```

のように始まるであろう（引用マークに使用に注意）．この関数は `X%! y` の様に使うことが出来る.（バックスラッシュ記号そのものの使用は，この文脈では特別な問題を起こすので，都合の良い選択ではない.）

行列の積演算子 `%*%` や，行列の外積演算子 `%o%` はこのようにして定義された二項演算子の別の例である．

7.3 名前付引数と既定値．“...”

§1.3 で最初に注意したように，もし呼び出された関数への引数が “*name*=*object*” の形で与えられるなら，それらの順序は任意である．更に，引数列は名前が無く，位置順序に依存する形で始まることが出来るが，その際は名前付引数は位置依存の引数列の後に置く．

従って，もし関数 `fun1` が

```
> fun1 <- function(data, data.frame, graph, limit) {[関数本体は省略]}
```

と定義されたとする．

そうすると，関数は色々な方法で呼び出すことが出来る．例えば

[8] モデルの章で説明される方法を参照せよ．

```
> ans <- fun1(d, df, 20, T)
```

```
> ans <- fun1(d, df, graph=T, limit=20)
```

```
> ans <- fun1(data=d, limit=20, graph=T, data.frame=df)
```

は皆同じことである．

多くの場合，引数には常識的に適当な既定値を与えることが出来る．この時，既定値でよければそれらは関数の呼び出しからすべて省略することが出来る．例えば，もし関数 fun1 が

```
> fun1 <- function(data, data.frame, graph=T, limit=20) {...[8]}
```

と定義されたとすると，それは上の三つの例と同値な

```
> ans <- fun1(d, df)
```

という形で呼び出すことも出来るし，既定値の一つを変更する

```
> ans <- fun1(d, df, limit=10)
```

という風にも使うことが出来る．

重要な注意として，既定値は任意の表現で良く，同じ関数への他の引数を含むことすら可能であり，ここでの我々の簡単な例のように，定数に制限されることもないことを述べておく．

もう一つの頻繁に必要になる要請としては，関数の引数の設定を，他の関数に引き継がせることである．例えば，多くの描画関数は関数 par() を利用し，plot() の様な関数はユーザーがグラフィックス出力を制御するために，作図パラメーターを par() に渡すことを可能にする．(par() 関数についての詳細は §9.4.1 を見よ)．これは関数に追加引数（文字通りに “...” と表される）を含め，この引数を継承させることにより可能になる．図 3 に概略的な例をあげる．

```
fun1 <- function(data, data.frame, graph=T, limit=20, ...)  {

        [省略された文]

        if (graph)
              par(pch="*", ...)

        [新たな省略された文]
    }
```

図 3: 省略引数 “...” の用法

7.4 関数内の付値は局所的である．フレーム

「関数内で行なわれる全ての付値は局所的かつ一時的であり，関数から抜け出す時に失われる」ことを注意しよう．従って，付値 X <- qr(X) は呼び出しプログラム中の引数の値に影響しない．

R における付値のスコープを支配する完全な規則を理解するためには，評価の「フレーム (frame)」に詳しい必要がある．これは決して難解では無いが少々高等な話題であり，このノートではこれ以上触れることはしない．

もし関数中で，大局的で永続的な付値を行ないたいならば 'superassignment' 演算子 '<<-' を使うか，関数 `assign()` を使うことが出来る．詳細については `help` 文書を見よ．S-PLUS のユーザーは R では `<<-` が異なった意味を持つことを注意すべきである．これらは §7.6 でさらに議論される．

7.5 より進んだ例

7.5.1 ブロック計画における効率因子

平凡かも知れないがより完全な関数の例として，ブロック実験計画の効率因子を見い出すことを考えよう．(この問題については既に §**??**で触れた．)

ブロックデザインは二つの因子，例えば `blocks` （b 水準）と `varieties` （v 水準）からなる．もし繰り返し行列 $R^{v\times v}$，ブロックサイズ行列 $K^{b\times b}$，そして一致行列 $N^{b\times v}$ が与えられると，効率因子は次の行列の固有値として定義される

$$E = I_v - R^{-1/2}N'K^{-1}NR^{-1/2} = I_v - A'A$$

ここで $A = K^{-1/2}NR^{-1/2}$ である．この関数を書く一つの方法が図 4 に与えられている．

```
> bdeff <- function(blocks, varieties) {
         blocks <- as.factor(blocks)              # 安全のため仮に移動
         b <- length(levels(blocks))
         varieties <- as.factor(varieties)        # 安全のため仮に移動
         v <- length(levels(varieties))
         K <- as.vector(table(blocks))            # 次元属性の除去
         R <- as.vector(table(varieties))         # 次元属性の除去
         N <- table(blocks, varieties)
         A <- 1/sqrt(K) * N * rep(1/sqrt(R), rep(b, v))
         sv <- svd(A)
         list(eff=1 - sv$d^2, blockcv=sv$u, varietycv=sv$v)
  }
```

図 4: ブロックデザインの効率を計算する関数

この場合，固有値ルーチンよりも特異値分解を使う方が数値的に少し好ましい．

この関数の結果は，最初のコンポネントである効率因子だけでなく，`block` と `variety` に対する canonical contrast も与えるリストである．なぜなら，これらは付随的で有益な定量的情報だからである．

7.5.2 表示される配列中のすべての名前を取り去る

大きな行列や配列を表示するためには，配列の名前や数を取り去った閉じた形で表示することがしばしば有用である．*dimnames* 属性を取り除くことでこの効果を得ることは出来ず，むしろ配列に空の文字列からなる *dimnames* 属性を与える必要がある．例えば行列 X を表示するには

```
> temp <- X

> dimnames(temp) <- list(rep("", nrow(X)), rep("", ncol(X))

> temp; rm(temp)
```

とする．これをいっそう簡便に行なうには，同じ結果を達成する「一まとめにした」ものとしての，図 5 に示された関数 `no.dimnames()` を使う．これはまた，効率的で有用なユーザー定義関数が如何に簡潔であり得るかということを説明している．

```
no.dimnames <- function(a){
#
# Remove all dimension names from an array for compact printing.
#
        d <- list()
        l <- 0
        for(i in dim(a)) {
                d[[l <- l + 1]] <- rep("", i)
        }
        dimnames(a) <- d
        a
}
```

図 5: 配列をコンパクトな形で表示する関数

この関数を定義すれば，配列を閉じた形で次のように表示出来る

```
> no.dimnames(X)
```

これは値よりもパターンこそが興味の対象である，大きな整数値配列に対し特に有益である．

7.5.3 再帰的な数値積分

関数は再帰的であることができ，関数自身の中で関数を定義することが出来る．しかしながら，そうした関数，実際には変数，はもしそれらが検索リスト上にあった時にそうなるようには，より高い順位の評価フレームにある呼び出し関数により継承されない，ことを注意しよう．

図 6 にあげた例は一次元の数値積分を行なう素朴な方法である．被積分関数は積分範囲の両端と中央で評価される．もし台形を一つだけ使った台形公式の値が，台形を二つ使った台形公式の値と十分近ければ．後者が積分の値として返される．さもなければ同じプロセスが再帰的に同じパネルに対し適用される．結果は関数の評価が一次関数から遠い場所に集中する適応的な積分のプロセスで

ある．しかしながら，重いオーバーヘッドが存在し，被積分関数が十分滑らかであるとともに評価が極めて難しい時にのみ，他のアルゴリズムと肩を並べることが出来る．

この例は同時に R プログラミングにおける小さなパズルを与える．

```
area <- function(f, a, b, eps = 1.0e-06, lim = 10)
{
   fun1 <- function(f, a, b, fa, fb, a0, eps, lim, fun)
   {
      d <- (a + b)/2
      h <- (b - a)/4
      fd <- f(d)
      a1 <- h * (fa + fd)
      a2 <- h * (fd + fb)
      if(abs(a0 - a1 - a2) < eps || lim == 0)
         return(a1 + a2)
      else {
         return(fun(f, a, d, fa, fd, a1, eps, lim - 1, fun) +
                fun(f, d, b, fd, fb, a2, eps, lim - 1, fun))
      }
   }
   fa <- f(a)
   fb <- f(b)
   a0 <- ((fa + fb) * (b - a))/2
   fun1(f, a, b, fa, fb, a0, eps, lim, fun1)
}
```

図 6: 関数の内部で定義された再帰関数

7.6 スコープ

この節における議論はこの文書の他の部分よりも少々より技術的である．しかしながら，それは S-PLUS と R の主要な違いの一つを明らかにする．

関数の本体に現れる記号は三つのクラスに分けられる．形式的パラメータと局所変数と自由変数である．関数の形式的パラメータは関数の引数リストに現れるものである．これらの値は実際の関数引数を形式的パラメータに「結び付ける」プロセスにより決定される．局所変数は関数の本体中で表現の評価により決定される．形式的パラメータでも局所変数でも無いものは，自由変数と呼ばれる．自由変数はもしそれが付値されれば局所変数になる．次の関数定義を考えよう．

```
f<-function(x) {
      y<-2*x
      print(x)
      print(y)
```

```
    print(z)
}
```

この例で x は形式的パラメータ，y は局所変数，そして z は自由変数である．

R では自由変数の中身は，関数が作られた環境を最初に探すことにより決定される．先ず cube という名前の関数を定義しよう．

```
cube<-function(n){
    sq<-function() n*n
    n*sq()
}
```

関数 sq 中の変数 n はその関数への引数ではない．従ってスコープ規則を用いて，それに結びつけられるべき値を決めなければならない．静的なスコープの下では，値は n という名前の大局的変数に結び付いた値である．字句解析式のスコープの下では，それは関数 cube にあたえられたパラメータである．なぜなら，それが関数 sq が定義された時に変数 n に実際に結びつけられたものだからである．S-PLUS と R における評価の違いは，S-PLUS が n という名前の大局的変数を探すのに，R は関数 cube が起動された時作られる環境中で n という名前の変数を先ず探すことにある．

```
#first evaluation in S
S> cube(2)
Error in sq(): Object "n" not found
Dumped
S> n<-3
S> cube(2)
[1] 18
#then the same function evaluated in R
R> cube(2)
[1] 8
```

字句解析式のスコープはまた関数に「様々に変わり得る状態」を与えるのに使うことが出来る．次の例では R を用いていかに銀行勘定を物真似することが出来るかを示す．銀行勘定機能では，収支または合計を持ち，引き出し機能，預け入れ機能，そして現在の収支バランスを示す機能が必要になる．これを acount の中で三つの関数を生成し，それからそれらを含むリストを返すことにより達成する．account が呼び出されると，それは数値引数 total を取り，三つの関数を含むリストを返す．これらの関数は total を含む環境で定義されるので，その値を利用することが出来る．

特殊な付値演算子 <<- を用いて total に付随する値を変えることが出来る．この演算子はシンボル total を含む環境を包括する環境を振り返り，もしそうした環境があればその環境の中で total の値を右辺値で置き換える．もしシンボル total を見つけることなく，大局的もしくは最上位の環境に到達すれば，それに対する変数が作られ，それに右辺値が付値[9]される．<<- が別の関数の返り値である関数の中で使われた時にのみ，ここで説明された特別な行動が起こる．

[9] ある意味でこれは S-PLUS における行動を真似ている．なぜなら S-PLUS ではこの演算子は常に大局的変数を作り出すかそれに付値するからである．

```
open.account <- function(total) {

  list(
    deposit = function(amount) {
        if(amount <= 0)
            stop("Deposits must be positive!\n")
          total <<- total + amount
          cat(amount,"deposited. Your balance is", total, "\n\n")
    },
    withdraw = function(amount) {
        if(amount > total)
            stop("You don't have that much money!\n")
        total <<- total - amount
        cat(amount,"withdrawn.  Your balance is", total, "\n\n")
    },
    balance = function() {
        cat("Your balance is", total, "\n\n")
    }
  )
}

ross <- open.account(100)
robert <- open.account(200)

ross$withdraw(30)
ross$balance()
robert$balance()

ross$deposit(50)
ross$balance()
ross$withdraw(500)
```

図 7: 字句解析式のスコープを利用する関数

7.7 環境のカスタム化

ユーザーは自分の環境を幾つかの異なった方法でカスタマイズ出来る．システムの初期化ファイルがあり，全てのディレクトリーはそれ自身の初期化ファイルを持つことが出来る．特殊な関数 `.First` と `.Last` を利用することも出来る．

システムの初期化ファイルは `Rprofile` と呼ばれ，R のホームのサブディレクトリーである `library` 中にある．このファイルは R があなたのシステムで起動されるたびに実行したい命令を含むべきである．二つ目の個人的プロファイルのファイルである `.Rprofile` は任意のディレクトリーに置

く[10]ことが出来る．もし R がそのディレクトリーで起動されると，このファイルが初期設定用に読みとられる．このファイルは個々のユーザーに，彼らの作業空間の制御仕様を与え，異なった作業ディレクトリーでは異なった始動手順を持つことを許す．

上の二種類のプロファイルファイル中か .RData イメージ中の .First() という名前の任意の関数は特殊な挙動を持つ．それは R セッションの冒頭に自動的に実行され環境を初期化するのに用いることが出来る．例えば図 8 中の定義はプロンプトを $ に変え，他の様々なことがらをセットアップして，事後のセッション中で所与とする．

このようにファイルの実行順序は Rprofile, .Rprofile，そして .First() となる．後のファイル中の定義は事前のファイル中の定義を覆い隠す．

```
> .First <- function() {
      options(prompt="$ ", continue="+\t")        # $ is the prompt
      options(digits=5, length=999)    # custom numbers and printout
      options(gui="motif")         # default graphics user interface
      tek4014()                                 #  for terminal work
      par(pch = "+")                            # plotting character
      attach(paste(unix("echo $HOME"), "/.Data", sep = ""))
                                    # Home of my personal library
      library(examples)            # attach also the system examples
      }
```

図 8: .First() 関数の例

同様に .Last() 関数が定義されていれば，セッションのまさに最後に実行される．一つの例が図 9 に与えられている．

```
> .Last <- function() {
      graphics.off()                       # a small safety measure.
      cat(paste(unix("date"),"\nAdios\n")) # Is it time for lunch?
  }
```

図 9: .Last() 関数の例

7.8 クラス，総称的な関数，そしてオブジェクト指向性

あるオブジェクトのクラスは「総称的 (generic) な関数」として知られているものにより，それがどう処理されるかを決める．逆に総称的な関数は，引数自身のクラスに固有のある作業や行動を引数に対して行なう．もし引数が如何なるクラス属性も持たないか，問題になっている総称的な関数に対し特別にあつらえているのではないクラスを持つならば，常に「既定の動作」が用意されている。

例をあげるとわかりやすいであろう．クラスのメカニズムはユーザーが特別な目的のために，総称的な関数をデザインし書く機能を提供する．総称的な関数の例としては，オブジェクトをグラ

[10]従ってこのファイルは Unix の隠しファイルである．

フィックに表示する `plot()`，各種の解析の要約のための `summary()`，統計モデルの比較のための `anova()` がある．

あるクラスを特殊な方法で処理することが出来る総称的関数の数は極めて多い．例えば，クラス `data.frame` のオブジェクトにある流儀で役立つことの出来る関数には

[, [[¡-, any, as.matrix,
[¡-, model, plot, summary,

がある．現在の完全なリストは `methods()` 関数を使って得ることが出来る．

Sprompt methods(class=”data.frame”)

逆に，一つの総称的な関数が処理できるクラスの数も極めて多くなることがある．例えば，`plot()` 関数は次のような（おそらくそれ以上の）オブジェクトのクラスに対する変種を持つ：

data.frame, default, density, factor,

完全なリストは再び `methods()` 関数を用いて

