R 基本統計関数マニュアル

間瀬 茂

目次

第 1 章

序

R はフリーの統計解析システム・環境である．統計解析用に開発された S 言語・システムのフリーな移植であり，S 言語に基づく商用システムである S-PLUS に本質的に影響を受けている．1991 年にニュージーランドのオークランド大学統計学科の二人の若い講師により雛型が開発され，1995 年にオープンソースのソフトとして公開されるや否や，全世界の統計家・ユーザの無償の熱狂的支援を獲得し，現在も急速な改良が進行中である．現在の R は，S-PLUS 等の S 言語の商用版と比較しても遜色のない水準に達している．実際，既に R は統計的手法の共通インフラの地位を占めているといっても過言ではない．また R は Linux, Microsoft Windows, Mac OS X を始めとするほとんど全ての OS の下で稼働する．

1.1 このマニュアルに付いて

このマニュアルは統計環境 R の基本パッケージ[*1] 中の統計関連関数の全体をカテゴリー別に紹介すること，およびそのヘルプドキュメント[*2] の「ほぼ忠実な」和訳を提供することを主な目的にしている．又，各関数に付属する参考例を実際に実行した結果，その理解を助けるための簡易な注釈，そして参考画像をまとめている．

R は完全な計算機言語でもあるが，その本来の目的は統計解析である．実際 R 本体だけでも，通常の統計解析には十分過ぎる程の膨大な統計関連機能を実装している．このマニュアルでは，統計手法の各分野毎に章を設け，更に節毎に細分し，最終的に各関数毎に副節を設けることで，目次だけからでも R の統計関連関数の全貌を把握できるようにまとめてある．

R の各関数には，一見過剰とも思える程多数のオプション引数が用意されている．これらの引き数には普通合理的な既定値が設定されているため，ユーザはごく僅かな引数を意識するだけで良い．しかしながら，こうしたオプション引数が用意されている理由は，それを必要とするユーザがいればこそである．R を前提とした解説本は多く典型的な使用例を中心にまとめてあるが，入門的レベルを了え実際のデータの解析を始めるや否や，こうした引数の意味の理解が不可欠になることも多い．

R の各関数には詳細なオンライン・ドキュメントと参考実行コードが用意されており，

[*1] つまり，R 本体を構成するパッケージ群であり，R をインストールすれば必ず含まれる機能からなる．

[*2] 基本的に R バージョン 2.6 および 2.7 に基づくが，それ以前のバージョンの R による箇所もあり，適宜最新のバージョンを参照して欲しい．

使用についての最良の手引になる．しかしながら，それらの意味を把握するのは，語学的な障壁を別にしても，統計学そのものや計算機言語としての R の知識を相当必要とすることがしばしばで，普通のユーザには困難が多いと思われる．そもそもどのような関数が R に存在するかについても，相当経験を積まない限り把握が不可能なことも多い．このことが，このマニュアルで採用している網羅的かつ直訳に近い方式の理由である．但し，実行例の注釈を始めとして，必要に応じて著者独自の解説も加えているが，一々区別していないので注意して欲しい．

統計システムとしての R の魅力のもう一つが，実際的なデータセットを豊富に備えていることである．これらのデータセットは各種関数の実行例で頻繁に用いられており，例のための例に留まらない迫真性を与えているが，一方で適用手法の本質が分かりにくくなっていることも事実である．こうした解析の結果を解釈するには，データセットの意味と，そのデータ構造についての最小限の知識が欠かせない．各データセットにもそうした解説がオンライン・ドキュメントとして与えられているが，これも各応用分野毎の特殊な用語が理解を困難にしている．このマニュアルでは，特に一章を設けて R 本体の組み込みデータセットの一覧とその簡易解説[*3] を提供している．

残念ながら，R を現代統計学の共通基盤としている膨大なユーザ提供のパッケージ群[*4] については，推奨パッケージと呼ばれる基本的なものも含めごく一部しか触れることができなかった．また，R の統計機能と不可分なグラフィックス機能については，ほとんど触れていない．

著者は先に計算機言語としての R の機能を，このマニュアルと同じ精神でまとめたマニュアル本[*5] を刊行し，幸い多くの R ユーザから好意的な評価を得ている．このマニュアルも同様に，日本の統計ユーザが R を一層深く使いこなすための座右の書となることを期待している．

1.2 その他の注意

以下に，このマニュアルを理解するために必要になる R の言語機能の基本概念・キーワードを参考のためにまとめておく．より詳しくは「R プログラミングマニュアル」を参照して欲しい．

1.2.1 名前付き引数

R の関数には名前付き引数 (named variable) と呼ばれる特別な引数が可能である．名前付き引数は「`option = value`」という形で与えられ，`value` 部分には名前付き引数が省略された際の既定の値・動作を指定しておく．

実際，この名前付き引数を省略すると，対応する既定省略値が使われる．普通名前 `option` には意味がはっきり分かるように，長い名前を付けることが多い．これを一々入力するのは，手間がかかりすぎるので，他の名前付き引数名と一意的に区別できる限り先頭の数文字を与えれば良いようになっている．

[*3] 但し，完全な理解は困難なことも多く，「機械翻訳」レベルに留まることもあることを予めお詫びしておく．

[*4] 2008 年 7 月現在で，CRAN に登録されているパッケージの総数は 1,500 を越えており，これら全てを解説することは，現代統計学そのものを解説するに等しい．

[*5] 「R プログラミングマニュアル」，数理工学社， 2007 年．

例えば，時系列解析の章で紹介される次の自己共分散関数 acf() には時系列オブジェクトを表す名前の無い引数[*6] x と，5 つの名前付き引数 lag.max, type, plot, na.action, demean がある．ここで，引数 plot の既定省略値は論理値 TRUE である．もしこれを論理値 FALSE としたければ明示的に plot = FALSE とする．また，p で始まる名前付き引数はただ一つしか無いので，p = FALSE，または pl = FALSE としても良い．また論理値 TRUE, FLASE は混乱が無い限り，それぞれ T, F と省略することもできる．名前付き引数 type には 3 つの選択肢 "correlation", "covariance", "partial" が可能であるが，その先頭の値 "correlation" が省略時既定値である．また，こうした文字列は一意的に定まる限り，先頭の数文字を type="co"のように与えれば良い．結局指定 type = "partial" は t="p" だけでも良いことになる．

更に入力を簡便にする機構として，例えば acf(x, 10, "covariance", FALSE) とすれば最初の 4 つの引数の値を順に指定したものとされ，... を除く残りの引数は，既定値が使われることになる．

普通引数リストの最後に置かれる特殊引数「...」は，オプションの任意個数の任意名前付き引数を表す．必ずしも実際に与える必要は無い．これは，例えば関数中で呼び出される他の関数の名前付き引数に，特別な値を指定するのに使われる．関数 acf() の例では，既定動作 plot = TRUE で使われ，計算結果のプロット用の関数 plot() (実際は plot.acf() 関数) に引き渡される名前付き引数を任意に指定できる．注意として引数 ... の後に置かれた名前付き引数は上に述べた先頭数文字による省略ができない．

R のオブジェクトは，ほとんど全て詳細なオンラインヘルプドキュメントを持ち，ヘルプ関数 help() 関数で直ちに参照できる．例えば help(foo)，もしくは ?foo とする．単項，二項演算子等は，例えば help("+"), ?"+" のように二重引用符で囲む必要がある．ヘルプドキュメントにはオブジェクトの書式，引数，返り値，説明，注意，参考文献，関連関数等の詳細な説明が含まれている．また，ヘルプドキュメントの末尾には，説明用の参考コードが付いており，実際の使い方を調べるのに便利である．実例実行関数 example() 関数を用い example(foo) とすると，この例示用コードが実際に実行される．しばしば例示用コードは複数のグラフィックス出力を伴うが，瞬時に切り替ってしまう．事前に par(ask=TRIE) を実行しておくと，各グラフィックス出力の前に確認を求められるので，ゆっくり眺めることができる．

注意：この本で紹介される参考用の R コードと，その出力中の，シャープ記号 # 以後行末までは，説明のために加えたコメントであり，実際に入力する必要は無く，また出力された内容の一部ではない．実際，R コードでは # 記号以下行末まではコメントとして無視される．

注意：この本全体を通じ，R のオブジェクト名，その成分，要素，属性，そしてコードはタイプライタ体で表示する．特に一目で区別できるように，関数名は help() の様に括弧を付けて表す．

注意: R の付値 (代入) 演算 y <- f(x) では付値結果は表示されないが，全体を丸括弧で囲み (y <- f(x)) とすると，付値とともに結果が表示され便利である．また R の命令は，セミコロンを間に挟めば，一行にいくつでも並べることができる．この本では，こうした機能を例示用コード紹介のスペースの節約のために随時使う．

[*6] 実際は x = baa という形の指定が可能であり，名前付き引数ともいえる．

注意：このマニュアルで紹介する関数以外にも R には内部的に使われるドキュメント化されていない (したがって `library(help=stats)` 等で表示されない) 関数が相当ある．例えば `?simpleLoess`, `help.search("simpleLoess")` では該当無しとされるが，`getAnywhere(simpleLoess)` はそのコードを表示する (ドキュメントは無い)．またドキュメント化されている関数でも，独立して使われる可能性が少ない一部の関数は省略されていることがある．

注意：このマニュアルで参照されている文献は原則著者名だけをあげる．詳しい情報は対応関数のヘルプ文章にあるので，必要に応じ参照して欲しい．

注意：このマニュアルの実行参考例は主として対応関数の参考実行例であるが，独自の例もある．記述の簡潔さのために一部編集・省略されているものもある．又乱数を使用する例では実行毎に結果が異なることもあるので注意して欲しい．又，例中のコメントもヘルプ文章中のものと，独自なものが混在しているが特に区別されていない．

1.2.2 パッケージ

R には本体と一緒に配布される「標準パッケージ」群とともに，「アドオンパッケージ (add-on package)」と呼ばれる全世界のユーザによる特定手法用のフリーのパッケージが数多く存在し，R 本体に簡易に組み込み，機能を拡張することができる．

R の標準パッケージと呼ばれるパッケージは以下の 12 種類である．特に最初の 7 種類は，R システムの中核を形作り，R を起動しさえすれば `library()` 関数で特に読み込むこと無く，直ちに利用できる．更に，CRAN で「推奨 (recommended)」とされるアドオンパッケージが 15 種類[*7] ある．

	基本パッケージ		推奨パッケージ
base	R 言語本体	survival	生存解析
datasets	データセット	class	分類用関数
graphics	グラフィックス関数	cluster	クラスタリング
grDevices	グラフィックスデバイス	foreign	他システムデータ読み込み
methods	クラス，メソッド	KernSmooth	カーネル法による平滑化
stats	基本統計関数	lattice	ラティスグラフィックス
utils	ユーティリティ関数	splines	スプライン回帰
grid	グリッドグラフィックス	mgcv	GAM と GAMM モデル
stats4	S4 方式の統計関数	nlme	(非) 線形混合モデル
tools	パッケージ管理	nnet	ニューラルネット
		rpart	再帰的分割と回帰木
		spatial	空間解析
		tcltk	Tcl/Tk インタフェイス
		MASS	文献 Venables & Ripley 中のオブジェクト
		boot	文献 Davison et al. 中の関数

[*7] 推奨パッケージ (バンドル) `VR` は実際は 4 種類のパッケージ `MASS`, `class`, `nnet`, `spatial` を含む．

1.2.3 オブジェクトの属性とクラス

R のオブジェクトには属性 (attribute) と呼ばれる付加的な情報を加えることができる．属性は実体としては単なる文字列 (もしくは文字列ベクトル) であり，任意にいくつでも付け加えることができる．R 言語が既定で持つ特殊な属性が幾つかあり，オブジェクトの素性を表し，他の関数がそのオブジェクトを処理する方法に関する情報を与える．例えば線形回帰関数 `lm()` の返り値には，自動的にクラス属性 (class attribute) `"lm"` が付け加えられ，それが線形回帰分析の結果であるという情報を与える (当然，オブジェクトはこのクラス属性を持つオブジェクトに期待される内容を全て備えている)．R はオブジェクトに種々の属性を付加することにより，オブジェクトを「構造化」する．例えば，R の行列・配列はそのものとしては単なるベクトルであるが，次元属性 (dimension attribute) `"dim"` を与えることにより，それが行列・配列であることを他の関数に教える．属性は隠された情報であり，`attr()` 関数や `str()` 関数を使わない限り，ユーザーの目からは隠されている．

1.2.4 クラス，メソッドと総称的関数

R 言語で頻繁に使われる `plot()` 等の関数は総称的関数 (generic function) と呼ばれる．これらの関数は引数である R オブジェクト `x` に応じて，実際に使われる関数が決まる．例えば `x` が線形回帰関数 `lm()` の返り値であれば，`x` はクラス属性`"lm"` (線形回帰オブジェクト) を持ち，`plot()` はこれを確認すると，線形回帰オブジェクトのプロット用に設計されたプロット関数 `plot.lm()` を起動する．関数 `plot.lm()` はプロット関数 `plot()` の一つのメソッド (method) と呼ぶ．`plot()` 関数はその他にも様々な属性タイプに応じたメソッドを持つ．また，ユーザが独自のクラスに対する独自のメソッドを定義することも可能である．この機構は，多くの同種の関数を同じ関数で代表させることで，ユーザがいちいちそれを意識せずに済むという点で，コードの大幅な簡略化を可能にする．結果として，R は自然にオブジェクト指向言語になっている．

1.3 R キーワード集

以下ではこのマニュアルを読むに当たって必要となる，幾つかの概念と用語をまとめておく．より詳しくは拙著「R プログラミングマニュアル，数理工学社 (2007 年)」 を参照されたい．

因子 (factor) 例えば，データフレームの文字列からなる変数は内部的にはその場限りの整数コードとして表現される．こうした表現は必要メモリー量を減らし，操作を簡易化するが，本来のデータ値とは異なるため場合によると混乱を引き起こすこともある．

S3, S4 それぞれ S 言語仕様の第 3,4 版を意味する．特にオブジェクトのクラスの定義が大きく異なり，S4 方式は本格的なオブジェクト指向言語を目指している．現在の R のかなりのオブジェクトは S3 方式で実装されているが，標準パッケージ `stats4` 等，今後 S4 方式で実装されたパッケージが増えていく．

オブジェクト (object) 一連の機能・情報を持つ構造．R では関数，データ等は全てオブ

ジェクトである．

返り値 (return value) R の関数は原則全て返り値を持つ．描画関数等には描画という副作用のみで明確な返り値が無いものもある．返り値にはコンソールに出力される可視な返り値と，コンソールに出力されない「不可視 (invisible) 返り値[*8] 」がある．不可視返り値はある変数に付値すれば内容が確認できる．大量で可読性が低い返り値はしばしば不可視返り値とされる．

環境・フレーム (environment, frame) オブジェクトの名前と値が登録されている R オブジェクト (フレームとは S 言語での環境の呼称だったらしい)．R の全オブジェクトはいずれかの環境に登録されている．環境が異なれば，名前が同じでも別の内容を持つことができる．環境は複数存在できる．ある環境が内部に別の環境を入れ子で持つとき，親・子環境と呼ぶ．全ての関数は実行時環境である一時的環境中に，引数や変数名とその値を保管しており，関数終了時にその親環境 (その関数を呼び出した環境) に返り値を返した後，自動的に消滅する．R 起動時の環境である大局的環境 (global environment) `.GlovalEnv` が存在する．コンソールへの対話的操作は通常この環境で行われる．環境自体も R のオブジェクトであり，独自に生成・抹消，変数の登録・抹消等の操作できる

観測値・データベース (observation, database) 解析対象となるデータの全体，もしくは一部．明確に定義された概念ではなく，むしろ慣用的呼称．普通データフレームである．

クラス属性 (class object) R の関数，データはクラスと呼ばれる属性情報を持つことができる．例えば，線形モデル当てはめ関数はその結果にクラス属性 `"lm"` を付加する．このクラスの属性は一定の書式に従う内容を持つことが約束されており，関連関数はその引数オブジェクトのクラスを確認し，適切な処理を行う

クラスの継承 (inheritance) あるクラス属性 `A` を持つオブジェクトが，更にその内容を特定化する他のクラス属性を持つとき，クラス `A` を継承するという

ケース (case) 一つの対象に関する観測値群．特にデータフレームの一つの行を指す．

工場出荷時既定値 (factory-fresh default) R のインストール時に設定される関数引数の省略時既定値．明示的に指定しない場合や，`options` に該当既定動作が指定されていないとこれが使われる．

作業スペース (working space) 大局的環境の別称．R での作業の基本となる環境である．

スコープ規則 (scoping rule) ある名前のオブジェクトをどの環境で探索し，参照するかを決める規則．この規則により，例えばある関数内でその親環境中の変数や関数を特に意識することなく使用することが可能になる．R コード中に現れたオブジェクトは，まずその付随環境中で探され，見付からなければ順にその親環境を順に遡って探索され，最後に大局的環境中に見付からなければエラーとされる．特定の環境中のオブジェクトを指定して参照することもできる．

ストリング (string) R のヘルプドキュメントではしばしばストリングと文字列 (character) が登場する．後者は定数としての文字列を意味する．このマニュアルでは区別無く文字列と訳している．

総称 (的) 関数 (generic function) 同種であるが実体は異なる関数を一つの名前の関数で

[*8] `return(invisible(x))` 等とすれば不可視返り値になる．

アクセスするための関数．引数オブジェクトのクラス属性に応じて実際に使われる (メソッド) 関数が異なってくる．広範囲のオブジェクト用のメソッド関数を持つ有用な総称的関数に `summary()` (オブジェクトの内容の簡潔な要約)，`str()` (オブジェクトの全構造の簡易表示)，`print()` (より整形されたコンソールへの出力)，`plot()` (オブジェクトのプロット) がある．，

属性 (attribute)　R のオブジェクトは属性情報を持つことができる．これは文字通りにはオブジェクトに付け加えられた (一つもしくは複数の) 文字列であり，オブジェクトが何であるかを示す．R にはシステムで予約された多くの属性があり，各種関数は引数の属性に応じて処理方法を変える，またその返り値に幾つかの属性を自動的に加える．オブジェクトの属性は任意に変えることができる (ベクトルを行列に変えるなど)．

遅延評価 (lazy loading)　R の表現式はそれが呼び出された時点ではまだ未評価であり，実際にそれが必要になった際に始めて評価される．関数引き数の既定値 (動作) も関数呼び出し時の値と，それが実際に使われる時の値が異なることがあり得る．

データフレーム (data frame)　R の多くの関数が引数または返り値として想定するデータオブジェクトの型．行列風の外観を持つが，実際はリストである．各列はケース (個々のケースに伴う変数の集まり) を，各行は変数を表す．各行は数値，文字列等異なったタイプであっても良い．行や列には普通名前ラベルが付き，それを用いて操作できる．

名前 (name)　オブジェクトの名前．これは文字列ではなく，実際二重引用符で囲む必要は無い．名前の実態であるオブジェクトは関連環境中で探される．データフレームの行・列に与えられた文字ラベル (名前属性) も名前と呼ばれる．R 端末にオブジェクト名を打ち込むと，その内容が表示される．

(表現) 式 (expression)　未評価の R のコード．例えば関数本体．これを `eval()` 関数等で評価すると式の値が得られる．

付値 (assignment)　オブジェクトをある名前 (変数) の値とする操作[*9] ．左付値演算子 `<-`, `=`，右付値演算子 `->`，そして付値関数 `assign()` がある．特殊な付値として永続付値 (permanent assignment)，`<<-`, `->>`，がある．現在の環境中ではなく，その親環境 (通例大局的環境 `.GlobalEnv`) 中のオブジェクトに対する付値操作となる．

変数 (variable)　ある項目に対するデータ値の集まり，もしくはそれに対する名前．データフレームの各列は一つの変数を表す

メソッド (method)　一つの総称的関数が総称する，特定のクラス属性を持つオブジェクト用の関数．

モード (mode)　オブジェクトのモード．`mode()` 関数で確認できる．例えば `function`, `numeric`, `list`, `complex`, `name` 等．その他に `storage.mode()` で確認できる「保管モード (strage mode)」がある．例えば，`integer`, `double`, `symbol`, `function`, `language` 等．更に保管モードにしばしば一致する型 (type) があり，`typeof()` 関数で確認できる．

モデル (公) 式 (model formula)　複数の変数間の関係をシンボリックに表現する式．回帰

[*9] 「代入」ではなく特に「付値」という訳を与えるのは `substitute` という異なった操作と区別するためである．

分析やグラフィックス等で使われる

モデルフレーム (model frame) モデル式，およびモデル式の評価に必要な変数値と追加引数を含んだデータフレーム．

呼出し式 (call) ある関数・モデル式の呼出し時に使われた表現式．

ラッパ関数 (wrapper function) ある関数を特定の状況・対象に簡単に適用するために設けられた関数．例えば `aov()` 関数は実験計画法データに `lm()` 関数を適用するためのラッパ関数であり，内部的に適切な引数の組合せで `lm()` を呼び出す．

第 2 章

基本記述統計関数

R は当然平均，分散等の基本的記述統計関数を持つ．これらは他の多くの統計関数の内部で用いられる．

2.1 平均，共分散，相関係数

2.1.1 平均 mean()

mean() は数値データフレーム，数値ベクトル，日付の平均を計算する．データフレームに対しては変数 (列) 毎に平均を取った名前付きベクトルを返す．trim 引数が 0 で無ければ，データの前後からそれぞれ指定した割合を取り除いた刈詰め平均 (trimmed mean) を計算する．NA 値を含めば，na.rm=TRUE でない限り結果も NA となる．

書式：

```
mean(x, ...)
mean(x, trim = 0, na.rm = FALSE, ...) # 既定の S3 メソッド
```

引数：

x	R オブジェクト．現在数値データフレーム，数値ベクトル，日付に対するメソッドがある．複素ベクトルは trim=0 の時だけ許される
trim	平均の計算の前に x の前後から取り除かれる割合 (0 から 0.5)．これ以外の値は 0 もしくは 0.5 の近い方とされる
na.rm	論理値．計算の前に NA 値を取り除くか？
...	他のメソッドに渡される追加引数

```
> x <- rnorm(100)
> c(mean(x), mean(x,trim=0.1), mean(x,trim=0.3))  # 平均と刈詰め平均
[1] -0.079828550 -0.035981575 -0.003364033
> x <- c(1:20, NA)                                # NA 値を含むデータ
> mean(x)                                         # 結果は NA
[1] NA
> mean(x, na.rm=TRUE)                             # 事前に NA 値を除いて計算
[1] 10.5
> mean(data.frame(a=1:10,b=runif(10)))            # データフレームの場合 (変数毎の平均)
        a         b
5.5000000 0.5164158
> mean(matrix(1:12, 3,4))                         # 行列はベクトル化される
[1] 6.5
> mean(as.data.frame(matrix(1:12, 3,4)))          # データフレーム化すれば OK
V1 V2 V3 V4
 2  5  8 11
> ( week <- seq(Sys.Date(), len=7, by="1 week") )
[1] "2008-07-25" "2008-08-01" "2008-08-08" "2008-08-15" "2008-08-22"
```

```
[6] "2008-08-29" "2008-09-05"
> mean(week)                                          # 日付オブジェクトの平均例
[1] "2008-08-15"
```

```
# 数値行列の行毎の平均を求める関数 rowMeans() がある．列毎なら colMeans() 関数を使う
> ( x <- matrix(1:16, 4,4) )
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> c(mean(x[1,]),mean(x[2,]),mean(x[3,]),mean(x[4,]))
[1]  7  8  9 10
> rowMeans(x)
[1]  7  8  9 10
```

2.1.2　重み付き平均 weighted.mean()

weighted.mean() は数値ベクトルの重み付き平均を計算する．もし重みベクトル w が無ければ等しい重みとされ，mean() と同じになる．重みベクトルは非負である必要は無い．最終的に総和が 1 になるように正規化されるため，総和が 0 になってはならない．

書式： weighted.mean(x, w, na.rm = FALSE)

引数：

x	数値ベクトル
w	x と同じ長さの重みベクトル
na.rm	論理値．計算の前に NA 値を取り除くか？

```
> x <- runif(5)
> w <- c(0.5,2,5,2,0.5)/10                            # 重みベクトル
> mean(x)                                             # 平均
[1] 0.5229157 0.4974361

> weighted.mean(x,w)                                  # 重み付き平均
[1] 0.4974361
> weighted.mean(x, 10*w)                              # 重みは事前に総和 1 に正規化される
[1] 0.4974361
> weighted.mean(x, c(1,2,5,-2,-1))                    # 重みは非負である必要は無い
[1] 0.4065080
> weighted.mean(x, c(1,2,0,-2,-1))                    # 重みの総和が 0 であってはならない
[1] Inf
```

2.1.3　分散，共分散，相関行列 cor(), cov(), var(), cov2cor()

var(), cov() そして cor() はベクトル x の分散，ベクトル x と y 間の共分散，相関を計算する．もし x と y が行列なら x と y の列間の共分散，相関からなる共分散行列，相関行列を計算する．cov2cor() は共分散行列を効率的に対応する相関行列に変換する．

書式：

```
var(x, y=NULL, na.rm=FALSE, use)
cov(x,y=NULL,use="all.obs",method=c("pearson","kendall","spearman"))
cor(x,y=NULL,use="all.obs",method=c("pearson","kendall","spearman"))
cov2cor(V)
```

引数：

x	数値ベクトル，行列，もしくはデータフレーム
y	NULL(既定値) もしくは x と比較可能な次元を持つベクトル, 行列 そしてデータフレーム．既定値は y=x であるがより効率的である
na.rm	論理値．欠損値を取り除くべきか?
use	欠損値があるとき共分散を計算する方法を与える文字列 (省略形可)．"all.obs" (既定), "complete.obs", "pairwise.complete.obs" のどれか
method	どの相関係数 (または共分散) を計算するか指示する文字列．"pearson" (既定), "kendall" または "spearman" で，省略形が可能
V	対称数値行列．普通共分散行列のように正定値符号

cov() と cor() に対しては，x もしくは x と y の双方に行列かデータフレームを与える必要がある．

var() は単に cov() へのインタフェイスであり，na.rm は use が与えられないときにその既定動作を与えるために使われる．もし na.rm=TRUE なら，分散の計算には全ての完全な観測値 (行) が使われる (use="complete")．さもなければ (use="all")，var() は欠損値があればエラーを与える．

もし use="all.obs" なら欠損値はエラーを与え，use="complete.obs" なら欠損値を含むケース (行) は予め取り除かれる．また use="pairwise.complete.obs" なら，対応列対から欠損値を含む行に相当する要素を取り除いて計算する (従って各要素毎に使われる変数の長さが異なる可能性がある)．これにより結果は必ずしも非負定値符号行列にならない．

計算にあたり データ数 − 1 で割ることにより，(共) 分散はデータが独立同分布の時不偏推定量になる．データ数 1 なら結果は NA であり，0 ならエラーとなる．

cor() に対して method="kendall" は Kendall の τ 統計量，method="spearman" ならば Spearman の ρ 統計量を与える．これらはランク統計量に基づく関係性の指標であり，より頑健であるため，データが二変量正規分布からのものでないときに推奨されてきた．

```
> x <- 1:10; y <- runif(10)
> var(x)                                          # 不偏分散
[1] 9.166667
> sum((x-mean(x))^2)/(length(x)-1))               # 不偏分散の定義．length(x)-1 で割る
[1] 9.166667
> var(x)*(length(x)-1)/length(x)                  # データ数で割る (非不偏) 分散
[1] 8.25

> cov(x,y)                                        # 共分散
[1] 0.3097297
> sum((x-mean(x))*(y-mean(y)))/(length(x)-1)      # 共分散の定義．length(x)-1 で割る
[1] 0.3097297

> cor(x,y)                                        # x,y の相関係数
[1] 0.3139105
# cor(x,y) の定義．比を取るので length(x)-1 で割っても length(x) で割っても同じになる
> cov(x,y)/(sqrt(var(x))*sqrt(var(y)))
[1] 0.3139105
```

```
> x <- matrix(runif(9),3,3)                       # 変数が行数の同じ行列の場合
> y <- matrix(runif(6),3,2)
> cov(x)
```

```
           [,1]       [,2]       [,3]
[1,] 0.008069159 0.01088460 0.01698830
[2,] 0.010884595 0.11052953 0.09253636
[3,] 0.016988300 0.09253636 0.08633651
> cov(x[,1],x[,1])                                  # x の列毎の共分散を計算する
[1] 0.008069159
> all.equal(cov(x),var(x))                          # var() は cov() のインタフェイス
[1] TRUE

> cov(x,y)                                          # x,y の共分散行列
            [,1]        [,2]
[1,]  0.001271333 -0.01446632
[2,] -0.074080210  0.08572905
[3,] -0.052378817  0.04598891
> cov(x[,1],y[,1])
[1] 0.001271333

> cor(x)                                            # x,y の相関行列
          [,1]      [,2]      [,3]
[1,] 1.0000000 0.3644677 0.6436333
[2,] 0.3644677 1.0000000 0.9472748
[3,] 0.6436333 0.9472748 1.0000000

> all.equal(cov2cor(cov(x)),cor(x))                 # 実数精度の範囲内で一致
[1] TRUE
```

```
> dfx <- as.data.frame(x)                           # 行数が同じデータフレーム．結果は行列
> dfy <- as.data.frame(y)
> cor(dfx)                                          # 行・列ラベルを引き継ぐ
          V1        V2        V3
V1 1.0000000 0.3644677 0.6436333
V2 0.3644677 1.0000000 0.9472748
V3 0.6436333 0.9472748 1.0000000

> cov(dfx)
            V1         V2         V3
V1 0.008069159 0.01088460 0.01698830
V2 0.010884595 0.11052953 0.09253636
V3 0.016988300 0.09253636 0.08633651

> cov(dfx,dfy)                          # 二つのデータフレームの列（変数）毎の相関からなる行列
            V1          V2
V1  0.001271333 -0.01446632
V2 -0.074080210  0.08572905
V3 -0.052378817  0.04598891
```

```
# Kendall の τ 統計量と Spearman の ρ 統計量
> x <- runif(10); y <- rnorm(10)
> rx <- rank(x,na.last="keep")                      # ランク統計量に変換
> ry <- rank(y,na.last="keep")
> c(cor(x,y),cor(x,y,method="pearson"))             # cor() の既定値は Pearson
[1] 0.0868436 0.0868436

> cor(x,y,method="kendall")
[1] 0.02222222

> c(cor(rx,ry), cor(x,y,method="spearman"))         # 相関係数とほとんど同じ値
[1] 0.05454545 0.05454545
```

```
> Cl <- cor(longley)                                # 組み込みデータ longley を使用
> symnum(Cl)                                        # 相関行列の文字グラフィックス表示
             GNP. GNP U A P Y E                     # 高度な相関
GNP.deflator 1
GNP          B    1
Unemployed   ,    ,   1
Armed.Forces .    .     1
Population   B    B   , . 1
Year         B    B   , . B 1
Employed     B    B   . . B B 1
attr(,"legend")
[1] 0 ‘ ’ 0.3 ‘.’ 0.6 ‘,’ 0.8 ‘+’ 0.9 ‘*’ 0.95 ‘B’ 1
```

```
> (x <- rbind(c(1,4,3),c(NA,2,8), c(-1,7,8)))     # 欠損値の扱い
     [,1] [,2] [,3]
[1,]    1    4    3
[2,]   NA    2    8
[3,]   -1    7    8
> cov(x)                                          # 欠損値があるとエラー
以下にエラー cov(x) :  cov/cor 関数に欠損した観測値があります

> (z <- var(x,na.rm=TRUE))
     [,1] [,2] [,3]
[1,]    2 -3.0 -5.0
[2,]   -3  4.5  7.5
[3,]   -5  7.5 12.5
> cov(x[c(1,3),1],x[c(1,3),1:3])                  # z[1,] の意味
     [,1] [,2] [,3]
[1,]    2   -3   -5

> cov(x,use="complete.obs")                       # var(x,na.rm=TRUE) と同じ
     [,1] [,2] [,3]
[1,]    2 -3.0 -5.0
[2,]   -3  4.5  7.5
[3,]   -5  7.5 12.5

> (w <- cov(x,use="pairwise.complete.obs"))
     [,1]       [,2]       [,3]
[1,]    2 -3.0000000 -5.0000000
[2,]   -3  6.3333333  0.8333333
[3,]   -5  0.8333333  8.3333333
> cov(x[,2:3])                                    # w[2:3,2:3] と一致
          [,1]      [,2]
[1,] 6.3333333 0.8333333
[2,] 0.8333333 8.3333333
> eigen(w)$values                                 # 負の固有値．非負定値符号行列ではない
[1] 11.985257  6.105029 -1.423619
```

```
# 数値行列の行毎の分散を計算．列毎なら colSums(),colMeans() を使う
> x <- matrix(1:16, 4,4)
> c(var(x[1,]),var(x[2,]),var(x[3,]),var(x[4,]))
[1] 26.66667 26.66667 26.66667 26.66667
> rowSums((x^2-rowMeans(x)^2))/(ncol(x)-1)
[1] 26.66667 26.66667 26.66667 26.66667
```

注意： Kendall の τ 統計量は x と y の順序一致対の数 P

$$P = \#\{(i,j) : \ i < j, \ \mathrm{sign}(x_i - x_j) = \mathrm{sign}(y_i - y_j)\}$$

を用い $\tau = 2P/(n(n-1)/2) - 1$ と定義され，$-1 \le \tau \le 1$ となる．Spearman の ρ 統計量 (ランク相関係数) は `rank(x, na.last="keep")` と `rank(y, na.last="keep")` の (Pearson) 相関係数に他ならない．

関連: 重みつき共分散の計算には `cov.wt()`．標準偏差 (ベクトル) は `sd()`．

2.1.4 標準偏差 sd()

`sd(x)` は標準偏差であり，不偏分散の平方根 `sqrt(var(x))` である．

書式： `sd(x, na.rm = FALSE)`

引数：

`x`	数値ベクトル，行列，もしくはデータフレーム
`na.rm`	論理値．欠損値を取り除くか？

```
> x <- runif(10)
> sd(x)
[1] 0.3457176
> all.equal(sd(x), sqrt(var(x)))
[1] TRUE
```

2.1.5　重み付き共分散 cov.wt()

cov.wt() は重み付き共分散行列と平均値からなるリストを返す．オプションで重み付き相関行列も返す．既定の method="unbiased" では共分散行列は 1 から重みの 2 乗和を引いたもので割られる．もし重みが既定値 (等分) ならこれは cov() と同じ結果を与える．method="ML" では割らない．重みベクトルは非負である必要は無い．最終的に総和が 1 になるように正規化されるため，総和が 0 になってはならない．

書式： cov.wt(x, wt = rep(1/nrow(x), nrow(x)), cor = FALSE,
center = TRUE, method = c("unbiased", "ML"))

引数：

x　行列もしくはデータフレーム．行は観測値であり，列は変数を表す

wt　各観測値に与えられる非負値からなる零ベクトルで無い重みのベクトル．長さは x の行数に等しくなければならない

cor　論理値．重み付き相関行列も返すか？

center　論理値か共分散を計算するとき使われるべき中心値のベクトル．もし TRUE なら各変数の重み付き平均が使われる．もし FLASE なら 0 が使われる．もし数値ベクトルなら長さは x の列数に等しくなければならない

method　結果のスケーリング法を与える文字列で "unbiased" か "ML"

返り値：　次の名前付き成分を含むリスト：

cov　重み付き共分散行列

center　データの中心 (重み付き平均)

n.obs　観測数 (x の行数)

wt　使われた重みベクトル．引数に与えられたときだけ返される

cor　相関行列．cor=TRUE の時だけ返される

```
> (xy <- cbind(x = 1:10, y = c(1:3, 8:5, 8:10)))
       x  y
 [1,]  1  1
 [2,]  2  2
 [3,]  3  3
 [4,]  4  8
 [5,]  5  7
 [6,]  6  6
 [7,]  7  5
 [8,]  8  8
 [9,]  9  9
[10,] 10 10
> w1 <- c(0,0,0,1,1,1,1,1,0,0)
> ( a <- cov.wt(xy, wt = w1) )                    # 既定の method="unbiased"
$cov
     x    y
x  2.5 -0.5
y -0.5  1.7
```

```
$center
  x   y
6.0 6.8
$n.obs
[1] 10
$wt                                              # 重みは総和 1 に正規化される
 [1] 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0

> colMeans(xy*(a$wt))                            # a$center は重み付き平均
   x    y
0.60 0.68
> (b <- cov.wt(xy, wt = w1, method = "ML", cor = TRUE))
$cov
     x     y
x  2.0 -0.40
y -0.4  1.36
$center
  x   y
6.0 6.8
$n.obs
[1] 10
$wt
 [1] 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0
$cor                                             # cor=TRUE で計算される重み付き相関行列
           x          y
x  1.0000000 -0.2425356
y -0.2425356  1.0000000
> all.equal(cov2cor(b$cov), b$cor)
[1] TRUE
> (Wt <- 1- sum(a$wt^2))
[1] 0.8
> all.equal(a$cov, b$cov/Wt)                     # method="unbiased","ML"の違い
[1] TRUE
```

2.2　度数分布

2.2.1　分割表 table()

table() は因子水準の組合せ毎にオブジェクトを集計して表にする．

書式： table(..., exclude=c(NA,NaN), dnn=list.names(...),
deparse.level=1)

引数：

... 因子と解釈できるオブジェクト (文字列を含む)，またはそうした変数を含むデータフレーム

exclude 除外されるべき水準

dnn 結果に与えられる次元名

deparse.level (= 0, 1, 2)．次元名とその名前のつけ方を決める

responseName 表項目に与えられる名前．普通カウント数

返り値： 属性 "table" を持つ整数値配列である分割表 (contingency table)．table() には要約関数 summary() のメソッドがあり，$\chi 2$ 乗検定も行う

もし dnn が与えられないと dimname names が内部的に計算される．... 中の引数が名前を持てば，それが使われる．

関連： ftable(), xtabs().

```
> table(rpois(100, 5))                              # 平均 5 の 100 個のポアッソン乱数の集計表
 0  1  2  3  4  5  6  7  8  9 10
 1  1 10 22 19 12 11 13  6  4  1
# 組み込みデータフレーム warpbreakas を使った例．変数 wool,tension の組合せで二元分類
> with(warpbreaks, table(wool, tension))
    tension
wool L M H
   A 9 9 9
   B 9 9 9

# 組み込みデータフレームを使った例．変数 Temp をクォンタイル値で分類し，Month とともに二元分類
> with(airquality, table(cut(Temp, quantile(Temp)),  Month))
         Month
           5  6  7  8  9
  (56,72] 24  3  0  1 10                            # 行は変数 Temp を大小順に 4 等分する区間
  (72,79]  5 15  2  9 10
  (79,85]  1  7 19  7  5
  (85,97]  0  5 10 14  5
```

```
> x <- letters[1:3]
> table(x, sample(x), deparse.level=0)     # deparse.level による次元名と次元名ラベルの違い
    a b c
  a 1 0 0
  b 0 1 0
  c 0 0 1

> table(x, sample(x))                               # 既定の deparse.level=1
x   a b c                                           # 行ラベル x
  a 1 0 0
  b 0 1 0
  c 0 0 1
```

```
> table(a, sample(a), deparse.level=2)
   sample(x)                                    # 行ラベル x, 列ラベル sample(x)
x   a b c
  a 1 0 0
  b 0 1 0
  c 0 0 1
```

```
> b <- factor(rep(c("A","B","C"),10))
> d <- factor(rep(c("A","B","C"),10), levels=c("A","B","C","D","E"))
> table(d, exclude = "B")                      # 水準 B を除外
d
 A  C  D  E
10 10  0  0
> print(table(b, d), zero.print=".")            # print メソッドの例
   d                                          # カウント 0 をドットで表示
b    A  B  C  D  E
  A 10  .  .  .  .
  B  . 10  .  .  .
  C  .  . 10  .  .
```

```
> table(UCBAdmissions)                        # 組み込みデータセット UCBAdmissions を使用
 table [1:2, 1:2, 1:6] 512 313 89 19 353 207 17 8 120 205 ...
 - attr(*, "dimnames")=List of 3
  ..$ Admit : chr [1:2] "Admitted" "Rejected"
  ..$ Gender: chr [1:2] "Male" "Female"
  ..$ Dept  : chr [1:6] "A" "B" "C" "D" ...
> is.array(UCBAdmissions)                      # 3 次元配列である
[1] TRUE
> is.table(UCBAdmissions)                      # 3 元分類表でもある
[1] TRUE
> class(UCBAdmissions)                         # クラス属性"table"
[1] "table"
> UCBAdmissions[,, 1]                          # 配列としての添字操作
          Gender
Admit      Male Female
  Admitted  512     89
  Rejected  313     19
> is.table(UCBAdmissions[,, 1])                # 結果は表ではなくなる
[1] FALSE

> summary(UCBAdmissions[,,1])                  # summary メソッド適用
      Male           Female
 Min.   :313.0   Min.   :19.0
 1st Qu.:362.8   1st Qu.:36.5
 Median :412.5   Median :54.0
 Mean   :412.5   Mean   :54.0
 3rd Qu.:462.2   3rd Qu.:71.5
 Max.   :512.0   Max.   :89.0

> summary(as.table(UCBAdmissions[,,1]))        # 分類表として summary メソッド適用
Number of cases in table: 933
Number of factors: 2
Test for independence of all factors:          # 性別と入試結果間の χ 2 乗独立性検定
      Chisq = 17.248, df = 1, p-value = 3.280e-05 # 到底独立とは見做せないという結果
```

2.2.2 幹葉表示 stem()

stem() はデータの幹葉表示 (stem-and-leaf plot) を行う．幹葉表示は文字グラフィックスで，データの表集計とそのヒストグラムを同時に表示する．

書式： stem(x, scale = 1, width = 80, atom = 1e-08)

引数：

x　　数値ベクトル
scale　プロットの長さを制御する．scale=2 とすれば既定のほぼ 2 倍になる
width　プロット幅の指定
atom　寛容度

```
> x <- rgamma(50, shape=3)
> options(digits=1)
> (xx <- sort(x))
 [1] 0.3 0.6 0.7 0.8 0.9 1.2 1.2 1.2 1.3 1.3 1.4 1.4 1.5 1.5 1.5 1.5 1.6 1.6 1.7
[20] 2.2 2.3 2.4 2.5 2.5 2.6 2.7 2.8 2.8 3.1 3.2 3.3 3.3 3.7 3.8 3.9 4.0 4.2 4.2
[39] 4.3 4.3 4.4 4.5 4.5 4.5 4.6 4.9 5.1 5.9 6.6 8.1
> stem(x)
  The decimal point is at the |                    # 縦線位置が小数点に相当

  0 | 36789                                        # データ 0.3,...,0.9 に対応
  1 | 22233445555667                               # データ 1.2 が 3 回，1.3 が 2 回登場等
  2 | 234556788
  3 | 1233789
  4 | 02233455569
  5 | 19                                           # データ 5.1,5.9 に対応
  6 | 6                                            # データ 6.6 に対応
  7 |
  8 | 1                                            # データ 8.1 に対応

> stem(x, scale=2)                                 # スケールを大きくすると幹が長くなる
  The decimal point is at the |

  0 | 3                                            # データ 0.3 に対応
  0 | 6789                                         # データ 0.6,0.7,0.8,0.9 に対応
  1 | 2223344
  1 | 5555667
  2 | 234
  2 | 556788
  3 | 1233
  3 | 789
  4 | 022334
  4 | 55569
  5 | 1
  5 | 9
  6 |
  6 | 6
  7 |
  7 |
  8 | 1
```

2.3　順序統計量に基づく要約

平均，分散等の記述統計量は外れ値の存在で大きく影響を受けるため，探索的なデータ解析では順序統計量に基づくより頑健な統計量が好ましいとされる．

2.3.1　中央値 median()

median() は標本中央値 (sample median) を計算する．既定のメソッドはともに総称的な sort() 関数と mean() 関数を用いるため，Date を始めとする中央値が意味を持つ様々なクラスに適用できる．

書式： median(x, na.rm=FALSE)

引数：
x　　メソッドが定義できるオブジェクトか数値ベクトル

na.rm 論理値．NA 値を予め取り除くか？ na.rm=FALSE でない限り，NA 値を含むデータは NA 値を返す

返り値： 長さ 1 のオブジェクトであり，多くの場合 x と同じ型

```
> x <- 1:10
> median(x)
[1] 5.5
> median(c(x,NA))                                  # NA 値が含まれると結果も NA
[1] NA
> median(c(x,NA), na.rm=TRUE)
[1] 5.5
> median(c(x,100, 1000))                           # 中央値は外れ値に対し頑健
[1] 6.5
> median(c(-Inf,x,100,1000))                       # -Inf,Inf があっても良い
[1] 6
```

2.3.2 MAD (median absolute deviation) mad()

mad() はデータの散布度の一種である MAD (median absolute deviation) を計算する．MAD は中央値からの絶対偏差の中央値 (オプションで上・下中央値 (low-median, high-median)) であり，既定では正規乱数に対し標準偏差の漸近推定量になるように係数補正される．

書式： mad(x, center=median(x), constant=1.4826, na.rm=FALSE, low=FALSE, high=FALSE)

引数：

x 数値ベクトル

center 中心として使われる値．既定では中央値

constant 補正係数

na.rm 論理値．欠損値を予め取り除くべきか？

low もし TRUE ならば low-median を計算する．つまり，偶数個のデータに対しては二つある中間値を平均せず，小さい方を取る

high もし TRUE ならば high-median を計算する．つまり，偶数個のデータに対しては二つある中間値を平均せず，大きい方を取る

実際に計算される値は constant*cMedian(abs(x-center)) であり，center の既定値は中央値である．ここで cMedian は中央値もしくは low-median, high-median を表す．constant の既定値は 1.4826 (ほぼ 1/qnorm(3/4)) で，正規乱数に対し mad(x) の理論平均が理論標準偏差に一致するように選ばれる．

```
> x <- rnorm(100, mean=1, sd=2)
> mad(x)                                           # 標準偏差 2 の (頑健) 推定量
[1] 2.07
> median(abs(x-median(x)))*1.4826                  # mad() の定義に従い計算
[1] 2.07
> sd(x)                                            # この場合標本標準偏差がより正確
[1] 2.03

> xx <- c(x, rnorm(10, mean=5, sd=10))             # 外れ値を混入する
> sd(xx)                                           # 標本標準偏差は大きく変化
```

```
[1] 3.9
> mad(xx)                                          # mad はこの場合も頑健
[1] 2.30
```

2.3.3 IQR (inter quatile range) IQR()

IQR() はデータの散布度の一種である IQR(四分位偏差, inter quatile range) を計算する．mad() より単純であるが頑健さは劣る．

書式： IQR(x, na.rm = FALSE)

引数：

x　　数値ベクトル

na.rm　論理値．欠損値を予め取り除くか？

IRQ は IQR(x)=quantile(x,3/4)-quantile(x,1/4) で計算される．正規分布 N(m,1) に従う独立同分布データ x に対し IQR(x) の期待値は 2*qnorm(3/4)=1.3490 であり，IQR(x)/1.349 が標準偏差の一致推定量になる．

```
> x <- rnorm(100, mean=1, sd=2)                    # 正規乱数
> IQR(x)
[1] 2.41
> quantile(x,3/4)-quantile(x,1/4)                  # IQR の定義
 75%
2.41
> IQR(x)/1.39                                      # 理論標準偏差 sd=2 の推定値
[1] 1.74
```

2.3.4 最大・最小値 max(), min(), pmax(), pmin(), pmax() pmax.int(), pmin.int()

max(), min() はデータの最大値と最小値を計算する．pmax(), pmin() はその並列版である．pmax.int(), pmin.int() はアトミックなデータに対する pmax(), pmin() の高速内部 (internal) 関数版である．

書式：

max(..., na.rm = FALSE), min(..., na.rm = FALSE)

pmax(..., na.rm = FALSE), pmin(..., na.rm = FALSE)

pmax.int(..., na.rm = FALSE), pmin.int(..., na.rm = FALSE)

引数：

...　数値もしくは文字列引数

na.rm　論理値．欠損値を予め除くか？ na.rm=TRUE としない限り欠損値を含むデータの返り値は NA となる

返り値： min(), max() に対しては長さ 1 のベクトル．pmin(), pmax() に対しては入力ベクトルの最大の長さと同じ長さのベクトル．データが全て整数値および論理値からなれば，整数，全てが実数ならば実数，それ以外は文字列である．実数

が混在して良い

空な集合の最大値と最小値はそれぞれ Inf および -Inf となり，推移規則

```
min(x1, min(x2)) == min(x1, x2)
max(x1, max(x2)) == max(x1, x2)
```

が成り立つ．長さ 0 の数値ベクトル x の最大値と最小値もそれぞれ Inf および -Inf となる．もし全ての並列要素が NA であれば，na.rm=TRUE であっても pmax(), pmin() は NA を返す．pmax.int(), pmin.int() は全ての引数がアトミックでクラスを持たない場合にだけ使える内部関数による高速版である．定義から NaN 値を含むデータの最大・最小値は NaN になる．但し NA 値が更に含まれれば NA が返される．max(NA, Inf) は NA が何であろうと Inf となるべきであるが，実際は NA を返す．空の文字列ベクトルの最大・最小値は NA となる．

```
> x <- 1:10; y <- 11:15; z  <- matrix(1:9, 3,3)
> c(min(x), max(x))
[1]  1 10
> c(min(z), max(z))                              # 行列はベクトルとみなされる
[1] 1 9

> pmax(x, x+0.5, x+0.7)                          # 並列版
 [1]  1.7  2.7  3.7  4.7  5.7  6.7  7.7  8.7  9.7 10.7
> pmin(x, x+0.5, x+0.7)
 [1]  1  2  3  4  5  6  7  8  9 10

# 長さが不揃いなときはリサイクル規則により最長の引数の長さに揃えられる
> (a <- pmax(x, y))
 [1] 11 12 13 14 15 11 12 13 14 15               # a[10]=max(x[10],y[5])
> (b <- pmin(x, y))
 [1]  1  2  3  4  5  6  7  8  9 10               # b[10]=min(x[10],y[5])
```

```
# 文字列の辞書式大小順序による最大・最小
> max(letters[1:10])                             # アルファベット順での最大
[1] "j"
> min(letters[1:10])
[1] "a"
> max(c(letters[1:10], LETTERS[1:10]))           # 大文字は小文字より「小さい」
[1] "j"
> min(c(letters[1:10], LETTERS[1:10]))
[1] "A"
> min("a", "abc", "abd")                         # 文字列は辞書式順序で比較される
[1] "a"
> max("a", "abc", "abd")
[1] "abd"
> max(c(1:3,"a","bc"))                           # 数値と文字列の混在する場合
[1] "bc"                                         # 文字（列）は数値より「大きい」
> min(c(1:3,"a","bc"))
[1] "1"
> pmax(1:3, c("a","bc"))
[1] "a"  "bc" "a"
> pmin(1:3, c("a","bc"))
[1] "1" "2" "3"
```

2.3.5 範囲 range()

range() は与えられた全ての引数の最大・最小値からなるベクトルを返す．

書式：

```
range(..., na.rm = FALSE)
```

`range(..., na.rm = FALSE, finite = FALSE)  # 既定の S3 メソッド`

引数：

`...`　任意個数の数値，文字列オブジェクト

`na.rm`　論理値．欠損値を予め取り除くか？

`finite`　論理値．有限でない値 `Inf, -Inf` を予め取り除くか？

返り値：　もし `na.rm=FALSE` なら，引数中に `NA, NaN` 値 があれば `NA` 値が返される．さもなければ `NA` 値は無視される

```
> range(1:10)
[1]  1 10
> diff(range(1:10))                                    # 標本範囲
[1] 9
> range(c(1:10, NA))                                   # 以下 NA,NaN を含む場合
[1] NA NA
> range(c(1:10, NaN))
[1] NaN NaN
> range(c(1:10, NA), na.rm=TRUE)
[1]  1 10
> range(c(1:10, NA, NaN), na.rm=TRUE)
[1]  1 10
> range(c(1:10, -Inf, Inf))                            # 以下 Inf,-Inf を含む場合
[1] -Inf  Inf
> range(c(1:10, -Inf, Inf), finite=TRUE)
[1]  1 10
```

2.3.6　5 数要約 fivenum()

`fivenum()` は入力データの Tukey の 5 数要約 (最小値，下側ヒンジ, 中央値，上側ヒンジ，最大値) を返す．

書式： `fivenum(x, na.rm = TRUE)`

引数：

`x`　数値．`NA,-Inf,Inf` を含んでも良い

`na.rm`　論理値．もし `TRUE` なら `NA` 値と `NaN` 値は予め取り除かれる

返り値：　要約情報を含む長さ 5 のベクトル．より詳しい情報は `boxplot.stats()` で得られる

```
> x <- rnorm(100)
> fivenum(x)
[1] -2.1141793288 -0.5312374346  0.0006125566  0.6055708976  3.2287810533
> fivenum(c(x, 0/0))
[1] -2.1141793288 -0.5312374346  0.0006125566  0.6055708976  3.2287810533
> fivenum(c(x, (-1):1/0))
[1]          -Inf -0.5324221854  0.0006125566  0.6320303839           Inf
```

2.3.7　クォンタイル quantile()

`quantile()` は与えられた確率に対応する標本分位数 (sample quantile) を計算する．最小値は確率 0，最大値は確率 1 に対応する．

書式：

```
quantile(x, ...)
quantile(x, probs = seq(0,1,0.25), na.rm = FALSE,
         names = TRUE, type = 7, ...)              # 既定の S3 メソッド
```

引数：

`x` 数値ベクトル．欠損値は無視される

`probs` 確率値のベクトルで，区間 `[0,1]` 中の値からなる

`na.rm` 論理値．もし `TRUE` なら全ての `NA`, `NaN` 値は予め `x` から取り除かれる

`names` 論理値．もし `TRUE` なら結果は名前属性を持つ．多くの確率値を与えた時は `FALSE` にすると早くなる

`type` 1 から 9 までの整数で，クォンタイルを計算するアルゴリズムを指定する

`...` 他のメソッドに渡される追加引数

返り値： 長さ `length(probs)`．もし `names=TRUE` なら名前属性を持つ．確率中の `NA`, `NaN` 値は結果に伝播する

$F(x)$ を標本分布関数とすると，確率 p に対し $F(x) = p$ となる値 x が標本クォンタイルであるが，標本経験分布関数が階段関数になるため，具体的な定義には曖昧さが残る．引数 `type` にはクォンタイルを計算する 9 通りの方法が指定できる．既定値は `type=7` で S システムのそれを踏襲している．`type=3` は SAS，`type=6` は Minitab, SPSS の方式である．詳しくは `help(quantile)` を参照されたい．

```
> x <- rnorm(1000)
> quantile(x, prob=(0:4)/4)
        0%         25%         50%         75%        100%
-3.65567706 -0.72480728 -0.01213069  0.68629984  3.26797410
> quantile(x, prob=(0:4)/4, names=FALSE)                    # 名前属性無し
[1] -3.65567706 -0.72480728 -0.01213069  0.68629984  3.26797410
```

```
# 計算方式による違い
> options(digits=3)
> (z <- quantile(x,prob=(1:5)/6,names=FALSE))          # 既定の type=7
[1] -0.9704 -0.4326 -0.0121  0.4355  0.9881
> z - quantile(x,prob=(1:5)/6,names=FALSE,type=1)
[1]  0.00228  0.00000  0.00227  0.00000 -0.00176
> z - quantile(x,prob=(1:5)/6,names=FALSE,type=2)
[1]  0.00228  0.00000  0.00000  0.00000 -0.00176
> z - quantile(x,prob=(1:5)/6,names=FALSE,type=3)      # SAS 方式との差
[1] 0.00228 0.00427 0.00227 0.00000 0.00176
> z - quantile(x,prob=(1:5)/6,names=FALSE,type=4)
[1] 0.003242 0.002846 0.002274 0.000934 0.000587
> z - quantile(x,prob=(1:5)/6,names=FALSE,type=5)
[1]  0.001521  0.000712  0.000000 -0.000332 -0.001174
> z - quantile(x,prob=(1:5)/6,names=FALSE,type=6)      # Minitab,SPSS 方式との差
[1]  0.002762  0.001423  0.000000 -0.000663 -0.002802
> z - quantile(x,prob=(1:5)/6,names=FALSE,type=8)
[1]  2.03e-03  9.49e-04  2.57e-16 -4.42e-04 -1.57e-03
> z - quantile(x,prob=(1:5)/6,names=FALSE,type=9)
[1]  0.001902  0.000889  0.000000 -0.000415 -0.001468
```

2.3.8 箱型図統計量 boxplot.stats()

`boxplot.stats()` は箱型図 (box plot) を書くために主に他の関数から呼ばれるが，単独でも使用できる．

書式： boxplot.stats(x, coef = 1.5, do.conf = TRUE, do.out = TRUE)

引数：

x　箱型図情報を計算すべき数値ベクトル．NA, NaN 値があれば取り除かれる

coef　箱型図のヒゲがどれくらい箱から延びるかを決める．もし正ならば，ヒゲは箱から，箱の長さの coef 倍を越えない最も離れたデータ値まで引かれる．0 にすればヒゲは最大・最小値まで引かれ，外れ値は無いことになる

do.conf　論理値．もし FALSE ならば返り値の conf 成分は空になる

do.out　論理値．もし FALSE ならば返り値の out 成分は空になる

返り値：　次の名前付き成分を含むリスト：

stats　長さ 5 のベクトルで，順に，下側ヒゲの端点，下側ヒンジ，中央値，上側ヒンジ，上側ヒゲの端点の位置

n　標本中の NA 値以外のデータ数

conf　do.conf=TRUE の時切り欠き (notch) の上下の端点位置

out　do.out=TRUE の時ヒゲの端点の外側にあるデータの値 (もしあれば)

二つのヒンジは，データ数が奇数なら第 1,3 四分位数 quantile(x,c(1,3)/4) に一致し，偶数なら近いが一致しない．切り欠けは範囲

[-1.58*IQR/sqrt(n), 1.58*IQR/sqrt(n)]

を表す．二つの箱型図の切り欠けが重ならなければ，対応データの中央値は異なる可能性が高いとされる．

```
> x <- rnorm(100)                                    # 正規乱数
> boxplot.stats(x)
$stats
[1] -2.600 -0.817 -0.106  0.764  2.550
$n
[1] 100
$conf
[1] -0.355  0.144
$out                                                 # 外れ値無し
numeric(0)

> x <- rcauchy(100)                                  # コーシー乱数
> boxplot.stats(x)
$stats
[1] -3.065 -1.202 -0.281  0.470  2.282
$n
[1] 100
$conf
[1] -0.5448 -0.0164
$out                                                 # 多くの外れ値
 [1] -10.74  10.82  22.80  -7.52  17.03   4.29 -75.60  13.53   6.90 -14.11
[11] -16.76  -4.61 -14.69  18.70  10.84  12.89  34.36  15.34  -5.07  14.52
```

2.4 基本記述統計グラフ

2.4.1 経験分布関数 ecdf()

ecdf() はデータの経験 (累積) 分布関数 (empirical cumulative distribution function) を計算する.

書式:

```
ecdf(x)
# クラス "ecdf" に対する S3 メソッド
plot(x, ..., ylab="Fn(x)", verticals=FALSE, col.01line="gray70")
print(x, digits= getOption("digits")-2, ...)
```

引数:

x	数値ベクトル. メソッドに対してはクラス ecdf を継承するオブジェクト
...	メソッド関数に引き渡される追加引数
ylab	plot 関数に対する y 軸ラベル
verticals	plot.stepfun() を見よ
col.01line	確率 0,1 に対応する水平線の色を指定する数値もしくは文字列
digits	print 関数で出力する際の有効桁数

返り値: ecdf() に対してはクラス "ecdf" の関数で, "stepfun" クラスを継承する

観測値 $x = (x_1, x_2, \ldots, x_n)$ に対して, 経験分布関数 F は

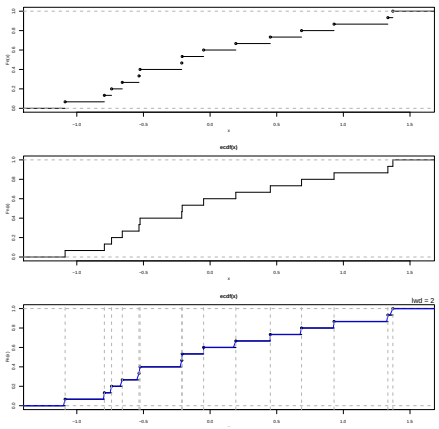
$$F(t) = \#\{i : \ x_i \le t\}/n, \quad -\infty < t < +\infty$$

と定義される. 欠損値は無視される. 関数 plot.ecdf() はクラス "ecdf" に対する plot() 関数メソッドで, stepfun() 関数を呼び出す.

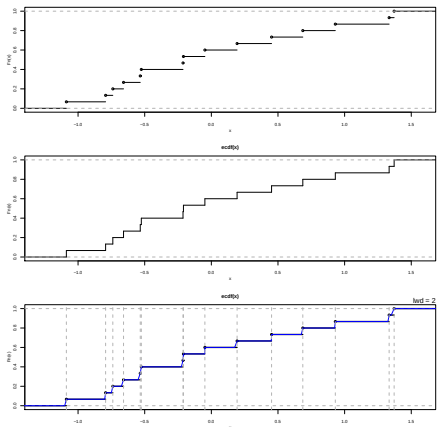
```
> x <- rnorm(15)
> F <- ecdf(x)
> F                                                 # F の素性は関数
Empirical CDF
Call: ecdf(x)
 x[1:15] = -1, -0.8, -0.7,  ...,  1,  1
> F(x)                                              # ジャンプ点での値
 [1] 0.6667 0.6000 0.4667 0.8667 0.2667 0.0667 0.9333 0.4000 0.7333 0.5333
[11] 1.0000 0.3333 0.8000 0.1333 0.2000
> knots(F)                                          # ジャンプ点位置 (データ x そのもの)
 [1] -1.0878 -0.7932 -0.7392 -0.6588 -0.5334 -0.5254 -0.2126 -0.2093 -0.0485
[10]  0.1932  0.4519  0.6863  0.9302  1.3340  1.3715
> utils::ls.str(environment(F))                     # 関数 F の素性をより詳しく見る
f :  num 0
method :  int 2
n :  int 15
x :  num [1:15] -1.088 -0.793 -0.739 -0.659 -0.533 ...
y :  num [1:15] 0.0667 0.1333 0.2000 0.2667 0.3333 ...
yleft :  num 0
yright :  num 1
```

```
# 上の経験分布関数を 3 通りプロット (図参照)
op <- par(mfrow=c(3,1),mgp=c(1.5, 0.8,0),mar= .1+c(3,3,2,1))
plot(F)                                             # 既定のプロット
plot(F, verticals= TRUE, do.points = FALSE)         # ジャンプ位置を縦の線分で結ぶ
plot(F, lwd = 2) ; mtext("lwd = 2", adj=1)          # より凝ったプロット
```

```
xx <- unique(sort(c(seq(-3, 2, length=201), knots(F))))
lines(xx, F(xx), col='blue')
abline(v=knots(F),lty=2,col='gray70')
par(op)
```



経験分布関数のプロット



経験分布関数のプロット

2.4.2　QQ プロット qqnorm(), qqline(), qqplot()

qqplot() は二つのデータの QQ(quantile-quantile) プロットを描く．qqnorm() はデータの正規 QQ プロットを描く．qqline() は正規 QQ プロットに，データの第 1，第 3 四分位数を通る直線を描き加える．

二つのデータ x, y の QQ プロットは，同じ確率 p に対するクォンタイル quantile(x, prob=p) と quantile(y, prob=p) を座標とする点を幾つかの p に対してプロットしたもので，二組のデータの理論分布の近さ (全体として直線 y=x に近いか) を視覚的にとらえる．一方，正規 QQ プロットはデータ x のクォンタイル quantile(x, prob=p) と正規分布の理論クォンタイル qnorm(p) を座標とする点を幾つかの p に対してプロットしたもので，x のデータの理論分布が正規分布に近い (全体として直線 qqline(x) に近いか) を視覚的にとらえる．

書式：

```
qqnorm(y, ...)
# 既定の S3 メソッド:
qqnorm(y, ylim, main = "Normal Q-Q Plot",
       xlab = "Theoretical Quantiles", ylab = "Sample Quantiles",
       plot.it = TRUE, datax = FALSE, ...)
qqline(y, datax = FALSE, ...)
qqplot(x, y, plot.it = TRUE, xlab = deparse(substitute(x)),
       ylab = deparse(substitute(y)), ...)
```

引数：

x　　qqplot() に対する最初の標本

y　　2 番目もしくは唯一のデータ

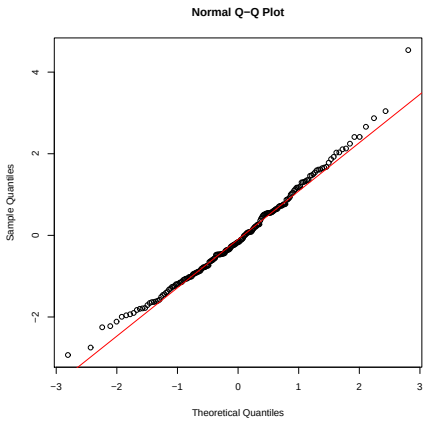
xlab, ylab, main　プロットラベル．もし datax=TRUE なら xlab, ylab はそれぞれ

y, x 軸を指す
plot.it 論理値．結果をプロットすべきか？
datax 論理値．データ値は x 軸上にあるべきか?
ylim, ... 描画パラメータ

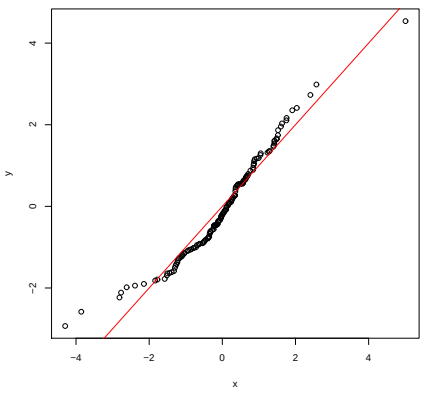
返り値： qqnorm() と qqplot() は次の 2 成分を持つリストを返す：
x qqplot() に対する最初の標本
y 2 番目もしくは唯一のデータ
xlab, ylab, main プロットラベル．もし datax=TRUE なら xlab, ylab はそれぞれ x, y 軸を指す
plot.it 論理値．結果をプロットすべきか？
datax 論理値．データ値は x 軸上にあるべきか?
ylim, ... 描画パラメータ

qqnorm() が内部で使う ppoints() 関数は，正規分布における期待順序統計量値の近似値を計算する．

```
> y <- rt(200, df = 5)                    # 自由度 5 の t 乱数 200 個
> qqnorm(y)                               # 正規 QQ プロット
> qqline(y, col = 2)                      # 直線性のチェック用直線を描き加える

> x <- rt(150, df = 6)                    # 自由度 6 の t 乱数 150 個
> qqplot(x, y)                            # x,y データの QQ プロット
> abline(0, 1, col = 2)                   # 直線 y=x を描き加える
```



t 乱数の正規 QQ プロット



二組の t 乱数の QQ プロット

2.4.3 ヒストグラム hist()

hist() はデータのヒストグラムを計算する．もし plot=TRUE なら plot.histogram() を用いて更に描画される．

書式：
hist(x, ...)
# 既定の S3 メソッド

```
hist(x, breaks = "Sturges", freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE, density = NULL, angle = 45,
     col = NULL, border = NULL, main = paste("Histogram of" , xname),
     xlim = range(breaks), ylim = NULL, xlab = xname, ylab,
     axes = TRUE, plot = TRUE, labels = FALSE, nclass = NULL, ...)
```

引数：

x データベクトル

breaks 以下の一つ，

- ヒストグラムのセル間の分割点を与えるベクトル
- セル数を与える単一の数
- セル数を計算するアルゴリズムを与える文字列 (以下を参照)
- セル数を計算する関数

freq 論理値．もし TRUE ならヒストグラムは結果の counts 成分である度数の表示．もし FALSE なら density 成分である確率密度がプロットされ，総面積は 1 になる． breaks が等間隔である (そして probability が指定されない) ときだけ既定値は TRUE

probability !freq のエイリアス (S との互換性のため)

include.lowest 論理値． TRUE なら breaks に等しい x[i] は左の棒 (もし right=FALSE なら右) に含められる． breaks がベクトルでなければ無視され警告がでる

right 論理値．もし TRUE ならヒストグラムのセルは右閉じ左空き区間

density 陰影斜線の密度 (インチあたりの線数)．既定値の NULL(もしくは負値) では斜線は引かれない

angle 陰影斜線の傾き．度単位の角度 (反時計周り)

col 棒を塗りつぶす色．既定の NULL では塗りつぶし無し

border 棒の周囲の色．既定では標準前景色と同じ

main, xlab, ylab これらの title 引数は有用な既定値を持つ

xlim, ylim x, y 値の範囲． x 軸範囲指定 xlim はヒストグラムの定義には使われず，plot=TRUE の際のプロットで使われることを注意

axes 論理値．もし既定の TRUE なら，プロットの際軸が描かれる

plot 論理値．もし既定の TRUE なら，ヒストグラムが描かれる．さもなければ，breaks と counts のリストが返される．後者の場合， plot=TRUE でだけ意味を持つ引数が指定されると警告がでる

labels 論理値もしくは文字． FALSE でなければ棒の上部にラベルが追加される．help(plot.histogram) を参照

nclass 整数値．S(-PLUS) との互換用．数値，文字引数に対する breaks と同値

... plot=TRUE の時 plot.histogram() に渡される追加引数

返り値： 返り値は次の成分を持つクラス "histogram" のリスト：

breaks n+1 個のセル境界．breaks がベクトルならそれと一致

counts n 個の整数．各セルに対する，その内側にある x データの数

density 推定密度の x[i] における値 f^(x[i])．もし all(diff(breaks)==1) な

ら相対頻度で，一般に sum[i; f^(x[i])(breaks[i+1]-breaks[i])]=1
intensities density と同じ．廃止予定
mids n 個のセルの中点
xname 実際の x 引数の名前の文字列
equidist 論理値． breaks の間隔が全て等しいか？

ヒストグラムの定義はいろいろあるが，R の既定である等間隔分点のヒストグラムでは分点で決まるセル中の度数に比例する高さで棒をプロットする．等間隔分点では棒の面積にも比例する．非等間隔の分点の場合の既定プロットでは，面積が各セルの相対度数に比例する棒を描く．

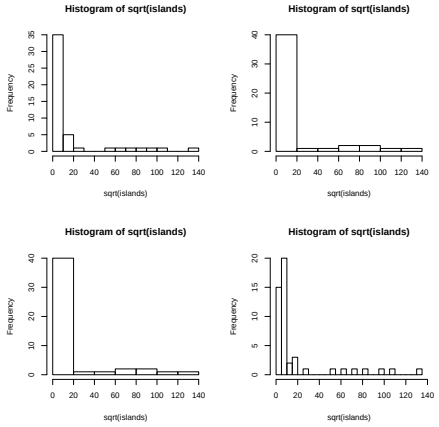
もし既定の right=TRUE ならヒストグラムのセルは左半開区間 (a,b] である．もし include.lowest=TRUE ならデータの最小値は最初のセルに入る．もし right=FALSE なら右半開区間 [a,b) であり，include.lowest=TRUE ならデータの最大値は最後のセルに入る．セルの端点上のデータ値に対してはセル幅の中央値の 1e-7 倍の寛容度が適用される．

breaks の既定手法は "Sturges" である (help(nclass.Sturges 参照)．他に用意されている手法は "Scott"と"FD"("Freedman-Diaconis") である，help(nclass.scott と help(nclass.FD) 参照．手法名には大・小文字は無視され，部分的マッチングが可能．分点を計算する x の関数を独自に指定することもできる．

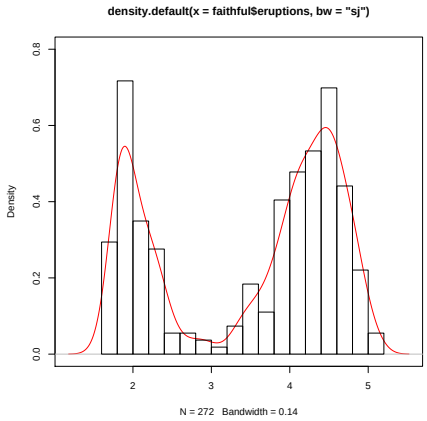
```
> str(hist(islands, col="gray", labels = TRUE, plot=FALSE))
List of 7                                          # ヒストグラム情報 (だけ)
 $ breaks     : num [1:10] 0 2000 4000 6000 8000 10000 12000 14000 16000
                           18000
 $ counts     : int [1:9] 41 2 1 1 1 1 0 0 1
 $ intensities: num [1:9] 4.27e-04 2.08e-05 1.04e-05 1.04e-05 1.04e-05 ...
 $ density    : num [1:9] 4.27e-04 2.08e-05 1.04e-05 1.04e-05 1.04e-05 ...
 $ mids       : num [1:9] 1000 3000 5000 7000 9000 11000 13000 15000 17000
 $ xname      : chr "islands"
 $ equidist   : logi TRUE
 - attr(*, "class")= chr "histogram"
Warning message:                                   # 描画しないと警告
In hist.default(islands, col = "gray", labels = TRUE, plot = FALSE) :
  arguments 'col', 'labels' are not made use of
```

```
> hist(sqrt(islands), breaks=12)                   # 分点計算法による違い．12 等間隔分点
> hist(sqrt(islands), breaks="Sturges")            # Sturges 法
> hist(sqrt(islands), breaks="Scott")              # Scott 法
> hist(sqrt(islands), breaks="FD")                 # FD 法
```

```
# Old Faithful geyser(間欠泉) の噴出時間間隔データ．カーネル法による推定密度関数プロットと比較
> d <- density(faithful$eruptions, bw = "sj")
> plot(d, ylim=c(0,0.8), col=2)
> hist(faithful$eruptions, breaks=20, freq=FALSE, add=TRUE)
```



分点計算法による違い



カーネル法による推定密度関数との比較

2.4.4　散布図 plot()

plot() は多くの R オブジェクト用のメソッドを持つ総称的なプロット関数であるが，特に散布図 (scatterplot) を描くのに使われる．指定可能な様々な描画用パラメータに付いては help(par) を参照．

書式： plot(x, y, ...)

引数：

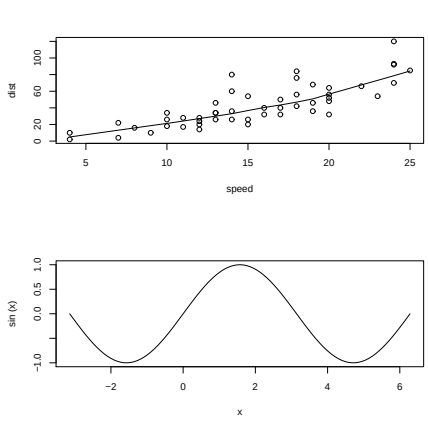
x	プロット中の点の座標．もしくは，単一のプロット構造，関数， plot 用メソッドを持つ任意の R オブジェクト
y	プロット中の点の y 座標．もし x が適切な構造を持てばオプション
...	描画パラメータ等メソッド関数に渡される引数 (群)．多くのメソッドは以下のパラメータを受け付ける:
type	プロットタイプ．以下が可能 (他は全て警告もしくはエラーになる)
	"p" 点，"punkte"と同値
	"l" 線分
	"b" 点と線双方，
	"c" "b" の線部分だけ’
	"o" 両者を重ね書き，
	"h" ヒストグラム風の垂直線
	"s" 階段関数風
	"S" 今一つの階段関数風プロット
	"n" 何もプロットしない
main	主タイトル文字列
sub	副タイトル文字列
xlab	x 軸タイトル文字列
ylab	y 軸タイトル文字列
asp	両軸のスケール比 (縦横比 y/x)

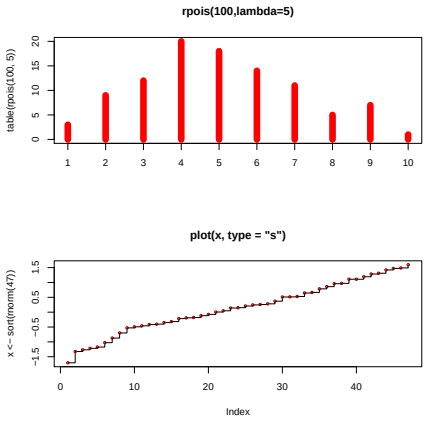
単純な散布図にはメソッド関数 plot.default() が使われる．その他にも function,

data.frame，density 等のオブジェクト用のメソッドがあり，各オブジェクトのヘルプ文章に解説がある．公式によるプロット関数 plot.formula() も参照．行列の列変数の対毎の散布図を同時にプロットする行列散布図 matplot() 関数，複数の変数の対毎の散布図を行列風にプロットする散布図行列 pairs() 関数も有用である．

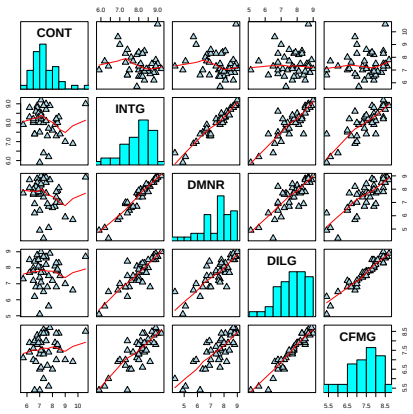
```
> plot(cars)                                          # 組み込みデータセット cars の散布図
> lines(lowess(cars))                                 # 平滑化曲線を上書き
> plot(sin, -pi, 2*pi)                                # 関数グラフ

# ポアッソン分布の密度関数のプロット
> plot(table(rpois(100,5)), type="h", col="red", lwd=10,
       main="rpois(100,lambda=5)")

# 経験分布関数（専用関数 ecdf() の利用が好ましい）
> plot(x <- sort(rnorm(47)), type="s", main="plot(x, type = \"s\")")
> points(x, cex=.5, col="dark red")
```

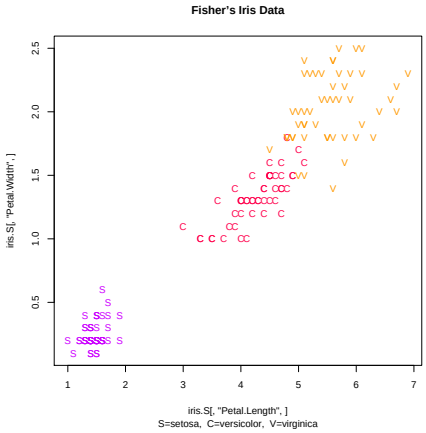
```
> pairs(USJudgeRatings[1:5], panel=panel.smooth,cex=1.5, pch=24,
        diag.panel=panel.hist, cex.labels=2, font.labels=2)              # 散布図行列
> matplot(iris.S[,"Petal.Length",], iris.S[,"Petal.Width",], pch="SCV",
          col=rainbow(3, start = .8, end = .1),
          sub=paste(c("S", "C", "V"), dimnames(iris.S)[[3]],
                    sep="=", collapse=",  "),
          main="Fisher's Iris Data")                                     # 行列散布図
```





散布図と関数グラフ　　離散分布の密度関数と経験分布関数のプロット





散布図行列の例　　行列散布図の例

2.4.5　箱型図 boxplot()

boxplot() はデータ (グループ) から箱型図 (箱ヒゲ図，boxplot, box and whisker plot) を画く．箱型図は現代統計学の最も基本的な記述統計用グラフの一つである．

書式:

```
boxplot(x, ...)
# クラス formula に対する S3 メソッド
boxplot(formula, data=NULL, ..., subset, na.action=NULL)
# 既定の S3 メソッド
boxplot(x, ..., range=1.5, width=NULL, varwidth=FALSE, notch=FALSE,
        outline=TRUE, names, plot=TRUE, border=par("fg"), col=NULL,
        log="", pars=list(boxwex=0.8, staplewex=0.5, outwex=0.5),
        horizontal=FALSE, add=FALSE, at=NULL)
```

引数：

formula　y ~ grp といった公式．ここで y はグルーピング変数 (普通因子) grp によってグループ化される数値データ

data　formula 中の変数含まれるデータフレームもしくはリスト

subset　プロットされる観測値の部分集合を指定するオプションベクトル

na.action　データに欠損値があるときの処理を指定する関数．既定では欠損値は無視される

x　箱型図を書くべきデータ．数値ベクトルかそうしたベクトルを含む単一のリスト．任意個の名前無しの追加引数は別のデータを指定し，それぞれ一つの箱型図に対応する． NA 値があっても良い

...　formula メソッドに対しては既定のメソッドに引き渡される名前付きの引数．既定メソッドでは名前ラベル無しの引数は追加データ (x がリストなら無視される) とされ，名前ラベル付きの引数は関数 bxp() に渡される引数と作画パラメータ (pars で指定されたパラメータを上書きする)

range　ヒゲが箱からどれだけ延びるかを決める．もし range が正なら，ヒゲは箱から IQR の range 倍を越えない最も離れたデータ位置まで引かれる．0 ならヒゲは最大・最小値まで延びる

width　箱の相対的な幅を決めるベクトル

varwidth　もし TRUE なら，箱はグループ中の観測値の数の平方根に比例する幅になる

notch　もし TRUE なら，箱の両側面に切り欠け (notch) が描かれる．もし二つの箱型図の切り欠けが重ならなければ，対応データの中央値が異なる可能性が高い

outline　もし FALSE なら外れ値は描かれない

names　各プロットの下に書かれるグループのラベル．文字列ベクトルでも表現式でも良い (plotmath() を参照)

boxwex　全ての箱に適用されるスケール因子．もしグループ数が少なければ，箱の幅を狭くした方が見栄えが良くなる

staplewex　ステープル線 (ヒゲの先端部) の幅拡大率，箱幅への比例率

outwex 外れ値線の幅拡大率，箱幅への比例率
plot もし TRUE (既定値) なら箱型図が描かれる．もし FALSE なら箱型図情報の要約が出力される
border 箱型図の外側線に対する色のベクトル．もし箱型図の数より少なければリサイクル使用される
col もし NULL でなければ，箱型図の本体を彩色する色情報を含むとされる．既定では背景色とされる
log x 座標， y 座標もしくは双方を対数スケールでプロットするかどうかを指示する文字
pars 追加の作図パラメータのリスト．例えば boxwex や outpch．plot=TRUE ならこれらは bxp() に渡される
horizontal 箱型図を水平に描くかどうかを指示する論理値． FALSE なら垂直
add 論理値． TRUE なら現在のプロットに箱型図が追加される
at 箱型図が描かれる位置を与える数値ベクトル (特に add=TRUE の時意味を持つ)．既定では n を箱の数として 1:n

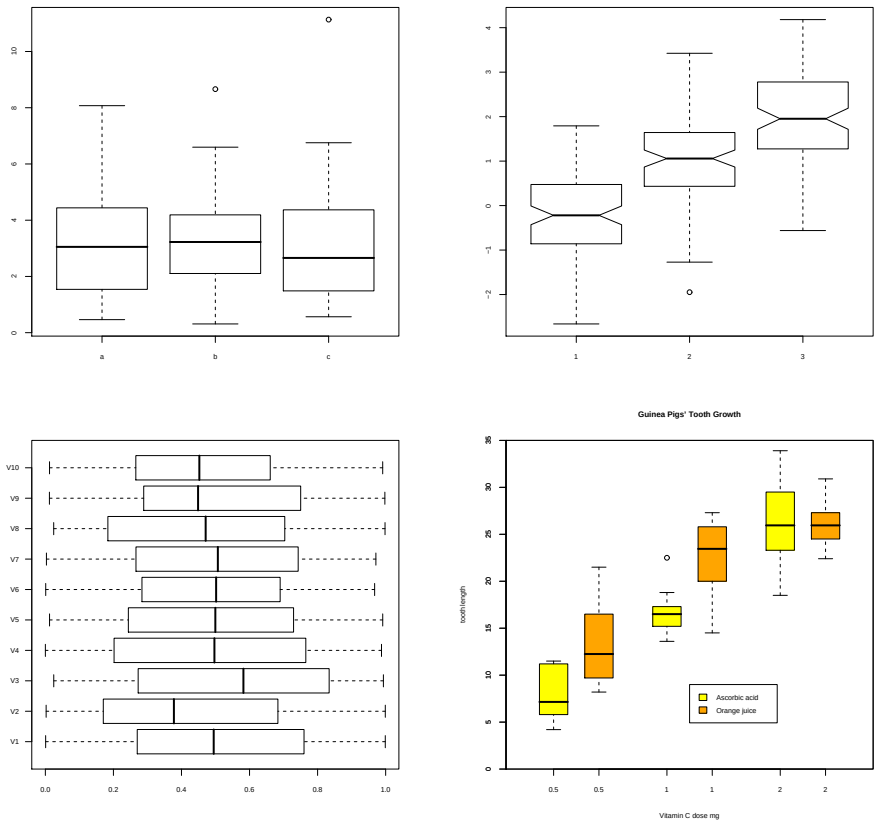
---

返り値： 以下の成分を持つリスト：
stats 行列で，各列は各箱の下側のヒゲの端点位置，下側ヒンジ，中央値，上側ヒンジ，上側ヒゲの端点位置を含む．もし全ての入力データが同じクラス属性を持てば，この成分も同じクラス属性を持つ
n 各グループ中の観測値数のベクトル
conf 行列で，各列は切り欠けの下側，上側端点位置を含む
out ヒゲの外側にある全てのデータ点の値
group out と同じ長さのベクトルで，各外れ値がどのグループに属するかを指示する
names グループの名前のベクトル

総称的な関数 boxplot() は現在既定メソッド関数 boxplot.default() と公式によるインタフェイス関数 boxplot.formula() を持つ．もし複数のグループが複数引数もしくは公式で与えられると，平行箱型図が引数もしくは因子の水準の順序で描かれる．欠損値は無視される．関数 bxp() は箱型図情報から箱型図を画く．

```
# 因子 G=c("a","b","c") で変数 A をグループ化し三つの箱型図を描く
x <- data.frame(A=rgamma(100, shape=3),
                G=sample(letters[1:3], 100, replace=TRUE))
boxplot(A ~ G, data=x)

# ベクトルを並べる．中央値の比較用に切り欠けを加える
x1 <- rnorm(100); x2 <- rnorm(100)+1;  x3 <- rnorm(100)+2
boxplot(x1,x2,x3, notch = TRUE)

# 数値行列の各列を箱型図に（先ずデータフレームに変換する）
x <- matrix(runif(1e3), nrow=100, ncol=10)
dfx <- as.data.frame(x)
boxplot(dfx)

# help(boxplot) 中の例（二つの boxplot 命令を重ねる）．組み込みデータフレーム ToothGrowth 使用
boxplot(len ~ dose, data = ToothGrowth,
        boxwex = 0.25, at = 1:3 - 0.2,              # 箱幅と位置を指定
        subset = supp == "VC", col = "yellow",      # 黄色で塗りつぶし
        main = "Guinea Pigs' Tooth Growth",         # 主タイトル
        xlab = "Vitamin C dose mg",                 # x 軸ラベル
```

```
        ylab = "tooth length",                          # y 軸ラベル
        xlim = c(0.5, 3.5), ylim = c(0, 35),            # x,y 軸範囲指定
        yaxs = "i")                                     # y 軸タイプ指定
boxplot(len ~ dose, data = ToothGrowth, add = TRUE, # 二つ目の箱型図を重ねる
        boxwex = 0.25, at = 1:3 + 0.2,                  # 少しずらして画く
        subset = supp == "OJ", col = "orange")          # オレンジ色で塗りつぶし
legend(2, 9, c("Ascorbic acid", "Orange juice"),
       fill = c("yellow", "orange"))                    # 凡例を加える
```



例示用コードの出力箱型図 (左から右，上から下の順)

2.4.6 一次元散布図 stripchart()

stripchart() は一次元散布図 (ストリップチャート，ドットプロット) を描く．これはデータ数が小さいとき箱型図の良い代りになる．

書式：

```
stripchart(x, ...)
# クラス formula に対する S3 メソッド
stripchart(x, data=NULL, dlab=NULL, ..., subset, na.action=NULL)
# 既定の S3 メソッド
stripchart(x, method = "overplot", jitter = 0.1, offset = 1/3,
           vertical = FALSE, group.names, add = FALSE,
           at = NULL, xlim = NULL, ylim = NULL,
           ylab=NULL, xlab=NULL, dlab="", glab="",
           log = "", pch = 0, col = par("fg"), cex = par("cex"),
           axes = TRUE, frame.plot = axes, ...)
```

引数：

x　データ．既定メソッドでは単一の数値ベクトルでも良いし数値ベクトルのリストでも良い (各成分毎にプロットされる)．公式メソッドでは y ~ g といったシンボリックな指定ができ，ベクトル y 中のデータを因子 g によりグループ化することを指示する．データ中に欠損値があっても良い

data　データフレームもしくはリストで，そこから x が取り出される

subset　プロットに使われるデータの部分集合を指定するオプションベクトル

na.action　欠損値がある場合の動作を指定する関数．既定では欠損値を除く

...　既定メソッドに渡される追加パラメータ．もしくは追加描画パラメータ

method　同一値データを分離するために使われる方法．既定の "overplot" は重ね書き，"jitter" は少しずらす，"stack" は積み重ねる．

jitter　method="jitter" の時，点を互いに少しずらす量を指定する

offset　method="stack" の時，点は行高さの offset 倍だけ離れて積み重ねられる

vertical　もし TRUE なら点は既定の水平でなく，垂直に描かれる

group.names　各プロットの下に書かれるグループラベル

add　もし TRUE なら現在のプロットに重ね描きされる

at　チャートが描かれるべき位置を指定する数値ベクトル．既定では n をデータ数として 1:n

ylab,xlab　ラベル，title を参照

dlab,glab　軸ラベルを指定するもう一つの方法，下を参照

xlim,ylim　プロットの範囲，help(plot.window) を参照

log　対数スケールを適用する軸，help(plot.default) 参照

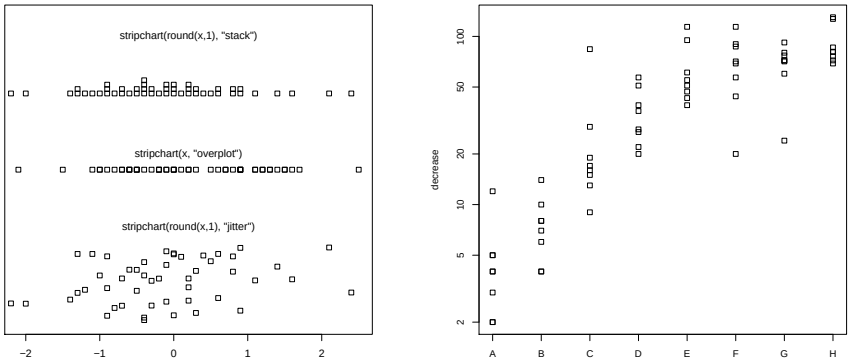
pch,col,cex　描画パラメータ，help(par) 参照

axes,frame.plot　軸の制御，help(plot.default) 参照

xlab, ylab が与えられていないとき，dlab と glab が代りに使える．dlab は連続な

データ軸 (vertical 指定が無ければ x 軸)，glab はグループ軸に適用される．

```
> x <- round(rnorm(50), 1)
> stripchart(x)                                                  # 既定
> text(mean(par("usr")[1:2]), 1.04, "stripchart(x, \"overplot\")")
> stripchart(xr, method = "stack", add = TRUE, at = 1.2)         # 同一値データを積み重ね
> text(m, 1.35, "stripchart(round(x,1), \"stack\")")

> stripchart(xr, method = "jitter", add = TRUE, at = 0.7)        # 同一値データを少しずらす
> text(m, 0.85, "stripchart(round(x,1), \"jitter\")")

> stripchart(decrease ~ treatment,              # 組み込みデータ OrchardSprays 使用，対数 y 軸
             main = "stripchart(OrchardSprays)",
             vertical = TRUE, log = "y", data = OrchardSprays)
```



一次元散布図 (ストリップチャート)

第 3 章
古典的検定

R は豊富な代表的古典的検定関数 (パラメトリック，ノンパラメトリック検定) を持つ．この章では R の基本パッケージ中にある検定関数を紹介する．これらは `help.search("ctest")` で (使用システムにインストールされているアドオンパッケージ中の検定関数と併せて) 一覧できる． R の検定関数は，オプション引数 `conf.int=TRUE` を指定すれば，関連パラメータの信頼区間を同時に計算できるようになっている．

ノンパラメトリク検定とは，通常母集団分布に対しては密度関数を持つことだけを仮定し，順序 (ランク) 統計量を基礎とする検定統計量を用いる検定のことである．それ以外が一括してパラメトリック検定と呼ばれ，仮説はモデルパラメータで表現される．この中には漸近理論による近似帰無分布を用い，あらかじめ母集団分布に付いて具体的な仮定を置かないものもある．

検定に於ては，帰無仮説 (null hypothesis) とともに，対立仮説 (alternative hypothesis) を考えるのが基本であるが，対立仮説が曖昧なものもある．帰無分布 (帰無仮説の下での検定統計量の分布) に付いては，正確な分布を用いるものと，近似分布を用いるものがある．両者がオプションで選べるものもあれば，近似分布しか使えないものもある．又モンテカルロ法を用いるものもある．統計量の帰無仮説の下での分布を漸近近似で求めるものが多数を占める．こうした場合一般論としてデータの数がある程度大きいことが前提条件になる．特にヘルプドキュメントに注意が無いことが多いが，実際は近似が不十分と思われる場合は警告が出る関数がある．

検定結果の p 値は，データから計算した検定統計量 T の値を t とすると帰無仮説の下での確率 $\boldsymbol{P}\{|T| > t\}$ (両側検定)，$\boldsymbol{P}\{T > t\}$ (上側片側検定)，$\boldsymbol{P}\{T < t\}$ (下側片側検定) である．従って，指定した有意水準を α とすると p 値 $\leq \alpha$ であれば，有意水準 α で帰無仮説が棄却 (帰無仮説からの有意な差があった，帰無仮説の下ではこうした値 t が観測される可能性は確率 α 以下) とされる．同値であるが，帰無仮説の下での信頼係数 $1-\alpha$ の両側もしくは片側信頼区間が t を含めば (含まなければ) 帰無仮説は有意水準 α で棄却されない (される)．

3.1 ノンパラメトリック検定関数

ノンパラメトリック検定 (nonparametric test) とは，通常母集団分布に対しては密度関数を持つことだけを仮定し，順序 (特にランク) 統計量を基礎とする検定統計量を用いる検定のことである．これらの検定関数の中にはオプションでパラメトリック検定を実行できるものもある．

3.1.1 Ansari-Bradley のスケール差に対する二標本検定 ansari.test()

書式：

```
# 既定のメソッド
ansari.test(x, y, alternative = c("two.sided", "less", "greater"),
            exact = NULL, conf.int = FALSE, conf.level = 0.95, ...)
# クラス "formula" に対する S3 メソッド
ansari.test(formula, data, subset, na.action, ...)
```

引数：

`x` データ値の数値ベクトル

`y` データ値の数値ベクトル

`alternative` 対立仮説．`"two.sided"`, `"greater"` もしくは `"less"` のいずれか．最初の一文字だけでよい

`exact` 論理値．正確な p 値を計算するか？

`conf.int` 論理値．信頼区間を計算するか？

`conf.level` 信頼区間の信頼係数

`formula` 形式 `lhs~rhs` のモデル式で，`lhs` はデータ値を与える数値変数，`rhs` は対応するグループを与える 2 水準の因子

`data` モデル式中の変数を含むオプションのデータフレーム

`subset` 観測値の部分集合を指示するオプションのベクトル

`na.action` 欠損値処理を指示する関数．既定 `getOption("na.action")`

`...` メソッドに (から) 引き渡される追加引数

返り値： クラス `"htest"` のオブジェクトで，次の成分を持つリスト：

`statistic` Ansari-Bradley 検定統計量の値

`p.value` 検定の p 値

`null.value` 帰無仮説下でのスケール比 s で常に 1

`alternative` 対立仮説を示す文字列

`method` 文字列 `"Ansari-Bradley test"`

`data.name` データ名を示す文字列

`conf.int` スケールパラメータの信頼区間 (`conf.int=TRUE` の時だけ存在)

`estimate` スケール比の推定値 (`conf.int=TRUE` の時だけ存在)

x, y をそれぞれ密度関数 $f((t-m)/s)/s$ と $f(t-m)$ を持つ独立な標本とする．ここで，m は未知の局外母数で，スケール比 s が関心のあるパラメータである．Ansari-Bradley 検定は $s = 1$ という帰無仮説を，両側対立仮説 $s \neq 1$ (二つの分布は分散が違う)，片側対立仮説 $s > 1$ (x の分布のほうが分散が大，`"greater"`)，もしくは $s < 1$ (x の分布のほうが分散が小，`"less"`)，に対して検定する．

(`exact = TRUE` を指定しない限り) 既定では双方の標本が 50 個未満の有限な観測値を含みタイ (同じ値のデータ) がなければ正確な p 値を計算する．さもなければ，正規近似が使われる．

オプションで，ノンパラメトリックな信頼区間と s の推定値が計算される．もし正確な p 値が得られれば，Bauer で説明されたアルゴリズムを用いて，正確な信頼区間が計算され，Hodges-Lehmann 推定量が用いられる．さもなければ，信頼区間と推定量は正規近似に基づく．

注意：Ansari-Bradley 検定の結果を，二標本の分散を比較する F 検定のそれと比較するためには，s がスケール比であり，したがって s^2 は分散比 (存在する限り) になることを注意しよう．F 検定では分散比自身が興味のあるパラメータである．特に信頼区間は，Ansari-Bradley 検定では s 用であるが，F 検定では s^2 用である．
関連：k 標本の分散の同一性に対する (ノンパラメトリックな) ランクに基づく `fligner.test()`．スケールパラメータの同一性に対するもう一つのランクによる検定 `mood.test()`．分散の同一性に対するパラメトリック検定 `var.test()`, `bartlett.test()`．

```
# Hollander & Wolfe の対照血清を用いた血清中の鉄分量定量データ
> ramsay <- c(111, 107, 100, 99, 102, 106, 109, 108, 104, 99,
              101, 96, 97, 102, 107, 113, 116, 113, 110, 98)
> jung.parekh <- c(107, 108, 106, 98, 105, 103, 110, 105, 104,
                   100, 96, 108, 103, 104, 114, 114, 113, 108, 106, 99)
# 例えば有意水準 10% で帰無仮説 (スケール比が 1) は棄却されない
> ansari.test(ramsay, jung.parekh)
      Ansari-Bradley test
data:  ramsay and jung.parekh                                  # 二つのデータ変数名
AB = 185.5, p-value = 0.1815                                   # 検定統計量値と p 値
alternative hypothesis: true ratio of scales is not equal to 1  # 両側対立仮説
Warning message:                                # データにタイがあるため正確な p 値が計算不能
Cannot compute exact p-value with ties in:
  ansari.test.default(ramsay, jung.parekh)

# 平均は同じだが分散が異なる二組の正規データを使用 (95% 信頼区間も計算)
# 例えば有意水準 10% で帰無仮説 (スケール比が 1) は棄却されない
> ansari.test(rnorm(10), rnorm(10, 0, 2), conf.int = TRUE)
      Ansari-Bradley test
data:  rnorm(10) and rnorm(10, 0, 2)                           # 二つのデータ変数名
AB = 65, p-value = 0.1521                                      # 検定統計量値と p 値
alternative hypothesis: true ratio of scales is not equal to 1  # 両側対立仮説
95 percent confidence interval:  0.2494223 1.3557745          # スケール比の 95% 信頼区間
sample estimates: ratio of scales  0.5090272                   # 2 標本のスケール比の推定値
```

3.1.2 Fligner-Killeen の分散同一性の (メディアン) 検定 `fligner.test()`

`fligner.test()` は各標本群の分散が同一であるという帰無仮説に対する Fligner-Killeen (メディアン) 検定を行う．

書式：

```
fligner.test(x, ...)
fligner.test(x, g, ...) # 既定の S3 メソッド
# クラス "formula" に対する S3 メソッド
fligner.test(formula, data, subset, na.action, ...)
```

引数：

x	データ値の数値ベクトル，もしくは数値データベクトルのリスト
g	x の対応する要素のグルーピングを与える因子オブジェクトのベクトルで，もし

x がリストなら無視される
formula 形式 lhs~rhs のモデル式． lhs はデータ値， rhs は対応するグループ
data モデル式中の変数を含むオプションのデータフレーム
subset 観測値の部分集合を指示するオプションのベクトル
na.action 欠損値処理を指示する関数．既定 getOption("na.action")
... メソッドに (から) 引き渡される追加引数.

返り値： クラス "htest" のオブジェクトで，次の成分を持つリスト：
statistic Fligner-Killeen のメディアンカイ 2 乗検定統計量
parameter 検定統計量の近似カイ 2 乗分布の自由度
p.value 検定の p 値
method 文字列 "Fligner-Killeen test for homogeneity of variances"
data.name データの名前を与える文字列

もし x がリストなら，その成分が分散の同一性を比較される標本とされ，従って数値ベクトルでなければならない．この場合 g は無視され，検定は単に fligner.test(x) とすれば良い．もし標本がまだリストに含まれていないなら fligner.test(list(x, ...)) とせよ．さもなければ x は数値データベクトルでなければならず，g は x と同じ長さのベクトルか因子オブジェクトで，x の対応する要素のグルーピングを与える．

Fligner-Killeen (メディアン) 検定は，正規性からの逸脱に対し最も頑健な分散同一性に対する多くの検定に対するシミュレーション研究から提案された．Conover, Johnson を参照せよ．これは k 標本単純線形ランクで，中心化された標本の絶対値のランクと重み $a(i) = \mathrm{qnorm}((1 + i/(n + 1))/2)$ を用いる．ここに移植された版は各標本のメディアンで中心化されている (F-K:med X^2 と参照される).

関連：スケールパラメータの違いに対するランクに基づく二標本検定 ansari.test()，mood.test()．分散の同一性に対するパラメトリック検定 var.test(),bartlett.test().

```
# 殺虫剤の効き目データ InsectSprays を使用．0.05>p 値>0.01 であるから，
# 帰無仮説は有意水準 5% で棄却されるが，有意水準 1% では棄却されない
> fligner.test(InsectSprays$count, InsectSprays$spray)
     Fligner-Killeen test for homogeneity of variances
data:  InsectSprays$count and InsectSprays$spray
Fligner-Killeen:med chi-squared = 14.4828, df = 5, p-value = 0.01282

# モデル式によるデータの指定．bartlett.test() と比較してみよ
> fligner.test(count ~ spray, data = InsectSprays)
     Fligner-Killeen test for homogeneity of variances
data:  count by spray
Fligner-Killeen:med chi-squared = 14.4828, df = 5, p-value = 0.01282
```

3.1.3 Friedman のランク和検定 friedman.test()

friedman.test() は繰りかえしの無いブロックデータに対する Friedman のランク和検定を行う．

書式：
friedman.test(y, ...)

```
friedman.test(y, groups, blocks, ...) # 既定の S3 メソッド
# クラス "formula" に対する S3 メソッド
friedman.test(formula, data, subset, na.action, ...)
```

引数：

y　データ値の数値ベクトル，もしくはデータ行列

groups　もし x がベクトルなら，その対応するグループを与えるベクトル．もし y が行列なら無視される．もし因子オブジェクトでなければ，因子に強制変換される

blocks　もし y がベクトルならばその対応する要素のブロックを与える．もし y が行列なら無視される．因子オブジェクトでなければ，因子に強制変換される

formula　形式 a ~ b | c のモデル式で， a, b, c はそれぞれデータ値と対応するグループ，ブロック

data　モデル式中の変数を含むオプションのデータフレーム

subset　観測値の部分集合を指示するオプションのベクトル

na.action　欠損値処理を指示する関数．既定 getOption("na.action")

...　メソッドに (から) 引き渡される追加引数

返り値：　クラス "htest" のオブジェクトで，次の成分を持つリスト：

statistic　Friedman のカイ 2 乗統計量値

parameter　検定統計量の近似カイ 2 乗分布の自由度

p.value　検定の p 値

method　文字列 "Friedman rank sum test"

data.name　データの名前を与える文字列

friedman.test() は繰りかえしの無い完全なブロックデザイン (つまり，groups と blocks の水準の各組合せに対し，y 中の観測値が唯一つある) を解析するのに使われる．データの正規性は仮定しない．帰無仮説は blocks からの影響を除けば，y の位置パラメータが groups の各々において同じというものである．もし y が行列なら，groups と blocks はそれぞれ列と行の添字から得られる．groups と blocks は欠損値 NA を含んではならない．もし y が欠損値を含めば，対応するブロックは除かれる．

関連：quade.test().

```
# Hollander & Wolfe の例．一塁をまわる三種類の方法 ("丸く", "鋭角に", そして"鈍角に") を比較
# 18 人の選手と 3 種類の方法毎の，本塁から 35 フィートの一塁ベース上の位置から，二塁ベースのショ
# ートより 15 フィートの位置までの二回の走塁時間の平均値を記録した
> RoundingTimes <-
   matrix(c(5.40,5.50,5.55,5.85,5.70,5.75,5.20,5.60,5.50,5.55,5.50,5.40,
            5.90,5.85,5.70,5.45,5.55,5.60,5.40,5.40,5.35,5.45,5.50,5.35,
            5.25,5.15,5.00,5.85,5.80,5.70,5.25,5.20,5.10,5.65,5.55,5.45,
            5.60,5.35,5.45,5.05,5.00,4.95,5.50,5.50,5.40,5.45,5.55,5.50,
            5.55,5.55,5.35,5.45,5.50,5.55,5.50,5.45,5.25,5.65,5.60,5.40,
            5.70,5.65,5.55,6.30,6.30,6.25),
          nr = 22, byrow = TRUE,
          dimnames=list(1:22, c("Round Out","Narrow Angle","Wide Angle")))
# 走塁時間は一塁をまわる方法には無関係という帰無仮説を否定する強い証拠 (0.5% 有意)
> friedman.test(RoundingTimes)
        Friedman rank sum test
data:  RoundingTimes
Friedman chi-squared = 11.1429, df = 2, p-value = 0.003805
```

```
# 織機の縦糸の切断回数データ warpbreaks を使用．結果は縦糸の張力の影響を除けば，切断
# 回数の平均値（の位置パラメータ）は羊毛糸の種類によらないという帰無仮説は棄却されない
> wb <- aggregate(warpbreaks$breaks,
                  by=list(w=warpbreaks$wool, t=warpbreaks$tension), FUN=mean)
> wb
  w t        x                                        # w は羊毛糸の種類
1 A L 44.55556                                       # t は縦糸の張力
2 B L 28.22222                                       # x は切断回数の平均値
3 A M 24.00000
4 B M 28.77778
5 A H 24.55556
6 B H 18.77778
> friedman.test(wb$x, wb$w, wb$t)
       Friedman rank sum test
data:  wb$x, wb$w and wb$t
Friedman chi-squared = 0.3333, df = 1, p-value = 0.5637

> friedman.test(x ~ w | t, data = wb)              # モデル式によるデータ指定
       Friedman rank sum test
data:  x and w and t
Friedman chi-squared = 0.3333, df = 1, p-value = 0.5637
```

3.1.4　Kruskal-Wallis のランク和検定 kruskal.test()

kruskal.test() は x 分布の位置パラメータが各グループ (標本) に対して同一であるという帰無仮説に対する Kruskal-Wallis のランク和検定を行う．対立仮説は少なくとも一つが異なるというものである．

書式：

```
kruskal.test(x, ...)
kruskal.test(x, g, ...)  # 既定の S3 メソッド
# クラス "formula" に対する S3 メソッド
kruskal.test(formula, data, subset, na.action, ...)
```

引数：

x	データ値の数値ベクトル，もしくは数値データベクトルのリスト
g	x の対応する要素に対するグループを与えるベクトルもしくは因子オブジェクト．もし x がリストなら無視される
formula	形式 lhs~rhs のモデル式で， lhs はデータ値， rhs は対応するグループ
data	モデル式中の変数を含むオプションのデータフレーム
na.action	欠損値処理を指示する関数．既定 getOption("na.action")
...	メソッドに (から) 引き渡される追加引数．

返り値： クラス "htest" のオブジェクトで，次の成分を持つリスト：

statistic	Kruskal-Wallis のランク和統計量
parameter	検定統計量の近似カイ 2 乗分布の自由度
p.value	検定の p 値
method	文字列 "Kruskal-Wallis rank sum test"
data.name	データの名前を与える文字列

もし x がリストなら，その成分が比較される標本とされ，従って数値データベクトルでなければならない．この場合 g は無視され，検定は単に kruskal.test(x) とすれば良

い．もし標本がまだリストに結合されていなければ `kruskal.test(list(x, ...))` とする．さもなければ `x` は数値データベクトルでなければならず，`g` は `x` と同じ長さのベクトルか因子オブジェクトで，`x` の対応する要素のグルーピングを与える．

関連：二標本に対する特別な場合である Wilcoxon ランク和検定 `wilcox.test()`．正規分布の仮定下での一元配置位置パラメータ解析を行う `lm()` と `anova()` の組合せ．二標本に対する特別な場合としての Student の t 検定 `t.test()`．

```
# 粘液線毛排除機構の効率性．3 グループの分布中心が異なるという帰無仮説は棄却されない
> x <- c(2.9, 3.0, 2.5, 2.6, 3.2)                 # 健常者
> y <- c(3.8, 2.7, 4.0, 2.4)                      # 閉塞性気道疾患患者
> z <- c(2.8, 3.4, 3.7, 2.2, 2.0)                 # 石綿肺症患者
> kruskal.test(list(x, y, z))
Kruskal-Wallis rank sum test
data:  list(x, y, z)
Kruskal-Wallis chi-squared = 0.7714, df = 2, p-value = 0.68

> x <- c(x, y, z)
> g <- factor(rep(1:3, c(5, 4, 5)),               # 同じことだが
            labels = c("Normal subjects",
                       "Subjects with obstructive airway disease",
                       "Subjects with asbestosis"))
> kruskal.test(x, g)
Kruskal-Wallis rank sum test
data:  x and g
Kruskal-Wallis chi-squared = 0.7714, df = 2, p-value = 0.68

# モデル公式による指定．ニューヨーク市の毎日の大気観測値データ airquality を使用
# 月別のオゾン値の比較．オゾン量の月別データの分布中心は明らかに異なる
> kruskal.test(Ozone ~ Month, data = airquality)
Kruskal-Wallis rank sum test
data:  Ozone by Month
Kruskal-Wallis chi-squared = 29.2666, df = 4, p-value = 6.901e-06
```

3.1.5　分布の同一性に対する Kolmogorov-Smirnov 検定 ks.test()

ks.test() は一・二標本 Kolmogorov-Smirnov 検定を実行する．

書式： ks.test(x, y, ..., alternative=c("two.sided","less","greater"), exact = NULL)

引数：

x　　データ値の数値ベクトル

y　　データ値の数値ベクトル，もしくは分布名を与える文字列

...　y(文字列) により指定された分布のパラメータ

alternative　対立仮説． "two.sided", "greater" もしくは "less" のいずれか．最初の一文字だけでよい

exact　NULL または論理値．正確な p 値を検査するか？ NULL の意味は以下を参照．二標本で両側対立仮説の場合だけ使われる

返り値：　クラス "htest" のオブジェクトで，次の成分を持つリスト：

statistic　検定統計量の値

p.value　検定の p 値

alternative　対立仮説を示す文字列

method　どのようなタイプの検定が使われたかを示す文字列

data.name　データの名前を与える文字列

もし y が数値なら x と y が同じ分布に由来するという帰無仮説に対する二標本検定が実行される．さもなければ，y に連続分布の名前を与えることができる．この場合 x が ... で指定されるパラメータを持つ分布 y に由来するという帰無仮説に対する一標本検定が実行される．連続分布ではありえないはずなので，タイが存在すると警告がでる．

対立仮説を指示する引数 alternative の値 "two.sided", "less" そして "greater" は，x の真の分布が想定された分布 (一標本ケース) もしくは y 標本の分布 (二標本ケース) 「と異なる」,「より大きくない」, そして「より小さくない」事を意味する．検定統計量は累積分布関数の差の最大値

$$D^+ = \max_u [F_x(u) - F_y(u)]$$

である．従って alternative="greater" は x の分布が y のそれより確率的に小さい (x の累積分布関数が y のそれの上，従って左側にある) 場合を含む点で，t.test() や wilcox.test() と異なる．

正確な p 値はタイの無い両側二標本検定に対してだけ得られる．この場合，もし exact = NULL (既定値) で標本数の和が 10,000 以下なら正確な p 値が計算される．さもなければ，漸近分布が使用されるが，その近似は小標本の場合不正確かも知れない．

もし一標本検定が使われると，... で指定されるパラメータは事前に与えられる必要があり，データから推定されることは無い．推定パラメータを使った KS 検定に対してはあるより洗練された分布論があるが，これは ks.test() には移植されていない．

関連：正規性に対する Shapiro-Wilk 検定 shapiro.test().

```
> x <- rnorm(50)                                    # 標準正規乱数
> y <- runif(30)                                    # 一様乱数
# 二標本 K-S 検定．x と y は同じ分布に由来するか？
> ks.test(x, y)
        Two-sample Kolmogorov-Smirnov test
data:  x and y
D = 0.64, p-value = 9.731e-08                       # 強く有意
alternative hypothesis: two.sided

# x は形状パラメータ 3，スケールパラメータ 2 のシフトガンマ分布に由来するか？
> ks.test(x+2, "pgamma", 3, 2)                      # 一標本 K-S 検定，両側帰無仮説
        One-sample Kolmogorov-Smirnov test
data:  x + 2
D = 0.2653, p-value = 0.001753                      # 帰無仮説棄却
alternative hypothesis: two.sided

# 一標本 K-S 検定，対立仮説「より大」
> ks.test(x+2, "pgamma", 3, 2, alternative = "gr")
        One-sample Kolmogorov-Smirnov test
data:  x + 2
D^+ = 0.0614, p-value = 0.6856                      # 帰無仮説棄却できず
alternative hypothesis: greater
```

3.1.6 2 元配置分割表の対称性に対する Mcnemar 検定 mcnemar.test()

mcnemar.test() は二元配置分割表の行と列の間の対称性に対する McNemar のカイ 2 乗検定を行う．

書式： mcnemar.test(x, y = NULL, correct = TRUE)

引数：

x	行列形式の二元配置分割表，もしくは因子オブジェクト
y	因子オブジェクト．もし x が行列なら無視される
correct	論理値．検定統計量を計算する際に連続補正をするか？

返り値： クラス "htest" のオブジェクトで，次の成分を持つリスト：

statistic	McNemar 検定統計量の値
parameter	検定統計量の近似カイ 2 乗分布の自由度
p.value	検定の p 値
method	どのようなタイプの検定が使われたかを示す文字列
data.name	データの名前を与える文字列

帰無仮説はセル $[i,j]$ と $[j,i]$ に分類される確率が同一というものである．もし x が行列なら二元配置分割表と見なされ，その成分は非負整数でなければならない．さもなければ x と y の双方は同じ長さのベクトルである必要がある．不完全なケースは除かれ，ベクトルは因子オブジェクトに強制変換され，それから分割表が計算される．correct = TRUE で 2×2 の場合だけ連続補正がされる．

```
# ランダムに選んだ 1600 人の米国有権者による大統領の能力への判断．一月を挟んだ二回の調査結果
# 二度の調査間に強い関連．賛成から反対，反対から賛成へと変えた人は同じ割合とは見なせない
> Performance <-
    matrix(c(794, 86, 150, 570),
           nr = 2,
           dimnames = list("1st Survey" = c("Approve", "Disapprove"),
```

```
                          "2nd Survey" = c("Approve", "Disapprove")))
> Performance                                          # 二期の賛否の二元配置分割表
           2nd Survey
1st Survey   Approve Disapprove
  Approve        794        150
  Disapprove      86        570
> mcnemar.test(Performance)
       McNemar's Chi-squared test with continuity correction
data:  Performance
McNemar's chi-squared = 16.8178, df = 1, p-value = 4.115e-05
```

3.1.7 二標本のスケールパラメータの差異に対する Mood 検定 mood.test()

mood.test() は二標本のスケールパラメータの差異に対する Mood 検定を行う.

書式：

```
# 既定の S3 メソッド
mood.test(x, y, alternative = c("two.sided", "less", "greater"), ...)
# クラス "formula" に対する S3 メソッド
mood.test(formula, data, subset, na.action, ...)
```

引数：

x,y　データ値の数値ベクトル

alternative　対立仮説. "two.sided", "greater" もしくは "less" のいずれか. 最初の一文字だけでよい

formula　形式 lhs~rhs のモデル式で, lhs はデータ値, rhs は対応するグループを与える二つの水準を持つ因子

data　モデル式中の変数を含むオプションのデータフレーム

subset　観測値の部分集合を指示するオプションのベクトル

na.action　欠損値処理を指示する関数. 既定 getOption("na.action")

...　メソッドに (から) 引き渡される追加引数.

返り値： クラス "htest" のオブジェクトで，次の成分を持つリスト：

statistic　検定統計量の値

p.value　検定の p 値

alternative　対立仮説を示す文字列

method　文字列 "Mood two-sample test of scale"

data.name　データの名前を与える文字列

背景にあるモデルは二標本がそれぞれ分布 $f(x-\ell)$ と $f((x-\ell)/s)/s$ に従うデータであるというものであり，ここで ℓ は共通位置パラメータで s はスケールパラメータである．帰無仮説は $s=1$．この問題に対してはもっと役に立つ検定がある．

関連： ランクに基づく (ノンパラメトリックな) 分散の同一性に対する k 標本 fligner.test(). ランクに基づくスケールの同一性に対する 2 標本検定 ansari.test(). パラメトリックな分散の同一性に対する検定 var.test() と bartlett.test().

```
# Hyland 対照血清を用いた血清中の鉄分量定量データ．スケール比が異なるという帰無仮説は棄却できず
# ansari.test(ramsay, jung.parekh) の結果と比較せよ
> ramsay <- c(111, 107, 100, 99, 102, 106, 109, 108, 104, 99,
              101, 96, 97, 102, 107, 113, 116, 113, 110, 98)
> jung.parekh <- c(107, 108, 106, 98, 105, 103, 110, 105, 104,
                   100, 96, 108, 103, 104, 114, 114, 113, 108, 106, 99)
> mood.test(ramsay, jung.parekh)
        Mood two-sample test of scale
data:  ramsay and jung.parekh
Z = 0.9919, p-value = 0.3212
alternative hypothesis: two.sided
```

3.1.8 繰りかえしの無いブロックデータに対する Quade 検定 quade.test()

quade.test() は繰りかえしの無いブロックデータに対する Quade 検定を実行する.

書式:

```
quade.test(y, ...)
quade.test(y, groups, blocks, ...) # 既定の S3 メソッド
# クラス "formula" に対する S3 メソッド
quade.test(formula, data, subset, na.action, ...)
```

引数:

y データ値の数値ベクトル, もしくはデータ行列

groups もしベクトルなら, y の対応する要素のグループを与える. y が行列なら無視される. もし因子オブジェクトでなければ, 因子に強制変換される

blocks もしベクトルなら, y の対応する要素のブロックをを与える. y が行列なら無視される. もし因子でなければ, 因子に強制変換される

formula 形式 a~b|c のモデル式で, a, b そして c はそれぞれデータ値と対応するグループ, ブロック

data モデル式中の変数を含むオプションのデータフレーム

na.action 欠損値処理を指示する関数. 既定 getOption("na.action")

... メソッドに (から) 引き渡される追加引数

返り値: クラス "htest" のオブジェクトで, 次の成分を持つリスト:

statistic Quade 検定統計量の値

parameter 検定統計量の近似 F 分布の分子と分母の自由度を与えるベクトル

p.value 検定の p 値

method 文字列 "Quade test"

data.name データの名前を与える文字列

quade.test() は繰りかえしの無い完備ブロックデザイン (つまり groups と blocks の水準の各組合せに付いて丁度一つの観測値が y 中にある) を解析するのに使われる. 正規性の仮定は無くても良い. 帰無仮説は, blocks の効果を除いて, y の位置パラメータが groups の各々で同一, である. もし y が行列なら, groups と blocks はそれぞれ列と行の添字から得られる. groups もしくは blocks 中の NA は許され無い. もし y が NA を含めば対応するブロックは除去される.

関連: friedman.test().

```
# 7 店舗における 5 種類のハンドローションの週間売上数
> y <- matrix(c(5,4,7,10,12, 1,3,1,0,2, 16,12,22,22,35, 5,4,3,5,4,
                10,9,7,13,10, 19,18,28,37,58, 10,7,6,8,7),
              nr = 7, byrow = TRUE,
              dimnames = list(Store = as.character(1:7),
                              Brand = LETTERS[1:5]))
> y
```

```
     Brand                                          # 店舗がグループ，銘柄がブロックにあたる
Store  A  B  C  D  E
    1  5  4  7 10 12
    2  1  3  1  0  2
    3 16 12 22 22 35
    4  5  4  3  5  4
    5 10  9  7 13 10
    6 19 18 28 37 58
    7 10  7  6  8  7
> quade.test(y)                                    # 店舗による差が無いといえるかどうか微妙
        Quade test                                 # 5% 有意だが，1% 有意でない
data:  y
Quade F = 3.8293, num df = 4, denom df = 24, p-value = 0.01519
```

3.1.9 正規性に対する Shapiro-Wilk 検定 shapiro.test()

shapiro.test() は正規性に対する Shapiro-Wilk 検定を行う．

書式： shapiro.test(x)

引数： x データ値の数値ベクトル．サイズは 3 から 5000 でなければならない．欠損値は許されない

返り値： クラス "htest" のオブジェクトで，次の成分を持つリスト：

statistic Shapiro-Wilk 検定統計量の値

p.value 検定の p 値

method 文字列 "Shapiro-Wilk normality test"

data.name データの名前を与える文字列

関連：正規 QQ プロットを描く qqnorm()．

```
> shapiro.test(rnorm(100, mean = 5, sd = 3))       # 正規乱数データへの適用
        Shapiro-Wilk normality test                 # 正規性は棄却できない
data:  rnorm(100, mean = 5, sd = 3)
W = 0.9925, p-value = 0.8529                        # 検定統計量値と p 値

> shapiro.test(runif(100, min = 2, max = 4))       # 一様乱数データへの適用
        Shapiro-Wilk normality test                 # 正規性は強く棄却
data:  runif(100, min = 2, max = 4)
W = 0.9393, p-value = 0.0001743
```

3.1.10 Wilcoxon のランク和検定 wilcox.test()

wilcox.test() は一・二変量データベクトルに対する Wilcoxon のランク和検定と符号付きランク検定を実行する．

書式：

```
wilcox.test(x, ...)
# 既定の S3 メソッド:
wilcox.test(x, y=NULL, alternative=c("two.sided","less","greater"),
            mu = 0, paired = FALSE, exact = NULL, correct = TRUE,
            conf.int = FALSE, conf.level = 0.95, ...)
```

```
# クラス "formula" に対する S3 メソッド：
wilcox.test(formula, data, subset, na.action, ...)
```

引数：

x　データ値の数値ベクトル

y　オプションのデータ値の数値ベクトル

alternative　対立仮説．"two.sided", "greater" もしくは "less" のいずれか．最初の一文字だけでよい

mu　オプションの位置パラメータを指定する数

paired　論理値．対検定をするかどうか指示

exact　論理値．正確な p 値を計算するかどうか指示

correct　論理値．p 値の正規近似で連続補正をするかどうか指示

conf.int　論理値．信頼区間を計算するかどうか指示

conf.level　信頼区間の信頼係数

formula　形式 lhs~rhs のモデル式で，lhs はデータ値，rhs は対応するグループ

data　モデル式中の変数を含むオプションのデータフレーム

subset　観測値の部分集合を指示するオプションのベクトル

na.action　欠損値処理を指示する関数．既定 getOption("na.action")

...　メソッドに (から) 引き渡される追加引数

返り値：　クラス "htest" のオブジェクトで，次の成分を持つリスト：

statistic　名前付きの検定統計量の値

parameter　検定統計量の正確な分布のパラメータ

p.value　検定の p 値

null.value　位置パラメータ μ

alternative　対立仮説を示す文字列

method　どのようなタイプの検定が使われたかを示す文字列

data.name　データの名前を与える文字列

conf.int　位置パラメータの信頼区間 (conf.int=TRUE の時だけ存在)

estimate　位置パラメータの推定値 (conf.int=TRUE の時だけ存在)

モデル式によるインタフェイスは二標本の場合だけ使える．もし x だけが与えられるか，x と y の双方が与えられ paired = TRUE なら，x (一標本ケース) か x-y (二標本ケース) の分布が mu のまわりに対称であるという帰無仮説に対する Wilcoxon の符号付きランク検定が実行される．さもなければ，もし x と y が与えられ paired = FALSE なら Wilcoxon のランク和検定 (Mann-Whitney 検定と同値) が実行される．この場合帰無仮説は x と y の位置パラメータが mu だけ異なるというものになる．既定では (もし exact が指定されないと)，標本が 50 個以下の有限な値を含みタイが無いなら正確な p 値が計算される．さもなければ，正規近似が使われる．

オプションとして (引き数 conf.int = TRUE なら)，疑似中央値 (一標本ケース) か位置パラメータの差 x-y に対するノンパラメトリックな信頼区間と推定値が計算される．(分布 F の疑似中央値とは，u,v を共に F に従う独立な確率分布としたときの $(u+v)/2$ の中央値である．) もし F が対称なら疑似中央値は中央値に一致する．Hollander & Wolfe

を見よ．もし正確な p 値が得られれば，正確な信頼区間が Bauer 中のアルゴリズムを用いて計算され，Hodges-Lehmann 推定量が用いられる．さもなければ，返される信頼区間と推定値は正規近似に基づく．

関連：2 つ以上の標本の位置パラメータの同一性の検定 `kruskal.test()`．正規標本の仮定下でのパラメトリックな代替検定 `t.test()`．

```
# 一標本検定．不安と抑鬱状態が混じった患者 9 人に対する Hamilton の抑鬱尺度測定値
# 当初 (x) と精神安定剤投与開始後 (y) に測定．x-y の分布が原点対称という帰無仮説
> x <- c(1.83,  0.50,  1.62,  2.48, 1.68, 1.88, 1.55, 3.06, 1.30)
> y <- c(0.878, 0.647, 0.598, 2.05, 1.06, 1.29, 1.06, 3.14, 1.29)
> wilcox.test(x, y, paired = TRUE, alternative = "greater")
        Wilcoxon signed rank test
data:  x and y,  V = 40, p-value = 0.01953        # 5% 有意だが，1% 有意でない
alternative hypothesis: true mu is greater than 0

> wilcox.test(y - x, alternative = "less")        # 異なった片側対立仮説
        Wilcoxon signed rank test
data:  y - x,  V = 5, p-value = 0.01953
alternative hypothesis: true mu is less than 0

> wilcox.test(y - x, alternative = "less",
              exact = FALSE, correct = FALSE)     # H&W 大標本近似使用
        Wilcoxon signed rank test
data:  y - x,  V = 5, p-value = 0.01908
alternative hypothesis: true mu is less than 0
```

```
# 二標本 x,y はそれぞれ出産予定期と妊娠週令 12〜26 週の人間の絨毛膜（胎盤膜）の透過性
# 興味のある対立仮説は「出産予定期の透過性が高い」．結果は 5% 有意だが，1% 有意でない
> x <- c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46)
> y <- c(1.15, 0.88, 0.90, 0.74, 1.21)
> wilcox.test(x, y, alternative = "g")            # 対立仮説「より高い」
        Wilcoxon rank sum test
data:  x and y,     W = 35, p-value = 0.1272
alternative hypothesis: true mu is greater than 0
> wilcox.test(x, y, alternative = "greater",
              exact = FALSE, correct = FALSE)     # H&W 大標本近似使用
        Wilcoxon rank sum test
data:  x and y,     W = 35, p-value = 0.1103
alternative hypothesis: true mu is greater than 0

# モデル式の使用．ニューヨーク市の大気観測値データ airquality 使用．結果は強く有意
> boxplot(Ozone ~ Month, data = airquality)
> wilcox.test(Ozone ~ Month, data = airquality,
             subset = Month %in% c(5, 8))
        Wilcoxon rank sum test with continuity correction
data:  Ozone by Month, W = 127.5, p-value = 0.0001208
alternative hypothesis: true mu is not equal to 0
Warning message:                                  # データにタイがあったので警告
Cannot compute exact p-value with ties in: wilcox.test.default
```

3.2 パラメトリック検定関数

3.2.1 多標本の分散の同一性に対する Bartlett 検定 bartlett.test()

`bartlett.test()` は「複数の標本グループの分散が同一」という帰無仮説を検定する．

書式：

```
# 既定の S3 メソッド
bartlett.test(x, g, ...)
# クラス "formula" に対する S3 メソッド
```

`bartlett.test(formula, data, subset, na.action, ...)`

引数：

`x` データ値の数値ベクトル，各々の標本を表す数値データのリスト，又は (クラス `"lm"` を継承する) 当てはめ線形モデルオブジェクト

`g` x の対応する要素に対するグループを与えるベクトルもしくは因子オブジェクト．もし `x` がリストなら無視される

`formula` 形式 `lhs~rhs` のモデル式で， `lhs` はデータ値， `rhs` は対応グループ

`data` モデル式中の変数を含むオプションのデータフレーム

`subset` 用いられるデータの部分集合を指定するオプションのベクトル

`na.action` 欠損値処理を指示する関数．既定 `getOption("na.action")`

`...` メソッドに (から) 引き渡される追加引数．

返り値： クラス `"htest"` のオブジェクトで，次の成分を持つリスト：

`statistic` Bartlett のカイ 2 乗検定統計量

`parameter` 検定統計量の近似カイ 2 乗分布の自由度

`p.value` 検定の p 値

`method` 文字列 `"Bartlett test for homogeneity of variances"`

`data.name` データの名前を与える文字列

もし `x` がリストなら，その要素が分散の同一性を比較する標本，もしくは当てはめ線形モデルとされる．この場合，要素は全ての数値データベクトルであるか，当てはめ線形モデルオブジェクトである必要がある．`g` は無視され，単に `bartlett.test(x)` とすれば良い．もし標本がまだリストに含まれていなければ，`bartlett.test(list(x, ...))` とする．さもなければ，`x` は数値データベクトルでなければならず，`g` は `x` と同じ長さのベクトルか因子オブジェクトでなければならず．`x` 中の対応する要素のグループ情報を与える．

関連： 二つの正規標本の分散の同一性に付いては `var.test()`．ランクに基づく (ノンパラメトリック) k-標本の分散の同一性に付いては `fligner.test()`．ランクに基づくスケールの差異に関する二標本検定に付いては `ansari.test()` と `mood.test()`．

```
# 6 種類の殺虫剤の効き目データ InsectSprays．効き目の分散は同一という帰無仮説を強く否定
> bartlett.test(InsectSprays$count, InsectSprays$spray)
      Bartlett test for homogeneity of variances
data:  InsectSprays$count and InsectSprays$spray
Bartlett's K-squared = 25.9598, df = 5, p-value = 9.085e-05

> bartlett.test(count ~ spray, data = InsectSprays) # モデル式による指定
      Bartlett test for homogeneity of variances
data:  count by spray
Bartlett}s K-squared = 25.9598, df = 5, p-value = 9.085e-05
```

3.2.2 正確な二項検定 binom.test()

binom.test() はベルヌイ試行の成功確率に関する単純帰無仮説の正確な検定を行う．

書式：

```
binom.test(x, n, p=0.5, alternative=c("two.sided","less","greater"),
           conf.level = 0.95)
```

引数：

x　成功数，もしくは成功数と失敗数を与える長さ 2 のベクトル

n　試行数．もし x が長さ 2 なら無視される

p　帰無仮説である成功確率

alternative　対立仮説を指示し，"two.sided", "greater" もしくは "less"．頭文字だけで十分

conf.level　信頼区間の信頼係数

返り値：　クラス "hetst" のオブジェクトで，次の成分を持つリスト：

statistic　成功数

parameter　試行数

p.value　検定の p 値

conf.int　成功確率に対する信頼区間

estimate　成功確率の推定値

null.value　帰無仮説の下での成功確率 p

alternative　対立仮説を指定する文字列

method　文字列 "Exact binomial test"

data.name　データの名前を与える文字列

信頼区間は Clopper & Pearson により最初に与えられた方法で計算される．これは信頼係数が少なくとも conf.level であることを保証するが，一般には最短にはならない．

関連：一般 (近似) 的な比率の一致，または指定値との一致を検定する prop.test()．

```
# 植物の交配実験での矮小・巨大な子どもの数．メンデルの遺伝法則によれば巨大な子ども
# の比率は 3/4 になる (帰無仮説)．メンデル比に従うという帰無仮説を棄却できない
> binom.test(c(682, 243), p = 3/4) # binom.test(682,682+243,p=3/4) と同一
        Exact binomial test
data:  682 and 925
number of successes = 682, number of trials = 925, p-value = 0.3825  # p 値
alternative hypothesis: true probability of success is not equal to 0.75
95 percent confidence interval:                    # 95% 信頼区間 (3/4 を含む)
 0.7076683 0.7654066
sample estimates:                                  # 成功確率推定値
probability of success
             0.7372973
```

3.2.3 計数データに対する Pearson のカイ 2 乗検定 chisq.test()

chisq.test() は分割表に対するカイ 2 乗検定を実行する．

書式：

```
chisq.test(x, y=NULL, correct=TRUE, p=rep(1/length(x),length(x)),
           simulate.p.value=FALSE, B=2000)
```

引数：

x　ベクトルもしくは行列

y　ベクトル．もし x が行列なら無視される

correct　論理値．検定統計量の計算において連続補正を行うか？

p　x と同じ長さの確率値のベクトル

simulate.p.value　論理値．p 値をモンテカルロシミュレーションで求めるか？

B　モンテカルロシミュレーションの繰りかえし数

返り値：　クラス "htest" のオブジェクトで，次の成分を持つリスト：

statistic　カイ 2 乗検定統計量の値

parameter　検定統計量の近似カイ 2 乗分布の自由度．p 値がモンテカルロシミュレーションで与えられる場合は NA

p.value　検定の p 値

method　文字列で，実行された検定のタイプと，モンテカルロシミュレーションもしくは連続補正が使われたかどうかを示す

data.name　データの名前を与える文字列

observed　観測度数

expected　帰無仮説下での期待度数

residuals　Pearson 残差 (observed-expected)/sqrt(expected)

x が一行または一列の行列であるか，x がベクトルで y が与えられないと，x は一元配置の分割表とされる．この場合，仮説は母集団確率が p で与えられたものに等しいか，または p が与えられなければ，全て等しいかどうか，とされる．

もし x が行と列をそれぞれ少なくとも 2 以上持てば，それは二元配置の分割表と見なされ，したがってその項目は非負整数でなければならない．さもなければ，x と y は同じ長さのベクトルか因子でなければならない．不完全な観測例は取り除かれ，オブジェクトは因子オブジェクトに強制変換され，それらから分割表が計算される．それから，二元配置分割表の同時分布が行と列の周辺分布の積であるという帰無仮説に対する Pearson のカイ 2 乗検定が実行される．もし simulate.p.value = FALSE なら，p 値は検定統計量の漸近カイ 2 乗分布から計算される．連続補正は 2×2 分割表で correct = TRUE の時だけ行われる．さもなければ，もし simulate.p.value = TRUE なら，p 値は B 回のモンテカルロシミュレーションで計算される．これは 与えられた周辺和を持つ全ての分割表の集合からのランダムサンプリングにより行われ，周辺和が正のときだけ使える．

```
# 6 種類の殺虫剤の効き目データ InsectSprays を使用．帰無仮説「殺虫剤毎の効き目が同じ」
> chisq.test(InsectSprays$count > 7, InsectSprays$spray)
        Pearson's Chi-squared test
data:  InsectSprays$count > 7 and InsectSprays$spray
X-squared = 60.9915, df = 5, p-value = 7.582e-12      # 強く否定される
```

```
> x <- matrix(c(12, 5, 7, 7), nc = 2)
> chisq.test(x)$p.value                              # カイ 2 乗分布近似による p 値
[1] 0.4233054                                        # シミュレーションで p 値を計算する
> chisq.test(x, simulate.p.value = TRUE, B = 10000)$p.value
[1] 0.2957704                                 # モンテカルロ法による p 値 (大きく異なる!)
```

```
# 母集団確率の検定. 表形式データ
> x <- c(A = 20, B = 15, C = 25)                     # 一元配置分割表データ
> chisq.test(x)                                      # chisq.test(as.table(x)) と同じ
        Chi-squared test for given probabilities
data:  x
X-squared = 2.5, df = 2, p-value = 0.2865

# 母集団確率の検定. 生データ
> x <- trunc(5 * runif(100))
> chisq.test(table(x))                               # 生データは table() 関数でまず分割表に
        Chi-squared test for given probabilities
data:  table(x)
X-squared = 4.6, df = 4, p-value = 0.3309
```

3.2.4 対になった二標本間の関連／相関の検定 cor.test()

cor.test() は対になった標本[*1] 間の関連度の検定を，Pearson の相関係数 (通常の意味の相関係数)，Kendall の (ランク相関係数) τ，または Spearman の (ランク相関係数) ρ を用いて実行する．

[*1] 対になった二標本 $x = \{x_i\}, y = \{y_i\}$ とは，各対 (x_i, y_i) が $i = 1, \ldots, n$ 番目の対象の二つの属性を表すデータ値であることを意味する．x_i, y_i 間は独立である必要は無いが，対間は独立である必要がある．つまり二次元の独立データが n 個ある状況になる．

書式：

```
# 既定の S3 メソッド
cor.test(x, y, alternative = c("two.sided", "less", "greater"),
         method = c("pearson", "kendall", "spearman"),
         exact = NULL, conf.level = 0.95, ...)
# クラス "formula" に対する S3 メソッド
cor.test(formula, data, subset, na.action, ...)
```

引数：

`x`, `y`　データ値の数値ベクトル．`x` と `y` は同じ長さを持たねばならない

`alternative`　対立仮説を指定する．`"two.sided"`, `"greater"` もしくは `"less"` のいずれか．頭文字を指定するだけで良い．`"greater"` は正の連関，`"less"` は負の連関を意味する

`method`　文字列で，どの相関係数を使うかを指示する．`"pearson"`, `"kendall"` もしくは `"spearman"` のどれかで，頭文字だけで良い

`exact`　論理値．正確な p 値を使うかどうか指定する．Kendall の τ の場合だけ意味がある．既定の `NULL` の意味は以下を参照せよ

`conf.level`　信頼区間の信頼係数．現在，少なくとも 4 組以上の完全なペアがあるときの Pearson の相関係数だけに意味がある

`formula`　形式 `~u+v` のモデル式で，`u` と `v` は一つの標本のデータ値を与える数値ベクトル．二つの標本は同じ長さでなければならない

`data`　モデル式中の変数を含むオプションのデータフレーム

`subset`　観測値の部分集合を指示するオプションのベクトル

`na.action`　欠損値処理を指示する関数．既定 `getOption("na.action")`

`...`　メソッドに (から) 引き渡される追加引数．

返り値：　クラス `"htest"` のオブジェクトで，次の成分を持つリスト：

`statistic`　検定統計量の値

`parameter`　検定統計量が t 分布に従う場合のその自由度

`p.value`　検定の p 値

`estimate`　連関度の推定値．用いた手法に応じて `"cor"`, `"tau"` または `"rho"`

`null.value`　帰無仮説の下での連関度の値で，常に 0

`alternative`　対立仮説を表す文字列

`method`　連関をどのように計ったかを示す文字列

`data.name`　データの名前を与える文字列

`conf.int`　連関度の信頼区間．現在，少なくとも 4 組以上の完全なペアがあるときの Pearson の相関係数だけに与えられる

三つの手法は，それぞれ対になった標本間の関連度を推定し，それが 0 であるという検定を行う．これらは異なった関連度を計算し，全て範囲 $[-1, 1]$ にあり，値 0 は関連の無いことを意味する．これらは時おり無相関性の検定と呼ばれるが，この言葉はしばしば既定の手法に限定される．

もし method が "pearson" ならば，検定統計量は Pearson の積率相関係数 cor(x, y) であり，もし標本が独立な正規分布に従うなら，帰無分布は自由度 length(x)-2 の t 分布に従う．もし最低 4 対の完全データがあれば，漸近信頼区間が Fisher の Z 変換で与えられる．

もし method が "kendall" もしくは "spearman" ならば，Kendall の τ もしくは Spearman の ρ 統計量がランクに基づく関連度を推定するのに使われる．これらの検定はもしデータが二変量正規分布に従わない場合でも使うことができる．

Kendall の検定は，既定 (exact が NULL) の場合，もし有限な値の対標本が 50 組以下で，タイが無ければ，正確な p 値が計算される．さもなければ，検定統計量は平均 0 で単位分散にスケール化された推定値であり，近似的に正規分布に従う．Spearman の検定では，p 値はアルゴリズム AS 89 を用いて計算される．

```
# まぐろの缶詰 9 ロットに対し Hunter の L 明度と，消費者グループの得点採点
# (1 から 6 の整数で 80 組の平均) を比較する
> x <- c(44.4, 45.9, 41.9, 53.3, 44.7, 44.1, 50.7, 45.2, 60.1)
> y <- c( 2.6,  3.1,  2.5,  5.0,  3.6,  4.0,  5.2,  2.8,  3.8)
# 興味のある対立仮説「Hunter の L 値は消費者の採点と正の連関」
> cor.test(x, y, method = "kendall", alternative = "greater")
       Kendall's rank correlation tau
data:  x and y
T = 26, p-value = 0.05972                          # 5% 有意でない
alternative hypothesis: true tau is greater than 0
sample estimates:  tau  0.4444444                  # Kendall のランク相関係数 (τ 統計量) 値

# 大標本近似を用いる
> cor.test(x, y, method = "kendall", alternative = "greater",
            exact = FALSE)
       Kendall's rank correlation tau
data:  x and y
z = 1.6681, p-value = 0.04765                      # 5% 有意
alternative hypothesis: true tau is greater than 0
sample estimates:  tau  0.4444444

# Spearman のランク相関係数 (ρ 統計量) 値を指定
> cor.test(x, y, method = "spearm", alternative = "g")
      Spearman's rank correlation rho
data:  x and y
S = 48, p-value = 0.0484                           # 5% 有意
alternative hypothesis: true rho is greater than 0
sample estimates:  rho  0.6

# Pearson の (普通の意味の) 相関係数値を指定
> cor.test(x, y, alternative = "g")
      Pearson's product-moment correlation
data:  x and y
t = 1.8411, df = 7, p-value = 0.05409              # 5% 有意でない
alternative hypothesis: true correlation is greater than 0
95 percent confidence interval:                    # 相関係数の 95% 信頼区間
 -0.02223023  1.00000000
sample estimates: cor 0.5711816
```

```
# モデル式による指定．米国最高裁判事の評価データ USJudgeRatings を使用
> pairs(USJudgeRatings)                            # 散布行列図を描く
# 法律家との接触度と法的公正さの連関を検定 (関連無しの結論)
> cor.test(~ CONT + INTG, data = USJudgeRatings)
        Pearson's product-moment correlation
data:  CONT and INTG
t = -0.8605, df = 41, p-value = 0.3945
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:                    # 相関係数の 95% 信頼区間
 -0.4168591  0.1741182
sample estimates:  cor  -0.1331909                 # Pearson の (普通の意味の) 相関係数値
```

3.2.5　計数データに対する Fisher の正確検定 fisher.test()

fisher.test() は，周辺和が固定された分割表の行と列が独立であるという帰無仮説に対する Fisher の正確検定を実行する．

書式： fisher.test(x, y=NULL, workspace=200000, hybrid=FALSE,
control=list(), or=1, alternative="two.sided",
conf.level=0.95)

引数：

x　行列形式の二元配置分割表か，因子オブジェクト

y　因子オブジェクト．もし x が行列なら無視される

workspace　整数．ネットワークアルゴリズムで使われる作業空間のサイズ

hybrid　論理値．正確な確率 (既定値) か，その近似を計算するかを指示する．近似の場合，近似カイ 2 乗確率は「Cochran 条件」が成り立つときだけ使われる

control　低レベルのアルゴリズム制御のための名前付き成分を持つリスト

or　帰無仮説となるオッズ比．2×2 の場合だけ使われる

alternative　対立仮説を指示し，"two.sided", "greater" または "less" のどれか．頭文字だけで良い．2×2 の場合だけ使われる

conf.level　信頼区間の信頼係数．2×2 分割表の場合だけ使われる

返り値：　クラス "htest" のオブジェクトで，次の成分を持つリスト：

p.value　検定の p 値

conf.int　オッズ比に対する信頼区間．2×2 の場合存在する

estimate　オッズ比の推定値．無条件最尤推定量 (標本オッズ比) ではなく，条件付き最尤推定量が使われることを注意．2×2 の場合だけ存在する

null.value　帰無仮説に対するオッズ比．2×2 の場合だけ存在する

alternative　対立仮説を表す文字列

method　文字列 ”Fisher’s Exact Test for Count Data”

data.name　データの名前を与える文字列

もし x が行列なら，それは二元配置分割表とされ，その項目は非負整数でなければならない．さもなければ，x と y の双方は同じ長さのベクトルでなければならない．不完全なケースは除かれ，ベクトルは因子オブジェクトに強制変換され，これから分割表が計算される．

片側対立仮説の 2×2 分割表の場合，p 値は超幾何分布を用いて直接に計算される．さもなければ，計算は Mehta & Patel により開発され，Clarkson, Fan & Joe により改良されたネットワークアルゴリズムを移植した FORTRAN サブルーチン FEXACT の C 版でなされる．これは，表の項目が大きすぎると失敗することを注意しよう．

2×2 分割表の場合，条件付き独立性の帰無仮説はオッズ比が 1 であるという仮説に等しい．一般に，全ての周辺和が固定されると，分割表の最初の要素は非中心化パラメータがオッズ比で与えられる非中心化超幾何分布に従う (Fisher) ことを注意すれば，正確な推測が可能になる．

関連：chisq.test().

```
# Fisher の喫茶データ．ある英国女性がカップに先に加えられたのが紅茶かミルクかを判別できると主張，
# 8 杯のカップ（内 4 杯は先にミルクが注がれた）を使った実験を行った．帰無仮説は真の順序と女性の推測
# には何の関係もない，対立仮説は正の関連がある（つまりオッズ比が 1 より大きい）
> TeaTasting <- matrix(c(3, 1, 1, 3), nr = 2,
                       dimnames = list(Guess = c("Milk", "Tea"),
                                       Truth = c("Milk", "Tea")))
> TeaTasting
      Truth
Guess  Milk Tea
  Milk    3   1                                  # ミルクが先と答えたうち正解 3 件
  Tea     1   3                                  # 紅茶が先と答えたうち正解 3 件
> fisher.test(TeaTasting, alternative = "greater")
       Fisher's Exact Test for Count Data
data:  TeaTasting                                # 正の相関は検出できなかった
p-value = 0.2429                                 # （データ数 8 では所詮この程度）
alternative hypothesis: true odds ratio is greater than 1
95 percent confidence interval:                  # オッズ比の 95% 信頼区間
  0.3135693       Inf
sample estimates: odds ratio  6.408309           # オッズ比の推定値
```

```
# Fisher のデータ．同性双子（一卵性と二卵性）がともに有罪となったかどうか
# 双子の犯罪性向が類似するかどうかを知りたい
> Convictions <-  matrix(c(2, 10, 15, 3), nr = 2,
                         dimnames = list(c("Dizygotic", "Monozygotic"),
                                         c("Convicted", "Not convicted")))
> Convictions
            Convicted Not convicted
Dizygotic           2            15               # 二卵性双生児がともに犯罪歴を持つかどうか
Monozygotic        10             3               # 一卵性双生児がともに犯罪歴を持つかどうか
> fisher.test(Convictions, alternative = "less")  # 対立仮説：オッズ比<=1
       Fisher's Exact Test for Count Data
data:  Convictions
p-value = 0.0004652                              # 強い関連
alternative hypothesis: true odds ratio is less than 1
95 percent confidence interval:                  # オッズ比の 95% 信頼区間
 0.0000000 0.2849601
sample estimates: odds ratio  0.04693661         # オッズ比の推定値
>   fisher.test(Convictions, conf.level = 0.99)$conf.int
[1] 0.001386333 0.578851645                      # 両側対立仮説に対する 99% 信頼区間
attr(,"conf.level")
[1] 0.99                                         # 信頼係数 99%
```

3.2.6 Cochran-Mantel-Haenszel カイ 2 乗検定 mantelhaen.test()

mantelhaen.test() は，3 次の相互作用が無いという仮定の下で，二つの名義尺度変数が各層毎に条件付き独立であるという帰無仮説の Cochran-Mantel-Haenszel カイ 2 乗検定を実行する．

書式：

```
mantelhaen.test(x, y = NULL, z = NULL,
                alternative = c("two.sided", "less", "greater"),
                correct = TRUE, exact = FALSE, conf.level = 0.95)
```

引数：

x	各次元が少なくとも 2 で最後が層に対応する配列形式の 3 元配置分割表か，少なくとも 2 水準を持つ因子オブジェクト
y	少なくとも 2 水準を持つ因子オブジェクト．もし x が配列なら無視
z	少なくとも 2 水準を持つ因子オブジェクトで， x と y 中の要素がどの層に属す

るかを指定する．もし x が配列なら無視される

alternative 対立仮説． "two.sided", "greater" もしくは "less"．最初の一文字だけでよい．$2 \times 2 \times K$ ケースだけで使われる

correct 論理値．検定統計量を計算するとき連続補正をするかどうかを指示する．$2 \times 2 \times K$ ケースだけで使われる

exact 論理値．Mantel-Haenszel 検定を使うか，(層の周辺和を与えた) 正確な条件付き検定を使うかを指示する．$2 \times 2 \times K$ ケースだけで使われる

conf.level 信頼区間の信頼係数．$2 \times 2 \times K$ ケースだけで使われる

返り値： クラス "htest" のオブジェクトで，次の成分を持つリスト：

statistic 正確検定が使われないときだけ存在する．古典的な $2 \times 2 \times K$ 表の場合 (背景変数が二値のケース)，Mantel-Haenszel カイ 2 乗検定統計量，さもなければ一般化された Cochran-Mantel-Haenszel 統計量

parameter 検定統計量の近似カイ 2 乗分布の自由度 (古典的ケースでは 1)．正確検定が使われないときだけ存在する

p.value 検定の p 値

conf.int 共通オッズ比に対する信頼区間．$2 \times 2 \times K$ ケースだけ存在する

estimate 共通オッズ比に対する推定値．もし正確検定が実行されると，条件付き最尤推定量が与えられる．さもなければ Mantel-Haenszel 推定値．$2 \times 2 \times K$ ケースだけ存在する

null.value 帰無仮説下での共通オッズ比．$2 \times 2 \times K$ ケースだけ存在する

alternative 対立仮説を示す文字列．$2 \times 2 \times K$ の場合だけ存在する

method どのようなタイプの検定が使われたか，連続補正が使われたかを示す文字列

data.name データの名前を与える文字列

もし x が配列なら，各次元は少なくとも 2 でなければならず，要素は非負整数である必要がある．欠損値 NA は許されない．さもなければ x, y そして z は同じ長さを持つ必要がある．NA を含む三つ組は除かれる．全ての変数は少なくとも 2 つの異なった値を持たねばならない．

注意：漸近分布は 3 次の相互作用が無いときだけ意味を持つ．古典的な $2 \times 2 \times K$ ケースでは，これは各層での条件付きオッズ比が同一であることと同値である．現在のところ，オッズ比の同一性に対する如何なる検査もされない．以下の例を見よ．

```
# 兎への致死的なβ溶血連鎖球菌の注射に対する，直後と一時間半後のペニシリン注射の効果
> Rabbits <-
    array(c(0,0,6,5, 3,0,3,6, 6,2,0,4, 5,1,6,0, 2,5,0,0),
          dim = c(2, 2, 5),
          dimnames = list(Delay = c("None", "1.5h"),
                          Response = c("Cured", "Died"),
                          Penicillin.Level = c("1/8","1/4","1/2","1","4")))
> mantelhaen.test(Rabbits)                          # 古典的 Mantel-Haenszel 検定
       Mantel-Haenszel chi-squared test with continuity correction
data:  Rabbits                                                    # 5% 有意
Mantel-Haenszel X-squared = 3.9286, df = 1, p-value = 0.04747     # 1% 有意でない
alternative hypothesis: true common odds ratio is not equal to 1
95 percent confidence interval:   1.026713 47.725133             # 95% 信頼区間
sample estimates: common odds ratio  7                            # オッズ比の推定値
> mantelhaen.test(Rabbits, exact = TRUE)                          # 正確な条件付き検定
```

```
        Exact conditional test of independence in 2 x 2 x k tables
data:  Rabbits
S = 16, p-value = 0.03994                                          # 5% 有意
alternative hypothesis: true common odds ratio is not equal to 1   # 1% 有意でない
95 percent confidence interval:  1.077401 529.837399
sample estimates: common odds ratio 10.36102

# 「直後の注射により高い治癒率」という片側対立仮説に対する正確な条件付き検定
> mantelhaen.test(Rabbits, exact = TRUE, alternative = "greater")
        Exact conditional test of independence in 2 x 2 x k tables
data:  Rabbits
S = 16, p-value = 0.01997                                          # 5% 有意
alternative hypothesis: true common odds ratio is greater than 1   # 1% 有意でない
95 percent confidence interval: 1.384239      Inf
sample estimates: common odds ratio 10.36102
```

```
# UC Berkley 校の入学希望者データ UCBAdmissions を使用
> mantelhaen.test(UCBAdmissions)
        Mantel-Haenszel chi-squared test with continuity correction
data:  UCBAdmissions
Mantel-Haenszel X-squared = 1.4269, df = 1, p-value = 0.2323
alternative hypothesis: true common odds ratio is not equal to 1
95 percent confidence interval: 0.7719074 1.0603298
sample estimates: common odds ratio 0.9046968

# 学部の違いで補正すれば入学者と性別に関連があるという証拠は無い
# しかし (条件付きの) オッズ比を検査すると一様とは思われない
> apply(UCBAdmissions, 3, function(x) (x[1,1]*x[2,2])/(x[1,2]*x[2,1]))
        A         B         C         D         E         F
0.3492120 0.8025007 1.1330596 0.9212838 1.2216312 0.8278727

# 交互作用に対する伝統的な Woolf 検定で調べてみると p 値 0.003
# これは有意な異質性を示唆し，Mantel-Haenszel 検定を使うべきではない
> woolf <- function(x) {
    x <- x + 1 / 2; k <- dim(x)[3]
    or <- apply(x, 3, function(x) (x[1,1]*x[2,2])/(x[1,2]*x[2,1]))
    w <-  apply(x, 3, function(x) 1 / sum(1 / x))
    1 - pchisq(sum(w * (log(or) - weighted.mean(log(or), w)) ^ 2), k - 1)
  }
> woolf(UCBAdmissions)
[1] 0.003427200
```

3.2.7 一元配置正規標本の平均の同一性検定 oneway.test()

oneway.test() は一元配置の 2 もしくはそれ以上の正規標本の平均の同一性を検定する．分散は必ずしも同一でなくても良い．

書式: oneway.test(formula, data, subset, na.action, var.equal = FALSE)

引数：

formula　形式 lhs~rhs のモデル式で， lhs はデータ値， rhs は対応するグループ

data　モデル式中の変数を含むオプションのデータフレーム

subset　観測値の部分集合を指示するオプションのベクトル

na.action　欠損値の処理を指示する関数．既定は getOption("na.action")

var.equal　論理値．標本の分散が同一と見なすかどうかを指示する． TRUE なら一元配置分散分析による平均の同一性に対する単純な F 検定が実行される．もし FALSE なら Welch の近似検定が使われる．これは普通の二標本 Welch 検定を任意多標本に拡張したものである

返り値： クラス "htest" のオブジェクトで，次の成分を持つリスト：

statistic　検定統計量の値
parameter　検定統計量の正確もしくは近似 F 分布の自由度
p.value　検定の p 値
method　どのタイプの検定が使われたかを示す文字列
data.name　データの名前を与える文字列.

関連：二標本に対する特殊例である標準 t 検定 t.test(). 一元配置に於ける位置パラメータの同一性に対するノンパラメトリック検定 kruskal.test().

```
# Student の睡眠薬データ sleep を使用. 分散の同一性を仮定しない場合
# 睡眠薬の効き目の同一性は有意水準 5% で棄却できず
> oneway.test(extra ~ group, data = sleep)
        One-way analysis of means (not assuming equal variances)
data:  extra and group
F = 3.4626, num df = 1.000, denom df = 17.776, p-value = 0.0794

# 分散の同一性を仮定した場合 (同一性は有意水準 5% で棄却できない)
> oneway.test(extra ~ group, data = sleep, var.equal = TRUE)
        One-way analysis of means
data:  extra and group
F = 3.4626, num df = 1, denom df = 18, p-value = 0.07919

> anova(lm(extra ~ group, data = sleep))          # 分散分析は同じ結果を与える
Analysis of Variance Table
Response: extra
          Df Sum Sq Mean Sq F value  Pr(>F)
group      1 12.482  12.482  3.4626 0.07919 .
Residuals 18 64.886   3.605
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

3.2.8　複数のグループの比率の同一性の検定 prop.test()

prop.test() は複数のグループの比率 (成功確率) が等しい，または与えられた値に等しいという帰無仮説を検定する.

書式： `prop.test(x, n, p = NULL,`
`alternative = c("two.sided", "less", "greater"),`
`conf.level = 0.95, correct = TRUE)`

引数：

`x` 成功回数のベクトル，または成功・失敗回数を与える二つの列を持つ行列

`n` 試行回数のベクトル．もし `x` が行列ならば無視される

`p` 成功確率のベクトル． `p` の長さは `x` で指定されるグループ数と同じ必要

`alternative` 対立仮説． `"two.sided"`, `"greater"` もしくは `"less"` のいずれか．最初の一文字だけでよい．単一の母集団の比率が指定値に等しいか，二つの比率が等しいという帰無仮説の検定だけで使われる．さもなければ無視される

`conf.level` 返される信頼区間の信頼係数．単一の母集団の比率が指定値に等しいか，二つの比率が等しいという帰無仮説の検定だけで使われる．さもなければ無視

`correct` 論理値．Yates の連続補正をするか

返り値： クラス `"htest"` のオブジェクトで，次の成分を持つリスト：

`statistic` Pearson のカイ 2 乗検定統計量の値

`parameter` 検定統計量の近似カイ 2 乗分布の自由度

`p.value` 検定の p 値

`estimate` 標本比率 x/n のベクトル

`conf.int` グループが一つなら真の比率，二つのグループがあり p が与えられなければ比率の差に対する信頼区間．さもなければ `NULL`．これが `NULL` でない場合は返される信頼区間は `conf.level` で指定される漸近信頼水準を持ち，指定された対立仮説に対応する

`null.value` 帰無仮説で指定された場合の p の値，さもなければ `NULL`

`alternative` 対立仮説を示す文字列

`method` どのようなタイプの検定が使われたか，Yates 連続補正行ったかを示す文字列

`data.name` データの名前を与える文字列．

成功・失敗回数が有限であるグループだけが使われる．成功・失敗回数は非負で，したがって正でなければならない対応する試行回数よりも大きくてはいけない．全ての有限なカウント数は整数でなければならない．

もし `p` が `NULL` で，複数のグループがあれば，検定される帰無仮説は各グループの比率が等しいというものである．もし二つのグループがあれば，対立仮説は `alternative` で指定された内容に応じて，第一グループの成功確率が第二グループのそれよりも「より小さい」，「等しくない」そして「より大きい」である．`conf.level` で指定された信頼係数を持つ，比率の差異に対する信頼区間は区間 $[-1, 1]$ 中に切り詰められる．連続補正は，それが絶対値で標本比率差を越えないときだけ使われる．さもなければ，もし 3 つ以上のグループがあれば，対立仮説は常に `"two.sided"` であり，返される信頼区間は `NULL` であり，連続補正は決して使われない．

もし一つのグループだけがあれば，検定される帰無仮説は根底にある成功確率が `p` であるというものである．もし `p` が与えられなければ `p=0.5` とされる．対立仮説は

alternative で指定された内容に応じて，成功確率が p もしくは 0.5 より「小さい」，「等しくない」そして「大きい」となる．conf.level で指定された信頼係数を持つ，比率の差異に対する信頼区間は区間 $[-1, 1]$ 中に切り詰められる．連続補正は，それが絶対値で標本比率と帰無仮説比率の差を越えないときだけ使われる．信頼区間はスコア検定を逆転して計算される．

最後に，もし p が与えられ二つ以上のグループがあれば，帰無仮説は根底にある成功確率が p で与えられたものになるということである．対立仮説は常に "two.sided" であり，返される信頼区間は NULL であり，連続補正は決して使われない．

関連：二項分布仮説に対する正確検定 binom.test().

```
> heads <- rbinom(1, size=100, pr = .5)
> prop.test(heads, 100)                              # 既定で連続補正される
        1-sample proportions test with continuity correction
data:  heads out of 100, null probability 0.5
X-squared = 0.01, df = 1, p-value = 0.9203
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:  0.4086512 0.6105719
sample estimates:   p  0.51

> prop.test(heads, 100, correct = FALSE)
        1-sample proportions test without continuity correction
data:  heads out of 100, null probability 0.5
X-squared = 0.04, df = 1, p-value = 0.8415
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:  0.4134801 0.6057800
sample estimates:   p  0.51
```

```
# 帰無仮説「患者が属する 4 つの母集団が同じ喫煙者比率を持つ」. 1% 有意
> smokers  <- c( 83, 90, 129, 70 )
> patients <- c( 86, 93, 136, 82 )
> prop.test(smokers, patients)
        4-sample test for equality of proportions without continuity
        correction
data:  smokers out of patients
X-squared = 12.6004, df = 3, p-value = 0.005585
alternative hypothesis: two.sided
sample estimates:    prop 1    prop 2    prop 3    prop 4
                  0.9651163 0.9677419 0.9485294 0.8536585
```

3.2.9 比率中の傾向に対するカイ 2 乗検定 prop.trend.test()

prop.trend.test() は比率中の傾向に対するカイ 2 乗検定を行う．つまり，対数オッズが score に応じて変化するという局所対立仮説に対する漸近的に最適な検定である．既定では score はグループ数とされる．

書式： prop.trend.test(x, n, score = 1:length(x))

引数：

x	事象数
n	試行数
score	グループスコア

返り値： クラス "htest" のオブジェクトで，タイトル，検定統計量値，p 値を含む

関連：prop.test().

```
> smokers  <- c( 83, 90, 129, 70 )                # 4 患者グループ中の喫煙者数
> patients <- c( 86, 93, 136, 82 )                # 4 患者グループの数
> prop.test(smokers, patients)
         4-sample test for equality of proportions without continuity
         correction
data:   smokers out of patients
X-squared = 12.6004, df = 3, p-value = 0.005585
alternative hypothesis: two.sided
sample estimates:                                 # 喫煙者数比率
   prop 1     prop 2    prop 3    prop 4
0.9651163  0.9677419 0.9485294 0.8536585

> prop.trend.test(smokers, patients)
         Chi-squared Test for Trend in Proportions
data:  smokers out of patients ,
 usingscores: 1 2 3 4          # スコアは 4 グループ (比率が 4 グループで異なるという対立仮説)
X-squared = 8.2249, df = 1, p-value = 0.004132

> prop.trend.test(smokers, patients,c(0,0,0,1))
         Chi-squared Test for Trend in Proportions
data: smokers out of patients ,
 using scores: 0 0 0           # 比率が第 4 グループだけで異なるという対立仮説
X-squared = 12.1731, df = 1, p-value = 0.0004848
```

3.2.10 Student の t 検定 t.test()

t.test() はデータベクトルに対し一・二標本 t 検定を行う.

書式：

```
t.test(x, ...)
# 既定の S3 メソッド
t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
       mu=0, paired=FALSE, var.equal=FALSE, conf.level=0.95, ...)
# クラス "formula" に対する S3 メソッド
t.test(formula, data, subset, na.action, ...)
```

引数：

`x`　データ値の数値ベクトル

`y`　オプションのデータ値の数値ベクトル

`alternative`　対立仮説．`"two.sided"`, `"greater"` もしくは `"less"`．最初の一文字だけでよい

`mu`　平均の真値を指定する数 (もしくは，二標本検定なら平均差)

`paired`　論理値．対になった t 検定を行うか

`var.equal`　論理値．二つの分散が等しいと見なすかどうか．`TRUE` なら分散の推定はプールした分散から計算され，さもなければ Wlech(または Satterthwaite) 近似による自由度が使われる

`conf.level`　信頼区間の信頼係数

`formula`　形式 `lhs~rhs` のモデル式で，`lhs` はデータ値，`rhs` は対応する 2 グループを与える 2 水準の因子

`data`　モデル式中の変数を含むオプションのデータフレーム

`subset`　観測値の部分集合を指示するオプションのベクトル

`na.action`　欠損値処理を指示する関数．既定 `getOption("na.action")`

`...`　メソッドに (から) 引き渡される追加引数

返り値：　クラス `"htest"` のオブジェクトで，次の成分を持つリスト：

`statistic`　t 検定統計量の値

`parameter`　t 検定統計量の自由度

`p.value`　検定の p 値

`conf.int`　対立仮説に応じた信頼区間

`estimate`　一標本か二標本かに応じて，平均か平均の差の推定値

`null.value`　一標本か二標本かに応じて，平均か平均の差の帰無仮説値

`alternative`　対立仮説を示す文字列

`method`　どのようなタイプの検定が使われたかを示す文字列

`data.name`　データの名前を与える文字列

モデル式によるインタフェイスは二標本の場合だけ使える．もし `paired = TRUE` なら，`x` と `y` の双方を指定する必要があり，同じ長さでなければならない．欠損値は除かれる (`paired = TRUE` なら対で)．もし `var.equal = TRUE` ならプールされた分散の推定値が使われれる．既定では，もし `var.equal = FALSE` なら分散は両群で個別に推定され，自由度の Welch 修正が使われる．

関連：prop.test().

```
# Student の睡眠薬データ sleep を使用．帰無仮説「二種類の睡眠薬の効き目は同一」
> with(sleep, t.test(extra[group == 1], extra[group == 2]))
       Welch Two Sample t-test                          # Welch 二標本検定が使われた
data:  extra[group == 1] and extra[group == 2]
t = -1.8608, df = 17.776, p-value = 0.0794              # 有意水準 5% で棄却されず
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:  -3.3654832  0.2054832  # 平均差の 95% 信頼区間
sample estimates: mean of x mean of y                   # 二標本の平均値の推定値
                  0.75      2.33

> t.test(extra ~ group, data = sleep)                   # モデル式によるインタフェイス
        Welch Two Sample t-test
data:  extra by group
t = -1.8608, df = 17.776, p-value = 0.0794
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval: -3.3654832  0.2054832
sample estimates: mean in group 1 mean in group 2
                  0.75            2.33
```

3.2.11 二つの分散の同一性に対する F 検定 var.test()

var.test() は正規データからの二標本の分散を比較する F 検定を行う．

書式：

```
var.test(x, ...)
# 既定の S3 メソッド
var.test(x, y, ratio = 1,
         alternative = c("two.sided", "less", "greater"),
         conf.level = 0.95, ...)
# クラス "formula" に対する S3 メソッド
var.test(formula, data, subset, na.action, ...)
```

引数：

x, y　データの数値ベクトル．もしくは当てはめ線形モデル (クラス "lm" を継承する)

ratio　帰無仮説である x と y の母集団分散の比

alternative　対立仮説．"two.sided", "greater" もしくは "less"．最初の一文字で可

conf.level　信頼区間の信頼係数

formula　形式 lhs~rhs のモデル式で，lhs はデータ値，rhs は対応するグループを与える 2 水準の因子

data　モデル式中の変数を含むオプションのデータフレーム

subset　観測値の部分集合を指示するオプションのベクトル

na.action　欠損値処理を指示する関数．既定 getOption("na.action")

...　メソッドに (から) 引き渡される追加引数．

返り値：　クラス "htest" のオブジェクトで，次の成分を持つリスト：

statistic　F 検定統計量の値

parameter　検定統計量の F 分布の自由度

p.value　検定の p 値

conf.int　母集団分散比の信頼区間
estimate　x と y の分散比の推定値
null.value　帰無仮説下での分散比
alternative　対立仮説を示す文字列
method　文字列 "F test to compare two variances"
data.name　データの名前を与える文字列

帰無仮説は x と y，もしくは x と y の線形モデルが当てはめられたデータの母集団の分散の比が ratio に等しいというものである．

関連：正規分布からの複数標本の分散の同一性を検定する bartlett.test()．スケールの違いをランクに基づいて (ノンパラメトリック) 検定する ansari.test() と mood.test()．

```
> x <- rnorm(50, mean = 0, sd = 2)
> y <- rnorm(30, mean = 1, sd = 1)
> var.test(x, y)                                    # x と y は同じ分散を持つか？
        F test to compare two variances
data:  x and y
F = 4.7971, num df = 49, denom df = 29, p-value = 2.426e-05
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval: 2.410192 9.025412 # 95% 信頼区間
sample estimates: ratio of variances  4.797137    # 推定分散比

> var.test(lm(x ~ 1), lm(y ~ 1))                   # 同じこと
```

3.3　多重比較補正付きの検定関数

一つのデータ $\boldsymbol{X}$ から複数の検定を行うとする．各検定の帰無仮説を $H_1, H_2, \ldots, H_k$ とし，集合 $C_i(\alpha)$ を帰無仮説 H_i の下での有意水準 α の棄却域とする．全ての帰無仮説 H_i が同時に成り立つという帰無仮説 H を考える．ある帰無仮説 H_i が棄却されれば H も棄却される (おなじことだが，全ての H_i が棄却されなければ H も棄却されない) ようにするのが一見自然と思われる (これは $\cup_i C_i(\alpha)$ を H の棄却域にすることと同値)．しかし，こうした検定の有意水準は普通 α にはならず，一般的にいえることは

$$\boldsymbol{P}\{\boldsymbol{X} \in \cup_i C_i(\alpha)\} \leq \sum_i \boldsymbol{P}\{\boldsymbol{X} \in C_i(\alpha)\} = k\alpha$$

だけである (検定の多重比較問題)．ここで

$$\boldsymbol{P}\{\boldsymbol{X} \in \cup_i C_i(\alpha/k)\} \leq k(\alpha/k) = \alpha$$

であるから $\cup_i C_i(\alpha/k)$ は帰無仮説 H の有意水準 α の棄却域を与える．つまり，各検定 H_i を有意水準 α/k で行えば，その結果は同時帰無仮説 H の有意水準 α の検定として使える．これは Bonferroni の検定の多重比較補正と呼ばれる方法である．これは検定の多重比較補正の基本となる考え方であるが，当然 k が少し大きくなると α/k が小さくなりすぎ，実際上無意味となる．

検定の多重比較補正法としては，より精緻な方法が幾つも提案されているが，いずれも個別の検定をある有意水準 α_i で行えば，それが結果として同時帰無仮説 H の有意水準

α の検定となるような補正有意水準 $\alpha_1, \alpha_2, \ldots, \alpha_k$ を与える (一般に $\alpha_i \leq \alpha$). 同じことを p 値の言葉で表現すると，本来の p 値 $p_1, p_2, \ldots, p_k$ の多重比較補正値と呼ばれる数値 $p_1^*, p_2^*, \ldots, p_k^*$ (一般に $p_i^* \geq p_i$) を定め，ある i で $p_i^* \leq \alpha$ であれば，同時帰無仮説 H が有意水準 α で棄却されたことになるようにする．例えば Bonferroni の多重比較補正法に対しては，p 値の多重比較補正を $p_i^* = kp_i$ と取れば良い:

$$p_i^* \leq \alpha \Longleftrightarrow p_i \leq \alpha/k \Longleftrightarrow H_i \text{ を有意水準 } \alpha/k \text{ で棄却.}$$

3.3.1 p 値のベクトルの多重比較補正 p.adjust()

`p.adjust()` は p 値のベクトルを与えて，幾つかの方法を用いて多重比較補正された p 値のベクトルを返す.

書式：
```
p.adjust(p, method=p.adjust.methods, n=length(p))
p.adjust.methods  # 可能な手法名の文字ベクトル
```
引数：

p	p 値のベクトル
method	補正方法．以下の文字列のいずれか "holm","hochberg","hommel","bonferroni","BH","BY","fdr","none"
n	比較の個数

返り値：　補正された p 値のベクトル

補正方法の一つは Bonferroni 補正 ("bonferroni") で，p 値は比較回数倍される．それほど保守的でない補正法がそれぞれ Holm ("holm")，Hochberg ("hochberg")，Hommel ("hommel") そして Benjamini & Hochberg ("fdr") で与えられる．無補正オプション ("none") もある．補正法をオプションとして持ち，それを引き数 p.adjust に指定する必要がある手法の便宜用に，手法名の文字列ベクトル p.adjust.methods がある．

最初の 4 つの方法はファミリ単位の誤差率を強力にコントロールするように作られている．Bonferroni 補正はそのままでは，如何なる場合も使用可能な Holm 法より劣るので使用する理由は無い．Hochberg と Hommel の方法は，検定が独立か，非負の連関を持つ (Sarkar, Sarkar & Chang) という仮定下で有効である．Hommel 法は Hochberg 法よりもより強力であるが，差異は普通小さく，Hochberg 法は p 値をより高速に計算する．Benjamini & Hochberg の "fdr" 法は棄却された仮説中の誤った発見の平均の割合である「誤発見率 (false discovery rate)」をコントロールする．誤発見率はファミリ単位の誤判別率よりも厳しい条件であり，従って Benjamini & Hochberg 法は他の手法よりもより強力な方法である．

関連：stats パッケージ中の pairwise.t.test() 等の pairwise.*() 関数．

```
> p.adjust.methods                                          # 可能な手法名の文字ベクトル
[1] "holm"  "hochberg"  "hommel"  "bonferroni"  "fdr"  "none"
> x <- rnorm(10, m=c(rep(0,5),rep(3,5)))                    # 平均 0,3 の正規乱数 5 個ずつ
> p <- 2*pnorm(-abs(x))                                     # テスト用の p 値ベクトル
> round(p, 3)
 [1] 0.165 0.282 0.621 0.435 0.696 0.040 0.001 0.102 0.001 0.010
> round(p.adjust(p), 3)                                     # 既定の Holm 法による補正
 [1] 0.825 1.000 1.000 1.000 1.000 0.280 0.006 0.615 0.006 0.084
> round(p.adjust(p,"bonferroni"), 3)                        # Bonferoni 法による補正
 [1] 1.000 1.000 1.000 1.000 1.000 0.399 0.006 1.000 0.007 0.105
> round(p.adjust(p,"fdr"), 3)                       # false discovery rate 法による補正
 [1] 0.275 0.402 0.690 0.544 0.696 0.100 0.003 0.205 0.003 0.035
```

3.3.2　多重比較補正を伴う対毎の比率の比較 pairwise.prop.test()

pairwise.prop.test() は，多重比較補正を伴う対毎の比率の比較を行う．

書式：
```
pairwise.prop.test(x, n, p.adjust.method=p.adjust.methods, ...)
```

引数：

x	成功回数のベクトル．または，それぞれ成功・失敗回数を 2 つの列に持つ行列
n	試行回数のベクトル．もし x が行列なら無視される
p.adjust.method	p 値を補正する方法 (p.adjust() を参照)
...	prop.test() に引き渡される追加引き数

返り値： クラス "pairwise.htest" を持つオブジェクト

関連：prop.test(), p.adjust().

```
# 患者 4 グループ中の喫煙者数のデータ．各グループ対毎に喫煙者の割合が同一という帰無仮説の検定
# prpo.test() の参考例を参照 (4 グループ込みの比率の同一性の検定)
> smokers <- c(83, 90, 129, 70)
> patients <- c(86, 93, 136, 82)
# 各グループ対毎の喫煙者数比率の同一性を検定し，p 値を Holm 法で多重比較補正
> pairwise.prop.test(smokers, patients)
$method
[1] "Pairwise comparison of proportions"
$data.name
[1] "smokers out of patients"

# 対毎の比較の補正済み p 値．第 2,4 グループ間の補正 p 値が 0.1 未満なので帰無仮説は 10% 有意
$p.value
          1          2         3
2 1.0000000         NA        NA
3 1.0000000 1.00000000        NA
4 0.1185648 0.09321728 0.1237680
$p.adjust.method                                    # Holm 法で多重比較補正 (既定)
[1] "holm"
attr(,"class")
[1] "pairwise.htest"
(以下省略)

> pairwise.prop.test(smokers, patients, p.adjust.method="fdr") # "fdr"法で多重比較補正
        Pairwise comparisons using Pairwise comparison of proportions
data:  smokers out of patients
  1     2     3
2 1.000 -     -
3 0.965 0.965 -
4 0.062 0.062 0.062
P value adjustment method: fdr
(以下省略)
```

3.3.3 多重比較補正を伴うグループ水準間の対毎の t 検定による比較 pairwise.t.test()

pairwise.t.test() は，多重比較補正を伴うグループ水準間の，対毎の t 検定による比較を行う．

書式： pairwise.t.test(x, g, p.adjust.method = p.adjust.methods, pool.sd = TRUE, ...)

引数：

x	応答ベクトル
g	グルーピング用のベクトルか因子
p.adjust.method	p 値を補正する方法 (p.adjust() を参照)
pool.sd	プールした標準偏差の使用を許すか許さないか

...　t.test() に引き渡される追加引き数
返り値：　クラス "pairwise.htest" を持つオブジェクト

関連：t.test(), p.adjust().

```
# NY 市の毎日の大気観測値データ airquality. 各月対毎にオゾン量平均が等しいという帰無仮説の t 検定
> attach(airquality)                                  # 成分を名前で参照できるようにする
> Month <- factor(Month, labels = month.abb[5:9]) # 月数を省略文字名で因子化
> pairwise.t.test(Ozone, Month)
$method                                               # プールした標準偏差を用いた t 検定
[1] "t tests with pooled SD"
$data.name
[1] "Ozone and Month"
# 多重比較補正済み p 値. (Jul,Sep),(Aug,Sep) 間の補正 p 値が 0.005 より小さく，0.5% 有意
$p.value
             May        Jun          Jul          Aug
Jun 1.0000000000         NA           NA           NA
Jul 0.0002638036 0.05112741           NA           NA
Aug 0.0001949061 0.04987333 1.000000000           NA
Sep 1.0000000000 1.00000000 0.004878798 0.003878108
$p.adjust.method                                      # Holm 法で多重比較補正法 (既定)
[1] "holm"
attr(,"class")
[1] "pairwise.htest"
```

```
# Bonferroni 法で多重比較補正すると (Aug,Sep) 間のみで p 値が 0.005 以下
> pairwise.t.test(Ozone, Month, p.adj = "bonf")
$method                                               # プールした標準偏差を用いた t 検定
[1] "t tests with pooled SD"
$data.name
[1] "Ozone and Month"
$p.value
             May        Jun          Jul          Aug
Jun 1.0000000000         NA           NA           NA
Jul 0.0002931151 0.10225483           NA           NA
Aug 0.0001949061 0.08312222 1.000000000           NA
Sep 1.0000000000 1.00000000 0.006969712 0.004847635
$p.adjust.method                                      # Bonferroni 法で多重比較補正
[1] "bonferroni"
attr(,"class")
[1] "pairwise.htest"
```

```
> pairwise.t.test(Ozone, Month, pool.sd = FALSE)
$method                                               # プールした標準偏差を用いない t 検定
[1] "t tests with non-pooled SD"
$data.name
[1] "Ozone and Month"
$p.value                                              # 今度は 1% 有意という結果
             May        Jun          Jul         Aug
Jun 1.0000000000         NA           NA          NA
Jul 0.0002646679 0.01527179           NA          NA
Aug 0.0019517090 0.02134659 1.000000000          NA
Sep 0.8632062149 1.00000000 0.005888826 0.01720974
$p.adjust.method                                      # Holm 法で多重比較補正
[1] "holm"
attr(,"class")
[1] "pairwise.htest"
> detach()                                            # 成分を消去
```

3.3.4　Wilcoxon のランク和検定に対する多重比較補正 pairwise.wilcox.test()

pairwise.wilcox.test() はグループ水準の各組合せ毎に Wilcoxon のランク和検定を実施し，p 値に対する多重比較補正を行う．

書式： `pairwise.wilcox.test(x,g,p.adjust.method=p.adjust.methods,...)`

引数：

x　応答ベクトル

g　グルーピング用ベクトル，もしくは因子

`p.adjust.method`　p 値を補正する方法 (`p.adjust()` を参照)

`...`　`wilcox.test()` に引き渡される追加引き数

返り値：　クラス属性 `"pairwise.htest"` を持つオブジェクト

関連：`wilcox.test()`, `p.adjust()`.

```
# NY 市の大気観測値データ airquality 使用．月の組合せ毎に wilcox.test() を適用し多重比較補正
> Month <- factor(Month, labels = month.abb[5:9]) # 5,6,7,8,9 月名を因子に変換
> pairwise.wilcox.test(Ozone, Month)
$method
[1] "Wilcoxon rank sum test"
$data.name
[1] "Ozone and Month"

# 多重比較補正された p 値．(May,Jul) 間で 0.0003 以下なので 0.05% 有意
$p.value
            May         Jun          Jul         Aug
Jun 0.5775049432          NA           NA          NA
Jul 0.0002996390 0.08481748           NA          NA
Aug 0.0010872705 0.12953880 1.000000000          NA
Sep 0.4743723336 1.00000000 0.005954083 0.02273569
$p.adjust.method                                  # Holm 法で補正 (既定)
[1] "holm"
attr(,"class")
[1] "pairwise.htest"
Warning messages:                                 # データにタイがあるので警告がでる (省略)

# Bonferroni 法で多重比較．再び 0.05% 有意という結果
> pairwise.wilcox.test(Ozone, Month, p.adj = "bonf")
$method
[1] "Wilcoxon rank sum test"
$data.name
[1] "Ozone and Month"
$p.value
             May       Jun          Jul         Aug
Jun 1.0000000000        NA           NA          NA
Jul 0.0002996390 0.1413625           NA          NA
Aug 0.0012080783 0.2590776 1.000000000          NA
Sep 1.0000000000 1.0000000 0.007442604 0.03247955
$p.adjust.method
[1] "bonferroni"
attr(,"class")
[1] "pairwise.htest"
Warning messages:                                 # データにタイがあるので警告がでる (省略)
> detach()
```

3.4 検定の検出力の計算

帰無仮説 H の有意水準 α の棄却域を C とする．この検定の検出力 (power) とは，対立仮説 A の下での確率 $p(A) = \boldsymbol{P}\{T \notin C\}$ (対立仮説が正しいとき，それを正しく判定する確率) である．複合対立仮説であれば，検出力は各対立仮説毎に異なった値を取る．与えられた有意水準に対しては，検出力のより大きな検定が好ましいが，両仮説 H,A に対する確率分布が近ければ検出力は小さめにならざるを得ない．従って，両仮説 (に対応

するパラメータ値) がどの程度離れていれば十分な検出力が得られるかが問題になる．また，検出力は一般にデータ数とともに大きくなるので，指定した検出力を得るのに必要なデータ数の見積りも問題になる．

3.4.1　一元配置分散分析検定の検出力計算 power.anova.test()

power.anova.test() は一元配置分散分析検定の検出力 (パワー) を計算する．または目的の検出力を得ることができるパラメータ値を計算する．

書式：

```
power.anova.test(groups = NULL, n = NULL, between.var = NULL,
                 within.var = NULL, sig.level = 0.05, power = NULL)
```

引数：

groups　グループ数
n　　(グループ毎の) 観測値数
between.var　グループ間分散
within.var　グループ内分散
sig.level　信頼係数 (タイプ I エラー確率)
power　検定の検出力 (1 からタイプ II エラー確率を引いたもの)

返り値：　クラス属性 "power.htest" を持つオブジェクトで，method と note 成分が加えられた，与えられた引き数 (計算されたものを含む) を成分に持つリスト

パラメータ groups, n, between.var, power, within.var そして sig.level の内丁度一つが値 NULL と指定される必要があり，そのパラメータは他のパラメータから計算される．sig.level は NULL でない既定値を持っており，これを計算したければ明示的に NULL と指定する必要があることを注意しよう．

注意：未知パラメータを得るために検出力方程式を解くために uniroot() が使われ，したがって，特に不正な引き数が与えられたとき根をくくり出せないという，エラーを得るかも知れない．

関連：anova(), lm(), uniroot().