```
> methods(plot)
```

とすれば得られる．このメカニズムに対する完全な議論については公式の参考文献を参照されたい．

8 R における統計モデル

この章では読者が統計学の方法論，特に回帰分析と分散分析にある程度なじみがあることを前提にしている．もっと後では，一般化線形モデルと非線形回帰について多少知識があるという，より注文の多い仮定をする．

広範囲な問題に応用される一般的な道具を作ることを可能にするために，統計モデルを当てはめることを可能にするために必要な要請は，十分明瞭に定義されている．1991 年の秋のリリース以来 R は統計モデルを極めて簡単に当てはめるために，一連の連係された機能を提供している．しかしながらそれらは，例えば Genstat における機能ほどは高レベルではない．特に R の一般的な方針に合わせて，結果の出力形式は最小限に限られている．

8.1 統計モデルの定義；公式

統計モデルに対する雛型 (template) は，独立で同型の誤差を持つ線形回帰モデル

$$y_i = \sum_{j=0}^{p} \beta_j x_{ij} + e_i, \qquad e_i \sim \mathrm{NID}(0, \sigma^2), \qquad i = 1, 2, \ldots, n$$

である．行列の言葉ではこれは

$$\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{e}$$

と書くことが出来る．ここで $\boldsymbol{y}$ は応答変数，$\mathbf{X}$ は「モデル行列」もしくは「計画行列」で非ランダムな列ベクトル $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, …, $\boldsymbol{x}_p$ を持つ．ごく普通には $\boldsymbol{x}_0$ は「切片項」を定義する，全て 1 からなる列ベクトルである．

例.

形式的な仕様を与える前に，幾つかの例をあげると分かりやすいであろう．

数値変数 `y`, `x`, `x0`, `x1`, `x2`, …，行列 `X`，因子 `A`, `B`, `C`, … を考える．以下の左側の公式は，右側に与えられた統計モデル[11]を特定する．

`y ~ x` `y ~ 1 + x`	両者はともに y の x への単純な線形回帰モデルを意味する．最初の式では切片項は明示されていず，第二式では明示されている．
`y ~ -1 + x` `y ~ x - 1`	原点を通る（つまり切片項を持たない） y の x への単純な線形回帰モデル．
`log(y) ~ x1 + x2`	変換された変量 $\log(y)$ の x_1 と x_2 への重回帰（暗黙の切片項を持つ）．
`y ~ poly(x,2)` `y ~ 1 + x + I(x^2)`	y の x への二次の多項式回帰．前者は基底関数として直交多項式を用い，後者は単なる巾乗式を用いる．
`y ~ X + poly(x,2)`	行列 X と x の二次式からなる計画行列を持つ y の重回帰．
`y ~ A`	A で定まるクラスを持つ y の一元配置の分散分析．

[11] 訳注：実験計画は最初 R.A. Fisher により農業実験用に開発され，その歴史を反映するいくつかの用語を持つ．全区画 (whole plot) とは基本的栽培条件が同じになるような一区切りの実験用圃場であり、特定の栽培条件（水準）を変えて栽培するためにそれをいくつかの副区画 (subplot) に分割する．これを分割区画実験 (split plot) と称する．各副区画にどの水準を割り当てるかを無作為に決めるやり方が乱塊法 (randomized block) である．

`y ~ A + x`	A で定まるクラスと，共変量 x を持つ y の一元配置の共分散分析.

`y ~ A*B`
`y ~ A + B + A:B`
`y ~ B %in% A`
`y ~ A/B`
	二つの因子 A と B を持つ y の二元配置の非加法的 (non-additive) モデル．最初の二つは同じ crossed classification を，残りの二つは同じ nested classification を指定する．抽象的には，これら四つは同じモデル空間を特定する.

`y ~ (A + B + C)^2` `y ~ A*B*C - A:B:C`	主効果と二次交互作用のみを持つ三因子実験．両者は同じモデルを指定する.

`y ~ A * x`
`y ~ A/x`
`y ~ A/(1 + x) - 1`
	A の水準内で y を x に回帰する様々な単純線形回帰モデルで，異なったコーディングを持つ．最後の形は A に含まれる水準と同じ数の，異なった切片と傾きの推定値を陽に与える.

`y ~ A*B + Error(C)`	二つの処理因子 A と B と，因子 C が定める誤差層 (error strata) を持つ実験．例えば，因子 C から定まる全区画（従って同じく副区画）を持つ分割区画実験.

演算子 ~ は R の「モデル式」を定義する．通常の線形モデルの形式は

$$response \sim [\pm]\ term_1 \pm term_2 \pm term_3 \pm \cdots$$

となる.

response はベクトルか行列（もしくは結果がベクトルか行列になる評価）で反応（目的）変数を定義する．
± は演算子で + か - である．ある項をモデルに含むか除外するかを表す.（前者はオプションである.）
term は

- ベクトルか行列表現，もしくは `1`,
- 因子，もしくは
- 「式演算子」で結ばれた因子，ベクトル，または行列からなる「式表現」.

全ての場合で，各項はモデル行列に加えられるか，除外される列ベクトルの集まりを定義する．`1` は切片列ベクトルで，既定では明示的に除外されない限りモデル行列に含められる.

「式演算子」は Glim や `Gnestat` のようなプログラムで用いられる Wilkinson と Rogers の表記法と効果において似通っている．一つの避けられない変更は R ではピリオッドが意味のある名前用の文字であるために，演算子 “.” が “:” になることである．表 1 に演算子の表記が要約されている（Chambers & Hastie, p. 29）に基づく.

関数の引数を普通囲む括弧の内部では，全ての演算子は通常の数値計算における意味を持つことを注意しよう．関数 `I()` は恒等関数で，モデル式中の項が数値演算子を用いて定義されることを許すためだけに使われる.

8.2 回帰モデル：当てはめられたモデルオブジェクト

通常の多重モデルを当てはめるための基本的関数は `lm()` であり，その呼び出しの streamlined 版は

`> `*fitted.model* `<- lm(`*formula*`, data=`*data.frame*`)`

となる．例えば

```
> fm2 <- lm(y ~ x1 + x2, data=production)
```

は y の x_1 と x_2 による（切片項を暗黙のうちに含む）重回帰モデルを当てはめる．重要だがテクニカルなオプションであるパラメータ `data=production` は，モデルを構成するのに必要な全ての変数は，先ず「`production` データフレーム」から取られることを指示する.「これはデータフレーム `production` が既に検索リストに付け加えられているか否かによらない.」

形式	意味
Y ~ M	Y は M と同様のモデル
M_1 + M_2	M_1 と M_2 を含め
M_1 - M_2	M_1 を含め，M_2 の項は除外せよ
M_1:M_2	M_1 と M_2 のテンソル積．もし両者が因子なら ”サブクラス因子”
M_1 %in% M_2	M_1:M_2 に似るがコーディングが異なる
M_1*M_2	M_1 + M_2 + M_1:M_2
M_1/M_2	M_1 + M_2 %in% M_1
M^n	M の全ての項と，n 次までの交互作用
I(M)	M を孤立させる．M の中では全ての演算子はその通常の数値演算での意味を持ち，その項はモデル行列に登場する

表 1: モデル演算子の構文の要約

8.3 情報抽出のための総称的な関数

lm() の返り値は当てはめられたモデルのオブジェクトである．技術的にはクラス lm の返り値であるリストである．当てはめられたモデルに関する情報は，その後でクラス lm のオブジェクトに方向づけられた総称的 (generic) な関数により，表示，抽出，描画等を行なうことが出来る．こうした関数の現在の完全なリストは

add1	coef	effects	kappa	predict	residuals
alias	deviance	family	labels	print	summary
anova	drop1	formula	plot	proj	

である．これらの内次のものは今の所組み込まれていないが，近い将来加えられる予定である．

add1	kappa	alias	labels	drop1	proj

表 2 に最も普通に使われるものの簡単な説明をあげる．

8.4 分散分析；モデルの比較

aov() はこれまでの所実装されて「いない」ことを注意しよう．

モデル当てはめ関数 aov(formula, data=*data.frame*) は，最も単純なレベルでは lm() と極めて似通った操作を行ない，表 2 にあげたほとんどの総称的な関数が使える．

aov() は更に分割区画実験や balanced incomplete block designs with recovery of inter-block information といった多重の誤差層を持つモデルの解析も行なえることを注意しよう．モデル式

response ~ *mean.formula* + Error(*strata.formula*)

は *strata.formula* で定義される誤差層を持つ多重誤差層実験を指定する．因子の水準の級内，級間という二層実験を定義するといういい最も簡単な場合，*strata.formula* は単に因子である．

たとえば，次の例のようなモデル式

```
> fm <- aov(yield ~ v + n*p*k + Error(farms/blocks), data=farm.data)
```

関数	返り値もしくは効果
anova(*object*$_1$, *object*$_2$)	サブモデルと外部モデルを比較し，分散分析表を生成.
coefficients(*object*)	回帰係数（行列）を抽出. 省略形：coef(*object*).
deviance(*object*)	（必要なら重み付けられた）残差平方和.
formula(*object*)	モデル式を抽出.
plot(*object*)	二種類のプロットを生成．一つは観測値と当てはめ値，もう一つは当てはめ値に対する残差の絶対値.
predict(*object*, newdata=*data.frame*) predict.gam(*object*, newdata=*data.frame*)	提供されるデータフレームが元のものと同じラベルを持つことを強制．値は *data.frame* 中の非ランダムな変量に対応する予測値のベクトルもしくは行列. predict.gam() は predict() に対する安全な代替物で，lm，glm そして gam による当てはめオブジェクトに対して使うことが出来る．例えば，それは直交多項式がオリジナルな基底関数として使われ，新しいデータの追加が元のものと異なる基底関数を意味する場合に使われるべきである.
print(*object*)	オブジェクトの簡略版を表示. 最もしばしば暗黙の内に使われる.
residuals(*object*)	（必要なら重み付けられた）残差（の行列）．省略形：resid(*object*).
summary(*object*)	回帰分析の結果の完全な要約を表示.

表 2: クラス lm のオブジェクトに対して通常使われる総称的な関数

は典型的に平均モデル v +n*p*k と三つの誤差層，つまり「圃場間誤差」，「圃場内誤差」，そして「ブロック内誤差」を持つ実験を記述するのに用いられる.

8.4.1 ANOVA（分散分析）表

分散分析表は当てはめられたモデルの系列に対するものであることを，同様に注意しよう．表示された自乗和は，モデル中の「その項」を系列中の「その場所」に含めたことに由来する残差平方和の減少分である．したがって，直交実験においてのみ，項を含める順序が非本質的になる.

多層実験に対する手順は，先ず応答を誤差層の上に射影（再び順番に）し，それから平均モデルを各射影に当てはめる．これ以上の詳細については Chambers and Hastie の第 5 章を見よ.

既定値である完全な分散分析表に対するより柔軟な代替物は，二つもしくはそれ以上のモデルを anova() 関数を使って比較することである.

```
> anova(fitted.model.1, fitted.model.2, ...)
```

結果は，すると順番に当てはめられた当てはめモデル間の差を示す分散分析表になる．比較されているモデルは，勿論普通階層的な系列であろう．これは既定値と異なる情報を与えるわけではなく，むしろ把握と制御を

より簡単にするのである．

8.5 当てはめモデルの更新．同前名“.”

update() 関数は多く，既に当てはめられたモデルに，普通ごくわずかの項を加えたり除いたりして得られるモデルの当てはめを容易に行なうために使われる．その形式は

> *new.model* <- update(*old.model*, *new.formula*)

である．*new.formula* 中ではピリオッド「.」だけからなる特別な名前を「元のモデル式中の対応する部分」を表すのに使うことが出来る．例えば

```
> fm05 <- lm(y ~ x1 + x2 + x3 + x4 + x5, data=production)
> fm6  <- update(fm05, . ~ . + x6)
> smf6 <- update(fm6, sqrt(.) ~ .)
```

は，データフレーム（仮定では）production からのデータを用いた 5 変量の重回帰の当てはめを行ない，次に 6 番目の説明変数を取り込んで当てはめ，最後に目的変数に平方根変換を行なった変形モデルを当てはめている．

もし data= 引数が，最初のモデル当てはめ関数に指定されると，この情報は当てはめられたモデルオブジェクトを通じて update() やその仲間に伝えられることを，特に注意しよう．

名前“.”はまた別の文脈でも使うことが出来るが，少し異なった意味を持つ．例えば

```
> fmfull <- lm(y ~ .  , data=production)
```

は目的変数 y と「データフレーム production 中の他の全ての説明変数」を用いたモデルの当てはめを行なうであろう．

モデルの拡大系列を探求するための他の関数には add1(), drop1(), step() そして stepwise() がある．これらの関数の名前はその意味への良いヒントとなるが，完全な詳細についてはヘルプ文章を見よ．

8.6 一般化線形モデル；そのファミリー

一般化線形モデルは，線形モデルを，非正規分布を持つ応答変数（目的変数）とその線形化変換の双方を，明解で直接的な方法で取り扱得るように拡張したものである．一つの一般化線形モデルは，次の一連の仮定を用いて記述できる：

- 関心のある応答変数 y と，刺激変数（説明変数） $x_1, x_2, \ldots$ があり，説明変数の値は目的変数の分布に影響を持つ．
- 刺激変数は y の分布に「ただ一つの線形関数を通じてのみ」影響を及ぼす．この線形関数は「線形予測量」と呼ばれ，通常
  $$\eta = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$
  と書かれる．従って，x_i は $\beta_i = 0$ の時，そしてその時のみ，y の分布に影響を及ぼさない．
- y の分布は次の形
  $$f_Y(y; \mu, \varphi) = \exp\left[\frac{A}{\varphi}\left\{y\lambda(\mu) - \gamma\left(\lambda(\mu)\right)\right\} + \tau(y, \varphi)\right]$$
  を持つ．ここで φ は「スケールパラメータ」（既知の場合もある）で，全ての観測値に対し定数である．A は予め与えられた重みで，既知と仮定されるが，観測値ごとに異なるかも知れない．μ は y の平均である．従って y の分布はその平均，そしておそらくスケールパラメータにも依存して決まる．
  平均 μ は線形予測量の可逆で滑らかな関数
  $$\mu = m(\eta), \qquad \eta = m^{-1}(\mu) = \ell(\mu)$$
  であり，この可逆関数 $\ell(.)$ は「リンク関数」と呼ばれる．

これらの仮定は，実際の統計解析で有用な幅広いクラスのモデルを包含するのに十分なくらい緩いが，推定と推測の統一した理論を展開するのに（少なくとも近似的には）十分なほど狭い．詳細については，この話題を扱った次のような参考文献を見ることをすすめたい．

> *Generalized linear models* by Peter McCullagh and John A Nelder,
> 2nd edition, Chapman and Hall, 1989, もしくは
>
> *An introduction to generalized linear models* by Annette J Dobson, Chapman and Hall, 1990.

8.6.1 ファミリー

R が提供する機能で扱える一般化線形モデルのクラスは応答分布（応答変数の分布）が正規分布，二項分布，ポアッソン分布，逆正規分布，そしてガンマ分布の場合を含む．また応答分布がはっきりと特定されていない場合に対する疑似尤度モデルを含む．最後の場合には「分散関数」が平均の関数として特定されている必要があり，それ以外ではこの関数は応答分布から決まる．

各応答分布には，平均と線形予測量を結ぶ様々なリンク関数を対応させることが出来る．自動的に使用可能なものを表 3 にあげた．

	ファミリー名					
リンク関数	binomial	gaussian	Gamma	inverse.gaussian	poisson	quasi
logit	⋆					⋆
probit	⋆					⋆
cloglog	⋆					⋆
identity		⋆	⋆		⋆	⋆
inverse			⋆			⋆
log			⋆		⋆	⋆
1/mu^2				⋆		⋆
sqrt					⋆	⋆

表 3: ファミリーとそれらが使用できるリンク関数

モデル作成に必要な，応答分布，リンク関数，そして，その他の様々な情報の集まりは，一般化線形モデルの「ファミリー (family)」と呼ばれる．

8.6.2 `glm()` 関数

応答分布は「ただ一つ」の線形変換を通じてのみ刺激変数に依存しているので，一般化モデルの線形部分を特定するのには，線形モデルに対して使われたのと同じメカニズムを使うことができる．ファミリーは別の方法で特定されねばならない．

一般化線形モデルを当てはめる R の関数は `glm()` で

> *fitted.model* `<-` `glm(`*formula*`, family =`*family.generator*`, data=`*data.frame*`)`

の形を持つ．

唯一の新しい特徴はファミリーを記述する道具である「ファミリー生成子 (*family.generator*)」である．これは，一緒になってモデルと推測の過程を定義し制御する関数と表現のリストを生成する関数の名前である．これは，最初一見すると，少々複雑に見えるかも知れないが，その使用法は極めて簡単である．

標準的に提供されているファミリー生成子の名前は表 3 の「ファミリー名」の欄に与えられている．もしリンク関数が選択できるなら，リンクの名前を括弧にいれて，パラメーターとしてファミリー名に加えることができる.「疑似 (quasi)」ファミリーの場合には，分散関数を同様に特定することができる．

いくつかの例をあげると仕組みが分かりやすいであろう．

正規分布族

次のような呼出し

```
> fm <- glm(y ~ x1+x2, family=gaussian, data=sales)
```

は

```
> fm <- lm(y ~ x1+x2, data=sales)
```

と同じ結果を与えるが，ひどく効率が悪くなる．リンク関数の選択が自動的に正規分布族を選ぶのでは無いこと，したがっていかなるパラメータも許されないことを注意しよう．もし問題が非標準的なリンクを持つ正規分布族を必要とするならば，後で分かるように，それは通常 quasi ファミリーによって達成される．

二項分布族

簡単な人工的な例を考えよう．

ギリシャのカリトス島の男性住民は，年齢とともに進行する，遺伝的な目の病気にかかる．さまざまな年齢の島の男性住民の盲目度が調査され，記録された．データは表 4 に示してある．

年齢:	20	35	45	55	70
No. 被検者数:	50	50	50	50	50
No. 盲目者数:	6	17	26	37	44

表 4: カリトス島の盲目データ

我々の考える問題は，このデータにロジスチックモデルとプロビットモデルを当てはめ，各モデルに対する LD50 値，つまり男性住民が盲目になる可能性が 50%を越える年齢を推定することである．

y を年齢 x での盲目者数，n を被検者数とすると，両方のモデルは

$$y \sim \mathrm{B}(n, F(\beta_0 + \beta_1 x))$$

の形になる．ここで，$F(z) = \Phi(z)$ はロジットモデル（既定）では標準正規分布，プロビットモデルでは $F(z) = e^z/(1 + e^z)$ である．双方のモデルで LD50 値は

$$\mathrm{LD50} = -\beta_0/\beta_1$$

となる．つまり分布関数の変数が零となる点である．

最初のステップはデータをデータフレームとしてセットすることである

```
> kalythos <- data.frame(x=c(20,35,45,55,70), n=rep(50,5),
                         y=c(6,17,26,37,44))
```

glm() を用いて二項モデルを当てはめるには応答変数として二つの可能性がある：

- もし応答が「ベクトル」ならば，それは「バイナリー（二値）」データを含んでいると仮定され，結局 0, 1 ベクトルでなければならない．

- もし応答が「二列の行列」ならば，最初の列は試行における成功の数，第二列は失敗の数を含んでいると仮定される．

ここでは，後者の場合を考えるので，データフレームに行列を加える：

```
> kalythos$Ymat <- cbind(kalythos$y, kalythos$n - kalythos$y)
```

モデルを当てはめるには

```
> fmp <- glm( Ymat~x, family=binomial(link=probit), data=kalythos)
```

```
> fml <- glm( Ymat~x, family=binomial, data=kalythos)
```

とする．ロジットリンクが既定値であるので，第二の呼出しではパラメータを省略しても良い．各当てはめの結果を見るには

```
> summary(fmp)
```

```
> summary(fml)
```

とする．ともに（うますぎるほど）良い当てはめを示す．LD50 値の推定値を見付けるには簡単な関数

```
> ld50 <- function(b) -b[1]/b[2]
```

```
> ldp <- ld50(coef(fmp)); ldl <- ld50(coef(fmp)); c(ldp, ldl)
```

を用いることができる．このデータから得られる実際の推定値は，それぞれ 43.663 歳と 43.601 歳である．

ポアッソンモデル

ポアッソンモデルに対する規定のリンクは `log` であり，実際ではこのファミリーの主な利用は頻度データに代理 (surrogate) ポアッソン対数線形モデルを当てはめることにあり，その実際の分布はしばしば多項分布になる．これは重要な大きな主題であり，ここではこれ以上議論することは無い．それは全般的な非正規一般化モデルの利用の主要な部分を占めさえする．

しばしば真にポアッソン分布に従うデータが現実に登場し，過去にはそれは対数か平方根変換した後で正規データとして解析されることがしばしばあった．後者に対する上品な代用品として，次の例のように，ポアッソン一般化線形モデルを当てはめることができる：

```
> fmod <- glm(y ~ A+B + x, family=poisson(link=sqrt), data=worm.counts)
```

疑似尤度モデル

すべてのファミリーで，応答の分散は平均に依存し，スケールパラメーター（尺度母数）が乗算因子として掛かって来る．平均への依存の仕方は応答変数の分布の特性である；たとえばポアッソン分布では $\mathrm{Var}[y] = \mu$ となる．

疑似尤度モデルによる推定と推測では，応答分布の正確な形は特定されず，リンク関数と分散関数の平均への依存の形式だけが指定される．疑似尤度推定は，形式的には正規分布に対するのと同一のテクニックを使うので，このファミリーは非標準的なリンク関数や分散関数を持つ正規モデルを当てはめる手法を，たまたま提供する．

たとえば，次の非線形回帰

$$y = \frac{\theta_1 z_1}{z_2 - \theta_2} + e \tag{1}$$

を考える．これは次のようにも表せる

$$y = \frac{1}{\beta_1 x_1 + \beta_2 x_2} + e$$

ここで $x_1 = z_2/z_1$, $x_2 = -1/x_1$, $\beta_1 = 1/\theta_1$ そして $\beta_2 = \theta_2/\theta_1$ である．適当なデータフレームがセットされていると仮定すると，この非線形回帰は次のように当てはめることができる

```
> nlfit <- glm(y~x1+x2-1,family=
      quasi(link=inverse,variance=constant), data=biochem)
```

これ以上の情報に付いては，必要に応じてマニュアルとヘルプ文章を参照してほしい．

8.7 非線形回帰モデル；パラメトリックなデータフレーム

これらの機能はまだ実装されて「いない」ことを注意してほしい．

R は一般化線形モデルの部分的線形 (partially linear) パラダイムにすら適合しない非線形モデルを当てはめる二つの関数を提供する．`ms()` は，目的関数が類似した項の和であるような任意の最小化問題用である．`nls()` は正規非線形回帰モデルに対する定型的な非線形最小自乗法用である．

この短い紹介では，非線形回帰関数 `nls()` のみを考える．`ms()` に付いては，読者が必要に応じて調べてほしい．

8.7.1 モデル式の形への変更

線形，もしくは一般化線形モデルを特定する際には，我々は回帰パラメータが暗黙のうちに定義されており，それらが乗算されるべきモデル行列の列から譲り受けた名前を持つと仮定しても良かった．

非線形モデルではそのような単純化はできず，モデルを定義変数 (determining variable) とパラメータをともに含む通常の表現式として特定しなければならない．たとえば先に述べた `storm` のような非線形回帰に対する一つのモデルを特定するには，

```
y ~ t1*x1/(x2 - t2)
```

とすることができよう．ここで，`y` は応答変数, `x1` と `x2` は尺度パラメータである．

このようなモデル式では，すべての演算子はそれらの数値式としての通常の意味を持っており，因子や，交差そして入れ子の構造を展開する有用な機能は，もはや使うことができない．すべてのパラメータは，たとえそれらがモデルの線形な部分に由来したとしても，式の中で正確に定義されねばならない．

8.7.2 パラメータの特定

モデル式が，今や定義変数とパラメータの双方を含んでいるので，どっちががどっちであるのか特定する何らかのメカニズムがなければならない．しかしながら，パラメータが特定されれば，当然モデル式の中の残りの変数は定義変数でなければならない．

どれがパラメータかを特定するのと同時に，この場合には，繰り返しによる推定の手順を始める際の初期値を特定する必要がある．

この情報を特定する二つの方法がある：

- もし `nls()` の呼出しが `start=` によるパラメータの指定を持てば，その値は名札付き成分のリストでなければならない．リストの名札は初期値を与えるパラメータの名札と値を指定する．
- もしデータがデータフレームに保持されているならば，パラメータは同様にデータフレームの「パラメータ属性 (*parameters attribute*)」としても定義できるであろう．

我々の原則はできるだけデータフレームを使って作業することにあるので，次の例では第二の可能性を示めそう．

例

再び式 1 の非線形回帰を考える．パラメータの初期値を見付ける簡単な方法は x_2y を x_1 と x_2 に回帰することである：

```
> fm0 <- lm(x2*y ~ x1 + x2 - 1, data=biochem)

> th <- coef(fm0)
```

パラメータに名札を付け，それらを biochem に関連づけるには，次のようにする：

```
> parameters(biochem) <- list(t1=th[1], t2=th[2])
```

ここで非線形回帰モデル

```
> fm <- nls(y ~ t1*x1/(x2 - t2), data=biochem)
```

を当てはめる．ここでモデルを調査し情報を表示するために summary() 関数や，他の多くの総称的な関数をを使うこともできるであろう．係数を抽出するためにたとえば

```
> th <- coef(fm)
```

を使うことができるし，これらの最小自乗推定値を biochem に関連するパラメータの新しい値にするためには，単にステップ

```
> parameters(biochem) <- list(t1=th[1], t2=th[2])
```

を繰り返せば良い．

関数 parameters() は，データフレームからパラメータのリストを抜き出すための表現式として使うこともできるし，指定されたデータフレームにパラメータリストを付け加えるための，付値の目標としても使うことができる．この点で，これは attributes() 関数に非常に良く似ている．attr() に類似した 関数 param() もあり，これは名札文字列を用いて，一時に一つのパラメータを処理する．

8.8 幾つかの非標準モデル

これらの機能はほんの「わずか」だけインプリメントされているに過ぎないことを注意しよう．

R で使える特別な回帰とデータ解析問題に対する他の機能をほんの簡単に述べて，この章を終えよう．

局所近似回帰 関数 loess() は，局所的な重みを用いた回帰を用い，ノンパラメトリックな回帰の当てはめを行う．こうした回帰はややこしいデータの傾向を際だたせたり，大規模データに対する何らかの知見を得る目的でデータの縮約を行うために有効である．

ロバストな回帰 データ中の極端な外れ値の影響に対し抵抗力を持たせるように回帰モデルを当てはめるための関数がいくつかある．これらのうち，最も洗練された物は rreg() であるが，他にも最小自乗中央値回帰のための lmsfit() 関数や，L_1 ノルムを用いた回帰のための l1fit() 関数がある．しかしながら，これらはまだ公式を用いて，たとえばモデル関数を特定したり古い手順にあわせたりする機能を持たないため，しばしばかなり使うのが面倒である．さらに glm ファミリーのオブジェクトを glm() モデル当てはめ関数で使用するためにロバスト化するための robust() 関数もある．

一般化加法モデル この手法は，決定変数の滑らかな加法的関数（普通，各決定変数にたいし一つずつ）から回帰関数をつくり出すことを目指している．関数 gram() は多くの点で，先に概要を述べた他のモデル当てはめ関数に似ている．さらに同じ仕事をする他のモデル当てはめ関数もある．関数 avas() や ace() がそれである．一方で射影追跡 (projection persuit) 回帰のために ppreg() が使えるが，この技術は依然として多くの完全な理論的解析と一層の実際の経験を必要としている．これらの関数は，やはりモデル当てはめ関数の旧式の手順にしたがっており，より新しい関数の持つ便宜さを欠いている．

木に基づくモデル 木（ツリー）に基づくモデルは，予測や解釈のために正確な大局的線形モデルを求めるというよりは，最終的にデータを，グループ内ではできる限り等質的，グループ間ではできる限り異質的になるように，決定変数のクリティカルな値で再帰的にデータを分岐させることを目指している．その結果は，しばしば，他のデータ解析の手法では得にくいような洞察を与える．

モデルは再び通常の線形モデルの形式で指定される．モデル当てはめ関数は `tree()` であるが，`plot()` や `text()` といった多くの他の関数は，木に基づくモデルの当てはめの結果をグラフィカルに表示するのにうまく適合している．

9 作図手続き

作図機能は重要なで極めて多岐にわたる R 環境の要素である．この機能を用いて，多彩な統計グラフを表示したり，また全く新しいタイプのグラフを構成することが出来る．

作図機能は対話的・非対話的なモードの双方で利用できるが，多くの場合対話的な利用の方がより生産的である．対話的な利用は，起動時に R が対話的なグラフ表示のための特殊な「グラフィックスウィンドウ」を開く作図「デバイスドライバー (*device driver*)」を初期化するので，また簡単でもある．これは自動的になされるが，使用されている命令が Unix 上では `X11()`，Windows 95 や Windows NT では `Windows()`，Macintosh では `Macintosh()` であることを知っておくと便利である．

一度デバイスドライバーが稼働し始めると R の作図命令は，様々なグラフ表示を生成したり，全く新しい種類の表示を作るために利用できる．

作図命令は 3 つの基本的なグループに分けられる．

高水準 作図関数は，作図デバイスに新しい図形を表示し，これは普通軸・ラベル・表題などを伴う．

低水準 作図関数は，既存の図に余分の点・直線・ラベルなどの付加情報を追加する．

対話的 作図関数は，マウスなどの位置指示デバイスを使って，対話的に既存の図に情報を追加したり，抜きだしたりすることを可能にする．

更に R は使用者が図のカスタマイズを処理するために使うことができる「作図パラメータ」のリストを保持している．

9.1 高水準作図命令

高水準作図関数は，関数に引数として渡されたデータの，完全なプロットを生成するように設計されている．もしそれが適当なら，軸・ラベルそして表題が自動的（別の事を指示しない限り）に生成される．高水準作図命令は常に新しい図を開始し，必要ならば現在の図を消しさる．

9.1.1 `plot()` 関数

R において最も頻繁に利用される作図関数の一つが `plot()` 関数である．これは「総称的な」関数である．生成されるプロットのタイプは第一引数のタイプや「クラス」に依存する．

`plot(x,y)` `plot(xy)`	`x` と `y` がベクトルなら，`plot(x,y)` は `y` に対する `x` の散布図を作成する．同じ効果は，2 つの成分 `x` と `y` を含むリストや，2 列の行列のいずれかを，単独の引数として与える（第二の形式）ことによっても得られる．
`plot(x)`	もし `x` が時系列なら，時系列プロットを生成し，`x` が数値ベクトルなら，ベクトル中の数値を，そのベクトルの添字に対してプロットし，またもし `x` が複素ベクトルなら，ベクトル成分の実部に対する虚部のプロットを生成する．
`plot(f)` `plot(f,y)`	`f` は因子オブジェクト，`y` は数値ベクトルである．最初の形式は `y` の棒グラフを，第二の形式は `f` の各水準に対する `y` の箱型図 (box plot) を生成する．

`plot(df)`
`plot(~ expr)`
`plot(y ~ expr)`
	`df` はデータフレーム，`y` は任意のオブジェクト，`expr` は ‘`+`’ で仕切られたオブジェクト名のリスト（例えば `a + b + c`）である．最初の二つの形式は，データフレーム中の変量（第一形式），または名前が与えられたオブジェクト（第二形式），の分布関数プロット (distributional plot) を生成する．第三の形式は `expr` に名前が与えられた全てのオブジェクトに対して `y` をプロットする．

9.1.2 多変量データの表示

R は多変量データを表現する二つの大変便利な関数を提供する．X が数値行列またはデータフレームならば，命令

```
> pairs(X)
```

は X の列で定義される変量の対毎の散布図の行列（pairwise scatterplot matrix）を生成する，つまり X の全ての列が他の全ての列に対してプロットされ，得られた $n(n-1)$ 個のプロットは行列型に配置され，その作図スケールは行列の行と列に対して共通となる．

三つもしくは四つの変量が含まれるときは「共変量プロット (*coplot*)」がより分かりやすいであろう．もし a と b が数値ベクトルで，c が（全てが同じ長さの）数値ベクトル，もしくは因子オブジェクトなら，命令