```
> power.anova.test(groups=4, n=5, between.var=1, within.var=3)
    Balanced one-way analysis of variance power calculation
        groups = 4                                  # グループ数
             n = 5                                  # 指定されたグループ毎の標本数
   between.var = 1                                  # 指定された級間分散値
    within.var = 3                                  # 指定された級内分散値
     sig.level = 0.05                               # 指定された有意水準
         power = 0.3535594                          # 対応する検出力
NOTE: n is number in each group
```

```
# 一元配置分散分析検定で指定検出力を得るのに必要な標本数
> power.anova.test(groups=4, between.var=1, within.var=3,
                   power=.80)
    Balanced one-way analysis of variance power calculation
        groups = 4
             n = 11.92613                           # 必要とされるグループ毎の標本数
   between.var = 1
```

```
    within.var = 3
     sig.level = 0.05
         power = 0.8                              # 指定検出力
NOTE: n is number in each group
```

```
> groupmeans <- c(120, 130, 140, 150)
> power.anova.test(groups = length(groupmeans),   # 群平均に付いて予め知っている時
                   between.var=var(groupmeans),
                   within.var=500, power=.90)
    Balanced one-way analysis of variance power calculation
        groups = 4
             n = 15.18834                         # 指定されたグループ毎の標本数
   between.var = 166.6667
    within.var = 500
     sig.level = 0.05
         power = 0.9                              # 指定検出力
NOTE: n is number in each group
```

3.4.2 比率の検定に対する検出力 power.prop.test()

power.prop.test() は比率の検定に対する検出力を計算する．または目標の検出力を得るために必要なパラメータを計算する．

書式：

```
power.prop.test(n=NULL, p1=NULL, p2=NULL, sig.level=0.05,
                power=NULL, alternative=c("two.sided","one.sided"),
                strict=FALSE)
```

引数：

n	(グループ毎の) 観測値数
p1	一つのグループの確率
p2	もう一つのグループの確率
sig.level	信頼係数 (タイプ I エラー確率)
power	検定の検出力 (1 からタイプ II エラー確率を引いたもの)
alternative	片側・両側検定
strict	両側検定の際，厳密な解釈を行う

返り値： クラス属性 "power.htest" を持つオブジェクトで，method と note 成分が加えられた，与えられた引き数 (計算されたものを含む) を成分に持つリスト

パラメータ n, p1, p2, power そして sig.level の内丁度一つが値 NULL と指定される必要があり，そのパラメータは他のパラメータから計算される．sig.level は NULL でない既定値を持っており，これを計算したければ明示的に NULL と指定する必要があることを注意しよう．もし strict=TRUE ならば，両側検定の場合，検出力は真の効果の反対方向への棄却確率を含む．これがないと真の差が 0 ならば検出力は信頼係数の半分になるであろう．

注意：未知パラメータを得るために検出力方程式を解くために uniroot() が使われる．したがって，特に不正な引き数が与えられたとき，根をくくり出せないというエラーを得るかも知れない．もしどちらかが計算されると p1 < p2 となるが，両方が指定されると

必ずしもこうならない.

関連：prop.test(), uniroot().

```
# n=50,p1=.50,p2=.75,sig.level=0.05 なら検出力は 0.74
> power.prop.test(n=50, p1=.50, p2=.75)
     Two-sample comparison of proportions power calculation
              n = 50
             p1 = 0.5
             p2 = 0.75
      sig.level = 0.05                          # 指定有意水準
          power = 0.7401659                     # 計算された検出力
    alternative = two.sided                     # 両側対立仮説
 NOTE: n is number in *each* group
```

```
# p1=0.5,p2=.75,sig.level=0.05 の時，power=0.9 であるためには n=77 が必要
> power.prop.test(p1=.50, p2=.75, power=.90)
     Two-sample comparison of proportions power calculation
              n = 76.70693                      # 計算された必要観測値数
             p1 = 0.5
             p2 = 0.75
      sig.level = 0.05
          power = 0.9
    alternative = two.sided
 NOTE: n is number in *each* group
```

```
# n=50,p1=0.5,sig.level=0.05 の時，power=0.9 であるためには p2=0.80 が必要
> power.prop.test(n=50, p1=.5, power=.90)
     Two-sample comparison of proportions power calculation
              n = 50
             p1 = 0.5
             p2 = 0.8026141                    # 計算された p2 値
      sig.level = 0.05
          power = 0.9
    alternative = two.sided
 NOTE: n is number in *each* group
```

3.4.3　t 検定の検出力の計算 power.t.test()

power.t.test() は一・二標本の t 検定を行う．または，目標の検出力を得るために必要なパラメータの決定を行う．

書式：

```
power.t.test(n=NULL, delta=NULL, sd=1, sig.level=0.05, power=NULL,
             type=c("two.sample", "one.sample", "paired"),
             alternative=c("two.sided", "one.sided"), strict=FALSE)
```

引数：

n	(グループ毎の) 観測値数
delta	平均の真の差
sd	標準偏差
sig.level	信頼係数 (タイプ I エラー確率)
power	検定の検出力 (1 からタイプ II エラー確率を引いたもの)
type	t 検定のタイプ
alternative	片側・両側検定
strict	両側検定の際，厳密な解釈を行う

返り値： クラス "power.htest" のオブジェクトで，method と note 成分が加えられた，与えられた引き数 (計算されたものを含む) を成分に持つリスト

パラメータ n, delta, power, sd そして sig.level の内丁度一つが値 NULL と指定される必要があり，そのパラメータは他のパラメータから計算される．最後の二つは NULL でない既定値を持っており，これらを計算したければ明示的に NULL と指定する必要があることを注意しよう．もし strict=TRUE ならば，両側検定の場合，検出力は真の効果の反対方向への棄却確率を含む．これがないと真の差が 0 ならば検出力は信頼係数の半分になるであろう．

注意：未知パラメータを得るために検出力方程式を解くために uniroot() が使われ，したがって，特に不正な引き数が与えられたとき根をくくり出せないという，エラーを得るかも知れない．

関連： t.test(), uniroot().

```
> power.t.test(n=20, delta=1)                    # 観測値数 20 の時の検出力
     Two-sample t test power calculation          # 二標本 t 検定
              n = 20                              # 二標本のそれぞれのデータ数
          delta = 1                               # 真の平均差
             sd = 1                               # 二標本の標準偏差
      sig.level = 0.05                            # 第一種の誤り確率
          power = 0.8689528                       # 計算された検出力
    alternative = two.sided                       # 両側対立仮説
 NOTE: n is number in *each* group

> power.t.test(power=.90, delta=1)               # 検出力 0.90 を得るために必要な観測値数
     Two-sample t test power calculation          # 二標本 t 検定
              n = 22.02110                        # 必要な両群の観測値数
          delta = 1
             sd = 1
      sig.level = 0.05
          power = 0.9
    alternative = two.sided
 NOTE: n is number in *each* group

> power.t.test(power=.90, delta=1, alt="one.sided")
     Two-sample t test power calculation          # 二標本 t 検定
              n = 17.84713                        # 必要な両群の観測値数
          delta = 1
             sd = 1
      sig.level = 0.05
          power = 0.9
    alternative = one.sided                       # 片側対立仮説を指定
 NOTE: n is number in *each* group

> power.t.test(n=50, delta=1, sd=1, sig.level=0.05, power=NULL,
               type="one.sample", alternative="one.sided")
     One-sample t test power calculation          # 一標本 t 検定
              n = 50
          delta = 1
             sd = 1
      sig.level = 0.05
          power = 1                               # そのときの検出力
    alternative = one.sided                       # 片側対立仮説を指定
```

3.5 その他，モンテカルロ法による検定検出力の計算

非典型的な状況で，検定の棄却域や検出力を求める簡便な方法がモンテカルロ法である．以下では，それぞれ二項分布 $B(n,p)$, $B(m,q)$ に従う二組の独立標本 N 個を有意水

準 $1-\alpha$，帰無仮説 $p=q$，両側対立仮説 $p \neq q$ で検定する例である．モンテカルロ法では，繰り返し数が異なれば異なった結果 (使った乱数系列にも依存) を得るので，繰り返し数を変えて何度も確かめてみる方が賢明である．また，計算時間は当然余計にかかるが，あれこれ文献を探すよりは早い．

```
> foo <- function(N, n, p, m, q, alpha) {
           X <- rbinom(N,n,p)                      # B(n,p) に従う標本 N 個
           Y <- rbinom(N,m,q)                      # B(m,q) に従う標本 N 個
           phat <- (X+Y)/(n+m)                     # 帰無仮説 p=q 下での共通比率の推定値
           S <- sqrt((1/n+1/m)*phat*(1-phat))      # その時の X/n-Y/m の標準偏差推定値
           z <- qnorm(1-alpha/2)                   # 両側検定棄却限界値
           mean(abs(X/n - Y/m)/S > z)              # N 回の検定で棄却される標本比率
         }

> bar <- function(n) { # m=0.2*n のケース．N=1e7 で実行
           x <- foo(1e7, n, p, ceiling(0.2*n), q, alpha)
           cat(n, ceiling(0.2*n), x, fill=TRUE) }
> p=0.3; q=0.5; alpha=0.05; beta <- 0.8

> set.seed(1)                                     # 再現性のために乱数種を指定
> sapply(260:300, bar)                            # n=260:300 で実行
260 52 0.7875916
261 53 0.7932245
262 53 0.7959385
263 53 0.7920496
264 53 0.7943363
265 53 0.7971905
266 54 0.8025416                                  # 標本検出力が初めて 0.8 を越える標本数
267 54 0.799305
268 54 0.8011001
(以下省略)

> bar <- function(n) {                            # 繰り返し数 1e6 で再実行
           x <- foo(1e6, n, p, ceiling(0.2*n), q, alpha)
           cat(n, ceiling(0.2*n), x, fill=TRUE) }

> set.seed(1)
> sapply(260:300, bar)
260 52 0.787136
261 53 0.793089
262 53 0.795644
263 53 0.792638
264 53 0.794402
265 53 0.797296
266 54 0.802223                                   # <- 標本検出力が初めて 0.8 を越える標本数
267 54 0.798358
268 54 0.801
(以下省略)
```

参考：パッケージ Hmisc には成功確率とサイズが異なる二組の二項分布データの平均の差の検定の検出力を計算する関数 bpower() がある．これによれば必要サイズは 266 と 54 である．

```
> library(Hmisc)                                  # パッケージ Hmisc を読み込む
> baz <- function(N) {                            # 標本検出力が初めて 80% を越える標本数
          power <- bpower(p1=P1, p2=P2,
                          n1=N, n2=ceiling(Nratio*N), alpha=Alpha)
          if (power >= Power)                     # 要求パワーを越えた時点でストップ
            {cat(N, ceiling(Nratio*N), power, fill=TRUE); stop()}
        }
> P1 <- 0.3; P2 <- 0.5; Nratio <- 0.2; Alpha <- 0.05; Power <- 0.8
> sapply(100:10000, baz))                  # N を変えながら検出力が 0.8 を越えるまで繰り返し
266 54 0.8003697                           # ループ中断によるエラーメッセージ．無視可能
```

第 4 章
時系列解析

R の基本パッケージ (base パッケージおよび付属パッケージ stats) には時系列データを処理し，解析する基本的な関数が用意されている．更に CRAN にはより高度な時系列解析用のアドオンパッケージがいくつも[*1] ある．

時間の経過の中で観測されるデータは，データの値 x_i とともに，しばしば観測された時間 t_i の情報が重要なことがある．したがってデータは，観測時間 $\{t_i\}$ と対応する観測値 $\{x_i\}$ という対で表現される．こうしたデータの中で，特に $\{t_i\}$ が秒・分・時間・日・週・月・四半期・年等の等間隔の間隔であるものを (離散) 時系列データ (discrete time series data) と呼ぶ．時系列データを解析する統計理論を時系列解析 (time series analysis) と総称し，その興味の中心は異なった時間での観測値間に想定される従属性構造のモデル化にある．極端な場合として $\{x_i\}$ が互いに独立な確率変数列の実現値とみなすことができれば，時間情報は無意味になる．時系列解析は，従って，本質的に従属性を持つ確率変数列の理論となり，「独立・同分布データの解析」という，統計学の基本的枠組から外れ，その理論・解析は一層困難となり，R 等の高機能の統計解析ソフトの重要性が一層ます．

以下では，base および stats パッケージ中の時系列解析用の関数を解説する．R の時系列解析用の関数は，時系列オブジェクト (time-series object) という，時間と対応する観測値の対，および観測期間・周期等の情報を持つ特殊なオブジェクトを扱う．こうしたオブジェクトは "ts" というクラスとされ，オブジェクト自身のクラス属性として記録されている．時系列解析用の関数は，与えられた引数がクラス "ts" であることをまず確認し，それに基づいて処理を行うようになっている．

時系列オブジェクトは (等間隔) 観測時間情報を持つため，時間に関する以下のような幾つかの特殊な情報を持つ:

[*1] CRAN にある Task View と呼ばれる各種パッケージをテーマ別にまとめたもののうち，テーマ Finance と Econometrics を参照．

自然な時間単位　年，月，週，一時間等
観測開始時間　start
観測終了時間　end
周波数　frequency．単位時間内の観測時間の数．月別データなら，自然な時間単位「年」に対する周波数は 12
サンプリング比率　deltat．自然な時間単位に対するサンプリング間隔を表す比率．月別データなら，自然な時間単位「年」に対する deltat は 1/12．周波数とサンプリング比率どちらか一方を与えれば良い
周期　cycle．各データの観測時間情報を表す自然な時間単位とその中での周波数を表す対．例えば年と月 (1993,5)，年と第 2 四半期 (2002,Qtr2)，月と日 (12,13)，週と曜日 (43,Fri) 等

注意：時系列データに固有の特徴として，系統的にデータ値が欠損することがある．例えば，株価等の経済時系列は，土日・祭日はデータが本来無い．
注意：R における多変量時系列は，観測値がベクトルの時系列ではなく，時間情報が同一の複数の一変量時系列の集まりである．

4.1　時系列オブジェクトの生成と処理

4.1.1　時系列オブジェクトを生成する，ts(), as.ts(), is.ts()

クラス "ts" のオブジェクトをベクトルもしくは行列から生成する基本関数は ts() である．時系列データには，一年およびそれを 12 分割した月，一週間とそれを 7 分割した日，等の自然な時間単位と，それの分割単位が備わっていることが多い．

書式：

```
ts(data = NA, start = 1, end = numeric(0), frequency = 1,
   deltat = 1, ts.eps = getOption("ts.eps"), class = , names = )
as.ts(x)
is.ts(x)
```

引数：
data　時系列データの引き続く観測値である数値ベクトル，もしくは複数の観測値を並べた数値行列．もしこれが，データフレームなら，まず数値行列に強制変換される．観測値の一部は欠損値 NA であっても良い
start　data の観測開始時間．単一の数値か，2 つの数値からなるベクトル (自然な時間単位と，その単位内での観測開始時間を指定する)
end　観測完了時間． start と同様の意味を持つ
frequency　周波数，単位時間内の観測値の数
deltat　自然な時間単位に対するサンプリング間隔を表す比率．例えば，年単位に対する月別データなら deltat=1/12．引き数 deltat と frequency は一方だけを指定できる
ts.eps　時系列の比較用許容度．絶対値がこれより小さな周波数は等しいとされる

class	結果のクラス名．もし NULL ならクラス名は与えられない．既定値は単一の時系列に対しては "ts"，多変量時系列に対しては "mts" と "ts"
names	多変量系列中の個々の系列に与えられる名前の文字ベクトル．既定値は data の列名か，さもなければ Series 1, Series 2,... とされる
x	任意の R オブジェクト

幾つかの引き数には，省略時既定値が指定されている．R の関数の名前付き引数名の省略規則から，この場合順に最低 da, s, e, f, de, t, c, n だけで良い．

as.is() は引き数のオブジェクトを時系列オブジェクトに強制変換し，is.ts() は引き数のオブジェクトが時系列オブジェクトかどうか判定する．ts() はクラス "ts" (そして追加の属性を持つ) の時系列オブジェクトを作る．これらはベクトルか行列で，等間隔の時刻で観測されたデータを表す．行列の場合，各列は単一の (一次元) 時系列を含むとされる．時系列は少なくとも一つの観測値を含まなければならない．観測値は必ずしも数値である必要は無いが，非数値の時系列に対するサポートは極めて限られている．

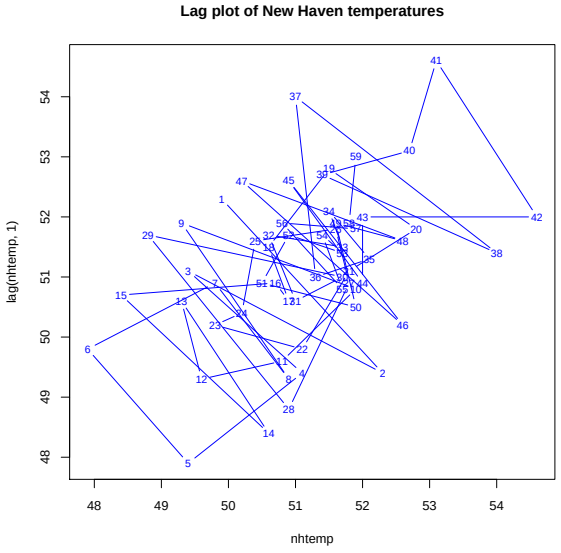
クラス "ts" は幾つかのメソッドを持つ．例えば，加減乗除の算術演算は (可能である限り) 時間軸を合わせて行われる．鈎括弧演算子による部分時系列の抽出ができる (例えば，EuStockMarkets[, "DAX"])．しかしながら，第一 (もしくは唯一の) 次元に関する部分抽出は，行列に対する部分抽出と同様，行列かベクトルを返す．引き数 frequency の値は，系列が各単位時間区間中の整数個の時刻で観測される場合に使用される．例えば，自然な単位時間が一週間の時，データが日別に観測されるなら frequency=7，自然な単位時間が一年間の時，データが月別に観測されるなら frequency=12，とする．値 4 と 12 は，例えば print メソッド (オブジェクト内容のコンソール出力) では，それぞれ四半期，月別系列と仮定される．

もしオブジェクトが時系列属性 tsp を持てば，関数 as.ts() はそれらを，時系列の始点，終点そして周波数とみなす．関数 is.ts() は時系列オブジェクトへの強制変換関数である．

```
> ts(1:10, freq=4, start=c(1959, 2))              # 1959 年の第 2 四半期から始まる時系列
     Qtr1 Qtr2 Qtr3 Qtr4                          # 年度名と四半期名が自動的に付く
1959         1    2    3
1960    4    5    6    7
1961    8    9   10

> ts(1:15, freq=12, start=c(1959, 2))             # 年と月
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1959       1   2   3   4   5   6   7   8   9  10  11
1960  12  13  14  15

> ts(1:15, freq=31, start=c(5, 19))               # 月と曜日
Time Series:
Start = c(5, 19)
End = c(6, 2)
Frequency = 31
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15

> ts(1:15, freq=7, start=c(5, 19))                # 週と曜日
Time Series:
Start = c(7, 5)
End = c(9, 5)
Frequency = 7
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```
> print(ts(1:10, freq=7, start=c(12,2)), calendar=TRUE) # 時系列の出力
   p1 p2 p3 p4 p5 p6 p7                                 # print.ts() が使われる
12     1  2  3  4  5  6
13  7  8  9 10

# 正規乱数の累積和からなる時系列 (1954 年 7 月から始まる)
gnp <- ts(cumsum(1 + round(rnorm(100), 2)), start = c(1954, 7), freq = 12)
plot(gnp)                          # plot() による時系列 gnop のプロット. plot.ts() が使われる

# 多変量時系列 (引数名の省略記法に注意)
> z <- ts(matrix(rnorm(300), 100, 3), s=c(1961,1), f=12)
> attributes(z)
$dim                                                 # 行列次元属性
[1] 100   3
$dimnames                                            # 行名属性 (この場合無い)
$dimnames[[1]]
NULL
$dimnames[[2]]                                       # 列名属性 (自動的に付加されている)
[1] "Series 1" "Series 2" "Series 3"
$tsp                                                 # "tsp"属性
[1] 1961.00 1969.25   12.00
$class                                               # クラスは"ts"(時系列で)
[1] "mts" "ts"                                       # かつ"mts"(多変量)
> z                                                  # z の内容 (3 つの時系列からなる)
            Series 1    Series 2    Series 3
Jan 1961  0.31182310 -1.13148093 -0.29498438
Feb 1961  1.29541452 -1.36851379  0.29349392
Mar 1961 -0.20436604  0.14915958  0.57753954
  (途中省略)
Apr 1969 -0.73845864 -0.34393409  0.32143158
> plot(z)                                            # Series1,2,3 毎に別個にプロット
> plot(z, plot.type="single", lty=1:3)               # 線種を変えて同時にプロット
```

```
# フェイズのプロット. 組込み時系列データ nhtemp を使用
> class(nhtemp)                                      # nhtemp のクラスは"ts"
[1] "ts"
> is.ts(nhtemp)                                      # nhtemp は確かに時系列オブジェクト
[1] TRUE

# 引き続く 2 年分の気温の対の散布図 (以下の図を参照)
> plot(nhtemp, lag(nhtemp, 1), cex = .8, col="blue",
       main = "Lag plot of New Haven temperatures")
```



時系列のラグプロット
横軸は元の値，縦軸は一つ後の値

4.1.2 時系列オブジェクトに対する数値演算

時系列オブジェクトに対しては，スカラーによる算術演算，関数の適用等が可能であり，結果は同じ時間範囲を持つ時系列オブジェクトになる．行列風の鈎括弧演算子による部分抽出ができる．また二つの時系列同士の算術演算も可能で，共通時間範囲で各時間毎に演算を行った結果からなる時系列になる．

```
# 3次元月別時系列オブジェクトの定義 (順に名前 A,B,C を付ける)
> z <- ts(matrix(1:15, 5, 3), s=c(1961,1), f=12, n=c("A","B","C"))
> z
         A  B  C
Jan 1961 1  6 11
Feb 1961 2  7 12
Mar 1961 3  8 13
Apr 1961 4  9 14
May 1961 5 10 15

> z[,"A"]                                          # 第一時系列の抽出. z[,1] でも良い
     Jan Feb Mar Apr May
1961   1   2   3   4   5

> 2*z[,"A"]+1                                      # スカラーによる算術演算
     Jan Feb Mar Apr May
1961   3   5   7   9  11

> exp(z[,"A"])                                     # 関数の適用
           Jan        Feb        Mar        Apr        May
1961  2.718282   7.389056  20.085537  54.598150 148.413159

> z[,"A"]^2                                        # 2乗
     Jan Feb Mar Apr May
1961   1   4   9  16  25

> z[,"A"] + z[,"B"]                                # 時系列同士の和
     Jan Feb Mar Apr May
1961   7   9  11  13  15

> z[,1]^z[,2]                                      # 時系列の時系列巾乗
         Jan     Feb     Mar     Apr     May
1961       1     128    6561  262144 9765625

> lag(z[,"A"],1)                                   # lag() 関数により一時点過去にずらす
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1960                                           1
1961   2   3   4   5

> z[,"A"]+lag(z[,"A"],1)                           # 和は共通時間範囲の時系列になる
     Jan Feb Mar Apr
1961   3   5   7   9
```

4.1.3 時系列オブジェクトに対する演算 diff()

時系列オブジェクトに対して，その階差 (difference) を取る diff() 関数がある．diff() は既定のメソッドを持つ総称的関数であり，クラス "ts" と "POSIXt"[*2] に対するメソッドを持つ．欠損値 NA があれば伝播する．

書式：

[*2] R は二種類の基本日付・時刻クラス POSIXct と POSIXlt を扱う．前者は 1970 年始めからの経過秒数を表す (符号付き) 数値ベクトルであり，後者は秒・分・時・日にち・月・(1900 年以来の) 経過年数からなる名前付きベクトルリストである．両者は時間帯を表す "tzone" 属性を持つこともある (OS 依存)．POSIXt は両クラスを継承する日付・時刻クラスであり，論理比較，加減等の限られた算術演算ができる．

```
diff(x, ...)
diff(x, lag = 1, differences = 1, ...) # 既定の S3 メソッド
diff(x, lag = 1, differences = 1, ...) # クラス"POSIXt"に対する S3 メソッド
diff(x, lag = 1, differences = 1, ...) # クラス"Date"に対する S3 メソッド
```

引数：

x	時系列オブジェクト (一般に数値ベクトル，行列)
lag	整数 (ラグ，階差を取る長さ)
differences	階差の重複度 (1 階階差，2 階階差等)
...	オプションの任意追加引数

もし x が長さ n のベクトルで differences=1 ならば，計算結果は連続する階差 $\mathtt{x}[(1+\mathtt{lag}):n]-x[1:(n-\mathtt{lag})]$ となる．もし difference が 1 以上なら，アルゴリズムは x に再帰的に適用される．返り値は x よりも短いベクトルになることを注意しよう．もし x が行列なら，階差演算子は各列に個別に適用される．

```
> (z <- ts(1:12, start=c(1961, 1), freq=12))      # テスト時系列
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   1   2   3   4   5   6   7   8   9  10  11  12

> (zz <- diff(z))                                 # ラグ 1 で 1 階階差
     Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   1   1   1   1   1   1   1   1   1   1   1

> diff(z, lag=2)                                  # ラグ 2 で 1 階階差
     Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   2   2   2   2   2   2   2   2   2   2

> diff(z, 1, 2)                                   # ラグ 1 で 2 階階差
     Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   0   0   0   0   0   0   0   0   0   0
```

4.1.4　時系列オブジェクトに対する演算 diffinv()

ラグ階差関数 diff() の逆処理 (離散積分)．diffinv() はクラス "ts" に対するメソッドを持つ総称的関数であり，ベクトルと行列に対する既定動作を持つ．欠損値は処理できない．

```
diffinv(x, ...)
# 既定の S3 メソッド
diffinv(x, lag = 1, differences = 1, xi, ...)
# クラス "ts" に対する S3 メソッド
diffinv(x, lag = 1, differences = 1, xi, ...)
```

引数：

x	数値ベクトル，行列もしくは時系列オブジェクト
lag	スカラのラグパラメータ
differences	階差の重複度 (1 階階差，2 階階差等)

xi	数値ベクトル，行列もしくは時系列で，離散積分に対する初期値．もし無ければ 0 が使われる
...	他のメソッドに (から) 引き渡される引数
返り値：	x の離散積分を表す数値ベクトル，行列，もしくは時系列．欠損値は処理できない

```
> (z <- ts(1:12, start=c(1961, 1), freq=12))          # テスト時系列
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   1   2   3   4   5   6   7   8   9  10  11  12

> (zz <- diff(z))                                     # ラグ 1 で 1 階階差
     Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   1   1   1   1   1   1   1   1   1   1   1

> diff(z, lag=2)                                      # ラグ 2 で 1 階階差
     Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   2   2   2   2   2   2   2   2   2   2

> diff(z, 1, 2)                                       # ラグ 1 で 2 階階差
     Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   0   0   0   0   0   0   0   0   0   0

> diffinv(zz)                                         # 既定初期値 0 で逆階差
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   0   1   2   3   4   5   6   7   8   9  10  11

> diffinv(zz, xi=z[1])                                # 初期値時系列を与え，完全復元
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   1   2   3   4   5   6   7   8   9  10  11  12

> diffinv(zz, xi=ts(1,c(1961,1)))                     # 同じこと
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   1   2   3   4   5   6   7   8   9  10  11  12

> z2 <- diff(z, 1, 2)                                 # ラグ 1 の 2 階階差

> diffinv(z2, 1, 2, xi=z[1:2])                        # 二つの初期値による 2 階逆階差
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1961   1   2   3   4   5   6   7   8   9  10  11  12
```

4.1.5 時系列の合併と共通部分，ts.union(), ts.intersection()

周波数が同じ複数の時系列オブジェクトを，時間範囲の合併と共通部分上の多変量時系列に変換する関数 ts.union(), ts.intersection() がある．ts.union() は該当観測値が無ければ欠損値 NA で埋める．ts.intersect() は全ての時系列に共通の期間に制限した時系列を与える．

書式：

```
ts.intersect(..., dframe = FALSE)
ts.union(..., dframe = FALSE)
```

引数：

... 二つ以上の時系列，もしくは多変量時系列．または，時系列に強制変換できるオブジェクト

dframe 論理値．もし TRUE なら結果をデータフレームとして返す

返り値： 返り値は dframe=FALSE なら時系列オブジェクトで，さもなければデータフレーム

```
# 組込みデータ UKLungDeaths は英国の 1974-1979 に渡る肺ガン，肺気腫もしくは喘息による死亡者数時系列
# 両性，男性，女性の合併時間範囲（この場合同一）における，男性，女性観測値からなる 2 変量時系列
> ts.union(mdeaths, fdeaths)
         mdeaths fdeaths
Jan 1974    2134     901
Feb 1974    1863     689
(途中省略)
Dec 1979    1341     574
> cbind(mdeaths, fdeaths)                          # 関数 cbind() でも同じことができる
         mdeaths fdeaths
Jan 1974    2134     901
Feb 1974    1863     689
(途中省略)
Dec 1979    1341     574

# 男性データの 1976 年度と，女性データの 1974-1978 年度分の共通時間範囲における 2 変量時系列
> ts.intersect(window(mdeaths, 1976), window(fdeaths, 1974, 1978))
         window(mdeaths, 1976) window(fdeaths, 1974, 1978)
Jan 1976                  2020                         767
Feb 1976                  2750                        1141
(途中省略)
Jan 1978                  2019                         796
```

```
# 組込み時系列 BJsales は BJsales(売上高) と BJsales.lead(景気動向先行指数) の
# 二つの時系列オブジェクト
> BJsales.lead
Time Series: Start = 1 End = 150 Frequency = 1
  [1] 10.01 10.07 10.32  9.75 10.33 10.13 10.36 10.32 10.13 10.16 10.58 10.62
 (途中省略)
[145] 13.25 13.50 13.58 13.51 13.77 13.40

> lag(BJsales.lead, -3)                              # 時間を未来に 3 だけずらす
Time Series: Start = 4 End = 153 Frequency = 1
[1] 10.01 10.07 10.32  9.75 10.33 10.13 10.36 10.32 10.13 10.16 10.58 10.62
(途中省略)
[145] 13.25 13.50 13.58 13.51 13.77 13.40

> ts.intersect(sales1, lead3 = lag(BJsales.lead, -3)) # 共通時間範囲上の 2 変量時系列になる
Time Series: Start = 4 End = 150 Frequency = 1
    sales1.BJsales sales1.lead lead3
  4          198.9        9.75 10.01
  5          199.0       10.33 10.07
 (途中省略)
150          262.7       13.40 13.58
```

4.1.6 時系列に関する情報を得る，start()，end()，frequency()，cycle()，tsp()

時系列オブジェクトに関する情報を得る関数には start()，end()，frequency()，time()，cycle()，tsp() 等がある．時系列オブジェクトは付加情報を持つベクトルもしくは行列であるから，長さ関数 length() 関数や，次元関数 dim() を用いて，サイズを知ることができる．

書式：

```
start(x, ...), end(x, ...), time(x, ...)
# 既定の S3 メソッド
time(x, offset=0, ...), cycle(x, ...), frequency(x, ...),
deltat(x, ...), tsp(x), tsp(x) <- value, hasTsp(x)
```

引数：

x	一変量もしくは多変量時系列．ベクトルや行列でも良い
offset	時間単位で観測がいつ開始されたかを指示するのに使える．既定の 0 では単位の最初，0.5 では中間，1 では区間の最後
value	長さ 3 の数値ベクトルもしくは NULL
...	将来のメソッドのための追加引数

start() と end() はバージョン 2 の S との互換性のためだけに存在する．引き数 x の tsp 属性に基づき元の時間単位での開始・終了時刻を抜きだす．例えば月毎の時系列ならば 1995 年 7 月は c(1995,7) になる．一般に frequency=f の時系列では時刻 n+i/f は c(n,i) と表現される．従って frequendy が与えられていることが大前提である．

frequency() は周波数 (1 時間単位が幾つに分割されているか) を返す．cycle() は各時間単位内での周期 (何番目の周波数か) を返す．deltat() は基本時間単位に対する観測間隔の比率で 1/frequnecy() の関係がある．

tsp() はバージョン 2 の S との互換性のために存在する．引数の時系列属性 tsp，もしくは NULL を返す．tsp(x) <- value は tsp 属性を付与する．tsp(x) <- NULL とすれば tsp 属性とクラス属性 "ts" (もしくは "mts") を取り除く．

hasTsp(x) は，もしオブジェクト x が tsp 属性を持たなければ，それを与える．

```
> z <- ts(1:5, start=c(1961,1), freq=4)          # 四半期時系列
> z
     Qtr1 Qtr2 Qtr3 Qtr4
1961    1    2    3    4
1962    5

> start(z)                                       # 開始時期
[1] 1961    1

> end(z)                                         # 終了時期
[1] 1962    1

> frequency(z)                                   # 周波数
[1] 4

> time(z)                                        # 観測時間を基本時間単位との比率で与える
        Qtr1     Qtr2     Qtr3     Qtr4
```

```
1961 1961.00 1961.25 1961.50 1961.75
1962 1962.00

> deltat(z)                                          # 基本時間単位に対する観測間隔
[1] 0.25

> cycle(z)                                           # 各観測値の周期
     Qtr1 Qtr2 Qtr3 Qtr4
1961    1    2    3    4
1962    1

> tsp(z)     # 時間属性"tsp"を得る（開始・終了時間と，1 単位当たりの観測周波数の長さ 3 のベクトル）
[1] 1961 1962    4
```

```
# 多変量時系列
> z <- ts(matrix(rnorm(300), 100, 3), start=c(1961, 1), frequency=12)
> dim(z)                                             # 行列としての次元は 100x3
[1] 100   3
> length(z)                                          # ベクトルとしての総サイズ
[1] 300
> length(z[,1])                                      # 第一時系列のサイズ
[1] 100
```

```
# 米国大統領指示率時系列データ presidents を使用
> time(presidents)                                   # presindents データのサンプリング時刻
        Qtr1    Qtr2    Qtr3    Qtr4
1945 1945.00 1945.25 1945.50 1945.75
(途中省略)
1974 1974.00 1974.25 1974.50 1974.75

> c(time(presidents))                                # ベクトル化関数 c() でただのベクトルに変換
  [1] 1945.00 1945.25 1945.50 1945.75 1946.00 1946.25 1946.50 1946.75 1947.00
(途中省略)
[118] 1974.25 1974.50 1974.75

> c(presidents)                                      # 時系列データそのものもただのベクトルに変換
  [1] NA 87 82 75 63 50 43 32 35 60 54 55 36 39 NA NA 69 57 57 51 45 37 46 39
(途中省略)
[101] 66 53 61 52 51 48 54 49 49 61 NA NA 68 44 40 27 28 25 24 24

> plot(c(time(presidents)), c(presidents), type="l") # データのプロット例
```

4.1.7　時系列オブジェクトの部分時系列を得る window()

window() 関数は，一つの時系列オブジェクトから，時間範囲を指定して部分時系列を取り出したり，置き換えたりする．

書式：

```
window(x, ...)
# クラス ts に対する S3 メソッド
window(x, ...)
# 既定の S3 メソッド
window(x, start = NULL, end = NULL, frequency = NULL, deltat = NULL,
       extend = FALSE, ...)
window(x, ...) <- value
# クラス ts に対する S3 置換メソッド
window(x, start, end, frequency, deltat, ...) <- value
```

引数：

x　時系列オブジェクト
start　指定時間帯の始点 (始・終点時間は ts() 関数と同様に指定される)
end　指定時間帯の終点
frequency　新しい周波数を指定
deltat　新しい周波数を指定
extend　論理値．もし TRUE なら始・終点時間は現在の時間範囲を拡大しても良い (時系列は必要に応じて NA 値で埋められる)． FALSE なら，時間範囲を拡大しようとすると無視される
...　追加任意引数
value　置き換える値

返り値：　返り値はメソッドによる．window.default() は適当な tsp 属性を持つベクトルか行列を返す．window.ts() はクラス ts のオブジェクトを返す点だけが異なる．もし extend=TRUE なら，必要に応じて系列は NA 値で埋められる

始・終点時間は ts() と同様に指定できる．もし指定時間に観測値が無いと，直後 (start の場合) もしくは直前 (end の場合) の値が使われる．置き換え関数は ts オブジェクトに対するメソッドを持ち，時間範囲を拡大できる (警告がでる)．既定のメソッドは無い．

```
# 米国大統領の四半期毎の支持率データ presidents 使用
> window(presidents, 1960, c(1969,4))             # 1960 年代の値
     Qtr1 Qtr2 Qtr3 Qtr4
1960   71   62   61   57
1961   72   83   71   78
(途中省略)
1968   36   49   35   44
1969   59   65   65   56

> window(presidents, deltat=1)                    # 第 1 四半期全て
Time Series:
Start = 1945
End = 1974
Frequency = 1
 [1] NA 63 35 36 69 45 36 25 59 71 71 76 79 60 57 71 72 79 76 80 71 63 44 36 59
[26] 66 51 49 68 28

> window(presidents, start=c(1945,3), deltat=1)   # 第 3 四半期全て
Time Series:
Start = 1945.5
End = 1974.5
Frequency = 1
 [1] 82 43 54 NA 57 46 32 NA 75 71 79 67 63 48 61 61 71 62 62 69 69 56 38 35 65
[26] 61 54 NA 40 24

> window(presidents, 1944, c(1979,2), extend=TRUE) # 拡大部は NA で埋められる
     Qtr1 Qtr2 Qtr3 Qtr4
1944   NA   NA   NA   NA
1945   NA   87   82   75
1946   63   50   43   32
(途中省略)
1978   NA   NA   NA   NA
1979   NA   NA

> pres <- window(presidents, 1945, c(1949,4))     # 1940 年代の値
> window(pres, 1945.25, 1945.50) <- c(60, 70)
> window(pres, 1944, 1944.75) <- 0                # 時間範囲を拡大するので警告がでる
Warning message:
 値の置き換えの際，時系列を拡大しました
```

```
# 1945 年第 4 四半期から単位時間毎（つまり一年おき）に値を置き換え
> window(pres, c(1945,4), c(1949,4), frequency=1) <- 85:89
> pres
     Qtr1 Qtr2 Qtr3 Qtr4
1944    0    0    0    0
1945   NA   60   70   85
1946   63   50   43   86
1947   35   60   54   87
1948   36   39   NA   88
949   69   57   57   89
```

4.1.8　時系列のラグ演算 lag()

時系列のラグ (遅延演算) を取る．与えられた観測値数分，時間軸を過去にずらす．

書式：
```
lag(x, ...)
lag(x, k = 1, ...)  # 既定の S3 メソッド
```
引数：

x	ベクトル，行列，または一 (多) 変量時系列
k	ラグ数 (観測値数単位の)
...	他のメソッドへ (から) 引き渡される追加引数

返り値： 返り値は時系列オブジェクト．正の k でラグを取った時系列は単位時間で k だけ早く始まり，長さはその分短くなる

x がベクトルもしくは行列なら時系列に強制変換される．lag() は総称的関数であり，ここでは既定のメソッドを説明する．

```
# 英国の呼吸器疾患死亡者数データ UKLungDeaths を使用
> ldeaths                                                    # 元の時系列
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1974 3035 2552 2704 2554 2014 1655 1721 1524 1596 2074 2199 2512
(途中省略)
1979 3084 2605 2573 2143 1693 1504 1461 1354 1333 1492 1781 1915
> lag(ldeaths, 12)                                           # 12 ヵ月分早く始まる
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1973 3035 2552 2704 2554 2014 1655 1721 1524 1596 2074 2199 2512
(途中省略)
1978 3084 2605 2573 2143 1693 1504 1461 1354 1333 1492 1781 1915
```

4.1.9 時系列オブジェクトに対するメソッド，diff(), na.omit()

クラス "ts" の時系列オブジェクト (普通関数 ts() の結果) に対するメソッド．

書式：
```
diff(x, lag=1, differences=1, ...) # クラス"ts"に対する S3 メソッド
na.omit(object, ...)               # クラス"ts"に対する S3 メソッド
```
引数：

x	階差を取るべきクラス "ts" のオブジェクト
lag	用いるラグを指示する整数
differences	用いる階差の階数を指示する整数
object	一変量，もしくは多変量時系列
...	他のメソッドに (から) 引き渡される追加引き数

メソッド na.omit() は時系列の先頭と末尾にある (引き続く) 欠損値を取り去る．内部に含まれる欠損値はエラーとなる．返り値は欠損値の無い時系列となる．object のクラスは保存される．na.omit(), na.fail() そして na.contiguous() を参照せよ．

4.1.10 時系列を部分的に集約する aggregate()

データを部分集合に分割し，各々に対する要約統計量を計算し，結果を適切な形式で返す．

書式：

```
aggregate(x, ...)
aggregate(x, by, FUN, ...) # クラス "data.frame" 用メソッド
# クラス "ts" 用メソッド
aggregate(x, nfrequency = 1, FUN = sum, ndeltat = 1,
          ts.eps = getOption("ts.eps"), ...)
```

引数：

x	R オブジェクト
by	グループ化要素のリストで，各々は x 中の変数と同じ長さを持つ．
FUN	要約統計量を計算する，全てのデータサブセットに適用可能なスカラ値関数
nfrequency	時間単位毎の新しい観測値数． x の周波数の約数でなければならない
ndeltat	引き続く観測値間のサンプリング間隔の新しい比率． x のサンプリング間隔の約数でなければならない
ts.eps	nfrequency が元の周波数の約数かどうかを判定する許容度
...	メソッドに引き渡される (使われる) 追加引き数

aggregate() はデータフレームと時系列に対するメソッドを持つ総称的関数である．既定のメソッド aggregate.default() は，もし x が時系列なら時系列用メソッドを使い，さもなければ，x をデータフレームに強制変換してからデータフレーム用メソッドを呼び出す．

aggregate.data.frame() はデータフレーム用のメソッドである．もし x がデータフレームでなければ，データフレームに強制変換される．それから，x 中の変数 (列) の各々が by の成分の同一の組合せを持つケース (行) のサブセットに分割される．そして，FUN が ... 中の追加引き数を用いて，各サブセットに適用される．(つまり，x 中の各変数 VAR に対して，実際は lapply() の一回の呼び出しに置き換えられた，tapply(VAR, by, FUN, ..., simplify = FALSE) が実行される．) 空のサブセットは除去され，結果は by と x 中の変数を含むデータフレームに再整形される．by に由来する変数はサブセットを決定するのに使われたグルーピング変数の一意的な組合せを含み，x に由来する変数は，対応する x 中のサブセットに対する要約統計量を含む．

aggregate.ts() は時系列用のメソッドである．もし x が時系列でなければ，時系列に強制変換される．それから，x 中の変数が長さ frequency(x)/nfrequency の適当なブロックに分割され，... 中の (名前付き) 引き数を備えた関数 FUN が各ブロックに適用される．返される結果は集約された値を含む周波数 nfrequency の時系列である．

```
# 米国 50 州の各種データ state.x77 中の変数を，4 地域毎にグループ化し平均を取る
> aggregate(state.x77, list(Region = state.region), mean)
        Region Population   Income Illiteracy Life Exp    Murder  HS Grad
```

```
1     Northeast   5495.111 4570.222   1.000000 71.26444  4.722222 53.96667
2         South   4208.125 4011.938   1.737500 69.70625 10.581250 44.34375
3 North Central   4803.000 4611.083   0.700000 71.76667  5.275000 54.51667
4          West   2915.308 4702.615   1.023077 71.23462  7.215385 62.00000
     Frost      Area
1 132.7778  18141.00
2  64.6250  54605.12
3 138.8333  62652.00
4 102.1538 134463.00
```

```
# 地域と霜が 130 日以上降りたかどうかで平均を取る (South にはそうした州が無い)
> aggregate(state.x77, list(Region = state.region,
            Cold = state.x77[,"Frost"] > 130), mean)
         Region  Cold Population   Income Illiteracy Life Exp    Murder
1     Northeast FALSE  8802.8000 4780.400  1.1800000 71.12800  5.580000
2         South FALSE  4208.1250 4011.938  1.7375000 69.70625 10.581250
3 North Central FALSE  7233.8333 4633.333  0.7833333 70.95667  8.283333
4          West FALSE  4582.5714 4550.143  1.2571429 71.70000  6.828571
5     Northeast  TRUE  1360.5000 4307.500  0.7750000 71.43500  3.650000
6 North Central  TRUE  2372.1667 4588.833  0.6166667 72.57667  2.266667
7          West  TRUE   970.1667 4880.500  0.7500000 70.69167  7.666667
   HS Grad    Frost      Area
1 52.06000 110.6000  21838.60
2 44.34375  64.6250  54605.12
3 53.36667 120.0000  56736.50
4 60.11429  51.0000  91863.71
5 56.35000 160.5000  13519.00
6 55.66667 157.6667  68567.50
7 64.20000 161.8333 184162.17
```

```
# 米国大統領支持率データ presidents の年毎の支持率を求める
> aggregate(presidents, nf = 1, FUN = mean)
 Time Series: Start = 1945 End = 1974 Frequency = 1
  [1]    NA 47.00 51.00    NA 58.50 41.75 28.75    NA 67.00 65.00 72.75 72.25
 [13] 65.25 52.25 61.50 62.75 76.00 71.50 64.75 72.75 66.50 52.25 45.00 41.00
 [25] 61.25 58.00 50.50    NA 44.75 25.25

# 夏期により少ない重みを与え平均
> aggregate(presidents, nf = 1, FUN = weighted.mean, w = c(1, 1, 0.5, 1))
 Time Series: Start = 1945 End = 1974 Frequency = 1
  [1]       NA 47.57143 50.57143       NA 58.71429 41.14286 28.28571       NA
  [9] 65.85714 64.14286 71.85714 73.00000 65.57143 52.85714 61.57143 63.00000
 [17] 76.71429 72.85714 65.14286 73.28571 66.14286 51.71429 46.00000 41.85714
 [25] 60.71429 57.57143 50.00000       NA 45.42857 25.42857
```

4.1.11 欠損値を含まない最長の部分時系列を取り出す na.contiguous()

メソッド na.omit() (実際は関数 na.omot.ts()) は時系列の先頭と末尾にある (引き続く) 欠損値を取り去る．内部に含まれる欠損値はエラーとなる．返り値は欠損値の無い時系列となる．object のクラスは保存される．na.omit(), na.fail() そして na.contiguous() を参照せよ．

na.contiguous() は引き続いて欠損値を含まないような，最長の部分時系列を取り出す．もしそうしたものが複数あれば，最初のものを返す．引き数 frame は一次元，もしくは多次元時系列である．frame のクラスは保存される．

書式：

```
# クラス "ts" に対する S3 メソッド
na.omit(object, ...)
na.contiguous(object, ...)
```

引数：

object 一変量，もしくは多変量時系列
... 他のメソッドに (から) 引き渡される追加引き数

```
> (x <- ts(c(NA,NA,1:10,NA), start=c(2008,1), frequency=12)) # 先頭と末尾に NA を含む時系列
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2008  NA  NA   1   2   3   4   5   6   7   8   9  10
2009  NA
> na.omit(x)                                              # 先頭と末尾の NA が全て除かれる
     Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2008   1   2   3   4   5   6   7   8   9  10

> (x <- ts(c(NA,NA,1:5,NA,6:10,NA), start=c(2008,1), frequency=12))
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2008  NA  NA   1   2   3   4   5  NA   6   7   8   9
2009  10  NA                                                # 途中に NA があるとエラー
> na.omit(x) 以下にエラー na.omit.ts(x) :  時系列は内部に NA 値を含んでいます

> (x <- ts(cbind(c(NA,NA,1:10,NA),c(1:11,NA,NA)), start=c(2008,1),
           frequency=12))                                 # 多変量時系列の場合
         Series 1 Series 2
Jan 2008       NA        1
Feb 2008       NA        2
Mar 2008        1        3
Apr 2008        2        4
May 2008        3        5
Jun 2008        4        6
Jul 2008        5        7
Aug 2008        6        8
Sep 2008        7        9
Oct 2008        8       10
Nov 2008        9       11
Dec 2008       10       NA
Jan 2009       NA       NA
> na.omit(x)
         Series 1 Series 2
Mar 2008        1        3
Apr 2008        2        4
May 2008        3        5
Jun 2008        4        6
Jul 2008        5        7
Aug 2008        6        8
Sep 2008        7        9
Oct 2008        8       10
Nov 2008        9       11
```

```
> (x <- ts(c(NA,NA,1:5,NA,6:11,NA), start=c(2008,1), frequency=12))
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2008  NA  NA   1   2   3   4   5  NA   6   7   8   9
2009  10  11  NA

> na.contiguous(x)                                  # NA 値を含まない最長の副時系列を取り出す
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2008                                   6   7   8   9
2009  10  11

> (x <- ts(cbind(c(NA,NA,1:5,NA,6:9,NA),c(1:11,NA,NA)), start=c(2008,1),
           frequency=12))                           # 多変量時系列の場合
         Series 1 Series 2
Jan 2008       NA        1
Feb 2008       NA        2
Mar 2008        1        3
Apr 2008        2        4
May 2008        3        5
Jun 2008        4        6
Jul 2008        5        7
Aug 2008       NA        8
Sep 2008        6        9
Oct 2008        7       10
Nov 2008        8       11
Dec 2008        9       NA
Jan 2009       NA       NA
```

```
> na.contiguous(x)
         Series 1 Series 2
Mar 2008        1        3
Apr 2008        2        4
May 2008        3        5
Jun 2008        4        6
Jul 2008        5        7
```

4.2 時系列の自己共分散・相関係数，スペクトル密度

4.2.1 時系列の自己共分散と自己相関係数 acf(), pacf(), acf()

時系列データに関する最も基本的な統計量は自己共分散 (auto-covariance) と自己相関係数 (auto-correlation) である．R の関連する関数は acf(), pacf(), acf() である．関数 acf() は時系列オブジェクトの自己共分散と自己相関係数を計算し，既定でそれをプロットする．関数 pacf() は 部分自己相関係数 (partial auto-correlations) を計算する．関数 ccf() は二つの一次元時系列間のクロス相関係数 (cross-correlation) とクロス共分散 (cross-cavarinace) を計算する．

書式：

```
acf(x, lag.max=NULL, type=c("correlation","covariance","partial"),
    plot=TRUE, na.action=na.fail, demean=TRUE, ...)
pacf(x, lag.max, plot, na.action, ...)
# 既定の S3 メソッド
pacf(x, lag.max = NULL, plot = TRUE, na.action = na.fail,  ...)
ccf(x, y, lag.max = NULL, type = c("correlation", "covariance"),
    plot = TRUE, na.action = na.fail, ...)
x[i, j] # クラス acf に対する S3 メソッド
```

引数：

x, y　一変量もしくは多変量 (ccf() を除く) 数値時系列，数値ベクトルもしくは行列．もしくはクラス "acf" のオブジェクト

lag.max　自己共分散・相関を計算する最大ラグ．既定では 10*log10(N/m) で， N は観測値総数で m は系列の総数．自動的に時系列の長さ引く 1 に制限される

type　文字列で "correlation" (既定), "covariance"，もしくは "partial"．計算する量を指定する

plot　論理値．もし TRUE (既定) なら結果をプロットする

na.action　欠損値を処理するために呼び出される関数． na.pass() が使える

demean　論理値．共分散は標本平均周りで計算すべきか？

...　プロット用関数 plot.acf() に渡される追加引数

i　保持されるラグ (時間差) 集合

j　保持される系列 (名前もしくは数) 集合

返り値：　クラス "acf" のリストであり，次のような成分を持つ：

lag　自己相関係数等が評価されたラグ値を含む 3 次元配列

acf　推定された自己相関係数等を含む lag と同じ次元の配列

type	相関・共分散の種類 (type 引き数と同じ)
n.used	時系列中の観測数
series	時系列 x の名前
snames	多変量時系列中の個別系列の名前

type="correlation" と type="covariance" では推定値は標本共分散に基づく．ラグ 0 の自己相関係数は便宜的にラグ 1 とされる．

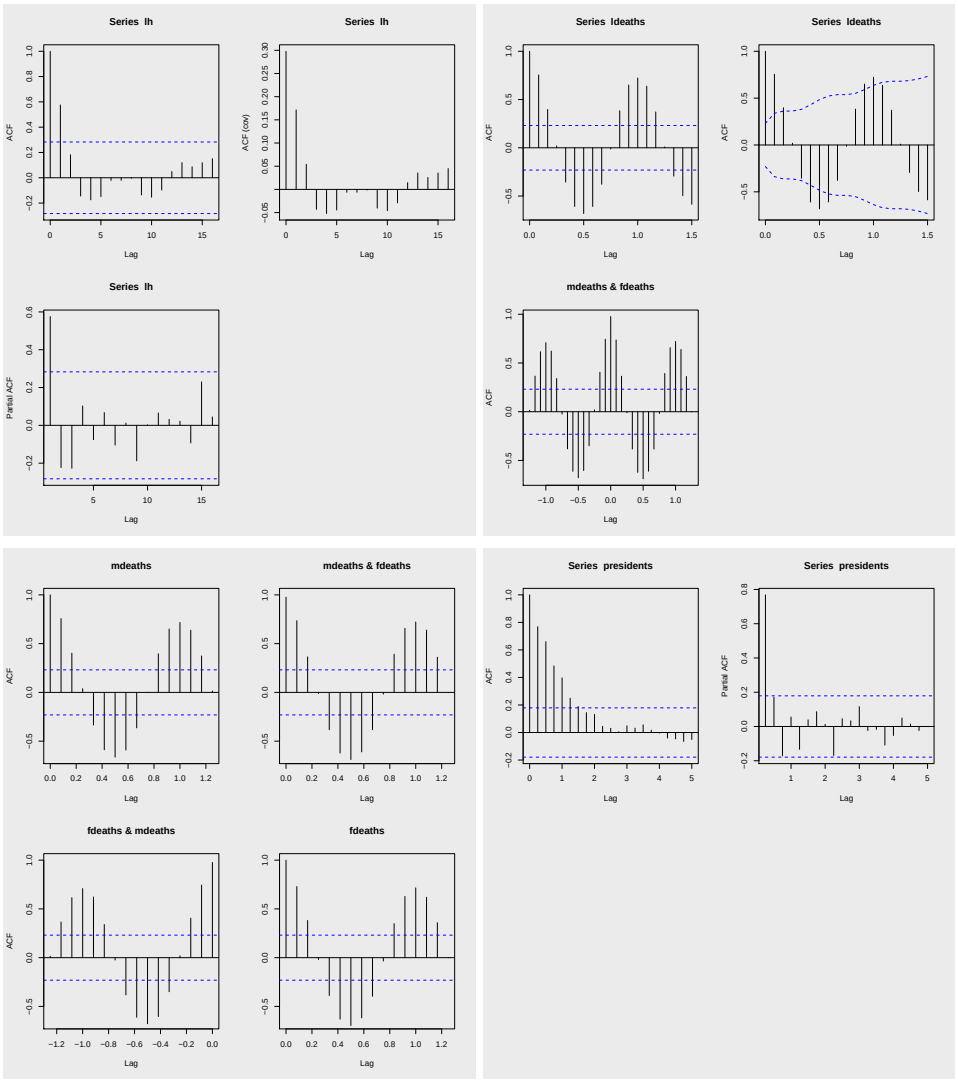
既定では，欠損値 NA は許されない．もし na.action 関数が欠損値を (na.pass() 関数のように) 放置するならば，共分散は観測値が完全なものから計算される．これは計算された推定値が正確な自己相関ではないかも知れないこと，そして欠損値を含むかも知れないことを意味する．多変量時系列から部分自己相関係数を計算する際は，欠損値は許されない．部分自己相関係数は，最大 lag.max 次までの自己回帰モデルを，順に当てはめることにより推定される．プロット関数 plot() は総称的であり，クラス "acf" のオブジェクトに対するメソッド plot.acf() を持つ．acf() 関数等が表示するプロットは，実際は plot.acf() が実行する．ラグ値が返され，観測値数ではなく，時間の単位に対しプロットされる．クラス "acf" のオブジェクトは print() メソッドと，[] による部分抽出メソッドを持つ．

推定値 ccf(x,y) が返すラグ k の値は x[t+k] と y[t] 間の相関の推定値である．もし plot=TRUE なら不可視返り値になる．

```
# 黄体形成ホルモン量データ lh を使用 (以下の図を参照)
acf(lh)                                          # 自己相関係数
acf(lh, type = "covariance")                     # 自己共分散
pacf(lh)                                         # 自己部分相関係数
```

```
# 英国の呼吸器疾患死亡者数データ UKLungDeaths を使用 (以下の図を参照)
> acf(ldeaths)                                   # 両性データの自己相関係数
> acf(ldeaths, ci.type = "ma")                   # 信頼区間を MA 過程として計算
> acf(ts.union(mdeaths, fdeaths))                # 自己・クロス相関係数を計 4 種類計算
> ccf(mdeaths, fdeaths)                          # 男女死亡率間のクロス相関係数
```

```
# 欠損値を含む時系列データ presidents を使用 (以下の図を参照)
> acf(presidents, na.action = na.pass)           # 欠損値を含む自己相関係数
> pacf(presidents, na.action = na.pass)          # 欠損値を含む部分自己相関係数
```



自己相関係数．(左上) 黄体形成ホルモン量データ，(右上) 呼吸器疾患死亡者数，(左下) 英国の呼吸器疾患死亡者数データ，(右下) 米国大統領支持率 (欠損値を含む)

4.3 スペクトル密度関数

4.3.1 時系列のスペクトル密度関数の推定 `spectrum()`

`spectrum()` は時系列のスペクトル密度関数 (spectral density) を推定する．

書式： `spectrum(x, ..., method = c("pgram", "ar"))`

引数：

`x`　一変量，もしくは多変量時系列

`method`　文字列で，スペクトル密度を推定する手法を指定する．可能な手法は`"pgram"` (既定手法) と `"ar"` (AR モデルの当てはめに基づく)

`...`　密度推定法，もしくはプロットメソッド `plot.spec()` に渡される追加引数

返り値：　クラス `"spec"` を持つオブジェクトで，少なくとも次の成分を持つリスト：

`freq`　スペクトル密度が推定された周波数のベクトル (可能性としてフーリエ周波数を近似)．単位は，単位時間当たり (観測値間隔当たりではなく) のサイクルの逆数

`spec`　`freq` に対応する周波数位置に於けるスペクトル密度のベクトル (一変量時系列の場合)，もしくは行列 (多変量時系列の場合)

`coh`　一変量時系列では `NULL`．多変量時系列の場合，`coh` の `i+(j-1)*(j-2)/2` 列は `x` の `i`, `j` 列 (`i<j`) 間のコヒーレンシ (coherency，干渉性) の 2 乗値を含む

`phase`　一変量時系列では `NULL`．多変量時系列の場合，異なる時系列間のクロススペクトルのフェイズ (cross-spectrum phase) を含む．形式は `coh` と同様

`series`　時系列の名前

`snames`　多変量時系列の場合，成分時系列の名前

`method`　スペクトル密度を計算した手法

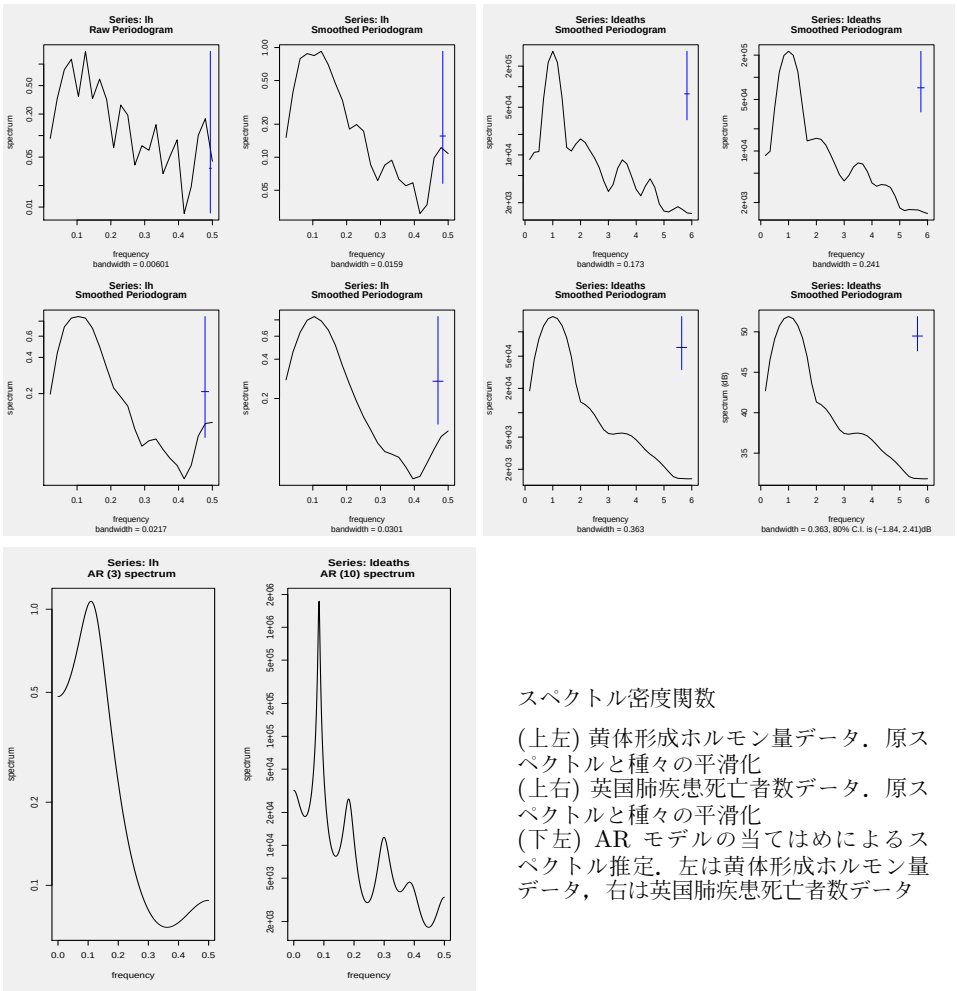
`spectrum()` は二つのメソッド `spec.gram()`, `spec.ar()` に対するラッパ関数 [*3] である．スペクトル密度関数は S-PLUS との互換性のために，スケール `1/frequency(x)` を掛けてあり，関数の範囲は `(-frequency(x)/2, +frequency(x)/2]` となる．より普通には，スケールを 2π とし範囲を $(-0.5, 0.5]$ としたり，1 として $(-\pi, \pi]$ とする．もし可能なら `plot.spec()` 関数を用いて信頼区間がプロットされる．これは非対称であり，センターマークの幅はバンド幅と等値である．

注意：クラス `"spec"` のオブジェクトの既定のプロットは非常に複雑であり，エラーバーと既定のタイトル，副タイトルそして軸ラベルを持つ．これらの既定値は，適当な作図パラメータを補うことで，変更することができる．

関連： `spec.ar()`, `spec.pgram()`, `plot.spec()`.

```
# 黄体形成ホルモン量データ lh を使用 (以下の図を参照)
> par(mfrow=c(2,2))                                  # 画面を 2x2 分割
> spectrum(lh); spectrum(lh, spans=3); spectrum(lh, spans=c(3,3))
> spectrum(lh, spans=c(3,5))

# 英国の呼吸器疾患死亡者数データ UKLungDeaths を使用 (以下の図を参照)
> spectrum(ldeaths, spans=c(3,3)); spectrum(ldeaths, spans=c(3,5))
> spectrum(ldeaths, spans=c(5,7))
> spectrum(ldeaths, spans=c(5,7), log="dB", ci=0.8)

# AR モデルフィットによるスペクトル密度推定例 (以下の図を参照)
# 多変量時系列のプロットに付いては spec.pgram() のヘルプを参照
> par(mfrow=c(1,2))                                  # 画面を 1x2 分割
> spectrum(lh, method="ar"); spectrum(ldeaths, method="ar")
```

[*3] wrapper function. (より複雑な) 関数 (群) を簡便に使用するための関数.



スペクトル密度関数

(上左) 黄体形成ホルモン量データ．原スペクトルと種々の平滑化
(上右) 英国肺疾患死亡者数データ．原スペクトルと種々の平滑化
(下左) AR モデルの当てはめによるスペクトル推定．左は黄体形成ホルモン量データ，右は英国肺疾患死亡者数データ

4.3.2 AR モデルの当てはめによりスペクトル密度を推定する，spec.ar()

spec.ar() は x に AR モデルを当てはめ (もしくは既に存在する当てはめ結果) により，スペクトル密度を推定する．

返り値はクラス "spec" のオブジェクトで，もし plot = TRUE ならコンソールには出力されない．

書式：

```
spec.ar(x, n.freq, order = NULL, plot = TRUE, na.action = na.fail,
        method = "yule-walker", ...)
```

引数：

x	一変量 (多変量版はまだ未移植) 時系列．もしくは ar() による当てはめ結果
n.freq	プロットされる点の数
order	当てはめられる AR モデルの次数．省略されると次数は AIC で選ばれる
plot	ペリオドグラムをプロットするか？
na.action	欠損値に対する処理関数
method	ar() 当てはめに対する手法

... plot.spec() に引き渡される作図パラメータ

返り値：　返り値はクラス "spec" のオブジェクト．もし plot=TRUE なら値は不可視

注意：Thomson 等は AR スペクトルは誤解を与えやすいと強く警告している．多変量ケースはまだ移植されていない．

4.3.3　ペリオドグラムの計算 spec.pgram()

spec.pgram() は高速フーリエ変換によりペリオドグラムを計算する．オプションとして結果を修正 Daniel 平滑法 (終端の重みを半分にする移動平均) による系列で平滑化する．もし plot = TRUE なら結果はコンソールに表示されない．

書式：
```
spec.pgram(x, spans=NULL, kernel, taper=0.1, pad=0, fast=TRUE,
           demean=FALSE, detrend=TRUE, plot=TRUE,
           na.action=na.fail, ...)
```

引数：

x　一変量もしくは多変量時系列

spans　奇数からなるベクトルで，ペリオドグラムの平滑化に使われる修正 Daniell 平滑法の幅を与える

kernel　もしくはクラス "tskernel" の平滑法

taper　テイパリングされるデータの割合．半 cosine bell テーパが，データの先頭と末尾のこの比率の部分に適用される

pad　埋められるデータの割合．データの長さを増すために，データの末尾にこの割合分 0 が加えられる

fast　論理値．TRUE なら，0 で埋めることにより，系列の長さを約数が極めて多い数にする

demean　論理値．TRUE なら，時系列から平均を引く

detrend　論理値．TRUE なら，時系列から線形傾向を引く．同時に平均も引き去る

plot　ペリオドグラムをプロットするか？

na.action　欠損値処理関数

... plot.spec() に引き渡される作図パラメータ

返り値：　クラス "spec" (spectrum() を見よ) のリストで，次の追加成分を持つ：

kernel　kernel 引き数．もしくは spans から構成された核関数

df　スペクトル密度関数推定値の分布を近似するカイ 2 乗分布の自由度

bandwidth　Bloomfield で定義されているような核関数平滑法の同値バンド幅

taper　taper 引き数の値

pad　pad 引き数の値

detrend　detrend 引き数の値

demean　demean 引き数の値

生のペリオドグラムはスペクトル密度の一致推定量ではないが，隣接した値は漸近的に

独立である．したがって，スペクトル密度が滑らかという仮定の下で，生のペリオドグラムを平滑化することによりスペクトル密度の一致推定量を得ることができる．高速フーリエ変換を助けるために，時系列はその長さが高度な合成数 (約数が多い) になるように 0 で埋められる．これは pad ではなく fast で制御される．時系列の平均が引かれているので，原点でのペリオドグラムは理論的に 0 になる (テーパリングにより影響を受けるかも知れない)．これは平滑化の過程で隣接値の線形補間で置き換えられ，その周波数での値は返されない．

関連：spectrum(), spec.taper(), plot.spec(), fft().

4.3.4 時系列の両端を削る spec.taper()

cosine-bell 関数[*4] を，時系列 x[,i] の最初と最後の p[i]/2 分の観測値に適用する．返り値は新しい時系列である．

書式： spec.taper(x, p=0.1)

引数：

x 一変量もしくは多変量時系列

p 削られる両端部分の割合．スカラー (全ての時系列に適用される割合)，もしくは時系列の長さ (各時系列に適用される割合) のベクトル

4.3.5 累積ペリオドグラムをプロットする cpgram()

cpgram() は累積ペリオドグラムをプロットする．

書式：

```
cpgram(ts, taper=0.1, main=paste("Series: ",deparse(substitute(ts))),
       ci.col="blue")
```

引数：

ts 一変量時系列

taper ペリオドグラム作製時にテーパリングする割合

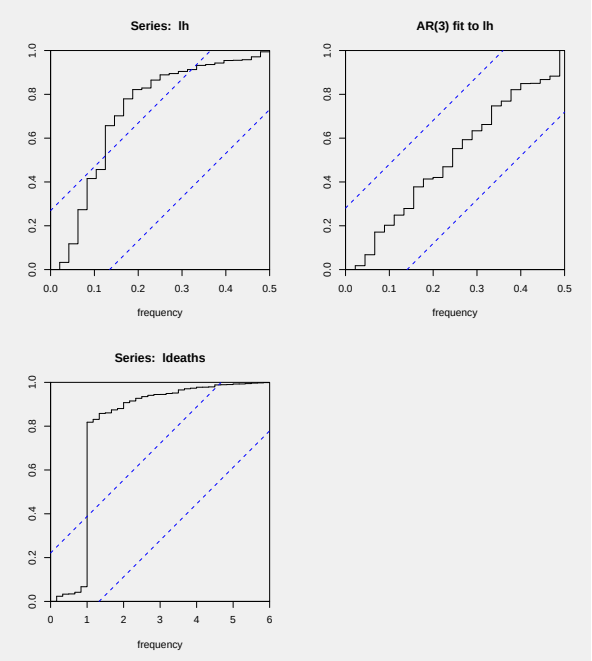
main 主タイトル

ci.col 信頼区間幅用の色

返り値： 返り値は無く，副作用として正方領域に累積ペリオドグラムが作図される

```
# 黄体形成ホルモン量データ lh 使用 (以下の図を参照)
> par(pty = "s", mfrow = c(2,2))                # 画面を 2x2 に分割．プロット点のタイプ指定
> cpgram(lh)                                    # 原データの cpgram
> lh.ar <- ar(lh, order.max = 9)                # AR(3) モデルを当てはめ
> cpgram(lh.arresid, main = "AR(3) fit to lh")  # その残差系列の cpgram
# 英国の肺疾患死亡者数データ ldeaths を使用 (以下の図を参照)
> cpgram(ldeaths)                               # 両性死亡者数数の cpgram
```

[*4] taper とは端を削って尖らすことを意味する．cosine bell とは関数 $(1-\cos(x))/2$ を指す．この 0 から 1 まで値が変化する関数を時系列の両端部分に掛けることにより，時系列の値を両端で徐々に 0 に近づける操作が cosine taper である．



黄体形成ホルモン量データの累積ペリオドグラム

原データ (上左)
AR(3) モデルを当てはめた残差 (上右)
英国の肺疾患死亡者数 (両性) データの累積ペリオドグラ (下左)

4.4　AR モデル

4.4.1　時系列への AR モデルの当てはめ ar()

時系列オブジェクトへの AR モデル (自己回帰モデル，auto-regressive model) の当てはめと予測を行う関数群.

```
書式：
ar(x, aic = TRUE, order.max = NULL,
   method=c("yule-walker", "burg", "ols", "mle", "yw"),
   na.action, series, ...)
ar.burg(x, ...)
# 既定の S3 メソッド
ar.burg(x, aic = TRUE, order.max = NULL, na.action = na.fail,
        demean = TRUE, series, var.method = 1, ...)
# クラス "mts" に対する S3 メソッド
ar.burg(x, aic = TRUE, order.max = NULL, na.action = na.fail,
        demean = TRUE, series, var.method = 1, ...)
ar.yw(x, ...)
# 既定の S3 メソッド
ar.yw(x, aic = TRUE, order.max = NULL, na.action = na.fail,
      demean = TRUE, series, ...)
# クラス "mts" に対する S3 メソッド
ar.yw(x, aic = TRUE, order.max = NULL, na.action = na.fail,
      demean = TRUE, series, var.method = 1, ...)
ar.mle(x, aic = TRUE, order.max = NULL, na.action = na.fail,
```

```
       demean = TRUE, series, ...)
# クラス "ar" に対する S3 メソッド
predict(object, newdata, n.ahead = 1, se.fit = TRUE, ...)
```

引数：

x　一変量，もしくは多変量時系列

aic　論理値フラグ．もし TRUE なら，自己回帰モデルの次数を選択するために赤池の情報量規準が使われる．もし FALSE なら次数 order.max のモデルが当てはめられる

order.max　当てはめられるモデルの (最大) 次数．既定では N を観測数として N-1 と 10*log10(N) の小さいほうが使われる．method="mle" では，この数と 12 の小さいほうとされる

method　モデルの当てはめに使われる手法を与える文字列．既定の引数のどれかである必要があるが，最初の数文字だけで十分である．既定では"yule-walker"

na.action　欠損値を処理するために使われる関数名

demean　論理値．当てはめの過程で平均値を推定するか?

series　時系列の名前．既定では deparse(substitute(x))

var.method　イノベーション分散を推定する手法

...　特定の手法のためのオプション追加引数

object　ar() 関数による当てはめ結果

newdata　予測を試みるデータ

n.ahead　何ステップ先まで予測を行うか?

se.fit　論理値．予測誤差の標準誤差の推定値を返すか?

返り値：　以下の成分を持つクラス "ar" のリスト：

order　当てはめられたモデルの次数．aic=TRUE ならば，AIC を最小化するものが，さもなければ order.max そのもの

ar　当てはめられたモデルの自己回帰係数の推定値

var.pred　予測分散．時系列の分散のうち，自己回帰モデルで説明できなかった部分の推定値

x.mean　時系列の平均値の推定値で，当てはめ過程と予測で用いられる

x.intercept　ar.ols() 専用．x-x.mean に対するモデルの切片値

aic　aic 引数の値

n.used　時系列中の総観測値数

order.max　order.max 引数の値

partialacf　最大ラグ order.max までの部分自己相関系数の推定値

resid　「次数 1」の観測値で条件付けられた，当てはめモデルの残差．「次数 1」の残差は NA とされる．もし x が時系列なら resid も時系列になる

method　引数 method の値

series　時系列の名前

frequency　時系列の周波数

call　マッチした呼出し式

asy.var.coef　(一次元時系列で，order>0 の場合) 漸近理論に基づく系数推定値の

分散行列

predict.ar()，予測値時系列，もしくは se.fit=TRUE に対しては，予測値である成分 pred と推定標準誤差である成分 se を持つリストである．成分はともに時系列になる．

時系列 $\{x_t\}$ に対する p 次の AR モデル AR(p) は次の式で定義される:

$$(x_t - m) = a_1(x_{t-1} - m) + \ldots + a_p(x_{t-p} - m) + \varepsilon_t$$

ここで，m は平均値，$a_1, \ldots, a_p$ は自己回帰係数，そして ε_t は時点 t での誤差項である．

関数 ar() は様々な手法で自己回帰モデルの当てはめを行う関数群，ar.yw(), ar.burg(), ar.ols() そして ar.mle(), へのラッパ関数である．これらは, aic=TRUE ならば AIC 法でモデルの選択を行う．しかしながら，真正の最尤推定法を用いる ar.mle() を除けばこれは問題なしとしない．AIC 値は分散の推定値が最尤推定値であるかのように計算され，尤度から行列式項は除かれる．これは，推定パラメータを用いて評価された正規分布尤度とは異なるものであることを注意しよう．ar.yw() では，イノベーションは当てはめ係数と x の自己共分散から計算される．

ar.burg() ではイノベーション分散，したがって AIC 値，の計算に二つの手法を用いることができる．手法 1 は S-PLUS と同様に Levinson-Durbin の更新式を更新のために用いる．手法 2 は，前進・後退予測誤差の 2 乗和の平均を用いる．推定係数は分散の推定法に (多少) 依存するであろう．

ar() 関数は，x の全体平均を引き去るか，引き去る定数を推定 (ar.mle()) することにより，既定でモデル中に定数を含むことを注意しよう．

注意： ar.mle() は一次元時系列に対してだけ実装されている．method="mle" のよる長い時系列当てはめは非常に時間がかかる可能性がある．

関連： ar.ols(), ARMA モデルに対しては arima0()，自己相関係数からの AR モデルの構築には acf2AR().

```
# 黄体形成ホルモン量データ lh を使用
> ar(lh)                                          # 既定の Yule-Walker 法 ar.yw() による当てはめ
Call:
ar(x = lh)                                              # 呼出し式
Coefficients:
      1        2        3                               # 当てはめモデルの係数
 0.6534  -0.0636  -0.2269
Order selected 3  sigma^2 estimated as  0.1959    # 3 次の AR モデル選択

> str(ar(lh))                                           # 当てはめ結果の詳細
List of 14
 $ order       : int 3
 $ ar          : num [1:3]  0.6534 -0.0636 -0.2269
 $ var.pred    : num 0.196
 $ x.mean      : num 2.4
 $ aic         : Named num [1:17] 18.307  0.996  0.538  0.000  1.490 ...
  ..- attr(*, "names")= chr [1:17] "0" "1" "2" "3" ...
 $ n.used      : int 48
 $ order.max   : num 16
 $ partialacf  : num [1:16, 1, 1]  0.576 -0.223 -0.227  0.103 -0.076 ...
 $ resid       : Time-Series [1:48] from 1 to 48: NA NA NA -0.200 -0.169 ...
 $ method      : chr "Yule-Walker"
 $ series      : chr "lh"
 $ frequency   : num 1
 $ call        : language ar(x = lh)
 $ asy.var.coef: num [1:3, 1:3]  0.02156 -0.01518 0.00482 -0.01518 0.03117 ...
 - attr(*, "class")= chr "ar"
```

```
> ar(lh, method="burg")                          # ar.burg() 使用
Call:
ar(x = lh, method = "burg")
Coefficients:
      1        2        3
 0.6588  -0.0608  -0.2234
Order selected 3  sigma^2 estimated as  0.1786

> ar(lh, method="ols")                           # ar.ols() 使用
Call:
ar(x = lh, method = "ols")
Coefficients:
    1
0.586
Intercept: 0.006234 (0.06551)
Order selected 1  sigma^2 estimated as  0.2016

> ar(lh, FALSE, 4)                               # 4 次の AR モデル当てはめ
Call:
ar(x = lh, aic = FALSE, order.max = 4)
Coefficients:
      1        2        3        4
 0.6767  -0.0571  -0.2941   0.1028
Order selected 4  sigma^2 estimated as  0.1983
```

```
> (sunspot.ar <- ar(sunspot.year))               # 太陽黒点数データ sunspot 使用
Call:
ar(x = sunspot.year)
Coefficients:
      1        2        3        4        5        6        7        8
 1.1305  -0.3524  -0.1745   0.1403  -0.1358   0.0963  -0.0556   0.0076
      9
 0.1941
Order selected 9  sigma^2 estimated as  267.5     # 9 次の AR モデル選択

> predict(sunspot.ar, n.ahead=25)                # 当てはめモデルによる予測
$pred                                            # 予測値時系列
Time Series:
Start = 1989
End = 2013
Frequency = 1
 [1] 135.25933 148.09051 133.98476 106.61344  71.21921  40.84057  18.70100
 [8]  11.52416  27.24208  56.99888  87.86705 107.62926 111.05437  98.05484
[15]  74.84085  48.80128  27.65441  18.15075  23.15355  40.04723  61.95906
[22]  80.79092  90.11420  87.44131  74.42284
$se                                              # 予測標準誤差時系列
Time Series:
Start = 1989
End = 2013
Frequency = 1
 [1] 16.35519 24.68467 28.95653 29.97401 30.07714 30.15629 30.35971 30.58793
 [9] 30.71100 30.74276 31.42565 32.96467 34.48910 35.33601 35.51890 35.52034
[17] 35.65505 35.90628 36.07084 36.08139 36.16818 36.56324 37.16527 37.64820
[25] 37.83954
```

```
# 2 変量時系列への当てはめ例. 売上高データ BJsales 使用
> ar(cbind(BJsales, BJsales.lead), method="burg")
Call:
ar(x = cbind(BJsales, BJsales.lead), method = "burg")
$ar                                              # 4 次の 2 変量 AR モデルを選択
, , 1                                            # 以下各次数の (行列) 係数

             BJsales BJsales.lead
BJsales      1.21197      0.07312
BJsales.lead 0.07411      0.45684
, , 2

              BJsales BJsales.lead
BJsales      -0.26022      -0.1120
BJsales.lead -0.06904       0.3111
, , 3

              BJsales BJsales.lead
```

```
BJsales      -0.01754      3.93591
BJsales.lead  0.01792      0.09632
, , 4

             BJsales BJsales.lead
BJsales      -0.07746     -1.33836
BJsales.lead -0.01158     -0.09118
$var.pred
             BJsales BJsales.lead
BJsales      0.38426      0.01315
BJsales.lead 0.01315      0.07657

> ar(BJsales, method="burg")                    # 1 変量で当てはめると AR(5) を選択
Call:
ar(x = BJsales, method = "burg")
Coefficients:
      1        2        3        4        5
 1.2213  -0.0619  -0.0649   0.0668  -0.1668
Order selected 5  sigma^2 estimated as  1.759
```

4.4.2 最小自乗法による AR モデルの当てはめ ar.ols()

通常最小自乗法 (OLS, Ordinary Least Square) により，時系列に AR モデルを当てはめる．既定では AIC 法で次数を決める．

書式： `ar.ols(x, aic = TRUE, order.max = NULL, na.action = na.fail, demean = TRUE, intercept = demean, series, ...)`

引数：

`x` 一 (多) 変量時系列

`aic` 論理値フラグ．もし `TRUE` なら，AIC 法を用いて AR モデルの次数を選択する．もし `FALSE` なら次数 `order.max` のモデルを使用する

`order.max` 当てはめモデルの (最大) 次数．既定では，観測値数を `N` とすると `10*log10(N)` が使われる

`na.action` 欠損値を処理する関数

`demean` `x` から平均値を引き去った AR モデルを当てはめるべきか？

`intercept` 切片 (定数) 項を別個に当てはめるべきか？

`series` 時系列の名前．既定では `deparse(substitute(x))`

`...` 他のメソッドへ (から) 引き渡される追加引き数

返り値： クラス `"ar"` オブジェクトで，以下の成分を持つ：

`order` 当てはめモデルの次数．これは，もし `aic=TRUE` なら AIC の最小化で選ばれる．さもなければ `order.max` である

`ar` 当てはめモデルの AR 係数

`var.pred` 予測分散．つまり，時系列の分散のうち，AR モデルで説明できない部分

`x.mean` 当てはめで使われた推定平均 (もし `demean=FALSE` なら 0) で，予測でも使われる

`x.intercept` `x-x.mean` に対するモデルの切片項．`intercept=FALSE` なら 0

`aic` `aic` 引き数の値

`n.used` 時系列中の観測値数

`order.max` `order.max` 引き数値

partialacf NULL. ar() との互換性のためにある
resid 第「一次」の観測値で条件付けた，当てはめモデルからの残差．第「一次」の残差は NA とされる．もし x が時系列なら，rsid も時系列
method 文字列 "Unconstrained LS"
series 時系列の名前
asy.se.coef 係数推定値の漸近理論による標準誤差

ar.ols() は一般の AR モデルを時系列 x に当てはめる．多変量時系列でも，可能性として非定常でも良い．たとえ，時系列の一部分が非定常でも，また (同じ階数で) 階差をとってあっても，結果の制約無し最小自乗推定値は一致性を持つ．正確には AR 係数は次のように符号が決められている：

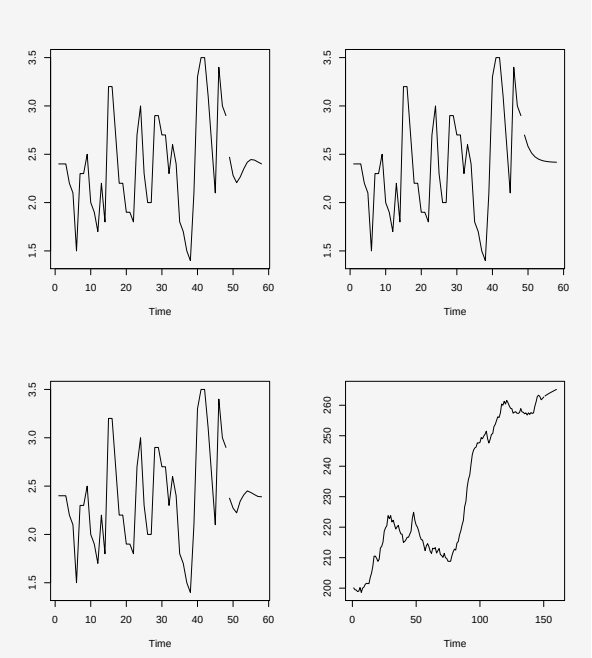
$$(x_t - m) = a_0 + a_1(x_{t-1} - m) + \ldots + a_p(x_{t-p} - m) + e_t$$

ここで a_0 は intercept = TRUE でない限り 0 であり，m は demean = TRUE なら標本平均，さもなければ 0 である．次数の選択は，もし aic = TRUE なら AIC 法で行われる．これは，ar.ols() が真の最尤推定を行わないので，問題無しとはしない．AIC は (残差の分散行列から計算された) 分散推定値があたかも MLE であるかのように計算され，尤度から行列式項を除外してある．これは，推定パラメータ値を用いて計算した正規分布尤度とは異なるものであることを注意しよう．もし intercept = TRUE で demean = FALSE であるときは，少し注意がいる．時系列がほぼ 0 の周りに中心化されているときだけこれを使用しよう．さもなければ，計算は不正確になるか，全く失敗するかも知れない．

```
# 黄体形成ホルモン量データ lh を使用 (以下の図を参照)
> (ts <- ar(lh, method="burg"))                    # ar() 法による AR モデルの当てはめ
> ts.plot(lh, predict(ts, n.ahead = 10)$pred)      # 時系列と予測値の同時プロット
> (ts <- ar.ols(lh))                               # ar.ols() 法で AR モデル当てはめ
Call:
ar.ols(x = lh)
Coefficients:
    1
0.586
Intercept: 0.006234 (0.06551)
Order selected 1  sigma^2 estimated as  0.2016
> ts.plot(lh, predict(ts, n.ahead = 10)$pred)      # 時系列と予測値の同時プロット
> ts <- ar.ols(lh, FALSE, 4)                       # AR(4) の当てはめ
> ts.plot(lh, predict(ts, n.ahead = 10)$pred)      # 時系列と予測値の同時プロット
> ts <- ar.ols(lh, order.max=6, demean=FALSE, intercept=TRUE)

> data(BJsales)                                    # 企業売り上げデータ読み込み
> ts <- ar.ols(BJsales)                            # AR モデルのの当てはめ
> ts.plot(BJsales, predict(ts, n.ahead=10)$pred)   # 系列，予測値の同時プロット
```

```
# ヨーロッパ株価指標データ EuStockMarkets を使用
> x <- diff(log(EuStockMarkets))                   # 対数値の一階階差を取る
# 最大 6 次までの AR モデルの当てはめ，平均化せず，定数項を含める
> ar.ols(x, order.max=6, demean=FALSE, intercept=TRUE)
Call:
ar.ols(x = x, order.max = 6, demean = FALSE, intercept = TRUE)
$ar
, , 1

           DAX       SMI       CAC    FTSE
DAX   0.004560 -0.095781  0.039975 0.04856
SMI  -0.009204 -0.007142  0.037758 0.06826
CAC  -0.026624 -0.113688  0.063807 0.09154
```

```
FTSE -0.010299 -0.089246 -0.003195 0.16409
$x.intercept
      DAX       SMI       CAC      FTSE
0.0006941 0.0007813 0.0004866 0.0004388
$var.pred
            DAX       SMI       CAC      FTSE
DAX  1.056e-04 6.683e-05 8.274e-05 5.192e-05
SMI  6.683e-05 8.496e-05 6.252e-05 4.254e-05
CAC  8.274e-05 6.252e-05 1.207e-04 5.615e-05
FTSE 5.192e-05 4.254e-05 5.615e-05 6.224e-05
```



黄体形成ホルモン量データへの AR モデル当てはめと 10 期先までの予測

(上左) `ar()` による当てはめ
(上右) `ar.ols()` による当てはめ
(下左) AR(4) の当てはめ
(下右) `ar.ols()` による企業売り上げデータへの AR モデルの当てはめ 10 期先までの予測

4.5　ARMA，ARIMA モデル

4.5.1　ARIMA モデルのシミュレーション `arima.sim()`

ARIMA モデルをシミュレーションする．返り値はクラス `"ts"` のオブジェクトである．

書式：

```
arima.sim(model, n, rand.gen = rnorm, innov = rand.gen(n, ...),
          n.start = NA, start.innov = rand.gen(n.start, ...), ...)
```

引数：

`model`　それぞれ AR,MA 部分の係数を与える成分 `ar` および (または) `ma` を持つリストで，オプションとして成分 `order` を持つことができる．空のリストを与えると白色雑音である `ARIMA(0,0,0)` になる

`n`　(階差を取られていない) 出力時系列の長さ

`rand.gen`　オプション．イノベーション項を与える疑似乱数の生成関数

`innov`　イノベーション項を与えるオプションの時系列．もし与えられなければ `rand.gen()` が使われる

`n.start`　空回し期間 (burn-in time，定常状態に入るまでに捨てられる系列) の長さ．

もし NA なら適当な値が計算される

start.innov 空回し期間として使われるオプションのイノベーション時系列．もし与えるなら，少なくとも n.start 個の値が必要 (既定では，n.start の値は内部的に計算される)

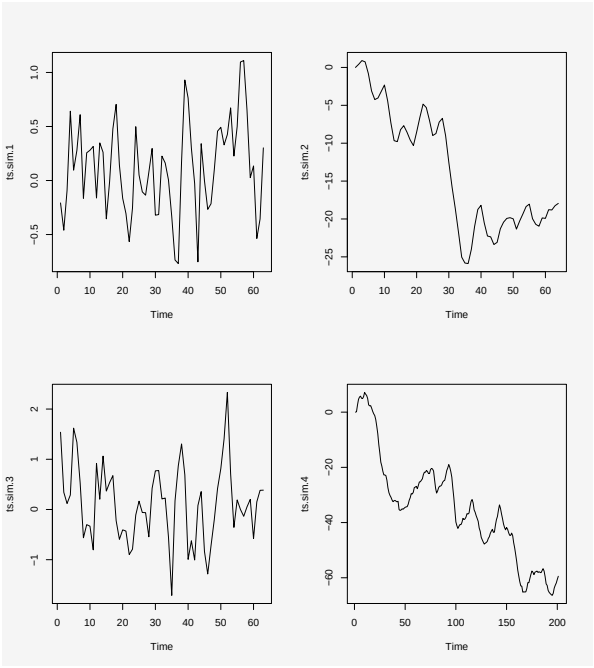
... rand.gen() に対する追加引数．最も有用な使い方として，正規疑似乱数 rnorm() で生成されるイノベーション項の標準偏差を sd で指示できる

ARIMA モデルの詳しい定義に付いては arima() を見よ．ARMA モデルは定常性のチェックがされる．ARIMA モデルは arima() 関数と同様に，リスト model の成分 order により指定される．order 成分の他の情報は無視されるが，MA と AR 次数の一貫しない指定は検出される．逆階差操作は前の値が 0 と仮定するが，このことをユーザに注意するために，これらの値が返される．burn-in 期間に対するランダム入力は rand.gen() を呼び出して生成される．

```
# ARIMA(2,0,2)=ARMA(2,2) モデルのシミュレーション (以下の図を参照)
# イノベーション分布は正規分布 rnorm(0,0.1796)
> (ts.sim.1 <- arima.sim(n = 63, list(ar = c(0.8897, -0.4858),
                          ma = c(-0.2279, 0.2488)), sd = sqrt(0.1796)))
Time Series: Start = 1 End = 63 Frequency = 1
 [1] -0.207476233 -0.459828160 -0.091900586  0.641633583  0.095884527
 [6]  0.278002757  0.608866695 -0.164613875  0.256478159  0.279978109
(途中省略)
[61] -0.536482207 -0.356062766  0.302301130
> ts.plot(ts.sim.1)
```

```
# ARIMA(2,1,2) モデルのシミュレーション (以下の図を参照)
> ts.sim.2 <- arima.sim(n = 63, list(order=c(2,1,2), ar=c(0.8897,-0.4858),
                        ma = c(-0.2279, 0.2488), sd = sqrt(0.1796)))
> ts.plot(ts.sim.2)
```

```
# ARIMA(2,0,2)=ARMA(2,2)：イノベーション分布の裾が中程度に長い例 (以下の図を参照)
# イノベーション分布は自由度 5 の t 分布乱数の srqt(0.1796) 倍
> ts.sim.3 <- arima.sim(n = 63, list(ar=c(0.8897,-0.4858),
                        ma = c(-0.2279,0.2488)),
                        rand.gen = function(n,...) sqrt(0.1796)*rt(n, df=5))
> ts.plot(ts.sim.3)

# ARIMA(1,1,0) モデルのシミュレーション (以下の図を参照)
> ts.sim.4 <- arima.sim(list(order = c(1,1,0), ar = 0.7), n = 200)
> ts.plot(ts.sim.4)
```



ARIMA モデルのシミュレーション
(左上) ARIMA(2,0,2)=ARMA(2,2) モデル
(右上) ARIMA(2,1,2) モデル
(左下) ARIMA(2,0,2)=ARMA(2,2) モデル (イノベーション分布の裾が中程度に長い例)
(右下) ARIMA(1,1,0) モデル

4.5.2 ARIMA モデルの当てはめ arima()

一変量時系列に ARIMA モデルを当てはめる．

書式：

```
arima(x, order = c(0, 0, 0),
      seasonal = list(order = c(0, 0, 0), period = NA),
      xreg = NULL, include.mean = TRUE, transform.pars = TRUE,
      fixed = NULL, init = NULL, method = c("CSS-ML", "ML", "CSS"),
      n.cond, optim.control = list(), kappa = 1e6)
```

引数：

`x` 一変量時系列

`order` ARIMA モデルの非季節部分の指定．三つの成分 (`p`,`d`,`q`) は AR 次数，階差階数，そして MA 次数

`seasonal` ARIMA モデルの季節部分の指定，と周期 (既定では `frequency()`)．これは成分 `order` と `period` を持つリストでなければならないが，単に長さ 3 の数値ベクトルを与えれば `order` のようなリストに変換される

`xreg` オプション．外部説明変数のベクトル，もしくは行列で，`x` と同じ数の行を持たなければならない

`include.mean` 論理値．ARIMA モデルは平均項を持つべきか？ 既定値は階差を取っていない時系列に対しては `TRUE` で，階差を取った時系列に対しては `FALSE` (この場合平均は当てはめにも，予測にも影響を与えないから)

`transform.pars` 論理値．もし真なら，AR パラメータは定常領域に留まるように変換される． `method="CSS"` の場合は使われない

`fixed` 全パラメータの数に等しい長さのオプションの数値ベクトル．もし与えられれば， `fixed` 中の `NA` である項目だけが変化する．もしどれかの AR パラメータが固定されれば，指示 `transform.pars=TRUE` は (警告とともに) 無効とされる．もし MA パラメータを固定する際 (特に非可逆ケースに近い場合) は，`transform.pars=FALSE` と設定したほうが賢明である

`init` オプションの初期パラメータ値の数値ベクトル．欠損値は回帰係数を除き 0 で埋められる． `fixed` で既に指定された値は無視される

`method` 当てはめ手法．最尤推定 `"ML"`，もしくは条件付き自乗和の最小化`"CSS"`．(欠損値が無い限り) 既定手法は，条件付き自乗和の最小化を用いて初期値を見付け，それから最尤法を使う `"CSS-ML"`

`n.cond` 条件付き自乗和により当てはめる場合だけ使われる．無視される最初の観測値の数．もし AR 項の最大ラグより小さければ無視される

`optim.control` 最適化関数 `optim()` に対する制御パラメータのリスト

`kappa` 階差時系列中の過去の観測値に対する事前分散 (イノベーション分散に対する比率で与える)．これを減らしてはならない

返り値： クラス `"Arima"` のリストで，次の成分を持つ：

`coef` AR,MA そして回帰係数のベクトルで， `coef()` メソッドで取り出せる

`sigma2` イノベーション分散の最尤推定量
`var.coef` 係数 `coef` の推定分散行列で，`vcov()` メソッドで取り出すことができる
`loglik` (階差データの) 最大対数尤度，もしくは用いられた手法によるそれに対する近似値
`arma` 簡潔なモデルの記述で，AR，MA，季節 AR，季節 MA 成分の係数の個数をベクトルとして与える．加えて周期と非季節・季節成分の数を持つ
`aic` 対数尤度に対する AIC 値．手法 `method="ML"` に対してだけ意味を持つ
`residuals` 標準化された残差
`call` マッチした呼出し式
`series` 時系列 `x` の名前
`convergence` `optim()` が返した値
`n.cond` 当てはめで使われなかった最初の観測値数
`model` 当てはめ中に使われたカルマンフィルタを表現するリスト．`KalmanLike()` を参照

ARMA モデルの定義には AR かつ (または) MA 係数の符号の取り方が一定していない．ここで用いる定義は：

$$X[t] = a[1]X[t-1] + \ldots + a[p]X[t-p] + e[t] + b[1]e[t-1] + \ldots + b[q]e[t-q]$$

であり，したがって S-PLUS とは MA 部分の係数の符号が異なる．更に，もし `include.mean=TRUE` (ARMA モデルに対する既定値) なら，この式が X ではなく，平均値 m を引き去った $X - m$ に適用される．階差を取った ARIMA モデルに対しては，階差時系列は平均 0 の ARMA モデルに従う．もし `xreg` 項が含められるなら，誤差項に対して (もし `include.mean=TRUE` なら定数項を含む) 線形回帰で ARMA モデルが当てはめられる．推定値の分散行列は，対数尤度のヘッシアン行列から計算され，従って，単におおよその見当と考えるべきである．最適化は `optim()` でなされる．これは，`xreg` 中の列がおおよそ平均 0 で単位分散にスケール化されている場合に最も上手く働くが，適当なスケーリングを推定することを試みる．

当てはめ手法：ARIMA プロセスの状態空間表現により正確な尤度が計算され，カルマンフィルタによりイノベーションとその分散が見出される．階差 ARMA プロセスの初期化は定常性を用い，Gardner に基づく．階差プロセスに対する非定常成分は (`kappa` で制御される) 拡散プライア (diffuse prior) を与えられる．拡散プライアにより依然として制御される (少なくとも `1e4` のカルマン利得を持つことで決定される) 観測値は尤度計算から除外される．(除外観測値がちょうど階差処理で失われるものと一致する場合，これは欠損値が無い場合の `arima0()` と比較可能な結果を与える．) 欠損値があっても良く，`"ML"` 手法では正確に処理される．もし `transform.pars = TRUE` なら，最適化は Jones により提案されたものの変形である，定常性を持つ別種のパラメータ化を用いて行われる．AR(p) モデルでは，パラメータ化は部分自己相関の `atanh()` を用いてなされ，同じ手順が (個別に) AR と季節 AR 項に適用される．MA 項は最適化の途中で可逆的であるとの制約をされないが，もし `transform.pars = TURE` なら最適化後に可逆な形に変換される．

条件付き自乗和法は主に説明のために提供されている．これは n.cond (n.cond は最低でも AR 項の最大ラグ数でなければならない) 以降の観測値に当てはめられたイノベーションの自乗和を計算し，それ以前の全てのイノベーションは 0 とする．引数 n.cond は異なった当てはめの比較の互換性を許すために使うことができる．part log-likelihood は最初の項で，推定平均 2 乗の対数の半分である．欠損値は許されるが，多くのイノベーションが欠損するであろう．説明変数が指定された場合は，係数が固定されない限り，それらは当てはめに対する直交化プライアである．説明変数を大まかに平均 0 で分散 1 とするのが役に立つ．

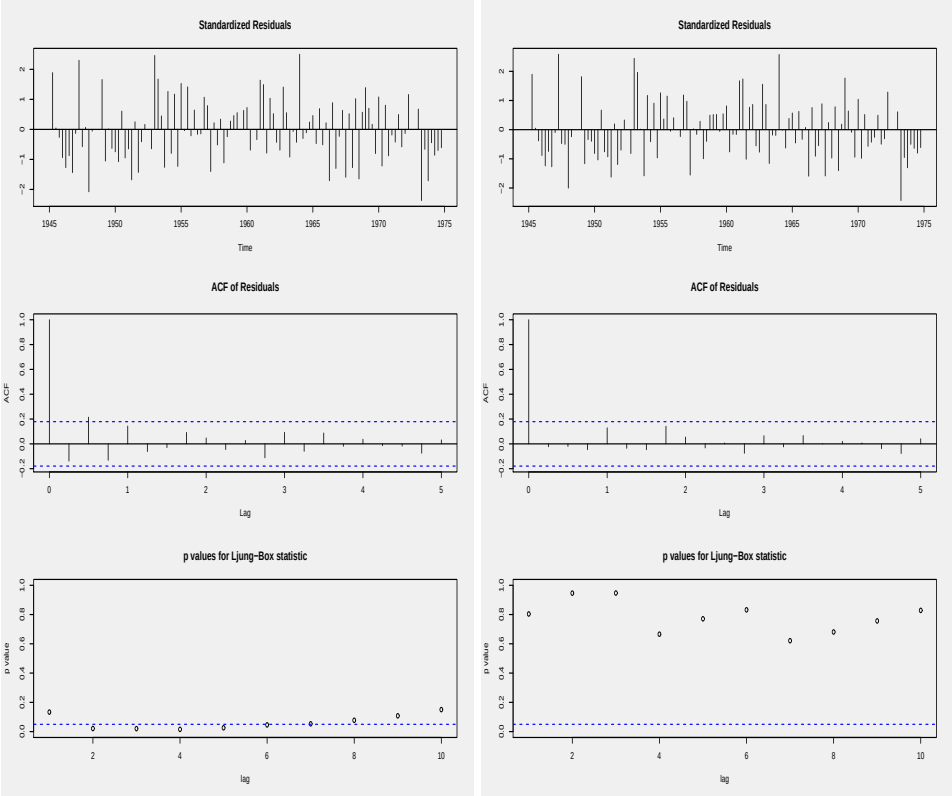
注意：結果は S-PLUS の arima.mle() とは異なる可能性が高い．後者は条件付き尤度を計算し，モデルに平均値を含めない．更に arima.mle() とは MA 係数の符号が異なる．arima() は ARIMA モデル，もしくは欠損値が無い階差モデルに対する arima0() と類似しているが，欠損値を持つ階差モデルを正確に扱う．arima() は arima0() よりも少し遅く，特に季節項を持つ階差モデルに対してそうである．
関連： predict.Arima(), tsdiag(), arima0(), ar().