```
> coplot(a ~ b | c)
```

は c の与えられた値における b に対する a の散布図を複数生成する．もし c が因子なら，これは単に c の全ての水準毎に，a が b に対してプロットされることを意味する．c が数値の場合，それはいくつかの「条件を与える区間」に分割され，それぞれの区間毎に，その区間に含まれる c の値に対して a が b に対してプロットされる．区間の数と位置は coplot() に対する given.values= 引数により制御することができる — 関数 co.intervals() は区間の選択に便利である．また二つの「与えられた」変量を次のような命令

```
> coplot(a ~ b | c + d)
```

で使うこともでき，これは c と d の条件付け区間をあわせたものにおける a の b に対する散布図を生成する．

coplot() 関数と pairs() 関数はともに， panel= 引数をもち，それぞれのパネルに現れるプロットのタイプをカスタマイズすることが出来る．既定では散布図を生成する points() であるが，panel= の値として x と y の 2 つのベクトルの他の低水準作図関数を与えることにより，希望するいかなるタイプのプロットも生成することができる．共変量プロットに対する便利なパネル関数の例は panel.smooth() である．

9.1.3 グラフィックスの表示

他の高水準作図関数は異なったタイプのプロットを生成する．いくつかの例は：

`tsplot(x1,x2,...)`	同じ尺度で任意個数の時系列をプロットする．この自動的な同時尺度化は $\mathtt{x}_i$ が通常の数値ベクトルのときにも便利であり，その場合それらは数値 $1, 2, 3, \ldots$ に対してプロットされる．

`qqnorm(x)`
`qqline(x)`
`qqplot(x,y)`
	分布を比較するプロット．最初の形式は x に対する期待正規ランクスコアをプロットし（正規スコアプロット），第二の形式はそうしたプロットに，データの上四分位数と下四分位数を結ぶ直線を描くことにより，そのプロットに直線を追加する．第三の形式は，それぞれの分布を比較するために x の確率点に対する y の確率点をプロットする．

`hist(x)`
`hist(x,nclass=`n`)`
`hist(x,`
`  breaks=...)`
	数値ベクトル x のヒストグラムを生成する．通常は適切な階級数が選ばれるが nclass= 引数で望みの階級数を指定することが出来る．もしくは breaks= 引数により，区切り点 (breakpoints) を正確に指定することが出来る．probability=T 引数が与えられたとき，柱は度数ではなく相対頻度を表す．

`dotchart(x,...)`	x におけるデータの点グラフ (dotchart) を構成する．点グラフにおいては $y-$ 軸は x におけるデータのラベル付けを与え，$x-$ 軸はその値を与える．例えばこのグラフは，特定の範囲にある全てのデータ項目を容易に視覚的に選択することを可能にする．

9.1.4 高水準作図関数の引数

高水準作図関数に渡すことができる以下のようないくつかの引数がある：

`add=T`	その関数が低水準作図関数として動作するよう強制し，現在のプロットにプロットを上描きする（使える関数は限られる）.
`axes=F`	軸の生成を抑制する — `axis()` 関数を用いて自分自身の設定した軸を追加するのに有用である．既定である `axes=T` は軸を含むことを意味する.

`log="x"`
`log="y"`
`log="xy"`
	x，y もしくは双方の軸を対数軸とする．これは多くの作図の場合可能であるが，いつでも可能であるわけではない.

`type=`	`type=` 引数は生成されるプロットのタイプを以下のように制御する：
`type="p"`	個々の点を描く（既定）
`type="l"`	線を描く
`type="b"`	線で連結された点を描く (*both*)
`type="o"`	点を描き，線で上描き
`type="h"`	点からゼロ軸（x 軸）まで鉛直線を描く (*high-density*)
`type="s"` `type="S"`	階段関数プロット．最初の形式は鉛直方向の頂点が，第二形式では基部がその点を定義する.
`type="n"`	全くなにも描かない．しかし（既定では）軸はやはり描かれ，座標軸がデータに基づいて設定される．引き続いて低水準作図関数を用いた作図をするのに最適.
`xlab="string"` `ylab="string"`	x 軸と y 軸の軸ラベル．既定のラベル（高水準作図関数の呼び出し時には一般にオブジェクト名）をかえるためにこれらの引数を使え.
`main="string"`	図表の上部に置かれる，大きなフォントで描かれる図の表題.
`sub="string"`	x-軸の直下に置かれる，より小さなフォントで描かれる副題.

9.2 低水準作図命令

高水準作図関数は場合によって，希望する種類の図表を正確には生成しない．その場合低水準作図命令を用いて現在の図表に対して余分の情報（点や線やテキストなど）を追加するために利用することができる.

いくつかのより便利な低水準作図関数は：

`points(x,y)` `lines(x,y)`	現在の図表に点もしくは連結線を描く．`plot()` の `type=` 引数もまた，これらの関数に渡すことができる（既定では `points()` に対して `"p"` を，`lines()` に対して `"l"` を与える）.

`text(x, y, labels, ...)`	x, y で与えられる点にテキストを置く．通常 labels は整数もしくは文字ベクトルであり，そのとき labels[i] は点 (x[i], y[i]) に描かれる．既定では 1:length(x)． 注意: この関数は通常 `quad > plot(x, y, type="n"); text(x, y, names)` と引き続いて使われる．作図パラメータ type="n" はその点の表示を抑制するが，軸を構成する．そして，text() 関数は，その点に対して文字ベクトル names により特定される，特別な文字を提供する．

`abline(a, b)`
`abline(h=y)`
`abline(v=x)`
`abline(`*lm.obj*`)`
	傾きが b 切片が a の直線を現在の図表に追加する．h=y は図表を横切る水平線の高さの y 座標を特定するのに，同様に v=x は鉛直線の x-座標を指定するのに使うことができる．同様に *lm.obj* は（モデル当てはめ関数の結果等の）長さ 2 の $coefficients 成分を持つリストであり，それらは順に引かれるべき直線の切片と傾きになる．

`polygon(x, y, ...)`	(x,y) を順序の付いた頂点とする多角形を描き，(オプションで）陰影線により影をつけたり，作図デバイスが塗りつぶしを許す場合には塗りつぶしを行う．
`legend(x,y, legend,...)`	現在の図の指定された位置に凡例 (legend) を付加する．作図記号・線種・色などは，ベクトル legend で与えられた文字ベクトルのラベルにより識別される．少なくとも一つの他の，作図単位で表された値を持つ，引数 y （legend と同じ長さのベクトル）が以下のように与えられねばならない．
`legend( ,angle=v)`	影の角度
`legend( , density=v)`	影の密度
`legend( ,fill=v)`	箱を塗りつぶす色
`legend( ,col=v)`	点や線の色
`legend( ,lty=v)`	線分の種類
`legend( ,pch=v)`	作図文字（文字ベクトル）
`legend( ,marks=v)`	pch= に数値引数が与えられたとき得られる作図記号 (numeric vector)．
`title(main,sub)`	表題 main を現在の図表の上部に大きな文字で描く，(オプションとして）副題 sub をより小さな文字で基部に描く．
`axis(side,...)`	現在の図表に，第一引数により与えられた側（1 から 4，基部から時計回りで数える）に軸を追加する．他の引数は軸を図表の内側か外側に描くかどうか，目盛り位置，ラベルなどを制御する．axes=F 引数付きで plot() を呼んだ後で，好みの軸を加える場合に便利である．

低水準作図関数は通常新しい作図要素を置く位置を決めるために，位置情報を要求する（例えば x と y の座標）．座標は先立つ高水準作図関数により定義された「ユーザ定義座標」に対して与えられ，提供されたデータに基づいて選ばれる．

x と y 引数が要求されるところでは，x と y と名付けられた要素を持つリストである一つの引数を与えても同じく十分である．同様に二つの列からなる行列もまた正当な入力である．このような方法で locator()（後述）のような関数も図表における位置を対話的に指定するために使うことができる．

9.3 対話的な作図関数

R は使用者がマウスを使って，図表から情報を除いたり追加したりする事を可能にする関数も用意している．最も簡単なものは locator() 関数である：

`locator(n)`	ユーザが図表においてマウスの左ボタンで位置を選択するの待つ．これはユーザが `n`（既定値は 512）点選ぶまでか，中央ボタンを押すまでつづけられる．`locator()` は選択された位置を 2 つの要素 `x` と `y` をリストとして返す.

`locator()` は通常引数を伴わずに呼ばれる．これは凡例やラベルなどのグラフ要素の位置を，グラフのどの位置に置くかあらかじめ計算するのが困難なときに，対話的に選択するために特に便利である．例えば外れ値の近くにいくつかの有益なテキスト置く場合，次の命令

```
> text(locator(1), "Outlier", adj=0)
```

は便利だろう．`locator()` は現在のデバイスがマウスをサポートしていない場合にも動作する．この場合はユーザが x と y 座標を入力することを促すプロンプトが現れる．

`identify(x,y, labels)`	ユーザが（マウスの左ボタンを使って）定義した `x` と `y` の点を，その近くに関連する `labels` の要素（もしくは `labels` がなければ観測番号）を描くことにより，強調表示することを可能にする．中ボタンを押すと，選択した点の観測番号を返す.

場合によっては我々は作図上の特定の「点」の位置を知るよりも，それを識別したい．例えばいくつかの興味ある観測値を作図表示から選択し，それらを何らかの方法で操作したい．二つの数値ベクトル `x` と `y` のうちのいくつかの (x, y) 座標が与えられれば，我々は `identify()` 関数を以下のように利用することができるであろう．

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

```
> identify(x,y)
```

`identify()` 関数はそれ自身では作図は行わないが，ユーザがマウスポインターを動かし，ある点の近くでマウスの左ボタンをクリックすることを許す．マウスポインターに最も近い点が，観測番号値を近くに表示する（つまり，`x`/`y` ベクトルにおける順番）ことにより強調される．また使用者は `identify()` に対して `labels` 引数を利用することにより,（例えばケース名など）有益な文字列を強調表示として利用したり，`plot=F` 引数を与えて強調を全くしないようにすることもできる．中ボタンが押されたとき，`identify()` は選択された点の番号を返す．つまり使用者はこれらの添字をもともとの `x` と `y` のベクトルから選び出すために利用することができる．

9.4 作図パラメータの利用

グラフを作るとき，特に発表や出版の目的の場合，R は必ずしも要求された通りのものを生成しない．しかしながら「作図パラメータ」を使えば，表示のほとんど全ての見掛けをカスタマイズできる．R は中でも線分の形式・色・図表の配置，及びテキスト位置の整形等の様々なものを制御する数多くの作図パラメータのリストを維持する．全ての作図パラメータは（色を制御する ‘col’ などの）名前と値（例えば色番号）を持っている．

活動中の各デバイスのために，別々のグラフパラメータが保持されており，各デバイスは初期化の際既定値からなるパラメータのセットを持つ．二つの方法で作図パラメータを設定することができる：現在のデバイスにアクセスする全ての作図関数に影響する恒久的なものと，一つの作図関数だけに影響する一時的なもののどちらかである．

9.4.1 パラメータの変更: `par()` 関数

`par()` 関数は現在の作図デバイスに対する作図パラメータのリストにアクセスし，修正するために利用される．

`par()`	引数が無いと，現在のデバイスに対する全ての作図パラメータとそれらの値を返す.
`par(c("col", "lty"))`	文字ベクトル引数を伴うと，その名前の付いた作図パラメータのみを（再びリストとして）返す.
`par(col=4,lty=2)`	名前付きの引数（もしくは単一のリスト引数）により，その名前の作図パラメータの値を設定し，もとのパラメータの値をリストとして返す.

`par()` 関数で作図関数を指定することは，その後の全ての（現在のデバイスにおける）作図関数の呼び出しが新しい値の影響を受けると言う意味で「恒久的に」パラメータの値を変更する．このように作図パラメータを設定することは，パラメータの「既定値」を設定することと考えられ，別の値が与えられるまでこの値が全ての作図関数で使われるであろう.

`par()` の呼び出しは「常に」，たとえ `par()` がある関数において呼び出されたときでさえ，作図パラメータの大局的な値に影響を与えることを注意せよ．このことはしばしば望ましい振る舞いではない — 通常我々はいくつかの作図パラメータを設定し，作図を行い，そしてもとの値を回復し，ユーザの R セッションに影響を与えないようにしたい．変更をする際に `par()` の結果を保存し，作図が完了したときに初期値に戻すことによって，初期値を回復させることができる.

```
> oldpar <- par(col=4,lty=2)
```

. . . 作図命令 . . .

```
> par(oldpar)
```

9.4.2 一時的な変更: 作図関数に対する引数

作図パラメータは同様に，名前付きの引数として,（ほとんど）全ての作図関数に渡すことができる．これは，変更が関数呼び出し中だけであることを除けば，`par()` 関数に引数を渡すのと同様の効果がある．例えば

```
> plot(x,y,pch="+")
```

はプラス記号を作図文字として使う散布図を生成するが，将来の作図に対する既定作図文字を変更しない.

9.5 作図パラメータのリスト

以下の節では普通に使用される多くの作図パラメータを詳しく述べる．R の `par()` 関数のヘルプドキュメントにはもっと簡潔な要約が提供されているが，これは少しより詳しい代替物を与える.

作図パラメータは次の形式で提示されるだろう：

`name=value`	パラメータの効果の記述．`name` はパラメータ名，つまり `par()` や作図関数を呼ぶ際に使う引数の名前である．`value` はパラメータを設定する際に使用者が使うであろう典型的な値である.

9.5.1 グラフ要素

R の作図は，点・線・テキスト，そして多角形（塗りつぶされた領域）から構成されている．以下のような，どのように「グラフ要素」を描くかを制御する作図パラメータがある：

`pch="+"`	点を描くために用いられる文字．既定値は作図デバイスに応じて変わるが，通常は‘○’である．作図文字として `"."` を用いると中心に点を置かれる以外は，作図された点は適切な位置より若干上もしくは下に現れる傾向がある．
`pch=4`	`pch` が 0 から 18（18 を含む）の間に含まれる整数として与えられたとき，指定された作図記号が生成される．どんな記号かを知るためには次の命令を使う `> legend(locator(1),as.character(0:18),marks=0:18)`
`lty=2`	線分の種類．他の線種が全ての作図デバイスで使えるわけではない（また作図デバイスにより振る舞いも異なる）が，しかし線種 1 は常に実線であり，線種 2 とそれ以上の値は点線もしくは波線，もしくはその双方を組み合わせたものである．
`lwd=2`	線分の幅．希望する線の幅，標準の線幅の倍数となる．`lines()` などを伴って描かれる線と同様に軸線にも影響する．
`col=2`	点・線・テキスト・塗りつぶし領域，及び画像に使われる色．これらのグラフ要素は指定可能な色のリストを持ち，このパラメータの値はそのリストへの添字である．当然ながらこのパラメータは限られた範囲のデバイスのみに適用できる．
`font=2`	テキストに対して使われるフォントを指定する．可能ならばデバイスドライバーが，`1` には通常のテキスト，`2` には太字 (**bold face**)，`3` にはイタリック体 (*italic*) が，そして `4` には太字イタリック体 (***bold italic***) が対応するように調整する．
`font.axis`, `font.lab`, `font.main`, `font.sub`	それぞれ，軸の注釈，x, y-軸のラベル，表題・副題に使われるフォント．
`adj=-0.1`	作図位置に関するテキストの位置調整．作図位置に対して，0 は左詰め，1 は右詰め，0.5 は水平中央．実際の値は作図位置の左に現れるテキストに比例し，そのため値-0.1 はテキストと作図位置の間に，テキスト幅の 10% の隙間を作る．
`cex=1.5`	文字の拡大率．値は既定の文字の大きさに対する，希望するテキスト文字（作図文字を含む）のサイズ．

9.5.2 軸と刻み

多くの R の高水準作図は軸を持っており，低水準 `axis()` 作図関数で好みの軸を構成することができる．軸は 3 つの主要な要素を持つ：「軸線」（線種は `lty` 作図パラメータで制御される），「刻みマーク」（軸線を単位長さ毎に刻むマーク），そして「目盛り」（長さの単位のラベル）である．それらの要素は以下の作図パラメータで調整する事が出来る．

`lab=c(5,7,12)`	最初の二つの数字は，それぞれ x 軸と y 軸の希望する目盛り間隔．第三の値は希望する軸ラベルの文字数単位での長さ（小数点を含む）．このパラメータに小さすぎる値を選ぶと，全ての軸目盛りが丸められて同じ数になる！
`las=1`	軸ラベルの向き．0 は常に軸に対して平行であること，1 は常に水平であること，そして 2 は常に軸に対して垂直であることを意味する．
`mgp=c(3,1,0)`	軸要素の位置．第一要素はテキスト行数を単位とした，軸ラベルから軸位置までの距離．第二要素は刻みラベルまでの距離で，最後の要素は軸位置から軸線までの距離（通常は 0）．正の数値は作図領域の外側へと測り，負の数値は内側へと測る．

`tck=0.01`	図形領域のサイズの分数としての刻みマークの長さ．`tck` が小さいとき（0.5 より小） x 軸と y 軸の刻みは同じ大きさにされる．1 の値は格子を与える．負の値は刻みマークを図形領域の外に与える．刻みマークを内部に置くためには `tck=0.01` や `mgp=c(1,-1.5,0)` を使え．
`xaxs="s"` `yaxs="d"`	それぞれ x 軸と y 軸の軸形式．形式 `"s"`(standard) と `"e"`(extended) では最小と最大の刻みマークはデータの範囲の外にある．拡張された軸は，どれかの点が端に非常に近いとき，わずかに広げられるかも知れない．この軸の形式は場合によっては，端の近くに大きな空白の隙間を残す．形式 `"i"`(internal) と `"r"`（既定）では刻みマークはデータの範囲に入るが，形式 `"r"` は端付近に若干の空白を残す．
このパラメータを `"d"(direct axis)` にすると，現在の軸を「ロック」し，その後の全ての（もしくは，少なくともそのパラメータが，上の他の値のどれかに設定されるまで）プロットに対してそれを利用する．固定された尺度の一連のプロットを生成する場合に便利である．	

9.5.3 図表の余白

R の単一のプロットは「図 (figure)」として知られ，余白（軸ラベル，タイトルなどを含むかも知れない）に囲まれる「作図領域」からなり（普通）軸そのもので仕切られている．典型的な図が図 10 にある．図表のレイアウトを制御する作図パラメータが示されている：

`mai=` `c(1,0.5,0.5,0)`	それぞれ下・左・上・右余白の幅で，インチ単位．
`mar=c(4,2,2,1)`	単位がテキスト行の高さであることを除き `mai` と同様．

`mar` と `mai` は，どちらかを設定すると他方の値を変化させると言う意味で同等である．このパラメータに選ばれている既定値は通常大きすぎる，つまり右側の余白は滅多に必要でなく，表題が無い場合には上側余白も同じく必要ないためである．下側余白と左側余白は，軸と目盛りラベルを入れるのに十分な大きさである必要がある．更に既定値はデバイス面の大きさを考慮せずに選ばれている．例えば `postscript()` デバイスを引数 `height=4` で利用すると `mar` や `mai` がはっきり指定されない限り，50% の余白を持つプロットになる．複数図表が使われたとき（後をみよ）は，余白は半分に縮小されるが，これは多くの図表が同一ページを共有するには十分ではないかも知れない．

9.5.4 複数図表用の環境

R は図の $n \times m$ 配列を一つのページに作成することが可能である．それぞれの図表はそれ自身の余白を持ち，図の配列は，図 11 に示されるように，オプションで「外側余白」で囲まれる．

複数図表に関係する作図パラメータは以下の通りである：

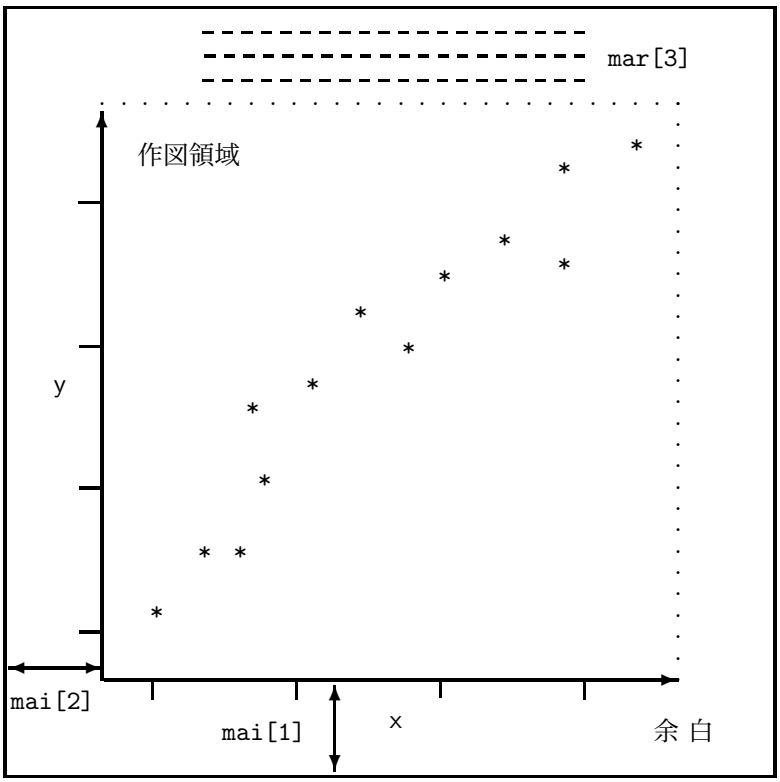

図 10: R の図の解剖図

`mfcol=c(3,2)` `mfrow=c(2,4)`	複数図の配列の大きさを指定する．第一の値は行数で，第二は列数である．これらの二つのパラメータの唯一の違いは `mfcol` の指定が列順で図を埋め `mfrow` は行順で埋めるということだけである．図 11 における配置は `mfrow=c(3,2)` と指定することにより作られるであろう；図は 4 つの図が描かれた後のページを示す．
`mfg=c(2,2,3,2)`	複数図表環境における現在の図表の位置．最初の二つの数値は，現在の図表の行と列，最後の二つは複数図表配列にある行と列の数である．配列中の図の間をジャンプするためにこのパラメータを用いよ．同じページの異なった大きさの図表にたいし,「真の」値とは異なる値を後の 2 つの値に使うことすらできる．
`fig=c(4,9,1,4)/10`	そのページにおける現在の図表の位置．値は左下隅から測ったページのパーセンテージとしての左・右・下・上端の位置である．例の値は，ページの右下に図を置くためのものである．ページ内の任意の位置に図表を置くために，このパラメータを指定せよ．
`oma=c(2,0,3,0)` `omi=c(0,0,0.8,0)`	外側余白のサイズ．`mar` や `mai` と同様に，最初はテキスト行単位で数え，二つ目はインチ単位で下側余白から時計回りに指定する．

外側余白は特にページ毎の表題などのためにとくに有用である．テキストは `outer=T` 引数を付けた `mtext()` 関数で，外側余白に追加することができる．しかしながら，既定では外側余白はなく，従って `oma` や `omi` を使って明示的に作成しなければならない．

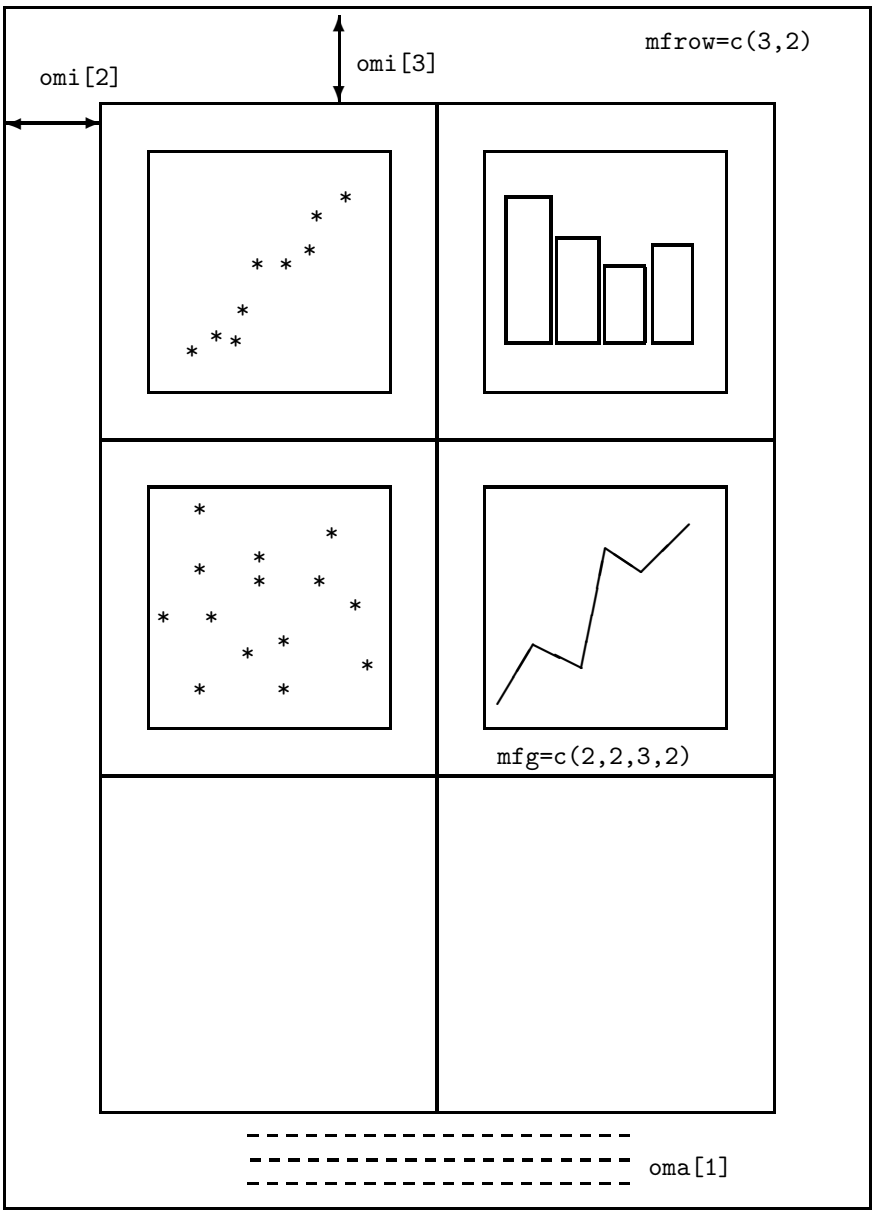

図 11: 複数図表モードにおけるページのレイアウト

9.6 デバイスドライバ

R はほとんどいかなるタイプの表示・印刷デバイスにおいても,（異なる品位水準の）グラフを作ることができる．しかしながらその前に R はどんなタイプのデバイスを扱っているのか知らされる必要がある．これは「デバイスドライバー (*device driver*)」を起動することによりなされる．デバイスドライバーの目的は R からの作図指示（例えば「線を引く」）を特定のデバイスが理解できる形式に変換することである．

デバイスドライバーはデバイスドライバー関数の呼び出しにより起動される．全てのデバイス指定に対してそのような関数が一つある：それらの全てのリストを得るには `help(Devices)` と入力せよ．例えば，次の命令を実行すると

```
> postscript()
```

以降の全てのグラフ出力は「ポストスクリプト」形式でプリンターに送られる．一般に使われるデバイスドライバーとしては：

`X11()`	「X11」ウィンドウシステム用.
`postscript()`	「ポストスクリプト」プリンタへの印刷，もしくは「ポストスクリプト」形式のグラフィックファイルの作成用.
`pictex()`	LaTeX ファイルを生成する.

あるデバイスの利用が終了した場合には，次の命令を実行して確実にデバイスドライバーを停止させる.

```
> dev.off()
```

これにより，そのデバイスがきれいに終了する．例えばハードコピー (hardcopy) デバイスの場合，この命令により全てのページが完結させられ，プリンタに送られる.

9.6.1 タイプセット文書に対する「ポストスクリプト」図表

`postscript()` デバイス指定関数に `file` 引数を渡すことにより，指定したファイルに「ポストスクリプト (PostScript)」形式でグラフを保存できる．そのプロットは `horizontal=FALSE` 引数を与えない限り横置きであり，`width` や `height` 引数でグラフのサイズを調整できる.（プロットはこれらの大きさに合うように適当にスケールが調整される.）例えば，命令

```
> postscript("file.ps", horizontal=FALSE, height=5)
```

は文書に収めるであろう 5 インチの高さの図表の「ポストスクリプト」コードを含むファイルを生成する.[12]命令中のファイル名が，既に存在するものであったとき，そのファイルは上書きされることを心に留めておくことが重要である．これは同一の R のセッション中で先に作成したファイルに対してすら当てはまる.

9.6.2 複数の作図デバイス

R の高度な利用においては，同時に使用される複数の作図デバイスを持つことがしばしば有益である．もちろん一時にはただ一つの作図デバイスだけが作図命令を受け入れることができ，これは「現在のデバイス (*current device*)」と呼ばれている．複数のデバイスが開かれているとき，それらは任意位置のデバイスの種類を与える名前とともに，番号順の列を作っている.

複数のデバイスを操作するための主要な命令とその意味は以下の通り：

`X11()`
`postscript()`
...
	デバイスドライバー関数の新規呼出しは新しい作図デバイスを開き，したがってデバイスリストを一つ伸長する．このデバイスが現在のデバイスとなり，そこにグラフ出力が送られる.

`dev.list()`	全てのアクティブなデバイスの番号と名前を返す．リストにおける第一位置のデバイスは，常に「無効デバイス (*null device*)」であり，これは全く作図命令を受け付けない.
`dev.next()` `dev.prev()`	それぞれ，現在のデバイスの次，および前の位置の作図デバイスの番号と名前を返す.
`dev.set(which=k)`	現在のデバイスをデバイスリストの `k` 番目の位置のデバイスに変更するために使うことができる．そのデバイスの番号とラベルを返す.

[12]**注意:** `postscript()` デバイスドライバーにより生成された「ポストスクリプト」コードは EPS(Encapsulated PostScript) では「なく」，したがって文書に電子的に取り込む（物理的な切り貼りとは異なり）ことは面倒になるかも知れない．少なくとも LaTeX では問題が無いことはわかっている.

`dev.off(k)`	デバイスリストの `k` 番目の位置の作図デバイスを終了させる．`postscript` デバイスなどのいくつかのデバイスに対して，そのデバイスの初期化の方法に依存して，すぐにファイルを印刷するか，後で印刷するために正しくファイルを完結させる．
`dev.copy(device, ..., which=k)` `dev.print(device, ..., which=k)`	デバイス `k` のコピーを作る．ここで `device` は `postscript` などのデバイス関数で，必要ならば `...` で指示される余分な引数を伴う．`dev.print` は似ているが，複製されたデバイスは直ちに閉じ，ハードコピー印刷等の終了処理が直ちに行われる．(`printgraph()` も参照せよ)．
`graphics.off()`	無効デバイスを除き，リスト中の全ての作図デバイスを終了する．

A R：入門的なセッション

以下のセッションは R 環境のいくつかの特徴を，それを使ってみることにより紹介することを目標にしている．システムの多くの特徴は，最初なじみがなく困惑させられるであろうが，すぐに理解できるであろう．このセッションは Unix ユーザー向けに書かれている．Macintosh や Windows の利用者は議論を適当に変更する必要がある．

`login:` `...`	ログイン，ウインドウシステムを立ち上げる．データファイル `morley.data` が作業ディレクトリーになければならない．もししなければ，近くの詳しい人に尋ねよ．もしあれば，先に進む．
`> R`	`R` を開始する．
	R プログラムが始まり，起動画面が現れる． （R 内では左側のプロンプトは混乱を避けるために表示しない.）グラフィックスウインドウが自動的に画面に現れるはずである．

`x <- rnorm(50)`
`y <- rnorm(x)`
`plot(x, y)`
その点を画面にプロットする．
	$x-$ と $y-$ 座標用の二つの疑似正規乱数ベクトルを生成する．

`objects()`	現在の R イメージにどのような R のオブジェクトがあるか見る．
`rm(x,y)`	不要なオブジェクトを消し去る（清掃）．
`x <- 1:20`	$x = (1, 2, \ldots, 20)$ とする．
`w <- 1 + sqrt(x)/2`	標準偏差の「重み」ベクトル．

`dummy <- data.frame(x=x,`
`  y= x + rnorm(x)*w)`
`dummy`
	二つの列ベクトル x と y の「データフレーム」を作りそれを見る．

`fm <- lm(y~x, data=dummy)` `summary(fm)`	y の x への単純線形回帰当てはめを行い，解析結果を眺める．

`fm1 <- lm(y~x, data=dummy,`
`  weight=1/w^2)`
`summary(fm1)`
	標準偏差が分かっているから，重み付き回帰を行うことができる．

`attach(dummy)`	データフレーム中の列ベクトルを変数として見えるようにする．
`lrf <- lowess(x, y)`	ノンパラメトリックな局所回帰関数を作る．
`plot(x, y)`	標準的な点プロット．
`lines(x, lrf$y)`	局所回帰に加える．
`abline(0, 1, lty=3)`	真の回帰直線：(切片 0，傾き 1)．
`abline(coef(fm))`	重み無しの回帰直線.
`abline(coef(fm1),lty=4)`	重み付の回帰直線.
`detach()`	データフレームを検索リストから除く．

`plot(fitted(fm),`
`  resid(fm),`
`  xlab="Fitted values",`
`  ylab="Residuals", main=`
`  "Residuals vs Fitted")`
	標準偏差の不均一性をチェックするための標準的な回帰診断プロット．不均一性が認められるだろうか？

`qqnorm(resid(fm), main=` `  "Residuals Rankit Plot")`	歪み，尖り，そして外れ値をチェックするための正規スコアプロット.(ここではあまり役にたたない.)
`rm(fm,fm1,lrf,x,dummy)`	再び掃除.