```
# 黄体形成ホルモン量データ lh を使用
> arima(lh, order = c(1,0,1))                    # ARIMA(1,0,1)=ARMA(1,1) モデルの当てはめ
Call:
arima(x = lh, order = c(1, 0, 1))
Coefficients:
         ar1     ma1  intercept
      0.4522  0.1982     2.4101
s.e.  0.1769  0.1705     0.1358
sigma^2 estimated as 0.1923:  log likelihood = -28.76,  aic = 65.52
```

```
# 米国の月別事故死亡者数 USAccDeaths．季節項 ARIMA(0,1,1) を持つ ARIMA(0,1,1) モデルの当てはめ
> arima(USAccDeaths, order = c(0,1,1), seasonal = list(order=c(0,1,1)))
Call:
arima(x = USAccDeaths, order=c(0,1,1), seasonal=list(order = c(0,1,1)))
Coefficients:
          ma1     sma1
      -0.4303  -0.5528
s.e.   0.1228   0.1784
sigma^2 estimated as 99347:  log likelihood = -425.44,  aic = 856.88
```

```
# ヒューロン湖の水位データ LakeHuron を使用．1920 年からの年数を外部説明変数とする AR(2) モデル
> arima(LakeHuron, order = c(2,0,0), xreg = time(LakeHuron)-1920)
Call:
arima(x = LakeHuron, order = c(2, 0, 0), xreg = time(LakeHuron) - 1920)
Coefficients:
         ar1      ar2  intercept  time(LakeHuron) - 1920
      1.0048  -0.2913   579.0993                 -0.0216
s.e.  0.0976   0.1004     0.2370                  0.0081
sigma^2 estimated as 0.4566:  log likelihood = -101.2,  aic = 212.4
```

```
# 大統領支持率データ presidents を使用 (以下の図を参照)．example(acf) は次数 1 か 3 を示唆
(fit1 <- arima(presidents, c(1, 0, 0)))           # ARIMA(1,0,0)=AR(1) モデル
tsdiag(fit1)                                      # 当てはめ診断図
(fit3 <- arima(presidents, c(3, 0, 0)))           # ARIMA(3,0,0)=AR(3) モデル，AIC 小
tsdiag(fit3)                                      # 当てはめ診断図
```



大統領支持率データへの ARIMA(1,0,0), ARIMA(3,0,0) モデルの当てはめの `tsdiag()` による診断図 (標準化残差，残差の自己相関，Ljung-Box 検定の p 値)

4.5.3 ARMA モデルの自己相関係数 `ARMAacf()`

ARMA モデルの理論自己相関係数もしくは部分自己相関係数を計算する．

書式： `ARMAacf(ar=numeric(0), ma=numeric(0), lag.max=r, pacf=FALSE)`

引数：

`ar` AR 係数の数値ベクトル

`ma` MA 係数の数値ベクトル

`lag.max` 整数．必要とされる最大ラグ．既定値は `max(p,q+1)` で，ここで `p`, `q` はそれぞれ AR,MA 項の数である

`pacf` 論理値．部分自己相関係数を返すべきか？

返り値： (部分) 自己相関係数で，ラグを名前にも持つ

使われる手法は Brockwell & Davis に従う．彼らの式 (3.3.8) がラグ $0, \ldots, \max(p, q+1)$ で自己共分散について解かれ，その他の自己相関係数は再帰的フィルタで得られる．

```
# ARMA(3,1) モデルのラグ 10 までの自己相関
> ARMAacf(c(1.0, -0.25), 1.0, lag.max = 10)
          0           1           2           3           4           5
1.000000000 0.875000000 0.625000000 0.406250000 0.250000000 0.148437500
          6           7           8           9          10
0.085937500 0.048828125 0.027343750 0.015136719 0.008300781
# 答えは 2^(-n)*(32/3+8*n)/(32/3) になるはず
```

```
> n <- 1:10; 2^(-n) * (32/3 + 8 * n) /(32/3)
 [1] 0.875000000 0.625000000 0.406250000 0.250000000 0.148437500 0.085937500
 [7] 0.048828125 0.027343750 0.015136719 0.008300781

> ARMAacf(c(1.0, -0.25), 1.0, lag.max = 10, pacf = TRUE) # 部分自己相関係数
[1]  0.8750000 -0.6000000  0.3750000 -0.2727273  0.2142857 -0.1764706
[7]  0.1500000 -0.1304348  0.1153846 -0.1034483
```

4.5.4 ARMA モデルを無限次数 MA モデルに変換する ARMAtoMA()

ARMAtoMA() は ARMA モデルを無限次数 MA モデルに変換する.

書式： ARMAtoMA(ar = numeric(0), ma = numeric(0), lag.max)

引数：

ar	AR 係数の数値ベクトル
ma	MA 係数の数値ベクトル
lag.max	必要な最大の (無限次数)MA 係数

返り値： 係数のベクトル

```
> ARMAtoMA(c(1.0, -0.25), 1.0, 10)    # ARMA(3,1) モデルの無限次数 MA 版の最初の 10 個の係数
[1] 2.00000000 1.75000000 1.25000000 0.81250000 0.50000000 0.29687500
[7] 0.17187500 0.09765625 0.05468750 0.03027344
```

4.6 時系列に対する検定

4.6.1 時系列の独立性帰無仮説に対する検定 Box.test()

与えられた時系列に対する「独立性帰無仮説」を調べる Box-Pierce もしくは Ljung-Box 検定統計量を計算する. 欠損値は許されない.

書式： Box.test(x, lag = 1, type = c("Box-Pierce","Ljung-Box"))

引数：

x	数値ベクトルもしくは一変量時系列
lag	統計量はラグ lag の自己相関係数に基づく
type	実行される検定名. 先頭の一部だけ指示すれば良い

返り値： クラス "htest" のリストで次の成分を持つ：

statistic	検定統計量の値
parameter	打切ラグパラメータ
p.value	検定の p 値
method	どの検定が使われたかを示す文字列
data.name	データの名前を示す文字列

```
> x <- rnorm (100)                                  # 独立乱数
> Box.test (x, lag = 1)                             # 既定の Box-Pierce 検定
```

```
        Box-Pierce test
data:  x
X-squared = 0.1036, df = 1, p-value = 0.7476      # 帰無仮説は棄却できない

> Box.test (x, lag = 1, type="Ljung")             # Ljung-Box 検定
        Box-Ljung test
data:  x
X-squared = 0.1067, df = 1, p-value = 0.7439      # 帰無仮説は棄却できない
```

4.6.2　Phillips-Perron の単位根検定 PP.test()

x が単位根を持つという帰無仮説を，定常対立仮説に対して検定する Phillips-Perron 検定を実行する．欠損値は許されない．

書式： PP.test(x, lshort = TRUE)

引数：

x　　数値ベクトルもしくは一変量時系列

lshort　論理値で，ラグパラメータを短くするか，長くするかを指示する

返り値：　クラス "htest" のリストで次の成分を持つ：

statistic　検定統計量の値

parameter　打切ラグパラメータ

p.value　検定の p 値

method　どの検定が使われたかを示す文字列

data.name　データの名前を示す文字列

定数項と線形トレンドを含む一般的な回帰式を用い，AR の最初の係数が 1 に等しいという仮説に対する修正 t 統計量が計算される．σ^2 の計算には Newey-West 推定量が使われる．もし lshort=TRUE なら，打ち切りラグパラメータは $\mathrm{trunc}(4(n/100)^{0.25})$ に設定され，さもなければ $\mathrm{trunc}(12(n/100)^{0.25})$ が使われる．p 値は Banerjee et al. の本の 103 頁の表を用いて補間される．

```
> x <- rnorm(1000)                                # 独立乱数
> PP.test(x)                                      # PP 検定，1% 有意
        Phillips-Perron Unit Root Test
data:  x
Dickey-Fuller = -31.9874, Truncation lag parameter = 7, p-value = 0.01

> y <- cumsum(x)                                  # 累積和．単位根を持つ非定常時系列
> PP.test(y)                                      # PP 検定，単位根帰無仮説は棄却できない
        Phillips-Perron Unit Root Test
data:  y
Dickey-Fuller = -1.6208, Truncation lag parameter = 7, p-value = 0.7388
```

4.6.3　時系列解析の診断図 tsdiag()

tsdiag() は時系列の当てはめ結果の診断図を描く総称的関数である．

書式： tsdiag(object, gof.lag, ...)

引数：

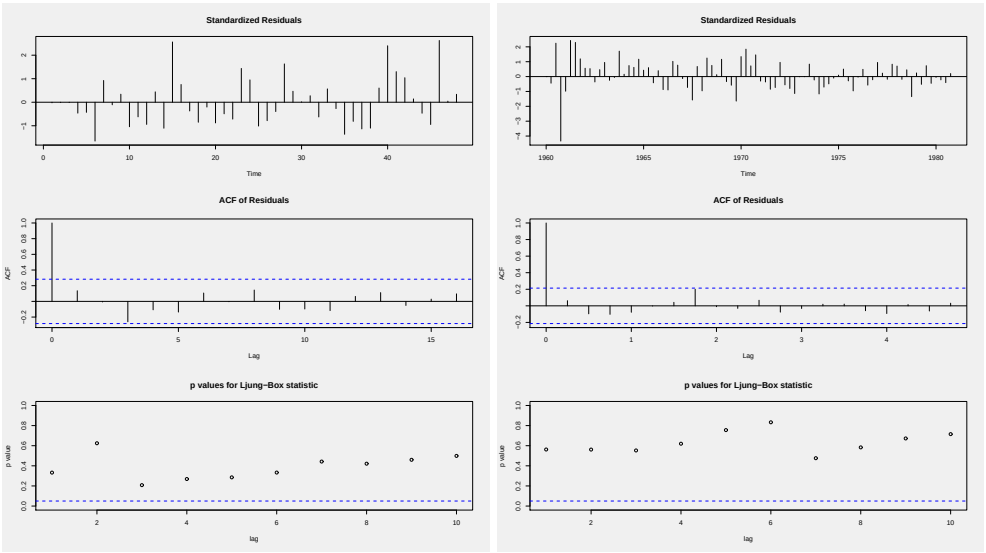
object　当てはめられた時系列

gof.lag　Portmanteau 適合度検定に対する最大ラグ数

...　特定のメソッドに引き渡される追加引き数

これは総称的関数である．普通，残差 (しばしば標準化される)，残差の自己相関，そしてラグ gof.lag までの Portmanteau 検定統計量[*5] に対する p 値をプロットする．arima() と StructTS() オブジェクトに対しては，残差はそれらの (個々の) 分散の推定値でスケール化され，検定統計量としては Ljung-Box 検定が使われる．

```
# 黄体形成ホルモン量データ lh を使用
> fit <- arima(lh, c(1,0,0))                    # ARIMA(1,0,0)=AR(1) モデル当てはめ
> tsdiag(fit)                                    # 当てはめ診断図 (以下の図を参照)

# 企業の利益データ JohnsonJohnson を使用．対数値に構造モデルを当てはめ
> fit <- StructTS(log10(JohnsonJohnson), type="BSM")
> tsdiag(fit)                                    # 当てはめ診断図 (以下の図を参照)
```



時系列の当てはめ診断図．(左) 黄体形成ホルモン量データへの AR(1) モデルの当てはめ．(右) 企業業績データへの構造モデルの当てはめ．

4.7 時系列の成分への分解

4.7.1 移動平均による古典的な時系列の成分への分解 decompose()

時系列を，移動平均により季節，傾向，不規則成分に分解する．加法，乗法モデルをともに扱える．

書式： decompose(x, type=c("additive","multiplicative"), filter=NULL)

[*5] Portmanteau 適合度検定は「かばん」検定と訳されることがある．複数の期間にわたる自己相関の有無をまとめて検定する．時系列解析では Box-Pierce, Ljung-Box 検定が代表的．

引数：

x　　時系列

type　季節成分のタイプ

filter　季節成分を取り除くためのフィルタ係数のベクトルで (AR,MA 係数のように) 時間に関して逆順に並べる． NULL なら，対称なウィンドによる移動平均が使われる

返り値：　クラス "decomposed.ts" のリストで次の成分を持つ：

seasonal　季節成分 (つまり，繰り返される季節パターン)

figure　推定された季節パターンのみ

trend　傾向成分

random　残りの不規則成分

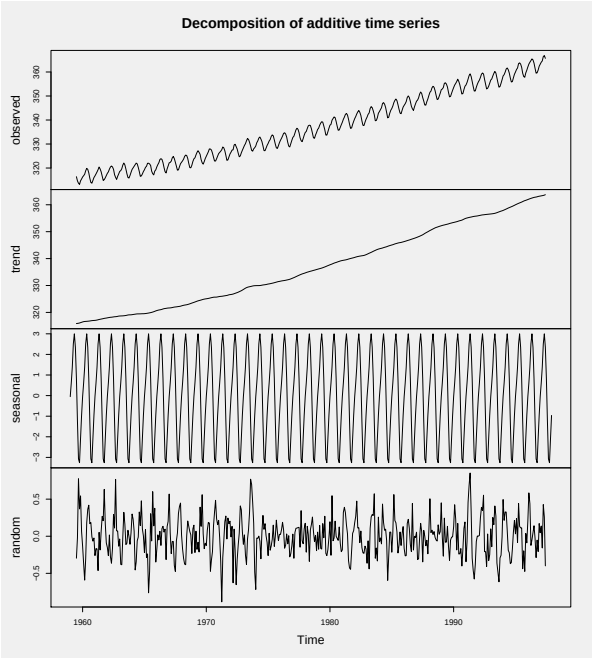
type　type 引き数値

加法，加法モデルはそれぞれ次式で定義される．T_t, S_t, e_t はそれぞれ傾向成分，季節成分，残差である：

$$Y_t = T_t + S_t + e_t, \qquad Y_t = T_t S_t + e_t.$$

注意：関数 stl() はより洗練された分解を与える．

```
# 炭酸ガス濃度データ co2 を使用（以下の図を参照）
> m <- decompose(co2)
> m$figure
[1] -0.05100038  0.61075638  1.38363926  2.50942755  2.99729917  2.34354917
[7]  0.82455143 -1.24613551 -3.07113551 -3.25780218 -2.08016704 -0.96298236
> plot(m)
```



炭酸ガス濃度データの成分への分解

上から，原データ，傾向成分，季節成分，不規則成分

4.7.2 loess を用いた時系列の成分への分解 stl()

loess() 関数を用いた時系列の，季節，傾向，不規則成分への分割 (Seasonal Decomposition of Time Series by Loess)．STL と略される．

書式：

```
stl(x, s.window, s.degree = 0, t.window = NULL, t.degree = 1,
    l.window = nextodd(period), l.degree = t.degree,
    s.jump = ceiling(s.window/10), t.jump = ceiling(t.window/10),
    l.jump = ceiling(l.window/10), robust = FALSE,
    inner = if(robust)  1 else 2,  outer = if(robust) 15 else 0,
    na.action = na.fail)
```

引数：

x　分解すべき一変量時系列．周期が 2 以上のクラス "ts" のオブジェクト

s.window　文字列 "periodic" か，季節成分取り出しのための loess() のウィンド幅 (ラグ単位) で奇数．既定値は無い

s.degree　季節成分取り出しのための局所的当てはめ多項式の次数．0 か 1

t.window　傾向成分を取り出すための loess() 用のウィンドの区間幅 (ラグ単位) で奇数． NULL なら既定値は ceiling((1.5*period)/(1-(1.5/s.window))) 以上の最小の奇数

t.degree　傾向成分取り出しのための局所当てはめ多項式の次数．0 か 1

l.window　各副系列に使われるローパスフィルタの loess() 用のウィンドの区間幅 (ラグ単位)．既定値は frequency(x) 以上の最小奇数で，傾向・季節成分間の競合を防ぐために勧められる．もし奇数でなければ，次に大きい奇数に増やされる

l.degree　副系列用のローパスフィルタの局所的当てはめ多項式の次数．0 か 1

s.jump,t.jump,l.jump　それぞれの平滑操作を加速するための整数で，最低でも 1．各 *.jump 番目の値間で線形補間がされる

robust　論理値で loesss() の適用に当たって頑健な当てはめを使うかどうか指示

inner　整数．「内的 (後退的当てはめ) な」繰り返し数．普通僅か (2 回) の繰り返しで十分

outer　整数．「外的な」頑健性のための繰り返し数

na.action　欠側値に対する処理

返り値：　クラス "stl" のリストで次の成分を持つ：

time.series　列 seasonal, trend, remainder を持つ多変量時系列

weights　最終的な頑健重み (もし当てはめが頑健に行われないならば全て 1)

call　呼出し式

win　"s", "t", "l" 平滑操作に対して使われる区間幅を持つ長さ 3 の整数ベクトル

deg　それぞれの平滑操作に対して使われる多項式の次数を持つ長さ 3 の整数ベクトル

jump　それぞれの平滑操作に対して使われる「ジャンプ」数を持つ長さ 3 の整数ベクトル

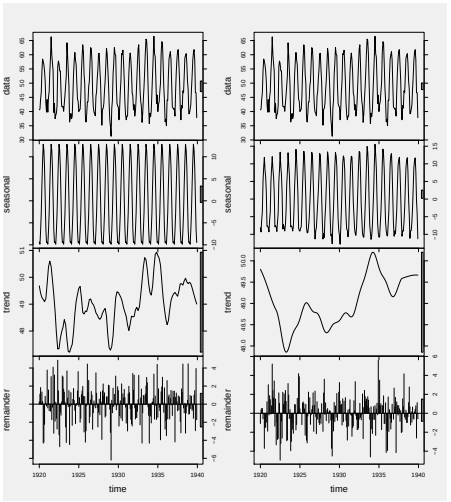
ni　内的な繰りかえし数

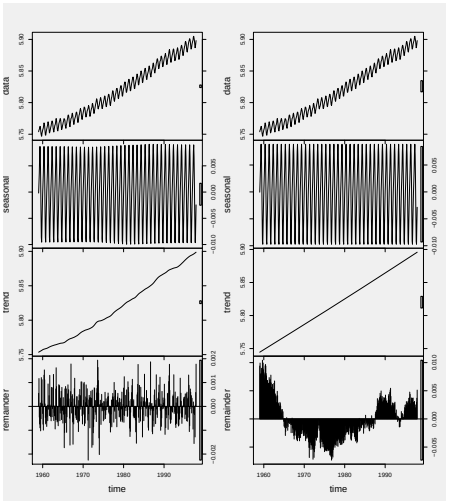
no	外的な繰りかえし数

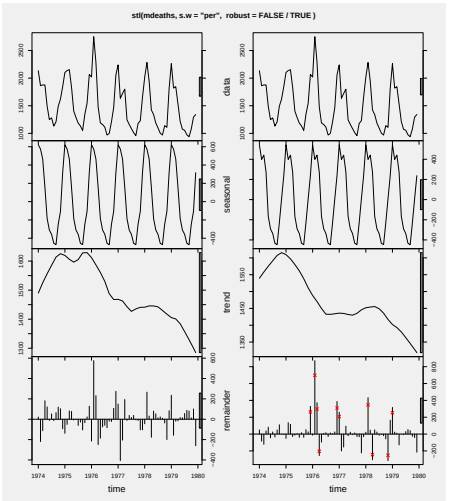
季節成分は季節副系列 (全ての一月の値, 全ての二月の値, ... からなる系列) を loess() で平滑化することにより得られる．もし s.window = "periodic" ならば，平滑化は実際には平均操作で置き換えられる．季節値が取り除かれ，残りが傾向成分を見出すため平滑化される．全体的水準が季節成分から除かれ，傾向成分に加えられる．この過程が数回繰り返される．remainder 成分は季節成分と傾向成分の和からの残差である．結果のクラス "stl" には plot.stl() 等の幾つかのメソッドがある．

注意： これは S-PLUS の stl() 関数に似ているが同一ではない．S-PLUS が与える remainder 成分は，この関数の trend，remainder 成分の和である．
関連： クラス stl オブジェクト用のプリントメソッド関数 plot.stl(), loess() (実際は stl() 中では使われない)，別種の分解を与える StructTS().

```
# ノッチンガム城の気温データ nottem を使用 (以下の図を参照)．図を横に並べるため余白の調整
> op <- par(mar = c(0, 4, 0, 3), oma = c(5, 0, 4, 0), mfcol = c(4, 2))
> plot(stl(nottem, "per"), set.pars=NULL)          # 周期条件で分解
> plot(stl(nottem, s.win = 4, t.win = 50, t.jump = 1), set.pars=NULL)
```







stl() による時系列の成分への分解
(上左) ノッチンガム城の気温データ．上から原データ季節成分，傾向成分，不規則成分．左は周期条件制約
(上右) 炭酸ガス濃度データの対数値．右は線形トレンド，周期制約条件
(下左) 英国の呼吸器疾患死亡者数．右は頑健版

```
# 炭酸ガス濃度データ co2 を使用
> plot(stllc <- stl(log(co2), s.window=21), set.pars=NULL)  # 対数値を分解
> summary(stllc)                                            # 分解結果要約

# 線形トレンド，周期条件
> plot(stl(log(co2), s.window="per", t.window=1000), set.pars=NULL)

> data(UKLungDeaths)                                        # 英国の呼吸器疾患死亡者数
> stmd <- stl(mdeaths, s.w = "per")                         # 頑健でない
> summary(stmR <- stl(mdeaths, s.w = "per", robust = TRUE)) # 頑健版の要約
Call:
stl(x = log(co2), s.window = 21)
Time.series components:
   seasonal                trend            remainder
Min.   :-9.939103e-03   Min.   :5.753541   Min.   :-2.255405e-03
1st Qu.:-4.536535e-03   1st Qu.:5.778556   1st Qu.:-4.586796e-04
Median : 8.777611e-04   Median :5.815125   Median :-8.867395e-06
Mean   :-1.304469e-06   Mean   :5.819267   Mean   :-1.965549e-06
3rd Qu.: 4.997747e-03   3rd Qu.:5.859806   3rd Qu.: 4.023465e-04
Max.   : 9.114691e-03   Max.   :5.898750   Max.   : 1.939626e-03
(途中省略)
> plot(stmd, set.pars=NULL, labels = NULL,
       main = "stl(mdeaths, s.w = \"per\",  robust = FALSE / TRUE )")
> plot(stmR, set.pars=NULL)
> iO <- which(stmR $ weights  < 1e-8)                    # 外れ値をマーク（外れ値は 10 個）
> sts <- stmR$time.series
> points(time(sts)[iO], .8* sts[,"remainder"][iO], pch = 4, col = "red")
```

4.8 フィルタリング，平滑化

4.8.1 線形フィルタ filter()

一変量時系列，もしくは多変量時系列の各時系列に線形フィルタを適用する．返り値は時系列オブジェクトである．

書式： `filter(x, filter, method = c("convolution", "recursive"), sides = 2, circular = FALSE, init)`

引数：

`x` 一変量もしくは多変量時系列

`filter` フィルタ係数のベクトルで (AR,MA 係数のように) 時間に関して逆順に並ぶ

`method` `"convolution"` か `"recursive"` (短縮省略可能)．もし `"convolution"` なら移動平均が使われる．もし `"recursive"` なら自己回帰が使われる

`sides` 畳み込みフィルタだけに関係する．もし `sides=1` ならフィルタ係数は過去の値にだけ使われる．もし `sides=2` なら，係数はラグ 0 の周りに中心化される．この場合フィルタの長さは奇数でなければならない．もし偶数なら，フィルタは時間に関して過去よりもより未来に作用する

`circular` 畳み込みフィルタだけに関係する． `TRUE` ならフィルタは時系列の終端で始端に巻き返される．さもなければ，外れた値は欠損値とされる

`init` 再帰的フィルタだけに関係する．時系列の始点に先立つ初期値を，逆時間順に指定する．既定値は全て 0

`x` 中の欠損値は許されるが，`filter` 中の欠損値は許されない (もしあれば出力中に欠損値が頻発することになる)．再帰的フィルタは次式で与えられるが，ラグ 0 に暗黙の係

数 1 があることを注意しよう：

$$y_i = x_i + f_1 y_{i-1} + \ldots + f_p y_{i-p}.$$

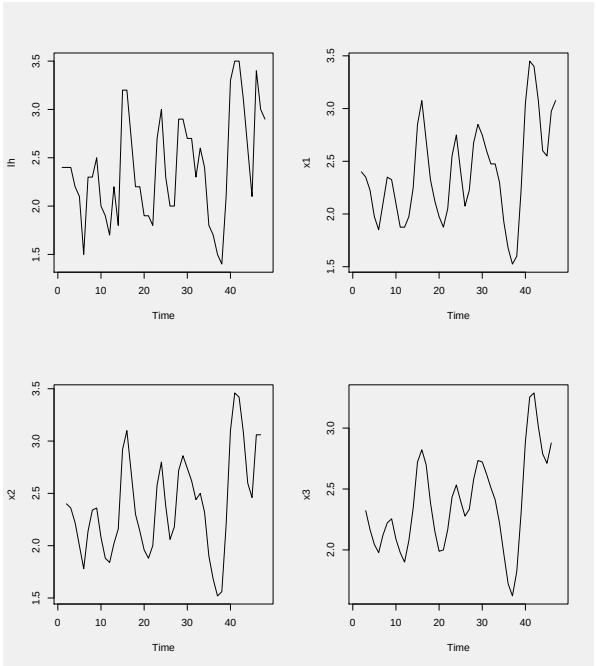
再帰的フィルタが可逆かどうかのチェックはされない．もし可逆でなければ出力は発散するかも知れない．畳み込みフィルタは次式で定義される：

$$y_i = f_1 x_{i+o} + \ldots + f_p x_{i+o-p-1}$$

ここで，o はオフセットである．これがどう決まるかは `sides` を見よ．

注意：関数 `convolve(, type="filter")` は計算に FFT を用い，従って長い一変量時系列に対してはより早いかも知れないが，時系列オブジェクトを返さない (時間の順序が不明瞭になる) し，欠損値も処理できない．例えば，`filter` は長さ 1000 の時系列に長さ 100 のフィルタを操作する場合はより早い．

```
> par(mfrow=c(2,2))
> data(lh)                                    # 黄体形成ホルモン量データ読み込み
> plot.ts(lh)
> x1 <-  filter(lh, c(1,2,1)/4)               # 畳み込みフィルタ
> plot.ts(x1)
> x2 <-  filter(lh, c(1,3,1)/5)               # 畳み込みフィルタ
> plot.ts(x2)
> x3 <-  filter(lh, c(1,2,3,2,1)/9)           # 畳み込みフィルタ
> plot.ts(x3)
```



黄体形成ホルモン量データに対する線形畳み込みフィルタ．原データ (上左) とフィルタ係数を変えた結果

4.8.2　各関数による平滑化 kernapply()

入力時系列の，特定の核関数による畳み込みによる平滑化を計算する．返り値は平滑化された結果の系列である．

書式：

```
kernapply(x, k, circular = FALSE, ...)
kernapply(k1, k2)
```

引数：

k,k1,k2　クラス "tskernel" の平滑化用核関数オブジェクト

x　　平滑化されるベクトル，行列，もしくは時系列

circular　論理値．入力系列を巡回的 (つまり，周期的) に扱うべきか？

...　他のメソッドへ (から) 引き渡される追加引き数

注意：例は kernel() を参照．

4.8.3 平滑化核関数オブジェクト kernel()

クラス "tskernel" は，離散的で対称な正規化された平滑化核関数を表現するようにデザインされている．これらの核関数は，ベクトル，行列，時系列オブジェクトを平滑化するのに使われる．

```
kernel(coef, m, r, name)
df.kernel(k)
bandwidth.kernel(k)
is.tskernel(k)
```

引数：

coef　平滑化核関数係数の上側半分 (0 位置の係数を含む)，もしくは核関数の名前．現在の所 "daniell", "dirichlet", "fejer" そして "modified.daniell"

m　　核関数の次元．核関数係数の数は 2*m+1 になる

name　核関数の名前

r　　Fejer 核関数の次数

k　　クラス "tskernel" のオブジェクト

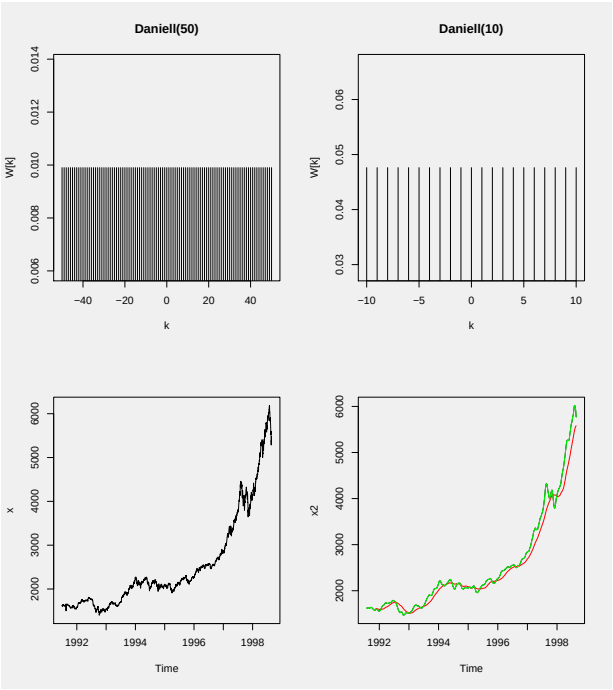
返り値：　kernel() はクラス "tskernel" のリストを返し，成分として係数 coef と核関数次元 m を持つ．追加の属性として"name" を持つ

kernel() は一般的な核関数や，名前付きの特定の核関数を構成するのに使われる．修正 Daniell 関数は (S-PLUS と同様に) 終端係数を持つ．df.kernel() は Brockwell & Davies で定義されているような核関数の「同値自由度 (equivalent degrees of freedom)」を返す．bandwidth.kernel() は連続補正された Bloomfield で定義されたような同値なバンド幅を返す．

```
# ヨーロッパ株価指標データ EuStockMarkets を使用
> k1 <- kernel("daniell", 50)                    # 長期の移動平均
> k2 <- kernel("daniell", 10)                    # 短期版
> plot(k1)                                       # k1 核関数の係数プロット
> plot(k2)                                       # k2 核関数の係数プロット
> x <- EuStockMarkets[,1]                        # そのうちのドイツの株価指数 DAX 系列
> x1 <- kernapply(x, k1)                         # ドイツ株価指数の核関数平滑化
> x2 <- kernapply(x, k2)                         # ドイツ株価指数の核関数平滑化
```

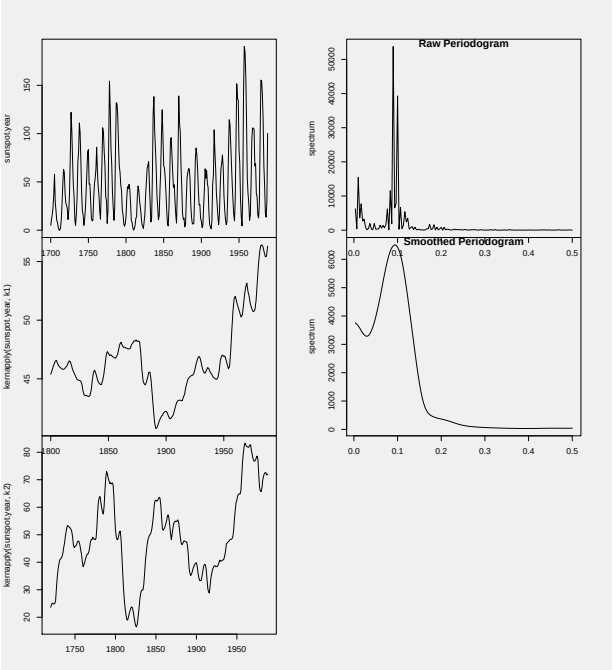
```
> plot(x)                                          # 原時系列プロット
> plot(x2)                                         # x2 系列をまずプロット
> lines(x1, col = "red")                           # x1 系列を赤の折れ線で上書き
> lines(x2, col = "green")                         # 次に x2 系列を緑の折れ線で上書き
```

```
# 太陽黒点数データ sunspots を Daniel 核関数で平滑化した結果のスペクトル
> ts.plot(sunspot.year, title = """)
> ts.plot(kernapply(sunspot.year, k1), main = "")
> ts.plot(kernapply(sunspot.year, k2), main = "")
> spectrum(sunspot.year, log="no", main = "")
> spectrum(sunspot.year, kernel=kernel("daniell",c(11,7,3)),log="no",main="")
```



ヨーロッパ株価指標データ EuStockMarkets の Daniel 核関数による畳み込みフィルタ平滑化

長期版 (上左)
短期版 (上右)
ドイツの株価指数 DAX 系列
長期版で平滑化 (下左)
短期版で平滑化 (下右)



太陽黒点数データを Daniel 核関数で平滑化

原データ (上左)
長期平滑化 (中左)
短期平滑化 (下左)
原データのスペクトル密度 (上右)
平滑化から計算したスペクトル密度 (中右)

4.8.4 Holt-Winters フィルタリング HoltWinters()

HoltWinters() は時系列に対する Holt-Winters フィルタリングを計算する．未知パラメータは 2 乗予測誤差の最小化で決定される．

書式：

```
HoltWinters(x, alpha = NULL, beta = NULL, gamma = NULL,
            seasonal = c("additive", "multiplicative"),
            start.periods = 3, l.start = NULL, b.start = NULL,
            s.start = NULL,
            optim.start = c(alpha = 0.3, beta = 0.1, gamma = 0.1),
            optim.control = list())
```

引数：

x　クラス "ts" のオブジェクト

alpha　Holt-Winters フィルタの α パラメータ

beta　Holt-Winters フィルタの β パラメータ．0 とすると指数関数平滑化

gamma　季節成分に使われる γ パラメータ．0 とすると季節成分は無い

seasonal　季節モデルを選択する文字列で，"additive" (既定) または "multiplicative"．最初の数文字で十分．$\gamma = 0$ でないときだけ意味がある

start.periods　開始値の自動検出で使われる開始時点．3 以上でなければならない

l.start　水準に対する開始値 (α_0)

b.start　傾向に対する開始値 (β_0)

s.start　季節成分 (s1,...,sp) に対する開始値のベクトル

optim.start　名前 alpha, beta そして gamma を持つ成分を持つリストで，最適過程の初期値を含む．必要な値だけ指定必要がある

optim.control　optim() に引き渡される追加制御パラメータを持つオプションのリスト

返り値：　クラス "HoltWinters" のリストで次の成分を持つ：

fitted　フィルタリングされた系列とともに，傾向と季節系列を各列に持つ多変量時系列で，経時的 (つまり時系列の最後ではなく各時間 t 毎) に推定されている

x　原時系列

alpha　フィルタリングに使われる α パラメータ値

beta　フィルタリングに使われる β パラメータ値

coefficients　名前 a, b, s1,..., sp 付き要素を持つベクトルで，水準，傾向，そして季節成分の推定値を含む

seasonal　指定された seasonal パラメータ

SSE　最適化過程で達成された最終的な 2 乗誤差和の値

call　用いられた呼出し式

加法的な Holt-Winters 予測関数 (周期長 p の時系列用) は

$$\begin{aligned}\hat{Y}_{t+h} &= a_t + hb_t + s_{t+1+(h-1) \pmod p},\\ a_t &= \alpha(Y_t - s_{t-p}) + (1-\alpha)(a_{t-1} + b_{t-1}),\\ b_t &= \beta(a_t - a_{t-1}) + (1-\beta)b_{t-1},\\ s_t &= \gamma(Y_t - a_t) + (1-\gamma)s_{t-p}.\end{aligned}$$

乗法的な Holt-Winters 予測関数 (周期長 p の時系列用) は

$$\begin{aligned}\hat{Y}_{t+h} &= (a_t + hb_t)s_{t+1+(h-1) \pmod p},\\ a_t &= \alpha(Y_t / s_{t-p}) + (1-\alpha)(a_{t-1} + b_{t-1}),\\ b_t &= \beta(a_t - a_{t-1}) + (1-\beta)b_{t-1},\\ s_t &= \gamma(Y_t / a_t) + (1-\gamma)s_{t-p}.\end{aligned}$$

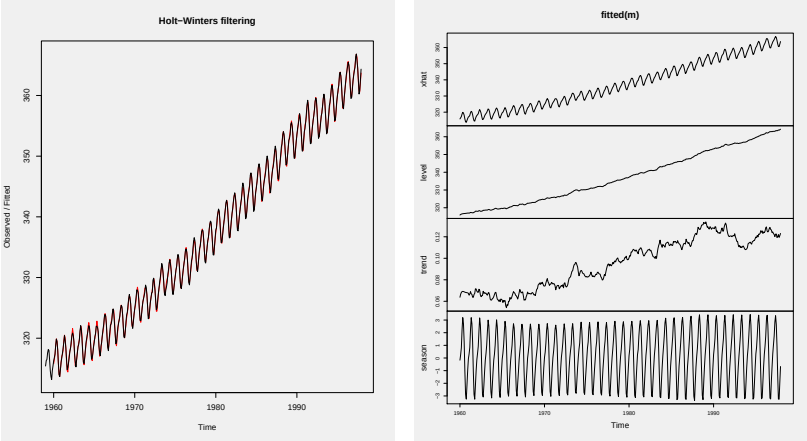
関数は，もしパラメータ α, β, γ のどれかが与えられないと，一時点先の予測誤差を最小にすることにより，それらの最適値を探す．

季節モデルに対しては，a, b そして s の初期値は，最初の start.periods 個の周期に対して移動平均 (関数 decompose() を見よ) を用いることにより，単純な傾向と季節成分への分解を行うことで見出される (水準と傾向に対しては，傾向成分に対する単純な線形回帰が用いられる)．(季節成分を持たない) 水準／傾向モデルに対しては，a, b そして s の初期値は，それぞれ x_2 と $x_2 - x_1$ である．水準だけのモデル (通常の指数型平滑化) に対しては，a の初期値は x_1 である．

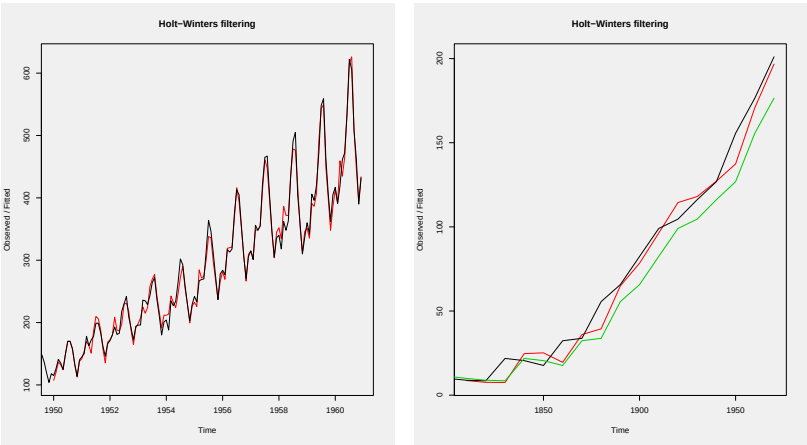
```
# 炭酸ガス濃度データ co2 を用いる (以下の図を参照)．季節的 (で加法的) な Holt-Winters モデル
> m <- HoltWinters(co2)
> plot(m)                                          # 結果のプロット
> plot(fitted(m))                                  # その副系列のプロット
```

```
# 国際航空路線総顧客数データ AirPassengers を用いる (以下の図を参照)
# 季節的 (で乗法的) な Holt-Winters モデル
> m <- HoltWinters(AirPassengers, seasonal="mult")
> plot(m)                                           # 結果のプロット
```

```
# 国勢調査による米国人口 uspop を用いる (以下の図を参照)．季節成分を持たない Holt-Winters モデル
> x <- uspop + rnorm(uspop, sd = 5)                # 標準偏差 5 の正規誤差を加える
> m <- HoltWinters(x, gamma = 0)                   # 季節成分無しを指定
> plot(m)                                          # 原データ，結果の同時プロット
> m2 <- HoltWinters(x, gamma = 0, beta = 0)        # 指数型平滑化
> lines(fitted(m2)[,1], col = 3)                   # 指数型平滑化系列を上書き
```



炭酸ガス濃度データ．(左) 季節的 (で加法的) な Holt-Winters モデルによるフィルタリング系列．(右) その副系列 (上からフィルタリング，水準，傾向，季節成分)



(左) 国際航空路線総顧客数データ．季節的 (で乗法的) な Holt-Winters モデルによるフィルタリング結果．(右) 国勢調査による米国人口データ．季節成分無し加法的 Holt-Winters モデルによる原データ，フィルタリング，水準系列

4.9 時系列関連の作図関数

総称的プロット関数 `plot()` は以下のように，時系列オブジェクトに関する多くのメソッド関数を持つ．通常これらは総称的関数 `plot()` に隠されており，特に特殊な指定を必要としない限り，ユーザーが個別に使う必要は無い．したがって，ここでは個別の紹介はしないことにする．

```
> methods(plot)
 [1] plot.HoltWinters*   plot.POSIXct        plot.POSIXlt
 [4] plot.TukeyHSD       plot.acf*           plot.data.frame
 [7] plot.decomposed.ts* plot.default        plot.dendrogram*
[10] plot.density        plot.factor         plot.formula
[13] plot.function       plot.hclust*        plot.histogram
[16] plot.isoreg*        plot.lm             plot.mlm
[19] plot.ppr*           plot.prcomp*        plot.princomp*
[22] plot.profile.nls*   plot.spec           plot.spec.coherency
[25] plot.spec.phase     plot.spec1*         plot.stl*
[28] plot.table          plot.ts             plot.tskernel*
     Non-visible functions are asterisked
```

4.9.1　時系列のラグプロット lag.plot()

時系列をそのラグ版に対してプロットする．自己相関が低い場合でも，「自己従属性を」を可視化するのに役立つ．

書式：

```
lag.plot(x, lags = 1, layout = NULL, set.lags = 1:lags,
         main = NULL, asp = 1,
         font.main=par("font.main"), cex.main=par("cex.main"),
         diag = TRUE, diag.col="gray", type="p", oma =NULL,
         ask = NULL, do.lines = n <= 150, labels = do.lines, ...)
```

引数：

x　一 (多) 変量時系列

lags　必要とするラグプロットの数．引数 set.lags を参照

layout　多重プロットのレイアウト．典型的には par() の mfrow 引数．既定では全てのプロットが一枚に収まる正方形レイアウト (関数 n2mfrow() を利用)

set.lags　正整数のベクトルで，使われるラグの集まりの指定を可能にする．既定値は 1:lags

main　各図表のトップに置かれる主ヘッダータイトルの文字列

asp　縦横比 (plot.default() を見よ)

font.main,cex.main　タイトルに対する属性 (par() を見よ)

diag　論理値．対角線 x=y を引くか？

diag.col　対角線を引く場合のその色指定

type　使われるプロットの種類．意味に付いては plot.ts() を参照

oma　周辺余白 (par() を参照)

ask　論理値．もし真なら，新しい画面を始める前に問い合わせがある

do.lines　論理値で，線を引くかどうかを指定する

labels　論理値で，ラベルを描くかどうかを指定する

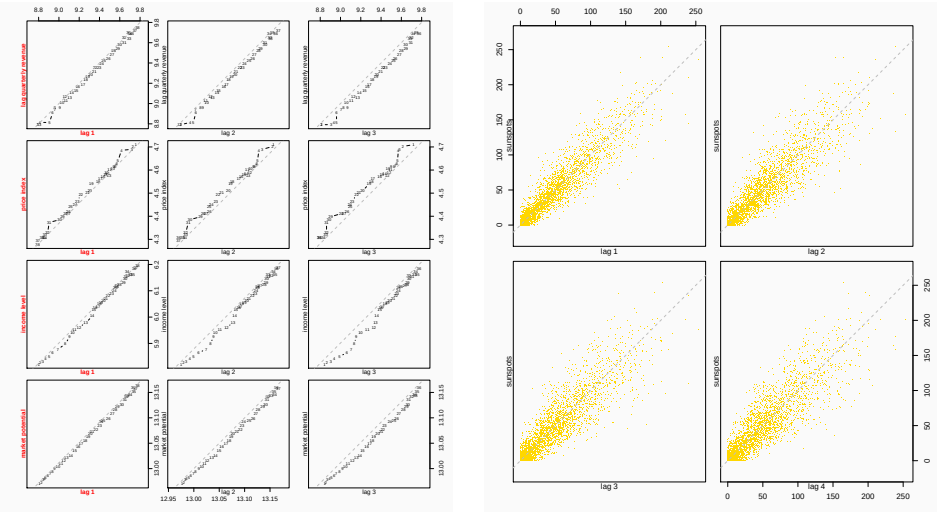
...　plot.ts() に対する追加引数

これは S 版よりもより柔軟で，異なった既定動作を持つ．内部的な一貫性から head = の代わりに main = を使う．実際の作業を行う plot.ts() 関数も参考にしよう．

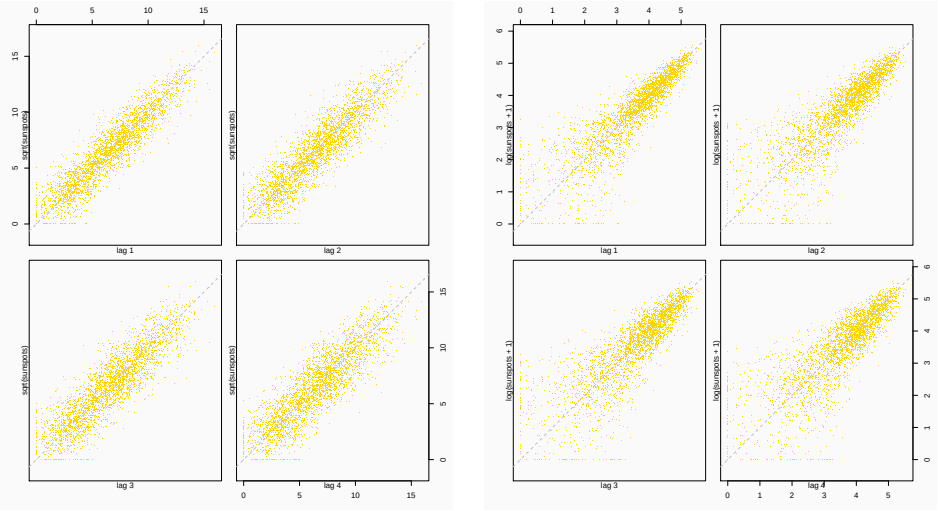
```
# New Haven の平均気温データ nhtemp を使用 (以下の図を参照)
> lag.plot(nhtemp, 8, diag.col = "forest green")
> lag.plot(nhtemp, 5, main="Average Temperatures in New Haven")

# 複数画面の場合は切替える前に問い合わせをする (以下の図を参照)
> lag.plot(nhtemp, 6, layout = c(2,1), asp = NA,
           main = "New Haven Temperatures", col.main = "blue")
```

```
# 四半期毎の歳入時系列 (4 変量時系列)freeny を使用 (以下の図を参照)
> lag.plot(freeny.x, lag = 3)                          # 多変量時系列 (しかし非定常)
# 太陽黒点数データ sunspots を使用．長いので線を引かない (以下の図を参照)
> lag.plot(sunspots,set=1:4,pch=".",col="gold")        # 原データ
> lag.plot(sqrt(sunspots),set=1:4,pch=".",col="gold")  # 平方根を取る
```

```
# 対数を取る（0 データがあるので 1 を加えておく）
> lag.plot(log(sunspots+1), set = 1:4, pch = ".", col = "gold")
```



ラグプロット．四半期毎の歳入データ (左，4 系列毎にラグ 3 まで)．太陽黒点数データ (右，原データ，ラグ 4 まで)．



ラグプロット．太陽黒点数データ (左，平方根変換，ラグ 4 まで)．太陽黒点数データ (右，対数変換，ラグ 4 まで)

4.9.2 時系列の季節成分等をプロットする monthplot()

時系列の季節成分 (または，他の副系列) をプロットする．各季節 (もしくは他のカテゴリ) 毎に時系列がプロットされる．現在の作図デバイスにプロットが行われるが，返り値は無い．

書式：monthplot(x, labels=NULL, times, phase, base, choice, ...)

引数：

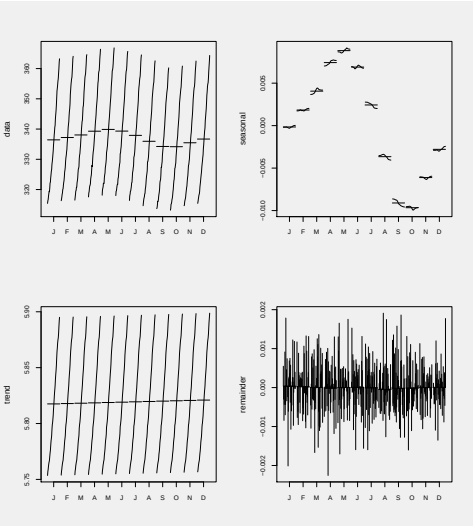
x	時系列，または関連するオブジェクト
labels	各「季節」に対するラベル
times	各観測値に対する時間

phase　各「季節」に対する指示
base　副系列の参考線に対して使われる関数
choice　stl または StructTS オブジェクトのどの副系列か？
...　作図パラメータ

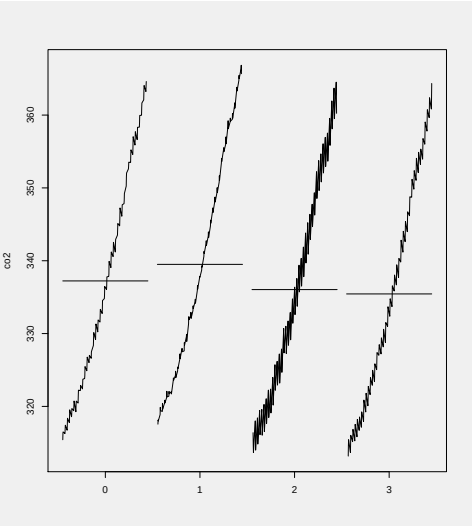
この関数は時系列から副系列を取り出し，それを一つの画面にプロットする．ts(), stl() そして StructTS() 用のメソッドは内部に記録された周波数，開始，終了時間をスケールと季節を設定するために使う．既定のメソッドは，観測値が 12 個毎のグループになっていると仮定するが，これは変更可能である．

labels が与えられないが，phase が与えられていると，labels の既定値は phase 中のユニークな値となる．もし双方が与えられると，phase 値は labels 配列中の添字と仮定される．つまり，それらは 1 から length(labels) の範囲中にあるべきである．

```
# 炭酸ガス濃度データ co2 を使用．対数値系列の stl() 関数による分解
> fit <- stl(log(co2), s.window = 20, t.window = 20)
> par(mfrow = c(2,2))                              # 画面を 2x2 分割

# 原データ，季節，傾向，残差系列を月別にプロット
> monthplot(co2, ylab = "data", cex.axis = 0.8)
> monthplot(fit, choice = "seasonal", cex.axis = 0.8)
> monthplot(fit, choice = "trend", cex.axis = 0.8)
> monthplot(fit, choice = "remainder", type = "h", cex.axis = 0.8)
> (quarter <- (cycle(co2) - 1) %/% 3)              # 12 ヵ月を四半期毎にラベル付け
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1959   0   0   0   1   1   1   2   2   2   3   3   3
1960   0   0   0   1   1   1   2   2   2   3   3   3
(以下省略)

> monthplot(co2, phase = quarter)                  # 四半期毎にプロット
```



炭酸ガス濃度の対数値系列の副系列のプロット．原データ (左上)，季節系列 (右上)，傾向系列 (左下)，残差系列 (右下)．四半期毎の系列．横線は副系列の平均値位置を示す



原データ (左上)，季節系列 (右上)，傾向系列 (左下)，残差系列 (右下)．四半期毎の系列．横線は副系列の平均値位置を示す

4.9.3 複数の時系列を一つの画面にプロット ts.plot()

複数の時系列を一つの画面にプロットする．plot.ts() とは異なり，各系列は異なった時間情報を持っても良いが，周波数は同じでなければならない．一つの時系列に対しても使えるが plot() の使用が簡単で好ましい．

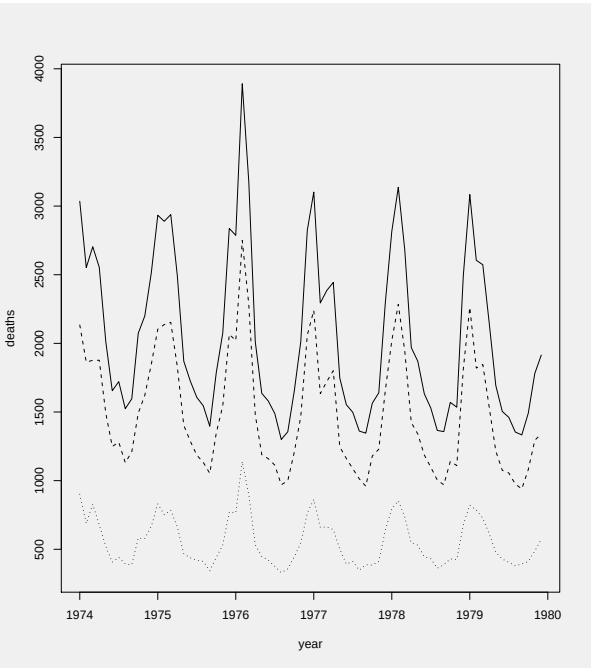
書式： ts.plot(..., gpars = list())

引数：

... 一つ以上の一変量時系列，もしくは多変量時系列

gpars プロット関数に渡される名前付きの作図パラメータ．普通使われるものを直接 ... に置くことができる

```
# 英国の呼吸器疾患死亡者数 UKLungDeaths を使用（以下の図を参照）
> ts.plot(ldeaths, mdeaths, fdeaths,              # 両性，男性，女性死亡者数を同時プロット
          gpars=list(xlab="year", ylab="deaths", lty=c(1:3)))
```



英国の呼吸器疾患死亡者数
上から，両性，男性，女性死亡者数

4.10　時系列の予測

予測用の総称的関数 predict() 関数は以下に示すように，時系列オブジェクトに関するメソッド関数を多数持つ．通常これらは総称的関数 predict() に隠されており，特に特殊な指定を必要としない限り，ユーザーが個別に使う必要は無い．

```
> methods("predict")                          # 現在システムに存在する predict() 関数のメソッド
 [1] predict.Arima*              predict.HoltWinters*
 [3] predict.StructTS*           predict.ar*
 [5] predict.arima0*             predict.glm
 [7] predict.lm                  predict.loess*
 [9] predict.mlm                 predict.nls*
[11] predict.poly                predict.ppr*
[13] predict.princomp*           predict.smooth.spline*
[15] predict.smooth.spline.fit*
    Non-visible functions are asterisked
```

4.11 状態空間モデルとカルマンフィルタ

4.11.1 時系列の構造モデル StructTS()

時系列に構造モデル (structual model) を当てはめる．

書式： `StructTS(x, type = c("level", "trend", "BSM"), init = NULL, fixed = NULL, optim.control = NULL)`

引数：

`x` 一変量数値時系列．欠損値があっても良い

`type` 構造モデルのクラス．もし与えられないと，BSM が `frequency(x)>1` の時系列に，さもなければ，局所傾向モデルが当てはめられる

`init` 分散パラメータの初期値

`fixed` パラメータの総数と同じ長さのオプションの数値ベクトル．もし与えられると，`fixed` 中の `NA` でない項目だけが変化する．おそらく最も有用な使い途は，分散を 0 と設定すること

`optim.control` `optim()` に対する制御パラメータのリスト． `"L-BFGS-B"` 法使用

返り値： クラス `"StructTS"` のリストで，次の成分を持つ：

`coef` 成分の推定分散値

`loglik` 最大対数尤度値．これらの全てのモデルは非定常であり，これは幾つかの観測値に対する拡散プライア (diffuse prior) を含み，従って `arima()` や異なったタイプの構造モデルと比較はできない

`data` 時系列 `x`

`residuals` 標準化残差

`fitted` 多変量時系列で，水準，傾き，季節成分に対する成分を持つ．これらは経時的 (時系列の最後ではなく，時間 `t` 毎に) に推定される

`call` 呼出し式

`series` 時系列 `x` の名前

`convergence` `optim()` が返した値

`model,model0` 当てはめで使われたカルマンフィルタを表すリスト． `KalmanLike()` を見よ． `model0` がフィルタの初期値であり， `model` が最終状態

`xtsp` `x` の `tsp` 属性

構造時系列モデルは，時系列の幾つかの成分への分解に基づく，一次元時系列の (線形ガウシアン) 状態空間モデルである．これは，幾つかの誤差分散により指定され，そのうちの幾つかは 0 であっても良い．

最も簡単なモデルは局所水準モデルで，`type = "level"` により指定される．これは次式で展開する背景水準 m_t を持つ：

$$m_{t+1} = m_t + \xi_t, \quad \xi_t \sim \mathrm{N}(0, \sigma^2\xi).$$

観測値のモデルは次式で与えられる：

$$x_t = m_t + \varepsilon_t, \quad \varepsilon_t \sim \mathrm{N}(0, \sigma^2\varepsilon).$$

二つのパラメータ $\sigma^2\xi$ と $\sigma^2\varepsilon$ がある．これは ARIMA(0,1,1) モデルであるが，パラメータセットに制約がある．

局所線形傾向モデル `type = "trend"` は同じ測定方程式を持つが，m_t 系列に次式 (3 つの分散パラメータを持つ) のように時間とともに変化する傾きを持つ：

$$m_{t+1} = m_t + n_t + \xi_t, \quad \xi_t \sim \mathrm{N}(0, \sigma_\xi^2),$$
$$n_{t+1} = n_t + \zeta_t, \quad \zeta_t \sim \mathrm{N}(0, \sigma_\zeta^2).$$

$\sigma_\zeta^2 = 0$ (局所水準モデルに帰着) や滑らかな傾向を意味する $\sigma_\xi^2 = 0$ に成ることも珍しくない．これは制約された ARIMA(0,2,2) モデルである．

基本の構造モデル `type = "BSM"` は追加の季節成分を持つ局所傾向モデルである．従って計測方程式は

$$x_t = m_t + s_t + \varepsilon_t, \quad \varepsilon_t \sim \mathrm{N}(0, \sigma_\varepsilon^2)$$

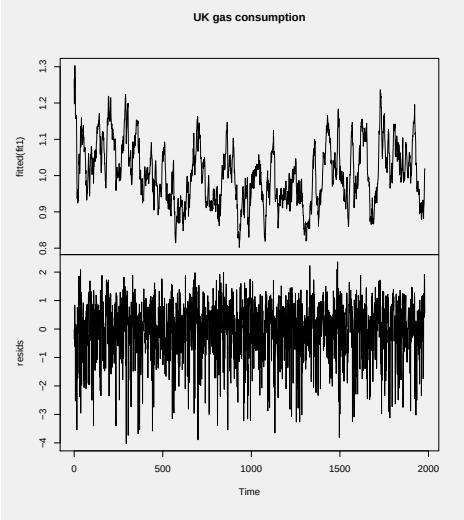
となり，ここで s_t は次の式に従い変化する季節成分である：

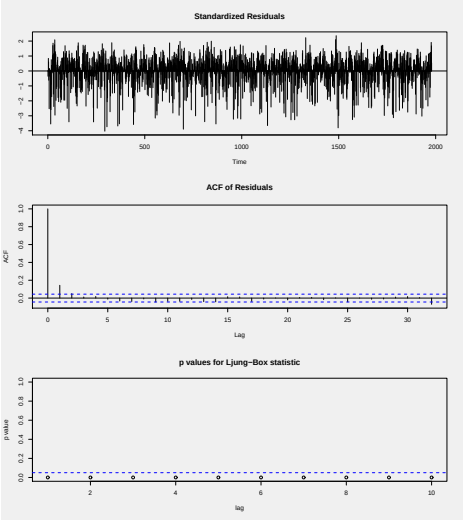
$$s_{t+1} = -s_t - \ldots - s_{t-s+2} + w_t, \quad w_t \sim \mathrm{N}(0, \sigma_w^2).$$

境界ケース $\sigma_w^2 = 0$ は非ランダム (しかし任意の) 季節パターンに相当する．(これはしばしば BSM の「ダミー変数」版と呼ばれる．)

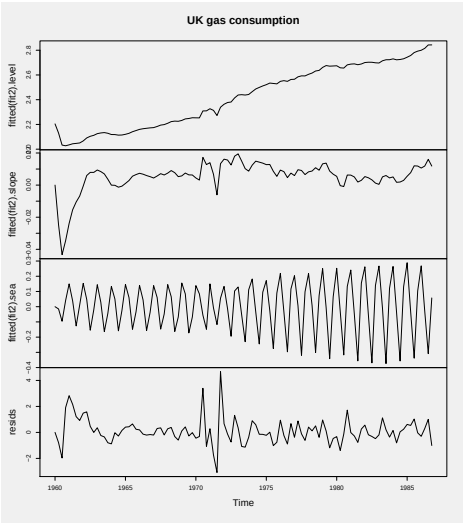
注意：構造モデルの当てはめは，多くの参考文献がそう述べているよりはるかに困難である．例えば，Brockwell & Davis は `AirPassengers` データを議論しているが，彼らの解は局所的最大解のように思われ，`StructTS()` が与える当てはめ程は良くは無い．一つもしくは複数の変数が 0 である当てはめを見出すことはごく普通であり，これは σ_ε^2 も例外ではない．

```
# 木の年輪幅データ treering を使用 (以下の図を参照)
> trees <- window(treering, start=0)              # 紀元後のデータだけ取り出す
> fit1 <- StructTS(trees, type = "level")         # 構造モデル当てはめ
> tsdiag(fit1)                                    # 年輪データ当てはめの診断図
> plot(cbind(fitted(fit1), resids=resid(fit1)), main = "treering")

# 英国のガス消費量データ UKgas を使用 (以下の図を参照)
> fit2 <- StructTS(log10(UKgas), type = "BSM")    # 構造モデル当てはめ
> plot(cbind(fitted(fit2), resids=resid(fit2)), main = "UK gas consumption")
```





(左) 年輪データへの構造モデル当てはめ結果．上から水準，残差系列．(右) 年輪データへの構造モデル当てはめ結果の診断図



(下左) 英国ガス消費量への構造モデルの当てはめ結果．上から，水準，傾き，周期，残差系列

4.11.2 固定区間平滑化 tsSmooth()

状態空間モデルを用い，一変量時系列に固定区間平滑化を実行する．固定区間平滑化は，全ての観測値に基づき，各時点での最良の状態の推定を与える．

書式： tsSmooth(object, ...)

引数：

object　当てはめ時系列．現在クラス "StructTS" のオブジェクトだけがサポートされている

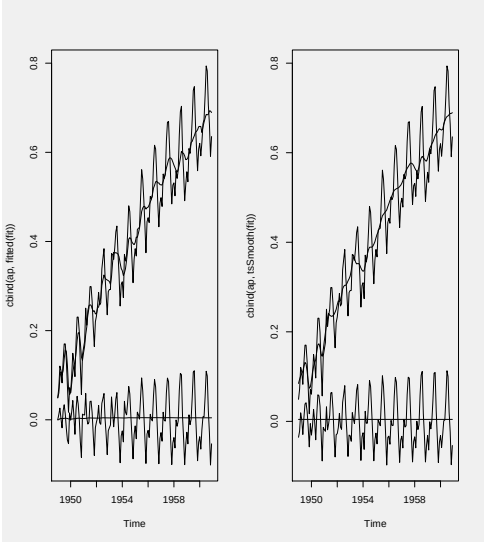
...　将来のメソッドへの可能な引き数

返り値：　一次元時系列で，原時系列の状態空間と各時点での結果に等しい次元を持つ．(季節モデルに対しては，現在の季節成分だけが返される)

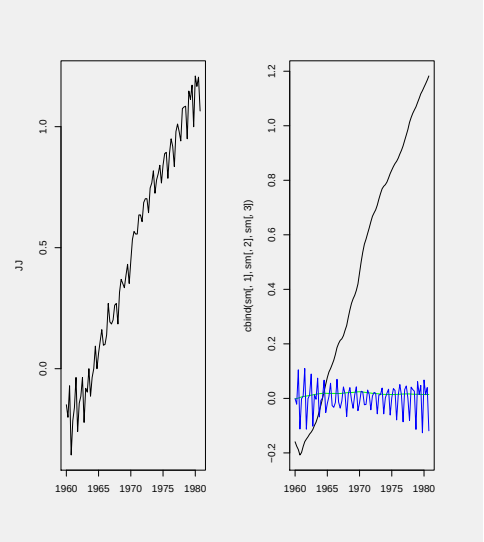
```
# 国際航空路線総顧客数データ AirPassengers を使用（以下の図を参照）
```

```
> ap <- log10(AirPassengers) - 2
> fit1 <- StructTS(ap, type= "BSM")                    # 構造モデルの当てはめ
> par(mfrow=c(1,2))
# 原データと構造モデルによる当てはめ系列のプロット
> plot(cbind(ap, fitted(fit1)), plot.type = "single")
# 原データと固定区間平滑化のプロット
> plot(cbind(ap, tsSmooth(fit1)), plot.type = "single")
```

```
# 企業の利益データ JohnsonJohnson を使用（以下の図を参照）
> JJ <- log10(JohnsonJohnson)
> plot(JJ)                                             # 原データのプロット
> fit2 <- StructTS(JJ, type="BSM")                     # 構造モデルの当てはめ
> sm <- tsSmooth(fit2)                                 # 固定区間平滑化

# 水準，傾き，季節変動のプロット
> plot(cbind(sm[, 1], sm[, 2], sm[, 3]), plot.type = "single",
       col = c("black", "green", "blue"))
> abline(h = -0.5, col = "grey60")
```



固定区間平滑化．航空路線顧客数データ．(左) 原データ，構造モデルと残差．(右) 原データ，固定区間平滑化と残差



固定区間平滑化．企業売上データ．(左) 原データ．(右) 固定区間平滑化による水準，傾向，季節系列

4.11.3　構造モデル用補助関数 KalmanLike()

カルマンフィルタを用いて，(ガウシアン) 対数尤度を求める，また予測や平滑化を行う．

書式：

```
KalmanLike(y, mod, nit = 0)
KalmanRun(y, mod, nit = 0)
KalmanSmooth(y, mod, nit = 0)
KalmanForecast(n.ahead = 10, mod)
makeARIMA(phi, theta, Delta, kappa = 1e6)
```

引数：

y	一変量時系列
mod	状態空間モデルを記述するリスト．以下の説明を参照

`nit` 初期化が計算される時間．`nit=0` は初期化が一時点先の予測様であることを意味し，したがって `Pn` は最初のステップでは計算すべきではないことを意味する
`n.ahead` 予測が必要な将来のステップ数
`phi,theta` 0 以上の長さの数値ベクトルで，AR と MA パラメータを与える
`Delta` 階差係数のベクトルで，ARMA モデルが `y[t]-Delta[1]*y[t-1]-...` に当てはめられる
`kappa` 階差モデル中の過去の観測値に対する事前分散 (イノベーション分散への割合で与える)

これらの関数は，状態ベクトル a，推移 $a \leftarrow Ta + e,\ e \sim \mathrm{N}(0, \kappa Q)$，観測値式 $y = Z'a + R\eta,\ \eta \sim \mathrm{N}(0, \kappa h)$. を持つ一般の一変量状態空間モデルに関連する．尤度は κ を推定した後で得られるプロファイル尤度[*6] である．モデルは少なくとも次の成分を持つリストで指定される：

`T` 推移行列
`Z` 観測値係数
`h` 観測値分散
`V` RQR
`a` 現在の状態推定
`P` 状態の不確定性行列の現在の推定値
`Pn` 状態の不確定性行列の時間 `t-1` の推定値
`KalmanSmooth` クラス `"tsSmooth"` に対する実動関数
`makeARIMA` ARIMA モデルに対する状態空間モデルを構成する

各関数の返り値リストの成分は以下のようになる：

`KalmanLike` 成分 `Lik`(尤度からある定数を引いたもの) と，κ の推定値である `s2`
`KalmanRun` `KalmanLike()` の出力である長さ 2 のベクトル `values`，残差 `resid`，各時点に対して一行を持つ行列である経時的状態推定値である `states`
`KalmanSmooth` 成分 `smooth` は全ての観測値に基づく状態推定の `nxp` 行列で，各時点に対して一つの行を持つ．成分 `var` は分散行列から成る `nxpxp` 配列
`KalmanForecast` 予測値を与える成分 `pred` と (`s2` 倍された) スケール化されていない予測誤差
`makeARIMA` その引き数を成分として持つモデルリスト

注意： これらの関数は，引き渡された引き数の整合性をチェックする他の関数から呼び出されるようにデザインされており，自分自身ではほとんどチェックを行わない．

4.12 その他

4.12.1 時系列を低次元空間に埋め込む `embed()`

時系列 `x` を低次元ユークリッド空間に埋め込む．

[*6] profile likelihood．局外母数に関して最大化した尤度．

書式： embed(x, dimension = 1)

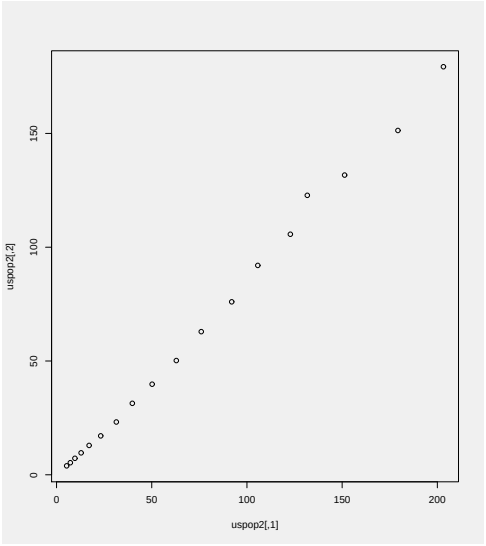
引数：

x 数値ベクトル，行列，または時系列

dimension 埋め込む空間の次元を表すスカラ

結果の行列の各行は系列 x[t],x[t-1],..., x[t-dimension+1] からなる．ここで，t は時系列 x の元の添字である．もし x が行列で，複数の変数を含めば，x[t] は各変数の t 番目の観測値を意味する．返り値は時系列 x を埋め込んだ行列である．

```
# 国勢調査による米国人口 uspop を使用（以下の図を参照）
> uspop2 <- embed(uspop, 2)                    # 2 次元空間に埋め込み
> plot(uspop2)                                 # 散布図
> uspop3 <- embed(uspop, 3)                    # 3 次元空間に埋め込み
> persp(uspop3)                                # 鳥瞰図
```



米国人口データの 2 次元空間への埋め込み (左)，3 次元空間への埋め込み (右)

4.12.2 対称テプリッツ行列 toeplitz()

最初の行を与えて対称テプリッツ (Toeplitz) 行列[*7] を作る．

書式：toeplitz(x)

引数： x テプリッツ行列の最初の行

```
> toeplitz(1:5)                                # 第一行が 1:5 であるテプリッツ行列
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
```

[*7] テプリッツ行列とはその全ての対角 (南東) 方向要素が定数であるような行列であり，それを含む数値演算が計算量的に効率的に実行できることが知られている．時系列解析では自己相関係数，MA モデル等の計算に用いられる．

```
[2,]    2    1    2    3    4
[3,]    3    2    1    2    3
[4,]    4    3    2    1    2
[5,]    5    4    3    2    1
```

第 5 章
多変量解析用関数

R は階層的クラスタリング，主成分分析，因子分析，正準相関，多次元尺度法等の古典的多変量解析手法[*1] 用の関数を持つ．R の推奨パッケージである cluster にはクラスタリング用関数がある．

5.1 クラスタリングと樹状図

5.1.1 階層的クラスタリング hclust()

非類似性のセットに基づく階層的クラスタ解析と，それを解析するメソッド．

書式：

```
hclust(d, method = "complete", members=NULL)
# クラス "hclust" に対するプロットメソッド
plot(x, labels = NULL, hang = 0.1,
     axes = TRUE, frame.plot = FALSE, ann = TRUE,
     main = "Cluster Dendrogram",
     sub = NULL, xlab = NULL, ylab = "Height", ...)
plclust(tree, hang = 0.1, unit = FALSE, level = FALSE, hmin = 0,
        square = TRUE, labels = NULL, plot. = TRUE,
        axes = TRUE, frame.plot = FALSE, ann = TRUE,
        main = "", sub = NULL, xlab = NULL, ylab = "Height")
```

引数：

d　dist() により作られた非類似度

method　用いられる集積法．これは文字列 (曖昧さのない限り省略可能) "ward", "single", "complete", "average", "mcquitty","median" もしくは"centroid" の一つである

members　NULL もしくは d の長さのベクトル．以下の説明を参照

x, tree　hclust() により作られたタイプのオブジェクト

hang　プロットの高さに対する比率で，プロットのこれ以降にラベルがぶら下げられる．負の値ならラベルは 0 からぶら下げられる

[*1] もう一つの代表的手法である判別分析用の関数は，推奨アドオンパッケージバンドルである VR 中に含まれる MASS パッケージに線形判別分析，二次判別分析用の関数がある．VR パッケージは R 本体には含まれないので，独自に入手しインストールする必要がある．クラスタリングと多変量解析関連の貢献パッケージついては CRAN の Task View のテーマ Cluster と Multivariate Analysis を参照せよ．

`labels` 木の葉に対するラベル用の文字列ベクトル．既定では，元のデータの行名ラベル，もしくは行番号が使われる．もし `labels = FALSE` なら，ラベルは付けられない

`axes,frame.plot,ann` `plot.default()` と同様の論理値フラグ

`main,sub,xlab,ylab` `title` に対する文字列． `sub` と `xlab` は `tree$call` が存在すればある既定値を持つ

`...` 追加の作図引き数

`unit` 論理値．もし真なら分離記号は，オブジェクト中の高さではなく，等間隔の高さでプロットされる

`hmin` 数値． `hmin` 未満の高さは `hmin` とみなされる．これは木の底辺部での詳細を抑制するのに使える

`level,square,plot` S-PLUS との互換性のために設けられる予定の `plclust()` の引き数で，未移植

返り値： クラス `"hclust"` のオブジェクトで，クラスタリングの過程でつくり出される木を記述する．オブジェクトはリストで，以下の成分を持つ：

`merge` `(n-1)x1` 行列． `merge` の第 i 行は，クラスタリングの第 `i` ステップのクラスタ結合を記述する．もし行中の第 j 番目の要素が負ならば，`-j` 番目の観測値がこの段階で結合される．もし `j` 番目が正ならば，アルゴリズムの (先の) `j` 段階で形成されたクラスタが結合される．このように， `merge` 中の負の要素は単一観測値の集積を意味し，正の要素は単一観測値でない物の集積を意味する

`height` `n-1` 個の非減少実数の集合．クラスタリングの高さ，つまり特定の集積に対するクラスタリング方法に伴う基準の値

`order` プロットに都合が良い原観測値の置換を与えるベクトル．クラスタリングは，この順序を用いてプロットを行い，行列 `merge` は枝の交差を含まない

`labels` クラスタリングされるオブジェクトそれぞれに対するラベル

`call` 結果を生成した関数呼び出し式

`method` 使用されたクラスタ法

`dist.method` `d` を計算するのに使われた距離 (距離オブジェクトが `"method"` 属性を持つときだけ返される)

この関数は，クラスタを持つ n 個のオブジェクトに対する非類似度のセットを用い，階層的クラスタ解析を行う．最初，各オブジェクトはそれ自身のクラスタに割り当てられる．それから，各段階で二つの最も近いクラスタを結ぶながら，最終的に唯一つのクラスタになるまで，アルゴリズムを繰り返す．各段階でクラスタ間の距離が，使用される個々のクラスタ法に従い，Lance-Williams の非類似度更新公式を用いて再計算される．

幾つかの異なったクラスタ法が用意されている．Ward の最小分散法は，コンパクトで球状のクラスタを見出すことを狙いとする．完全リンク (complete linkage) 法は，類似したクラスタを見付ける．単一リンク (single linkage) 法は，最小スパニング木に密接な関連があり，「友人の友人」戦略を採用する．他の手法は，単一リンク法と完全リンク法のある中間を狙いとしているとみなすことができる．

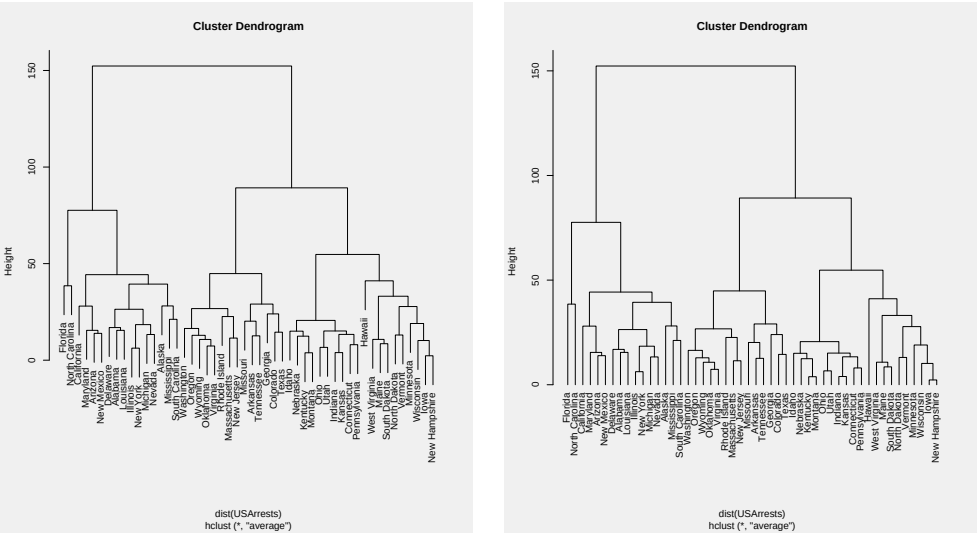
もし `members != NULL` ならば，`d` は個体間の非類似度ではなく，クラスタ間の非類

似度の行列と取ることができ，members はクラスタ毎の観測値の数を与える．この場合階層的クラスタアルゴリズムは，例えば木のあるカットの上部の部分を構成するために，「樹状図の真中から始める」ことができる (例を参照)，リンクの組合せにたいしてだけ効率的に計算できる (つまり，hclust 自体無しで)．最も簡単な場合は 2 乗ユークリッド距離と重心である．この場合，クラスタ間の非類似度は，クラスタ平均間の 2 乗ユークリッド距離になる．

階層的クラスタリングの表示に際し，各結合でどの副木が右に行くか左に行くかを決定する必要がある．n 個の観測値に対しては $n-1$ 回の結合が必要になるから，クラスタ木もしくは樹状図中の葉の可能な順序付けは全部で 2^{n-1} 通りになる．hlust() 中で使われるアルゴリズムは，より詰まったクラスタを左に置く (最後，つまり最も最近，の左副木の結合は，最も最近の右副木の結合よりも下の値に置かれる) ように副木を順序付ける，というものである．単独の観測値は可能な最も詰まったクラスタであり，二つの観測値の結合は，それらを該当観測値番号の順序にしたがって配置する．

hclust オブジェクトには print, plot メソッドがある．主として S-PLUS との下位互換性のため，plclust() 関数は基本的にプロットメソッド plot.hclust と同じである．その余分の引き数は未だ移植されていない．