```
                                  次のセクションは Michaelson と Morley による古典的な光速の測実験
                                  からのデータを調べる．
system("more morley.data")        オプション．一時的に R を中断しファイルを眺める．関数 system は
                                  OS のコマンドを起動する手段である．これは Unix でのみ使える．
mm <- read.table(                 Michaelson と Morley データをデータフレームとして読み込み眺める．
  "morley.data")                  五つの実験 (col. Expt) があり，各々は 20 回の試行からなる (col. Run),
mm                                そして sl は適当に変換された光速値である．
mm$Expt <- factor(mm$Expt)        Expt と Run を因子に変換する．
mm$Run <- factor(mm$Run)
attach(mm)                        データフレームを位置 2（デフォールト）で見えるようにする．
plot(Expt,Speed, main=            五種類の実験を単純案な箱型図で比較する．
  "Speed of Light Data",
  xlab="Experiment No.")
fm <- aov(Speed~Run+Expt,         「runs」と「experiments」を因子とし，乱塊 (randomized block) とし
  data=mm)                        て解析する．
summary(fm)
fm0 <- update(fm,                 「runs」を除いたサブモデルに当てはめを行い，形式的な分散分析を用
  .~.-Run)                        いて比較する．
anova(fm0,fm)
detach()                          次に移る前に掃除する．
rm(fm, fm0)
```

```
                                  ここで等高線，イメージプロットという作図機能を眺める．
x <- seq(-pi, pi, len=50)         x は −π ≤ x ≤ π を等間隔に分ける 50 個の値のベクトル．y も同様．
y <- x
f <- outer(x, y,                  f は関数 cos(y)/(1 + x²) の値からなる正方行列で，行と列はそれぞれ
  function(x,y)                   x と y を添字に持つ．
    cos(y)/(1+x^2))
oldpar <- par()                   作図パラメータを保存し，作図領域を「正方形」にする．
par(pty="s")
contour(x, y, f)                  f の等高線図を描く；詳しく見るために線の数を増やす．
contour(x, y, f,
  nlevels=15, add=T)
fa <- (f-t(f))/2                  fa は f の「非対称部分」(t() は転置)．
contour(x, y, fa, nint=15)        等高線図を作る，...
par(oldpar)                       ... そして元の作図パラメータを復帰する．
persp(x, y, f)                    高密度のきれいなイメージプロットを作る（もし希望すればハードコ
image(x, y, f)                    ピーを取ることができる)．
image(x, y, fa)
objects(); rm(x,y,f,fa)           次に移る前に掃除する．
```

```
th <- seq(-pi, pi,                R は複素数の演算もできる．1i は虚数 i の代わりに使われる．
  len=100)
z <- exp(1i*th)
par(pty="s")                      複素数引数のプロットとは，虚数部を実数部にたいしてプロットするこ
plot(z, type="l")                 とを意味する．結果は円になるはずである．
w <- rnorm(100) +                 単位円の内部から標本点を取りたいとする．一つの方法は標準 正規分
  rnorm(100)*1i                   布に従う実数部と虚数部を持つ複素数を取り...
w <- ifelse(Mod(w) > 1,           そして，円の外部にあるものは逆数を取って円の内部に写像することで
  1/w, w)                         ある．
```

```
plot(w, xlim=c(-1,1),
  ylim=c(-1,1), pch="+",
  xlab="x", ylab="y")
lines(z)
```
すべての点は円の内部にあるが，その分布は一様ではない．

```
w <- sqrt(runif(100))*
  exp(2*pi*runif(100)*1i)
plot(w, xlim=c(-1,1),
  ylim=c(-1,1), pch="+",
  xlab="x", ylab="y")
lines(z)
```
二つ目の方法は一様分布を使う．点は今やより円の内部に均一に散らばるように見える．

```
rm(th,w,z)
```
再び掃除．

```
q()
```
R プログラムを終了する...

```
>
```
... UNIX に戻る．

B R の組み込みコマンド行エディター

B.1 準備

1991 年 8 月にリリースされた R は組み込みコマンド行エディターを持ち，過去のコマンドの再呼出し，編集，再実行ができる．

エディターを使うには R プログラムを

```
$ Splus -e
```

で始める．エディターの内部では，シェルの環境変数 S_CLEDITOR に応じて，emacs または vi 風のキー操作を使うことができる．emacs 風のキー操作を使うためには，csh もしくはその互換シェルでは

```
$ setenv S_CLEDITOR emacs
```

とし，vi 風のキー操作を使うには ここで emacs の代わりに vi と置く．この実行文は普通ユーザーの .login ファイル（もしくはそれに代わるもの）に含められ，そうすればログイン時に自動的に実行されるであろう．-e オプションを含めるのを忘れないための .cshrx ファイルにたいする簡便なエイリアスは，例えば

```
alias S+ 'Splus -e'
```

であり，そうすればコマンド行エディター付きで R を開始する命令は S+ になる．

次のような普通のタイピング時の便法が使われる： ^M は「m キーを押しながら Control キーを押す」， Esc m は「Esc キーを最初に押し，それから m キーを押す」ことを表す．Esc の後では大・子文字の区別が意味を持つことを注意せよ．

B.2 編集動作

R プログラムはタイプされたコマンドの履歴をエラーがある行も含め保存し，履歴中のコマンドは再呼出しができ，必要なら変更の上新しいコマンドとして再実行できる．*emacs* 流のコマンド行編集では，この編集状態中に打ち込まれたすべてのタイプは編集中のコマンドに対応する文字を挿入し，カーソルより右にあるすべての文字を移動させる．vi モードでは，文字挿入モードは Esc i か Esc a で始まり，文字をタイプした後，再度の Esc で挿入 モードを抜ける．

いかなる時でも Return キーを押せばコマンドが再実行される．

その他の編集動作を次の表にまとめておこう．

残念ながら移動キーを例えば矢印キーに割り当てることは不可能のようであり，少し面倒である．

B.3 コマンド行エディター要約

1. コマンド再呼出と垂直移動	emacs スタイル	vi スタイル
直前のコマンドへ移動（履歴中を後進）	^P	Esc k
直後のコマンドへ移動（履歴中を前進）	^N	Esc j
text 文字列を含む直近のコマンドを探索	^R text	Esc ? text

2. カーソルの水平移動		
コマンドの先頭へ移動	^A	Esc ^
行末へ移動	^E	Esc $
一単語分戻る	Esc b	Esc b
一単語分進む	Esc f	Esc w
一文字分戻る	^B	Esc h
一文字分進む	^F	Esc l

3. 編集と再実行		
カーソル位置に text を挿入	text	Esc i text Esc
カーソルの後に text を挿入	^Ftext	Esc a text Esc
直前（カーソルの左）の文字を消去	Delete	Esc shift-x
カーソル位置のの文字を消去	^D	Esc x
カーソル位置の単語の残りを消しそれを「保存」	Esc d	Esc dw
カーソル位置からコマンドの最後までを消しそれを「保存」	^K	Esc shift-d
直近に「セーブ」されたテキストをここに挿入 (yank)	^Y	Esc shift-y
カーソル位置の文字と次の文字を入れ換える	^T	Esc xp
単語の残りの部分を小文字化	Esc l	
単語の残りの部分を大文字化	Esc l	
コマンドを R で再実行	Return	Return

注意：vi スタイルのコマンドでは，vi 流に Esc は最初の再読み込み命令の前に実行し，挿入と追加コマンドを終了しさえれば良い．

最後の Return はどちらのコマンドスタイルでも，コマンド行の編集列を終了させる．

C 実習

C.1 濁り点 (cloud point) データ

出典: Draper & Smith, *Applied Regression Analysis*, p. 162
カテゴリー: 多項式回帰. 散布図.

解説

液体の濁り点（訳注：非イオン系界面活性剤を加熱していくと，成分が析出し白濁し始める温度）は資料の結晶化の温度の尺度で，屈折率により測定される．基準資料の I_8 のパーセント値が，2 次式あるいは 3 次式のモデルを用いると，優れた予測値になることが示唆されてきた:

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + E, \qquad E \sim \mathrm{N}(0, \sigma^2)$$

データ

次のデータは既知の I_8 パーセント値を持つ資料から集められた:

I_8%	Cloud Point	I_8%	Cloud Point	I_8%	Cloud Point	I_8%	Cloud Point
0	21.9	2	26.1	5	28.9	8	31.4
0	22.1	3	26.8	6	29.8	8	31.5
0	22.8	3	27.3	6	30.0	9	31.8
1	24.5	4	28.2	6	30.3	10	33.1
2	26.0	4	28.5	7	30.4		

データはファイル `cloud.data` からデータフレームとして読み込むことが出来る.

提案された解析

関数 `lm()` を使って多項式回帰モデルをあてはめ，次数を注意深く選べ.

C.2 Janka 硬度データ

出典: E. J. Williams: *Regression Analysis*, Wiley, 1959.
カテゴリー: 多項式回帰. 変換.

Description

Janka 硬度はオーストラリアの木材の重要な構造的特性であるが，測定が難しい．しかしそれは木材の密度と関連しており，これは比較的測定が容易である．低次の多項式回帰が適当であるといわれている．

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + E$$

ここで Y は硬度，x は密度である．

データ

次のデータは 36 本のオーストラリア産ユーカリ硬木のサンプルから採取したものである．

D	H	D	H	D	H	D	H	D	H	D	H
24.7	484	30.3	587	39.4	1210	42.9	1270	53.4	1880	59.8	1940
24.8	427	32.7	704	39.9	989	45.8	1180	56.0	1980	66.0	3260
27.3	413	35.6	979	40.3	1160	46.9	1400	56.5	1820	67.4	2700
28.4	517	38.5	914	40.6	1010	48.2	1760	57.3	2020	68.8	2890
28.4	549	38.8	1070	40.7	1100	51.5	1710	57.6	1980	69.1	2740
29.0	648	39.3	1020	40.7	1130	51.5	2010	59.2	2310	69.1	3140

データはファイル `janka.data` からデータフレームとして読むことができる．

提案された解析

次数を注意深く選び、多項式回帰モデルにあてはめよ．残差を検討し、明らかな外れ値や分散の不均一を調べよ．あてはめ値に対してプロットされた残差のパターンにたいし, 平方根変換や対数変換がどのような効果を持つかチェックせよ．

より進んだ解析: 平均値に比例した分散と平方根リンクを持つ擬似尤度モデルを考慮せよ．

C.3 Tuggeranong の住宅価格のデータ

出典: Dr Ray Correll, Personal communication
カテゴリー: 重回帰分析, 条件付き散布図 (coplot).

Description

1987 年 2 月，Tuggeranong において，慎重な住宅購入希望者が住宅を買う前に市場での住宅に関するいくつかのデータを集めた．その 20 の住宅に関するデータが表に示され、また `house.dat` というファイルとして利用することができる．集められた変数は順に，価格・総床面積・敷地面積・居室の数・築年数・セントラルヒーティングの有無，である。

データ

Data

データは表 5 に与えられ，ファイル `house.data` として利用可能である．

Price ($000s)	Floor (m^2)	Block (m^2)	Rooms	Age (years)	Cent. Heat.
52.00	111.0	830	5	6.2	no
54.75	128.0	710	5	7.5	no
57.50	101.0	1000	5	4.2	no
57.50	131.0	690	6	8.8	no
59.75	93.0	900	5	1.9	yes
62.50	112.0	640	6	5.2	no
64.75	137.6	700	6	6.6	yes
67.25	148.5	740	6	2.3	no
67.50	113.5	660	6	6.1	no
69.75	152.0	645	7	9.2	no
70.00	121.5	730	5	4.3	yes
75.50	141.0	730	7	4.3	no
77.50	124.0	670	6	1.0	yes
77.50	153.5	795	7	7.0	yes
81.25	149.0	900	6	3.6	yes
82.50	135.0	810	6	1.7	yes
86.25	162.0	930	6	1.2	yes
87.50	145.0	825	6	0.0	yes
88.00	172.0	950	7	2.3	yes
92.00	170.5	870	7	0.7	yes

表 5: Tuggeranong の住宅価格のデータ

提案された解析

変数 `Age` と `CentHeat` で条件付けて `coplot()` を用いてデータを探索せよ．

注意深く重回帰モデルを選び，外れ値をチェックせよ（たとえばバーゲンや暴利販売があり得る）．

C.4 Yorke 半島の小麦の収穫量データ

出典: K. W. Morris (private communication)
カテゴリー: 重回帰分析.

解説

1931 年から 1955 年に至る南オーストラリアの Yorke 半島の小麦の周辺栽培地区での年間収穫量と，成長期 3ヶ月間の雨量．年代そのものが潜在的に品種と農業技術の改良を説明する代用的予測値となる．収穫量 (Yield) の単位はエーカーあたりのブッシェル，雨量 (rainfall) はインチ単位である．

データ

データは表 6 に与えられ，ファイル `sawheat.data` からデータフレームとして読み込むことが出きる．

Year	Rain0	Rain1	Rain2	Yield	Year	Rain0	Rain1	Rain2	Yield
1931	.05	1.61	3.52	.31	1944	3.30	4.19	2.11	4.60
1932	1.15	.60	3.46	.00	1945	.44	3.41	1.55	.35
1933	2.22	4.94	3.06	5.47	1946	.50	3.26	1.20	.00
1934	1.19	11.26	4.91	16.73	1947	.18	1.52	1.80	.00
1935	1.40	10.95	4.23	10.54	1948	.80	3.25	3.55	2.98
1936	2.96	4.96	.11	5.89	1949	7.08	5.93	.93	11.89
1937	2.68	.67	2.17	.03	1950	2.54	4.71	2.51	6.56
1938	3.66	8.49	11.95	16.03	1951	1.08	3.37	4.02	1.30
1939	5.15	3.60	2.18	6.57	1952	.22	3.24	4.93	.03
1940	6.44	2.69	1.37	8.43	1953	.55	1.78	1.97	.00
1941	2.01	6.88	.92	8.68	1954	1.65	3.22	1.65	3.09
1942	.73	3.30	3.97	2.49	1955	.72	3.42	3.31	2.72
1943	2.52	1.93	1.16	.98					

表 6: Yorke 半島の小麦の収穫量データ

提案された解析

重回帰モデルをあてはめ、特に残差プロットを用いて結果を確認せよ．

C.5 アイオワ州の小麦の収穫量データ

出典: CAED Report, 1964. Quoted in Draper & Smith.
カテゴリー: 重回帰分析，診断．

解説

データは 1930 年から 1962 年までの合衆国アイオワ州における，発育期 3ヶ月およびその直前の季節の降雨量，発育期 3ヶ月および収穫月の平均気温，年代，そして小麦収穫量からなる．

Data

データは表 7 に与えられ，ファイル `iowheat.data` からデータフレームとして読み込むことができる．

提案された解析

重回帰分析をあてはめ，予測変数を慎重に選べ．変数の前進選択もしくは後退消去を行え．各予測変数に対して残差を順にプロットすることにより，残差を吟味せよ．

説明変数の 2 次の項を加えた際の効果を考慮せよ．

このデータを良く似た期間のデータである Yorke 半島のデータと比較すると面白い．

C.6　ガソリンの収量データ

出典: *Estimate gasoline yields from crudes*
by Nilon H. Prater, Petroleum Refiner, **35**, #5.
カテゴリー: 分散・共分散分析と重回帰，現代的な回帰

解説

データは原油のパーセント値で表されたガソリン収量，y と置く，と収量に関連するかも知れないくつかの独立変数からなる．それらは，

x_1: ^{0}API で表された原油の重量,

x_2: 原油の蒸発圧力,

x_3: 原油の 10% 点， ASTM,

x_4: ガソリンの end point（訳注：化学反応が進行する際に観測される最大温度）.

データは 10 個の別々のサンプルから得られ，同じサンプル内では x_1, x_2 そして x_3 は同じ値である．

Year	Rain0	Temp1	Rain1	Temp2	Rain2	Temp3	Rain3	Temp4	Yield
1930	17.75	60.2	5.83	69.0	1.49	77.9	2.42	74.4	34.0
1931	14.76	57.5	3.83	75.0	2.72	77.2	3.30	72.6	32.9
1932	27.99	62.3	5.17	72.0	3.12	75.8	7.10	72.2	43.0
1933	16.76	60.5	1.64	77.8	3.45	76.4	3.01	70.5	40.0
1934	11.36	69.5	3.49	77.2	3.85	79.7	2.84	73.4	23.0
1935	22.71	55.0	7.00	65.9	3.35	79.4	2.42	73.6	38.4
1936	17.91	66.2	2.85	70.1	0.51	83.4	3.48	79.2	20.0
1937	23.31	61.8	3.80	69.0	2.63	75.9	3.99	77.8	44.6
1938	18.53	59.5	4.67	69.2	4.24	76.5	3.82	75.7	46.3
1939	18.56	66.4	5.32	71.4	3.15	76.2	4.72	70.7	52.2
1940	12.45	58.4	3.56	71.3	4.57	76.7	6.44	70.7	52.3
1941	16.05	66.0	6.20	70.0	2.24	75.1	1.94	75.1	51.0
1942	27.10	59.3	5.93	69.7	4.89	74.3	3.17	72.2	59.9
1943	19.05	57.5	6.16	71.6	4.56	75.4	5.07	74.0	54.7
1944	20.79	64.6	5.88	71.7	3.73	72.6	5.88	71.8	52.0
1945	21.88	55.1	4.70	64.1	2.96	72.1	3.43	72.5	43.5
1946	20.02	56.5	6.41	69.8	2.45	73.8	3.56	68.9	56.7
1947	23.17	55.6	10.39	66.3	1.72	72.8	1.49	80.6	30.5
1948	19.15	59.2	3.42	68.6	4.14	75.0	2.54	73.9	60.5
1949	18.28	63.5	5.51	72.4	3.47	76.2	2.34	73.0	46.1
1950	18.45	59.8	5.70	68.4	4.65	69.7	2.39	67.7	48.2
1951	22.00	62.2	6.11	65.2	4.45	72.1	6.21	70.5	43.1
1952	19.05	59.6	5.40	74.2	3.84	74.7	4.78	70.0	62.2
1953	15.67	60.0	5.31	73.2	3.28	74.6	2.33	73.2	52.9
1954	15.92	55.6	6.36	72.9	1.79	77.4	7.10	72.1	53.9
1955	16.75	63.6	3.07	67.2	3.29	79.8	1.79	77.2	48.4
1956	12.34	62.4	2.56	74.7	4.51	72.7	4.42	73.0	52.8
1957	15.82	59.0	4.84	68.9	3.54	77.9	3.76	72.9	62.1
1958	15.24	62.5	3.80	66.4	7.55	70.5	2.55	73.0	66.0
1959	21.72	62.8	4.11	71.5	2.29	72.3	4.92	76.3	64.2
1960	25.08	59.7	4.43	67.4	2.76	72.6	5.36	73.2	63.2
1961	17.79	57.4	3.36	69.4	5.51	72.6	3.04	72.4	75.4
1962	26.61	66.6	3.12	69.1	6.27	71.6	4.31	72.5	76.0

表 7: アイオワ州の小麦収穫量の歴史的データ

データ

データは表 8 に示され，ファイル `oil.data` からデータフレームとして読み込むことができる．

Sample	x_1	x_2	x_3	x_4	y	Sample	x_1	x_2	x_3	x_4	y
1	31.8	0.2	316	365	8.5	6	40.0	6.1	217	212	7.4
1	31.8	0.2	316	379	14.7	6	40.0	6.1	217	272	18.2
1	31.8	0.2	316	428	18.0	6	40.0	6.1	217	340	30.4
2	32.2	2.4	284	351	14.0	7	40.3	4.8	231	307	14.4
2	32.2	2.4	284	424	23.2	7	40.3	4.8	231	367	26.8
						7	40.3	4.8	231	395	34.9
3	32.2	5.2	236	267	10.0	8	40.8	3.5	210	218	8.0
3	32.2	5.2	236	360	24.8	8	40.8	3.5	210	273	13.1
3	32.2	5.2	236	402	31.7	8	40.8	3.5	210	347	26.6
4	38.1	1.2	274	285	5.0	9	41.3	1.8	267	235	2.8
4	38.1	1.2	274	365	17.6	9	41.3	1.8	267	275	6.4
4	38.1	1.2	274	444	32.1	9	41.3	1.8	267	358	16.1
						9	41.3	1.8	267	416	27.8
5	38.4	6.1	220	235	6.9	10	50.8	8.6	190	205	12.2
5	38.4	6.1	220	300	15.2	10	50.8	8.6	190	275	22.3
5	38.4	6.1	220	365	26.0	10	50.8	8.6	190	345	34.7
5	38.4	6.1	220	410	33.6	10	50.8	8.6	190	407	45.7

表 8: ガソリンの精製データ

提案された解析

`EndPt` を共変数として用い，サンプル間の差異が他の説明変数への回帰モデルによって説明可能かどうか確認せよ．

より進んだ解析：サンプル間とサンプル内を二つの層として用い、2 元配置分散分析を当てはめよ．

C.7 Michaelson と Morley の光の速度のデータ

出典: Weekes: A Genstat Primer.
カテゴリー: Analysis of Variance.

解説

Michaelson と Morley による光の速度に関する古典的なデータ．データは 5 組の実験からなり，各実験は 20 回の連続した「計測」(runs) からなる．応答は光速の測定値であり，適当にコード化されている．ここでデータは *experiment* と *run* を因子とする乱塊法実験計画として見ることができる．*run* はまた，一組の実験内での線形（または多項式）的な測定の変化を説明する量的な変数と考えることができるかもしれない．

データ

データは表 9 に与えられ，ファイル `morley.data` からデータフレームとして読み込むことができる．

Runs 1–10				
$\mathcal{E}_1$	$\mathcal{E}_2$	$\mathcal{E}_3$	$\mathcal{E}_4$	$\mathcal{E}_5$
850	960	880	890	890
740	940	880	810	840
900	960	880	810	780
1070	940	860	820	810
930	880	720	800	760
850	800	720	770	810
950	850	620	760	790
980	880	860	740	810
980	900	970	750	820
880	840	950	760	850

Runs 11–20				
$\mathcal{E}_1$	$\mathcal{E}_2$	$\mathcal{E}_3$	$\mathcal{E}_4$	$\mathcal{E}_5$
1000	830	880	910	870
980	790	910	920	870
930	810	850	890	810
650	880	870	860	740
760	880	840	880	810
810	830	840	720	940
1000	800	850	840	950
1000	790	840	850	800
960	760	840	850	810
960	800	840	780	870

表 9: Michaelson と Morley の光の速度のデータ

提案された解析

一元配置分散分析モデルを使って各実験間の差異をチェックし，結論を要約せよ．

C.8 ラットの遺伝子型データ

出典: Scheffe, H.: *The Analysis of Variance*「分散分析」に引用されている
カテゴリー: アンバランスな二重分類.

解説

A, F, I, J の 4 種類の遺伝子型のラットの母親と一腹の子の育成給餌実験データ．測定は給餌期間後の同腹の子ラットの体重の増加.

データ

表 10 に与えられたデータは `genotype.data` ファイルからのデータフレームとして読み込むことができる.

同腹の子ラットの遺伝子型	母ラットの遺伝子型			
	A	F	I	J
A	61.5	55.0	52.5	42.0
	68.2	42.0	61.8	54.0
	64.0	60.2	49.5	61.0
	65.0		52.7	48.2
	59.7			39.6
F	60.3	50.8	56.5	51.3
	51.7	64.7	59.0	40.5
	49.3	61.7	47.2	
	48.0	64.0	53.0	
		62.0		
I	37.0	56.3	39.7	50.0
	36.3	69.8	46.0	43.8
	68.0	67.0	61.3	54.5
				55.3
				55.7
J	59.0	59.5	45.2	44.8
	57.4	52.8	57.0	51.5
	54.0	56.0	61.4	53.0
	47.0			42.0
				54.0

表 10: ラットの遺伝子型データ

提案された解析

二重分類モデルを当てはめよ．形式的な解析と，`interaction.plot()` を使ったグラフィカルな方法の両方を用いて交互作用をチェックせよ．主効果を検定し要約せよ．

C.9 Fisher の砂糖大根データ

出典: R. A. Fisher, *Design of Experiments*「実験計画」.
カテゴリー: 分散共分散分析.

解説

大きさ 24 の 4 つのブロックにおける，古典的な 3×2^3 の乱塊法による実験．反応は，その区画からとられた砂糖大根の根の総重量であるが，これは観測された根の数を伴ったものである．株の数

は，異なる大きさの区画の共変量であろうと示唆されている．

因子は，種類 (*Variety*) (a, b , c の 3 水準) とそれぞれ「あり」「なし」の 2 水準の N, P, K（訳注：窒素，燐酸，カリ肥料）．

データ

表 11 に与えられたデータは `sugar.data` ファイルからデータフレームを作るのに適した形で読み込むことが出来る．

				ブロック 1		ブロック 2		ブロック 3		ブロック 4	
V	N	P	K	No	Wt	No	Wt	No	Wt	No	Wt
a	$-$	$-$	$-$	124	162	133	162	114	127	127	158
a	$-$	$-$	k	131	152	161	164	130	141	145	188
a	$-$	p	$-$	115	173	134	175	134	142	109	162
a	$-$	p	k	126	140	133	158	106	148	132	160
a	n	$-$	$-$	136	184	134	178	127	168	139	199
a	n	$-$	k	134	112	156	193	101	171	138	191
a	n	p	$-$	132	190	104	166	119	157	132	193