```
> data(USArrests)                                  # 米国犯罪検挙率データ
> hc <- hclust(dist(USArrests), "ave")             # average 法を用いたクラスタリング
> plot(hc)                                         # 結果のプロット
> plot(hc, hang = -1)                              # ラベルを最下部に付ける
```



米国犯罪検挙率の樹状図．プロットメソッドの hang 引き数の値 0.1 (既定) と-1 によるラベル位置の違いに注意

5.1.2　クラス "hclust" のオブジェクトへの変換 as.hclust()

関数 as.hclust() は他のクラスタリング関数の出力オブジェクトを，クラス "hclust" のオブジェクトに変換する．

書式：as.hclust(x, ...)

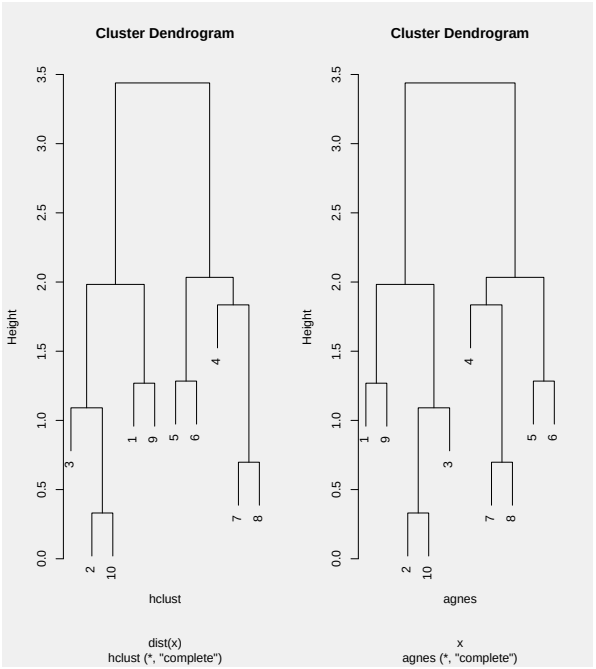
引数：

x	階層的クラスタリングオブジェクト
...	他のメソッドに引き渡される追加引数

返り値： クラス "hclust" のオブジェクト

現在，パッケージ cluster の関数 diana() と agnes() が返すようなクラス "twins" のオブジェクトだけがサポートされている．既定のメソッドはクラス "hclust" オブジェクトとして引き渡されないとエラーになる．

```
# パッケージ cluster 中の関数 agnes() 使用（以下の図を参照）
> x <- matrix(rnorm(30), ncol=3)
> hc <- hclust(dist(x), method="complete")
> require(cluster)                              # パッケージ cluster のロード
> ag <- agnes(x, method="complete")             # 関数 agnes() 使用
> hcag <- as.hclust(ag)                         # "hclust"オブジェクトに変換

# 少し体裁の異なる 2 種類の樹状図
> op <- par(mfrow=c(1,2))                       # 画面を横に 2 分割
> plot(hc) ;  mtext("hclust", side=1)           # hclust() による出力
> plot(hcag); mtext("agnes",  side=1)           # agnes() による出力
```



階層的クラスタリングによる樹状図．左は hclust()，右は cluster パッケージの agnes() を用いた例

5.1.3 階層型クラスタリングのための共表型距離 cophenetic()

関数 cophenetic() は階層型クラスタリングのための共表型[*2] 距離 (cophenetic distance) を計算する関数である．

書式：
```
cophenetic(x) # 既定の S3 メソッド
# クラス dendrogram に対する S3 メソッド
```

[*2] phenetics とは生物学用語で，「表現学，表型分類」と訳される．

`cophenetic(x)`

引数：　x 階層的クラスタリングを表現するオブジェクト．既定メソッドに対してはクラス hclust のオブジェクトか， agnes() のような as.hclust() に対するメソッドを持つオブジェクト

返り値：　クラス dist のオブジェクト

クラスタリングされるべき二つのデータ間の共表型距離は，二つの観測値が最初に一つのクラスタに結合された時のグループ内非類似度として定義される．この距離は多くのタイと制約を持つことを注意しよう．

樹状図 (デンドログラム) は，本来の距離と共表型距離の相関が高い場合にのみデータの適切な要約になっている．さもなければ，それはクラスタリングアルゴリズムの出力の単なる記述にすぎない．

cophenetic() は総称的関数である．階層的クラスタリングを表現するクラスへのサポートは，そうしたクラスに as.hclust() (より一般に何らかの cophenetic()) メソッドを用意することにより得られる．クラス "dendrogram" のオブジェクトへのメソッド関数は，デンドログラムの全ての葉が実際にラベルを持つことを前提にしている．

```
# 米国の犯罪検挙率データ USArrests を使用
> d1 <- dist(USArrests)
> hc <- hclust(d1, "ave")
> d2 <- cophenetic(hc)
> cor(d1, d2)                                          # ユークリッド距離と共表型距離の相関
[1] 0.7658983

> d0 <- c(1,3.8,4.4,5.1,4,4.2,5,2.6,5.3,5.4)       # Sneath & Sokal の例
> attributes(d0) <- list(Size = 5, diag = TRUE)
> class(d0) <- "dist"
> names(d0) <- letters[1:5]
> d0
    a   b   c   d
 b 1.0
 c 3.8 4.0
 d 4.4 4.2 2.6
 e 5.1 5.0 5.3 5.4

> str(upgma <- hclust(d0, method = "average"))
List of 7
 $ merge      : int [1:4, 1:2] -1 -3 1 -5 -2 -4 2 3
 $ height     : num [1:4] 1.0 2.6 4.1 5.2
 $ order      : int [1:5] 5 1 2 3 4
 $ labels     : chr [1:5] "a" "b" "c" "d" ...
 $ method     : chr "average"
  $ call       : language hclust(d = d0, method = "average")
 $ dist.method: NULL
 - attr(*, "class")= chr "hclust"

> plot(upgma, hang = -1)
> (d.coph <- cophenetic(upgma))
    1   2   3   4
2 1.0
3 4.1 4.1
4 4.1 4.1 2.6
5 5.2 5.2 5.2 5.2
> cor(d0, d.coph)
[1] 0.9911351
```

5.1.4 一般的なデンドログラム構造 "dendrogram"

クラス "dendrogram" 用の関数は，木構造を処理する一般的な関数を与える．階層的クラスタリングや分類／回帰木を対象とする同様の関数の代替物として，木のプロットや切断に対する共通のエンジンとなることを目指している．コードは依然テスト段階で，将来変更される可能性がある．

書式：

```
as.dendrogram(object, ...)
as.dendrogram(object, hang = -1, ...) # クラス hclus に対する S3 メソッド
# クラス dendrogram に対する S3 メソッド
plot(x, type = c("rectangle", "triangle"),
       center = FALSE,
       edge.root = is.leaf(x) || !is.null(attr(x,"edgetext")),
       nodePar = NULL, edgePar = list(),
       leaflab = c("perpendicular", "textlike", "none"),
       dLeaf = NULL, xlab = "", ylab = "", xaxt = "n", yaxt = "s",
       horiz = FALSE, frame.plot = FALSE, ...)
cut(x, h, ...)         # クラス dendrogram に対する S3 メソッド
print(x, digits, ...) # クラス dendrogram に対する S3 メソッド
rev(x)                 # クラス dendrogram に対する S3 メソッド
# クラス dendrogram に対する S3 メソッド
str(object, max.level = NA, digits.d = 3,
    give.attr = FALSE, wid = getOption("width"),
    nest.lev = 0, indent.str = "", stem = "--", ...)
is.leaf(object)
```

引数：

object　クラス dendrogram のいずれかに変換できる任意の R オブジェクト

x　クラス dendrogram のオブジェクト

hang　数値スカラ関数で，親の高さから枝の高さを計算する． plot.hclust() を見よ

type　プロットタイプ

center　論理値． TRUE ならノードは枝中の葉に関して中央位置にプロットされる．さもなければ (既定値) 全ての直接の子ノードの中間にプロットする

edge.root　論理値．もし TRUE なら，ルートノードに向かって辺を引く

nodePar　ノードに対して使われるプロットパラメータのリスト (points を見よ)．既定値の NULL はノード位置にシンボルを描かない．リストは名前付きの成分 pch, cex, col そして (もしくは) bg を持つ．各々は内部ノードと葉に対する二つの別個の属性を指示するために長さが 2 であっても良い

edgePar　エッジの線分と (もし edgetext があれば) そのラベルに対して使われるプロット用パラメータのリスト．リストは名前付きの成分 col,(線分に対する) lty と lwd, p.col, (テキスト周りの多角形用の) p.lwd と p.lty そしてテキス

トの色用の t.col を含んで良い． nodePar に対するのと同様に，各々は葉と内部のノードを区別するために長さが 2 であって良い

leaflab 葉のラベルを指定する文字列．既定の "perpendicular" はテキストを垂直に書く． "textlike" はテキストを矩形内で水平に書く．そして，"none" は葉のラベルを書かない

dLeaf 葉の頂点とそのラベル間のユーザ座標での距離を指定する数．もし既定の NULL ならば文字幅もしくは高さの 3/4 が使われる

horiz 論理値で，樹状図を水平に描くかどうかを指示する

frame.plot 論理値で，プロットの周囲に枠を描くかどうかを指示する．plot.default を見よ

h 木がカットされる高さ位置

...,xlab,ylab,xaxt,yaxt 作画パラメータ，もしくは他のメソッドへの引き数

digits 出力時の精度を指定する整数． print.default を見よ

max.level,digits.d,give.attr,wid,nest.lev,indent.str str() への引数．str.default() を見よ． give.attr=FALSE としても，各ノードに対する height と members 属性は依然として示される

stem 各デンドログラムの枝に対して使う幹を指定する， str() に対して使われる文字列

樹状図は直接に入れ子リストとして表現されており，各成分は木の枝に相当する．したがって，木 z の最初の枝は z[[1]] であり，対応する副木の二番目の枝は z[[1]][[2]] である，等々．各ノードは，プロットと，性質による分割を効率的に行うため，幾つかの情報を所持しているが，members, height そして葉に対する leaf だけが必須である．

members 枝中の葉の総数

height ノードがプロットされる高さである非負の数値

midpoint 枝の左端しからのノードの数値距離．これは少なくとも plot(*,center=FALSE) に対して必要である

label 文字列．ノードのラベル

x.member cut()$upper に対して，「前の」メンバーの数．より一般的には，水平 (horiz=FALSE の時，さもなければ垂直) 方向の整列に対して使われる member 成分に対する代入物

edgetext 文字列．ノードへ向かう辺に対するラベル

nodePar 点のプロットに対するノード固有の特性を指定する長さ 1 のベクトルの名前付きリスト．上の nodePar 引き数を見よ

edgePar 長さ 1 の成分からなる名前付きリスト．ノードへ向かう辺のプロットに対する属性を指定し，もしあれば edgetext を書く．上の edgePar 引き数を見よ

leaf 論理値．もし TRUE ならノードは木の葉になる

cut.dendrogram() 成分 upper と lower を持つリストを返す．前者は元の木の簡略版でやはりクラス dendrogram ．後者は木を分割して得られた枝を持つリストで，各枝はクラス dendrogram

cut.dendrogram() は成分 upper と lower を持つリストを返す．最初の成分は元の木の簡略版であり，やはりクラス "dendrogram" である．後者は木を切断して得られた枝からなるリストで，それぞれデンドログラムである．

"dendrogram" オブジェクトには [[, print そして str メソッドがある．最初のもの (抜きだし) は副枝の選択によりクラスが保存されることを保証する．

クラス "hclust" のオブジェクトはメソッド関数 as.dendrogram() を用いてクラス "dendrogram" に変換できる．

rev.dendrogram() は単にデンドログラム x のノードを逆転したものを返す (reorder.dendrogram() を参照せよ)．

is.leaf(object) は object が葉 (もっとも簡単なデンドログラム) かどうかをテストする．plotNode() と plotNodeLimit() は補助関数である．

警告：プロットを含むデンドログラムの操作は再帰的手順を使うため，特に深い木構造に対しては options("expressions") を増やす必要があるかも知れない．このためには，C のスタックサイズを OS の既定値よりも大きめに設定することが必要とされる可能性が高い (これができるかどうかは OS 依存で，Windows では不可)．
注意：type="triangle" を使う際は center=TRUE を使う方が普通見栄えが良くなる．
関連：デンドログラムの labels メソッドに関しては order.dendrogram()．

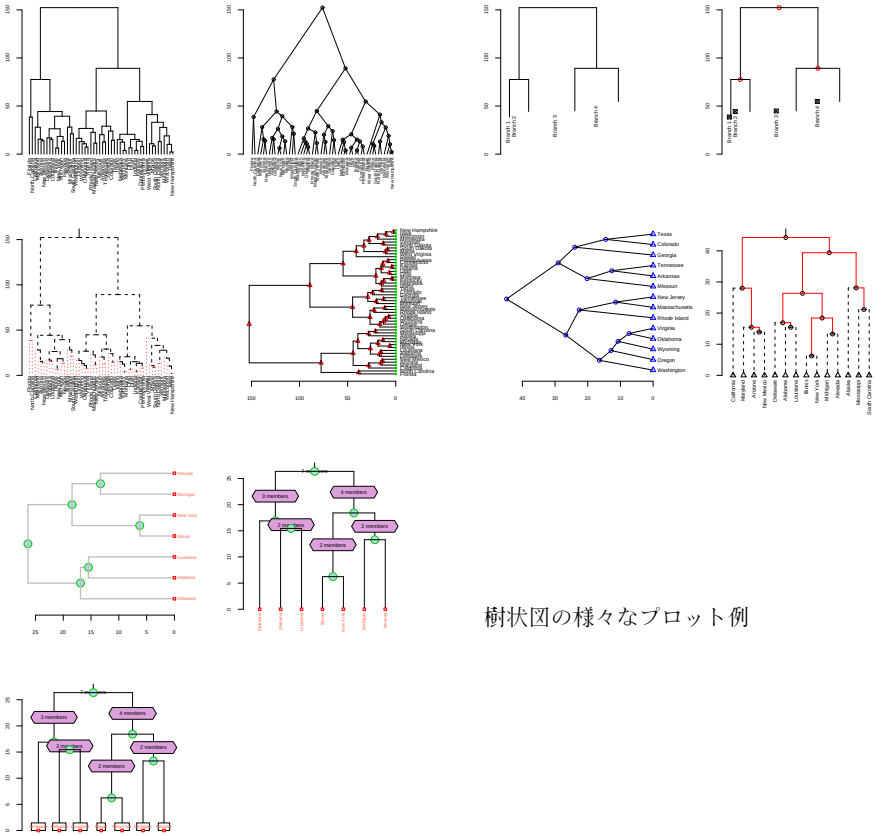
```
# 樹状図に対するメソッド．米国の犯罪検挙率データ USArrests 使用
> hc <- hclust(dist(USArrests), "ave")            # 階層的クラスタリング
> (dend1 <- as.dendrogram(hc))                    # 樹状図に変換しプロット
'dendrogram' with 2 branches and 50 members total, at height 152.314

>      str(dend1)                                 # str() メソッド．dend1 の全構造
--[dendrogram w/ 2 branches and 50 members at h = 152]
  |--[dendrogram w/ 2 branches and 16 members at h = 77.6]
  |  |--[dendrogram w/ 2 branches and 2 members at h = 38.5]
  |  |  |--leaf "Florida"
  |  |  `--leaf "North Carolina"
(以下省略)

> str(dend1, max = 2)                             # 最初の 2 つの副レベルだけ表示
--[dendrogram w/ 2 branches and 50 members at h = 152]
  |--[dendrogram w/ 2 branches and 16 members at h = 77.6]
  |  |--[dendrogram w/ 2 branches and 2 members at h = 38.5] ..
  |  `--[dendrogram w/ 2 branches and 14 members at h = 44.3] ..
  `--[dendrogram w/ 2 branches and 34 members at h = 89.2]
     |--[dendrogram w/ 2 branches and 14 members at h = 44.8] ..
     `--[dendrogram w/ 2 branches and 20 members at h = 54.7] ..
```

```
# 見掛けを様々に変える (次の図を参照)
> plot(dend1)

# "triangle"タイプで，内部ノードも表示
> plot(dend1, nodePar=list(pch = c(1,NA), cex=0.8, lab.cex = 0.8),
       type = "t", center=TRUE)
> plot(dend1, edgePar=list(col = 1:2, lty = 2:3),
       dLeaf=1, edge.root = TRUE)
> plot(dend1, nodePar=list(pch = 2:1,cex=.4*2:1, col = 2:3),
       horiz=TRUE)
```

```
> dend2 <- cut(dend1, h=70)                       # 高さ 70 でカットした樹状図 (次の図を参照)
> plot(dend2$upper)
> plot(dend2$upper, nodePar=list(pch = c(1,7), col = 2:1)) # 水平方向の葉が変

#  dend2$lower はデンドログラムでなく，リスト
> plot(dend2$lower[[3]], nodePar=list(col=4), horiz = TRUE, type = "tr")
```

```
# "inner","leaf"エッジを異なったタイプと色で
> plot(dend2$lower[[2]], nodePar=list(col=1),# non empty list
       edgePar = list(lty=1:2, col=2:1), edge.root=TRUE)
```

```
# 次の図を参照
> nP <- list(col=3:2, cex=c(2.0, 0.75), pch= 21:22,
             bg= c("light blue", "pink"),
             lab.cex = 0.75, lab.col = "tomato")
> par(mfrow= c(2,2), mar = c(5,2,1,4))
> plot(d3, nodePar= nP, edgePar = list(col="gray", lwd=2), horiz = TRUE)
> addE <- function(n) {                              # 補助関数定義
       if(!is.leaf(n)) {            # leaf でなければ属性"edgePar","edgetext"を追加
         attr(n, "edgePar") <- list(p.col="plum")
         attr(n, "edgetext") <- paste(attr(n,"members"),"members")
       }
       n
  }
> d3e <- dendrapply(d3, addE)                        # d3 のノードに再帰的に関数 addE() を適用
> plot(d3e, nodePar= nP)
> plot(d3e, nodePar= nP, leaflab = "textlike")
```



樹状図の様々なプロット例

5.1.5　樹状図のクラスタの周りに枠を描く rect.hclust()

関数 rect.hclust() は，樹状図のクラスタの周りに枠を描く．

書式：

```
rect.hclust(tree, k = NULL, which = NULL, x = NULL, h = NULL,
            border = 2, cluster = NULL)
```

引数：

`tree` `hclust()` が生成するようなオブジェクト

`k, h` スカラ．樹状図を，ちょうど `k` 個のクラスタが作られるように分割するか，高さ `h` で分割する

`which, x` 周りに矩形を描くクラスタを選ぶためのベクトル． `which` はクラスタを番号 (木の左から右へ) で選び， `x` は対応する水平座標値を含むクラスタを選ぶ．既定値は `which=1:k`

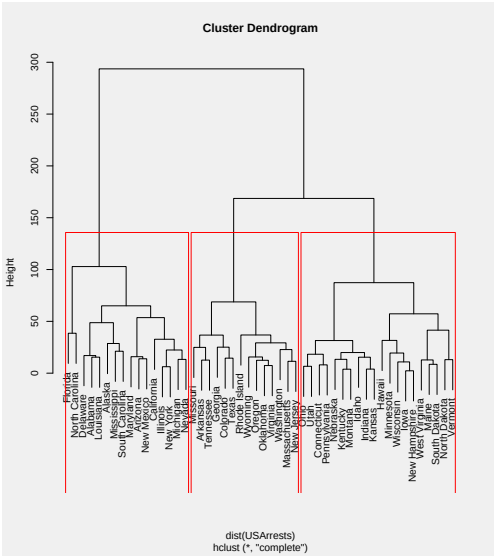
`border` 矩形の境界色のためのベクトル

`cluster` オプションとして， `cutree(hclust.obj,k=k)` が返すようなクラスタ所属情報を持つベクトルが既に計算済みなら，効率性のために指定できる

返り値： (コンソールに表示されない) リストで，各成分は対応するクラスタ中のデータ点のベクトルを含む

樹状図の枝の周りに矩形の枠を描き，クラスタを強調する．最初に樹状図があるレベルで分割され，それから矩形枠が選ばれた枝の周りに描かれる．

```
# 米国の犯罪検挙率データ USArrests を使用（次の図を参照）
> hca <- hclust(dist(USArrests))
> plot(hca)
> rect.hclust(hca, k=3, border="red")              # レベル 3 の枝をハイライト
```



樹状図の枝を矩形で囲む

5.1.6 樹状図を並べ換える `reorder()`

関数 `reorder()` は樹状図を並べ換える．

書式：

```
reorder(x, ...)
# クラス factor に対する S3 メソッド S3
reorder(x, X, FUN = mean, ...,  order = is.ordered(x))
```

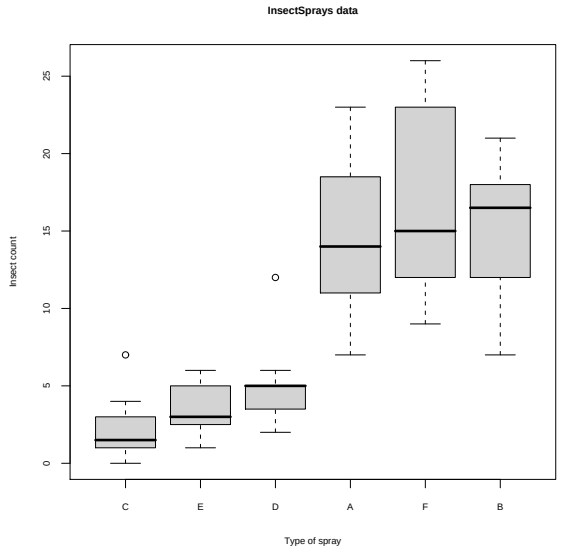
引数：

x	その水準の順序を変更する因子 (順序付きで良い)
X	x と同じ長さのベクトルで， x のユニークな水準のそれぞれに対する値の部分集合が最終的なその水準の順序を決定する
FUN	x の水準で決まる X の各部分集合に適用される関数で，最初の引数はベクトルで，スカラを返す
...	オプションの FUN の追加引き数
order	論理値．返り値は単なる因子でなく順序付き因子にするか?

返り値： 因子か，(order の値に応じて) x によってグルーピングされる X に適用された FUN 値によって決まる水準の順序を持つ因子である．水準は FUN の値が増加するように順序付けられる．更に，X の部分集合に FUN を適用して得られる値が "scores" 属性として与えられる

関連：reorder.dendrogram(), levels(), relevel().

```
# 殺虫剤の効き目データ InsectSprays 使用.
# count 変数の中央値の昇順に spray 因子を並べ変え (以下の図を参照)
> bymedian <- with(InsectSprays, reorder(spray, count, median))
> boxplot(count ~ bymedian, data = InsectSprays,
          xlab = "Type of spray", ylab = "Insect count",
          main = "InsectSprays data", varwidth = TRUE,
          col = "lightgray")
```



箱型図を中央値が増加するように並べる

5.1.7 クラスタリング木の分割 cutree()

cutree() は，hclust() の出力であるような木を，必要なグループ数，もしくは分割の高さを与えて，幾つかのグループに分割する．

書式： cutree(tree, k = NULL, h = NULL)

引数：

tree hclust() 等により作られた木．cutree() は，それぞれ適切な内容を含む成分 merge, height そして labels を持つリストだけを必要とする

k 要求されるグループ数を表す整数スカラー，もしくはベクトル

h 木がカットされる高さを表す数値スカラー，もしくはベクトル

返り値： k または h がスカラーならグループ情報を持つベクトルを返す．さもなければ，グループ情報を持つ行列を返す．この行列の各列は k もしくは h の要素にそれぞれ対応する (列名にも使われる)

注意：少なくとも k, h の何れかを指定しなければならない．双方が与えられると k が優先使用される

```
> data(USArrests)                                  # 米国の犯罪検挙率データ
> hc <- hclust(dist(USArrests))
> cutree(hc, k=1:5)                                # k=1 は自明
               1 2 3 4 5
Alabama        1 1 1 1 1
Alaska         1 1 1 1 1
(途中省略)
Wyoming        1 2 2 2 2

> cutree(hc, h=250)
     Alabama          Alaska         Arizona        Arkansas      California
           1               1               1               2               1
    Colorado     Connecticut        Delaware         Florida         Georgia
           2               2               1               1               2
(途中省略)
    Virginia      Washington   West Virginia       Wisconsin         Wyoming
           2               2               2               2               2

> g24 <- cutree(hc, k = c(2,4))                    # 第 2,3 グループを比較
> table(g24[,"2"], g24[,"4"])
    1  2  3  4
 1 14  0  0  2
 2  0 14 20  0
```

5.1.8 ヒートマップ heatmap()

ヒートマップは疑似的なカラーイメージ (基本的に image(t(x))) で，樹状図が左と上の端に描かれる．典型的には，樹状図が作られた際の制約下で，行と列をある値のセット (行，または列平均) に従い，並べ換える．

書式：

```
heatmap(x, Rowv=NULL, Colv=if(symm)"Rowv" else NULL, distfun = dist,
        hclustfun = hclust, add.expr, symm = FALSE,
        revC = identical(Colv, "Rowv"),
        scale=c("row", "column", "none"), na.rm = TRUE,
        margins = c(5, 5), ColSideColors, RowSideColors,
        cexRow = 0.2 + 1/log10(nr), cexCol = 0.2 + 1/log10(nc),
        labRow = NULL, labCol = NULL, main = NULL, xlab = NULL,
        ylab = NULL, verbose = getOption("verbose"), ...)
```

引数：

x　プロットされる値の数値行列

Rowv　行樹状図を描くかどうか，どのように計算し並べ換えるかを決める．dendrogram か，行樹状図を並べ変えるのに使われる値のベクトル，もしくは NA なら行樹状図のプロット (と並べ換え) をしない．既定では NULL．以下の説明を参照せよ

Colv　列樹状図をどのように並べ換えるかを決める．上の Rowv 引き数と同じ意味を持つ．加えて x が正方行列なら， Colv="Rowv" は列を行と同様に扱うことを意味する

distfun　列と行の双方間の距離を計算する関数．既定は dist()

hclustfun　Rowv と Colv が樹状図でないときに，階層的クラスタリングを計算する関数．既定は hclust()

add.expr　image() の呼び出し後に評価される表現式．プロットに要素を加えるのに使うことができる

symm　論理値で， x が正方行列の時，対称に扱うかどうかを指示する

revC　論理値で，例えば対称ケースで通例のように対称な軸を使うように，列順序をプロット時に逆転するかどうかを指示する

scale　文字列で，値を行・列方向のどちらに中心化しスケール化するか，もしくはなにもしないか，を指示する．既定値は，もし symm=FALSE なら "row"，さもなければ "none"

na.rm　論理値で， NA 値を取り除くかどうかを指示する

margins　長さ 2 の数値ベクトルで，それぞれ行と列名を含む余白を含む (par(mar=*) を見よ)

ColSideColors　オプション．長さ ncol(x) の文字ベクトルで， x の列を注釈するのに使われる水平な棒に対する色名を含む

RowSideColors　オプション．長さ nrow(x) の文字ベクトルで， x の行を注釈するのに使われる垂直な棒に対する色名を含む

cexRow,cexCol　正数で，行もしくは列軸のラベリングのために作図パラメータ cex.axis として使われる．既定値は，それぞれ行・列数だけを使う

labRow,labCol　行と列ラベルを含む文字ベクトル．既定値はそれぞれ rownames(x) または colnames(x)

main,xlab,ylab　主， x 軸そして y 軸タイトル．既定では無し

verbose　論理値で，情報を出力するかどうかを指示する

...　image() 関数用追加引き数．例えば色を指示する col

返り値：　コンソールに表示されないリストで，以下の成分を持つ：

rowInd　order.dendrogram() が返すような，行添字置換ベクトル

colInd　列添字置換ベクトル

Rowv　行デンドログラム (Rowv が NA でなく， keep.dendro=TRUE の時だけ)

Colv　列デンドログラム (Colv が NA でなく， keep.dendro=TRUE の時だけ)

もし Rowv と Colv のどちらかが樹状図なら，それらは尊重される (並べ換

えられない). さもなければ樹状図が, X を x または t(x) の一方とした dd <- as.dendrogram(hclustfun(distfun(X))) のように計算される.

もしどちらかが (weights の) ベクトルなら, 適当な樹状図が, 樹状図による制約に従う与えられた値に応じて並べ換えられる (行ケースでは reorder(dd, Rowv) が使われる). 既定値のように, もしどちらも欠けていれば, 対応する樹状図の並べ換えは行・列平均に基づく. つまり, 行ケースでは Rowv <- rowMeans(x, na.rm=na.rm). もしどちらかが NULL ならば, 対応する側に対しては何も行われない.

既定 (scale = "row") では, 行は平均 0 で標準偏差 1 になるように, スケール化される. これが有用であるというゲノム解析への応用プロットからのある経験的証拠がある.

既定の色は見栄えが良くない. CRAN 登録のカラーパレット RColorBrewer パッケージのような拡張を検討しよう.

注意: Rowv = NA (もしくは Colv = NA) でない限り, 元の行と列は樹状図にマッチするように並べ換えられる. 例えば, Rowv を (もしかすると reorder() 関数で並べ換えた) 行樹状図としたときの, order.dendrogram(Rowv). heatmap() は layout() 関数を使い, 2×2 分割されたレイアウトの右下隅にイメージを描く. したがって, par(mfrow= *) や par(mfcol= *) を使った多重行・列レイアウトでは使用できない.

```
# 自動車性能データ mtcars を使用 (以下の図を参照)
> x  <- as.matrix(mtcars)
> rc <- rainbow(nrow(x), start=0, end=.3)         # 虹色色調利用
> cc <- rainbow(ncol(x), start=0, end=.3)
> hv <- heatmap(x, col = cm.colors(256), scale="column",
                RowSideColors = rc, ColSideColors = cc, margins=c(5,10),
                xlab = "specification variables", ylab= "Car Models",
                main = "heatmap(<Mtcars data>, ..., scale = \"column\")")

> str(hv)                                         # 添字ベクトルの二種類の並べ換え
List of 4
 $ rowInd: int [1:32] 31 17 16 15 5 25 29 24 7 6 ...
 $ colInd: int [1:11] 2 9 8 11 6 5 10 7 1 4 ...
 $ Rowv  : NULL
 $ Colv  : NULL

# 列樹状図を描かない (並べ換えも無し)(以下の図を参照)
> heatmap(x, Colv = NA, col = cm.colors(256), scale="column",
          RowSideColors = rc, margins=c(5,10),
          xlab = "specification variables", ylab= "Car Models",
          main = "heatmap(<Mtcars data>, ..., scale = \"column\")")

# 樹状図無し (以下の図を参照)
> heatmap(x, Rowv = NA, Colv = NA, scale="column",
          main = "heatmap(*, NA, NA) ~= image(t(x))")
```

```
# 従業員勤務態度データ attitude 使用 (以下の図を参照)
> round(Ca <- cor(attitude), 2)                   # 丸め処理
           rating complaints privileges learning raises critical advance
rating       1.00       0.83       0.43     0.62   0.59     0.16    0.16
complaints   0.83       1.00       0.56     0.60   0.67     0.19    0.22
privileges   0.43       0.56       1.00     0.49   0.45     0.15    0.34
learning     0.62       0.60       0.49     1.00   0.64     0.12    0.53
raises       0.59       0.67       0.45     0.64   1.00     0.38    0.57
critical     0.16       0.19       0.15     0.12   0.38     1.00    0.28
advance      0.16       0.22       0.34     0.53   0.57     0.28    1.00

> symnum(Ca)                                      # 単純な文字グラフィックス
           rt cm p l rs cr a
rating     1
complaints +  1
privileges .  .  1
```

```
learning   ,  .  . 1
raises     .  ,  . , 1
critical             .  1
advance          . . .     1
attr(,"legend")
[1] 0  ' '  0.3  '.'  0.6  ','  0.8  '+'  0.9  '*'  0.95  'B'  1

> heatmap(Ca, symm = TRUE, margins=c(6,6)) # reorder() 使用
> heatmap(Ca, Rowv=FALSE, symm = TRUE, margins=c(6,6)) # reorder() 不使用
```

```
# 距離を定義する関数を自前で与える例 (以下の図を参照)
# 米国最高裁判事の法律家による評価データ USJudgeRatings 使用
> symnum( cU <- cor(USJudgeRatings) )
     CO I DM DI CF DE PR F O W PH R
CONT 1
INTG    1
DMNR    B 1
DILG    + +  1
CFMG    + +  B  1
DECI    + +  B  B  1
PREP    + +  B  B  B  1
FAMI    + +  B  *  *  B  1
ORAL    * *  B  B  *  B  B 1
WRIT    * +  B  *  *  B  B B 1
PHYS    , ,  +  +  +  +  + + + 1
RTEN    * *  *  *  *  B  * B B *  1
attr(,"legend")
[1] 0  ' '  0.3  '.'  0.6  ','  0.8  '+'  0.9  '*'  0.95  'B'  1

> hU <- heatmap(cU, Rowv = FALSE, symm = TRUE, col = topo.colors(16),
               distfun = function(c) as.dist(1 - c), keep.dendro = TRUE)

# 同じ再並べ替えを持つ相関行列
> round(100 * cU[hU[[1]], hU[[2]]])
     CONT INTG DMNR PHYS DILG CFMG DECI RTEN ORAL WRIT PREP FAMI
CONT  100  -13  -15    5    1   14    9   -3   -1   -4    1   -3
INTG  -13  100   96   74   87   81   80   94   91   91   88   87
DMNR  -15   96  100   79   84   81   80   94   91   89   86   84
PHYS    5   74   79  100   81   88   87   91   89   86   85   84
(以下省略)

> str(hU$Colv)                                        # 列デンドログラム
--[dendrogram w/ 2 branches and 12 members at h = 1.15]
  |--leaf "CONT"
  '--[dendrogram w/ 2 branches and 11 members at h = 0.258]
     |--[dendrogram w/ 2 branches and 2 members at h = 0.0354]
     |  |--leaf "INTG"
     |  '--leaf "DMNR"
     '--[dendrogram w/ 2 branches and 9 members at h = 0.187]
(以下省略)
```

5.1.9　樹状図のインタラクティブな操作 identify.hclust()

identify.hclust() は，第一マウスボタンが押された時の，グラフィックスポインターの位置を読み取る．それから，木をポインターの垂直位置で切断し，ポインターの水平位置を含むクラスタをハイライトする．オプションとして，この関数はクラスタに含まれるデータ点の添字に適用できる．(identify.hclust() は identify() 関数のクラス "hclust" に対するメソッドである．)

書式： identify.hclust(x, FUN = NULL, N = 20, MAXCLUSTER = 20,
DEV.FUN = NULL, ...)

引数：

x　　hclust() がつくり出すタイプのオブジェクト

FUN　クラスタ中のデータ点の添字数に適用される (オプションの) 関数．下の詳細を見よ

N　照合されるべきクラスタの最大数

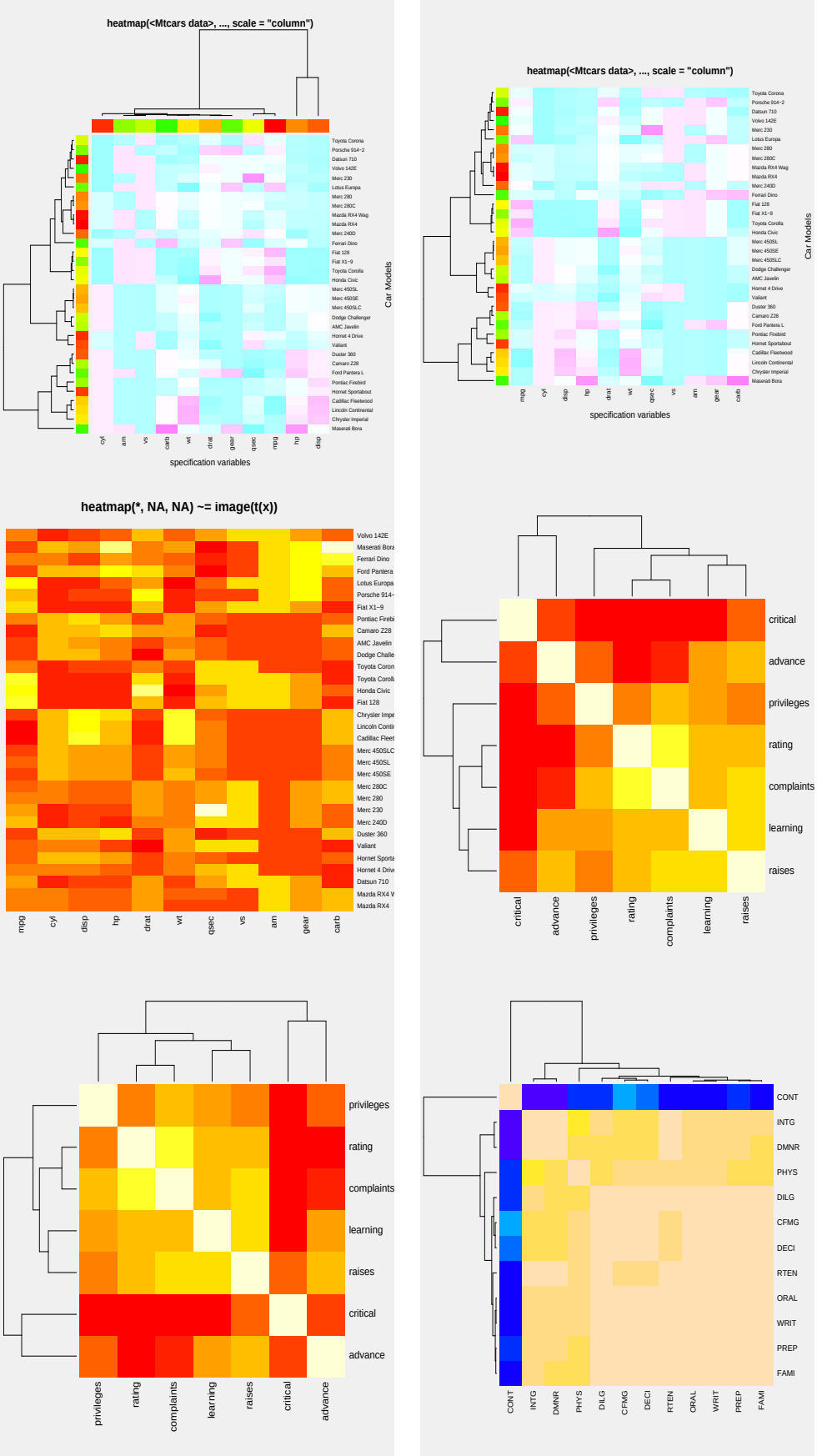
MAXCLUSTER　切断により作り出すことのできるクラスタの最大数 (ポインタの有効な垂直範囲を制限する

DEV.FUN　オプションの整数値．もし指定されると，対応するグラフィックスデバイスが FUN の適用前にアクティブになる

...　FUN への追加引き数

返り値：　データ点の添字のベクトルのリストか，関数 FUN の返り値のリスト

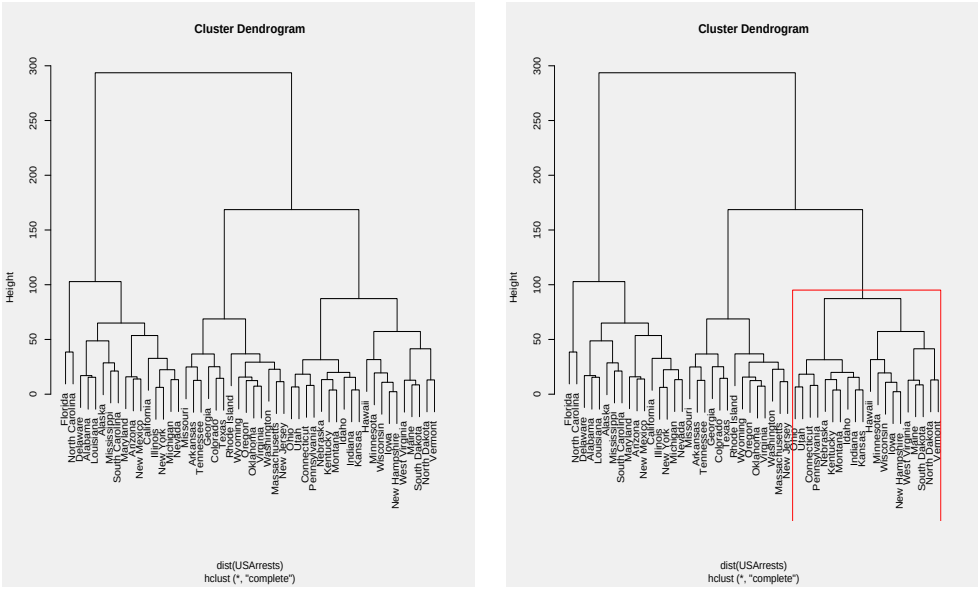
既定では，クラスタはマウス用いて特定され，該当データ点の「コンソールには表示されない (invisible)」添字のリストが返される．もし，FUN が NULL で無ければ，データ点の添字ベクトルは，この関数の第一引き数として引き渡される．以下の例を参照せよ．FUN に対するアクティブなグラフィックスデバイスは DEV.FUN を用いて指定できる．特定の過程はマウスの第一ボタン以外を押すと停止する．identify() 関数を参照せよ．



樹状図に対するヒートマップ

```
# 米国犯罪検挙率データ USArrests 使用（次の図を参照）
> hca <- hclust(dist(USArrests))                    # 階層的クラスタリング
> plot(hca)                                         # そのプロット

# マウス第一ボタンでで樹状図の特定クラスタ選択すると該当副木が枠で囲まれ，
# その部分の情報が返される．同時に複数の選択ができる
> (x <- identify(hca))                              # 選択中止はマウスの第 2 ボタンを押す
[[1]]
     Arkansas      Colorado       Georgia Massachusetts      Missouri
            4             6            10            21            25
   New Jersey      Oklahoma        Oregon  Rhode Island     Tennessee
           30            36            37            39            42
        Texas      Virginia    Washington       Wyoming
           43            46            47            50
```

```
# アヤメデータ iris を使用
> hci <- hclust(dist(iris[,1:4]))                   # 階層的クラスタリング
# 新しいデバイスを開く（樹状図と棒グラフ用に二つ）
> get(getOption("device"))()                        # 元のデバイスの脇に
> nD <- dev.cur()                                   # 棒グラフ用
> dev.set(dev.prev())                               # 樹状図用の元のデバイス
> plot(hci)                                         # 樹状図を描く

# マウスでクラスタを選択すると，クラスタ中のアヤメの種類の棒グラフが描かれる
> identify(hci, function(k) barplot(table(iris[k,5]),col=2:4), DEV.FUN=nD)
```



樹状図中の副木のマウスによる操作 (米国犯罪検挙率データ)．元の樹状図 (左)．クラスタを選択した様子 (右)

5.1.10 樹状図中の葉の順序を得る order.dendrogram()

関数 order.dendrogram() は樹状図の葉の順序 (添字) を返す．これは，任意の追加データの適当な成分を得ることに使える．葉の添字の順序が左から右へ与えられる．左から右への順で，葉のインデックスまたはラベルが取り出される．

書式：

```
order.dendrogram(x)
labels(object, ...)  # クラス dendrogram に対する S3 メソッド
```

引数：

x, object　デンドログラム (as.dendrogram() を参照)

...　追加引数

返り値：　長さがデンドログラム中の葉の数に等しいベクトル．樹状図の葉の数の長さに等しいベクトルが返される．各要素は (それから樹状図が計算された) 原データに於ける添字である．r <- order.dendrogram() とすると，各要素はデンドログラムがそれから計算された元のデータへのインデックスである

```
> set.seed(123)                                    # 乱数種 123 を設定
> x <- rnorm(10)                                   # 10 個の正規乱数
> hc <- hclust(dist(x))                            # 階層的クラスタリング
> hc$order                                         # 要素の順序
 [1]  3  6  7  2  4  5  8  9  1 10

> dd <- as.dendrogram(hc)                          # hc を樹状図に変換
> order.dendrogram(dd)                             # 樹状図から葉の順序を得る
 [1]  3  6  7  2  4  5  8  9  1 10

> set.seed(123)                                    # 乱数種 123 を設定
> x <- rnorm(10)                                   # 10 個の正規乱数
> hc <- hclust(dist(x))                            # 階層的クラスタリング
> hc$order
 [1]  3  6  7  2  4  5  8  9  1 10
> dd <- as.dendrogram(hc)                          # デンドログラムに変換
> order.dendrogram(dd)                             # hc$order と同じ
 [1]  3  6  7  2  4  5  8  9  1 10

> d2 <- as.dendrogram(hclust(dist(USArrests)))
> labels(d2)
 [1] "Florida"        "North Carolina" "Delaware"       "Alabama"
 [5] "Louisiana"      "Alaska"         "Mississippi"    "South Carolina"
 [9] "Maryland"       "Arizona"        "New Mexico"     "California"
(途中省略)
[49] "North Dakota"   "Vermont"
```

5.1.11 k-means 法によるクラスタリング kmeans()

関数 kmeans() はデータ行列に対し，k-means 法によるクラスタリングを実行する．

書式：

```
kmeans(x, centers, iter.max = 10, nstart = 1,
       algorithm = c("Hartigan-Wong","Lloyd","Forgy","MacQueen"))
```

引数：

x	数値データ行列，またはそうした行列に変換できるオブジェクト (例えば，数値ベクトルや，列が全て数値のデータフレーム)
centers	クラスタの数か，クラスタ中心の初期値のセット．前者なら x 中の列が初期中心としてランダムに選ばれる
iter.max	許される最大繰り返し回数
nstart	もし centers が数値ならば，選ばれるランダム集合の数
algorithm	文字．省略可能

返り値： 次の成分を持つクラス "kmeans" のリスト：

cluster	各点が所属するクラスタを指示する整数のベクトル
centers	クラスタ中心の行列
withinss	各クラスタに対するクラスタ内 2 乗和
size	各クラスタ内の点の数

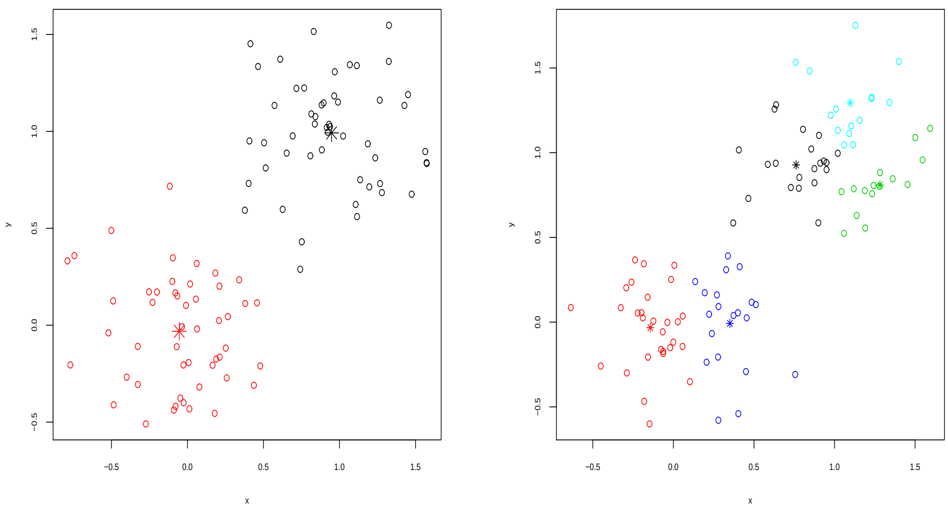
x で与えられたデータが k-means 法アルゴリズムでクラスタ化される．k 個のクラスタ中心を仮に与え，次に点群 x をグループ毎に各点からクラスタ (グループ) 中心までの距離の 2 乗和が最小になるように k 個のグループに分割する．更に距離 2 乗の総和が最小となるクラスタ中心を定める．これが終了したとき，全てのクラスタ中心はそれらのボロノイ細胞 (そのクラスタ中心に最も近いデータ点の集まり) の中心にある．

既定では Hartigan & Wong のアルゴリズムが使われる．k-means 法という言葉は，一般的手法ではなく特定のアルゴリズムを指すことがある．最も広く使われるのが MacQueen のそれで，Lloyd や Forgy のアルゴリズムも使われる．Hartigan-Wong アルゴリズムは一般にこれらよりも良い結果を与えるが，ランダムな初期値を複数与えることが勧められる．

Lloyd, Forgy 法を除き，数を指定すると k 個のクラスターが必ず返される．もし中心位置の初期行列が与えられると，一つもしくは複数の中心に最も近い点が無いこともあり得，これは Hartigan-Wong 法では現在エラーとされる．

```
# 人工的なデータを用いた例 (以下の図を参照)
> x <- rbind(matrix(rnorm(100,sd=0.3), ncol=2),
           matrix(rnorm(100,mean=1,sd=0.3), ncol=2))
> colnames(x) <- c("x", "y")                    # 列名ラベル"x","y"を付ける
> (cl <- kmeans(x, 2))                          # クラスタ数 2 で実行
> plot(x, col = cl$cluster)         # 決定クラスタ毎に色を変えてデータ散布図を描く
> points(cl$centers, col = 1:2, pch = 8, cex=2)   # クラスタ中心点を上描き

# 2 次元の人工的な例 (以下の図を参照)
> x <- rbind(matrix(rnorm(100,sd=0.3), ncol=2),
```

```
            matrix(rnorm(100,mean=1,sd=0.3), ncol=2))
> colnames(x) <- c("x", "y")
> (cl <- kmeans(x, 2))                            # クラスタ数 2 で実行
K-means clustering with 2 clusters of sizes 51, 49
Cluster means:                                    # 最終的に選ばれた 2 つのクラスタ中心座標
            x           y
1  0.94776039  0.99215383
2 -0.05411242 -0.03127728
Clustering vector: # 各点が 2 つのクラスタのどちらに属するかを示すインデックス
 [1]  2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 (行末まで省略)
[38]  2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 (行末まで省略)
[75]  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Within cluster sum of squares by cluster:
[1] 9.382877 8.520439                         # 各クラスタ内の点とその中心までの距離の 2 乗和
Available components:
[1] "cluster"  "centers"  "withinss" "size"       # 結果リストの成分名

> plot(x, col = cl$cluster)                       # データ点をクラスタ毎に色分けして表示
> points(cl$centers, col = 1:2, pch = 8, cex=2)   # クラスタ中心をアステリスクで表示

# クラスタ数が多いときは，複数初期値をランダムに与えると良い
# クラスタ数 5 を指定．初期クラスタ中心をランダムに 25 種類試す
> (cl <- kmeans(x, 5, nstart = 25))
K-means clustering with 5 clusters of sizes 13, 20, 21, 29, 17
Cluster means:
            x           y
1 -0.46445153  0.03291262
2  0.08408127 -0.26684702
3  0.21920145  0.28654229
4  0.79109191  1.11934593
5  1.32327176  0.91164204
Clustering vector:
 [1]  3 2 3 3 2 2 1 2 1 2 3 3 3 1 2 3 2 2 1 2 2 1 3 2 1 3 (行末まで省略)
[38]  1 2 3 3 3 3 1 1 1 3 1 2 3 3 4 4 5 3 5 4 5 4 5 4 5 4 (行末まで省略)
[75]  4 4 5 4 4 4 4 5 3 4 4 4 5 4 4 4 5 4 4 4 5 5 4 5 5 4
Within cluster sum of squares by cluster:
[1] 1.435313 1.074598 2.407736 2.062220 1.630782
Available components:
[1] "cluster"  "centers"  "withinss" "size"
> plot(x, col = cl$cluster)                       # 点位置をクラスタ毎に色分けして表示
> points(cl$centers, col = 1:5, pch = 8)          # クラスタ中心をアステリスクで表示
```



k-means 法によるクラスタリング例．クラスタ中心が 2 つの例 (左)．クラスタ中心が 5 つの例 (右)

5.1.12 推奨パッケージ cluster

推奨パッケージ cluster はクラスタリング専用のパッケージであり，R 本体のクラスタリング用関数を拡張，代替する関数を提供する．主要な関数には次のようなものがある．

agnes Kaufman & Rousseeuw の第 5 章に解説されている「集積的ネスティング (agglomerative nesting)」を実装．hclust() と比較すると，クラスタリング構造を計る集積係数 (agglomerative coefficient) を計算し，そのバナー表示を提供

clara データを k 個のクラスターで表示するリストである "clara" オブジェクト (clustering large applications) を計算

daisy データ変数の全ての対毎の非類似度 (dissimilarity distance) を計算する．データには，数値，因子，順序付き因子が混在して良い

diana クラス "diana" のオブジェクト (分解可能な階層的クラスタリング, divisive hierarchical clustering) を計算する

fanny ファジークラスタリング (fuzzy analysis clustering) 法でデータを k 個のクラスタに分類

mona 2 値データに対する分解可能な階層的クラスタリング

pam より頑健な k-means 法 (partitioning around medoids)．カテゴリ変数では中心座標が考えられないので，「中心に近いデータ値 (medoid)」からのずれを最小化

5.2 主成分分析

5.2.1 主成分分析 prcomp()

関数 prcomp() は与えられたデータ行列に主成分分析を実行し，結果をクラス "prcomp" のオブジェクトとして返す．

書式：

```
prcomp(x, ...)
# クラス formula に対する S3 メソッド
prcomp(formula, data = NULL, subset, na.action, ...)
# 既定の S3 メソッド
prcomp(x, retx = TRUE, center = TRUE, scale. = FALSE,
       tol = NULL, ...)
predict(object, newdata, ...) # クラス prcomp に対する S3 メソッド
```

引数：

formula 応答変数を持たない公式で，数値変数のみを参照する

data オプションデータフレーム (もしくは類似物， model.frame() 参照) で，公式 formula 中の変数を含む．既定では変数は environment(formula) から取られる

subset オプションベクトル．データ行列 x から行 (観測値) を選択するのに使う

na.action データが NA を含むときの処理を指示する関数．既定では options で設定

された na.action．未設定なら na.fail()．「工場出荷時」既定値は na.omit()

... 他のメソッドに (から) 渡される引数．もし x が公式なら， scale や tol を指定できる

x 主成分分析のためのデータである，数値または複素数値行列 (もしくはデータフレーム)

retx 論理値で，回転された変数を返すべきかどうかを指示する

center 論理値で，変数を平均が 0 になるように移動するかどうかを指示する．別の方法として， x の列数に等しいベクトルを与えても良い．この値は scale 関数に渡される

scale 論理値で，解析前に変数を標準偏差が 1 になるようにスケール化するかどうかを指示する．既定値は S との互換性のために FALSE であるが，一般的にスケール化が望ましい．別の方法として， x の列数に等しいベクトルを与えても良い．この値は scale() 関数に渡される

tol それ以下の大きさの成分を除外すべきかどうかを指示する値．(成分は，もしそれらの標準偏差が第一成分の標準偏差の tol 倍以下であれば除外される)．既定である無指定では如何なる成分も除外されない．その他の tol の設定としては tol=sqrt(.Machine$double.eps} もしくは tol=0 が考えられ，実質的に定数値の成分を取り除く

object "prcomp" を継承するクラスのオブジェクト

newdata それに関する予測値を求めるオプションのデータフレームもしくは行列．もし与えられなければ，スコアが用いられる．もし元の当てはめが公式，データフレームもしくは列名を持つ行列だったならば，newdata は同じ名前の列を含むべきである．さもなければ，同じ数の列を含むべきで，同じ順序で使用される

返り値： クラス "prcomp" のリスト．次の成分を含む：

sdev 主成分の標準偏差 (つまり，共分散・相関行列の固有値の平方根であるが，実際の計算はデータ行列の特異値分解を用いて行われる

rotation 変数の負荷量行列 (つまり，列が固有値を含む行列)．関数 princomp() はこれを成分 loadings として返す

x もし retx=TRUE なら，中心化 (要求されればスケール化) され， rotation 行列で回転されたデータが返される．従って， cov(x) は対角行列 diag(sdev}^2 である．公式メソッドに対しては， na.action() で除外された値の処理に napredict() が使われる

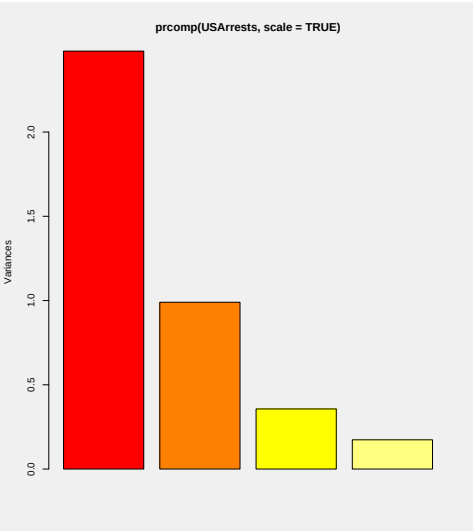
center, scale 使われた中心とスケール，もしくは FALSE

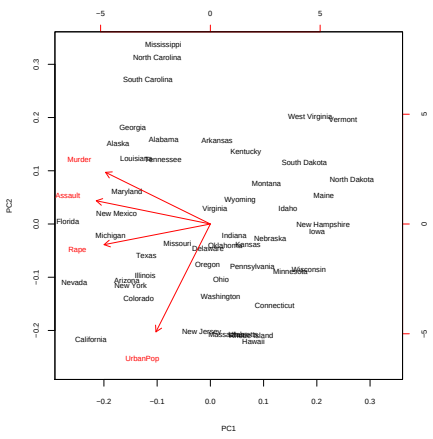
計算は (中心化，もしかするとスケール化された) データ行列の特異値分解を用いて行われ，共分散行列の固有値を使わない．これは一般的に数値精度の点で好ましい方法である．これらのオブジェクトの print メソッドは結果を体裁良く出力し，plot メソッドはスクリープロット (screeplot) を行う．もし 0 もしくは定数値 (center=TRUE の場合) の変数を持てば，scale=TRUE は使うべきではない．

注意： 回転行列の列の符号は任意であり，したがって主成分分析用の他のプログラムと異なるかも知れない．R の構築次第でも異なり得る．

関連: biplot.prcomp(), screeplot(), princomp(), cor(), cov(), svd(), eigen().

```
# 米国の犯罪検挙率データ USArrests を使用 (次の図を参照)
# このデータ中の変数の分散の大きさは非常に異なるため，スケーリングが望ましい
> prcomp(USArrests)                                  # スケーリング無し (不適当)
Standard deviations:
[1] 83.732400 14.212402  6.489426  2.482790
Rotation:
                PC1         PC2         PC3         PC4
Murder   0.04170432 -0.04482166  0.07989066 -0.99492173
Assault  0.99522128 -0.05876003 -0.06756974  0.03893830
UrbanPop 0.04633575  0.97685748 -0.20054629 -0.05816914
Rape     0.07515550  0.20071807  0.97408059  0.07232502

> prcomp(USArrests, scale = TRUE)                    # スケーリングあり
Standard deviations:
[1] 1.5748783 0.9948694 0.5971291 0.4164494
Rotation:
                PC1        PC2        PC3         PC4
Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
Rape     -0.5434321 -0.1673186  0.8177779  0.08902432

> prcomp(~ Murder+Assault+Rape,data=USArrests,scale=d TRUE) # 公式による変数指定
Standard deviations:
[1] 1.5357670 0.6767949 0.4282154
Rotation:
               PC1        PC2        PC3
Murder  -0.5826006  0.5339532 -0.6127565
Assault -0.6079818  0.2140236  0.7645600
Rape    -0.5393836 -0.8179779 -0.1999436
> plot(prcomp(USArrests))                            # 2 主成分の分散の棒グラフ

> summary(prcomp(USArrests, scale = TRUE))           # summary メソッド
Importance of components:
                          PC1   PC2    PC3    PC4
Standard deviation     1.57 0.995 0.5971 0.4164
Proportion of Variance 0.62 0.247 0.0891 0.0434
Cumulative Proportion  0.62 0.868 0.9566 1.0000
> biplot(prcomp(USArrests, scale = TRUE))  # biplot メソッド (データと変数の双方をプロット)
```





米国犯罪検挙率の 4 主成分の分散 (左) とその biplot 図 (右)

5.2.2　主成分分析 princomp()

関数 princomp() は与えられた数値データ行列に対し主成分分析を行い，結果をクラス princomp のオブジェクトとして返す．prcomp() とほぼ同じ関数であるが，S-PLUS との互換性のために設けられており，eigen() 関数を用いる．

書式：

```
# クラス formula に対する S3 メソッド
princomp(formula, data = NULL, subset, na.action, ...)
# 既定メソッド
princomp(x, cor = FALSE, scores = TRUE, covmat = NULL,
         subset = rep(TRUE, nrow(as.matrix(x))), ...)
# クラス princomp に対する S3 メソッド
predict(object, newdata, ...)
```

引数：

formula　目的変数の無いモデル式で，数値変数だけを参照する

data　モデル式 formula 中の変数を含むオプションのデータフレーム (もしくは類似物， model.frame() を参照)．既定では environment(formula)

subset　オプションのベクトル．データ行列 x 中の行 (観測値) を選択する

na.action　データが NA を含む場合の処理法を指示する関数．既定動作は options の na.action の設定に従い，もしこれが無ければ na.fail() が使われる．「工場出荷時」動作は na.omit()

x　主成分分析用のデータを提供する行列かデータフレーム

cor　論理値で，相関・共分散行列のどちらを計算で使うかを指示する．相関行列は定数値変数が無いときだけ使える

scores　論理値で，各主成分の得点 (score) を計算するかどうかを指示する

covmat　共分散行列か， cov.wt()(もしくは MASS パッケージの cov.mve() または cov.mcd()) が返すような共分散のリスト．もし与えられると x の共分散の代わりに使われる

...　他のメソッドへ (から) 引き渡される引き数．もし x がモデル式なら，cor や scores を指定しようとすればできる

object　クラス "princomp" を継承するクラスのオブジェクト

newdata　それに関して予測すべき変数を含むオプションのデータフレームか行列．もしこれが無ければ，得点が使われる．もし元の当てはめが公式，データフレームもしくは列名付きの行列を使っているならば，newdata は同じ名前の列を含まねばならない．さもなければ，同じ個数の列を含むべきで，それらが与えられた順に使われる

返り値：　クラス "princomp" のリスト．次の成分を持つ：

sdev　主成分の標準偏差

loadings　変数の負荷量の行列 (つまりその列が固有値を含む行列)．これはクラス"loadings" のオブジェクトである．その print メソッドに付いては

loadings() を参照
center 引き去られた平均値
scale 各変数に適用されたスケーリング
n.obs 観測値数
scores もし scores=TRUE なら，主成分に関する与えられたデータの得点．これは x が与えられたときだけ NULL でなく，もし更に covmat が与えられれば共分散のリストである．公式メソッドに対しては， na.action() で除外された値の処理に napredict() 関数が使われる
call マッチした呼び出し式
na.action 欠損値があった場合の処理法

princomp() はメソッド "formula" と "default" を持った総称的関数である．計算は，cor() で決められる相関行列，または共分散に対して eigen() 関数を用いて行われる．これは，S-PLUS との互換性のためである．より好ましいやり方は，prcomp() が用いる x に svd() 関数を用いる方法である．

既定の計算は，共分散に対し N-1 ではなく N で割ることを注意しよう．

これらのオブジェクトに対する print メソッドは，結果を見栄えの良い書式で表示し，plot メソッドはスクリープロット (screeplot()) を行う．また biplot メソッドもある．

もし x がモデル式なら，標準的な欠損値処理が得点 (もし要求されたなら) に対して適用される，napredict() を参照せよ．

princomp() は，変数の特徴抽出であるいわゆる R-mode(量的) PCA だけを扱う．もしデータ行列が (モデル式経由かも知れないが) 与えられると，最低変数の数と等しいだけの単位が必要となる．Q-mode(質的) PCA に付いては prcomp() を使おう．

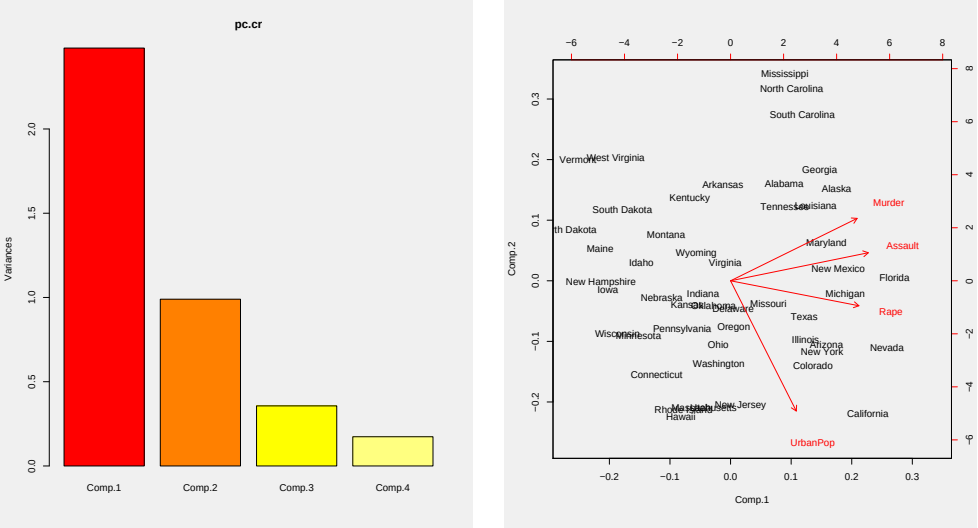
注意：負荷と得点の列の符号は任意であり，異なった主成分分析プログラムと異なるかも知れない．R の構築しだいでも異なり得る．

関連：summary.princomp(), screeplot(), biplot.princomp(), prcomp(), cor(), cov(), eigen().

```
# 米国の犯罪検挙率データ USArrests を使用 (以下の図を参照)
# データの変量の分散は大きく異なるので，スケーリングが適当
> (pc.cr <- princomp(USArrests))                    # 不適当
> princomp(USArrests, cor = TRUE)           # prcomp(USArrests,scale=TRUE) とほぼ等しい

# 似ているが異なる，標準偏差は sqrt(49/50) だけ異なる
> summary(pc.cr <- princomp(USArrests, cor = TRUE))
  Importance of components:
                          Comp.1    Comp.2    Comp.3     Comp.4
Standard deviation     1.5748783 0.9948694 0.5971291 0.41644938
Proportion of Variance 0.6200604 0.2474413 0.0891408 0.04335752
Cumulative Proportion  0.6200604 0.8675017 0.9566425 1.00000000

> loadings(pc.cr)                                   # 空の項は小さいが 0 ではないことを注意
Loadings:                                           # 固有値ベクトル
         Comp.1 Comp.2 Comp.3 Comp.4
Murder    0.536  0.418 -0.341  0.649
Assault   0.583  0.188 -0.268 -0.743
UrbanPop  0.278 -0.873 -0.378  0.134
Rape      0.543 -0.167  0.818
               Comp.1 Comp.2 Comp.3 Comp.4
```

```
SS loadings      1.00   1.00   1.00   1.00
Proportion Var   0.25   0.25   0.25   0.25
Cumulative Var   0.25   0.50   0.75   1.00

> plot(pc.cr)                                    # スクリープロットを表示
> biplot(pc.cr)                                  # バイプロット
```

```
# モデル式によるインタフェイス (+ 欠損値処理を行う)
> USArrests[1, 2] <- NA                          # わざと欠損値を入れる
> pc.cr <- princomp(~ Murder + Assault + UrbanPop,
                    data = USArrests, na.action=na.exclude, cor = TRUE)
> pc.cr$scores                                   # 因子得点
 Alabama              NA          NA            NA  # 結果も欠損
 Alaska       0.80197919  1.42047055 -0.642322193
 (途中省略)
 Wyoming     -0.31663111  0.30068300 -0.131356659
```



princomp() による米国犯罪検挙率の主成分分析．主成分分散 (左) とバイプロット (右)

5.2.3　主成分分析用のバイプロット biplot.princomp()

関数 biplot.princomp() は princomp もしくは prcomp() 関数の出力から，狭義のバイプロットグラフを描く．

書式：

```
# クラス prcomp に対する S3 メソッド
biplot(x, choices = 1:2, scale = 1, pc.biplot = FALSE, ...)
# クラス princomp に対する S3 メソッド
biplot(x, choices = 1:2, scale = 1, pc.biplot = FALSE, ...)
```

引数：

x　　クラス "princomp" のオブジェクト

choices　長さ 2 のベクトルで，プロットされる成分を指定する．既定値だけが狭義のバイプロットである

scale　変数は lambda^scale で，観測値は lambda^(1-scale) でスケール化される．ここで lambda は princomp() により計算された特異値である．普通 0 ≤

	scale ≤ 1 であり，この範囲を越えると警告がでる
pc.biplot	もし真なら，Gabriel が principal component biplot と呼んだものに相当し， lambda=1，観測値は sqrt(n) でスケールアップされ，変数は sqrt(n) でスケールダウンされる．変数間の内積は共分散を近似し，観測値間の距離はマハラノビス距離を近似する
...	biplot.default() 関数に引き渡されるオプションの引数

これは総称的関数 biplot() に対する一つのメソッドである．正確な定義についてはかなりの混乱がある．ここでは Gabriel の本来の定義にしたがった．Gabriel & Odoroff は同じ定義を使っているが，彼らのプロットは実際には pc.biplot = TRUE の場合に相当する．この関数は，副作用として現在のグラフィックスデバイスにプロットする．

```
> data(USArrests)                                  # アメリカの犯罪検挙率のデータ
> biplot(princomp(USArrests))                      # 主成分分析結果をバイプロット
```

関数 prcomp) (161 頁) の図を参照.

5.2.4 主成分の分散のプロット screeplot()

関数 screeplot()[*3] は主成分の数に対して分散をプロットする．これはクラス "princomp" に対する plot メソッドである.

書式：

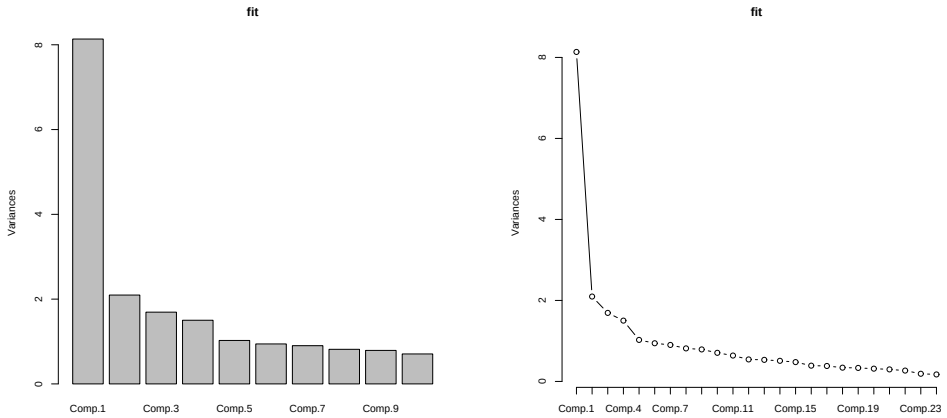
```
screeplot(x, npcs=min(10,length(x$sdev)), type=c("barplot","lines"),
          main=deparse(substitute(x)), ...)        # 既定の S3 メソッド
```

引数：

x	princomp() や prcomp() が返すような sdev 成分を含むオブジェクト
npcs	プロットすべき成分の数
type	プロットタイプ
main, ...	作図パラメータ

```
# 心理テストデータ Harman74.cor を使用 (次の図を参照)
> fit <- princomp(covmat=Harman74.cor)             # 主成分分析
> screeplot(fit)                                   # screeplot. 既定の棒グラフタイプ
> screeplot(fit, npcs=24, type="lines")            # screeplot. 折れ線グラフタイプ
```

[*3] "scree" とは地学用語で，砕石が堆積した斜面のガレ場を意味する.



screeplot による主成分毎の分散のプロット例

5.3　因子分析

5.3.1　因子分析 factanal()

共分散行列もしくはデータ行列に対し，最尤法による因子分析 (factor analysis) を実行する．

書式：

```
factanal(x, factors, data = NULL, covmat = NULL, n.obs = NA,
         subset, na.action, start = NULL,
         scores = c("none", "regression", "Bartlett"),
         rotation = "varimax", control = NULL, ...)
```

引数：

`x`　モデル式，数値行列，または数値行列に変換できるオブジェクト

`factors`　当てはめられる因子の数

`data`　オプションのデータフレーム (もしくは類似物，`model.frame()` 参照) で `x` が公式のときだけ使われる．既定では変数は `environment(formula)` から取られる

`covmat`　共分散行列か，`cov.wt()` が返すような共分散のリスト．相関行列でも良い

`n.obs`　観測値数で，`covmat` が共分散行列のとき使われる

`subset`　もし `x` が行列か公式の時，使われる部分データセットを指定する

`na.action`　もし `x` が公式の時，欠損値の処理法を与える

`start`　`NULL` か，初期値の行列で，各列が独自性 (uniqueness) の初期値を与える

`scores`　もしあれば，出力スコアのタイプ．既定では無し．`"regression"` は Thompson スコアを与える．`"Bartlett"` は Bartlett の重みつき最小 2 乗スコアを与える．文字列の一部を与えるだけで良い

`rotation`　文字列．`"none"` か，因子を回転するために使われる関数の名前．この関数は最初の引き数が負荷行列として呼び出され，回転された負荷を与える成分 `loadings` を持つ引き数か，もしくは単に回転された負荷そのものを返さなければならない

`control` 次の制御変数のリスト：
- `nstart` `start=NULL` の時，試みられる初期値の数．既定では 1
- `trace` 論理値．実行過程の情報を出力するか？既定では `FALSE`
- `lower` 最適化実行中の独自性に対する下限で正値 (既定値 0.005)
- `opt` 関数 `optim()` に引き渡される制御変数のリスト
- `rotate` 回転関数のための追加引き数リスト

`...` `control` の成分を `factanal()` への名前付き変数として引き渡すことができる

返り値： クラス `"factanal"` のリストで，以下の成分を持つ：

`loadings` 因子負荷量の行列で，一つの列が各因子に対応する．因子は負荷量の 2 乗和に関して降順に並べられ，負荷量の和が正になるように符号を付けられる

`uniquenesses` 計算された独自性

`correlation` 使われた相関行列

`criteria` 最適化の結果．負対数尤度の値と繰り返しに関する情報

`factors` 引き数 `factors`

`dof` 因子分析モデルに対する自由度の数

`method` メソッド，常に `"mle"`

`scores` もし必要とされたなら，得点の行列

`n.obs` もし得られるなら観測値の数，さもなければ `NA`

`call` マッチした呼び出し式

`na.action` もし関係するなら

`STATISTIC,PVAL` もし計算可能なら，有意性検定統計量と p 値

因子分析モデルは，n 個体に関する p 個の観測値ベクトルを表す $p \times n$ 観測値行列 x，$p \times k$ 因子負荷 (loading) 行列 Λ，$p \times k$ 個の因子得点 (score) 行列 f，そして $p \times n$ 個の誤差行列 (独自因子) ε からなる次のようなモデルである：

$$x = \Lambda f + \varepsilon$$

x 以外全て観測されないが，主な制約として，因子得点は無相関で単位分散，p 個の誤差ベクトルは互いに独立で分散ベクトル (独自性，uniquness) Φ を持つ，と仮定する．このように，因子分析は本質的に x の共分散行列

$$\Sigma = \Lambda^t \Lambda + \Psi$$

に対するモデルである (Ψ は Φ を対角成分に持つ対角行列)．もし Λ を $p \times p$ 直交行列 G を用いて $G\Lambda$ と変更してもモデルは変わらないから，依然として未確定さが残る．こうした行列 G は回転 (rotation) と呼ばれる (この言葉は必ずしも直交でない正則行列に対しても使われることがある)．

もし `covmat` が与えられれば，それが使われる．さもなければ，もしそれが行列であれば，`x` から共分散行列が作られる．もしくは，モデル式 `x` が「データ」に適応され，それから共分散行列が作られる．モデル式における目的変数は意味が無い，全ての変数は数値である必要がある．共分散が与えられるか，`x` から計算されると，それは解析のために相関行列に変換される．この相関行列は結果の `correlation` 成分として返される．

当てはめは，独自因子が多変量正規分布に従うとして，対数尤度を最適化することにより行われる．与えられた独自因子に対する因子負荷の最大化は解析的に求めることができる (Lawley & Maxwell)．start で与えられた全ての初期値が順に試めされ，得られた最良のものが使われる．もし start = NULL ならば，最初の当てはめは Jl‘eskog が提案し，Lawley & Maxwell で与えられている初期値を用い，それから，独自因子が全て等しいとしてランダムに選んだ control$nstart-1 個の他の値が試される．

独自性は技術的に区間 $[0, 1]$ にあるとされるが，0 に近い値は問題を引き起こしやすいので，最適化は既定値が 0.005 である 下限 control$lower を用いて行われる (Lawley & Maxwell).

因子得点はデータ行列が与えられ，そして使われたときだけ計算できる．最初の方法は Thomson の回帰法であり，二つ目は Bartlett の重みつき最小自乗法である．両者は未知の得点 f の推定値である．Thomson の方法は未知の f を (母集団で) s に回帰し

$$\widehat{f} = \Lambda^t \Sigma^{-1} x$$

とし，それから右辺の量の標本推定値を代入する．Bartlett の方法は標準誤差の 2 乗和を，(当てはめられた) Λ を与えた上で，f に付いて最小化する．

もし x がモデル式なら，標準的な欠損値処理が得点 (必要とされるなら) に対して適用される，napredict() を参照せよ．

注意：因子分析には無数の変種があり，他のプログラムの出力と比較するのは困難である．更に，因子分析における最大尤度の計算は困難であり，我々が比較した多くの他の例は，この関数による当てはめより劣る結果を示した．特に，Heywood のケース (一つもしくは複数の独自性が本質的に 0) である解は，多くのテキストや他のプログラムがそう思わせるよりも，はるかに普通に起こる．

関 連：print.loadings(), varimax(),princomp(),ability.cov, Harman23.cor, Harman74.cor.’

```
# テストデータベクトル
> v1 <- c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 4, 5, 6)
> v2 <- c(1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 3, 4, 3, 3, 3, 4, 6, 5)
> v3 <- c(3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 4, 6)
> v4 <- c(3, 3, 4, 3, 3, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 5, 6, 4)
> v5 <- c(1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 6, 4, 5)
> v6 <- c(1, 1, 1, 2, 1, 3, 3, 3, 4, 3, 1, 1, 1, 2, 1, 6, 5, 4)
> m1 <- cbind(v1, v2, v3, v4, v5, v6)            # 列ベクトルとして行列に結合

> cor(m1)                                        # 相関行列
          v1        v2        v3        v4        v5        v6
v1 1.0000000 0.9393083 0.5128866 0.4320310 0.4664948 0.4086076
v2 0.9393083 1.0000000 0.4124441 0.4084281 0.4363925 0.4326113
v3 0.5128866 0.4124441 1.0000000 0.8770750 0.5128866 0.4320310
v4 0.4320310 0.4084281 0.8770750 1.0000000 0.4320310 0.4323259
v5 0.4664948 0.4363925 0.5128866 0.4320310 1.0000000 0.9473451
v6 0.4086076 0.4326113 0.4320310 0.4323259 0.9473451 1.0000000
```

```
> factanal(m1, factors = 3)                      # 因子数 3 で因子分析実行
Call:
factanal(x = m1, factors = 3)
Uniquenesses:                                    # 独自性
   v1    v2    v3    v4    v5    v6
0.005 0.101 0.005 0.224 0.084 0.005
Loadings:                                        # 因子負荷量
   Factor1 Factor2 Factor3
```

```
v1 0.944   0.182   0.267
v2 0.905   0.235   0.159
v3 0.236   0.210   0.946
v4 0.180   0.242   0.828
v5 0.242   0.881   0.286
v6 0.193   0.959   0.196
               Factor1 Factor2 Factor3
SS loadings      1.893   1.886   1.797
Proportion Var   0.316   0.314   0.300
Cumulative Var   0.316   0.630   0.929
# 因子モデルの自由度と，負の最大尤度値
The degrees of freedom for the model is 0 and the fit was 0.4755
```

```
> factanal(m1, factors = 3, rotation = "promax")  # promax 回転を指定して再解析
Call:
factanal(x = m1, factors = 3, rotation = "promax")
Uniquenesses:                                     # 独自性（同じ値）
   v1    v2    v3    v4    v5    v6
0.005 0.101 0.005 0.224 0.084 0.005
Loadings:                                         # 負荷量（大きく異なる）
   Factor1 Factor2 Factor3
v1          0.985
v2          0.951
v3                  1.003
v4                  0.867
v5  0.910
v6  1.033
               Factor1 Factor2 Factor3            # ほぼ同じ値
SS loadings      1.903   1.876   1.772
Proportion Var   0.317   0.313   0.295
Cumulative Var   0.317   0.630   0.925
The degrees of freedom for the model is 0 and the fit was 0.4755
```

```
# モデル式を用いた因子分析（目的変数は指定する必要は無い）．得点だけ出力
> factanal(~v1+v2+v3+v4+v5+v6,factors=3,scores="Bartlett")$scores
        Factor1    Factor2    Factor3
  1  -0.9039949 -0.9308984  0.9475392
  2  -0.8685952 -0.9328721  0.9352330
  （途中省略）
  18  1.8822320  0.3086244  1.9547752
```

```
# 知的能力検査結果データ ability.cov を使用
> ability.FA <- factanal(factors = 1, covmat = ability.cov)

# 検定結果（1 因子帰無仮説は棄却される）
Test of the hypothesis that 1 factor is sufficient.
The chi square statistic is 75.18 on 9 degrees of freedom.
The p-value is 1.46e-12
> update(ability.FA, factors = 2)
（途中省略）

# 検定結果（2 因子帰無仮説は棄却されない）
Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 6.11 on 4 degrees of freedom.
The p-value is 0.191

# promax 回転を指定して再実行（update 関数の使用に注意）
> update(ability.FA, factors = 2, rotation = "promax")
（途中省略）

# 検定結果（やはり 2 因子帰無仮説は棄却されない）
Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 6.11 on 4 degrees of freedom.
The p-value is 0.191
```

5.3.2 因子分析用の回転 varimax(), promax()

因子分析用の回転メソッド．

書式：

```
varimax(x, normalize = TRUE, eps = 1e-5)
promax(x, m = 4)
```

引数：

x　　負荷行列で pxk, k<=p

m　　promax() の対象に対して使われた巾指数．値 2 から 4 が推薦される

normalize　論理値．Kaiser の正規化を行うべきか？もしそうなら， x の行は回転前に単位長に再スケーリングされ，後からスケールを元に戻される

eps　停止のための許容値．特異値の和における相対的な変化

返り値：　これらは負荷行列の構造を明らかにするための，因子の回転 x %*% T を求める．行列 T は varimax に対する回転 (反転を含む可能性) であるが，promax に対しては，因子の分散を保存するような, 一般の線形変換である．以下の成分を持つ：

loadings　回転された負荷行列 x%*%rotmat

rotmat　回転行列

```
> data(swiss)                                          # スイスの民生データ swiss 使用
# normalize=T の varimax 回転が既定値
> fa <- factanal( ~., 2, data = swiss)

> varimax(fa$loadings, normalize = FALSE)              # varimax 回転
$loadings                                              # 回転後の負荷量
                              Factor1              Factor2
Fertility        -0.649525084611467296  0.39805716121299634
Agriculture      -0.628122929947150577  0.33729925943080963
Examination       0.681274816678493433 -0.51530176873294109
Education         0.996756360319238199 -0.03848783395165099
Catholic         -0.117228035648530779  0.96205333457583009
Infant.Mortality -0.093396816199295879  0.17551695924190708
$rotmat                                                # その回転行列
                         [,1]                  [,2]
[1,] 0.99997388059112357616 -0.007227595418207540
[2,] 0.00722759541820755524 5  0.999973880591123576
> det(varimax(fa$loadings, normalize = FALSE)$rotmat)
 [1] 1                                                 # 行列式は 1(確かに回転)
```

```
>  promax(fa$loadings)                                 # promax 回転
$loadings                                              # 変換後の負荷量
                             Factor1              Factor2
Fertility        -0.59534907952360439  0.22697876522158150
Agriculture      -0.59892649312885127  0.16031145308224679
Examination       0.57684287591827432 -0.35956061733449363
Education         1.19227628645430772  0.36276771803372737
Catholic          0.32595021107552330  1.14692952536805226
Infant.Mortality -0.02802225529798414  0.17958353555697068
$rotmat                                                # その変換
                       [,1]                 [,2]
[1,] 1.21140451937334648 0.40292959570274711
[2,] 0.49561987503669275 1.24530627196540578

> det( promax(fa$loadings)$rotmat)                     # 変換行列の行列式
[1] 1.308869729992086                                  # 回転ではない
```

5.3.3 因子分析負荷量の出力 loadings()

関数 loadings() は因子分析に於ける負荷量 (loading) を出力する．

書式：
```
loadings(x)
# クラス "loadings" に対する print メソッド
print(x, digits = 3, cutoff = 0.1, sort = FALSE, ...)
# クラス "factanal" に対する print メソッド
print(x, digits = 3, ...)
```

引数：

x	クラス "factanal" または "princomp"，もしくはそうしたオブジェクトの loadings 成分
digits	独自性と負荷量の出力に用いられる有効桁数
cutoff	絶対値でこれ以下の負荷量は出力されない
sort	論理値．もし TRUE なら変数は，各因子に対するそれらの重要性に応じてソートされる．絶対値で 0.5 以上の負荷量を持つ各変数は最大の負荷量を持つ因子とされ，変数は割り当てられた因子の大きさに応じて出力され，次いで割り当てられないものが続く
...	他のメソッドに対する追加引き数．例えば print.factanal() に対する cutoff や sort 等

返り値： 因子分析 (または主成分分析) に於ける負荷量を取り出す，出力する

5.3.4 正準相関 cancor()

関数 cancor() は二つのデータ行列間の正準相関係数を計算する．複数の目的変数を同時に考える重回帰分析と考えることができる．

書式：cancor(x, y, xcenter = TRUE, ycenter = TRUE)

引数：

x	nxp1 数値行列で， x 値を含む
y	nxp2 数値行列で， y 値を含む
xcenter	論理値か，長さ p1 の数値ベクトルで，解析前に x 値に適用される任意のセンタリングを記述する．既定の TRUE なら行和を差し引く．FALSE なら列をそのままにする．さもなければ，列から引かれるべき値のベクトルを意味する
ycenter	xcenter に類似するが y の値用である

返り値： 以下の成分を含むリスト：

cor	相関値
xcoef	x 変数に対する推定係数
ycoef	y 変数に対する推定係数
xcenter	x 変数の補正用に使われた値

ycenter　y 変数の補正用に使われた値

正順相関は，x 変数の線型結合でうまく説明できる y 変数の線型結合を探す．「うまく説明できる」かどうかは相関値ではかられるため，この関係は対称である．

```
# LifeCycleSavings は 5 変数に関する 50 個の観測値からなるデータセット
> x <- LifeCycleSavings[, 2:3]                    # 第 2,3 変数分のデータ
> y <- LifeCycleSavings[, -(2:3)]                 # 第 1,4,5 変数分のデータ
> cancor(x, y)                                    # 正準相関実行
$cor                                              # 正準相関値
[1] 0.8247966 0.3652762
$xcoef                                            # 第 2,3 変数の線形結合の係数
              [,1]        [,2]
pop15 -0.009110856 -0.03622206
pop75  0.048647514 -0.26031158
$ycoef                                            # 第 1,4,5 変数の線形結合の係数
             [,1]          [,2]          [,3]
sr   0.0084710221  3.337936e-02 -5.157130e-03
dpi  0.0001307398 -7.588232e-05  4.543705e-06
ddpi 0.0041706000 -1.226790e-02  5.188324e-02
$xcenter                                          # 第 2,3 変数から引き去った値
  pop15   pop75
35.0896  2.2930
$ycenter                                          # 第 1,4,5 変数から引き去った値
       sr       dpi      ddpi
   9.6710 1106.7584    3.7576

> xx <- data.matrix(x)                        # 以下の演算のためにデータフレームを行列にする
> yy <- data.matrix(y)
# x,y 変数の線型結合の相関係数は (数値精度の範囲内で) 正準相関値に等しい
> all(abs(cor(xx %*% cxy$xcoef, yy %*% cxy$ycoef)[,1:2]
    - diag(cxy $ cor)) < 1e-15)
[1] TRUE

# xx%*%cxy$xcoef の相関行列は (数値精度の範囲内で)2 次元単位ベクトル
> all(abs(cor(xx %*% cxy$xcoef) - diag(2)) < 1e-15)
[1] TRUE
# yy%*%cxy$ycoef の相関行列は (数値精度の範囲内で)3 次元単位ベクトル
> all(abs(cor(yy %*% cxy$ycoef) - diag(3)) < 1e-15)
[1] TRUE
```

5.4　多次元尺度法，MDS (multidimensional scaling)

5.4.1　古典的 (距離) 多次元尺度法 cmdscale()

関数 cmdscale() はデータ行列の多次元尺度法用の関数である．principal coordinates analysis という名前で呼ばれることがある．

書式： cmdscale(d, k = 2, eig = FALSE, add = FALSE, x.ret = FALSE)
引数：
d　dist() が返すような距離構造，もしくは非類似度 (dissimilarity) を含む完全な対称行列
k　データをその中に表現する空間の次元．1:(n-1) 中の数でなければならない
eig　固有値を返すべきかどうかを指示する
add　加法定数 c* を返すかどうかを指示する論理値．TRUE なら，全ての n-1 個の固有値が非負になるように，この定数が非類似度行列の非対角成分に加えられる
x.ret　二重に中心化 (doubly centered) された対称距離行列を返すかどうかを指示

返り値： もし，`eig=TRUE` で `x.ret=FALSE` なら (既定動作)，k 列の行列で，その行は 非類似度を表現するために選ばれた点の座標を与える．さもなければ，返り値は以下の成分を含むリストである：

`points` k 列の行列で，その行は非類似度を表す点の座標値を与える．

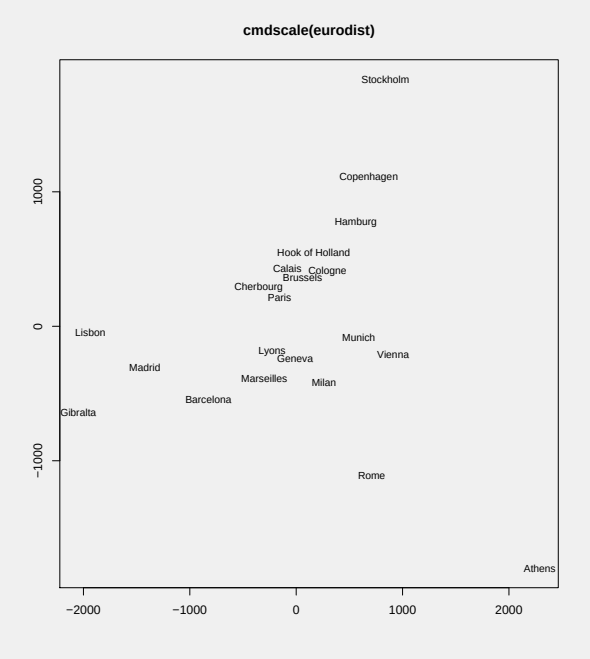
`eig` `eig=TRUE` の際返される，スケーリングの際に計算された `n-1` 個の固有値

`x` `x.ret=TRUE` の際返される，二重に中心化された距離行列

`GOF` 長さ 2 の数値ベクトル (`g1,g2`) で，`lambda` を減少順にソートされた固有値のベクトルとすると，`g1=sum(lambda)/sum(abs(lambda))`, `g2=sum(lambda)/sum(max(lambda,0))`

多次元尺度法 (metric MDS, multidimensional scaling) は非類似度の集まりから，点間のユークリッド距離が非類似度を近似[*4] するような，点配置を返す．もし `add=TRUE` なら，加法定数 c^* が計算され，非類似度 $d_{ij} + c^*$ が元々の非類似度 d_{ij} の代わりに使われる．S ではこの定数は Torgerson の方法で計算するが，R は Cailliez による解析的な解を使う (Cox & Cox も参照せよ)．距離行列は，データ行列から関数 `dist()` で計算できる．