a	n	p	k	120	175	147	155	107	139	148	192
b	$-$	$-$	$-$	145	133	147	130	139	138	127	128
b	$-$	$-$	k	156	117	152	137	107	121	147	147
b	$-$	p	$-$	152	140	138	101	125	124	120	143
b	$-$	p	k	137	127	145	132	125	132	143	139
b	n	$-$	$-$	124	163	138	159	140	166	159	174
b	n	$-$	k	136	143	142	144	133	142	148	159
b	n	p	$-$	140	168	142	150	133	118	138	157
b	n	p	k	146	144	135	160	138	155	140	153
c	$-$	$-$	$-$	113	122	138	132	119	123	127	146
c	$-$	$-$	k	91	107	149	171	118	142	129	151
c	$-$	p	$-$	123	118	139	142	127	120	124	138
c	$-$	p	k	129	140	126	115	129	130	142	152
c	n	$-$	$-$	121	118	141	152	127	149	127	165
c	n	$-$	k	126	148	128	152	107	147	110	136
c	n	p	$-$	103	112	144	175	102	152	143	173
c	n	p	k	120	162	125	160	129	173	137	185

表 11: Fisher の砂糖大根データ

提案された解析

重量 (Wt) を反応，数 (No) を共変量とした乱塊法による実験計画としてデータを解析せよ．そのモデルから全ての不要な交互作用項を取り除き要約せよ．

C.10　大麦の分割区画法の試行

出典: 不明. 伝統的なデータ.
カテゴリー: 多層の分散分析

説明

大麦の種類と肥料（窒素）に関する実験が，3 つの全区画からなる 6 つのブロックを用いて実施された．

それぞれの全区画は 4 つの部分区画に分けられた．実験には 3 種類の大麦が用いられ，各全区画には一つの種類が蒔かれ，それぞれの全区画の 4 つの部分区画において，4 水準（1 エーカーあたり 0, 0.01, 0.02, 0.04 トン）の肥料が使用された．表において V_i は i 番目の種類，N_j は j 番目の窒素の水準を示している．

データ

ブロック	種類	N_1	N_2	N_3	N_4	ブロック	種類	N_1	N_2	N_3	N_4
	V_1	111	130	157	174		V_1	74	89	81	122
I	V_2	117	114	161	141	IV	V_2	64	103	132	133
	V_3	105	140	118	156		V_3	70	89	104	117
	V_1	61	91	97	100		V_1	62	90	100	116
II	V_2	70	108	126	149	V	V_2	80	82	94	126
	V_3	96	124	121	144		V_3	63	70	109	99
	V_1	68	64	112	86		V_1	53	74	118	113
III	V_2	60	102	89	96	VI	V_2	89	82	86	104
	V_3	89	129	132	124		V_3	97	99	119	121

表 12: 分割区画法による大麦の圃場実験

表 12 に与えられたデータは barley.data ファイルからデータフレームとして読み込むことが出来る．

提案された解析

分割区画法による圃場実験として解析し，要約せよ．

C.11 カタツムリの死亡率データ

出典: アデレード大学，動物学科
Category: 一般化線形モデル．

解説

20 のカタツムリのグループが 1, 2, 3, 4 週間の期間（露出），気温（3 水準）と相対湿度（4 水準）を注意深くコントロールした状態に置かれた．カタツムリは 2 種類 A, B があり，実験は $4 \times 3 \times 4 \times 2$ 完全無作為実験計画として行われた．露出期間の終わりに，カタツムリが生存しているかどうか検査されたが，この検査自体は生物にとっては致命的である．実習の目的は，刺激 (stimulus) 変量に基づく生存確率のモデル化であり，特に種類間の差異を検定することである．

このデータは，ほとんどの場合に実験中の死亡が非常に少ない点で異例である．

データ

		相対湿度											
		60.0%			65.8%			70.5%			75.8%		
		温度			温度			温度			温度		
種類	露出	10	15	20	10	15	20	10	15	20	10	15	20
A	1	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	1	1	0	1	0	0	0	0	0	0	0
	3	1	4	5	0	2	4	0	2	3	0	1	2
	4	7	7	7	4	4	7	3	3	5	2	3	3
B	1	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	3	2	0	2	1	0	0	1	1	0	1
	3	7	11	11	4	5	9	2	4	6	2	3	5
	4	12	14	16	10	12	12	5	7	9	4	5	7

表 13: カタツムリの死亡率データ

表 13 に与えられたデータは `snails.data` ファイルからデータフレームとして読み込むことが出来る．

提案された解析

このデータは，極めて小さなセル度数が多くあるにもかかわらず，いくつかの尤度比の大標本理論が非常にうまく当てはまっているように見える，と言う点で興味深い．

ロジットもしくはプロビットリンクを用い二項分布モデルを当てはめよ．（ロジット，プロビット尺度において）それぞれの種類に対する `Temp`（温度），`Humid`（湿度），`Exposure`（露出期間），$\texttt{Exposure}^2$ への並行回帰を用いたモデルが合理的であることを示し，要約せよ．

C.12 Kalythos の盲目データ

出典: S. D. Silvey: *Statistical Inference*. （架空？）
カテゴリー: 一般化線形モデル

解説

ギリシャの Kalythos 島では，男性住民は先天的な目の疾患を患い，その症状は年齢が高いほど顕著になる．いくつかの年代の島の男性の標本が盲目かどうか検査され，その結果が記録された．

データ

年齢:	20	35	45	55	70
検査した人数:	50	50	50	50	50
盲目の人の数:	6	17	26	37	44

表 14: Kalythos の盲目データ

表 14 に与えられたデータは `kalythos.data` ファイルからデータフレームとして読み込むことが出来る．

提案された解析

Suggested analysis

ロジットもしくはプロビットモデルを用いて LD50 を推定せよ，つまり，盲目の人の割合が $p = \frac{1}{2}$ である年齢を標準誤差とともに推定せよ．この点で，ロジットモデルとプロビットモデルがどの程度異なるか調べよ．

C.13 Stormer 粘度計の測定データ

Source: E. J. Williams: *Regression Analysis*, Wiley, 1959
Category: 非線形回帰，特殊な回帰

解説

Stormer 粘度計は液体の粘度 (Viscosity) を，装置内のシリンダを重り (Weight) により回転させ，あらかじめ決められた回数だけ回転するのに要した時間を測ることにより測定する．粘度計は，装置を正確に粘度が知られている液体に漬け，重りを変えながら得られた時間を測ることにより校正される．データはそのような校正測定からきたもので，理論的な考察から時間と重さと粘度の間に

$$T_i = \frac{\beta v_i}{w_i - \theta} + E_i$$

の形式の非線形の関係があることが示唆される．ここで，β と θ は推定されるべき未知パラメータである．

データ

	Weight		
Viscosity	20	50	100
14.7	35.6	17.6	
27.5	54.3	24.3	
42.0	75.6	31.4	
75.7	121.2	47.2	24.6
89.7	150.8	58.3	30.0
146.6	229.0	85.6	41.7
158.3	270.0	101.1	50.3
161.1		92.2	45.1
298.3		187.2	89.0
			86.5

表 15: Stormer 粘度測定データ

表 15 に与えられたデータは `stormer.data` ファイルからデータフレームとして読み込むことが出来る．

提案された解析

`nlr()` 関数を利用して非線形回帰モデルを推定せよ．適当な初期値は回帰モデルを次の形

$$w_i T_i = \beta v_i + \theta T_i + (w_i - \theta) E_i$$

に取り，w_iT_i を v_i と T_i に通常の線形回帰することにより得られる．

C.14　塩素の利用可能性データ

Source:　Draper & Smith, *Applied Regression Analysis* ,（改変した）.
Category: 非線形回帰

解説

以下の工業化学データのセットは，製造後のさまざまな経過時間後におけるある製品中の利用可能な塩素量を示している．

$$Y = \beta_0 + \beta_1 \exp(-\theta x)$$

の形式の，塩素量の減少に対する非線形回帰モデルが理論的見地から提案されている．ここで y は時間 x における残存量である．

Data

週	パーセント利用可能量	週	パーセント利用可能量	週	パーセント利用可能量
8	49, 49	20	42, 43, 42	32	40, 41
10	47, 47, 48, 48	22	40, 41, 41	34	40
12	43, 45, 46, 46	24	40, 40, 42	36	38, 41
14	43, 43, 45	26	40, 41, 41	38	40, 40
16	43, 43, 44	28	40, 41	40	39
18	45, 46	30	38, 40, 40	42	39

表 16: 塩素の利用可能性データ.

表 16 に与えられたデータは `chlorine.data` ファイルからデータフレームとして読み込むことが出来る．

提案された解析

`nlr()` 関数を利用して，非線形回帰モデルを当てはめよ．初期値を見つける簡単な方法は，θ の値を推測，たとえば $\theta = 1$，し $\exp(-\theta x)$ に対して Y をプロットすることである．次にプロットが線形になるまで， θ を半分にするか倍にすることを繰り返す.（これは，組み込みの行編集エディタで簡単に行える.）ひとたび θ の初期値が得られたら，他のパラメータの初期値は線形回帰により得られる．

あやしいデータ点に対して，当てはめたモデルをチェックせよ．

C.15 飽和水蒸気圧のデータ

出典: Draper & Smith: *Applied Regression Analysis...* 中に掲載
分類: 非線形回帰.

解説

このデータ実験機器によって作られた飽和水蒸気の温度 (°C) と圧力 (Pascal) を与える．飽和水蒸気中の水圧 (Press) Y と温度 (Temp) x の関係は，つぎのようにかける．

$$Y = \alpha \exp\left\{\frac{\beta x}{\gamma + x}\right\} + E$$

しかしながら，より現実的なモデルは実験誤差を加法的というより乗法的に含んでおり，その場合は以下のモデル

$$\log Y = \log \alpha + \left\{\frac{\beta x}{\gamma + x}\right\} + E$$

を用いた対数尺度での解析がより適しているかも知れない.

データ

Temp	Press	Temp	Press	Temp	Press
0	4.14	50	98.76	90	522.78
10	8.52	60	151.13	95	674.32
20	16.31	70	224.74	100	782.04
30	32.18	80	341.35	105	920.01
40	64.62	85	423.36		

表 17: 飽和水蒸気の温度と圧力

表 17 に与えられたデータは `steam.data` ファイルからデータフレームとして読み込むことが出来る.

提案された解析

両方のモデルをあてはめ比較せよ．ただ一つの非線形パラメータしか無いので，初期値は塩素データに対して使用されたのと同様の方法により得られるだろう.

C.16 Rumford 伯爵の摩擦熱データ

出典: Bates & Watts: *Nonlinear Regression Analysis...*
分類: 非線形回帰

解説

摩擦により生じた熱量に関するデータが Rumford 卿により 1798 年に得られている．ドリルが固定された円筒にはめられ，ネジにより底面に押しつけられている．ドリルは 30 分間一群の馬により回転させられ，その後 Rumford 卿は「温度計を 45 分ほどそこに置きっぱなしにし，それが示す温度を短い時間間隔で観察し記録した」.

ニュートンの冷却法則は次の形の非線形回帰モデルを示唆する

$$Y = \beta_0 + \beta_1 \exp(-\theta x)$$

ここで Y は温度で，x は分で計った時間である.

Data

Time (min.)	Temp (^{0}F)	Time (min.)	Temp (^{0}F)
4.0	126	24.0	115
5.0	125	28.0	114
7.0	123	31.0	113
12.0	120	34.0	112
14.0	119	37.5	111
16.0	118	41.0	110
20.0	116		

表 18: Rumford の摩擦冷却データ

表 18 に与えられたデータは，`rumford.data` ファイルからデータフレームとして読み込むことが出来る.

提案された解析

このデータは主に歴史的興味からのものである．先の塩素データと同様に扱え.

C.17　クラゲデータ

出典: *Interactive Statistics*, Ed. Don McNeil.
カテゴリー: 二変量，二標本データ.

, Ed. Don McNeil.

解説

それぞれ Danger 島と Salamander 湾産のクラゲからなる二組の標本の，長さ (`Length`) と幅 (`Width`) が測定された．

データ

Danger 島				Salamander 湾			
Width	Length	Width	Length	Width	Length	Width	Length
6.0	9.0	11.0	13.0	12.0	14.0	16.0	20.0
6.5	8.0	11.0	14.0	13.0	17.0	16.0	20.0
6.5	9.0	11.0	14.0	14.0	16.5	16.0	21.0
7.0	9.0	12.0	13.0	14.0	19.0	16.5	19.0
7.0	10.0	13.0	14.0	15.0	16.0	17.0	20.0
7.0	11.0	14.0	16.0	15.0	17.0	18.0	19.0
8.0	9.5	15.0	16.0	15.0	18.0	18.0	19.0
8.0	10.0	15.0	16.0	15.0	18.0	18.0	20.0
8.0	10.0	15.0	19.0	15.0	19.0	19.0	20.0
8.0	11.0	16.0	16.0	15.0	21.0	19.0	22.0
9.0	11.0			16.0	18.0	20.0	22.0
10.0	13.0			16.0	19.0	21.0	21.0

表 19: クラゲデータ – Danger 島と Salamander 湾

表 19 に与えられたデータは， `jellyfish.data` ファイルからのデータフレームとして読み込むことが出来る．

提案される解析

二つの標本をプロットし，それらの凸包をマークしなさい．Hotelling の T^2 により差の検定を行いなさい．(解析を行う簡単な方法は，*Location* を表すダミー変量を *Length* と *Width* に回帰し，両方の回帰係数の有意性を同時に検定することである．)

C.18 古代の壷のデータ

出典: Tubb, A. *et al.* Archæometry, **22**, 153–171, (1980)
カテゴリー: 多変量解析

22, 153–171, (1980)

解説

このデータは Wales, Gwent と New Forest の，イギリスにローマ軍が駐留していた時代の窯跡からの出土陶器の 26 の標本に対する化学的な分析から得られた．変量は様々な金属の組合せを示しており，その金属の酸化物のパーセンテージで表現されている．

金属はアルミ・鉄・マグネシウム・カルシウム・ナトリウムで，窯の場所 (Site) は，

L: Llanederyn,　C: Caldicot　I: Island Thorns　A: Ashley Rails

データ

Site	Al	Fe	Mg	Ca	Na	Site	Al	Fe	Mg	Ca	Na
L	14.4	7.00	4.30	0.15	0.51	C	11.8	5.44	3.94	0.30	0.04
L	13.8	7.08	3.43	0.12	0.17	C	11.6	5.39	3.77	0.29	0.06
L	14.6	7.09	3.88	0.13	0.20	I	18.3	1.28	0.67	0.03	0.03
L	11.5	6.37	5.64	0.16	0.14	I	15.8	2.39	0.63	0.01	0.04
L	13.8	7.06	5.34	0.20	0.20	I	18.0	1.50	0.67	0.01	0.06
L	10.9	6.26	3.47	0.17	0.22	I	18.0	1.88	0.68	0.01	0.04
L	10.1	4.26	4.26	0.20	0.18	I	20.8	1.51	0.72	0.07	0.10
L	11.6	5.78	5.91	0.18	0.16	A	17.7	1.12	0.56	0.06	0.06
L	11.1	5.49	4.52	0.29	0.30	A	18.3	1.14	0.67	0.06	0.05
L	13.4	6.92	7.23	0.28	0.20	A	16.7	0.92	0.53	0.01	0.05
L	12.4	6.13	5.69	0.22	0.54	A	14.8	2.74	0.67	0.03	0.05
L	13.1	6.64	5.51	0.31	0.24	A	19.1	1.64	0.60	0.10	0.03
L	12.7	6.69	4.45	0.20	0.22						
L	12.5	6.44	3.94	0.22	0.23						

表 20: The pottery composition data

表 20 に与えられたデータは `pottery.data` ファイルからデータフレームとして読み込むことが出来る．

提案される解析

簡単な判別分析により数値的，グラフ的に調査せよ．座標軸として最初の二つの判別関数を用い，4 組の標本を表示せよ．要約せよ．

C.19　ボージョレ産ワインの質のデータ

出典:　Quoted in Weekes: *A Genstat Primer*

カテゴリー: 多変量解析

解説

新醸のボージョレ産ワインのいくつかの識別された標本に対する品質測定．M. G. Jackson, *et al* の表 1 からの抜粋：赤ワインの品質：新醸のボージョレ産ワインの色・香り・風味間の相関と色素やその他のパラメータ，*Journal of Science of Food and Agriculture*, **29**, 715–727, (1978).

データ

ラベル	OQ	AC	pH	TSO	ラベル	OQ	AC	pH	TSO
A	13.54	1.51	3.36	13.8	I	12.25	1.32	3.38	1.4
B	12.58	1.35	3.15	5.2	J	14.04	1.52	3.61	4.5
C	11.83	1.09	3.30	10.6	K	12.67	1.62	3.38	0.4
D	12.83	1.15	3.41	2.2	L	13.54	1.57	3.55	7.9
E	12.83	1.32	3.44	2.3	M	13.75	1.63	3.34	6.3
F	12.12	1.23	3.31	10.5	N	9.63	0.78	3.19	40.4
G	11.29	1.14	3.49	2.5	O	12.42	1.14	3.31	3.1
H	12.79	1.22	3.56	16.7					

表 21: 新醸のボージョレ産ワイン標本の品質の測定

表 21 に与えられたデータは，`beaujolais.data` ファイルからのデータフレームとして読み込むことが出来る．

提案された解析

グラフィカルにデータを表示する方法を探せ．共分散行列を用いた主成分分析を考察し，ひどい外れ値を見つけよ．

C.20 de Piles の画家のデータ

出典: Weekes: A Genstat Primer.
カテゴリー: 多変量解析: 判別分析.

解説

データは 54 人の古典派画家に対する 0–20 の整数尺度による主観的な評価を表している．画家は 4 つの特性，構成・描画法・色彩・表現で評価された．データは 18 世紀の絵画評論家 de Piles によるものである．

画家が属した流派は以下の文字コードで表されている．

A	Renaissance	E	Lombard
B	Mannerist	F	Sixteenth Century
C	Seicento	G	Seventeenth Century
D	Venetian	H	French

データ

表 22 に与えられたデータは `painters.data` ファイルからのデータフレームとして読み込むことが出来る．

提案された解析

分散の多変量解析を利用して流派間の違いを調べよ．尤度比検定を利用せよ．また正準 $F-$ 統計量と判別関数を見つけよ．

最初の二つの判別関数軸に画家をプロットし，流派のシンボルをプロット文字として使え．流派ごとの凸包をマークせよ．`identify()` を利用し，対話的にプロットにおいて極端なところに位置する画家や，だれがその流派の平均から著しく逸脱しているかを見つけよ．

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
Guilio Romano	15	16	4	14	A
Michelangelo	8	17	4	8	A
Perino del Vaga	15	16	7	6	A
Perugino	4	12	10	4	A
Raphael	17	18	12	18	A
F. Zucarro	10	13	8	8	B
Fr. Salviata	13	15	8	8	B
Parmigiano	10	15	6	6	B
Primaticcio	15	14	7	10	B
T. Zucarro	13	14	10	9	B
Volterra	12	15	5	8	B
Barocci	14	15	6	10	C
Cortona	16	14	12	6	C
Josepin	10	10	6	2	C
L. Jordaens	13	12	9	6	C
Testa	11	15	0	6	C
Vanius	15	15	12	13	C
Bassano	6	8	17	0	D
Bellini	4	6	14	0	D
Giorgione	8	9	18	4	D
Murillo	6	8	15	4	D
Palma Giovane	12	9	14	6	D
Palma Vecchio	5	6	16	0	D
Pordenone	8	14	17	5	D
Tintoretto	15	14	16	4	D
Titian	12	15	18	6	D
Veronese	15	10	16	3	D
Albani	14	14	10	6	E
Caravaggio	6	6	16	0	E
Corregio	13	13	15	12	E
Domenichino	15	17	9	17	E
Guercino	18	10	10	4	E
Lanfranco	14	13	10	5	E
The Carraci	15	17	13	13	E
Durer	8	10	10	8	F
Holbein	9	10	16	13	F
Pourbus	4	15	6	6	F
Van Leyden	8	6	6	4	F
Diepenbeck	11	10	14	6	G
J. Jordaens	10	8	16	6	G
Otho Venius	13	14	10	10	G
Rembrandt	15	6	17	12	G
Rubens	18	13	17	17	G
Teniers	15	12	13	6	G
Van Dyck	15	10	17	13	G
Bourdon	10	8	8	4	H
Le Brun	16	16	8	16	H
Le Suer	15	15	4	15	H
Poussin	15	17	6	15	H

表 22: de Piles の主観的評価データ