```
# ヨーロッパ主要都市間距離データ eurodist を使う
> loc <- cmdscale(eurodist)                          # 多次元尺度法実行
> x <- loc[,1]                                       # x 座標ベクトルの取り出し
> y <- -loc[,2]                  # y 座標ベクトルの取り出し (左右逆転のためマイナス化)
> plot(x, y, type="n", xlab="", ylab="", main="cmdscale(eurodist)")
> text(x, y, names(eurodist), cex=0.8)               # 点位置に都市名を描く
```



古典的距離多次元尺度法による，ヨーロッパ都市間距離データからの都市配置の復元

[*4] パッケージ `MASS` 中の関数 `isoMDS()` と `sammon()` は，別種の並べ方 (non-metric MDS, 非類似度の大小関係だけを表現) を与える．パッケージ `smacof` は metric, non-metric MDS 用である．

5.5　補助関数

5.5.1　距離行列を計算する dist()

この関数はデータ行列の各行間の距離を，指定された各種距離を用いて計算し，距離行列を返す.

書式：
```
dist(x, method = "euclidean", diag = FALSE, upper = FALSE)
as.dist(m, diag = FALSE, upper = FALSE)
# クラス "dist" に対するメソッド
print(x, diag = NULL, upper = NULL, digits = getOption("digits"),
      justify = "none", right = TRUE, ...)
as.matrix(x) # クラス "dist" に対するメソッド
```

引数：

x　数値行列，データフレーム，または "dist" オブジェクト

method　使われる距離の定義．"euclidean", "maximum", "manhattan", "canberra" もしくは "binary" のいずれか．曖昧な指定で良い

diag　論理値で，距離行列の対角要素を print.dist() で出力するかどうかを指示する

upper　論理値．距離行列の上三角部分を print.dist() で出力するかどうかを指示

p　Minkowski 距離の巾指数

m　"dist" オブジェクトに変換されるべき距離情報を持つオブジェクト．既定メソッドでは，"dist" オブジェクト，距離の行列，もしくは as.matrix() でそうした行列に変換できるオブジェクト．(下三角部分だけが使われる)

digits, justify　print() 関数の内部で format に引き渡される

right, ...　他のメソッドに引き渡される追加引き数

返り値：　クラス "dist" のオブジェクト．距離行列の下三角部分の列順でベクトルとしたものを例えば do とし，n を観測値総数 n <- attr(do, "Size") とすると，$i < j \leq n$ で第 i, j 行間の非類似度は do[n*(i-1)-i*(i-1)/2+j-i] である．ベクトルの長さは $n(n-1)/2$ で，n^2 のオーダである.. このオブジェクトは ("dist" に等しい "class" 属性の他に) 次の属性を持つ：

Size　整数．データセット中の観測値総数

Labels　もしあれば，データセット中の観測値のラベル

Diag, Upper　上の引き数 diag, upper に対応する論理値で，オブジェクトの出力法を指定する

call　オプション．オブジェクトを生成した呼び出し式

methods　オプション．使われた距離定義．メソッド dist() に由来する

指定できる距離の種類は (二つのベクトル x, y に対して表せば) 次のようになる：

euclidean　L^2 ノルム: sqrt(sum((x-y)^2))

maximum　sup ノルム: max(abs(x-y))

`manhattan` L^1 ノルム: `sum(abs(x-y))`
`canberra` `sum(abs(x-y)/(abs(x)+abs(y)))`．分子・分母がともにゼロの項は和から除外され，欠損しているかのように扱われる
`binary` (asymmetric binary と呼ばれることもある)．ベクトルを 2 進法でビット表現した際の，値 (0 か 1) が異なるビットの割合
`minkowski` L^p ノルム: `(sum((abs(x-y))^p))^(1/p)`

欠損値があっても良く，それを含む行は全ての計算から除外される．更に，もし `Inf` 値が含まれれば，距離が `NaN` や `NA` となるようならば全ての値の対が除外される．もしある列がユークリッド，マンハッタン，キャンベラ距離の計算時に除外されると，和は使われた列数に応じてスケール化される．もしある距離の計算時に，全ての対が除外されれば，値は `NA` になる．関数 `as.matrix.dist()` と `as.dist()` は，クラス `"dist"` のオブジェクトから通常の距離行列間の変換，そしてその逆に使える．

`as.dist()` は総称的関数である．その既定メソッドはクラス `"dist"` を継承するもしくは `as.matrix()` で行列に変換可能なオブジェクトを処理する．距離 (不一致度，dissimilarities) を表すクラスへのサポートは，対応する `as.matrix()` 関数か，もっと直接にそうしたクラスに対する `as.dist()` 関数を用意することで可能になる．

関連： `hclust()`，[連続値と類別値が混在する場合を扱う パッケージ `cluster` 中の関数 `daisy()`.

注意: 二つの n 次元ベクトル $x = \{x_i\}, y = \{y_i\}$ の Canberra 距離は

$$d(x, y) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i| + |y_i|}$$

で定義される．ここで $0/0 = 0$ とする．Canberra 距離は x,y の双方が零ベクトルに近いとき，それらの僅かな変化で値が大きく変わる．常に $d(x, y) \leq n$ である．

```
> x <- matrix(rnorm(100), nrow = 5)               # テスト用データ行列
> dist(x)
         1        2        3        4             # 距離行列（ユークリッド距離）
2 5.671858                                       # 下三角部分だけ表示
3 6.753511 6.758183
4 5.876094 7.594390 6.601465
5 5.709532 6.079841 5.603563 5.731869

> dist(x, diag = TRUE)                            # 対角線部分も表示
         1        2        3        4        5
1 0.000000
2 5.671858 0.000000
3 6.753511 6.758183 0.000000
4 5.876094 7.594390 6.601465 0.000000
5 5.709532 6.079841 5.603563 5.731869 0.000000

> dist(x, upper = TRUE)                           # 上・下三角部分をともに表示
         1        2        3        4        5
1          5.671858 6.753511 5.876094 5.709532
2 5.671858          6.758183 7.594390 6.079841
3 6.753511 6.758183          6.601465 5.603563
4 5.876094 7.594390 6.601465          5.731869
5 5.709532 6.079841 5.603563 5.731869
```

```
> x <- c(0, 0, 1, 1, 1, 1)
> y <- c(1, 0, 1, 1, 0, 1)
```

```
> dist(rbind(x, y), method = "binary")            # x,y 間のバイナリ距離
[1] 0.4

> dist(rbind(x, y), method = "canberra")          # x,y 間のキャンベラ距離
[1] 2.4

> x = 1:4; y = 5:8                                # テスト用ベクトル x,y
> dist(x, y)                                      # 誤った使い方!
Error in pmatch(x, table, duplicates.ok) :
        argument is not of mode character
> rbind(x,y)                                      # ベクトルの行結合
  [,1] [,2] [,3] [,4]
x    1    2    3    4
y    5    6    7    8
> dist(rbind(x,y))                                # これが x,y 間の距離そのもの
[1] 8

> cbind(x,y)                                      # 列結合
     x y
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8
> dist(cbind(x,y))                              # cbind(x,y) の 4 つの行ベクトル間の距離行列
         1        2        3
2 1.414214
3 2.828427 1.414214
4 4.242641 2.828427 1.414214
```

5.5.2　多変量データのバイプロット biplot()

関数 biplot() は多変量データのバイプロットグラフを描く．

書式：

```
biplot(x, y, var.axes = TRUE, col, cex = rep(par("cex"), 2),
       xlabs = NULL, ylabs = NULL, expand = 1,
       xlim  = NULL, ylim  = NULL, arrow.len = 0.1,
       main = NULL, sub = NULL, xlab = NULL, ylab = NULL, ...)
```

引数：

x　当てはめオブジェクト．biplot.default() に対しては最初の点の集まり (列数 2 の行列) で，普通，観測値に付随する

y　次の点の集まり (列数 2 の行列)

var.axes　もし TRUE なら，第二の点集合はそれらを (スケール化されていない) 軸として表現する矢印を持つ

col　長さ 2 のベクトルで，それぞれ第一と第二の点集合 (および対応する軸) の色を与える．もし一つの色だけ与えられると，それが双方の軸に対して使われる．もし，何も与えられないと，既定の色が色パレットの中から探される．もしそれがあれば，それと次の色が使われる．さもなければ，パレット中の最初の二つの色が使われる

cex　点のラベルに対して使われる文字の拡大率因子．長さ 2 のベクトルを与えれば，二つの点集合に対して異なったサイズを与えることができる

xlabs　第一の点集合のラベルとして使われる文字列ベクトル．既定値は x の行の次元名か，もしそれが無ければ数値 1:n が使われる

`ylabs` 第二の点集合のラベルとして使われる文字列ベクトル．既定値は `y` の行の次元名か，もしそれが無ければ数値 `1:n` が使われる

`expand` 第二の点を第一の点に対してプロットする際に適用される拡大率．これは，二つの点集合を物理的に比較可能にするために，スケールを調整する目的に使うことができる

`arrow.len` `var.axis=TRUE` の時にプロットされる軸上の矢印の「やじり」の長さ．`arrow.len=0` とすればやじり部分を描かない

`xlim,ylim` 第一変数セットの単位に関する x,y 軸の長さ限界

`main,sub,xlab,ylab,...` グラフィックスパラメータ

バイプロットは，多変量データ行列の観測値と変数の双方を同じプロットに表現することをねらったプロットである．バイプロットには多くの変種があり，おそらく最も広く使われているものは `biplot.printcomp()` に移植されているものであろう．関数 `biplot.default()` は二つの変数を同じ図に描く基礎となる機能を提供するだけである．`biplot()` には作図パラメータも引き渡すことができる．関数の副作用として現在のグラフィックスデバイスにプロットを描く．

5.6 判別分析 (discriminant analysis)

今一つの多変量解析の代表的古典的手法である「判別分析」(discriminant analysis) 用の関数は R の基本パッケージ中には無い．この節では，R の推奨パッケージである `MASS` パッケージ[*5] 中にある「線形判別分析」と「2 次判別分析」関数を紹介しておく．

5.6.1 線形判別関数 `lda()`

線形判別分析 (linear discriminant analysis) を行う．この関数は，クラス内共分散行列が特異かどうか細心の注意を払う．もしグループ間分散が $(\texttt{tol})^2$ 未満の変数があれば，処理は停止し，その変数が定数であると報告する．これは，問題のスケーリングが不適切である結果の可能性もあるが，定数値変数の結果であることがより尤もらしい．`predict.lda()` で上書きされない限り，`prior` を指定すると判別結果に影響を与える．重みつきのグループ間共分散行列が使用されるため，ほとんどの統計パッケージと異なり，これはまた線形判別子のそれらの空間内における回転にも影響を及ぼす．したがって，最初の幾つかの線形判別子は，グループ間の差異を事前分布で与えられる重みで強調するので，データセット中でのそれらの出現度とは異なる可能性がある．

書式：

```
lda(xm ...)
# 既定の S3 メソッド
lda(x, grouping, prior = proportions, tol = 1.0e-4,
    method, CV = FALSE, nu, ...)
```

[*5] Venables & Ripley の本 *Modern Applied Statistics with S*, Springer, のコンパニオンパッケージであり，同書中に解説がある．`MASS` はパッケージバンドル `VR` 中に含まれる 4 つのパッケージの一つで，`VR` をインストールすると，4 種類のパッケージが同時にインストールされる．利用に当たっては，予め命令 `library(MASS)` でパッケージを読み込んでおく．

```
# クラス "formula" 用の S3 メソッド
lda(formula, data, ..., subset, na.action)
# クラス "data.frame" 用の S3 メソッド
lda(x, ...)
# クラス "matrix" 用の S3 メソッド
lda(x, grouping, ..., subset, na.action = na.fail)
```

引数：

formula　groups ~ x1+x2+... の形のモデル式．つまり，目的変数はグルーピング因子で，右辺は (因子でない) 判別子を指定する

data　formula 中で指定された説明変数が優先的に選ばれるデータフレーム

x　行列，データフレーム，もしくは説明変数を含む行列．もし如何なるモデル式も主要引き数として与えられないとき必要

grouping　各観測値に対するクラスを指示する因子．もし如何なるモデル式も主要引き数として与えられないとき必要

prior　クラスへの所属可能性に対する事前確率．もし指定されないと，訓練データに対するクラス比率が使われる．もし与えるなら，確率は因子水準の順番で並べる必要がある

tol　行列が特異かどうかを判断する許容値．分散が tol^2 以下の変数と，単位分散の変数の線型結合は除外される

subset　訓練用サンプルとして使われる観測値例を指定する添字ベクトル．(もし与えられるなら，この引き数は名前付きでなければならない)

na.action　欠損値 NA があるときの処理法を指定する関数．既定では失敗させる．もう一つの選択肢は na.omit で，変数に欠損値が含まれる例があれば，それを除外する (もし与えられるなら，この引き数は名前付きでなければならない)

method　平均・分散を標準的な推定するなら "moment"，最尤推定なら "mle"，cov.mve() を使うなら "mve"，t 分布に基づく頑健推定量を使うなら "t"

CV　もし真なら，「一時に一つ取り除く」クロスバリデーションに対する結果 (クラスと事後確率) を返す．もし事前分布を推定するなら，全データセット中の比率が使われる

nu　method="t" に対する t 分布の自由度

...　他のメソッドへ (から) 引き渡される追加引き数

返り値：　もし CV=TRUE なら，返り値は成分，class (MAP 判別結果 (因子)) と posterior (クラスに対する事後確率) を含むリストである．さもなければ，返り値はクラス "lda" のオブジェクトで，以下の成分を含む：

prior　使用された事前分布

means　グループ平均

scaling　観測値を判別関数に変換する行列で，グループ内共分散行列が球系になるように正規化されている

svd　特異値で，線形判別変数に関するグループ間・内標準偏差の比を与える．これらの 2 乗は標準的な F 統計量である

N　観測値の数

call (マッチした) 関数呼び出し

注意：この関数は，最初の二つの引き数として，形式的そしてオプションのデータフレームを与えても，または行列とグルーピング因子を与えても，呼び出すことができる．他の全ての引き数はオプションであるが，もし subset と na.action 引き数が必要なら，引数名を省略してはならない．もしモデル式を主要引数として与える際は，オブジェクトは通例のように update() 関数を用いて変更できる．
注意：以下の例で使われる base パッケージの組込みデータセット iris3 は Anderson によるアイリス (西洋アヤメ) データ[*6] である．

```
> library(MASS)                                    # MASS パッケージの読み込み
> data(iris3)                                      # アイリスデータ読み込み
> train <- sample(1:150, 75)                       # 訓練データの添字 (ランダムに 75 例を選ぶ)
> Iris <- data.frame(rbind(iris3[,,1], iris3[,,2],
                           iris3[,,3]), Sp=rep(c("s","c","v"),
                           rep(50,3)))             # リストをデータフレームに変換

> table(Iris$Sp[train])                            # 訓練データを因子 (品種) 別に表集計
 c  s  v
24 23 28
# 訓練データを用いた，モデル式による線形判別実行
> ( z <- lda(Sp ~ ., Iris, prior = c(1, 1, 1)/3, subset = train))
Call:                      # 呼び出し式
lda(Sp ~ ., data = Iris, prior = c(1, 1, 1)/3, subset = train)
Prior probabilities of groups:                     # 事前確率 (等確率)
        c         s         v
0.3333333 0.3333333 0.3333333
Group means:                                       # グループ平均
  Sepal.L. Sepal.W. Petal.L.  Petal.W.
c 6.008696 2.769565 4.356522 1.3565217
s 4.933333 3.414815 1.433333 0.2296296
v 6.520000 2.968000 5.488000 2.0280000
Coefficients of linear discriminants:              # 線形判別子の係数
                LD1        LD2
Sepal.L. -0.6051669 -0.8211747
Sepal.W. -1.4893318  2.7247793
Petal.L.  2.2944885 -0.8671333
Petal.W.  2.9291641  3.5783644
Proportion of trace:
   LD1    LD2
0.9883 0.0117
```

```
# 線形判別子を用いた残りの 75 例の予測 (線形判別)
> predict(z, Iris[-train, ])$class
 [1] s s s s s s s s s s s s s s s s s s s s s s s s s c c c c c c c c c v
[37] c c c c c c c c v c c c c c c c c v v v v v v v v v v v v v v v v v v
[73] v v v
Levels: c s v

# 因子 Petal.W. を除いてやり直し
> (z1 <- update(z, . ~ . - Petal.W.))
Call:                                              # 呼び出し式 (同じに見える?)
lda(Sp ~ ., data=Iris, prior=c(1,1,1)/3, subset=train)
Prior probabilities of groups:                     # グループ毎の事前確率 (等確率指定)
        c         s         v
0.3333333 0.3333333 0.3333333
Group means:                                       # グループ別平均
  Sepal.L. Sepal.W. Petal.L.  Petal.W.
c 5.937500 2.750000 4.179167 1.2791667
s 5.030435 3.452174 1.495652 0.2260870
```

[*6] Fisher により引用されて以来有名となり，しばしば「Fisher のアイリスデータ」として引用される．base パッケージには，同一であるが形式の異なる二つの組込みデータセット iris (150 × 5 行列) と iris3 (50 × 4 × 3 配列) がある．3 種類の西洋アヤメ (Iris setosa, Iris versicolor, Iris virginica) 各々 50 個の花に付いて，4 種類の測定値，萼片の長さと幅，花弁の長さと幅，が与えられている．

```
v 6.692857 3.021429 5.603571 1.9964286
Coefficients of linear discriminants:              # 線形判別子の係数
               LD1        LD2
Sepal.L.  0.9055751  0.3902900
Sepal.W.  1.5851015 -2.5279530
Petal.L. -2.1554253  0.1300322
Petal.W. -3.0802797 -1.4416391
Proportion of trace:
   LD1    LD2
0.9912 0.0088
```

5.6.2　線形判別分析による予測 predict.lda()

多変量観測値を lda() に連動して判別する．そして，データを線形判別子に射影する．この関数はクラス "lda" に対する総称的関数 predict() の一つのメソッドである．これは適当なクラスのオブジェクト x に対して呼び出し predict() を行うか，または直接に x のクラスに無関係に呼び出し predict.lda(x) を行うことにより起動される．newdata 中の欠損値は，線形判別子が計算できなければ NA を返す．もし newdata が省略され，欠損値処理 na.action が欠損値の無視を指示していれば，これらは予測に対して無視される．このバージョンは線形予測子に主眼を置いているので，グループの重心の (prior で重みを付けた) 重みつき原点にある．

書式：

```
# クラス "lda" 用 S3 メソッド
predict(object, newdata, prior = object$prior, dimen,
         method = c("plug-in", "predictive", "debiased"), ...)
```

引数：

object　クラス "lda" のオブジェクト

newdata　判別すべき例のデータフレーム，もしくは， object がモデル式を持てば，使われた変数と同じ列名を持つデータフレーム．ベクトルは行ベクトルと見なされる．もし newdata が無ければ，引数 lda のオブジェクトの当てはめに使われたデータを探そうと試みる

prior　クラスに対する事前確率．既定では訓練データにおける比率か，引数 lda の呼び出しで設定された値

dimen　用いられる空間の次元．もしこれが min(p,ng-1) 未満であれば，最初の dimen 個の判別成分だけが使われる (method="predictive" の場合を除く)．そして，これらの次元だけが返り値の成分 x 中に返される

method　これはパラメータの推定をどうするか決める．既定の "plug-in" では通常の不偏推定量が使われ，正確な値と仮定される． "debiased" では対数事後確率の不偏推定量が使われる． "predictive" では，曖昧な事前分布を用い，パラメータ推定量が「integrate out[*7] 」される

...　他のメソッドから (へ) 引き渡される追加引数

返り値：　次の成分を持つリスト：

class　MAP(最大事後確率) 判別 (因子)

posterior　クラスに対する事後確率

x	最大 dimen 個の判別変数に関するテスト例のスコア

```
# データセット iris3 を使用する
> library(MASS)                                    # MASS パッケージ読み込み
> tr <- sample(1:50, 25)                          # 訓練用データ．25 例の添字をランダムに選ぶ

# 訓練用データ（3 種類のアヤメからそれぞれ添字 tr に該当するものを抜き出す）
> train <- rbind(iris3[tr, , 1], iris3[tr, , 2], iris3[tr, , 3])

# テスト用データ（残り，3 種類のアヤメ 25 例の 4 種類の性質）
> test <- rbind(iris3[-tr, , 1], iris3[-tr, , 2], iris3[-tr, , 3])
> cl <- factor(c(rep("s", 25), rep("c", 25), rep("v", 25))) # 種類を表す因子
> z <- lda(train, cl)                              # 線形判別解析実行

# 結果の線形判別子を用いてテスト用データを判別（つまり 4 種類の性質から種類を当てる）
> predict(z, test)$class
 [1] s s s s s s s s s s s s s s s s s s s s s s s c c c c c c c c c c c c c
[37] c c c c v c c c c c c c c c c v v v v v v v v v v v v v v v v v v v v v v v
[73] v v v
Levels: c s v

> (cl == predict(z, test)$class)                   # 判別結果は正しかったか?(正答率 74/75)
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[37]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[49]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[61]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[73]  TRUE  TRUE  TRUE
```

5.6.3 2 次判別関数 qda()

qda() は 2 次関数 (Maharanobis 距離による楕円状の判別領域を用いる) による判別分析 (quadratic discriminant analysis) を行う．この関数は標準パッケージには無く，パッケージ MASS 中にある．QR 分解を使い，もしグループ内共分散行列が特定のグループに対し特異なら，エラーメッセージを与える．

書式：

```
qda(x, ...)
# クラス "formula" 用の S3 メソッド
qda(formula, data, ..., subset, na.action)
# 既定の S3 メソッド
qda(x, grouping, prior = proportions,
    method, CV = FALSE, nu, ...)
# クラス "data.frame" 用の既定メソッド
qda(x, ...)
# クラス "matrix" 用の既定メソッド
qda(x, grouping, ..., subset, na.action)
```

引数：

formula	groups ~ x1+x2+... の形のモデル式．つまり目的変数はグルーピング因子で，右辺は (因子でない) 判別子を指定する
data	formula 中で指定された説明変数が優先的に選ばれるデータフレーム
x	行列，データフレーム，もしくは説明変数を含む行列 (もし如何なるモデル式も

主要引き数として与えられないとき必要)

grouping　各観測値に対するクラスを指示する因子 (もし如何なるモデル式も主要引き数として与えられないとき必要)

prior　クラスへの所属可能性に対する事前確率．もし指定されないと訓練データに対するクラス比率が使われる．もし与えられると，確率は因子水準の順番で並べる必要がある

subset　訓練用サンプルとして使われる観測値例を指定する添字ベクトル (もし与えられるなら，この引き数は名前付きでなければならない)

na.action　もし欠損値 NA が見つかったときの処理法を指定する関数．既定の処理は手続きを失敗させる．もう一つの選択肢は na.omit で，必要とされる変数に欠損値が含まれる例があれば，それを除外する (もし与えられるなら，この引き数は名前付きでなければならない)

method　平均・分散を標準的な推定するなら "moment"，最尤推定なら "mle"，cov.mve() を使えば "mve"，t 分布に基づく頑健推定量を使えば "t"

CV　もし真なら，「一時に一つ取り除く」クロスバリデーションに対する結果 (クラスと事後確率) を返す．もし事前分布を推定するなら，全データセット中の比率が使われることを注意する

nu　method="t" に対する自由度

...　他のメソッドへ (から) 引き渡される引き数

返り値：　クラス "qda" のリストで，次の成分を含む：

prior　使用された事前確率

means　グループ平均

scaling　各グループ i に対して， scaling[,,i] はグループ内共分散行列が球形になるように観測値を変換する配列である

ldet　偏差行列の行列式の対数値の半分のベクトル

lev　グループ因子の水準

terms　(もし formula がモデル式なら) モデル式を要約するモード表現とクラス項のオブジェクト

call　マッチした関数呼び出し

class　MAP(最大事後確率) 分類 (因子)

posterior　クラスに対する事後確率

```
# あやめデータ iris3 を使用
> library(MASS)                                          # MASS パッケージを読み込む
> tr <- sample(1:50, 25) # 訓練用データ 25 例の添字をランダムに選ぶ

# 訓練用データ (3 種類のアヤメからそれぞれ添字 tr に該当するものを抜き出す)
> train <- rbind(iris3[tr, , 1], iris3[tr, , 2], iris3[tr, , 3])

# テスト用データ (残り，3 種類のアヤメ 25 例の 4 種類の性質)
> test <- rbind(iris3[-tr, , 1], iris3[-tr, , 2], iris3[-tr, , 3])
> cl <- factor(c(rep("s", 25), rep("c", 25), rep("v", 25)))  # 種類を表す因子

> zq <- qda(train, cl)                                   # 2 次判別解析実行
> str(zq)                                                # 返り値の持つ全情報の簡易要約
List of 8
 $ prior  : Named num [1:3] 0.333 0.333 0.333           # 事前確率
  ..- attr(*, "names")= chr [1:3] "c" "s" "v"           # 対応する名前
```

```
 $ counts : Named int [1:3] 25 25 25              # 種類毎の度数
  ..- attr(*, "names")= chr [1:3] "c" "s" "v"     # 種類名
 $ means  : num [1:3, 1:4] 6.00 5.04 6.47 2.79 3.47 ... # グループ平均
  ..- attr(*, "dimnames")=List of 2               # 次元属性は行列
  .. ..$ : chr [1:3] "c" "s" "v"                  # 因子名
  .. ..$ : chr [1:4] "Sepal L." "Sepal W." "Petal L." "Petal W." # 変数名
                                                  # 観測値変換配列
 $ scaling: num [1:4, 1:4, 1:3] -1.79  0.00  0.00  0.00  1.01 ...
  ..- attr(*, "dimnames")=List of 3               # 次元属性は 3 次元配列
                                                  # 第一 (変数) 次元名
  .. ..$ : chr [1:4] "Sepal L." "Sepal W." "Petal L." "Petal W."
  .. ..$ : chr [1:4] "1" "2" "3" "4"              # 第二次元名
  .. ..$ : chr [1:3] "c" "s" "v"                  # 第三 (種類) 次元名
 $ ldet   : num [1:3] -10.70 -13.01  -8.84
 $ lev    : chr [1:3] "c" "s" "v"                 # 種類因子の 3 水準
 $ N      : int 75                                # 訓練用の観測値総数
 $ call   : language qda(x=train, grouping=cl)    # 呼び出し式
 - attr(*, "class")= chr "qda"                    # 返り値の属性

# 結果の 2 次判別子を用いてテスト用データを判別 (つまり 4 種類の性質から種類を当てる)
> predict(zq, test)$class
 [1] s s s s s s s s s s s s s s s s s s s s s s s s s c c c c c c c c c v c c
[37] c c c c v c c c c c c c c c c v v v v v v v v v v v v v v v v v v v v v v
[73] v v v
Levels: c s v
```

5.6.4 2 次判別関数による予測 predict.qda()

この関数はクラス "qda" に対する総称的関数 predict() の一つのメソッドである．これは適当なクラスのオブジェクト x に対して呼び出し predict() を行うか，または直接に x のクラスに無関係に呼び出し predict.qda(x) を行うことにより起動される．newdata 中の欠損値は，線形判別子が計算できなければ NA を返す．もし newdata が省略され，欠損値処理 na.action が欠損値の無視を指示していれば，これらは予測に対して無視される．2 次判別分析関数 qda() による判別子を用いて多変量観測値の判別を行う．

書式： # クラス "qda" に対する S3 メソッド

```
predict(object, newdata, prior = object$prior,
        method = c("plug-in","predictive","debiased","looCV"), ...)
```

引数：

object　クラス ”qda” のオブジェクト

newdata　判別すべき例のデータフレーム，もしくは，object がモデル式を持てば，使われた変数と同じ列名を持つデータフレーム．ベクトルは行ベクトルと見なされる．もし newdata が無ければ，引数 qda のオブジェクトの当てはめに使われたデータを探そうと試みる

prior　クラスに対する事前確率．既定では訓練データにおける比率か，引数 lda の呼び出しで設定された値

method　これはパラメータの推定をどうするか決める．既定の”plug-in”では通常の不偏推定量が使われ，正確な値と仮定される．”debiased”では対数事後確率の不偏推定量が使われる．”predictive”では曖昧な事前分布を用い，パラメータ推定量が「integrate out」される

...　他のメソッドから (へ) 引き渡される追加引数

返り値：　次の成分を持つリスト：

class　MAP(最大事後確率) 判別 (因子)

posterior　クラスに対する事後確率

x　　最大 dimen 個の判別変数に関するテスト例のスコア

```
# データ iris3 を使用
> library(MASS)          # MASS パッケージを読み込む
> tr <- sample(1:50, 25) # 訓練用データ 25 例の添字をランダムに選ぶ
# 訓練用データ (3 種類のアヤメからそれぞれ添字 tr に該当するものを抜き出す)
> train <- rbind(iris3[tr, , 1], iris3[tr, , 2], iris3[tr, , 3])

# テスト用データ (残り，3 種類のアヤメ 25 例の 4 種類の性質)
> test <- rbind(iris3[-tr, , 1], iris3[-tr, , 2], iris3[-tr, , 3])
> cl <- factor(c(rep("s", 25), rep("c", 25), rep("v", 25))) # 種類を表す因子

> zq <- qda(train, cl)   # 2 次判別解析実行
# 結果の 2 次判別子を用いてテスト用データを判別 (つまり 4 種類の性質から種類を当てる)
> predict(zq, test)$class
 [1] s s s s s s s s s s s s s s s s s s s s s s s s s c c c c c c (以下省略)
[39] c c c c c c c c c c c c c v v v v v v v v v v v v v v v v v v v (以下省略)
Levels: c s v

# 判別結果は正しかったか? (正答率 72/75)
# 注意：訓練用データをランダムに選ぶので毎回結果が異なる可能性がある
> (cl == predict(z, test)$class)
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
[37]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[49]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[61]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
[73]  TRUE  TRUE  TRUE
```

第 6 章
線形回帰モデル

R は回帰分析関連の関数を多数持ち，統計解析機能の中心的位置を占める．以下では，線形 (重) 回帰モデルと一般化線形回帰モデル[*1] を紹介する．現代の統計理論では，分散分析も線形回帰モデルとして処理することが普通であるため，分散分析関連の関数もここで一緒に紹介するのが適当である．射影追跡法等の現代的手法も紹介する．

6.1 線形回帰モデル

6.1.1 線形モデルを当てはめる lm()

関数 lm() は線形モデルの当てはめに使われる．回帰分析，および一元配置分散・共分散分析を行える (後者に付いては aov() 関数の方がより広範囲なインタフェイスを与える).

書式：

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

引数：

object　クラス 'lme' を継承するオブジェクトで，当てはめられた線形混合効果モデルを表す引数

formula　当てはめられるモデルのシンボリックな記述式．モデル指定の詳細は後で与えられる

data　モデル中の変数を含むオプションのデータフレーム．もし data 中に変数が見付からないと environment(formula)(典型的には lm() が呼び出された環境) から探される

subset　オプションのベクトルで，当てはめ過程で使われる観測値の部分集合を指定する

weights　当てはめ過程で使われるオプションの重みベクトル．もし指定されると，重み weights を使った重みつき最小自乗法 (つまり sum(w*e^2) を最小化する) が使われる．さもなければ通常の最小自乗法が使われる

na.action　データが NA を含むときそれを処理する関数．既定は options 中の

[*1] 推奨パッケージ nlme には，ランダムな説明変数項を持つ線形及び非線形混合モデル (mixed effect model) 用の関数がある．

na.action 設定であり，もしこれが未設定なら na.fail() が使われる．「工場出荷時」既定動作は na.omit()

method 使われるメソッド．当てはめに対しては現在 method="qr" だけがサポートされている． method="model.frame" はモデルフレームを返す (指定 model=TRUE と同じ，下を見よ)

model，x，y，qr 論理値．もし TRUE なら当てはめの成分 (モデルフレーム，モデル行列，目的変数，分解) が返される

singular.ok 論理値．もし FALSE (S の既定値であるが R では異なる) なら特異な当てはめはエラーとされる

contrasts オプションのリスト． model.matrix.default() の contrasts.arg を見よ

offset これは当てはめの途中で，線形予測子に含められるべきことが事前に知られている成分を指定するのに使える．代わりに，または同時にオフセット項をモデル式に含めても良い

... 低水準回帰当てはめ関数に引き渡される追加引き数 (以下参照)

返り値： lm() 関数はクラス "lm" のオブジェクトを返す．多変量目的変数に対してはクラス c("mlm", "lm") のオブジェクトを返す．関数 summary() と anova() は結果の出力と分散分析表を得るのに使える．総称的アクセス関数 coefficients(), effects(), fitted.values() そして residuals() は lm() が返す値から様々な有用な情報を取り出すのに使うことができる．クラス "lm" のオブジェクトは最低でも次のような成分を含むリストである．加えて，NULL でない当てはめは，summary や effects といった抽出関数用に，線形当てはめに関連する成分 assign, effects そして (不要とされない限り) qr を含む

coefficients 係数の名前付きベクトル

residuals 残差，つまり目的変数から当てはめ値を引いたもの

fitted.values 当てはめられた平均値

rank 当てはめ線形モデルの数値計算によるランク

weights (重みつき当てはめだけに該当) 指定された重み

df.residual 残差自由度

call 呼出し式

terms 使われた terms オブジェクト

contrasts (関係するときだけ) 使われた対比

xlevels (関係するときだけ) 当てはめで使われた因子の水準の記録

y もし要求されたなら使われた目的変数

x もし要求されたなら使われたモデル行列

model もし要求されたなら (既定)，使われたモデルフレーム

lm() に対するモデル式はシンボリックに与えられる．典型的なモデルは response ~ terms という形式を持ち，ここで response は (数値) 目的変数ベクトルで terms は response に対する説明変数を指定する幾つかの項である．形式 first + second の項指定は，first, second 中の全ての項 (重複項は除いて) を意味す

る．形式 first:second の項指定は，first と second 中の全ての項同士の交互作用項を意味する．形式 first*second の項指定は，first と second 中の全ての項のクロスを意味する．これは指定 first + second + first:second と同値である．もし response が行列なら，行列の全ての列に対して線形モデルが当てはめられる．詳しくは model.matrix() を見よ．モデル式中の項は主効果が最初，2 次交互作用項がその次，更に 3 次交互作用が続く，等と並べ変えられる．こうすることにより term オブジェクトをモデル式として引き渡すことを防げる．

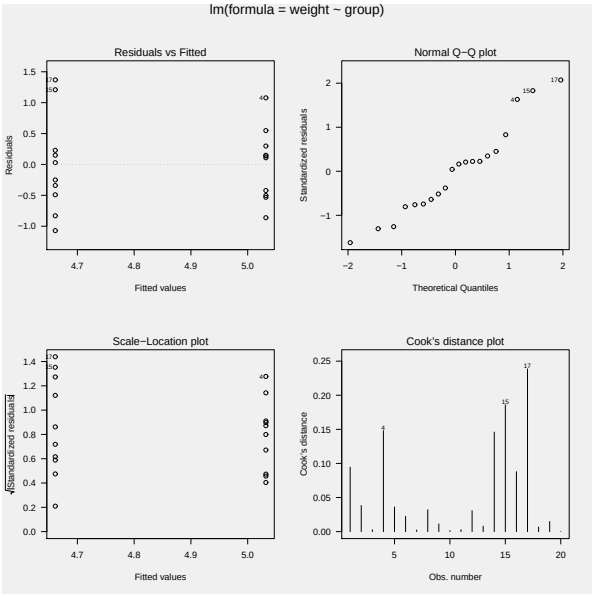
モデル式には切片項が暗黙のうちに含まれる．これを除くには y ~ x - 1 か y ~ 0 + x を使う．許されるモデル式に付いては formula() を見よ．lm() は実際の数値計算には lm.fit() 等の低水準関数を呼び出す．プログラムのためだけなら同様に低水準関数を使うことが考えられる．weights, subset そして offset 全ては formula 中の変数と同様に評価される．つまり，まず data 中，それから formula の環境中である．

注意： offset で指定されたオフセット項は predict.lm() による予測には含められない．他方で，モデル式のオフセット項で指定されたものは含められる．

関連： 要約用の summary.lm()，ANOVA 表用の anova.lm()．aov() は別のインタフェイスを与える．総称的関数 coef(), effects(), residuals(), fitted(), vcov()．信頼区間と予測区間を含む予測には predict.lm() (predict() を経由する)．回帰診断には lm.influence()，一般化線形モデルは glm()．背景にある低水準関数である，単純な当てはめ用の lm.fit()，重みつき回帰当てはめ用の lm.wfit()．

```
# Annette Dobson.  対照群と処理群の植物重量データ（次の図を参照）
> ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14) # 対照群
> trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69) # 処理群
> group <- gl(2,10,20, labels=c("Ctl","Trt"))
> group                                  # 2 群を因子化し説明変数とする（処理による差に関心）
[1] Ctl Ctl Ctl Ctl Ctl Ctl Ctl Ctl Ctl Ctl Trt Trt Trt Trt Trt Trt Trt Trt
[18] Trt Trt Trt
Levels: Ctl Trt                                  # 2 水準（Ctl:対照群，Trt:処理群）
> weight <- c(ctl, trt)                          # 目的変数

> anova(lm.D9 <- lm(weight ~ group))             # 線形回帰結果から分散分析表を計算
Analysis of Variance Table
Response: weight
          Df Sum Sq Mean Sq F value Pr(>F)
group      1 0.6882  0.6882  1.4191  0.249
Residuals 18 8.7293  0.4850

> str(lm.D9)                                     # 線形回帰結果の全情報を見る
List of 13
 $ coefficients : Named num [1:2]  5.032 -0.371
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "groupTrt"
 $ residuals    : Named num [1:20] -0.862 0.548 0.148 1.078 -0.532 ...
  ..- attr(*, "names")= chr [1:20] "1" "2" "3" "4" ...
 （以下省略）

> summary(lm.D90 <- lm(weight ~ group - 1))      # 切片項を除いた当てはめ結果の要約
Call:
lm(formula = weight ~ group - 1)                 # 呼出し式
Residuals:                                       # 残差の要約統計量
    Min      1Q  Median      3Q     Max          # 5 数要約
-1.0710 -0.4938  0.0685  0.2462  1.3690
Coefficients:              # 係数の推定値，標準偏差，t 値．p 値はともに有意（0 とは見なせない）
            Estimate Std. Error t value Pr(>|t|)
groupCtl   5.0320     0.2202   22.85 9.55e-15 ***
groupTrt   4.6610     0.2202   21.16 3.62e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.6964 on 18 degrees of freedom  # 残差の標準偏差と対応自由度
```

```
Multiple R-Squared: 0.9818,    Adjusted R-squared: 0.9798 # R2 乗値と自由度調整済み値
# 帰無仮説「傾きが 0(群による差無し)」は F 検定で棄却される
F-statistic: 485.1 on 2 and 18 DF,  p-value: < 2.2e-16
```



`lm()` 結果の回帰診断図
(上左) 残差プロット
(上右) 残差の Q-Q プロット
(下左) Scale-Location プロット
(下右) Cook 距離のプロット

6.1.2 回帰診断 `lm.influence()`

回帰当てはめの善し悪しを検査するための広範囲な診断手順に使われる基本的な量を計算する.

書式:

```
influence(model, ...)
# クラス "lm" に対する S3 メソッド:
influence(model, do.coef = TRUE, ...)
# クラス "glm" に対する S3 メソッド:
influence(model, do.coef = TRUE, ...)
lm.influence(model, do.coef = TRUE)
```

引数:

`model` `lm()` 等が返すオブジェクト

`do.coef` 論理値. 変更された `coefficients` (以下を参照) を望むか? これはオーダー $O(n^2p)$ の計算量を必要とする

`...` 他のメソッドへ (から) 引き渡される追加引き数

返り値: 次の成分を持つリスト. 各成分は同じ長さか, 行数 n と同じ長さを持ち, これは 0 で無い重みの数である. 当てはめの際に取り除かれたケースは, それらを復元する `na.action` 指定 (例えば `na.exclude`) が使われない限り, やはり取り除かれる

`hat` 「ハット」行列の対角成分を含むベクトル

coefficients (do.coef=FALSE でない限り)i 番目の行が，i 番目のケースを回帰当てはめから取り除いたために生じた推定係数の変化量であるような行列．エイリアス化された係数は行列に含まれないことを注意しよう

sigma i 番目の要素が，i 番目のケースを回帰当てはめから取り除いて得られた残差標準偏差の推定量であるようなベクトル

wt.res 重みつき残差 (もしくはクラス "glm" に対しては偏差) のベクトル

influence.measures() そして以下の関連事項中に挙げられた関数は，様々な回帰診断を計算する，よりユーザの便宜をはかった方法を提供する．これらの関数は全て lm.influence() に基づく．おそらく 1 であろうハット値は，実際 1 であるとして処理し，そして対応する sigma と coefficients の行は NaN となることを保証するように努められる．(そうしたケースを取り除くと普通ある変数が除去されることにつながり，したがって単純な「一つを取り除く」診断を与えることは不可能である．)

注意： R 版の lm.influence() が返す coefficients は S が計算するものと異なる．各ケースを除いて得られた結果である係数を返す代わりに，係数の変化量が返される．これは多くの診断量に対してより直接に有用である．$O(n^2p)$ の計算量が要るため，do.coef=FALSE ならばこれは計算されない．weights==0 であるケースは (S とは異なり) 除外される．もしモデルが na.action=na.exclude を用いて (na.exclude() を参照) 当てはめられると，当てはめで除外されたケースはここで考慮される．

関連： 関連メソッドについては summary.lm()．influence.measures()，ハット行列の対角成分については hat()，そして dfbetas()，dffits()，covratio()，cooks.distance()，lm()．

```
# 合計個人貯蓄を 15 歳以下，75 歳以上人口，一人当たり可処分所得とその増加率で説明
> data(LifeCycleSavings)                                  # 家計貯蓄率データ
> summary(lm.SR <- lm(sr ~ pop15 + pop75 + dpi + ddpi,
                      data = LifeCycleSavings), corr = TRUE)
Call:                                                     # 呼出し式
lm(formula = sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings)
Residuals:                                                # 残差の要約
    Min      1Q  Median      3Q     Max
-8.2422 -2.6857 -0.2488  2.4280  9.7509
Coefficients:                                             # 推定係数とその標準偏差，t 値，p 値
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 28.5660865  7.3545161   3.884 0.000334 ***
pop15       -0.4611931  0.1446422  -3.189 0.002603 **
pop75       -1.6914977  1.0835989  -1.561 0.125530
dpi         -0.0003369  0.0009311  -0.362 0.719173
ddpi         0.4096949  0.1961971   2.088 0.042471 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 3.803 on 45 degrees of freedom
Multiple R-Squared: 0.3385,     Adjusted R-squared: 0.2797
F-statistic: 5.756 on 4 and 45 DF,  p-value: 0.0007904
Correlation of Coefficients:                              # 係数の相関係数
      (Intercept) pop15 pop75 dpi
pop15 -0.98
pop75 -0.81        0.77
dpi   -0.17        0.18 -0.37
ddpi  -0.19        0.10 -0.05  0.26

> str(lmI <- lm.influence(lm.SR))                         # 回帰診断情報の全体の簡潔な要約
List of 4
 $ hat         : Named num [1:50] 0.0677 0.1204 0.0875 0.0895 0.0696 ...
  ..- attr(*, "names")= chr [1:50] "Australia" "Austria" "Belgium" ...
```

```
$ coefficients: num [1:50, 1:5]  0.0916 -0.0747 -0.4752  0.0429  0.6604 ...
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : chr [1:50] "Australia" "Austria" "Belgium" ...
 .. ..$ : chr [1:5] "(Intercept)" "pop15" "pop75" "dpi" ...
$ sigma       : Named num [1:50] 3.84 3.84 3.83 3.84 3.81 ...
 ..- attr(*, "names")= chr [1:50] "Australia" "Austria" "Belgium" ...
$ wt.res      : Named num [1:50]  0.864  0.616  2.219 -0.698  3.553 ...
 ..- attr(*, "names")= chr [1:50] "Australia" "Austria" "Belgium" ...
```

6.1.3　線形モデルの当てはめ関数 lm.fit()

lm.fit() は線形モデルを当てはめる lm() から呼び出される，基本計算エンジンである．経験あるユーザでなければ直接使うべきではない．

書式：

```
lm.fit(x, y, offset=NULL, method="qr", tol=1e-7,
       singular.ok=TRUE, ...)
lm.wfit(x, y, w, offset=NULL, method="qr", tol=1e-7,
        singular.ok=TRUE, ...)
```

引数：

x　次元 nxp の計画行列

y　長さ n の観測値ベクトル

w　wfit() による当てはめ過程で使われる (長さ n の) 重みベクトル．重み w の重みつき最小自乗法が使われる，つまり sum(w*e^2) が最小化される

offset　長さ n の数値ベクトル．これは当てはめ中に用いられるべきことが予め分かっている説明変数を指定するのに使うことができる

method　現在 method="qr" だけがサポートされている

tol　qr 分解に対する許容値．既定値は 1e-7

singular.ok　論理値．もし FALSE なら特異なモデルはエラーとされる

...　現在のところ無視される

返り値：　次の成分を持つリスト：

coefficients　長さ p のベクトル

residuals　長さ n のベクトル

fitted.values　長さ n のベクトル

effects　(空でない当てはめに対して) 単一自由度の直交効果の長さ n のベクトル．これらの第一 rank は別名化されていない係数に対応し，それに従う名前が付けられる

weights　長さ n のベクトル．wfit 関数に対してだけ存在する

rank　ランクを与える整数

df.residual　残差の自由度

qr　(空でない当てはめに対して)QR 分解．qr() を見よ

x が要素が全て 0 の列を含めば，関数 lm.fit.null() と lm.wfit.null() がそれぞれ lm.fit() もしくは lm.wfit() により呼び出される．

関連：良く理解していない限り，線形最小自乗法には `lm()` を使うべきである．

```
# 人工的な例
> set.seed(129)                                  # 乱数種を指定（いつも同じ結果を得るため）
> n <- 7 ; p <- 2
> X <- matrix(rnorm(n * p), n, p)                # 計画行列をランダムに生成，切片項は無い
> y <- rnorm(n)                                  # ランダムな目的変数
> w <- rnorm(n)^2                                # 重みベクトルをランダムに生成

> str(lmw <- lm.wfit(x=X, y=y, w=w))             # 重みつき最小自乗法実行，返り値を一覧
List of 9
 $ coefficients : Named num [1:2] -0.0432 -0.5612 # 係数の最小自乗推定値
  ..- attr(*, "names")= chr [1:2] "x1" "x2"
 $ residuals    : num [1:7] -0.132 -1.308 -0.256  1.468  0.439 ...    # 残差
 $ fitted.values: num [1:7] 0.0500 -0.52320.6151 -0.5766 0.0512 ...   #当てはめ値
 $ effects      : Named num [1:7] 0.804  1.722 -0.072  2.047  0.392 ... # 効果
  ..- attr(*, "names")= chr [1:7] "x1" "x2" "" "" ...
 $ weights      : num [1:7] 0.3195 0.0123 0.0569 1.8154 0.8359 ...    # 重み
 $ rank         : int 2                         # 計画行列のランク
 $ assign       : NULL
 $ qr           :List of 5                      # QR 分解情報
  ..$ qr   : num [1:7, 1:2]  3.0953  0.0355  0.1060  0.5901 -0.5898 ...
  ..$ qraux: num [1:2] 1.20 1.02
  ..$ pivot: int [1:2] 1 2
  ..$ tol  : num 1e-07
  ..$ rank : int 2
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 5                         # 残差自由度

> str(lm. <- lm.fit (x=X, y=y))                  # 重み無し最小自乗法の結果と比較
List of 8
 $ coefficients : Named num [1:2]  0.132 -0.553   # 結果は相当異なる
  ..- attr(*, "names")= chr [1:2] "x1" "x2"
 $ residuals    : num [1:7]  0.06433 -1.14333 -0.00708 1.69606 0.09095 ...
 $ effects      : Named num [1:7]  0.791  1.476 -0.415 1.983  0.133 ...
  ..- attr(*, "names")= chr [1:7] "x1" "x2" "" "" ...
 $ rank         : int 2
 $ fitted.values: num [1:7] -0.146 -0.688  0.366 -0.804  0.399 ...
 $ assign       : NULL
 $ qr           :List of 5
  ..$ qr   : num [1:7, 1:2]  3.321  0.298  0.414  0.408 -0.601 ...
  ..$ qraux: num [1:2] 1.34 1.32
  ..$ pivot: int [1:2] 1 2
  ..$ tol  : num 1e-07
  ..$ rank : int 2
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 5
```

6.1.4 lm オブジェクトの診断図 plot.lm()

lm オブジェクトの診断図を描く．現在 4 種類のプロット (`which` で選択できる) が提供されている．当てはめ値に対する残差プロット，当てはめ値に対する `sqrt(|residuals|)` の Scale-Location プロット，正規 Q-Q プロット，行ラベルに対する Cook の距離のプロットである．

書式：クラス "lm" に対する S3 メソッド：

```
plot(x, which = 1:4,
     caption = c("Residuals vs Fitted", "Normal Q-Q plot",
                 "Scale-Location plot", "Cook's distance plot"),
     panel = points, sub.caption = deparse(x$call), main = "",
     ask = prod(par("mfcol")) < length(which) && dev.interactive(),
```

```
    ..., id.n = 3, labels.id = names(residuals(x)), cex.id = 0.75)
```

引数：

x lm オブジェクト．典型的には lm() 又は glm() の結果

which もしプロットの一部分だけが必要なら数 1:4 の部分集合を指定する

caption プロットの上部に置かれる見出し

panel パネル関数．points() や panel.smooth() が有用

sub.caption 図が複数あるときその上に置かれる共通のタイトル．さもなければ，サブタイトルとされる

main 上の caption 以外の各図のタイトル

ask 論理値．もし TRUE なら，各プロットの表示前に問い合わせがある．par(ask=.) を参照

... 作図関数に引き渡される他のパラメータ

id.n 各プロット中のラベルが付けられる点の数．最も極端なものから付けられる

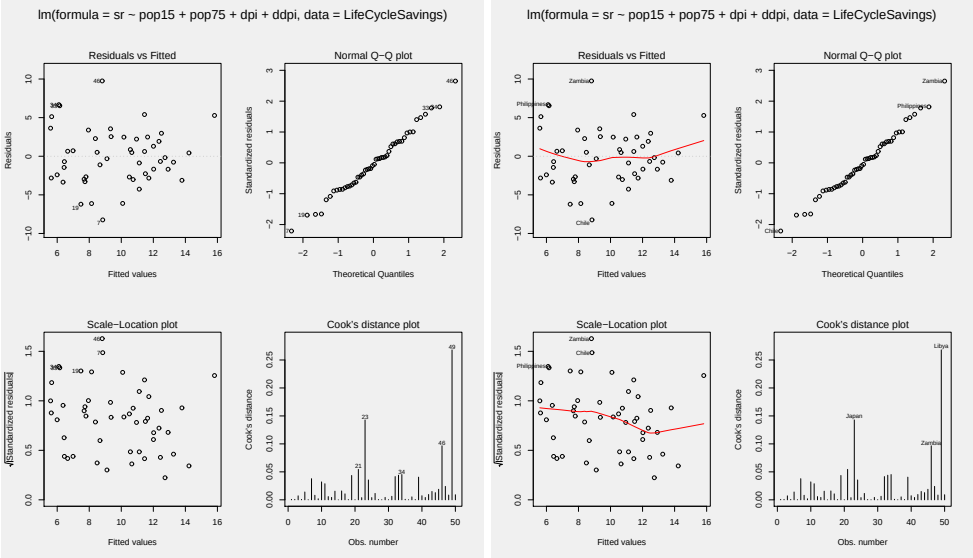
labels.id ラベルのベクトルで，極端な点のラベルはその中から選ばれる．NULL なら観測番号が使われる

cex.id 点ラベルの拡大率

既定で関数呼び出し式である sub.caption は，プロットが別の頁にあれば (x 軸タイトルの下の) サブタイトルとして表示される．もしくは，頁毎に複数のプロットがあれば外側余白 (もしあれば) 中のサブタイトルになる．「Scale-Location」 プロットは，また 「Spread-Location」 もしくは 「S-L」プロットと呼ばれ，歪みを減らすために残差の絶対値の平方根を取る (平均 0 の正規変数に対しては sqrt{|E|} は |E| よりも歪みが少ない)．S-L と Q-Q プロットは (仮定の下で) 同一分散を持つ標準化残差を使う．標準化残差は R[i]/(s*sqrt(1 - h.ii)) で与えられ，ここで h.ii はハット行列 influence()$hat の対角成分である hat() を参照せよ．

関連：termplot(), lm.influence(), cooks.distance().

```
# 合計個人貯蓄データ．15 歳以下，75 歳以上人口，一人当たり可処分所得，その増加率で説明
> data(LifeCycleSavings)                          # 家計貯蓄率データ
> lm.SR <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings)

# 一頁に 4 つの図，モデル式を余白に置くスペースを確保 (次の図を参照)
> par(mfrow = c(2, 2), oma = c(0, 0, 2, 0))
> plot(lm.SR, id.n = 5, labels.id = NULL)         # 極端な順に 5 つのデータにラベル付
> plot(lm.SR, panel = panel.smooth)               # 散布図に平滑化曲線を上描き
```



家計貯蓄率データへの線形モデル当てはめの回帰診断図．(左) 極端な順に 5 つのデータにラベル．(右) 残差散布図と S-L プロットに平滑化平滑化曲線を上描き

6.1.5 線形モデルによる予測 predict.lm()

`predict.lm()` は線形モデルオブジェクトに基づく予測メソッドである．

書式：

```
# クラス "lm" に対する S3 メソッド
predict(object, newdata, se.fit = FALSE, scale = NULL, df = Inf,
        interval = c("none", "confidence", "prediction"),
        level = 0.95, type = c("response", "terms"),
        terms = NULL, na.action = na.pass, ...)
```

引数：

`object` `"lm"` を継承するクラスのオブジェクト

`newdata` その中で予測を行うデータフレーム

`se.fit` 標準誤差が必要かどうか指示するスイッチ

`scale` 標準誤差の計算用のスケールパラメータ

`df` スケールに対する自由度

`interval` 区間計算のタイプ

`level` 許容／信頼水準

`type` 予測のタイプ (目的変数もしくはモデル項)

`terms` `type="terms"` ならどの項か (既定では全ての項)

`na.action` `newdata` 中に `NA` があるときそれを処理する関数を指定する．既定ではそのまま `NA` として扱う

`...` 他のメソッドへ (から) 引き渡される追加引き数

返り値： `predict.lm()` は予測値ベクトルもしくは行列と，`interval` が指定されていれば列名 `fit`, `lwr` そして `upr` を持つ限界を返す．もし `se.fit = TRUE` なら次の成分を持つリストが返される：

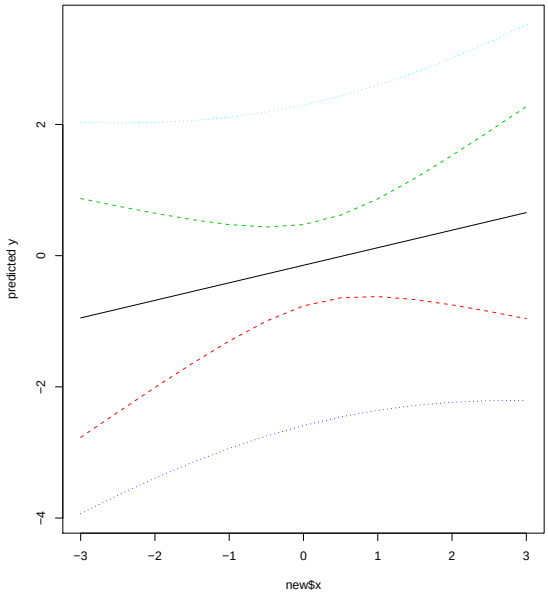
fit 上のようなベクトルもしくは行列
se.fit 予測の標準誤差
residual.scale 残差の標準誤差
df 残差の自由度

predict.lm() はデータフレーム newdata (既定では model.frame(object)) 中で回帰関数を評価して得られる予測値を求める．もし論理値 se.fit が TRUE ならば，予測値の標準偏差が計算される．もし数値引数 scale が (オプションの df とともに) 設定されると，それは標準誤差の計算過程で残差標準偏差として用いられる．さもなければこれは当てはめモデルから計算される．設定 intervals は与えられた level で信頼区間もしくは予測 (許容) 区間の計算を指定する．もし当てはめがランク落ちするときは，計画行列の幾つかの列が取り除かれる．そうした当てはめによる予測は，オリジナルのデータと同じ空間に newdata が含まれる場合だけ意味がある．これは正確にはチェックできないので，警告が出される．

注意： lm() による当てはめ中の offset で指定されるオフセットは予測値に含まれない．他方，モデル式中のオフセット項で指定された場合は含まれる．

関連： モデル当てはめ関数 lm(), predict(), SafePrediction().

```
> x <- rnorm(15); y <- x + rnorm(15)              # 人口的な例 (以下の図を参照)
> predict(lm(y ~ x))                              # x の各値で予測
         1          2          3          4          5          6
0.13509059 -0.50013084  0.11775229  0.08701150  0.15125832  0.15784952
         7          8          9         10         11         12
0.16340458 -0.29347217  0.17051878 -0.19833055 -0.60795300  0.03473943
        13         14         15
0.11293669 -0.13560638 -0.77556433

> new <- data.frame(x = seq(-3, 3, 0.5))          # 予測用の新しいデータフレーム
> predict(lm(y ~ x), new, se.fit = TRUE)
$fit                                              # new 中の値に対する予測値
         1          2          3          4          5          6
-0.94915270 -0.81534551 -0.68153832 -0.54773113 -0.41392393 -0.28011674
         7          8          9         10         11         12
-0.14630955 -0.01250236  0.12130484  0.25511203  0.38891922  0.52272641
        13
 0.65653361
$se.fit                                           # 各予測値の標準誤差
       1        2        3        4        5        6        7        8
0.842636 0.727062 0.614753 0.507881 0.410712 0.331880 0.286921 0.291929
       9       10       11       12       13
0.344733 0.427994 0.527480 0.635619 0.748670
$df                                               # 自由度
[1] 13
$residual.scale                                   # 残差の標準誤差
[1] 1.094054

> pred.w.plim <- predict(lm(y ~ x), new, interval="prediction") # 予測区間
> pred.w.clim <- predict(lm(y ~ x), new, interval="confidence") # 信頼区間
> matplot(new$x,cbind(pred.w.clim, pred.w.plim[,-1]),           # 同時にプロット
          lty=c(1,2,2,3,3), type="l", ylab="predicted y")
```



人工例に対する当てはめ直線，既定信頼水準 95% の信頼区間 (上から 2,4 番目の曲線) と予測区間 (上から 1,5 番目の曲線)

6.1.6 線形モデル当てはめの要約 summary.lm()

summary.lm() はクラス "lm" に対する summary メソッドである．

書式：

```
# クラス "lm" に対する S3 メソッド
summary(object, correlation = FALSE, symbolic.cor = FALSE, ...)
# クラス "lm" に対する S3 メソッド
print(x, digits = max(3, getOption("digits") - 3),
      symbolic.cor = x$symbolic.cor,
      signif.stars = getOption("show.signif.stars"), ...)
```

引数：

object クラス "lm" のオブジェクト．普通 lm() の呼び出し結果

x クラス "summary.lm" のオブジェクト．普通 summary.lm() の呼び出し結果

correlation 論理値．TRUE なら推定パラメータの相関行列が返され，出力される

digits 出力時に使われる有効桁数

symbolic.cor 論理値．もし TRUE なら相関を数値ではなく，シンボリックな形式で出力する (symnum() を参照)

signif.stars 論理値．もし TRUE なら各係数に「有意水準を表す星印」を添える

... 他のメソッドへ (から) 引き渡される追加引き数

返り値： 関数 summary.lm() は object で与えられた線形モデル当てはめオブジェクトの要約統計量を計算し返す．返り値は object の成分"call" と "terms" および以下の成分を持つリストである：

residuals 重みつき残差．通常の残差を lm() への呼び出し中で指定された重みの平方根で再スケール化されたもの

coefficients px4 行列で，その列は推定係数，その標準誤差，t 統計量，そして対応する (両側)p 値である．エイリアスに対する係数は除外される

aliased 名前付き論理値ベクトルで，元の係数がエイリアス化されているかどうかを示す

sigma ランダム誤差の推定分散 sum(residuals^2)/(n-p) の平方根

df 自由度の長さ 3 のベクトル c(p,n-p,p')，最後はエイリアス化されていない係数の数

fstatistic (非切片項を含むモデルに対し)F 統計量とその分母・分子の自由度からなる長さ 3 のベクトル

r.squared R2 乗値で「モデルで説明される分散の割合」で 1-sum(residuals^2)/sum((y- y')^2)．ここで y' はもし切片項があれば y の平均であり，さもなければ 0

adj.r.squared 「補整された」上の R2 乗値．大きな p 値に対しペナルティが付く

cov.unscaled (スケール化されていない) coefficients の pxp 共分散行列

correlation correlation=TRUE ならば，上の cov.unscaled に対応する相関行列

symbolic.cor (correlation=TRUE の場合だけ) 引数 symbolic.cor の値

print.summary.lm() は係数，標準偏差等を見栄え良く整形し出力する．もし signif.stars = TRUE なら更に有意水準を表す星印を添える．相関は二桁 (もしくはシンボリックに) 出力される．実際の相関を見るためには summary(object)$correlation を直接出力せよ．

関連：モデル当てはめ関数 lm(), summary()．関数 coef() は標準偏差，t 統計量，p 値を伴った係数の行列を取り出す．

```
> coef(lm.D90)                                          # lm() の例の続き．係数だけ取り出す
groupCtl groupTrt
   5.032    4.661

> ( sld90 <- summary(lm.D90 <- lm(weight ~ group -1)) ) # 切片無しのモデル当てはめ
Call:
lm(formula = weight ~ group - 1)
Residuals:
    Min      1Q  Median      3Q     Max
-1.0710 -0.4938  0.0685  0.2462  1.3690
Coefficients:
         Estimate Std. Error t value Pr(>|t|)
groupCtl   5.0320     0.2202   22.85 9.55e-15 ***
groupTrt   4.6610     0.2202   21.16 3.62e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.6964 on 18 degrees of freedom
Multiple R-Squared: 0.9818,     Adjusted R-squared: 0.9798
F-statistic: 485.1 on 2 and 18 DF,  p-value: < 2.2e-16

> coef(sld90)                                           # 係数を取り出す
         Estimate Std. Error  t value     Pr(>|t|)
groupCtl    5.032  0.2202177 22.85012 9.547128e-15
groupTrt    4.661  0.2202177 21.16542 3.615345e-14
```

6.2 一般化線形回帰モデル

一般化線形モデル (GLM, generalized linear model) は線形モデルの単純さを保ちつつ，非線形なモデリングを可能にし，更に正規分布に従う誤差という制限的な仮定も無くせるという点で，線形モデルの適用範囲を一挙に拡大する．

GLM はリンク関数[*2] と呼ばれる既知の可逆関数 $g(\mu)$, モデルパラメータベクトル $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_k)$，説明変数ベクトル $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$ を用いて形式的に次のように定式化される：

$$\mu = \boldsymbol{E}\{Y\} = g(\boldsymbol{x}^T\boldsymbol{\theta}), \quad \text{つまり } g^{-1}(\mu) = \boldsymbol{x}^T\boldsymbol{\theta}$$

具体的には $F(\rho)$ をパラメータ ρ を持つ分布族とし Y の分布が $F(g(\boldsymbol{x}^T\boldsymbol{\theta}))$ となる，と指定できる．パラメータの推定は最尤推定法を使うことが基本であり，離散分布を含む各種の分布が可能である．観測モデルは，説明変数ベクトルがそれぞれ $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ である互いに独立なデータ $y_1, y_2, \ldots, y_n$ の分布が

$$y_i \sim F(g(\boldsymbol{x}_i^T\boldsymbol{\theta})) \quad i = 1, 2, \ldots, n$$

となる．パラメータ値 ρ の値の範囲に正値等の制限があっても，パラメータ $\boldsymbol{\theta}$ には制限が無く，数値的にも取り扱い易くなる．誤差分布の具体的な形が分からない場合，平均値と分散値をパラメータの関数として指定し，それから形式的に正規尤度を構成する疑似尤度法でパラメータを推定する疑似ファミリーモデルが使われる．

6.2.1 一般化線形モデルの当てはめ glm()

線形予測子と誤差分布のシンボリックな記述を指定して，一般化線形モデルを当てはめる．

書式：

```
glm(formula, family = gaussian, data, weights, subset,
    na.action, start = NULL, etastart, mustart,
    offset, control = glm.control(...), model = TRUE,
    method = "glm.fit", x = FALSE, y = TRUE, contrasts = NULL, ...)
glm.fit(x, y, weights = rep(1, nobs),
        start = NULL, etastart = NULL, mustart = NULL,
        offset = rep(0, nobs), family = gaussian(),
        control = glm.control(), intercept = TRUE)
# クラス "glm" に対する S3 メソッド
weights(object, type = c("prior", "working"), ...)
```

引数：

`formula` 当てはめられるモデルのシンボリックな記述．モデル指定の詳細は以下で与えられる

`family` モデルで使われる誤差分布とリンク関数の指定．これはファミリ関数や，ファ

[*2] $g(\mu) = \mu$ とすればただの線形モデルになる．

ミリ関数の呼び出しの結果を示す文字列で良い．(ファミリ関数に関する詳細は family を見よ)

data モデル中の変数を含むオプションのデータフレーム．もし data 中に見付からなければ，変数は典型的に glm() が呼び出された環境である environment(formula) で探される

weights 当てはめ過程で使われるオプションの重みベクトル

subset オプションベクトル．当てはめ過程で使われる観測値の部分集合を指定

na.action データが NA を含むときそれを処理する関数．既定は options 中の na.action 設定であり，もしこれが未設定なら na.fail() が使われる．「工場出荷時」の既定動作は na.omit()

start 線形予測子中のパラメータに対する初期値

etastart 線形予測子に対する初期値

mustart 平均ベクトルに対する初期値

offset これは当てはめの途中で，線形予測子に含めるべき成分を指定するのに使える

control 当てはめ過程を制御するパラメータリスト．glm.control() を参照

model モデルフレームを返り値の成分として含めるべきかどうかを指示する論理値

method モデル当てはめに使われるメソッド．既定のメソッド glm.fit() は重みを繰り返し調整する最小自乗法 (IWLS, iteratively reweighted least squares) を用いる．現在の唯一の他の選択肢は "model.frame" であり，モデルフレームを返すが当てはめはしない

x, y glm() に対しては，当てはめに使われた目的変数ベクトルとモデル行列を返り値の成分として返すかどうかを指示する論理値． glm.fit() に対しては x は次元 nxp の計画行列． y は長さ n の観測値ベクトル

contrasts オプションのリスト． model.matrix.default() を参照

object クラス "glm" を継承するオブジェクト

type 文字列で部分的マッチングが可能．当てはめられたモデルオブジェクトから取り出される重みのタイプ

intercept 論理値．切片項を含めるべきか？

... 他のメソッドへ (から) 引き渡される追加引き数

返り値： glm() はクラス "lm" を継承する "glm" を継承するオブジェクトを返す．この節の後を参照せよ．関数 summary() (つまり summary.glm()) は結果の要約を得たり，プリントするのに使うことができる．関数 anova() (つまり anova.glm()) は分散分析表を作るのに使われる．総称的アクセス関数 coefficients(), effects(), fitted.values() そして residuals() は glm() が返す値から様々な有用な情報を取り出すのに使うことができる．weights() は (部分集合を取り出し，na.action で処理した後の) 各当てはめ毎に，当てはめ重みベクトルを取り出す．クラス "glm" のオブジェクトは少なくとも以下の成分を含むリストである：

coefficients 係数の名前付きベクトル

residuals 作業残差，つまり IWLS 当てはめの最終反復時の残差

fitted.values 当てはめ平均値．リンク関数の逆関数で線形予測子を変換したもの

rank 当てはめ線形モデルの数値的ランク
family 使用された family オブジェクト
linear.predictors リンクスケールでの線形な当てはめ
deviance 定数を除いて，最大対数尤度の 2 倍のマイナス値．意味があれば，定数は飽和モデルが逸脱度 (deviance) 0 となるように選ばれる
aic 赤池の情報量規準．最大対数尤度の 2 倍のマイナス値に係数個数の 2 倍を加えた値 (逸脱度が分かっていると仮定している)
null.deviance null モデルに対する逸脱度で， deviance と比較可能． null モデルはオフセットと，もしモデルに存在すれば切片項を含む
iter 使われた IWSL 反復数
weights 作業重み，つまり IWLS 当てはめの最終反復時の重み
prior.weights 最初に与えられた重み
df.residual 残差の自由度
df.null null モデルに対する残差自由度
y 使用された y ベクトル (これは二項モデルに対してもベクトル)
converged 論理値．IWLS アルゴリズムは収束と判定されたか？
boundary 論理値．当てはめ値は到達可能な値の境界上にあるか？
call 呼出し式
formula 与えられたモデル式
terms 使われた terms オブジェクト
data data 引き数
offset 使われたオフセットベクトル
control 使われた control 引き数の値
method 使われた当てはめ関数の名前．R では常に "glm.fit"
contrasts (もし意味があれば) 使われた対比
xlevels (関係するときだけ) 当てはめで使われた因子の水準の記録

典型的な予測子は response ~ terms という形を持ち，ここで response は (数値) 目的変数ベクトルで terms は response に対する線形予測子を指定する項の系列である．binomial モデルに対しては，目的変数はまた factor (最初の水準が失敗，他の全てが成功を表すとき) や，その列が成功と失敗数を表す 2 行の行列として指定することができる．first + second 形式の項指定は，first 中の全ての項と second 中の全ての項から重複する項を除いたものを意味する．モデル式中の項は主効果が最初，2 次交互作用項がその次，更に 3 次交互作用が続く，等と並べ変えられる．こうすることにより term オブジェクトをモデル式として引き渡すことが防げる．形式 first:second の項指定は，first と second 中の全ての項同士の交互作用項を意味する．形式 first*second の項指定は，first と second 中の全ての項のクロスを意味する．これは指定 first + second + first:second と同値である．

glm.fit() と glm.fit.null() は実働関数である．前者は (切片項を持たない) null モデルに対して後者を呼び出す．もし etastart, start そして mustart の内から複数が指定されると，リスト中の最初のものが使われる．weights, subset, offset, etastart そして mustart 全ては formula 中の変数と同様に評価される．つまりまず data で，そ

れから formula の環境中という順序である．

更に，空でない当てはめは最終的な重みつき当てはめに関係する成分 qr, R そして effects を持つ．クラス "glm" のオブジェクトは普通クラス "lm" を継承するクラス c("glm", "lm") を持ち，クラス "lm" のオブジェクトに対して工夫されたメソッドが IWLS の最終反復過程の重みつき線形モデルに適用可能である．しかしながら，residuals や weights といったクラス "glm" 用の抽出関数は当てはめ結果の同名の成分を単に取り出すのでは無いので注意がいる．もし二項 glm モデルが二つの列からなる行列形式の目的変数で指定されると，prior.weights が返す重みは (与えられたケース重み倍された) ケース総数であり，結果の y 成分は成功比率である．

関連：glm に対するメソッド関数 anova.glm(),summary.glm() 等．総称的関数 anova(), summary(), effects(), fitted.values() そして residuals(). 更に，一般化されていない線形モデルである lm(). 食道ガンデータ esoph，不妊症データ infert そして predict.glm() は二項 glm モデルによる当てはめ例を持つ．

注意：逸脱度 (deviance) は，一般化線形モデルの当てはめの良さを計る基本的な基準である．データ y に対する二つのモデル M と M_0 があるとする．M_0 は M を含むモデルで，しばしば考えられる限り最も一般的なモデル (飽和モデル，full model) とされる．二つのモデルに対するデータの尤度をそれぞれ $p(y \mid \theta)$ と $p(y \mid \theta_0)$，対応する最尤推定量をそれぞれ $\hat{\theta}$ と $\hat{\theta_0}$ すると逸脱度 $D(y)$ は

$$D = -2\{\log p(y \mid \hat{\theta})\} - \log p(y \mid \hat{\theta_0})\}$$

と定義される．これは全仮説 M_0 の中で M を帰無仮説とする検定問題に対する尤度比検定統計量と考えられる．両モデル間の実効的パラメータの数の差を k とすると，モデル M が真であれば D の分布は漸近的に自由度 k のカイ 2 乗分布に従うことから，当てはめの善し悪しの判断ができる．一部の文献では「尤離度」という訳語が当てられているが，全てのモデルで真の尤度が計算されるわけではなく，「疑似尤度」が代わりに使われることもある．

```
> counts <- c(18,17,15,20,10,20,25,13,12)          # Dobson, 無作為化処理実験
> ( outcome <- gl(3,1,9) )                         # 結果を表す因子 (3 水準)
[1] 1 2 3 1 2 3 1 2 3
Levels: 1 2 3
> ( treatment <- gl(3,3) )                         # 処理を表す因子 (3 水準)
[1] 1 1 1 2 2 2 3 3 3
Levels: 1 2 3
> print(d.AD <- data.frame(treatment, outcome, counts)) # データフレームに変換
  treatment outcome counts
1         1       1     18
2         1       2     17
(途中省略)
9         3       3     12

# ポアッソン誤差分布，対数リンク関数で一般化線形モデル当てはめ
> glm.D93 <- glm(counts ~ outcome + treatment, family=poisson())
> anova(glm.D93)
Analysis of Deviance Table                         # 逸脱度分析表
Model: poisson, link: log                          # 誤差分布とリンク関数
Response: counts
Terms added sequentially (first to last)
          Df Deviance Resid. Df Resid. Dev
NULL                          8    10.5814
outcome    2   5.4523         6     5.1291
treatment  2   0.0000         4     5.1291
```

```
> summary(glm.D93)                                  # 当てはめ結果の要約
Call:                                               # 呼出し式
glm(formula = counts ~ outcome + treatment, family = poisson())
Deviance Residuals:                                 # 逸脱度残差
       1        2        3        4        5        6        7        8
-0.67125  0.96272 -0.16965 -0.21999 -0.95552  1.04939  0.84715 -0.09167
       9
-0.96656
Coefficients:                                       # 推定係数値，標準誤差，z 値，p 値
             Estimate Std. Error  z value Pr(>|z|)
(Intercept)  3.045e+00  1.709e-01   17.815   <2e-16 *** # 強く有意
outcome2    -4.543e-01  2.022e-01   -2.247   0.0246 *   # 5% 有意
outcome3    -2.930e-01  1.927e-01   -1.520   0.1285
treatment2   7.073e-16  2.000e-01 3.54e-15   1.0000
treatment3   5.110e-16  2.000e-01 2.55e-15   1.0000
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for poisson family taken to be 1)
    Null deviance: 10.5814  on 8  degrees of freedom
Residual deviance:  5.1291  on 4  degrees of freedom
AIC: 56.761                                         # 当てはめモデルの AIC 値
Number of Fisher Scoring iterations: 4              # 当てはめ過程の繰り返し回数
```

```
# Venables & Ripley からのオフセットを持つ例
> library(MASS)                                     # MASS ライブラリを読込み
> data(anorexia)                                    # anorexia(拒食症) データを読込み
# 治療後の体重を，治療前体重，治療法，オフセット化された治療前体重で説明
# 正規誤差分布を指定（治療前体重をオフセット（係数 1）で必ず含める）
> anorex.1 <- glm(Postwt ~ Prewt + Treat + offset(Prewt),
                  family = gaussian, data = anorexia)

> summary(anorex.1)
Call:
glm(formula = Postwt ~ Prewt + Treat + offset(Prewt), family = gaussian,
    data = anorexia)
Deviance Residuals:                                 # 逸脱度残差の要約
     Min        1Q    Median        3Q       Max
-14.1083   -4.2773   -0.5484    5.4838   15.2922
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  49.7711    13.3910   3.717 0.000410 *** # 切片項
Prewt        -0.5655     0.1612  -3.509 0.000803 *** # 治療前体重（からの差）
TreatCont    -4.0971     1.8935  -2.164 0.033999 *   # 対照群（無処置）
TreatFT       4.5631     2.1333   2.139 0.036035 *   # 治療（家族によるケア）
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for gaussian family taken to be 48.69504)
    Null deviance: 4525.4  on 71  degrees of freedom
Residual deviance: 3311.3  on 68  degrees of freedom
AIC: 489.97
Number of Fisher Scoring iterations: 2
```

```
# ガンマ分布誤差指定，McCullagh & Nelder の例
> clotting <- data.frame(u = c(5,10,15,20,30,40,60,80,100),
                         lot1 = c(118,58,42,35,27,25,21,19,18),
                         lot2 = c(69,35,26,21,18,16,13,12,12))

# 変数 u の対数値に回帰
> summary(glm(lot1 ~ log(u), data=clotting, family=Gamma))
Call:
glm(formula = lot1 ~ log(u), family = Gamma, data = clotting)
Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.04008  -0.03756  -0.02637   0.02905   0.08641
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.0165544  0.0009275  -17.85 4.28e-07 ***
log(u)       0.0153431  0.0004150   36.98 2.75e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for Gamma family taken to be 0.002446013)
    Null deviance: 3.512826  on 8  degrees of freedom
Residual deviance: 0.016730  on 7  degrees of freedom
```

```
AIC: 37.99
Number of Fisher Scoring iterations: 3
```

6.2.2 GLM モデルファミリの指定 family()

ファミリオブジェクトは glm() 関数等でモデルの詳細を指定する便利な手段を提供する．そのようなモデルを使った当てはめがどのように行われるかに付いては glm() のヘルプドキュメントを参照せよ．

書式：

```
family(object, ...)
binomial(link = "logit")
gaussian(link ="identity")
Gamma(link = "inverse")
inverse.gaussian(link = "1/mu^2")
poisson(link = "log")
quasi(link = "identity", variance = "constant")
quasibinomial(link = "logit")
quasipoisson(link = "log")
```

引数：

link モデルリンク関数の指定．これは name/expression，「文字通り」の文字列，長さ一の文字ベクトル，もしくはクラス "link-glm" のオブジェクトが可能である．gaussian ファミリはリンク "identity", "log" そして "inverse" を受け付ける．binomial ファミリは "logit", "probit", cauchyit (それぞれロジスティック，正規，コーシー分布の累積度数分布に対応)，そして "log", "cloglog"，poisson ファミリは "log", "identity" そして "sqrt"，inverse.gaussian ファミリは "1/mu^2", "inverse", "identity" そして "log"，quasi ファミリは "logit", "probit", "cloglog", "identity", "inverse", "log", "1/mu^2" そして "sqrt"，また関数 power() は quasi ファミリに対するパワーリンク関数を作るのに使える quasi ファミリは分散関数として "constant", "mu(1-mu)", "mu", "mu^2" そして "mu^3" という文字列そのものによる指定を受け付ける．もしくは成分 varfun, validmu, dev.resids, initialize そして name を持つリストによる指定が可能である

object 関数 family() は glm() 関数等により作られたオブジェクトに保管された family オブジェクトにアクセスする

... メソッドに引き渡される追加引数

quasibinomial と quasipoisson ファミリは binomial および poisson ファミリと，散布度 (dispersion) パラメータが 1 に固定されていないという点だけで異なる．従って「over-dispersion なモデル」を扱える．二項分布ケースに付いては McCullagh & Nelder を見よ．彼らは疑似二項モデル (quasi-binomial model) の様に (ある制限の下で) 分散が平均に比例するモデルがあることを示しているが，glm() はそうしたモデルに対し

て最尤推定量を計算しないことを注意しよう．S の挙動はこの疑似変種により近い．

注意：ロジット (logit)，プロビット (probit)，cauchyit，cloglog (complementary log-log) 関数はいずれも区間 $(0,1)$ を $(-\infty,\infty)$ に変換する単調増加関数で，次のように定義される：

$$\begin{aligned}
\mathrm{logit}(p) &= \log(p/(1-p)) \quad \text{(logit 関数．ロジスティック分布の分布関数の逆関数)}\\
\mathrm{probit}(p) &= \Phi^{-1}(p) \quad \text{(probit 関数．正規分布の分布関数の逆関数)}\\
\mathrm{cauchyit}(p) &= F^{-1}(p) \quad \text{(cauchyit 関数．コーシー分布の分布関数の逆関数)}\\
\mathrm{cloglog}(p) &= \log(-\log(1-p)) \quad \text{(cloglog 関数)}
\end{aligned}$$

関連：`glm()`，`power()`．`summary()` と関連メソッドに付いては `summary.lm()`．

`influence.measures()`，`dfbetas()`，`dffits()`，`covratio()`，`cooks.distance()`，`lm()`．ハット行列の対角成分は `hat()`．

```
> ( nf <- gaussian() )                                  # 正規ファミリオブジェクト作成
Family: gaussian                                        # 正規誤差でリンク関数は恒等式
Link function: identity

> str(nf)                                               # 詳細な内部情報
List of 10
 $ family    : chr "gaussian"
 $ link      : chr "identity"
 $ linkfun   :function (mu)
 $ linkinv   :function (eta)
 $ variance  :function (mu)
 $ dev.resids:function (y, mu, wt)
 $ aic       :function (y, n, mu, wt, dev)
 $ mu.eta    :function (eta)
 $ initialize:  expression({n <- rep.int(1,nobs); mustart <- y})
 $ validmu   :function (mu)
 - attr(*, "class")= chr "family"

> ( gf <- Gamma() )                                     # ガンマファミリオブジェクト作成
Family: Gamma                                         # 誤差分布はガンマ分布に従い，リンク関数は逆数
Link function: inverse

> str(gf)                                               # 詳細な内部情報
List of 11
 $ family    : chr "Gamma"
 $ link      : chr "inverse"
 $ linkfun   :function (mu)
 $ linkinv   :function (eta)
 $ variance  :function (mu)
 $ dev.resids:function (y, mu, wt)
 $ aic       :function (y, n, mu, wt, dev)
 $ mu.eta    :function (eta)
 $ initialize: expression(
   {if (any(y<=0))
    stop(paste("Non-positive values not","allowed for the gamma family"))
        n <- rep.int(1, nobs) mustart <- y })
 $ validmu   :function (mu)
 $ valideta  :function (eta)
 - attr(*, "class")= chr "family"

> gf$linkinv
function (eta)
1/eta
<environment: 0x9237d44>
> gf$variance(-3:4)                                     # gf$variance() は 2 乗関数
[1]  9  4  1  0  1  4  9 16
```

```
# 疑似ポアッソンファミリ．example(glm) と比較せよ
> counts <- c(18,17,15,20,10,20,25,13,12)
```

```
> outcome <- gl(3,1,9)
> treatment <- gl(3,3)
> d.AD <- data.frame(treatment, outcome, counts)  # データフレームにまとめる

# リンク関数は既定値の log 関数が使われる
> ( glm.qD93 <- glm(counts ~ outcome + treatment, family=quasipoisson()) )
Call:  glm(formula = counts ~ outcome + treatment, family = quasipoisson())
Coefficients:                                     # 係数推定値
(Intercept)     outcome2     outcome3   treatment2   treatment3
  3.045e+00   -4.543e-01   -2.930e-01    7.073e-16    5.110e-16
Degrees of Freedom: 8 Total (i.e. Null);  4 Residual
Null Deviance:      10.58
Residual Deviance: 5.129        AIC: NA           # 疑似モデルなので AIC は無意味

> str(glm.qD93$family)                            # 使用されたファミリの情報
List of 11
 $ family    : chr "quasipoisson"
 $ link      : chr "log"
 $ linkfun   :function (mu)
 $ linkinv   :function (eta)
 $ variance  :function (mu)
 (途中省略)
 - attr(*, "class")= chr "family"

> glm.qD93$family$linkfun(1:3)                    # リンク関数は対数
[1] 0.0000000 0.6931472 1.0986123
> glm.qD93$family$linkinv(1:3)                    # 逆リンク関数は指数
[1]  2.718282  7.389056 20.085537
> glm.qD93$family$variance(1:3)                   # 分散は平均と一致
[1] 1 2 3

# 当てはめ結果から逸脱度分析表を作成．逸脱度は疑似尤度から形式的に計算
> anova(glm.qD93, test="F")
Analysis of Deviance Table
Model: quasipoisson, link: log
Response: counts
Terms added sequentially (first to last)
          Df Deviance Resid. Df Resid. Dev      F Pr(>F)
NULL                         8    10.5814
outcome    2   5.4523         6     5.1291 2.1079 0.2370
treatment  2   0.0000         4     5.1291 0.0000 1.0000

> summary(glm.qD93)                               # 当てはめ結果の要約
Call:
glm(formula = counts ~ outcome + treatment, family = quasipoisson())
Deviance Residuals:
       1        2        3        4        5        6        7        8        9
-0.67125  0.96272 -0.16965 -0.21999 -0.95552  1.04939  0.84715 -0.09167 -0.96656
Coefficients:
             Estimate Std. Error  t value Pr(>|t|)
(Intercept)  3.045e+00  1.944e-01   15.665  9.7e-05 ***
outcome2    -4.543e-01  2.299e-01   -1.976    0.119
outcome3    -2.930e-01  2.192e-01   -1.337    0.252
treatment2   7.073e-16  2.274e-01 3.11e-15    1.000
treatment3   5.110e-16  2.274e-01 2.25e-15    1.000
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for quasipoisson family taken to be 1.293300)
    Null deviance: 10.5814  on 8  degrees of freedom
Residual deviance:  5.1291  on 4  degrees of freedom
AIC: NA
Number of Fisher Scoring iterations: 4
```

```
# ポアッソンファミリの場合の逸脱度分析表は次のようにする
> anova(glm.qD93, dispersion = 1, test="Chisq")
Analysis of Deviance Table
Model: quasipoisson, link: log
Response: counts
Terms added sequentially (first to last)
          Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                         8    10.5814
outcome    2   5.4523         6     5.1291    0.0655
treatment  2   0.0000         4     5.1291    1.0000
```

```
> summary(glm.qD93, dispersion = 1)               # 当てはめ結果の要約
Call:
glm(formula = counts ~ outcome + treatment, family = quasipoisson())
Deviance Residuals:
       1         2         3         4         5         6         7
-0.67125   0.96272  -0.16965  -0.21999  -0.95552   1.04939   0.84715
       8         9
-0.09167  -0.96656
Coefficients:
             Estimate Std. Error  z value Pr(>|z|)
(Intercept)  3.045e+00  1.709e-01   17.815   <2e-16 ***
outcome2    -4.543e-01  2.022e-01   -2.247   0.0246 *
outcome3    -2.930e-01  1.927e-01   -1.520   0.1285
treatment2   7.073e-16  2.000e-01 3.54e-15   1.0000
treatment3   5.110e-16  2.000e-01 2.55e-15   1.0000
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for quasipoisson family taken to be 1)
    Null deviance: 10.5814  on 8  degrees of freedom
Residual deviance:  5.1291  on 4  degrees of freedom
AIC: NA
Number of Fisher Scoring iterations: 4
```

```
> x <- rnorm(100)                                 # quasi ファミリのテスト
> y <- rpois(100, exp(1+x))
> glm(y ~x, family=quasi(var="mu", link="log"))   # quasi ファミリによる回帰
Call:  glm(formula = y ~ x, family = quasi(var = "mu", link = "log"))
Coefficients:
(Intercept)            x
     1.0762       0.9166
Degrees of Freedom: 99 Total (i.e. Null);  98 Residual
Null Deviance:      476
Residual Deviance: 99.3         AIC: NA

> glm(y ~x, family=poisson)                       # 上は以下と同じ（しかし AIC 値が求まる）
Call:  glm(formula = y ~ x, family = poisson)
Coefficients:
(Intercept)            x
     1.0762       0.9166
Degrees of Freedom: 99 Total (i.e. Null);  98 Residual
Null Deviance:      476
Residual Deviance: 99.3         AIC: 372.7

> glm(y ~x, family=quasi(var="mu^2", link="log")) # ファミリ指定変更
Call:  glm(formula = y ~ x, family = quasi(var = "mu^2", link = "log"))
Coefficients:
(Intercept)            x
      1.062        0.942
Degrees of Freedom: 99 Total (i.e. Null);  98 Residual
Null Deviance:      85.04
Residual Deviance: 24.98        AIC: NA
```

6.2.3 GLM ファミリに対するリンクを作る make.link()

この関数は glm() 中の family() 関数とともに用いられる．リンクを与えると，リンク関数，逆リンク関数，導関数 $d\mu/d\eta$，そして領域チェックのための関数が返される．

書式： make.link(link)

引数： link 文字もしくは数値．"logit", "probit", "cloglog", "identity", "log", "sqrt", "1/mu^2", "inverse" のどれか．もしくは数値 lambda で巾乗リンク mu^lambda に対応

返り値： 次の成分を持つリスト：

linkfun リンク関数 function(mu)

linkinv　逆リンク関数 function(eta)

mu.eta　mu の eta による導関数

valideta　全ての eta が領域 linkinv にあれば TRUE となる関数

関連：glm(), family().

```
>  str(make.link("logit"))
List of 4
 $ linkfun :function (mu)
 $ linkinv :function (eta)
 $ mu.eta  :function (eta)
 $ valideta:function (eta)
>  l2 <- make.link(2)
>  l2$linkfun(0:3)                                  # リンク関数の値
[1] 0 1 4 9
>  l2$mu.eta(eta= 1:2)                              # 1/(2*sqrt(eta)) に等しい
[1] 0.5000000 0.3535534
```

6.2.4 巾乗リンクオブジェクトを作る power()

リンク関数 eta $= \mu^{\lambda}$ に基づくリンクオブジェクトを作る.

書式： power(lambda = 1)
引数： lambda 実数
返り値： 返り値は成分 linkfun, linkinv, mu.eta そして valideta を持つリスト. これらの意味は make.link() 関数を参照せよ

もし lambda が負なら 0 とされ，対数リンクが得られる．既定の lambda = 1 は恒等リンク関数を与える.

関連：make.link(), family(). 巾乗に付いては Arithmetic を見よ．検定の検出力に付いては，stats パッケージ中の様々な関数，例えば power.t.test() を見よ.

```
> power(1)        # 恒等リンク関数 eta=mu
$linkfun
function (mu)
mu
<environment: 0x9131a18>
$linkinv
function (eta)
eta
<environment: 0x9131a18>
$mu.eta
function (eta)
rep.int(1, length(eta))
<environment: 0x9131a18>
$valideta
function (eta)
TRUE
<environment: 0x9131a18>
```

```
# 1/3 乗リンク関数 eta=mu^(1/3)
> quasi(link=power(1/3))[c("linkfun", "linkinv")]
$linkfun
function (mu)
mu^lambda
<environment: 0x9150dd4>
$linkinv
function (eta)
pmax(.Machine$double.eps, eta^(1/lambda))
<environment: 0x9150dd4>
```

6.2.5 一般化線形モデルによる予測 predict.glm()

一般化線形モデル当てはめオブジェクトから予測値を得たり，オプションとしてこれらの予測値の標準偏差を推定する.

書式：

```
# クラス "glm" に対する S3 メソッド
predict(object, newdata=NULL, type=c("link","response","terms"),
        se.fit=FALSE, dispersion=NULL, terms=NULL,
        na.action=na.pass, ...)
```

引数：

object クラス "glm" を継承するクラスの当てはめオブジェクト

newdata そこで予測値を計算するオプションの新しいデータフレーム．もし省略されると，当てはめ線形予測値が使われる

type 必要な予測のタイプ．既定は線形予測量のスケールであり，もう一つの選択肢 "response" は目的変数のスケールである．例えば，既定の二項モデルでは既定の予測量は対数オッズ (ロジットスケールでの確率) であり，type="response" は予測された確率を与える．オプション "terms" はモデル式中の各項の線形予測量スケールでの予測値を与える行列を返す．引数の値は短縮化できる

se.fit 標準誤差が必要かどうか指定する論理値

dispersion 標準誤差を計算する際に仮定される GLM 当てはめの散布度パラメータ

terms 指定 type="terms" では既定で全ての項が返される．どの項を返すかを指示する文字列ベクトル

na.action newdata が NA を含むときの処理関数．既定では予測値 NA を返す

... 他のメソッドへ (から) 引き渡される追加引き数

返り値： もし se = FALSE なら予測値のベクトルもしくは行列で，もし se = TRUE なら次の成分を持つリスト：

fit 予測値

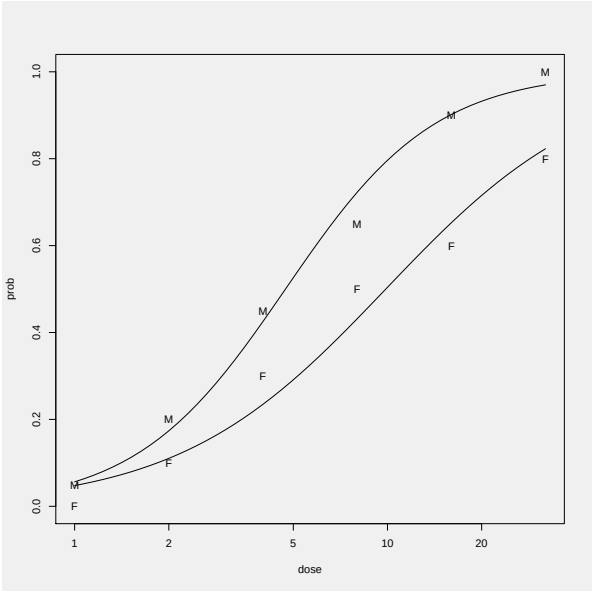
se.fit 推定された標準誤差

residual.scale 標準誤差の計算中に使われた逸脱度の平方根を与えるスカラ

関連：glm(), SafePrediction()

```
> ldose <- rep(0:5, 2)                              # Venables & Ripley の例
> numdead <- c(1, 4, 9, 13, 18, 20, 0, 2, 6, 10, 12, 16)
> sex <- factor(rep(c("M", "F"), c(6, 6)))
# 二項分布ファミリに対するデータ形式 (成功数と失敗数の対からなる 2 行の行列)
> SF <- cbind(numdead, numalive=20-numdead)

# 二項分布ファミリによる一般化線形モデル当てはめ (次の図を参照)
> budworm.lg <- glm(SF ~ sex*ldose, family=binomial)

> summary(budworm.lg)                               # その要約
Call:
glm(formula = SF ~ sex * ldose, family = binomial)
Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.39849  -0.32094  -0.07592   0.38220   1.10375
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -2.9935     0.5527  -5.416 6.09e-08 ***
sexM          0.1750     0.7783   0.225    0.822
ldose         0.9060     0.1671   5.422 5.89e-08 ***
sexM:ldose    0.3529     0.2700   1.307    0.191
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for binomial family taken to be 1)
    Null deviance: 124.8756  on 11  degrees of freedom
Residual deviance:   4.9937  on  8  degrees of freedom
AIC: 43.104
Number of Fisher Scoring iterations: 4

> plot(c(1,32), c(0,1), type = "n", xlab = "dose",
       ylab = "prob", log = "x")                    # 結果のプロット．まず空のグラフを描く
```

```
> text(2^ldose, numdead/20, as.character(sex))
> ld <- seq(0, 5, 0.1)

# 男性に対する予測確率値のグラフを上描き
> lines(2^ld, predict(budworm.lg, data.frame(ldose=ld,
        sex=factor(rep("M", length(ld)), levels=levels(sex))),
        type = "response"))
# 女性に対する予測確率値のグラフを上描き
> lines(2^ld, predict(budworm.lg, data.frame(ldose=ld,
        sex=factor(rep("F", length(ld)), levels=levels(sex))),
        type = "response"))
```



二項分布ファミリによる一般化線形モデル当てはめ結果による予測確率値

6.2.6 一般化線形モデル当てはめ結果の要約 summary.glm()

これらの関数はクラス "glm" もしくは "summary.glm" のオブジェクトのメソッド関数である.

書式:

```
# クラス "glm" に対する S3 メソッド
summary(object, dispersion = NULL, correlation = FALSE,
        symbolic.cor = FALSE, ...)
# クラス "summary.glm" に対する S3 メソッド
print(x, digits = max(3, getOption("digits") - 3),
      symbolic.cor = x$symbolic.cor,
      signif.stars = getOption("show.signif.stars"), ...)
```

引数:

object クラス "glm" のオブジェクトで, 典型的には glm() 呼び出しの結果

x クラス "summary.glm" のオブジェクト. 普通 summary.glm() 呼び出し結果

dispersion 当てはめファミリへの散らばりパラメータ. 既定では object から得られる

correlation 論理値. もし TRUE なら推定パラメータの相関行列が返される

digits 出力の際に使われる有効桁数
symbolic.cor 論理値．もし TRUE なら相関を数ではなく，シンボリック (symnum() を見よ) な形式で出力する
signif.stars 論理値．もし TRUE なら各係数への「有意水準を表す星印」を出力
... 他のメソッドへ (から) 引き渡される追加引き数

返り値： summary.glm() はクラス "summary.glm" のオブジェクトを返す，これは次の成分を持つリストである：
call object からの成分
family object からの成分
deviance object からの成分
contrasts object からの成分
df.residual object からの成分
null.deviance object からの成分
df.null object からの成分
deviance.resid 逸脱度残差． residuals.glm() を参照
coefficients 係数行列，標準誤差，z 値，そして p 値．エイリアス化された係数は取り除かれる
aliased 名前付きの論理値で，オリジナルの係数がエイリアス化されているかどうかを示す
dispersion 与えられた引数か，もしそれが NULL なら推定散らばりパラメータ
df モデルのランク，残差自由度，そしてエイリアス化されていない係数の数からなる長さ 3 のベクトル
cov.unscaled 推定係数のスケール化されていない (つまり dispersion=1) 推定共分散
cov.scaled 上と同じだが dispersion でスケール化されている
correlation (correlation=TRUE の時だけ) 推定係数の推定相関
symbolic.cor (correlation=TRUE の時だけ) 引数 symbolic.cor の値

print.summary.glm() は係数，標準誤差等を読みやすい形に整形しようと試み，もし signif.stars = TRUE なら追加で有意水準を表す星印を与える．エイリアス化された係数は返り値のオブジェクトから除外されるが，R1.8.0 から print() メソッドでは保存される．相関は 2 桁 (もしくはシンボリックに) 出力される．実際の出力 summary(object)$correlation を見よ．

関連： glm(), summary().

```
> summary(glm.D93)                                  # example(glm) に続く
Call:
glm(formula = counts ~ outcome + treatment, family = poisson())

Deviance Residuals:
      1       2       3       4       5       6       7       8       9
-0.6712  0.9627 -0.1696 -0.2199 -0.9555  1.0493  0.8471 -0.0916 -0.9665
Coefficients:
             Estimate Std. Error  z value Pr(>|z|)
(Intercept)  3.045e+00  1.709e-01   17.815   <2e-16 ***
```

```
outcome2    -4.543e-01  2.022e-01   -2.247   0.0246 *
outcome3    -2.930e-01  1.927e-01   -1.520   0.1285
treatment2   7.073e-16  2.000e-01 3.54e-15   1.0000
treatment3   5.110e-16  2.000e-01 2.55e-15   1.0000
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)
    Null deviance: 10.5814  on 8  degrees of freedom
Residual deviance:  5.1291  on 4  degrees of freedom
AIC: 56.761

Number of Fisher Scoring iterations: 4
```

6.2.7 GLM 当てはめを制御する glm.control()

glm.control() は glm() による当てはめに対するユーザインタフェイスを与える補助関数である．典型的には，glm() または glm.fit() を呼び出すときにだけ使われる．

書式： glm.control(epsilon=1e-8, maxit=25, trace=FALSE)

引数：

epsilon　収束許容度を与える正の小さな数．繰りかえしは次の条件が成立すれば収束したとされる，(dev-devold)/(dev+0.1) < epsilon

maxit　IWLS 繰りかえしの最大回数を与える整数

trace　繰りかえし毎に出力するかどうかを指示する論理値

返り値： 引数を成分に持つリスト

もし epsilon が小さいと，最小自乗解に対する許容度としても使われる．もし trace が真なら，cat() を用いて各 IWSL 繰りかえしに対する情報が出力される．したがって options(digits = *) を用いて表示精度を上げることができる．

関連： glm() による当てはめで使われる glm.fit().

```
> counts <- c(18,17,15,20,10,20,25,13,12)         # example(glm) の変形. Dobson の例
> outcome <- gl(3,1,9)
> treatment <- gl(3,3)
> oo <- options(digits = 12)                      # 途中結果の出力桁数を上げる

> glm.D93X <- glm(counts ~ outcome + treatment, family=poisson(),
                  trace = TRUE, epsilon = 1e-14)
Deviance = 5.17971906292 Iterations - 1
Deviance = 5.12914710976 Iterations - 2
Deviance = 5.129141077 Iterations - 3
Deviance = 5.129141077 Iterations - 4
Deviance = 5.129141077 Iterations - 5
> options(oo)                                     # オプションを元に戻す

> coef(glm.D93X)                                  # 最後の二つは?glm の glm.D93 より 0 に近い
  (Intercept)      outcome2      outcome3    treatment2    treatment3
 3.044522e+00 -4.542553e-01 -2.929871e-01 -2.031703e-16  5.093582e-17
```

6.3 分散分析表

6.3.1 分散分析モデルの当てはめ aov()

lm() を各層に適用し，分散分析モデルを当てはめる．

書式：
```
aov(formula, data = NULL, projections = FALSE, qr = TRUE,
    contrasts = NULL, ...)
```

引数：

formula　モデルを指定する式

data　モデル式中 w で指定された変数が含まれているデータフレーム．もし与えられなければ，変数は標準的なやり方で探される

projections　論理フラグ．射影を返すべきか？

qr　論理フラグ．QR 分解を返すべきか？

contrasts　モデル式中の因子の幾つかに使われる対比のリスト．これらはどの Error 項にも適用されない．もし Error 項中の因子だけに対比を与えると，警告がでる

... lm() に引き渡される引数，例えば subset や na.action

返り値：　返り値はクラス c("aov", "lm")，または多重応答 (multiple response) に対してはクラス c("maov", "aov", "mlm", "lm")，多重誤差層 (multiple error strata) に対してはクラス "aovlist" のオブジェクト．これらに対して利用できる print そして summary メソッドがある

この関数は釣合型や非釣合型の実験計画法データに lm() を用いて線形モデルを当てはめるためのラッパ関数を提供する．lm() との主な違いは当てはめ結果の print(), summary() 等の処理法であり，線形モデルではなく古典的な分散分析の言葉で表現される．もしモデル式が単一の Error 項を含めばこれが誤差層を指定するのに使われ，各誤差層内で適当なモデルが当てはめられる．モデル式は多重応答を指定できる．重みは weights 引数で与えることができるが，Error 項内で使うべきではなく，サポートは不完全である (例えば model.tables() はサポートしない)．

注意：aov() は釣合型実験計画法用にデザインされており，非釣合型実験計画に対しては結果の解釈は困難である．応答中の欠損値は釣合型でなくなる可能性が高いことを注意しよう．もし 2 つ以上の誤差層があれば，使われている手法は釣合型でなければ統計的に非効率的であり，lme() を使う方が好ましいかも知れない．データが釣合型かどうかは replications() 関数を用いて検査できる．

関連：lm(), summary.aov(), replications(), alias(), proj(), model.tables(), TukeyHSD().

```
# Venables & Ripley の例，古典的な圃場実験．窒素 N，リン酸 P，カリ K を与えた農業実験データ
> N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
> P <- c(1,1,0,0,0,1,0,1,1,1,0,0,0,1,0,1,1,0,0,1,0,1,1,0)
> K <- c(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
> yield <- c(49.5,62.8,46.8,57.0,59.8,58.5,55.5,56.0,62.8,55.8,69.5,55.0,
             62.0,48.8,45.5,44.2,52.0,51.5,49.8,48.8,57.2,59.0,53.2,56.0) # 対応する収量

# 変数をデータフレームに統合．各肥料を因子化，ブロック因子 (6 水準各 4 回の繰りかえし)
> npk <- data.frame(block=gl(6,4), N=factor(N), P=factor(P),
                    K=factor(K), yield=yield)
> op <- options(contrasts=c("contr.helmert", "contr.treatment")) # 対比の指定

# 分散分析実行．2 次交互作用，ブロック因子を考慮
> ( npk.aov <- aov(yield ~ block + N*P*K, npk) )
```

```
Call:
   aov(formula = yield ~ block + N * P * K, data = npk)
Terms:
                    block        N      P       K     N:P     N:K    P:K
Sum of Squares  343.2950 189.2817 8.4017 95.2017 21.2817 33.1350 0.4817
Deg. of Freedom        5        1      1       1       1       1      1
                Residuals
Sum of Squares   185.2867
Deg. of Freedom        12
Residual standard error: 3.929447
1 out of 13 effects not estimable
Estimated effects are balanced                    # データは釣合型と報告

> summary(npk.aov)                                # 結果の要約（分散分析表）
            Df Sum Sq Mean Sq F value   Pr(>F)
block        5 343.29   68.66  4.4467 0.015939 *  # ブロック効果 5% 有意
N            1 189.28  189.28 12.2587 0.004372 ** # 窒素肥料 0.5% 有意
P            1   8.40    8.40  0.5441 0.474904
K            1  95.20   95.20  6.1657 0.028795 *  # カリ肥料 5% 有意
N:P          1  21.28   21.28  1.3783 0.263165    # 交互作用は全て有意でない
N:K          1  33.14   33.14  2.1460 0.168648
P:K          1   0.48    0.48  0.0312 0.862752
Residuals   12 185.29   15.44
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> coefficients(npk.aov)                           # 各効果（係数）推定値
(Intercept)      block1      block2      block3      block4      block5
 54.8750000   1.7125000   1.6791667  -1.8229167  -1.0137500   0.2950000
         N1          P1          K1       N1:P1       N1:K1       P1:K1
  2.8083333  -0.5916667  -1.9916667  -0.9416667  -1.1750000   0.1416667
```

```
> aov(yield ~ block + N * P + K, npk)         # 窒素・リン間の 2 次交互作用だけを考慮し再実行
Call:
   aov(formula = yield ~ block + N * P + K, data = npk)
Terms:
                    block        N        P        K      N:P Residuals
Sum of Squares  343.2950 189.2817   8.4017  95.2017  21.2817  218.9033
Deg. of Freedom        5        1        1        1        1        14
Residual standard error: 3.954232
Estimated effects are balanced

> aov(terms(yield ~ block + N * P + K, keep.order=TRUE), npk)
Call:
   aov(formula = terms(yield ~ block + N * P + K, keep.order = TRUE),
       data = npk)
Terms:
                    block        N        P      N:P        K Residuals
Sum of Squares  343.2950 189.2817   8.4017  21.2817  95.2017  218.9033
Deg. of Freedom        5        1        1        1        1        14
Residual standard error: 3.954232
Estimated effects are balanced

> summary(aov(terms(yield ~ block + N * P + K, keep.order=TRUE), npk))
            Df Sum Sq Mean Sq F value   Pr(>F)
block        5 343.29   68.66  4.3911 0.012954 *
N            1 189.28  189.28 12.1055 0.003684 **
P            1   8.40    8.40  0.5373 0.475637
N:P          1  21.28   21.28  1.3611 0.262841
K            1  95.20   95.20  6.0886 0.027114 *
Residuals   14 218.90   15.64
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
# ブロック効果を誤差層とする（テストであり統計的には無意味）
> ( npk.aovE <- aov(yield ~  N*P*K + Error(block), npk) )
Call:
aov(formula = yield ~ N * P * K + Error(block), data = npk)
Grand Mean: 54.875
Stratum 1: block
Terms:
                   N:P:K Residuals
Sum of Squares   37.00167 306.29333
Deg. of Freedom        1         4
Residual standard error: 8.750619
```

```
Estimated effects are balanced
Stratum 2: Within
Terms:
                         N       P        K      N:P      N:K     P:K
Sum of Squares  189.28167 8.40167 95.20167 21.28167 33.13500 0.48167
Deg. of Freedom         1       1        1        1        1       1
                Residuals
Sum of Squares  185.28667
Deg. of Freedom        12
Residual standard error: 3.929447
Estimated effects are balanced

> summary(npk.aovE)
Error: block
          Df  Sum Sq Mean Sq F value Pr(>F)
N:P:K      1  37.002  37.002  0.4832 0.5252
Residuals  4 306.293  76.573
Error: Within
          Df  Sum Sq Mean Sq F value   Pr(>F)
N          1 189.282 189.282 12.2587 0.004372 **
P          1   8.402   8.402  0.5441 0.474904
K          1  95.202  95.202  6.1657 0.028795 *
N:P        1  21.282  21.282  1.3783 0.263165
N:K        1  33.135  33.135  2.1460 0.168648
P:K        1   0.482   0.482  0.0312 0.862752
Residuals 12 185.287  15.441
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> options(op)                                    # オプションを元に戻す
```

6.3.2 分散分析表を作る anova()

一つまたは複数の当てはめオブジェクトから分散 (または逸脱度) 分析表を計算する.

書式： `anova(object, ...)`

引数：

`object` モデル当てはめ関数 (例えば `lm()` や `glm()`) が返す結果を含むオブジェクト

`...` 同じタイプの追加オブジェクト

返り値： この総称的関数はクラス `"anova"` のオブジェクトを返す．こうしたオブジェクトは分散分析または逸脱度分析表を表す．もし単一の引き数が与えられるとモデル項が有意かどうか検定する表を生成する．もし複数のオブジェクトを与えると，`anova()` はモデルを他のモデルと指定された順序で検定する．`"anova"` クラスオブジェクトに対する print メソッドは表を見やすい形で出力する

注意： 二つもしくはそれ以上のモデルの比較はそれらが同じデータセットに当てはめられたときだけ意味がある．これは，もし欠損値があり，R の既定動作である `na.action = na.omit` が使われたときは問題となる可能性がある．

関連： `coefficients()`, `effects()`, `fitted.values()`, `residuals()`, `summary()`, `drop1()`, `add1()`.

6.3.3 分散分析モデルの要約 summary.aov()

分散分析モデルの要約を与える．

書式：

```
# クラス "aov" に対する S3 メソッド
summary(object, intercept = FALSE, split,
        expand.split = TRUE, keep.zero.df = TRUE, ...)
# クラス "aovlist" に対する S3 メソッド
summary(object, ...)
```

引数：

object　クラス "aov" もしくは "aovlist" のオブジェクト

intercept　論理値．切片項を含めるべきか？

split　オプションの名前付きリストで，名前はモデル中の項に対応する．各成分はそれ自体リストで，その寄与が総和されるべき対比を与える整数成分からなる

expand.split　論理値． split はその因子を含む交互作用にも適用すべきか？

keep.zero.df　論理値．自由度 0 の項も含めるべきか？

...　summary.aov() に対するものも含む summary.aovlist() に対する他のメソッドへ (から) 引き渡される引数

返り値：それぞれクラス c("summary.aov", "listof") もしくは "summary.aovlist" のオブジェクト

注意：expand.split = TRUE はほとんどテストされていない．これを FALSE に設定し，必要な全ての分割を指定することはいつでも可能である．

関連：aov(), summary(), model.tables(), TukeyHSD().

```
# Venables & Ripley より．窒素 N，リン酸 P，カリ K を与えた農業実験データ
> N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
> P <- c(1,1,0,0,0,1,0,1,1,1,0,0,0,1,0,1,1,0,0,1,0,1,1,0)
> K <- c(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,0,1,1,1,0,1,0)
> yield <- c(49.5,62.8,46.8,57.0,59.8,58.5,55.5,56.0,62.8,55.8,69.5,55.0, # 対応する収量
             62.0,48.8,45.5,44.2,52.0,51.5,49.8,48.8,57.2,59.0,53.2,56.0)
> npk <- data.frame(block=gl(6,4), N=factor(N), P=factor(P),              # 因子の指定
                    K=factor(K), yield=yield)

> ( npk.aov <- aov(yield ~ block + N*P*K, npk) )     # 交互作用を含む分散分析の結果
Call:
   aov(formula = yield ~ block + N * P * K, data = npk)
Terms:
                   block       N     P      K    N:P    N:K   P:K
Sum of Squares   343.295 189.281 8.402 95.202 21.282 33.135 0.482
Deg. of Freedom        5       1     1      1      1      1     1
                Residuals
Sum of Squares    185.2867
Deg. of Freedom         12
Residual standard error: 3.929447
1 out of 13 effects not estimable
Estimated effects may be unbalanced

> summary(npk.aov)                                  # 分散分析結果の要約
            Df Sum Sq Mean Sq F value   Pr(>F)
block        5 343.29   68.66  4.4467 0.015939 *
N            1 189.28  189.28 12.2587 0.004372 **
```

```
P            1   8.40    8.40  0.5441 0.474904
K            1  95.20   95.20  6.1657 0.028795 *
N:P          1  21.28   21.28  1.3783 0.263165
N:K          1  33.14   33.14  2.1460 0.168648
P:K          1   0.48    0.48  0.0312 0.862752
Residuals   12 185.29   15.44
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> coefficients(npk.aov)                          # 推定係数の取り出し
(Intercept)  block2  block3    block4   block5  block6
 51.82500  3.42500 6.75000  -3.90000 -3.50000 2.32500
       N1       P1       K1    N1:P1    N1:K1   P1:K1
  9.85000  0.41667 -1.91667 -3.76667  -4.7000 0.56667
```

```
# Cochran & Cox. 順序因子を持つ 3x3 要因実験，各々は 12 データの平均値
> CC <- data.frame(
      y = c(449, 413, 326, 409, 358, 291, 341, 278, 312)/12,
      P = ordered(gl(3, 3)), N = ordered(gl(3, 1, 9))
  )

> CC.aov <- aov(y ~ N * P, data = CC , weights = rep(12, 9))

> summary(CC.aov)
            Df  Sum Sq Mean Sq
N            2 1016.67  508.33
P            2  917.39  458.69
N:P          4  399.28   99.82

> summary(CC.aov, split=list(N = list(L=1,Q=2), P=list(L=1,Q=2)))
            Df  Sum Sq Mean Sq                    # 主効果を線形と 2 次部分に分割
N            2 1016.67  508.33
  N: L       1 1012.50 1012.50
  N: Q       1    4.17    4.17
P            2  917.39  458.69
  P: L       1  917.35  917.35
  P: Q       1    0.04    0.04
N:P          4  399.28   99.82
  N:P: L.L   1  184.08  184.08
  N:P: Q.L   1  152.11  152.11
  N:P: L.Q   1   49.00   49.00
  N:P: Q.Q   1   14.08   14.08

> summary(CC.aov, split = list("N:P" = list(L.L = 1, Q = 2:4)))
            Df  Sum Sq Mean Sq                    # 交互作用だけを分割
N            2 1016.67  508.33
P            2  917.39  458.69
N:P          4  399.28   99.82
  N:P: L.L   1  184.08  184.08
  N:P: Q     3  215.19   71.73

> summary(CC.aov, split = list(P = list(lin = 1, quad = 2)))
            Df  Sum Sq Mean Sq                    # 一つの変数についてだけ分割
N            2 1016.67  508.33
P            2  917.39  458.69
  P: lin     1  917.35  917.35
  P: quad    1    0.04    0.04
N:P          4  399.28   99.82
  N:P: lin   2  336.19  168.10
  N:P: quad  2   63.08   31.54

> summary(CC.aov, split = list(P = list(lin = 1, quad = 2)),
          expand.split=FALSE)                     # 交互作用は分割しない
            Df  Sum Sq Mean Sq
N            2 1016.67  508.33
P            2  917.39  458.69
  P: lin     1  917.35  917.35
  P: quad    1    0.04    0.04
N:P          4  399.28   99.82
```

6.3.4 線形モデル当てはめ結果に対する分散分析 anova.lm()

一つ，もしくは複数の線形モデル当てはめ結果に対する分散分析表を計算する．

書式：
クラス "lm" に対する S3 メソッド
`anova(object, ...)`
`anova.lmlist(object, ..., scale = 0, test = "F")`

引数：
`test` 使用される検定統計量を指定する文字列． `"F"`, `"Chisq"` または `"Cp"` のどれかで，部分的一致で良い．もしくは検定を用いないなら `NULL`
`scale` ノイズ分散の推定値．もし 0 なら考慮される最大モデルから推定される

返り値： クラス `"data.frame"` を継承するクラス `"anova"` のオブジェクト

単一のオブジェクトを指定すると，その当てはめに対する逐次的な分散分析を与える．つまり，各項が順に加えられた際の残差平方和に於ける減少と，残差平方和が表の行として与えられる．表は行に対する平均平方和と残差平方和を比較する F 統計量 (と P 値) を含む．もし複数のオブジェクトを指定すると，表は各モデルに対する残差自由度と残差平方和に対する行を持つ．最初のモデルを除く全てのモデルに対し，自由度と残差平方和の変化がまた与えられる．(これはモデルが入れ子になっているときだけ統計的に意味がある．) モデルを最小モデルから最大モデルへと並べるのが慣習であるが，しかしユーザの好みによる．

オプションとして表は検定統計量を含むことができる．通常 F 統計量が最も適当であり，これはある行の平均平方和と，最大モデルの残差平方和を比較する．もし `scale` が指定されると，カイ 2 乗統計量を使うことができる．Mallows の Cp 統計量は，残差平方和と残差自由度の $2\sigma^2$ 倍を加えたものである．

警告： 二つ以上のモデルの比較は，それらが同じデータセットに当てはめられたときだけ意味がある．このことは，もし欠損値があり R の既定動作である `na.action = na.omit` が使われた時は問題無しとはしない．`anova.lmlist()` はこれをエラーとして検出する．
注意： バージョン 1.2.0 以前の R は対毎の比較による F 検定を利用していた．この行動は `anovalist.lm()` を直接呼び出せば今でも得られる．
関連： モデル当てはめ関数の `lm()`, `anova()`．各項がそれらの階層性を保ちながら一時に一つずつ取り去られるいわゆる `type II` 分散分析表については `drop1()`．

```
# 家計貯蓄率データ LifeCycleSavings を使用した例
# 変数 pop15,ddpi の追加は 5% 有意 (0 と看做せない平均の変化がある)
# 変数 pop75,spi の追加は 5% 有意でない (平均の変化があるとはいえない)
> fit <- lm(sr ~ ., data = LifeCycleSavings)

> anova(fit)                    # 逐次的 (変数を一つずつ増やしていく) な分散分析表
Analysis of Variance Table
Response: sr
          Df Sum Sq Mean Sq F value    Pr(>F)
pop15      1 204.12  204.12 14.1157 0.0004922 ***
pop75      1  53.34   53.34  3.6889 0.0611255 .
```

```
dpi        1  12.40   12.40  0.8576 0.3593551
ddpi       1  63.05   63.05  4.3605 0.0424711 *
Residuals 45 650.71   14.46
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

# 同じことを個別のモデルで得る
> fit0 <- lm(sr ~ 1, data = LifeCycleSavings)     # 先ず切片項だけのモデル
> fit1 <- update(fit0, . ~ . + pop15)             # 以下逐次項を加えていく
> fit2 <- update(fit1, . ~ . + pop75)
> fit3 <- update(fit2, . ~ . + dpi)
> fit4 <- update(fit3, . ~ . + ddpi)
> anova(fit0,fit1,fit2,fit3,fit4,test="F") # 5 種類の当てはめの分散分析表をまとめて
Analysis of Variance Table
Model 1: sr ~ 1
Model 2: sr ~ pop15
Model 3: sr ~ pop15 + pop75
Model 4: sr ~ pop15 + pop75 + dpi
Model 5: sr ~ pop15 + pop75 + dpi + ddpi
 Res.Df    RSS Df Sum of Sq       F    Pr(>F)
1    49 983.63
2    48 779.51 1  204.12 14.1157 0.0004922 ***    # 先の分析表と同じ結果
3    47 726.17 1   53.34  3.6889 0.0611255 .
4    46 713.77 1   12.40  0.8576 0.3593551
5    45 650.71 1   63.05  4.3605 0.0424711 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> anova(fit4, fit2, fit0, test="F")               # 非慣習的な順序
Analysis of Variance Table
Model 1: sr ~ pop15 + pop75 + dpi + ddpi
Model 2: sr ~ pop15 + pop75
Model 3: sr ~ 1
  Res.Df     RSS Df Sum of Sq      F    Pr(>F)
1     45  650.71
2     47  726.17 -2    -75.45 2.6090 0.0847088 .
3     49  983.63 -2   -257.46 8.9023 0.0005527 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

6.3.5　GLM の分散分析統計量 stat.anova()

これは anova(..., test != NULL) に対する lm() と glm() メソッドであり，普通のユーザが使うべきではない．

書式： stat.anova(table, test=c("Chisq","F","Cp"), scale, df.scale, n)

引数：

table　anova.glm(..., test=NULL) の結果であるような数値行列

test　文字列．"Chisq", "F" または "Cp" のどれかと一致する

scale　重みつきの残差平方和

df.scale　scale に対応する自由度

n　観測値数

返り値：　関数の返り値は元の table と同じ行列で，test 引数に依存した検定統計量からなる列がつけ加わる

関連： anova.lm(), anova.glm().

```
> print(ag <- anova(glm.D93))                     # help(glm) の続き
Analysis of Deviance Table
```

```
Model: poisson, link: log
Response: counts
Terms added sequentially (first to last)
          Df Deviance Resid. Df Resid. Dev
NULL                          8    10.5814
outcome    2   5.4523         6     5.1291
treatment  2   0.0000         4     5.1291
```

6.3.6 GLM モデル当てはめによる逸脱度分析表 anova.glm()

一つもしくは複数の一般化線形モデル当てはめ結果に対する逸脱度分析表を計算する.

書式:

```
# クラス "glm" に対する S3 メソッド
anova(object, ..., dispersion = NULL, test = NULL)
```

引数:

object クラス "lm" のオブジェクトで, 普通 glm() の呼出し結果か, "glmlist" メソッドに対する objects のリスト

dispersion 当てはめファミリに対する散らばりパラメータ. 既定ではこれは glm.obj から得られる

test 使用される検定統計量を指定する文字列. "F", "Chisq" または "Cp" のどれかで, 部分的一致で良い. stat.anova() を参照せよ

返り値: クラス "data.frame" を継承するクラス "anova" のオブジェクト

単一のオブジェクトを指定すると, その当てはめに対する逐次的な逸脱度分析表を与える. つまり, 各項が順に加えられた際の残差逸脱度に於ける減少と, 残差逸脱度自体が表の行として与えられる. もし複数のオブジェクトを指定すると, 表は各モデルに対する残差自由度と残差逸脱度に対する行を持つ. 最初のモデルを除く全てのモデルに対し, 自由度と残差平方和の変化がまた与えられる. (これはモデルが入れ子になっているときだけ統計的に意味がある.) モデルを最小モデルから最大モデルへと並べるのが慣習であるが, しかしユーザの好みによる.

オプションとして表は行に対する逸脱度の減少と残差を比較する検定統計量 (と p 値) を含むことができる. 既知の散らばりパラメータ (例えば, 二項・ポアッソン当てはめ) ではカイ 2 乗検定統計量が最も適当であり, モーメントから散らばりパラメータが推定される場合 (例えば gaussian, quasibinomial, quasipoisson 当てはめ) は F 検定統計量が最も適当である. Mallows の Cp 統計量は, 残差逸脱度と残差自由度の $2\sigma^2$ 倍を加えたものであり, AIC (もし散らばりパラメータが既知ならその定数倍) に深い関係がある.

警告: anova() もしくは anova.glmlist() による二つ以上のモデルの比較は, それらが同じデータセットに当てはめられたときだけ意味がある. このことは, もし欠損値がありRの既定動作である na.action = na.omit が使われた時は問題無しとはしない. anova.glmlist() はこれをエラーとして検出する.

関連: glm(), anova(). 各項がそれらの階層性を保ちながら一時に一つずつ取り去られるいわゆる type II 分散分析表については drop1().

```
> anova(glm.D93)                                  # help(glm) の続き
Analysis of Deviance Table
Model: poisson, link: log
Response: counts
Terms added sequentially (first to last)
          Df Deviance Resid. Df Resid. Dev        # 検定統計量無し (既定)
NULL                          8    10.5814
outcome    2   5.4523         6     5.1291
```

```
treatment  2   0.0000         4     5.1291

> anova(glm.D93, test = "Cp")                    # Cp 統計量指定
Analysis of Deviance Table
Model: poisson, link: log
Response: counts
Terms added sequentially (first to last)
          Df Deviance Resid. Df Resid. Dev     Cp
NULL                          8    10.5814 12.581
outcome    2   5.4523         6     5.1291 11.129
treatment  2   0.0000         4     5.1291 15.129

> anova(glm.D93, test = "Chisq")                 # χ 2 乗統計量指定
Analysis of Deviance Table
Model: poisson, link: log
Response: counts
Terms added sequentially (first to last)
          Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                          8    10.5814
outcome    2   5.4523         6     5.1291    0.0655
treatment  2   0.0000         4     5.1291    1.0000
```

6.3.7 多変量分散分析 manova()

多変量分散分析に対するクラス属性を与える.

書式： manova(...)

引数： ... aov() に引き渡される引数

クラス "manova" はクラス "aov" と要約メソッド summary の選択が異なる．関数 manova() は aov() を呼び出し，それから各層に対する結果オブジェクトにクラス属性 "manova" を加える．詳しくは aov() とコメントを参照せよ.

注意：manova() は多層の分散分析をサポートしないので，モデル式は Error 項を含むべきではない.

関連：aov(), summary.manova().

6.3.8 多変量分散分析に対する要約メソッド summary.manova()

summary.manova() は多変量分散分析に対する要約メソッド関数である.

書式：

```
# クラス "manova" に対する S3 メソッド
summary(object, test=c("Pillai","Wilks","Hotelling-Lawley","Roy"),
        intercept=FALSE, ...)
```

引数：

object クラス "manova" もしくは "aov" のオブジェクトで，多変量目的変数を持つ

test 使用される検定統計量の名前．部分的マッチングによる省略が可能

intercept 論理値．もし TRUE なら切片項が表に含められる

... 他のメソッドへ (から) 引き渡される追加引き数

返り値： 次の成分を持つリスト：

SS 2 乗和と積行列の名前付きリスト
Eigenvalues 固有値の行列
stats 統計量の行列，近似 F 値，自由度，そして p 値

summary.manova() メソッドは要約表に対して多変量検定統計量を用いる．Wilks 統計量が文献では最も普通であるが，既定の Pillai-Bartlett 統計量が Hand & Taylor で勧められている．

関連：manova(), aov().

```
# Krzanowski のプラスチックフィルム生産データ
> tear <- c(6.5, 6.2, 5.8, 6.5, 6.5, 6.9, 7.2, 6.9, 6.1, 6.3,
            6.7, 6.6, 7.2, 7.1, 6.8, 7.1, 7.0, 7.2, 7.5, 7.6)
> gloss <- c(9.5, 9.9, 9.6, 9.6, 9.2, 9.1, 10.0, 9.9, 9.5, 9.4,
             9.1, 9.3, 8.3, 8.4, 8.5, 9.2, 8.8, 9.7, 10.1, 9.2)
> opacity <- c(4.4, 6.4, 3.0, 4.1, 0.8, 5.7, 2.0, 3.9, 1.9, 5.7,
               2.8, 4.1, 3.8, 1.6, 3.4, 8.4, 5.2, 6.9, 2.7, 1.9)
> Y <- cbind(tear, gloss, opacity)
> rate <- factor(gl(2,10), labels=c("Low", "High"))
> additive <- factor(gl(2, 5, len=20), labels=c("Low", "High"))

> fit <- manova(Y ~ rate * additive)              # manova 関数の使用
> summary.aov(fit)                                # 多変量 ANOVA 表要約
 Response tear :
              Df  Sum Sq Mean Sq F value   Pr(>F)
rate           1 1.74050 1.74050 15.7868 0.001092 ** # 0.5% 有意
additive       1 0.76050 0.76050  6.8980 0.018330 *  # 5% 有意
rate:additive  1 0.00050 0.00050  0.0045 0.947143
Residuals     16 1.76400 0.11025
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

 Response gloss :
              Df  Sum Sq Mean Sq F value  Pr(>F)
rate           1 1.30050 1.30050  7.9178 0.01248 *
additive       1 0.61250 0.61250  3.7291 0.07139 .
rate:additive  1 0.54450 0.54450  3.3151 0.08740 .
Residuals     16 2.62800 0.16425
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

 Response opacity :
              Df Sum Sq Mean Sq F value Pr(>F)
rate           1  0.420   0.420  0.1036 0.7517
additive       1  4.901   4.901  1.2077 0.2881
rate:additive  1  3.961   3.961  0.9760 0.3379
Residuals     16 64.924   4.058

> summary(fit, test="Wilks")                      # Wilks の λ 検定指定
              Df  Wilks approx F num Df den Df   Pr(>F)
rate           1 0.3819   7.5543      3     14 0.003034 **
additive       1 0.5230   4.2556      3     14 0.024745 *
rate:additive  1 0.7771   1.3385      3     14 0.301782
Residuals     16
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

6.3.9 Tukey の Honest Significant Difference 法 TukeyHSD()

TukeyHSD() は指定されたファミリワイズの被覆確率を持つ因子水準の平均の差に関する信頼水準のセットを作る．信頼区間はステューデント化範囲統計量 (Tukey の Honest Significant Difference 法) に基づく．プロットメソッドがある．

書式：TukeyHSD(x, which, ordered = FALSE, conf.level = 0.95, ...)

引数：

x 当てはめモデルオブジェクト．普通 aov() 関数による当てはめ結果

which 区間を計算すべき当てはめモデル中の項のリスト．既定では全ての項

ordered 論理値．差を取る前に，因子水準を標本の平均の増加順に並べるかどうかを指示する．もし TRUE なら計算された平均差は全て正になる．有意な差は lwr 端点が正であるようなものである

conf.level ファミリワイズの信頼係数を与える 0 と 1 の間の数

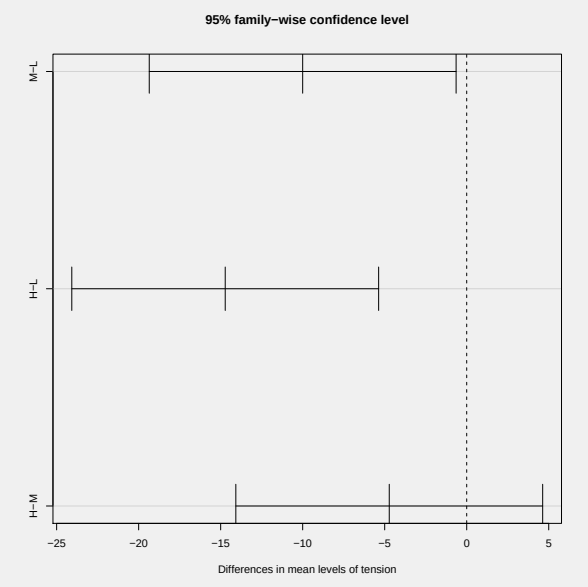
... オプションの追加引数．現在未使用

返り値： 各成分が which で指定された各項に対応する．各成分は行列で，列 diff は標本平均の差，lwr は区間の下端点，そして upr は区間の上端点

分散分析で因子の水準に対する平均を比較するさい，t 検定を使う単純な比較では，実際には存在しない差を有意と判断する確率を増加させてしまう．これは与えられた被覆確率に対する信頼区間が各区間毎に計算されるのに対し，被覆は普通全区間族に渡って解釈されるからである．John Tukey は個々の差ではなく，標本平均の範囲に基づく区間を提唱した．この関数が計算する区間はステューデント化範囲統計量に基づく．理論的にはこうした区間は，因子の各水準毎に同数の観測値がある釣合型デザインに対してだけ適用されるべきである．この関数は標本数に関する調整を行うことによって，観測数が多少異なる場合にも意味ある結果が得られるようにしてある．

関連： aov(), qtukey(), model.tables().

```
# 紡織機の縦糸の切断データ warpbreaks データ使用
> summary(fm1 <- aov(breaks ~ wool + tension, data = warpbreaks))
            Df Sum Sq Mean Sq F value   Pr(>F)
wool         1  450.7   450.7  3.3393 0.073614 .
tension      2 2034.3  1017.1  7.5367 0.001378 **
Residuals   50 6747.9   135.0
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> TukeyHSD(fm1, "tension", ordered = TRUE)
  Tukey multiple comparisons of means
    95% family-wise confidence level
    factor levels have been ordered
Fit: aov(formula = breaks ~ wool + tension, data = warpbreaks)
# 3 種類の tension グループ L,M,H 間の平均差の信頼区間 (L-H,L-M 間で 5% 有意な平均差検出)
$tension
         diff        lwr      upr                  # 区間が 0 を含む (5% 有意でない)
M-H  4.722222 -4.6311985 14.07564                  # 区間が 0 を含まず (5% 有意)
L-H 14.722222  5.3688015 24.07564                  # 区間が 0 を含まず (5% 有意)
L-M 10.000000  0.6465793 19.35342
```



TukeyHSD() による 3 種類の縦糸張力グループ L,M,H 間の平均差の信頼区間 (L-H,L-M 間で 5

6.4　モデル選択

6.4.1　変数増減による AIC モデル選択 step()

変数をステップ毎に増減し，AIC 規準でモデルを選択する．

書式：

```
step(object, scope, scale = 0,
     direction = c("both", "backward", "forward"),
     trace = 1, keep = NULL, steps = 1000, k = 2, ...)
```

引数：

`object`　適当なクラス (主として `"lm"` と `"glm"`) のモデルを表すオブジェクト．これはステップワイズ探索の初期モデルになる

`scope`　ステップワイズ探索で調べられるモデルの範囲を定義する．これは単一のモデル式か，二つの式 `upper` と `lower` を成分に持つリストでなければならない．式をどのように指定し，どのように使われるかは以下を参照せよ

`scale`　モデルを選ぶ際の AIC 統計量の定義で使われる．現在 `lm`, `aov` そして `glm` モデルだけがサポートされている

`direction`　ステップワイズ探索のモードで `"both"`, `"backward"` もしくは `"forward"` が選べ，既定値は `"both"` である．もし `scope` 引数が無ければ `direction` に対する既定値は `"backward"`

`trace`　もし正ならば，`step()` 実行途中の情報が出力される．値が大きい程より詳細な情報が与えられる

`keep`　入力が当てはめモデルと対応 AIC 統計量で，出力が任意であるフィルタ関数．典型的には `keep` はオブジェクトの成分の一部分を選択し，それらを返す．既定では何も保存しない

`steps`　考慮される最大ステップ数．既定値は 1000(本質的に任意回数に相当)．これは

典型的には探索過程を早めに停止させるのに使われる

k ペナルティとして使われる自由度数の倍数．k=2 が本来の AIC 法を与える．選択 k=log(n) はしばしば BIC や SBC と呼ばれる

... extractAIC() への任意追加引数

返り値： 返り値はステップワイズに選択されたモデルであり，最大二つの追加成分を持つ．探索途中のステップに対応する "anova" 成分と同時に，もし呼出し時に keep 引数が与えられると "keep" 成分が与えられる．逸脱度分析表の "Resid. Dev" 列は，ある定数から 2 倍された最大尤度を示す．これは飽和モデルが適切に定義できるときだけ逸脱度になる (したがって，例えば lm, aov そして survreg 当てはめは除外される)

step() は add1() と drop1() を繰りかえし使う．これはこれらの関数が使え extractAIC() に対する適正なメソッドを持つ任意のメソッドに対して使える．もし付加定数が AIC が Mallows の Cp に等しくなるように選ぶことができれば，そうなるようにされ，表は適切にラベル付けされる．

探索されるモデル集合は scope 引数で決定される．その lower 成分の右辺は常にモデルに含められる．もし scope が単独の式なら，それは upper 成分を指定し lower モデルは空である．もし scope が無ければ，初期モデルが upper モデルとされる．scope により指定されるモデルは update.formula() により使われるように object を更新する雛型として使える．

変化する scope を持つ glm 当てはめの使用に当たっては潜在的な問題がある．何故なら，その場合逸脱度は最大対数尤度と単純に関連しないからである．"glm" メソッドに対する extractAIC() 関数は gaussian ファミリに対して適当な補整を行うが，その他の場合は修正が必要になるかも知れない．(binomial と poisson ファミリは既定で固定した scale を持つが，scale 変数に対する特定の最尤問題には対応しない．)

警告： モデル当てはめは同じデータセットに適用されなければならない．これは欠損値があり，R の既定動作 na.action = na.omit が使われた場合は問題を引き起こす可能性がある．最初に欠損値を除いておくことを勧めたい．

注意： この関数は S のそれとかなり異なる．S の関数は幾つかの近似を用い，正しい AIC を計算しない．これは最小の移植であり，より広いオブジェクトクラスに対しては stepAIC() を使おう．step() はパッケージ MASS 中の stepAIC() を少し簡略化したものである．

関連： stepAIC(), add1(), drop1()

```
> step(lm.D9)                                    # example(lm) に続く
Start:  AIC= -12.58                              # 当初モデル
 weight ~ group
        Df Sum of Sq      RSS      AIC
- group  1    0.6882   9.4175 -13.0633
<none>                 8.7293 -12.5811
Step:  AIC= -13.06                       # group 変数を除いたモデル (この場合の選択モデル)
 weight ~ 1                                      # 切片 (定数) 項だけのモデル
Call:
lm(formula = weight ~ 1)
Coefficients:
(Intercept)
```

```
      4.846
```

```
# Swiss の出生数と 5 種類の社会・経済指標データ swiss を使用する
> summary(lm1 <- lm(Fertility ~ ., data = swiss))
Call:
lm(formula = Fertility ~ ., data = swiss)
Residuals:
     Min       1Q   Median       3Q      Max
-15.2743  -5.2617   0.5032   4.1198  15.3213
Coefficients:            # 5 種類の変数と切片項. Examination だけが 5% 有意でない
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)      66.91518   10.70604   6.250 1.91e-07 ***
Agriculture      -0.17211    0.07030  -2.448  0.01873 *
Examination      -0.25801    0.25388  -1.016  0.31546
Education        -0.87094    0.18303  -4.758 2.43e-05 ***
Catholic          0.10412    0.03526   2.953  0.00519 **
Infant.Mortality  1.07705    0.38172   2.822  0.00734 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 7.165 on 41 degrees of freedom
Multiple R-Squared: 0.7067,     Adjusted R-squared: 0.671
F-statistic: 19.76 on 5 and 41 DF,  p-value: 5.594e-10

> slm1 <- step(lm1)                                  # 変数を順に一つずつ除いたモデルを試す
Start:  AIC= 190.69                                  # 5 変数による線形回帰（当初モデル）
 Fertility ~ Agriculture + Examination + Education + Catholic +
    Infant.Mortality
                   Df Sum of Sq    RSS    AIC
- Examination       1      53.0 2158.1  189.9     # Examination 変数を除いたモデル
<none>                          2105.0  190.7     # 第一ステップ当初モデル
- Agriculture       1     307.7 2412.8  195.1     # 以下各変数を一つずつ除いたモデル
- Infant.Mortality  1     408.8 2513.8  197.0
- Catholic          1     447.7 2552.8  197.8
- Education         1    1162.6 3267.6  209.4
Step:  AIC= 189.86                                   # Examination 変数を除いたモデルを選択
# 更に残りの 4 変数を順に一つずつ除いたモデルを試す
# （これ以上の変数の増減で AIC 値の一層の低下は無く，探索は終了した）
 Fertility ~ Agriculture + Education + Catholic + Infant.Mortality
                   Df Sum of Sq    RSS    AIC
<none>                          2158.1  189.9     # 第一ステップの最終モデル
- Agriculture       1     264.2 2422.2  193.3     # 以下各変数を一つずつ除いたモデル
- Infant.Mortality  1     409.8 2567.9  196.0
- Catholic          1     956.6 3114.6  205.1
- Education         1    2250.0 4408.0  221.4

> summary(slm1) # ステップワイズ探索結果の要約（Examination 変数だけを除いたモデルを選択）
Call:
lm(formula = Fertility ~ Agriculture + Education + Catholic +
    Infant.Mortality, data = swiss)
Residuals:
     Min       1Q   Median       3Q      Max
-14.6765  -6.0522   0.7514   3.1664  16.1422
Coefficients:          # 4 変数と切片項は全て 5% 有意だが，（修正済み）決定係数はさほど高くない
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)      62.10131    9.60489   6.466 8.49e-08 ***
Agriculture      -0.15462    0.06819  -2.267  0.02857 *
Education        -0.98026    0.14814  -6.617 5.14e-08 ***
Catholic          0.12467    0.02889   4.315 9.50e-05 ***
Infant.Mortality  1.07844    0.38187   2.824  0.00722 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 7.168 on 42 degrees of freedom
Multiple R-Squared: 0.6993,     Adjusted R-squared: 0.6707
F-statistic: 24.42 on 4 and 42 DF,  p-value: 1.717e-10

> slm1$anova                                       # 逸脱度分析表（当初モデルと最終モデルの比較）
          Step Df Deviance Resid. Df Resid. Dev      AIC
1               NA       NA        41   2105.043 190.6913
2 - Examination  1 53.02656        42   2158.069 189.8606
```

6.4.2 AIC 規準 AIC()

AIC() は式 $-2\log$(最大尤度) $+ k \times$ npar により，尤度が計算できるような一つもしくは複数の当てはめモデルオブジェクトに対し赤池の情報量規準を計算する総称的関数である．ここで npar は当てはめモデル中のパラメータ数で，$k = 2$ なら通常の AIC，$k = \log n$ (n は観測値数) ならばいわゆる BIC または SBC (Schwarz のベイズ規準) となる．

書式： AIC(object, ..., k = 2)

引数：

object　対応する対数尤度が取り出せる logLik クラス，またはそれを継承するクラスの当てはめオブジェクト

...　オプションの他の当てはめ一つ以上のオブジェクト

k　数値．使用されるパラメータ毎のペナルティ．既定の k=2 は古典的な AIC

返り値：　一つのオブジェクトだけが与えられると，対応する AIC (k に応じて BIC やその他) 値が返される．二つ以上のオブジェクトが与えられると，列がオブジェクトに対応し，列がモデル中のパラメータ数 (df) と AIC 値に対応するデータフレームが返される

AIC() に対する既定のメソッドである AIC.default() は与えられたクラスに対する対数尤度を計算する logLik() メソッドに完全に依存する．当てはめオブジェクトを比較する際，AIC 値がより小さければ当てはめがより良い，とされる．

関連： extractAIC(), logLik().

```
# Swiss の出生数と社会・経済指標データ swiss を使用
> lm1 <- lm(Fertility ~ . , data = swiss)
> AIC(lm1)                                          # 当てはめモデルに対する本来の AIC 値
[1] 326.0716
> stopifnot(all.equal(AIC(lm1), AIC(logLik(lm1)))) # AIC は本来対数尤度から計算される
> AIC(lm1, k = log(nrow(swiss)))                    # BIC のバージョン，もしくは Schwarz の BC
[1] 339.0226
```

6.4.3 当てはめモデルの AIC 規準を計算 extractAIC()

当てはめパラメトリックモデルに対する (一般化) 赤池情報量規準 AIC を計算する．

書式： extractAIC(fit, scale, k = 2, ...)

引数：

fit　当てはめモデル．普通当てはめ関数 lm() 等の結果

scale　モデルのスケールパラメータを指定するオプションの数値．step() の引数 scale を参照せよ

k　AIC 公式中の等価自由度 (edf と置く) 部分の「重み」を指定する数値

... その他の引数 (現在 R では使われていない)

返り値： 長さ 2 のベクトルで次の要素を持つ：
edf 当てはめモデル fit の「同値自由度」
AIC fit に対する (一般化された) 赤池の情報量規準

これは総称関数で，クラス "aov", "coxph", "glm", "lm", "negbin" そして "survreg" に対するメソッド関数を R の基本パッケージ中に持つ．使用される規準は

$$\texttt{AIC} = -2\log L + k \times \texttt{edf}$$

であり，ここで L は最大尤度であり，edf は fit の等価自由度 (つまり普通のパラメトリックモデルに対するパラメータ数) である．未知のスケールを持つ線形モデル (つまり，lm() と aov()) に対しては，$-2\log L$ は逸脱度から計算され，AIC とは異なる付加定数を持つ．k = 2 は本来の AIC に対応し，k = log(n) とすれば BIC (Bayes IC) になる．これ以上，特に scale に付いて，は step() を見よ．

注意：これらの関数は add1(), drop1() そして step() で用いられ，それが主な使途である．
関連：AIC(), deviance(), add1(), step().

```
> extractAIC(glm.D93)                                  # example(glm) に続く
[1]  5.00000 56.76132
```

6.4.4 モデルの逸脱度 deviance()

deviance() は当てはめモデルの逸脱度 (deviance) を返す．

書式：deviance(object, ...)

引数：
object 逸脱度を求めたいオブジェクト
... 追加のオプション引数

返り値： オブジェクト object から取り出された逸脱度

これは当てはめモデルから逸脱度を取り出す総称的関数である．この関数をどう使うかの詳細は個別のモデリング関数を考慮すること．

関連：df.residual(), extractAIC(), glm(), lm().

6.4.5 回帰モデルの対数尤度を取り出す logLik()

この関数は総称的である．特定のクラスのオブジェクトに対するメソッド関数を書くことができる．この関数に対するメソッドを既に持つ関数には glm(), lm(), nls() そしてパッケージ nlme 中の関数 lme(), gls() がある．

書式：

```
# クラス "lm" に対する S3 メソッド
logLik(object, REML = FALSE, ...)
```

引数：

`object` 対数尤度値もしくはそれへの寄与を取り出すことができる任意のオブジェクト

`REML` オプションの論理値．もし TRUE なら制約付き対数尤度値が返される．既定値の `FALSE` ならば対数尤度値が返される

`...` この総称的関数のメソッド関数が必要とする追加の引数

返り値： クラス `logLik` のオブジェクト (`r` とする) を返す．これは幾つかの属性を持つ数値である．`attr(r, "df")` はモデル中のパラメータの数である．`logLik` オブジェクトは簡単な `print` メソッドを持つ．詳細は使われるメソッド関数に依存するので，適当なドキュメントを参照せよ

`glm()` による当てはめでは `family` は対数尤度をどのように計算するか必ずしも特定する必要はないでの，これは AIC を計算するファミリの関数に依存する．`gaussian`, `Gamma` そして `inverse.gaussian` ファミリでは，GLM の散布度が計算され AIC 中に含まれていると仮定されており，その他の全てのファミリでは散布度は既知と仮定されている．この手順は gamma そして inverse gamma ファミリでは，使われている散布度は最尤推定量ではないので，完全に正確とはいえない．

関連：`logLik.gls()`, `logLik.lme()` 等．

```
> x <- 1:5
> lmx <- lm(x ~ 1)
> logLik(lmx)                                   # print.logLik() ソッドを使用
'log Lik.' -8.82756 (df=2)
> str(logLik(lmx))
Class 'logLik' : -8.828 (df=2)

# lm メソッド．従業員勤務態度データ attitude 使用
> (fm1 <- lm(rating ~ ., data = attitude))
Call:
lm(formula = rating ~ ., data = attitude)
Coefficients:
(Intercept)   complaints   privileges     learning       raises     critical
   10.78708      0.61319     -0.07305      0.32033      0.08173      0.03838
    advance
   -0.21706
> logLik(fm1)
'log Lik.' -97.2499 (df=8)                    # 対数尤度値
> logLik(fm1, REML = TRUE)
'log Lik.' -102.6851 (df=8)                   # 制約付き対数尤度値
```

```
# パッケージ nlme 中の歯科矯正データ Orthodont を使用
> res <- try(data(Orthodont, package="nlme"))
> if(!inherits(res, "try-error")) {
    fm1 <- lm(distance ~ Sex * age, Orthodont)
    print(logLik(fm1))
    print(logLik(fm1, REML = TRUE))
  }
'log Lik.' -239.1209 (df=5)                   # 対数尤度値
'log Lik.' -241.7796 (df=5)                   # 制約付き対数尤度値
```

6.5　その他の回帰手法

6.5.1　射影追跡回帰 ppr()

ppr() は射影追跡回帰モデル (projection pursuit regression model) を当てはめる．

書式：

```
ppr(x, ...)
# クラス "formula" に対する S3 メソッド
ppr(formula, data, weights, subset, na.action,
    contrasts = NULL, ..., model = FALSE)
# 既定の S3 メソッド
ppr(x, y, weights = rep(1,n),
    ww = rep(1,q), nterms, max.terms = nterms, optlevel = 2,
    sm.method = c("supsmu", "spline", "gcvspline"),
    bass = 0, span = 0, df = 5, gcvpen = 1, ...)
```

引数：

formula　一つもしくは複数の目的変数と説明変数を指定するモデル

x　説明変数の数値行列．行は観測値を表し，列は変数を表す．欠損値は許されない

y　目的変数の数値行列．行は観測値を表し，列は変数を表す．欠損値は許されない

nterms　最終モデルに含められるべき項の数

data　formula 中で指定された変数の中で，優先的に採用されるもののデータフレーム

weights　各ケースに対する重み w_i のベクトル

ww　各目的変数に対する重み ww_j のベクトルで，当てはめの規準は ww_i*ww_j*(y_ij-fit_ij)^2 の i 番目のケースと j 番目の目的変数に対する和を ww_i の和で割ったものになる

subset　教師標本として使われるべきケースを指示する添字ベクトル．(注意：もしこれを与えるなら，この引数は名前付きでなければならない)

na.action　NA が発見されたときに取るべき処置を指示する関数．既定動作は getOption("na.action") で決まる．(注意：もしこれを与えるなら，この引数は名前付きでなければならない)

contrasts　因子説明変数をコードする際に使われる対比

max.terms　モデルを構築する際に選ばれる最大の項数

optlevel　0 から 3 迄の整数で，SRMAT プログラムによる最適化の程度を決める．下の解説を参照せよ

sm.method　リッジ関数を平滑化する手法．既定は Friedman の super smoother 関数 supsmu()．もう一つの選択肢は smooth.spline() 関数の基礎である平滑化スプラインコードであり，各リッジ関数に対する (同値) 自由度を指定するか，GCV で平滑度を決めさせる

bass　自動的な幅選択に用いられる super smoother の bass 制御変数 (supsmu() を見よ)．値の範囲は 0 から 10 で，大きい程滑らかになる

span　super smoother の span 制御変数 (supsmu() を見よ)．既定値は 0 で，局所的

なクロスバリデーションによる自動的な幅選択． span はまた (0,1] 中の値で良い

df もし sm.method="spline" ならば，各リッジ項の滑らかさを要求された同値自由度により指定する

gcvpen もし sm.method="gcvspline" ならば，これは使われる各自由度に対する GCV による選択で用いられるペナルティである

... 他のメソッドへ (から) 引き渡される引数

model 論理値．もし真ならモデルフレームが返される

返り値： 次の成分を持つリストで，多くはメソッド関数で使われる：

call マッチした呼び出し

p (任意のコーディング後の) 説明変数の数

q 目的変数の数

mu 引数 nterms

ml 引数 max.terms

gof 選択モデルに対する総合的な (重みつき) 残差平方和

gofn max.terms までの項数に対する総合的な (重みつき) 残差平方和．nterms 以下では不正 (そして 0) になる

df 引数 df

edf もし sm.method が "spline" か "gcvspline" なら，使われた各リッジ項に対する同値自由度

xnames 説明変数の数

ynames 目的変数の名前

alpha 射影方向の行列で，各リッジ項が列に対応する

beta 各目的変数に対してリッジ項に適用された係数の行列．行は目的変数で列はリッジ項

yb 各説明変数の重みつき和

ys 使われた総合的なスケール因子．内部的には目的変数は全重みつき平方和が 1 になるように ys で割られる

fitted.values 当てはめ値．もし q>1 なら行列となる

residuals 残差．もし q>1 なら行列となる

smod 内部的な作業配列で，訓練点集合に於いて評価されたリッジ関数を含む

model (model=TRUE の時だけ) モデルフレーム

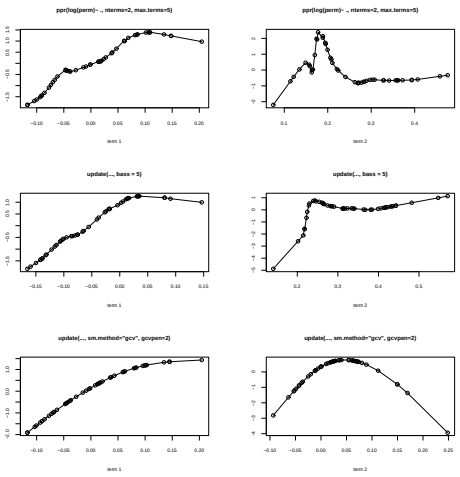
基本のメソッドは Friedman のそれであり，S-PLUS の ppreg() と本質的に同じコードである．このコードは非常に複雑であり，結果は使用されたコンパイラに敏感である．アルゴリズムはまず最大 max.terms 個のリッジ項を一度に一つずつ付け加える．十分な差を生じる付け加えるべき項が見付からなければ，最大個数未満の項を使用する．それから nterm 個の項が残るまで，「最も重要でない」項を各ステップ毎に取り除く．

最適化のレベル (引数 optlevel) は，この過程でモデルがどれくらい再当てはめをされるかを指示する．レベル 0 では既存のリッジ項は再当てはめされない．レベル 1 では射影方向は再当てはめされないが，リッジ関数と回帰係数は再当てはめされる．レベル 2,3 では全ての項が再当てはめされ，目的変数が一つなら同値になる．レベル 3 は各ステップで各説明変数からの寄与を再バランスさせるように最も注意を払う．したがって，2 乗和

基準の鞍点では少し収束しにくい.

```
# 注意：結果の数値は使用システムにより違う可能性がある
> data(rock)                                          # 油層の岩石標本の透水性データ読み込み
> attach(rock)                                        # その成分変数を現在の環境に展開する
> area1 <- area/10000; peri1 <- peri/10000

# 浸透率の自然対数を射影追跡回帰（最終項数 nterms=2 を指定）
> rock.ppr <- ppr(log(perm) ~ area1 + peri1 + shape,
                  data = rock, nterms = 2, max.terms = 5)
> rock.ppr
Call:
ppr(formula = log(perm) ~ area1 + peri1 + shape, data = rock,
    nterms = 2, max.terms = 5)
Goodness of fit:                                      # 項数を増やしたときの当てはめの良さ
 2 terms  3 terms  4 terms  5 terms
8.737806 5.289517 4.745799 4.490378

> summary(rock.ppr)                                   # 要約メソッド
Call:
ppr(formula = log(perm) ~ area1 + peri1 + shape, data = rock,
    nterms = 2, max.terms = 5)
Goodness of fit:
 2 terms  3 terms  4 terms  5 terms
8.737806 5.289517 4.745799 4.490378
Projection direction vectors:
      term 1      term 2
area1  0.34357179  0.37071027
peri1 -0.93781471 -0.61923542
shape  0.04961846  0.69218595
Coefficients of ridge terms:
   term 1    term 2
1.6079271 0.5460971
```

```
# 3 種類の射影追跡回帰結果をプロットする（以下の図を参照）
> par(mfrow=c(3,2), pty="s")                          # 画面を 3x2 分割，正方形領域指定
> plot(rock.ppr, main="ppr(log(perm)~ ., nterms=2, max.terms=5)")

# update() 関数で bass パラメータを変更し，滑らかさを増す
> plot(update(rock.ppr, bass=5), main = "update(..., bass = 5)")
# update() 関数でクロスバリデーションによる平滑化を採用
> plot(update(rock.ppr, sm.method="gcv", gcvpen=2),
       main = "update(..., sm.method=\"gcv\", gcvpen=2)")

# 透水性データ値とその予測値（小数点以下一桁で丸める）
> cbind(perm=rock$perm, prediction=round(exp(predict(rock.ppr)), 1))
     perm prediction
1     6.3        5.9
2     6.3        6.5
(途中省略)
48  580.0      571.5
> detach()                                            # rock データの変数を現在の環境から取り除く
```



3 種類の射影追跡回帰結果のプロット

左は結果の面積，右は周長による変化のプロット
上段は `nrterms=2` のときの既定当てはめ結果
中段は super smoother の平滑度を増やした結果
下段は平滑化にクロスバリデーションを使用した結果

6.5.2 isotonic 回帰関数 isoreg()

`isoreg()` 関数は区分的に定数である単調増加な関数を (ノンパラメトリックな) 最小自乗法で当てはめる．

書式： `isoreg(x, y = NULL)`

引数：

`x, y` 回帰点の座標ベクトル．または，単一のプロット構造を指定できる．`xy.coords()` を見よ

`...` メソッドに引き渡される追加引き数

返り値： 以下の成分を持つクラス `"isoreg"` のリスト：

`x` オリジナルの (構成された) `x` の水平座標値

`y` 対応する `y` 値

`yf` 大小順に並べ換えられた `x` 値に対応する当てはめ値

`yc` 大小順に並べ換えられた `x` 値に対応する `y` の累積値

`iKnots` 当てはめ曲線がそこでジャンプする (つまり convex minorant がそこで折れ曲がる) 添字を与える整数ベクトル

`isOrd` 元々の x 値が既に昇順に並んでいるかどうかを指示する論理値

`ord` `if(!isOrd)` の時．元々の x 値の整数置換である `order(x)`

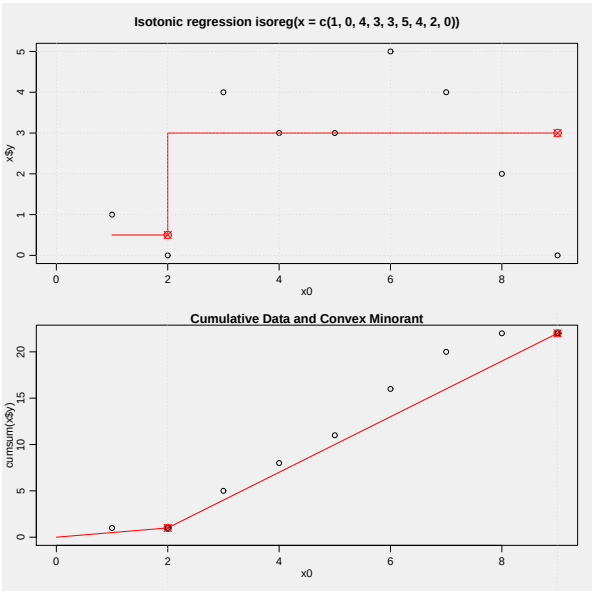
`call` 使われた `isoreg()` に対する呼出し式

アルゴリズムは累積データ (つまり `cumsum(y)`) に対する区分的に線形である convex minorant[*3] $m(x)$ を計算する．結果はその微分である $m'(x)$ であり，$m(x)$ が累積データの凸包に接し，傾きを変える箇所で水準が変化する．`as.stepfun()` はしたクラス `"stepfun"` (階段関数) のオブジェクトを返し，より節約した表現になる可能性がある．

関連： プロットメソッド `plot.isoreg()` はより多くの例を持つ．MASS パッケージの `isoMDS()` は内部的にアイソトニック回帰を使う．

[*3] ある集合 A の「convex minorant」とは，A の下にあるような直線全体の上限であるような凸曲線である．

```
> (ir <- isoreg(c(1,0,4,3,3,5,4,2,0)))            # x=1:9 に対する y 座標値ベクトルが引き数
Isotonic regression from isoreg(x = c(1, 0, 4, 3, 3, 5, 4, 2, 0)),
  with 2 knots / breaks at obs.nr. 2 9 ;          # x=2,9 で折れ曲がる（x=9 は実際は端点）
  initially ordered 'x'
  and further components List of 4
 $ x : num [1:9] 1 2 3 4 5 6 7 8 9                # 与えられた x 値
 $ y : num [1:9] 1 0 4 3 3 5 4 2 0                # 対応する y 値
 $ yf: num [1:9] 0.5 0.5 3 3 3 3 3 3 3            # 結果の回帰曲線の y 座標
 $ yc: num [1:10] 0 1 1 5 8 11 16 20 22 22        # その累積値
> plot(ir, plot.type = "row")                   # plot.isoreg() が使われる（以下の図を参照）

> (ir3 <- isoreg(y3 <- c(1,0,4,3,3,5,4,2,3)))     # 最後は"3"であり"0"でない
Isotonic regression from isoreg(x = y3 <- c(1, 0, 4, 3, 3, 5, 4, 2, 3)),
  with 3 knots / breaks at obs.nr. 2 5 9 ;
  initially ordered 'x'
  and further components List of 4
 $ x : num [1:9] 1 2 3 4 5 6 7 8 9
 $ y : num [1:9] 1 0 4 3 3 5 4 2 3
 $ yf: num [1:9] 0.50 0.50 3.33 3.33 3.33 ...
 $ yc: num [1:10] 0 1 1 5 8 11 16 20 22 25

> (fi3 <- as.stepfun(ir3))
Step function
Call: isoreg(x = y3 <- c(1, 0, 4, 3, 3, 5, 4, 2, 3))
 x[1:3] =      2,      5,      9
4 plateau levels =    0.5,    0.5, 3.3333,    3.5

> (ir4 <- isoreg(1:10, y4 <- c(5, 9, 1:2, 5:8, 3, 8)))
Isotonic regression from isoreg(x = 1:10, y = y4 <- c(5, 9, 1:2, 5:8, 3, 8)),
  with 5 knots / breaks at obs.nr. 4 5 6 9 10 ;
  initially ordered 'x'
  and further components List of 4
 $ x : num [1:10] 1 2 3 4 5 6 7 8 9 10
 $ y : num [1:10] 5 9 1 2 5 6 7 8 3 8
 $ yf: num [1:10] 4.25 4.25 4.25 4.25 5 6 6 6 6 8
 $ yc: num [1:11] 0 5 14 15 17 22 28 35 43 46 ...

# R2 乗値を小数点以下 2 桁に整形して出力
> cat("R^2 =", formatC(sum(residuals(ir4)^2)/(9*var(y4)), dig=2),"\n")
R^2 = 0.79
```



(上) アイソトニック回帰，増加階段関数の当てはめ
(下) 累積データの convex minorant

6.5.3　頑健な直線当てはめ line()

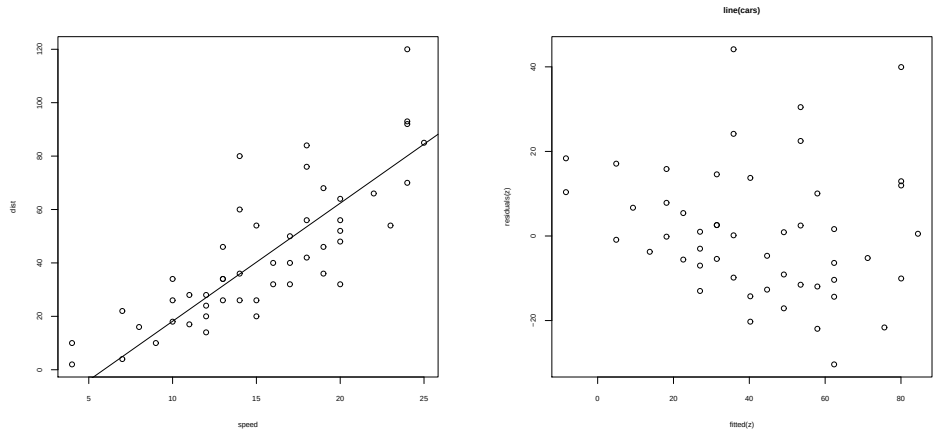
Tukey の「Exploratory Data Analysis」中で推薦されている頑健な直線当てはめ法.

書式： `line(x, y)`

引数： x, y　x と y の対を与える任意の引き数
返り値： 返り値はクラス "tukeyline" のオブジェクト．総称的関数 coef(), residuals(), fitted() そして print() 用のメソッドがある

関連：lm().

```
> plot(cars)                                          # 以下の図を参照
> (z <- line(cars))                                   # 頑健な直線当てはめ
Call:
line(cars)
Coefficients:
[1]  -26.053    4.421
> abline(coef(z))
> plot(residuals(z) ~ fitted(z), main = deparse(z$call)) # Tukey-Anscombe のプロット
```



頑健な直線当てはめ (左) とその残差 (右)

6.6 回帰モデル用総称的関数

以下は回帰モデルを当てはめた結果のオブジェクトから個々の情報を取り出す総称的関数群で，実際はオブジェクトのクラス属性に応じて適当なメソッド関数が呼び出される．線形モデルのみならず，非線形回帰モデルに対しても適切な情報を取り出すことができる．

6.6.1 モデル当てはめ値を取り出す fitted()

fitted() はモデリング関数が返すオブジェクトから当てはめ値を取り出す総称的関数である．fitted.values() はその別称である．モデリング関数が返す全てのオブジェクトクラスは fitted メソッドを持つべきである．(総称的関数は fitted() であり fitted.values() でないことを注意．) メソッドは欠損値の除外を補整する napredict メソッドを利用することができる．既定では lm と glm メソッドが利用している．

書式： fitted(object, ...), fitted.values(object, ...)
引数：
object　モデル当てはめ値の取り出しが意味のあるオブジェクト

...　他の引数
返り値：　オブジェクト x から取り出された当てはめ値

関連：coefficients(), glm(), lm(), residuals().

6.6.2　予測 predict()

predict() は様々なモデル当てはめ関数の結果から，予測を行う総称的関数である．この関数は最初の引数のクラスに依存する，特定のメソッド関数を呼び出す．

書式： predict(object, ...)
引数： object　予測を行いたいモデルオブジェクト ...　生成される予測に影響する追加引数
返り値：　predict() 関数が返す値の形式はその引数のクラスに依存する．メソッド関数が返す内容の詳細は個別のメソッドのドキュメントを参照せよ

線形モデル当てはめに類似した多くの予測メソッドは，予測に用いられる説明変数を指定する引数 newdata を持つ．newdata 中の列を，当てはめに使われるそれらと調和させるために，幾つかの少なからぬ試みがされる．例えば，それらが同一のタイプで，因子が同じ順序の同一の水準セットを持つか (もしくはそうなるように変換できるか) 等である．時系列予測メソッドはいくつ先のステップまで予測をするかを指示する引数 n.ahead を持つ．多くのメソッドは論理値引数 se.fit を持ち，標準誤差を返すかどうかを指示する．

関 連：predict.glm(), predict.lm(), predict.loess(), predict.nls(), predict.poly(), predict.princomp(), predict.smooth.spline(). 時系列予測は predict.ar(), predict.Arima(), predict.arima0(), predict.HoltWinters(), predict.StructTS().

```
# 全ての"predict"メソッドの表示．標準パッケージ中のほとんどのメソッドは隠されていることを注意
> for(fn in methods("predict"))
      try({
            f <- eval(substitute(getAnywhere(fn)$objs[[1]], list(fn = fn)))
            cat(fn, ":\n\t", deparse(args(f)), "\n")
          }, silent = TRUE)
predict.Arima :
  function (object, n.ahead = 1, newxreg = NULL, se.fit = TRUE,  ...)  NULL
predict.HoltWinters :
  function (object,n.ahead=1,prediction.interval=FALSE,level=0.95,...) NULL
(途中省略)
predict.smooth.spline.fit :
  function (object, x, deriv = 0, ...)  NULL
```

6.6.3　モデルの残差を取り出す residuals()

residuals() はモデリング関数が返すオブジェクトから，モデル残差を取り出す総称的関数である．省略形 resid() は residuals() の別名である．これはユーザが，オブジェクトのスロットを直接参照すること無く，アクセス関数でオブジェクト成分にアク

セスすることを推奨することを企図している．モデリング関数が返す全てのオブジェクトクラスは residuals メソッドを持つべきである．(メソッドは residuals() でありresid() ではないことを注意しよう．) 既定のメソッドがそうであるように，メソッドは欠損値の除外を補整するために naresid メソッドを使うことができる．

書式： residuals(object, ...), resid(object, ...)

引数：

object　モデル残差の取り出しが意味を持つオブジェクト
... 他の引数

返り値： オブジェクト object から取り出された残差

関連：coefficients(), fitted.values(), glm(), lm().

6.6.4 モデル項 terms()

関数 terms() は総称的関数で，様々な R データオブジェクトから "terms" オブジェクトを取り出すことができる．

書式： terms(x, ...)

引数：

x　処理するメソッドを選ぶのに使われるオブジェクト
... 他のメソッドへ (から) 引き渡される引数

返り値： 返り値はクラス c("terms", "formula") のオブジェクトで，シンボリックなモデルの項表現を含む．その構造に付いては terms.object() を見よ

クラス "aovlist", "terms", "formula" に対するメソッドがある (terms.formula() を参照)．既定のメソッドは単にオブジェクトの terms 成分 (もしあれば) を取り出す．

関連：terms.object(), terms.formula(), lm(), glm(), formula().

6.6.5 当てはめモデルからの影響 effects()

当てはめモデル，普通線形モデル，から (直交化された) 効果を返す．これは総称的関数であるが，現在クラス "lm"と "glm" を継承するオブジェクトに対するメソッドだけを持つ．

書式：

```
effects(object, ...)
effects(object, set.sign=FALSE, ...)  # クラス"lm"に対する S3 メソッド
```

引数：

object　R オブジェクト．典型的には lm() 等のモデル当てはめ関数の結果
set.sign　論理値．もし TRUE なら，モデル中の係数に対応する効果の符号は対応する係数の符号と同じにされる．さもなければ符号は任意である

... 他のメソッドへ (から) 引き渡される追加引き数

返り値： residuals と同じ長さの (名前付き) 数値ベクトルであるか，もし当てはモデルに多重目的変数があれば行列である．どちらのケースでもクラス "coef" とされる．最初の r 個の行は対応する係数でラベルが付けられ，その他の列はラベルが無い．フルランクでないモデルでは「対応する」係数はピボット操作が行われると順序が変わる

lm や aov により当てはめられた線形モデルに対しては，効果はデータを当てはめ過程で QR 分解により生成された逐次的な直交部分空間にデータを射影することにより得られた無相関の単一自由度値である．最初の r (モデルのランク) 個は係数に関連し，その他は残差の空間を張る (しかし特定の残差に関連しない)．空のモデルは効果を持たない．

関連： coef().

```
> y <- c(1:3,7,5)
> x <- c(1:3,6:7)
> ( ee <- effects(lm(y ~ x)) )                    # 人工的な例
(Intercept)           x                          # 各効果 (切片項と x 項にラベルがつく)
 -8.0498447   4.3655709   0.1483334   1.6144112  -1.2302295
attr(,"assign")
[1] 0 1
attr(,"class")
[1] "coef"

> c(round(ee - effects(lm(y+10 ~ I(x-3.8))),3))   # 最初だけが異なる
(Intercept)           x
     22.361       0.000       0.000       0.000       0.000
```

6.6.6　取り除きによる回帰診断 influence.measures()

この一連の関数は Belsley, Kuh & Welsch, Cook & Weisberg 等で論じられている，線形モデルと一般化線形モデルに対する幾つかの (一時に一つ取り除く) 回帰診断を計算するのに使うことができる．

書式：

```
influence.measures(model)
rstandard(model, ...)
# クラス "lm" に対する S3 メソッド
rstandard(model, infl = lm.influence(model, do.coef=FALSE),
          sd = sqrt(deviance(model)/df.residual(model)), ...)
# クラス "glm" に対する S3 メソッド
rstandard(model, infl = lm.influence(model, do.coef=FALSE), ...)
rstudent(model, ...)
# クラス "lm" に対する S3 メソッド
rstudent(model, infl = lm.influence(model, do.coef=FALSE),
         res = infl$wt.res, ...)
# クラス "glm" に対する S3 メソッド
rstudent(model, infl = influence(model, do.coef=FALSE), ...)
```

```
dffits(model, infl = , res = )
dfbeta(model, ...)
# クラス "lm" に対する S3 メソッド
dfbeta(model, infl = lm.influence(model, do.coef=TRUE), ...)
dfbetas(model, ...)
# クラス "lm" に対する S3 メソッド
dfbetas(model, infl = lm.influence(model, do.coef=TRUE), ...)
covratio(model, infl = lm.influence(model, do.coef=FALSE),
         res = weighted.residuals(model))
cooks.distance(model, ...)
# クラス "lm" に対する S3 メソッド
cooks.distance(model, infl = lm.influence(model, do.coef=FALSE),
               res = weighted.residuals(model),
               sd = sqrt(deviance(model)/df.residual(model)),
               hat = infl$hat, ...)
# クラス "lm" に対する S3 メソッド
cooks.distance(model, infl = influence(model, do.coef=FALSE),
               res = infl$pear.res,
               dispersion = summary(model)$dispersion,
               hat = infl$hat, ...)
hatvalues(model, ...)
# クラス "lm" に対する S3 メソッド
hatvalues(model, infl = lm.influence(model, do.coef=FALSE), ...)
hat(x, intercept = TRUE)
```

引数：

model　R オブジェクト．典型的には lm() や glm() の返り値

infl　lm.influence() や influence() が返す影響力構造 (後者では rstudent() と cooks.distance() に対する glm メソッドだけ)

res　(もしかすると重み付きの) 適当な既定仕様を持つ残差

sd　使われる標準偏差，既定動作を参照せよ

dispersion　(glm オブジェクトに対する) 使用される散布度，既定動作を参照せよ

hat　ハット値 H[i,i]．既定動作を参照せよ

x　X もしくは計画行列

intercept　x に切片項列を付け加えるか？

...　他のメソッドへ (から) 引き渡される追加引き数

主要な高水準関数は influence.measures() であり，クラス "infl" オブジェクトの表形式表示，各モデル変数の DFBETAS，DFFITS，共分散比，Cook の距離とハット行列の対角成分，を作る．これらの測度のどれかに対し影響力が大のケースはアステリスクが付けられる．

関数 dfbetas(), dffits(), covratio() そして cooks.distance() は 対応する診断量への直接のアクセスを提供する．関数 rstandard() と rstudent() はそれぞれ標

準化，スチューデント化残差を与える．(これらの残差は，それぞれ全体と，一つを取り除いた誤差分散を用い，分散が 1 となるように再正規化されている．)

一般化線形モデルに対する値は Williams で議論されている (Cook 距離がカイ 2 乗値ではなく，F 値となるようにスケール化されていることを除き) ような近似である．

オプションの `infl`, `res` そして `sd` 引数は，例えば背景にある基本的な影響力測度が既に (`lm.influence()` や総称的な `influence()` から) 得られているような状況で，これらの直接アクセス関数を使うことを奨励するためにここに置かれている．

`weights == 0` であるような全てのケースはこれら全ての関数から除外されるが，もし線形モデルがオプション `na.action = na.exclude` で当てはめられていれば，当てはめの途中で除外されたケースに対しては適当な値で埋められる．

関数 `hat()` は主に S (第 2 版) との互換性のために用意されている．代わりに `hatvalues()` を使うことを勧める．

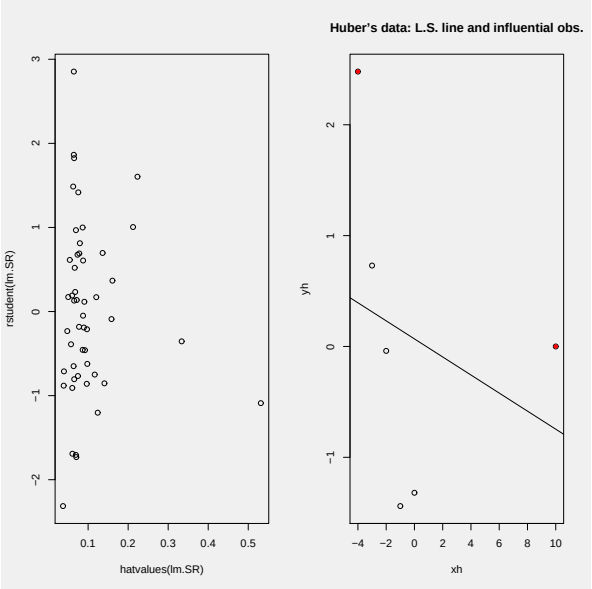
注意：`hatvalues()`, `dfbeta()` そして `dfbetas()` に対しては，線形モデルに対するメソッドが一般化線形モデルに対しても使える．

```
# 家計貯蓄率データ LifeCycleSavings を使う
> lm.SR <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings)
> inflm.SR <- influence.measures(lm.SR)

> which(apply(inflm.SR$is.inf, 1, any))           # どの観測値が影響を持つか
        Chile United States        Zambia         Libya
            7            44            46            49

> summary(inflm.SR)
Potentially influential observations of
         lm(formula = sr ~ pop15 + pop75 + dpi + ddpi, data=LifeCycleSavings) :
               dfb.1_ dfb.pp15 dfb.pp75 dfb.dpi dfb.ddpi dffit   cov.r   cook.d
Chile          -0.20   0.13     0.22    -0.02    0.12    -0.46    0.65*  0.04
United States   0.07  -0.07     0.04    -0.23   -0.03    -0.25    1.66*  0.01
Zambia          0.16  -0.08    -0.34     0.09    0.23     0.75    0.51*  0.10
Libya           0.55  -0.48    -0.38    -0.02   -1.02*   -1.16*   2.09*  0.27
               hat
Chile           0.04
United States   0.33*
Zambia          0.06
Libya           0.53*
```

```
> plot(rstudent(lm.SR) ~ hatvalues(lm.SR))     # 推薦する人がいるプロット（以下の図を参照）
> rs <- rstandard(lm.SR)
> iflSR <- influence(lm.SR)
> identical(rs, rstandard(lm.SR, infl = iflSR))   # infl 引数は不要であるが，再計算を防ぐ
[1] TRUE

> 1000 * round(dfbetas(lm.SR, infl = iflSR), 3)   # 最大値を見る
               (Intercept) pop15 pop75  dpi  ddpi
Australia               12   -10   -27   45     0
Austria                -10     6    41  -37    -8
(以下省略)
```

```
> xh <- c(-4:0, 10)                               # Huber のデータ
> yh <- c(2.48, .73, -.04, -1.44, -1.32, 0)
> summary(lmH <- lm(yh ~ xh))
Call:
lm(formula = yh ~ xh)
Residuals:
      1       2       3       4       5       6
 2.0858  0.4173 -0.2713 -1.5898 -1.3883  0.7463
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.06833    0.63279   0.108    0.919
```

```
xh          -0.08146    0.13595  -0.599    0.581
Residual standard error: 1.55 on 4 degrees of freedom
Multiple R-Squared: 0.08237,    Adjusted R-squared: -0.147
F-statistic: 0.3591 on 1 and 4 DF,  p-value: 0.5813

> (im <- influence.measures(lmH))
Influence measures of lm(formula = yh ~ xh) :
   dfb.1_    dfb.xh   dffit cov.r   cook.d   hat inf
1  1.1124 -9.56e-01  1.4667 0.329  0.52004 0.290   *
2  0.1261 -8.13e-02  0.1500 2.218  0.01464 0.236
3 -0.0775  3.33e-02 -0.0843 2.173  0.00469 0.197
4 -0.5320  1.14e-01 -0.5442 1.000  0.13454 0.174
5 -0.4361  2.57e-17 -0.4361 1.230  0.09627 0.167
6  8.5733  1.84e+01 20.3160 0.255 26.39859 0.936   *
> plot(xh,yh, main = "Huber's data & L.S. line and influential obs.")
> abline(lmH); points(xh[im$is.inf], yh[im$is.inf], pch=20, col=2)
```



回帰診断
(左) 標準化残差のハット値に対するプロット
(右) Huber のデータに対する回帰直線と影響力の大きな観測値 (2 個) のプロット

6.6.7 モデルフレーム model.frame()

総称的関数 model.frame() とそのメソッドは formula を使用するのに必要な変数と任意の追加引数 ... を含むデータフレームを返す.

書式：

```
# 既定の S3 メソッド
model.frame(formula, data = NULL,
            subset = NULL, na.action = na.fail,
            drop.unused.levels = FALSE, xlev = NULL, ...)
# クラス "aovlist" に対する S3 メソッド
model.frame(formula, data = NULL, ...)
# クラス "glm" に対する S3 メソッド
model.frame(formula, ...)
# クラス "lm" に対する S3 メソッド
model.frame(formula, ...)
```

引数：

formula　モデル式，"terms" オブジェクト，または R オブジェクト

data　formula 中の変数を含むデータフレーム，リスト，環境，もしくはデータフレームに変換できるオブジェクト．行列や配列は不可

subset　使われる行の指定．既定では全ての列．これは data の行，もしくはそれが与えられていなければ formula 中で使われている変数からなるデータフレーム，に対する任意の適正な添字ベクトルで良い ([.data.frame() を参照)

na.action　データが NA を含むときそれを処理する関数．既定処理はまず data の任意の "na.action" 属性，次に options の na.action 設定，最後にそれが未設定なら na.fail() .「工場出荷時」の既定動作は na.omit()

drop.unused.levels　因子は使用しない水準を捨て去るか？既定値は FALSE

xlev　各因子に対して仮定される水準の完全なセットを与える文字ベクトルの名前付きリスト

...　data, na.action, subset 等の追加引数．既定のメソッドに影響を及ぼす offset や weights といった任意の追加引数はモデルフレームに更に括弧で囲まれた "(offset)" のような欄を付け足す

返り値：　formula 中で使われる変数と ... で指定された引数を含むデータフレーム

正確に何が起こるかはオブジェクト formula のクラスと属性に依存する．もしこれが "lm" のような当てはめオブジェクトクラスなら，メソッドはモデルを当てはめた (もし存在すれば．しばしば引数 model = TRUE で選択される) 際に使われ保存されたモデルフレームを返すか，当てはめの際に使われた呼び出し式を既定のメソッドに引き渡すか，どちらかである．既定のメソッド自体は，他の追加引数が提供されなければ，クラス "lqs" や "ppr" といったかなり標準的なモデルオブジェクトをうまく処理できる．

この節の残りは既定のメソッドに対してだけ適用される．もし formula か data のどちらかが既にモデルフレーム ("terms" 属性を持ち，それ以外を欠いているデータフレーム) ならば，そのモデルフレームが返される．formula が "terms" オブジェクトでない限り，terms がそれに対して呼び出される．(もし terms.formula の keep.order 引数を使いたければ，モデル式ではなく，"temrs" オブジェクトを引き渡そう．)

formula, subset そして ... 中の全ての変数はまず data 中で探され，それから formula の環境中で探され (より詳しい詳細は formula() のヘルプ文章を参照せよ)，それからデータフレーム中に集められる．それから subset 表現が評価され，それがデータフレームの行添字として使われる．そして na.action 関数がデータフレームに適用される (そして属性が付け加えられるかもしれない)．データフレーム中の任意の因子の水準は drop.unused.levels と xlev 引数に従って補正される．

関連：計画行列については model.matrix()，モデル式については formula()，そしてモデルフレームの操作については expand.model.frame().

```
# 車の速度と停止時間のデータ cars を使用
> data.class(carsframe <- model.frame(dist ~ speed, data = cars))
[1] "data.frame"                                  # オブジェクト carsframe のクラス

# モデルフレームの全内容の簡易一覧 (モデル式 dist~speed を評価するのに必要な全ての情報を含む)
```

```
> str(carsframe)
'data.frame':   50 obs. of  2 variables:
 $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 - attr(*, "terms")=Classes 'terms', 'formula' length 3 dist ~ speed
  .. ..- attr(*, "variables")= language list(dist, speed)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:2] "dist" "speed"
  .. .. .. ..$ : chr "speed"
  .. ..- attr(*, "term.labels")= chr "speed"
  .. ..- attr(*, "order")= int 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=length 165 <environment>
  .. ..- attr(*, "predvars")= language list(dist, speed)
  .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. ..- attr(*, "names")= chr [1:2] "dist" "speed"
```

6.6.8 当てはめモデルの共分散行列を計算する vcov()

当てはめモデルオブジェクトの主要パラメータの分散共分散行列を計算する.

書式： `vcov(object, ...)`

引数：

`object` 当てはめモデルオブジェクト

`...` メソッド関数への追加引数. `glm` メソッドに対してはこれは `dispersion` 引数を引き渡すために使うことができる

返り値： モデルの線形もしくは非線形予測量中のパラメータ間の共分散の推定値

これは総称的関数である. 名前が `vcov.` で始まる関数はこの関数のメソッドである. この関数へのメソッドを持つ関数には `lm()`, `glm()`, `nls()`, `lme()`, `gls()`, `coxph()` そして `survreg()` がある.

6.6.9 モデル式 formula()

総称的関数 `formula()` とその個別のメソッドは, 他のオブジェクトに含まれているモデル式を取り出す. `as.formula()` はほとんど同一であるが `object` が既に`"formula"` 属性を継承している際に, 属性を保存する. `env` 引数の既定値はモデル式が環境を欠いている場合にだけ使われる.

書式：

```
formula(x, ...)
as.formula(object, env = parent.frame())
```

引数：

`x, object` R オブジェクト

`...` 他のメソッドへ (から) 引き渡される追加引数

`env` 結果に付属する環境

例えば `lm()` や `glm()` 関数によるモデル当てはめは簡潔でシンボリックな形式で指定される. チルダ演算子 `~` はそうしたモデルを作る際に基本となる. 形式 `y ~ model` は目

的変数 y が model によりシンボリックに指定された線形予測子によりモデル化されると解釈される．そうしたモデルは + 演算子で区切られた一連の項からなる．項自体は : 演算子で区切られた変数と因子名からなる．そうした項は項中に現れる全ての変数と因子の交互作用と解釈される．+ と : に加えて，以下の演算子がモデル式で有用である．

- * 演算子は因子の掛け合わせを意味する．例えば a*b は a+b+a:b を意味する．
- ^ 演算子は指定の巾乗数だけ掛け合わせることを意味する．例えば，(a+b+c)^2 は (a+b+c)*(a+b+c) と同値で，主効果 a,b,c とそれらの 2 次の交互作用を含むモデル式 a+b+c+a:b+a:c+b:c に展開される．
- %in% 演算子は，その左にある項が，右にある項と入れ子になっていることを意味する．例えば a+b%in%a は式 a+a:b に展開される．
- - 演算子は指定された項を取り除く．したがって (a+b+c)^2-a:b は a+b+c+b:c+a:c と同じになる．これはまた切片項を取り除くのにも使える．
- y ~ x - 1 は原点を通る直線である．切片項を持たぬモデルはまた y~x+0 や y~0+x でも指定可能.

モデル式は通常変数と因子名だけからなるが，数値表現を含んでも良い．式 log(y) ~ a + log(x) は全く正当なモデル式である．そうした算術演算式がモデル式でシンボリックに使われる演算子を含むと，演算子なのか算術演算式なのか混乱が生じる可能性がある．この混乱を防ぐために，「そのまま」関数 I() を使い，モデル式中で演算子が算術演算の意味で使われている箇所を囲むことができる．例えば，式 y ~ a + I(b+c) では項 b+c は b と c の和と解釈される．

R 1.8.0 より変数名はモデル式中で ‘like this‘ のように backtick 文字で引用できるようになったが，モデル式を用いる全てのコードがそうした名前を受け付ける保証は無い．

formula() が当てはめモデルオブジェクトに対して呼び出されると，既定のメソッドのいずれかの特定のメソッド (例えば "nls" のそれのような) が使われる．既定ではまずオブジェクトの "formula" 成分 (そしてそれを評価する)，それから "terms" 成分，そして呼び出し式の formula パラメータ (その値が評価される)，最後に "formula" 属性を探す.

これらの関数は全てシンボリックなモデル式を含む，クラス "formula" のオブジェクトを作り出す．モデル式は関連する環境を持ち，この環境 (親環境ではなく) は提供された data 引数中に無い変数を評価するために model.frame() により使用される．~ 演算子で作られたモデル式は，それが作られた際の環境を使用する．as.formula() で作られたモデル式は環境として env 引数を使う．as.formula() で取り出された既存のモデル式は env が明示的に与えられたときだけ，その環境が変わる.

関連： I()．モデル式の操作は terms(), all.vars()．典型的な使用法は lm(), glm(), coplot() を参照.

```
> class(fo <- y ~ x1*x2)                    # モデル式の例
[1] "formula"                               # そのクラス
> fo                                        # モデル式自体
y ~ x1 * x2
```

```
> typeof(fo)                                          # R の内部保管タイプは"language"
[1] "language"

> terms(fo)                                           # 項の内容
y ~ x1 * x2
attr(,"variables")
list(y, x1, x2)
attr(,"factors")                                      # 項は二値因子化されている
   x1 x2 x1:x2
y   0  0     0
x1  1  0     1
x2  0  1     1
attr(,"term.labels")                                  # 項のラベル
[1] "x1"    "x2"    "x1:x2"
attr(,"order")                                        # それらは一次と二次の交互作用項
[1] 1 1 2
attr(,"intercept")                                    # 切片項あり
[1] 1
attr(,"response")                                     # 目的変数は一つ
[1] 1
attr(,".Environment")
<environment: R_GlobalEnv>

> environment(fo)                                     # 対応する環境は R の大局的環境
<environment: R_GlobalEnv>

> environment(as.formula("y ~ x"))
<environment: R_GlobalEnv>

> environment(as.formula("y ~ x",env=new.env()))  # 新しい環境を指定
<environment: 0x9054544>
```

```
# 多数の変数を持つモデルに対するモデル式を作る．変数名文字列ベクトル c("x1","x2",...,"x25") の作
成
> xnam <- paste("x", 1:25, sep="")
# as.formula の引数に文字列"y ~ x1+x2+...+x25"を指定
> (fmla <- as.formula(paste("y ~ ", paste(xnam, collapse= "+"))))
y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10 + x11 +
    x12 + x13 + x14 + x15 + x16 + x17 + x18 + x19 + x20 + x21 +
    x22 + x23 + x24 + x25
```

6.6.10 モデルの係数を取り出す coef()

coef() はモデリング関数が返すオブジェクトからモデル係数を取り出すための総称的関数である．coefficients() はその別称である．

書式： coef(object, ...), coefficients(object, ...)

引数：

object　モデル係数の取り出しが意味のあるオブジェクト
...　他の引数

返り値：　モデルオブジェクト object から取り出した係数

モデル当てはめ関数が返すオブジェクトのクラスは全て coef メソッドを持つか，既定のメソッドを持つべきである．(メソッドは coef() であり coefficients() ではないことを注意．) クラス "aov" は別名係数 (alias を見よ) を報告しない coef メソッドを持つ．

関連：関連するメソッド fitted.values() と residuals()．モデル当てはめには glm(), lm().

```
> x <- 1:5; coef(lm(c(1:3,7,6) ~ x))
(Intercept)           x                          # 切片項と x 項の係数
       -0.7         1.5
```

6.6.11 モデルパラメータの信頼区間 confint()

当てはめモデルの一つもしくは複数のパラメータに対する信頼区間を計算する．クラス "lm" を継承するオブジェクトに対するメソッドを持つ．

書式： confint(object, parm, level = 0.95, ...)

引数：

object 当てはめモデルオブジェクト

parm どのパラメータの信頼区間を与えるかを指示する．数値ベクトルか名前のベクトル．もし与えられなければ，全てのパラメータが対象になる

level 要求される信頼係数

... メソッドに対する追加引数 (群)

返り値： 返り値は行列 (もしくはベクトル) で，列は各パラメータに対する信頼限界の上端，下端を与える．これらは (1-level)/2 と 1 - (1-level)/2 (既定では 2.5% と 97.5%) というラベルを持つ

confint() は既定のメソッドを持たない総称的関数である．クラス "lm" のオブジェクトに対しては t 値に基づく直接的な公式が使われる．パッケージ MASS は "glm" と "nls" オブジェクトに対するメソッドを持つ．

関連： confint.nls().

```
> data(mtcars)
> fit <- lm(100/mpg ~ disp + hp + wt + am, data=mtcars)

> confint(fit)
                   2.5 %      97.5 %
(Intercept) -0.774822875 2.256118188              # 各パラメータに対する 95% 信頼区間
disp        -0.002867999 0.008273849
hp          -0.001400580 0.011949674
wt           0.380088737 1.622517536
am          -0.614677730 0.926307310

> confint(fit, "wt")                              # パラメータ wt に対する 95% 信頼区間
       2.5 %   97.5 %
wt 0.3800887 1.622518
```

6.6.12 分散分析モデルの結果から表を計算する model.tables()

モデル当てはめ，特に複雑な aov 当てはめ，に対する要約表を計算する．

書式：

```
model.tables(x, ...)
# クラス "aov" に対する S3 メソッド
```

```
model.tables(x, type = "effects", se = FALSE, cterms, ...)
# クラス "aovlist" に対する S3 メソッド
model.tables(x, type = "effects", se = FALSE, ...)
```

引数：

x　モデルオブジェクト．普通 aov() により作られる

type　表のタイプ．現在のところ "effects" と "means" だけが移植されている

se　標準誤差を計算するか？

cterms　表を計算すべき項の名前を与える文字ベクトル．既定は全ての表

...　他のメソッドへ (から) 引き渡される追加引数

返り値：　クラス "tables.aov" のオブジェクトで，多くの成分を含むリスト：

tables　要求された各項に対する表のリスト

n　各項に対する繰りかえし情報

se　標準誤差情報

type = "effects" に対しては，各項の係数に対する表を与える．オプションで標準誤差が付く．type = "means" に対しては，項中の水準の各組合せに対する平均応答の表を与える．

警告：この移植は不完全であり，簡単な場合だけが完全にテストされている．重みつき aov 当てはめはサポートされていない．

関連：aov(), proj(), replications(), TukeyHSD(), se.contrast().

```
> example(model.tables)
> N <- c(0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0,
         0, 1, 0, 1, 0, 1, 1, 0, 0)
> P <- c(1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
         1, 1, 0, 0, 1, 0, 1, 1, 0)
> K <- c(1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
         0, 0, 0, 1, 1, 1, 0, 1, 0)
> yield <- c(49.5,62.8,46.8,57,59.8,58.5,55.5,
             56,62.8,55.8,69.5,55,62,48.8,45.5,44.2,52,51.5,
             49.8,48.8,57.2,59,53.2,56)
> npk <- data.frame(block = gl(6, 4), N = factor(N),
                    P = factor(P), K = factor(K), yield = yield)

> npk.aov <- aov(yield ~ block + N * P * K, npk)

> model.tables(npk.aov, "means", se = TRUE)
Design is unbalanced - use se.contrasts for se's
Tables of means
Grand mean
54.875
block
        1     2     3     4     5     6
    54.03 57.45 60.77 50.12 50.52 56.35
rep  4.00  4.00  4.00  4.00  4.00  4.00
N
        0     1
    52.07 57.68
rep 12.00 12.00

(途中省略)

P:K
     K
P     0     1
  0   57.60 53.33
```

```
  rep  6.00  6.00
  1   56.13 52.43
  rep  6.00  6.00
```

```
> options(contrasts = c("contr.helmert", "contr.treatment"))
> npk.aovE <- aov(yield ~ N * P * K + Error(block), npk)

> model.tables(npk.aovE, se = TRUE)
Standard error information not returned as design is unbalanced.
Standard errors can be obtained through se.contrast.
Tables of effects
 N
         0      1
    -2.808  2.808
rep 12.000 12.000

(途中省略)

 N:P:K
, , K = 0
     P
N     0       1
  0   -1.2417  1.2417
  rep  3.0000  3.0000
  1    1.2417 -1.2417
  rep  3.0000  3.0000
, , K = 1
     P
N     0       1
  0    1.2417 -1.2417
  rep  3.0000  3.0000
  1   -1.2417  1.2417
  rep  3.0000  3.0000
```

```
> model.tables(npk.aovE, "means")
Tables of means
Grand mean
54.875
 N
        0     1
    52.07 57.68
rep 12.00 12.00

(途中省略)

 N:P:K
 , , K = 0
     P
N     0     1
  0   51.43 54.33
  rep  3.00  3.00
  1   63.77 57.93
  rep  3.00  3.00
, , K = 1
     P
N     0     1
  0   52.00 50.50
  rep  3.00  3.00
  1   54.67 54.37
  rep  3.00  3.00
```

6.6.13　プロファイリングモデルに対する総称的関数 profile()

fitted で表される解の近くでの目的関数の挙動を調べる．これ以上の詳細はメソッド関数のドキュメントを見よ．

書式： profile(fitted, ...)

引数：
`fitted` 元の当てはめモデルオブジェクト
`...` 追加パラメータ．個別のメソッドのドキュメントを見よ

返り値： プロファイルされる各パラメータに対し一つの要素を持つリスト．詳細は個別のメソッド関数のドキュメントを見よ

関連： `profile.nls()`, パッケージ MASS 中の `profile.glm()` 等．プロファイル用のコードに付いては `Rprof()` を見よ．

6.6.14 モデル項の対比の標準誤差 `se.contrast()`

aov オブジェクト中の一つ，もしくは複数の対比の標準誤差を返す．

書式：
```
se.contrast(object, ...)
# クラス "aov" に対する S3 メソッド
se.contrast(object, contrast.obj,
            coef = contr.helmert(ncol(contrast))[, 1],
            data = NULL, ...)
```

引数：
`object` 適当な当てはめ，普通 `aov()` の結果
`contrast.obj` 標準誤差を計算したい対比．これはリストや行列で指定できる．単一の対比は，対比されるセルを与える論理値ベクトルのリストで指定できる．複数の対比は，各列が数値の (総和が 0 の) 対比ベクトルである行列で指定されるべきである
`coef` `contrast.obj` がリストのとき使われる．これは総和が 0 のリストである同じ長さのベクトルでなければならない．既定値は第一 Helmert 対比で，これはリストで与えられる第一と第二のセルの平均を対比させる
`data` `contrast.obj` を評価するのに使われるデータフレーム
`...` その他のメソッドへ (から) 引き渡される追加引数

返り値： 各対比に対する標準誤差を与えるベクトル

対比は普通，ある平均が有意に異なるかどうかを検定するために使われる．それらを係数から直接計算するよりも `se.contrast()` を使う方が簡単である．多層モデルでは，対比は一つ以上の層の中で現れることがある．対比と標準誤差は最下位の層の中で計算され，層間で効率性と比較のために補整される．`coef` と一緒に使うための適当な行列は `contrasts()` を呼出し，因子で列を添字操作することにより見出すことができる．

関連： `contrasts()`, `model.tables()`.

```
# Venables & Ripley の例.
# 肥料 (窒素 N, リン酸 P, カリ K) 有無と対応収量 yield
> N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
> P <- c(1,1,0,0,0,1,0,1,1,1,0,0,0,1,0,1,1,0,0,1,0,1,1,0)
```

```
> K <- c(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
> yield <- c(49.5,62.8,46.8,57.0,59.8,58.5,55.5,56.0,62.8,55.8,69.5,
             55.0, 62.0,48.8,45.5,44.2,52.0,51.5,49.8,48.8,57.2,59.0,53.2,56.0)
# データをデータフレームにまとめる．N,P,K は因子であると指定
# block 因子は 4 つずつの 6 グループの意味
> npk <- data.frame(block = gl(6,4), N = factor(N), P = factor(P),
                    K = factor(K), yield = yield)
> options(contrasts=c("contr.treatment", "contr.poly"))

> npk.aov1 <- aov(yield ~ block + N + K, npk)
> se.contrast(npk.aov1, list(N=="0", N=="1"), data=npk)
[1] 1.609175

> cont <- matrix(c(-1,1), 2, 1, dimnames=list(NULL, "N"))    # 行列を使うと
> se.contrast(npk.aov1, cont[N,,drop=FALSE]/12, data=npk)
       N
1.609175

> npk.aov2 <- aov(yield ~ N + K + Error(block/(N + K)), npk) # 多層モデルの検定
> se.contrast(npk.aov2, list(N == "0", N == "1"))
[1] 1.812166
```

6.7　その他の補助関数

線形モデル用の補助関数はその他にも多数存在するが，一般ユーザが直接利用することは少ないと思われるが参考までに以下にまとめる．これ以外にも総称的関数の各当てはめクラス用のメソッド関数も存在する．

`add1()`, `drop1()`　モデルに一つの項を加える，取り去る

`I()`　オブジェクトが，文字通りの意味で解釈されるべきことを指示する様にクラスを変更する

`alias()`　モデル式指定された線形モデル中のエイリアス (別名，alias)，つまり線形従属項，を見付ける

`case.names()`, `variable.names()`　当てはめモデルの (欠けていない) ケース名や，(除去されていない) 変数名を返す簡単なユーティリティ

`contr.helmert()`, `contr.poly()`, `contr.sum()`, `contr.treatment()`　対比行列 (contrast matrix) を返す

`contrasts()`　ある因子に関連する対比を設定したり，閲覧したりする

`C()`　因子に対比属性 `"contrasts"` を設定する

`dummy.coef()`　はコード化されたものではなく，本来の名前で係数を取り出す

`model.matrix()`　計画行列 (design matrix) を作る

`proj()`　データを線形モデルの項の空間へ射影した行列もしくは行列のリストを返す．`aov()` モデルで最もしばしば使われる

`gl()`　パターンを与えて因子の水準を作成する

`offset()`　モデル式にオフセット項を含める．オフセットとは一般化線形モデルに於けるような線形予測子に加えられる項で，係数は既知の係数 1 を持ち，推定されることは無い

`.checkMFClasses()`　予測メソッドで使われる変数が，当てはめで使われたそれらとタイプが一致するかどうかを検査する．`.MFclass()` はこのために変数を類別する

`expand.model.frame()`　新しい変数を，それが指定モデルのモデル式の一部であるか

のように評価する
is.empty.model() R のモデル式表記は切片項も説明変数もない空のモデルを許す．この関数はモデルが空かどうか検査する
model.extract() モデルフレームから成分を取り出す
proj() データの線形モデル項の上への射影を与える行列もしくは行列のリストを返す
update() モデルの更新と再当てはめ
update.formula() モデル式の更新
asOneSidedFormula() 片側モデル式への変換
delete.response() 同じモデルに対する目的変数を持たない terms オブジェクトを返す
drop.terms() モデルの右辺から変数を取り除く
reformulate() は文字列からモデル式を作る
factor.scope() ubsection モデル式に追加・除去可能な項を計算 se.contrast()
add.scope(), drop.scope() モデルから，項の階層性を保ちながら，個別に追加・除去可能な項を計算する
terms.formula() モデル式から term オブジェクトを作る
makepredictcall() poly や ns といった項を持つモデルから予測を行う際の正しい行列を作る
poly(), polym() 直交多項式を返す
factor() 因子へのコード化

6.8 その他，線形混合効果モデル (パッケージ nlme)

R の推奨パッケージ nlme には線形混合効果モデル (linear mixed effects model) 関連の豊富な関数群がある．$\boldsymbol{y}$ を観測値全てを適当に並べたベクトルとし，計画行列 X, Z とパラメータベクトル $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ を持つ線形モデル

$$\boldsymbol{y} = X\boldsymbol{\alpha} + Z\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

を考える．主として興味のある項は $X\boldsymbol{\alpha}$ であるが，項 $Z\boldsymbol{\beta}$ (普通グルーピング変数に関連する) が無視できないとすると，その分モデルパラメータが増加し推定が困難になる．線形混合効果モデルは $\boldsymbol{\beta}$ がランダムであると仮定し，追加パラメータを $\boldsymbol{\beta}$ の分布パラメータに限ることにより，モデルの複雑化を避ける工夫である．モデルを当てはめた結果から，$\boldsymbol{\beta}$ の実現値自体も事後的に予測できる．$\boldsymbol{\alpha}$ を固定効果，$\boldsymbol{\beta}$ をランダム (変量) 効果と呼ぶ．線形混合効果モデルは実際は $\boldsymbol{\beta}$ の値で条件を付けた条件付きモデルであり，形式的に

$$\boldsymbol{E}\{\boldsymbol{y} \mid \boldsymbol{\beta}\} = X\boldsymbol{\alpha} + Z\boldsymbol{\beta}$$

と表現される．従って，$\boldsymbol{y}$ の分布は誤差分布を $\boldsymbol{\beta}$ の分布で混合した混合分布になる．パラメータの推定は $\boldsymbol{y}$ の正規性を仮定した上で，制約付き最尤法 (REML, restricted maximum likelihood method) を使うのが基本である．

以下では，パッケージ nlme の中心的関数である lme() のみ[*4] を取り上げる．

[*4] 非線形混合効果モデル用関数 nlme() に付いては 286 頁を参照．

書式：

```
lme(fixed, data, random, correlation, weights, subset, method,
    na.action, control, contrasts = NULL, keep.data = TRUE)
# クラス'lme' に対する S3 メソッド
update(object, fixed., ..., evaluate = TRUE)
```

引数：

fixed モデルの固定効果部分を指定する両側線形モデル式オブジェクト．~ 演算子の左には応答変数，右側には + 演算子を挟んだ項， lmList オブジェクト，又は groupedData オブジェクトが置かれる．メソッド関数 lme.lmList() と lme.groupedData() は別々のドキュメントを持つ

fixed． 固定効果公式への変更．詳細は update.formula() を参照

data オプションのデータフレームで，引数 model, correlation, weights そして subset 中に名前がある変数を含む．既定では，変数は lme() が呼び出された環境中から探される

random オプションで以下のどれか；(1) x1+...+xn をランダム効果に対するモデルとして，片側公式 ~ x1+...+xn | g1/.../gm を持つ． g1/.../gm はグルーピング構造 (m は 1 でも良く，その時は / は不要)．多重グルーピング水準の場合はランダム効果公式はグルーピングの全ての水準に渡って繰り返される．(2) 形式 ~ x1+...+xn | ga の片側公式のリストで，各グルーピング水準に対して異なったランダム効果モデルを持っても良い．ネスティングの順序はリスト中の成分のそれと同じとされる．(3) 形式 ~ x1+...+xn の片側公式，又は公式を持つ (つまり formula(object) が NULL にならない) pdMat オブジェクト，もしくはそうした公式や pdMat オブジェクトを成分に持つリスト．この場合，グルーピング構造公式は線形混合効果モデルを当てはめるのに使われたデータから導出され，クラス groupedData を継承している必要がある．(4) (3) におけるような公式や pdMat オブジェクトの名前付きリストでグルーピング因子を名前に持つ．(5) reStruct オブジェクト．利用可能な pdMat クラスに付いては pdClasses のドキュメントを参照．既定では， fixed の右側からなる公式

correlation オプションの corStruct オブジェクトで級内相関構造を記述する．利用可能な corStruct クラスに付いては corClasses のドキュメントを見よ．既定値は NULL で，級内相関無しに対応する

weights オプションの varFunc オブジェクトか，級内異分散構造を記述する片側公式．公式で与えられた時は，固定分散重みに対応する varFixed への引数として使われる．利用可能な varFunc クラスの説明は varClasses のドキュメントを見よ．既定値は NULL で，等分散誤差に対応する

subset data の行のどの部分集合が当てはめに際して使われるべきかを指定すオプションの表現式．論理ベクトルや，どの番号の観測値を使うかを表す数ベクトルでも，行ラベルの文字列ベクトルでも良い．既定では全ての観測値が使われる

method 文字列．もし既定の "REML" なら，当てはめは制約付き対数尤度を最大化することにより行われる．もし "ML" なら対数尤度を最大化する

na.action データ中の NA 値を処理する関数．既定である na.fail() はエラーメッ

セージを表示し，不完全な観測値があれば停止する

control 推定アルゴリズムの制御値のリストで，関数 glsControl() が返す既定値を置き換える．既定では空リスト．

contrasts オプションリスト． model.matrix.default() の contrasts.arg 引数を見よ

keep.data 論理値． data 引数 (もしデータフレームとして与えられたとき) をモデルオブジェクトの一部として保存するか

verbose もし TRUE ならば反復アルゴリズムの中途経過が出力される．既定値は FALSE

... この総称的関数へのメソッドの幾つかは追加引数を必要とする．このメソッドではどれも使われない

evaluate もし TRUE ならば新しい呼び出しが評価される．さもなければ呼び出し式を返す

返り値： 返り値は線形混合効果モデルの当てはめ結果を表すクラス lme のオブジェクトである．print(), plot() そして summary() 等の総称的関数はこのオブジェクトに対するメソッド関数を持つ．当てはめの各成分に付いてはドキュメント lmeObject を参照．関数 resid(), coef(), fitted(), fixed.effects() そして random.effects() は成分の一部を取り出すのに使える

パッケージ nlme の組み込みデータ Orthodont は歯列矯正に伴うデータフレームで 108 行 4 列からなる．変数 distance は頭骸骨の X 線写真から測定された脳下垂体から翼状顎裂までの距離 (mm)，age は年齢，Sex は性別，そして Subject は男性は M01 から M16，女性は F01 から F11 までの順序付き因子である．年齢 8 歳から 14 歳の男児 16 人，女児 11 人が対象で，一人に付き年齢 8,10,12,14 歳で測定されている longitudinal data[*5] である．

```
# 推奨パッケージ nlme 中のデータセット歯列矯正データ Orthodont の中身
> library(nlme)
> class(Orthodont)                                  # 既に線形混合効果モデル用に整形されている
[1] "nfnGroupedData" "nfGroupedData"  "groupedData"    "data.frame"
> Orthodont
Grouped Data: distance ~ age | Subject
    distance age Subject    Sex
1       26.0   8     M01   Male
2       25.0  10     M01   Male
3       29.0  12     M01   Male
4       31.0  14     M01   Male
(途中省略)
107     28.0  12     F11 Female
108     28.0  14     F11 Female
```

この追加属性付きのデータフレームはパッケージ nlme の関数 groupedData() を用いて次のように作られている．

```
# 先ず Orthodont を属性無しのただのデータフレームにする
> temp.df <- data.frame(distance=Orthodont$distance, age=Orthodont$age,
                        Subject=Orthodont$Subject, Sex=Orthodont$Sex)
```

[*5] 経時観測データとか縦断データとか訳される．比較的短めの時系列データであることが多い．

```
> orthodont <- groupedData( distance ~ age | Subject, # 固定，ランダム効果指定モデル式
                 data = temp.df,                          # 対象データフレーム
                 FUN = function (x) max(x, na.rm = TRUE),   # 要約用補助関数
                 outer = ~ Sex,
                 labels = list(x = "Age",                 # プロット用軸ラベル
                   y = "Distance from pituitary to pterygomaxillary fissure"),
                 units = list( x = "(yr)", y = "(mm)") )    # 変数の単位
> all.equal(orthodont, Orthodont)                 # 確かに一致
[1] TRUE
```

以下の実行例 (example(nlme) の第一の例) では，distance を，固定効果項である age 項と (既定で指定される) 切片 (定数) 項で説明 (distance~1+age) しようとする．一方で，関数 lme のランダム項指定引数 random の省略時既定値として，同じくランダム項 ~1+age が暗黙のうちに指定される．ランダム項はデータセット Orthodont の groupedData クラス属性で暗黙のうちに指定されている Subject 変数でグルーピングされる (つまり個人毎に年齢 8,10,12,14 歳の変化を考える)．つまり d_{it} を個人 i の年齢 $t = 8, 10, 12, 14$ の距離データとすると

$$d_{it} = \underbrace{a + bt}_{\text{固定効果}} + \underbrace{\alpha_i + \beta_i t}_{\text{ランダム効果}} + \underbrace{r_{it}}_{\text{残差}}$$

であり，α_i, β_i は個人 (グループ) 毎に決まるランダムな係数と考える．$\{\alpha_i\}$, $\{\beta_i\}$ の (級内) 相関引数 corelation としては省略時既定値である NULL(つまり無相関) が指定され，同じく (級内) 分散構造引数 weights の省略時既定値である NULL(つまり等分散) が指定される．

```
# 距離 distance を変数 age と切片 (定数) 項を固定効果およびランダム効果として説明
# 固定効果は実際は切片項も含む~age+1 で，ランダム効果も~age+1 とされる
> fm1 <- lme(distance ~ age, data = Orthodont)

> summary(fm1)                                          # 当てはめ結果の要約
Linear mixed-effects model fit by REML                  # 既定の制約付き最尤推定法を使用
 Data: Orthodont                                        # 当てはめ結果の AIC,BIC と対数尤度
       AIC      BIC    logLik
  454.6367 470.6173 -221.3183

Random effects:                  # 個人毎にグルーピング，変数 age と切片項をランダム効果とする
 Formula: ~age | Subject
 Structure: General positive-definite
            StdDev    Corr
(Intercept) 2.3270339 (Intr)
age         0.2264276 -0.609
Residual    1.3100399

Fixed effects: distance ~ age               # 変数 distance を固定効果項 age と切片項で説明
               Value Std.Error DF   t-value p-value
(Intercept) 16.761111 0.7752461 80 21.620375       0
age          0.660185 0.0712533 80  9.265334       0
 Correlation:
    (Intr)
age -0.848

Standardized Within-Group Residuals:
         Min           Q1          Med           Q3          Max
-3.223106011 -0.493760868  0.007316632  0.472151090  3.916032745

Number of Observations: 108                         # 27 人の 4 つの年齢でのデータ
Number of Groups: 27
```

おなじ当てはめをより明示的に記述すると次のようになる．

```
> fm1b <- lme(fixed=distance~age+1, data=Orthodont, random=~age+1|Subject,
              correlation=NULL, weights=NULL, method="REML", keep.data=TRUE)

> summary(fm1b)
Linear mixed-effects model fit by REML
 Data: Orthodont
       AIC      BIC    logLik
  454.6367 470.6173 -221.3183

Random effects:
 Formula: ~age + 1 | Subject
 Structure: General positive-definite, Log-Cholesky parametrization
            StdDev    Corr
(Intercept) 2.3270340 (Intr)
age         0.2264278 -0.609
Residual    1.3100397

Fixed effects: distance ~ age + 1
                Value Std.Error DF   t-value p-value
(Intercept) 16.761111 0.7752460 80 21.620377       0
age          0.660185 0.0712533 80  9.265333       0
 Correlation:
    (Intr)
age -0.848

Standardized Within-Group Residuals:
         Min           Q1          Med           Q3          Max
-3.223106083 -0.493761144  0.007316631  0.472151121  3.916033211

Number of Observations: 108
Number of Groups: 27
```

次はデータフレームを新に作成して解析する例である．

```
# Orthodont データの 4 成分を取り出し，「唯の」データフレームにする
> distance <- Orthodont$distance
> age <- Orthodont$age
> Subject <- Orthodont$Subject
> Sex <- Orthodont$Sex
> orthodont <- data.frame(distance=distance, age=age, Subject=Subject, Sex=Sex)

> str(orthodont)
'data.frame': 108 obs. of  4 variables:
 $ distance: num  26 25 29 31 21.5 22.5 23 26.5 23 22.5 ...
 $ age     : num  8 10 12 14 8 10 12 14 8 10 ...
 $ Subject : Ord.factor w/ 27 levels "M16"<"M05"<"M02"<..: 15 15 15 15 3 3 3 3 7 7 ...
 $ Sex     : Factor w/ 2 levels "Male","Female": 1 1 1 1 1 1 1 1 1 1 ...

> fmm <- lme(fixed=distance ~ 1+age, data=orthodont, random=~ 1+age | Subject)
> summary(fmm)                                    # summary(fm1) と同じ内容
Linear mixed-effects model fit by REML
 Data: orthodont
       AIC      BIC    logLik
  454.6367 470.6173 -221.3183

Random effects:
 Formula: ~1 + age | Subject
 Structure: General positive-definite, Log-Cholesky parametrization
            StdDev    Corr
(Intercept) 2.3270340 (Intr)
age         0.2264278 -0.609
Residual    1.3100397

Fixed effects: distance ~ 1 + age
                Value Std.Error DF   t-value p-value
(Intercept) 16.761111 0.7752460 80 21.620377       0
age          0.660185 0.0712533 80  9.265333       0
 Correlation:
    (Intr)
age -0.848
```

```
Standardized Within-Group Residuals:
         Min           Q1          Med           Q3          Max
-3.223106083 -0.493761144  0.007316631  0.472151121  3.916033211

Number of Observations: 108
Number of Groups: 27
```

lme() 関数による当てはめ結果 fm1 から情報を取り出す関数を以下に紹介する:

summary(fm1)　当てはめ結果の要約
str(fm1)　当てはめ結果の全内容の簡易出力
plot(fm1)　当てはめ値と標準化残差のプロット図 (以下の図を参照)
formula(fm1)　当てはめに使われたモデル式 distance~age．実際は切片 (定数) 項も含み，更に固定効果，ランダム効果双方を指定
getGroups(fm1)　グルーピング構造 (順序付き因子)．各グループは個人 Subject 毎の 4 種類の年齢
fixed.effects(fm1)　固定効果としての切片項と age 項の係数 (各個人に共通)
random.effects(fm1)　個人毎のランダム効果としての切片項と age 項の係数
coeff(fm1)　固定効果とランダム効果を併せた切片項と age 項の係数
fitted(fm1)　各個人，各年齢毎の予測値
residuals(fm1)　各個人，各年齢毎の実際の distance 値と当てはめによる予測値との差
predict(fm1)　各個人の指定年齢での distance の予測値を与え，既定では fitted(fm1) と同じ結果
intervals(fm1)　固定効果，ランダム効果としての切片項と age 項係数の近似 95% 信頼区間

```
> formula(fm1)                                    # 指定モデル式
distance ~ age

> getGroups(fm1)                                  # グルーピング (順序付き因子). 被験者
  [1] M01 M01 M01 M01 M02 M02 M02 M02 M03 M03 M03 M03 M04 M04 M04 M04 M05 M05
 [19] M05 M05 M06 M06 M06 M06 M07 M07 M07 M07 M08 M08 M08 M08 M09 M09 M09 M09
 [37] M10 M10 M10 M10 M11 M11 M11 M11 M12 M12 M12 M12 M13 M13 M13 M13 M14 M14
 (途中省略)
 [91] F07 F07 F08 F08 F08 F08 F09 F09 F09 F09 F10 F10 F10 F10 F11 F11 F11 F11
attr(,"label")
[1] Subject
27 Levels: M16 < M05 < M02 < M11 < M07 < M08 < M03 < M12 < M13 < ... < F11

> fixed.effects(fm1)                              # 固定効果項係数
(Intercept)         age
 16.7611111   0.6601852

> random.effects(fm1)                             # ランダム効果項係数
    (Intercept)          age                      # 各グループ毎に与えられる
M16  -0.1877576 -0.068853674
M05  -1.1766673  0.025600295
(以下省略)

> coef(fm1)                                       # 両効果の和
    (Intercept)       age
M16    16.57335 0.5913315
M05    15.58444 0.6857855
(以下省略)
```

```
> fitted(fm1)                                      # 各データに対する当てはめ予測値
     M01      M01      M01      M01      M02      M02      M02      M02
24.81965 26.57139 28.32313 30.07487 21.43115 22.78054 24.12993 25.47931
(以下省略)

> residuals(fm1)                                   # その予測残差値
        M01          M01          M01          M01          M02          M02
 1.180347948 -1.571391542  0.676868968  0.925129479  0.068846228 -0.280539756
(以下省略)

> predict(fm1)                                  # 予測位置を未指定なので fitted(fm1) と同じ
     M01      M01      M01      M01      M02      M02      M02      M02
24.81965 26.57139 28.32313 30.07487 21.43115 22.78054 24.12993 25.47931
(以下省略)

> intervals(fm1)                                   # 95% 近似信頼区間
Approximate 95% confidence intervals

 Fixed effects:                                   # 固定項の推定値と近似信頼区間
                lower       est.      upper
(Intercept) 15.2183222 16.7611111 18.3039000      # 切片項
age          0.5183867  0.6601852  0.8019837      # 年齢項
attr(,"label")
[1] "Fixed effects:"

 Random Effects:               # ランダム固定項の標準偏差，相関値の推定値と近似信頼区間
  Level: Subject               # Subject 変数でグルーピングされているという意味
                        lower       est.     upper
sd((Intercept))       0.9481090  2.3270339 5.7114603 # 切片項の標準偏差
sd(age)               0.1024797  0.2264276 0.5002890 # 年齢項の標準偏差
cor((Intercept),age) -0.9382876 -0.6093328 0.2984514 # 両者の相関係数

 Within-group standard error:                     # グループ内変動の標準誤差
   lower     est.    upper
1.084822 1.310040 1.582014
```

当てはめ結果から新しいデータで予測するためには，元データと同一の形式のデータフレームを predict() 関数の引数として与える．固定効果だけによる予測値と，ランダム効果も含めた予測値の双方が与えられる．元データに無いケースでは固定効果だけによる予測値が与えられる (つまりランダム効果の予測値は 0 になる)．

```
> newOrth <- data.frame(Sex = c("Male","Male","Female","Female","Male","Male"),
                        age = c(15, 20, 10, 12, 2, 4),
                        Subject = c("M01","M01","F30","F30","M04","M04"))

> predict(fm1, newOrth, level=0)    # データフレーム newOrth に対する予測 (グルーピング無し)
[1] 26.66389 29.96481 23.36296 24.68333 18.08148 19.40185
attr(,"label")
[1] "Predicted values"

> predict(fm1, newOrth, level=0:1)  # データフレーム newOrth に対する予測 (グルーピングあり)
  Subject predict.fixed predict.Subject
1     M01      26.66389        30.95074    # M01 の 15 歳でのランダム効果を含む予測値
2     M01      29.96481        35.33009
3     F30      23.36296              NA    # 元データに無いケース
4     F30      24.68333              NA
5     M04      18.08148        20.95016
6     M04      19.40185        22.13877
```

次の実行例 (example(nlme) の第二の例) では，distance を，固定効果項である age 項，性別項 Sex と (既定で指定される) 切片 (定数) 項で説明 (distance~1+age+Sex) しようとする．一方で，関数 lme のランダム項指定引数 random でランダム切片項 ~1 が明示的に指定される．ランダム項はデータセット Orthodont の groupedData クラス属性で暗黙のうちに指定されている Subject 変数でグルーピングされる (つまり個人毎に年齢

8,10,12,14 歳の変化を考える)．つまり d_{ijt} を個人 i，性別 $j = 0, 1$，年齢 $t = 8, 10, 12, 14$ の距離データとすると

$$d_{ijt} = \underbrace{a + bt + cj}_{\text{固定効果}} + \underbrace{\alpha_i}_{\text{ランダム効果}} + \underbrace{r_{ijt}}_{\text{残差}}$$

であり，α_i は個人 (グループ) 毎に決まるランダムな切片項である．性別は男児 0，女児 1 と因子化されており，c は男児を基準とした女児の差分である．$\{\alpha_i\}$ の (級内) 相関引数 `corelation` としては省略時既定値である `NULL` (つまり無相関) が指定され，同じく (級内) 分散構造引数 `weights` の省略時既定値である `NULL` (つまり等分散) が指定される．

```
# 距離 distance を変数 age,Sex と切片（定数）項を固定効果およびランダム切片項で説明
> fm2 <- lme(distance ~ age + Sex, data = Orthodont, random = ~ 1)

> summary(fm2)
Linear mixed-effects model fit by REML
 Data: Orthodont
       AIC      BIC    logLik
  447.5125 460.7823 -218.7563

Random effects:
 Formula: ~1 | Subject                        # 個人でグルーピング，切片項をランダム効果とする
        (Intercept) Residual
StdDev:    1.807425 1.431592

Fixed effects: distance ~ age + Sex           # 変数 age,Sex と切片（定数）項を固定効果とする
               Value Std.Error DF   t-value p-value
(Intercept) 17.706713 0.8339225 80 21.233044  0.0000
age          0.660185 0.0616059 80 10.716263  0.0000
SexFemale   -2.321023 0.7614168 25 -3.048294  0.0054
 Correlation:
          (Intr) age
age       -0.813
SexFemale -0.372  0.000

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-3.74889609 -0.55034466 -0.02516628  0.45341781  3.65746539

Number of Observations: 108
Number of Groups: 27
```

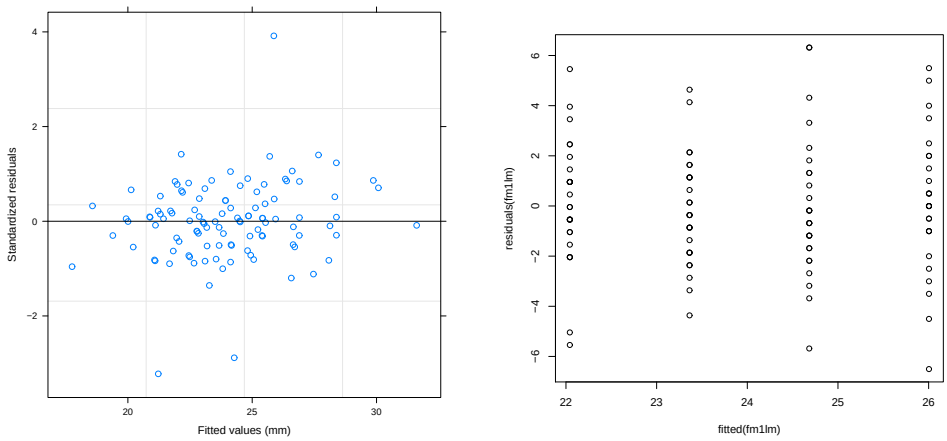
おなじ当てはめをより明示的に記述すると次のようになる．

```
> fm2b <- lme(fixed=distance~1+age+Sex, data=Orthodont, random=~1|Subject,
              correlation=NULL, weights=NULL, method="REML", keep.data=TRUE)
> summary(fm2b)
Linear mixed-effects model fit by REML
 Data: Orthodont
       AIC      BIC    logLik
  447.5125 460.7823 -218.7563

Random effects:
 Formula: ~1 | Subject
        (Intercept) Residual
StdDev:    1.807425 1.431592

Fixed effects: distance ~ 1 + age + Sex
               Value Std.Error DF   t-value p-value
(Intercept) 17.706713 0.8339225 80 21.233044  0.0000
age          0.660185 0.0616059 80 10.716263  0.0000
SexFemale   -2.321023 0.7614168 25 -3.048294  0.0054
 Correlation:
          (Intr) age
age       -0.813
```

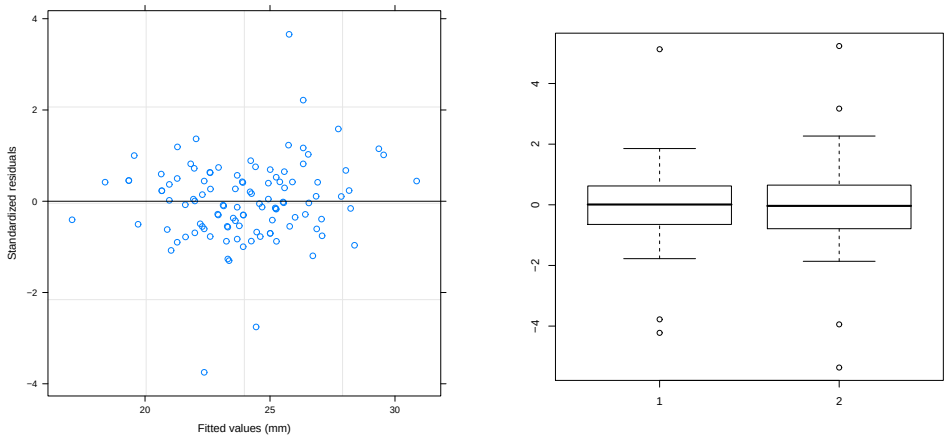
```
SexFemale -0.372  0.000

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-3.74889609 -0.55034466 -0.02516628  0.45341781  3.65746539

Number of Observations: 108
Number of Groups: 27
```

残差の比較では fm2 よりも fm1 の方が当てはまりが良い (以下の図を参照).

```
> fivenum(residuals(fm1))
         M13          M11          M07          M07          M09
-4.222396916 -0.647201919  0.009585077  0.621405059  5.130158960

> fivenum(residuals(fm2))
        M13         F05         M14         F06         M09
-5.36689013 -0.78800154 -0.03602786  0.64910937  5.23599866
```



(左) 線形混合効果モデルの当てはめ結果 fm1．当てはめ値と標準化予測誤差のプロット．(右) 線形モデルの当てはめ結果．当てはめ値と予測誤差のプロット．



(左) 線形混合効果モデルの当てはめ結果 fm2．当てはめ値と標準化予測誤差のプロット．(右) 線形混合効果モデルの当てはめ結果．fm1 (右) と fm2 (左) の当てはめ残差の平行箱型図

第 7 章
非線形回帰モデル

R の非線形回帰モデル用の関数を紹介する．非線形最小自乗法を用いて当てはめを行う nls() 関数が中心であり，予測・プロット用のメソッド関数を持つ．非線形回帰モデルの当てはめでは，適切な初期値を与えることが重要になるが，漸近回帰モデルと呼ばれる関数形が漸近的特性値を持つ場合は，その特徴を用いて適当なモデルパラメータの初期値を与えることが可能になる．特に特定の分野で良く使われる漸近回帰モデルに対しては自己開始 (SS, self-start) 型と呼ばれる，適切なモデルパラメータの初期値自体をデータから推定する関数が数多く用意されている．これらのパラメータの初期推定値はしばしばパラメータの最終推定値と一致する．

7.1 非線形回帰モデル

7.1.1 非線形モデルの当てはめ nls()

非線形モデルパラメータを非線形最小自乗法で決定し，クラス "nls" のオブジェクトを返す．

書式：

```
nls(formula, data = parent.frame(), start, control = nls.control(),
    algorithm = "default", trace = FALSE, subset,
    weights, na.action, model = FALSE)
```

引数：

formula　変数とパラメータを含む非線形モデル式

data　その中で formula 中の変数を評価するオプションのデータフレーム

start　初期推定値の名前付きリストまたは名前付き数値ベクトル

control　オプションの制御設定リスト．設定可能な制御変数とそれらの効果については nls.control() を見よ

algorithm　使用されるアルゴリズムを指定する文字列．既定の手法は Gauss-Newton 法．もう一つの選択肢は部分的線形最小自乗法モデルに対する Golub-Pereyra 法である "plinear"

trace　繰り返しの過程を出力するかどうかを指定する論理値．既定値は FALSE．もし TRUE なら各反復毎に残差平方和とパラメータ値が出力される．もし"plinear" アルゴリズムが使われると，線形パラメータの条件付き推定値が非線形パラメータの後に出力される

subset オプションのベクトル．当てはめ過程で使われる観測値の部分集合を指定
weights オプションの重み数値ベクトル．もし指定されると，目的関数は重みつき最小自乗和法となる．まだ移植されていない
na.action データが NA を含むときそれを処理する関数
model 論理値．もし真ならモデルフレームが一部として返される

返り値： 次の成分を持つリスト：
m モデルを含む nlsModel オブジェクト
data nls にデータ変数として引き渡された表現．実際のデータ値は m 成分の環境中に存在する

nls() 関数を人工的な「誤差が無い」モデルに適用してはならない．nls() 関数は，現在の推定パラメータに於ける数値的な不正確さを残差平方和と比較する「relative-offset」収束規準を使う．これは $\mathrm{var}(\varepsilon) > 0$ である $y = f(x,\theta) + \varepsilon$ の形のモデルに対して最も良く動作する．この規準は $y = f(x,\theta)$ の形のモデルに対しては，二つの丸め誤差からなる成分を比較することになり，失敗する．もし nls() を人工データでテストしたければ，以下の例のように，ノイズ項を加えること．nls() が返すオブジェクトは当てはめモデルオブジェクトの一種である．これは総称的関数 coef(), formula(), resid(), print(), summary(), AIC(), fitted() そして vcov() に対するメソッドを持つ．

```
# ロジスティック関数を用いた自己スタートモデルの使用
# 酵素免疫吸着測定法によるネズミ血清の組み換え分解酵素定量データ DNase 使用
> DNase1 <- DNase[DNase$Run==1, ]                    # 第一回の実験データを取り出す
> fm1DNase1 <- nls( density ~ SSlogis(log(conc),Asym,xmid,scal), DNase1 )

> summary(fm1DNase1)                                 # 非線形回帰結果要約出力
Formula: density ~ SSlogis(log(conc), Asym, xmid, scal) # 呼出し式
Parameters:                                          # パラメータ推定値，標準偏差，t 値，p 値
     Estimate Std. Error t value Pr(>|t|)
Asym  2.34518    0.07815   30.01 2.17e-13 ***
xmid  1.48309    0.08135   18.23 1.22e-10 ***
scal  1.04146    0.03227   32.27 8.51e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.01919 on 13 degrees of freedom # 残差の標準偏差
Correlation of Parameter Estimates:                  # パラメータ推定値の相関行列
       Asym   xmid
xmid 0.9868
scal 0.9008 0.9063
```

```
# 条件付き線形性 (plinear 法) を用いる当てはめ
> fm2DNase1 <- nls(density ~ 1/(1 + exp(( xmid - log(conc) )/scal ) ),
                   data = DNase1,
                   start = list( xmid = 0, scal = 1 ),
                   alg = "plinear", trace = TRUE )
0.7139315 : 0.000000 1.000000 1.453853
0.1445295 : 1.640243 1.390186 2.461754
0.008302151 : 1.620899 1.054228 2.478388
0.004794192 : 1.485226 1.043709 2.347334
0.004789569 : 1.483130 1.041468 2.345218
0.004789569 : 1.483090 1.041455 2.345180

> summary(fm2DNase1)
Formula: density ~ 1/(1 + exp((xmid - log(conc))/scal))
Parameters:
     Estimate Std. Error t value Pr(>|t|)
xmid  1.48309    0.08135   18.23 1.22e-10 ***
scal  1.04145    0.03227   32.27 8.51e-14 ***
.lin  2.34518    0.07815   30.01 2.17e-13 ***
---
```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.01919 on 13 degrees of freedom
Correlation of Parameter Estimates:
       xmid   scal
scal 0.9063
.lin 0.9868 0.9008
```

```
# 条件付き線形性無しの当てはめ
> fm3DNase1 <- nls(density ~ Asym/(1 + exp((xmid-log(conc))/scal ) ),
                   data = DNase1,
                   start = list( Asym = 3, xmid = 0, scal = 1 ),
                   trace = TRUE )
14.32279 :  3 0 1
0.4542698 :  2.1152456 0.8410193 1.2000640
0.05869603 :  2.446376 1.747516 1.189515
0.005663524 :  2.294087 1.412198 1.020463
0.004791528 :  2.341429 1.479688 1.040758
0.004789569 :  2.345135 1.483047 1.041439
0.004789569 :  2.345179 1.483089 1.041454

> summary(fm3DNase1)
Formula: density ~ Asym/(1 + exp((xmid - log(conc))/scal))
Parameters:
     Estimate Std. Error t value Pr(>|t|)
Asym  2.34518    0.07815   30.01 2.17e-13 ***
xmid  1.48309    0.08135   18.23 1.22e-10 ***
scal  1.04145    0.03227   32.27 8.51e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.01919 on 13 degrees of freedom
Correlation of Parameter Estimates:
       Asym   xmid
xmid 0.9868
scal 0.9008 0.9063
```

```
# 重みつき非線形回帰．ピューロマイシン処置による酵素反応での基質濃度データ Puromycin 使用
> Treated <- Puromycin[Puromycin$state=="treated",]  # 処理データを取り出す
# Michaelis-Menten モデル関数の重みつきバージョン．Vm,K は二つのパラメータ
> weighted.MM <- function(resp, conc, Vm, K) {
    pred <- (Vm * conc)/(K + conc)
    (resp - pred) / sqrt(pred) }
> Pur.wt <- nls( ~ weighted.MM(rate, conc, Vm, K), data = Treated,
                 start = list(Vm = 200, K = 0.1),
                 trace = TRUE)
112.5978 :  200.0   0.1                              # 以下当てはめ途中結果
17.33824 :  205.67588840   0.04692873
14.6097 :  206.33087394   0.05387279
14.59694 :  206.79883498   0.05457132
14.59690 :  206.83291310   0.05460917
14.59690 :  206.83468214   0.05461109

> summary(Pur.wt)                                  # 要約
Formula: 0 ~ weighted.MM(rate, conc, Vm, K)
Parameters:
    Estimate Std. Error t value Pr(>|t|)
Vm 2.068e+02  9.225e+00  22.421 7.00e-10 ***
K  5.461e-02  7.979e-03   6.845 4.49e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 1.208 on 10 degrees of freedom
Correlation of Parameter Estimates:
      Vm
K 0.7543
```

7.1.2　nls オブジェクトからモデル式を取り出す formula.nls()

object を当てはめるために使われるモデルを返す．

書式： formula(x, ...) # クラス"nls"に対する S3 メソッド

引数：

x　クラス "nls" を継承するオブジェクトで，非線形最小自乗当てはめ結果を表す
...　他のメソッドへ (から) 引き渡される追加引き数

返り値：　object を得るのに使われたモデルを表す式

```
# オレンジの木の成長データ Orange を使用
> fm1 <- nls(circumference ~ A/(1+exp((B-age)/C)), Orange,
             start = list(A=160, B=700, C = 350))
> formula(fm1)                                    # 非線形モデル式を取り出す
circumference ~ A/(1 + exp((B - age)/C))
```

7.1.3 非線形最小自乗当てはめの制御 nls.control()

nls() 関数による非線形最小自乗法アルゴリズムの幾つかの特性をユーザが設定できるようにする．

書式：nls.control(maxiter=50, tol=1e-05, minFactor=1/1024)

引数：

maxiter　許される最大繰りかえし回数を指定する正整数

tol　相対オフセット収束基準に対する許容レベルを指定する正数

minFactor　繰りかえしの各ステップで許される最小のステップサイズを指定する正数．増分は Gauss-Newton アルゴリズムで計算され，残差平方和が減少するか，ステップサイズがこれ以下になるまで，逐次 0.5 倍される

返り値：　引数の解説で与えられた意味を持つ丁度 3 つの成分 maxiter, tol, minFactor

関連：nls().

```
> nls.control(minFactor = 1/2048)                 # 制御パラメータの指定例
$maxiter
[1] 50                                            # 既定値のまま
$tol
[1] 1e-05                                         # 既定値のまま
$minFactor
[1] 0.0004882812                                  # =1/2048
```

7.1.4 nlsModel オブジェクトを作る nlsModel()

nlsModel オブジェクトを構成する．このオブジェクトはリスト形式の幾つかの関数の関数閉包である．この閉包は非線形モデル式，式に対するデータ値，そしてパラメータとそれらの値を含む．

書式：nlsModel(form, data, start)

引数：

form　非線形モデル式

data　モデル式からの変数をその中で表化するデータフレームもしくはリスト
start　モデル中のパラメータに対する初期推定値の名前付きリストまたは名前付き数値ベクトル

返り値：　共通の環境を共有する関数のリストで，以下の成分を持つ：
resid　現在のパラメータ値で評価された残差ベクトルを返す関数
fitted　現在のパラメータ値に於ける当てはめ目的変数とそれらのグラディエントを返す関数
formula　モデル式を返す関数
deviance　現在のパラメータ値での残差 2 乗和を返す関数
gradient　現在のパラメータ値でのモデル関数のグラディエントを返す関数
conv　現在のパラメータ値で評価された相対オフセット収束基準を返す関数
incr　Gauss-Newton 法により計算されたパラメータ増分を返す関数
setPars　一つの引数 pars を持つ関数．これは nlsModel オブジェクトに対するパラメータ値を設定し，グアディエント配列の特異性を示す論理値を返す関数
getPars　現在のモデルパラメータを数値ベクトルとして返す関数
getAllPars　モデルパラメータの現在値を数値ベクトルとして返す関数
getEnv　これらの関数が共有する環境を返す関数
trace　途中結果表示が指定されたとき，各繰りかえしで呼び出される関数
Rmat　現在のモデルパラメータに於けるグラディエント配列の上三角因子
predict　引数 newdata,data.frame を持ち，newdata に対する目的変数の予測値を返す

nlsModel オブジェクトは主に nls() 関数の内部で使われる．これはモデル，データ，そして環境中のパラメータ等を一まとめにし，モデルの特性量にアクセスする幾つかの方法を与える．これは nls() 関数が返すオブジェクトの重要な成分を形作る．

関連：nls().

```
# 酵素免疫吸着測定法によるネズミ血清の組み換え分解酵素定量データ DNase 使用
> DNase1 <- DNase[DNase$Run==1, ]
> mod <- nlsModel(density ~ SSlogis( log(conc), Asym, xmid, scal ),
                  DNase1, list( Asym = 3, xmid = 0, scal = 1 ))
> mod$getPars()                                    # パラメータをリストで返す
Asym xmid scal
   3    0    1
> mod$deviance()                                   # 残差平方和を返す
[1] 14.32279
> mod$resid()                                      # 残差ベクトルとグラディエントを返す
 [1] -0.1226648 -0.1216648 -0.3691961 -0.3661961 -0.6366966 -0.6276966
 [7] -0.9387895 -0.9417895 -1.2152683 -1.2202683 -1.2537273 -1.2717273
[13] -1.2522069 -1.2222069 -1.0477778 -1.0677778
attr(,"gradient")
            Asym       xmid       scal
 [1,] 0.04655493 -0.1331627  0.4020780
 [2,] 0.04655493 -0.1331627  0.4020780
 [3,] 0.16339869 -0.4100987  0.6697545
 (途中省略)
[16,] 0.92592593 -0.2057613 -0.5196973

> mod$incr()                                       # 推奨されるパラメータ増分を返す
      Asym       xmid       scal
-0.8847544  0.8410193  0.2000640
```

```
> mod$setPars(unlist(mod$getPars())+mod$incr())   # 新しいパラメータ値を設定
[1] FALSE
> mod$getPars()                                   # パラメータ値が変更されたかをチェック
     Asym      xmid      scal
2.1152456 0.8410193 1.2000640
> mod$deviance()                                  # パラメータ値変更が成功したか見る
[1] 0.4542698
> mod$trace()                                     # 途中経過追跡情報のチェック
0.4542698 :  2.1152456 0.8410193 1.2000640
> mod$Rmat()                                      # グラディエントの QR 分解からの R 行列
          Asym      xmid       scal
[1,] -1.870969 1.1570254  0.3360385
[2,]  0.000000 0.5594739 -1.0172407
[3,]  0.000000 0.0000000 -0.5391677
```

7.1.5 非線形回帰モデルによる予測 predict.nls()

フレーム newdata 中の回帰関数を評価して得られる予測値を計算する．もし論理値 se.fit が TRUE なら，予測の標準偏差が計算される．もし数値引数 scale が設定 (オプション df 付き) されていれば，それが標準偏差の計算の過程で残差標準偏差として使われる．さもなければ，これは当てはめモデルから取り出される．intervals は指定された水準 level で信頼区間，予測 (許容) 区間のどちらを計算するかを指示する．現在のところ se.fit と interval は無視される．

書式： # クラス "nls" に対する S3 メソッド

```
predict(object, newdata , se.fit = FALSE, scale = NULL, df = Inf,
        interval=c("none","confidence","prediction"),level=0.95,...)
```

引数：

object　クラス "nls" を継承するオブジェクト

newdata　object 中のモデルに対する入力データの値を持つ名前付きリストかデータフレーム．もし newdata が無ければ，元の点における当てはめ値が返される

se.fit　論理値で，予測の標準誤差を計算するかどうか指示する．既定値は FALSE．現在これは無視される

scale　数値スカラ．もしこれが (オプションの df とともに) 設定されると，標準誤差の計算で残差標準偏差として使われる．さもなければ，この情報は当てはめモデルから取り出される．現在これは無視される

df　正の数値スカラで scale 推定値の自由度を与える．現在これは無視される

interval　文字列で，平均目的変数に対する予測区間か信頼区間のどちらを計算するかを指示する．現在これは無視される

level　0 と 1 の間の数値スカラで，区間 (もし必要とされれば) の信頼水準を与える．現在これは無視される

...　追加のオプション引数．現在のところオプション引数は使われない

返り値：　predict.nls() は予測値のベクトルを計算する．移植後は interval は列名 fit, lwr そして upr を持つ予測値と限界からなる行列を計算する．移植後は，もし se.fit が TRUE なら，次のような成分を持つリストを返す：

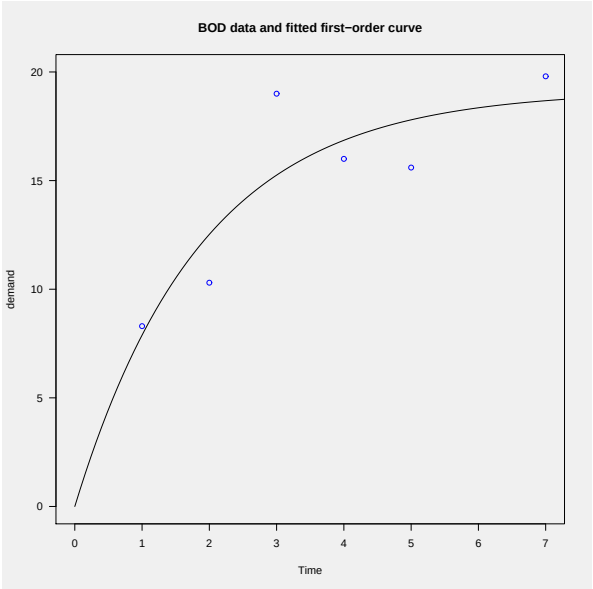
fit　上で述べたようなベクトルまたは行列

se.fit　予測の標準誤差

`residual.scale`　残差の標準偏差
`df`　残差に対する自由度

関連： モデル当てはめ関数 `nls()`, `predict()`.

```
# 生化学的酸素要求量の時間変化データ BOD 使用（次の図を参照）．原点を通る漸近回帰モデル当てはめ
> fm <- nls(demand ~ SSasympOrig(Time, A, lrc), data = BOD)
> predict(fm)                                          # 観測時間に於ける当てはめ値
[1]  7.887449 12.524977 15.251673 16.854870 17.797490 18.677580

# データをプロットし，滑らかな予測曲線を上描き
> opar <- par(las = 1)                                 # 軸ラベルを軸に対して水平に書く
> plot(demand ~ Time, data = BOD, col = 4,             # データの散布図を描く
       main = "BOD data and fitted first-order curve",
       xlim = c(0,7), ylim = c(0, 20) )
> tt <- seq(0, 8, length = 101)                        # 予測値を計算する時間値
> lines(tt, predict(fm, list(Time = tt)))              # 予測曲線を上描き
> par(opar)                                            # 作図パラメータを元に戻す
```



生化学的酸素要求量の時間変化データへの原点を通る非線形モデルの当てはめ，データ散布図と予測値曲線

7.1.6　nls オブジェクトのプロファイル profile.nls()

`nls` オブジェクトの `fitted` で表される解の近傍での対数尤度関数の挙動を調べる．

書式： `# クラス "nls" に対する S3 メソッド`

```
profile(fitted, which=1:npar, maxpts=100, alphamax=0.01,
        delta.t=cutoff/5, ...)
```

引数：

`fitted`　オリジナルの当てはめモデルオブジェクト
`which`　プロファイルされるべきオリジナルモデルのパラメータ．既定では全てのパラメータが対象
`maxpts`　各パラメータのプロファイルに使われる点の最大数
`alphamax`　プロファイル t 統計量に対して許される最大有意水準

delta.t プロファイル t 統計量のスケールに対する提案変化量．既定値は約 10 個のパラメータ値でプロファイルするように選ばれる

... 他のメソッドへ (から) 引き渡される追加引き数

返り値： プロファイルされた各パラメータに対する成分を持つリスト．成分は二つの変数を持つデータフレーム：

par.vals 各当てはめモデルに対するパラメータ値の行列

tau プロファイル t 統計量

プロファイル t 統計量は，2 乗和の変化を残差標準誤差で割ったものの平方根に適当な符号を付けたものと定義される．

関連：nls() , profile() , profiler.nls(), plot.profile.nls().

```
# 生化学的酸素要求量の時間変化データ BOD 使用．原点を通る漸近回帰モデル当てはめ
> ( fm1 <- nls(demand ~ SSasympOrig(Time, A, lrc), data = BOD) )
Nonlinear regression model
  model:  demand ~ SSasympOrig(Time, A, lrc)
   data:  BOD
         A        lrc
19.1425781 -0.6328217                              # パラメータ値の最小自乗推定値
 residual sum-of-squares:  25.99027

> pr1 <- profile(fm1)                              # 当てはめオブジェクトをプロファイル
# パラメータ A を最小自乗推定値の近傍で変化させ，対応するパラメータ lrc を計算
> pr1$A                                            # 二つのパラメータに対するプロファイル値
          tau  par.vals.A par.vals.lrc
1  -6.9431473   8.6140764    1.3100919
2  -5.6876147  10.1501074    0.7250247
3  -4.4000315  11.7982472    0.4130025
4  -3.0291032  13.6811484    0.1282910
5  -1.4572773  16.1474757   -0.2204915
6   0.0000000  19.1425781   -0.6328217              # <- 最小自乗推定値
7   0.8237658  21.6089054   -0.9323718
8   1.5987772  25.2016655   -1.2765739
9   2.2783012  30.7645672   -1.6544023
10  2.8696966  40.5883279   -2.0922188
11  3.3589747  60.5217150   -2.6278577
12  3.7278452 109.4101990   -3.3245892
# パラメータ lrc を最小自乗推定値の近傍で変化させ，対応するパラメータ A を計算
> pr1$lrc
         tau    par.vals.A  par.vals.lrc
1  -4.110333 37734.3686740    -9.2784429
2  -4.003207   363.4779547    -4.6068473
3  -3.598903    83.8735298    -3.0328939
4  -2.944324    42.0949599    -2.1743299
5  -2.111565    28.6211369    -1.5785154
6  -1.131065    22.5121290    -1.0916854
7   0.000000    19.1425781    -0.6328217           # <- 最小自乗推定値
8   1.256814    17.0705319    -0.1459917
9   2.302559    15.9214179     0.3188315
10  3.108871    15.1954572     0.8522192
11  3.511947    14.8533089     1.6460380
12  3.535610    14.8333333     4.0093204
```

7.1.7 非線型回帰オブジェクトに対するプロファイラを構成する profiler.nls()

クラス "nls" の当てはめオブジェクトに対するプロファイラを構成する．

書式： profiler(fitted, ...) # クラス"nls"に対する S3 メソッド

引数：
fitted　クラス nls の元の当てはめオブジェクト
...　追加パラメータ，どれも使われない

返り値：　クラス "profiler.nls" のオブジェクトで，次の関数成分を持つリスト：
getFittedModel()　fitted に対する nlsModel オブジェクト
getFittedPars()　profiler() の解説を見よ
setDefault(varying, params)　profiler() の解説を見よ
getProfile(varying, params)　返されたリスト中 fstat は残差平方和の変化量と残差標準偏差の比である

その他の詳細については関数 profiler() の解説を見よ．

注意：setDefault() と getProfile() を同時に使うと，当てはめモデルの内部状態は変化する可能性がある．したがってパラメータに対するプロファイリングが完了したら，内部状態は setDefault() を引数内で呼び出して元に戻すべきである．例えば，以下の例を見るか，profile.nls() のソースコードを参照せよ．
関連：nls() , nlsModel() , profiler() , profile.nls().

```
# 生化学的酸素要求量の時間変化データ BOD 使用
# 当てはめオブジェクトを得る（原点を通る漸近回帰モデル使用）
> fm1 <- nls(demand ~ SSasympOrig( Time, A, lrc ), data = BOD)
> prof1 <- profiler(fm1)   # 当てはめオブジェクトに対するプロファイラを得る

> prof1$getProfile(c(FALSE, TRUE), 16.0)          # A を 16.0 に固定したプロファイル
$fstat
[1] 2.379586
$parameters
         A        lrc
16.0000000 -0.1997343
$varying
[1] FALSE  TRUE

> prof1$setDefault(varying = c(FALSE, TRUE))      # lrc を変化させる

# A を 14.0 に固定し，lrc の初期値を-0.2 にする
> prof1$setDefault(params = c(14.0, -0.2))
> prof1$getProfile()                              # そしてプロファイルを得る
$fstat
[1] 7.896032
$parameters
         A         lrc
14.00000000  0.08265115
$varying
[1] FALSE  TRUE
> prof1$setDefault()                              # 最後に当初の推定値に戻す
```

7.1.8　"profile.nls" オブジェクトのプロット plot.profile.nls()

プロファイル関数の一連のプロットを表示し，nls() 関数で当てはめられ profile.nls() でプロファイルされた非線形回帰モデル中のパラメータに対する信頼区間を補間する．

書式：# クラス "profile.nls" に対する S3 メソッド

```
plot(x, levels, conf= c(99, 95, 90, 80, 50)/100,
     nseg = 50, absVal =TRUE, ...)
```

引数：

x　クラス "profile.nls" のオブジェクト

levels　t 統計量の絶対値のスケールでの，補間される信頼区間の信頼水準．普通 levels を直接与えるより， conf が使われる

conf　パラメータのプロファイルに基づく信頼水準の数値ベクトル．既定値は c(0.99,0.95,0.90,0.80,0.50)

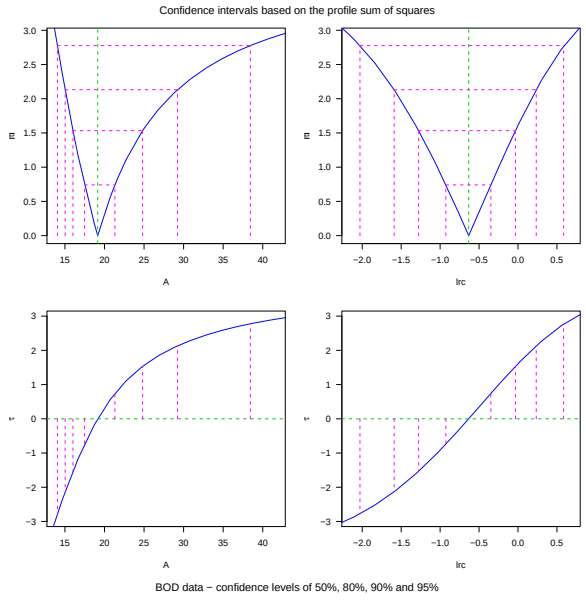
nseg　プロファイル t 曲線のスプライン補間で使われる線分の数を与える整数値．既定値は 50

absVal　プロットはプロファイル t の絶対値のスケールで行うべきかどうかを指示する論理値．既定値は TRUE

...　plot 関数に引き渡されるその他の引数

関連：nls(), profile(), profile.nls().

```
# 生化学的酸素要求量の時間変化データ BOD 使用（次の図を参照）
> fm1 <- nls(demand ~ SSasympOrig(Time,A,lrc ), data=BOD) # 当てはめオブジェクトを得る
> pr1 <- profile(fm1)                                    # 当てはめモデルに対するプロファイルを得る
> opar <- par(mfrow = c(2,2), oma = c(1.1, 0, 1.1, 0), las = 1)
> plot(pr1, conf = c(95, 90, 80, 50)/100)
> plot(pr1, conf = c(95, 90, 80, 50)/100, absVal = FALSE)
> mtext("Confidence intervals based on the profile sum of squares",
        side = 3, outer = TRUE)
> mtext("BOD data - confidence levels of 50%, 80%, 90% and 95%",
        side = 1, outer = TRUE)
> par(opar)
```



BOD データに対する，プロファイルによる，信頼水準 50%, 80%, 90%, 95% の (補間) 信頼区間 (例えば左右一番外側の縦線間が 95% 信頼区間に相当する)

7.1.9 対数線形モデル loglin()

比率の当てはめの繰り返しにより，対数線形モデルを多元分類分割表に当てはめる．

書式：

```
loglin(table, margin, start = rep(1, length(table)), fit = FALSE,
       eps = 0.1, iter = 20, param = FALSE, print = TRUE)
```

引数：

`margin` 当てはめられるべき交互作用を指定するベクトルのリスト．(階層的) 対数線形モデルは，モデルに含まれる「最大」因子部分集合を与える，こうした周辺和を用いて指定できる．例えば，3 因子モデルにおいて `list(c(1,2),c(1,3))` は，それぞれ一般和，各因子，そして 1-2,1-3 交互作用 (しかし例えば 2-3,1-2-3 交互作用は無い) に対するパラメータを含むモデルを指定する．つまり，因子 1 に対する条件付きで因子 2,3 が独立なモデルである (しばしば `[12][13]` と表される)．数値添字の代わりに因子名 (つまり `names(dimnames(table))`) を与えても良い

`start` 当てはめ表に対する初期推定値．このオプション引数は，当てはめ過程で保存されるべき構造的な 0 を持つ不完全な表に対して重要となる．この場合， `start` 中の対応項目は 0 でなければならず，その他は 1 として良い

`fit` 当てはめ値を返すべきかどうか指示する論理値

`eps` 観測値と当てはめ値の間に許される最大の食い違い

`iter` 最大繰り返し数

`param` パラメータ値を返すべきかどうか指示する論理値

`print` 論理値．もし `TRUE` なら，繰り返し回数と最終のずれが出力される

`table` 当てはめ対象の分割表．普通 `table()` の出力

返り値： 以下の成分を含むリスト：

`lrt` 尤度比検定統計量

`pearson` Pearson の (カイ 2 乗) 検定統計量

`df` 当てはめモデルに対する自由度．構造的な 0 に対する補正はされない

`margin` 当てはめられた周辺リスト．基本的に入力 `margin` と同じであるが，数字は可能なら名前で置き換えられる

`fit` 当てはめ値を含む `table` と同形式の配列．`fit=TRUE` の時だけ返される

`param` モデルの推定パラメータを含むリスト．周辺和 0 の「標準」対比 (例えば，二因子パラメータに対して行和と列和が 0) が用いられる．`param=TRUE` の時だけ返される

当てはめには，Haberman に紹介されているような繰り返し比率当てはめアルゴリズムが使われている．最大で `iter` 回の繰り返しが行われ，観測値と当てはめ値の周辺間の最大ずれが `eps` 以下になれば収束したと判定される．内部的な計算は倍精度で行われる．モデル中の因子の数 (表の次元) に制限はない．構造的な 0 がないと仮定すると，尤度比検定統計量と Pearson 検定統計量はともに漸近的に自由度 `df` のカイ 2 乗分布に従う．

注意 パッケージ `MASS` は `loglin()` の前処理用の関数 `loglm()` を提供し，これは `lm` や `glm` のような他の当てはめ関数と同様な，モデル式による対数線形モデルの指定と当てはめを可能にする．

```
# 髪，眼の色と性別のデータ HairEyeColor を使用
# 髪・眼色があわせて性別と独立であるというモデル（3 次の交互作用無しモデルが良く当てはまる）
> fm <- loglin(HairEyeColor, list(c(1, 2), c(1, 3), c(2, 3)))
5 iterations: deviation 0.08166832                # 繰り返し回数と最終ずれ
> fm
$lrt                                              # 尤度比検定統計量
[1] 8.187009
$pearson                                          # Pearson 検定統計量
[1] 8.50081
$df                                               # 自由度
[1] 9
$margin
$margin[[1]]
[1] "Hair" "Eye"
$margin[[2]]
[1] "Hair" "Sex"
$margin[[3]]
[1] "Eye" "Sex"
> 1 - pchisq(fm$lrt, fm$df)                       # 尤度比検定統計量に対する p 値
[1] 0.5154158
```

7.2 漸近回帰モデル

この節ではしばしば用いられる非線型回帰モデルを扱う関数を紹介する．これらはモデル曲線が漸近線を持ち，それがパラメータ値に関係することから，漸近回帰モデル (asymptotic regression model) と呼ばれる．こうしたモデルではモデルパラメータの適切な初期推定値を見出すことが困難な問題となるが，自己開始モデル (SS, Self-Start model) と呼ばれるメカニズムは，こうした初期推定値を計算する手順自身をモデルに含むことを可能にする．

7.2.1 漸近回帰モデル NLSstAsymptotic()

xy データに $b_0 + b_1(1 - \exp(-\exp(\texttt{lrc})x)$ の形の漸近回帰モデルを当てはめる．これはより複雑なモデルの初期推定値を決定するのに使うことができる．

書式： NLSstAsymptotic(xy)
引数： xy sortedXyData() で処理されたオブジェクト
返り値： 名前付き数値成分 b0, b1, lrc を持つリスト．b0 は y 軸に対する切片推定値，b1 は y 軸切片と漸近値との距離の推定値，そして lrc は比率 (rate) 定数の対数値の推定値

関連： SSasymp().

```
# loblolly pine tree(テーダ松) の成長データ Loblolly 使用
> Lob.329 <- Loblolly[ Loblolly$Seed == "329", ]  # 種子グループ 329 だけを利用
> NLSstAsymptotic(sortedXyData(expression(age), expression(height), Lob.329 ))
        b0         b1        lrc
 -8.250753 102.378957  -3.217578                 # 回帰係数推定値
```

7.2.2 非線型回帰モデルのパラメータ初期値を求める，getInitial()

非線型回帰モデルに対する初期パラメータ推定値を求める．もし data がパラメータ化されたデータフレームか，pframe オブジェクトなら，その parameter 属性が返される．さもなければ，オブジェクトが評価可能な"initial" 属性を持つ selfStart オブジェクトへの呼び出しを含むかどうかが検査される．

書式： getInitial(object, data, ...)

引数：

object モデル式，もしくは非線型回帰モデルを定義する selfStart モデル

data モデル式中の表現式や， selfStart モデルの引数をその中で評価できるデータフレーム

... オプションの追加引数

返り値： パラメータの初期推定値の名前付き数値ベクトル，もしくはリスト．多くの selfStart モデルは，こうした初期推定値が実際にも収束値となるように構成されている

関連：nls(), selfStart(), selfStart.default(), selfStart.formula().

```
# ピューロマイシン処置による酵素反応での基質濃度データ Puromycin を使用
> PurTrt <- Puromycin[ Puromycin$state == "treated", ] # 処理群のみ取り出す
# Michaelis-Menten モデルの当てはめ
> getInitial( rate ~ SSmicmen( conc, Vm, K ), PurTrt )
          Vm           K                                  # パラメータ Vm,K の初期推定値
212.68370735   0.06412123
```

7.2.3 自己開始型漸近回帰モデル SSasymp()

この「自己開始」モデルは漸近回帰関数とそのグラディエントを評価する．これは与えられたデータセットに対しパラメータ Asym, R0, lrc の初期推定値を評価する「初期属性」を持つ．

書式： SSasymp(input, Asym, R0, lrc)

引数：

input モデルをそこで評価する値の数値ベクトル

Asym 右端 (input の非常に大きな値) での水平漸近値を表す数値パラメータ

R0 input が 0 の時の目的変数値を表す数値パラメータ

lrc 比率 (rate) 定数の自然対数を表す数値パラメータ

返り値： input と同じ長さの数値ベクトル．これは式

$$\mathtt{Asym} + (\mathtt{R0} - \mathtt{Asym}) \exp(-e^{\mathtt{lrc}} \times \mathtt{input})$$

の値である．もし全ての引数 Asym, R0, lrc がオブジェクトの名前なら，これらの名前に関するグラディエント行列が属性 "gradient" として付加される

関連：nls(), selfStart().

```
# loblolly pine tree(テーダ松) の成長データ Loblolly を使用
> Lob.329 <- Loblolly[ Loblolly$Seed == "329", ]  # 種子グループ 329 だけを使用
> SSasymp( Lob.329$age, 100, -8.5, -3.2 )         # 目的変数だけ
[1]  3.988924 11.505611 27.822517 41.130854 51.985354 60.838463
> Asym <- 100
> resp0 <- -8.5
> lrc <- -3.2
> SSasymp( Lob.329$age, Asym, resp0, lrc )        # 目的変数とグラディエント
[1]  3.988924 11.505611 27.822517 41.130854 51.985354 60.838463
attr(,"gradient")                                 # グラディエント値の行列
          Asym     resp0      lrc
[1,] 0.1151053 0.8848947 11.74087
[2,] 0.1843835 0.8156165 18.03613
[3,] 0.3347697 0.6652303 29.42113
[4,] 0.4574272 0.5425728 35.99454
[5,] 0.5574687 0.4425313 39.14366
[6,] 0.6390642 0.3609358 39.90776

> getInitial(height ~ SSasymp( age, Asym, resp0, lrc), data = Lob.329)
$Asym                                             # SSasymp() が与えるパラメータ初期値
[1] 94.1282
$resp0
[1] -8.250753
$lrc
[1] -3.217578

> fm1 <- nls(height ~ SSasymp( age, Asym, resp0, lrc), data = Lob.329)
> summary(fm1)                                  # 初期値は実際は nls() による収束値でもある
Formula: height ~ SSasymp(age, Asym, resp0, lrc)
Parameters:
      Estimate Std. Error t value Pr(>|t|)
Asym   94.1282     8.4030  11.202 0.001525 **
resp0  -8.2508     1.2261  -6.729 0.006700 **
lrc    -3.2176     0.1386 -23.218 0.000175 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.7493 on 3 degrees of freed
resp0  0.7707
lrc   -0.9920 -0.8288
```

7.2.4 オフセットを持つ自己開始型漸近回帰モデル SSasympOff()

SSasympOff() はもう一つのパラメータ化を持つ漸近回帰関数とそのグラディエントを評価する．これはパラメータ Asym, lrc, c0 の初期推定値を計算する initial 属性を持つ．

書式： SSasympOff(input, Asym, lrc, c0)

引数：

input　そこでモデルを評価する値の数値ベクトル

Asym　右端 (input の非常に大きな値) での水平漸近値を表す数値パラメータ

lrc　比率 (rate) 定数の自然対数を表す数値パラメータ

c0　値が 0 となる input を表す数値パラメータ

返り値：　input と同じ長さの数値ベクトル．これはオフセット項 c0 を持つ式

$$\texttt{Asym} \times (1 - \exp(-e^{\texttt{lrc}} \times (\texttt{input} - \texttt{c0})))$$

の値である．もし全ての引数 Asym, lrc, c0 がオブジェクトの名前なら，これらの

名前に関するグラディエント行列が属性 "gradient" として付加される.

関連：nls(), selfStart().

```
# 加冷処理した草 Echinochloa crus-galli の二酸化炭素摂取量データ CO2 を使用
> CO2.Qn1 <- CO2[CO2$Plant == "Qn1", ]            # 1 番目の草のデータだけを取り出す
> SSasympOff( CO2.Qn1$conc, 32, -4, 43 )          # モデル値のみ
[1] 19.65412 29.14785 31.27791 31.88435 31.99259 31.99970 32.00000

> Asym <- 32; lrc <- -4; c0 <- 43                 # パラメータ値の変数
> SSasympOff( CO2.Qn1$conc, Asym, lrc, c0 )       # モデル値とグラディエント値
[1] 19.65412 29.14785 31.27791 31.88435 31.99259 31.99970 32.00000
attr(,"gradient")
          Asym          lrc            c0
[1,] 0.6141911 1.175838e+01 -2.261227e-01
[2,] 0.9108704 6.895531e+00 -5.223887e-02
(途中省略)
[7,] 1.0000000 1.369435e-05 -1.430967e-08

> getInitial(uptake ~ SSasymp(conc,Asym,lrc,c0), data=CO2.Qn1) # パラメータの初期値推定
$Asym
[1] 38.13978
$lrc
[1] -34.27658
$c0
[1] -4.380647

# 初期値は実は nls() による収束値でもある
> fm1 <- nls(uptake ~ SSasymp( conc, Asym, lrc, c0), data = CO2.Qn1)
> summary(fm1)                                    # 当てはめ結果要約
Formula: uptake ~ SSasymp(conc, Asym, lrc, c0)
Parameters:
     Estimate Std. Error t value Pr(>|t|)
Asym  38.1398     0.9164  41.620 1.99e-06 ***
lrc  -34.2766    18.9661  -1.807    0.145
c0    -4.3806     0.2042 -21.457 2.79e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 1.663 on 4 degrees of freedom
Correlation of Parameter Estimates:               # 推定パラメータ値間の相関
       Asym    lrc
lrc  0.4353
c0  -0.5728 -0.954
```

7.2.5 原点を通る自己開始型漸近回帰モデル SSasympOrig()

原点を通る漸近回帰モデルとそのグラディエントを評価する．与えられたデータに対してパラメータ Asym と lrc の初期推定値を計算する initial 属性を持つ.

書式： SSasympOrig(input, Asym, lrc)

引数：

input そこでモデルを評価する値の数値ベクトル

Asym 水平漸近値を表す数値パラメータ

lrc 比率 (rate) 定数の自然対数値を表す数値パラメータ

返り値： input と同じ長さの数値ベクトル．これは原点を通る式

$$\texttt{Asym} \times (1 - \exp(-e^{\texttt{lrc}} \times \texttt{input}))$$

の値である．もし全てのパラメータ Asym, lrc がオブジェクト名なら，これらの名前に関するグラディエントの行列が属性 "gradient" として付加される

関連： nls(), selfStart().

```
# loblolly pine tree(テーダ松) の成長データ Loblolly を使用
> Lob.329 <- Loblolly[Loblolly$Seed=="329", ]     # 番号 329 のデータを取り出す
> SSasympOrig( Lob.329$age, 100, -3.2 )           # モデル値だけ
[1] 11.51053 18.43835 33.47697 45.74272 55.74687 63.90642

> Asym <- 100; lrc <- -3.2                        # パラメータを名前で与える
> SSasympOrig(Lob.329$age, Asym, lrc)             # グラディエントも返される
[1] 11.51053 18.43835 33.47697 45.74272 55.74687 63.90642
attr(,"gradient")
          Asym      lrc
[1,] 0.1151053 10.82108
[2,] 0.1843835 16.62316
[3,] 0.3347697 27.11625
[4,] 0.4574272 33.17469
[5,] 0.5574687 36.07710
[6,] 0.6390642 36.78135

> getInitial(height ~ SSasympOrig(age, Asym, lrc), data = Lob.329)
      Asym        lrc
315.045692  -4.813894                           # パラメータ初期推定値

# nls() 関数で非線形回帰．初期値は実は収束値でもある
> fm1 <- nls(height ~ SSasympOrig( age, Asym, lrc), data = Lob.329)
> summary(fm1)
Formula: height ~ SSasympOrig(age, Asym, lrc)
Parameters:
     Estimate Std. Error t value Pr(>|t|)
Asym  315.046    443.071   0.711   0.5163
lrc    -4.814      1.527  -3.153   0.0344 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 2.822 on 4 degrees of freedom
Correlation of Parameter Estimates:              # パラメータ推定値の相関
       Asym
lrc -0.9997
```

7.2.6 自己開始型二重指数モデル SSbiexp()

この自己開始型モデルは二重指数モデルとそのグラディエントを評価する．これはパラメータ A1, lrc1, A2 そして lrc2 の初期推定値を生成する "initial" 属性を持つ．

書式： SSbiexp(input, A1, lrc1, A2, lrc2)

引数：

input　モデルを評価すべき値の数値ベクトル

A1　最初の指数関数の乗数を表す数値パラメータ

lrc1　最初の指数関数の比率 (rate) 定数の自然対数を表す数値パラメータ

A2　2 番目の指数関数の乗数を表す数値パラメータ

lrc2　2 番目の指数関数の比率 (rate) 定数の自然対数を表す数値パラメータ

返り値：　input と同じ長さの数値ベクトル．これは表現

$$\mathtt{A1}\exp(-e^{\mathtt{lrc1}}\times\mathtt{input})+\mathtt{A2}\exp(-e^{\mathtt{lrc2}}\times\mathtt{input})$$

の値である．もし全ての引数 A1, lrc1, A2 そして lrc2 がオブジェクト名なら，これらの名前に対するグラディエント行列が属性 "gradient" として付加される

関連：nls(), selfStart().

```
# インドメタシンの薬物動態学的データ Indometh を使用
> Indo.1 <- Indometh[Indometh$Subject==1, ]       # 個体番号 1 のデータだけ取り出す
> SSbiexp(Indo.1$time, 3, 1, 0.6, -1.3)          # 血液採取時間でのモデル式値だけ
 [1] 2.08098572 1.29421044 0.87967145 0.65483364 0.52711347 0.36094621
 [7] 0.26575722 0.20176113 0.15359129 0.11694936 0.06780767

> A1 <- 3; lrc1 <- 1; A2 <- 0.6; lrc2 <- -1.3     # パラメータ値
> SSbiexp( Indo.1$time, A1, lrc1, A2, lrc2 )      # グラディエントも評価
 [1] 2.08098572 1.29421044 0.87967145 0.65483364 0.52711347 0.36094621
 [7] 0.26575722 0.20176113 0.15359129 0.11694936 0.06780767
attr(,"gradient")                                 # グラディエント属性（グラディエントの行列）
               A1          lrc1        A2        lrc2
 [1,] 5.068347e-01 -1.033290e+00 0.9341363 -0.03818728
 [2,] 2.568814e-01 -1.047414e+00 0.8726106 -0.07134424
 （途中省略）
[11,] 3.595188e-10 -2.345456e-08 0.1130128 -0.14783797

# パラメータの初期推定値を得る
> getInitial(conc ~ SSbiexp(time, A1, lrc1, A2, lrc2), data = Indo.1)
       A1       lrc1         A2       lrc2
 2.0292774  0.5793887  0.1915475 -1.7877849

# 初期値は実際は収束値でもある
> fm1 <- nls(conc ~ SSbiexp(time, A1, lrc1, A2, lrc2), data = Indo.1)
> summary(fm1)
Formula: conc ~ SSbiexp(time, A1, lrc1, A2, lrc2)
Parameters:                                       # パラメータ推定値とその標準偏差，t 値，p 値
     Estimate Std. Error t value Pr(>|t|)
A1     2.0293     0.1099  18.464 3.39e-07 ***
lrc1   0.5794     0.1247   4.648  0.00235 **
A2     0.1915     0.1106   1.731  0.12698
lrc2  -1.7878     0.7871  -2.271  0.05737 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.04103 on 7 degrees of freedom
Correlation of Parameter Estimates:               # パラメータ推定値の相関行列
            A1   lrc1    A2
lrc1  0.002546
A2   -0.424384 0.8771
lrc2 -0.455538 0.7708 0.939
```

7.2.7　自己開始型一次コンパートメントモデル SSfol()

この「自己開始型」モデルは一次のコンパートメント関数とそのグラディエントを評価する．パラメータ lKe, lKa, lCl の初期推定値を計算する "initial" 属性を持つ．

書式： SSfol(Dose, input, lKe, lKa, lCl)

引数：

Dose 初期投薬量を表す数値ベクトル

input そこでモデルを評価する数値ベクトル

lKe 排泄割合 (elimination rate) 定数の自然対数を表す数値パラメータ

lKa 吸収割合 (absorption rate) 定数の自然対数を表す数値パラメータ

lCl 排除 (clearance) 定数の自然対数を表す数値パラメータ

返り値： input と同じ長さの数値ベクトルで式

$$\texttt{Dose} \times e^{\texttt{lKe}+\texttt{lKa}-\texttt{lCl}} \times \frac{e^{-e^{\texttt{lKe}}\times\texttt{input}} - e^{-e^{\texttt{lKa}}\times\texttt{input}}}{e^{\texttt{lKa}} - e^{\texttt{lKe}}}$$

の値である．もし全ての引数 lKe, lKa そして lCl がオブジェクト名であれば，これらの名前に対するグラディエント行列が "gradient" という属性名で付け加えられる．

関連：nls(), selfStart().

```
# テオフィリン（抗喘息薬）の薬物動態学データ Theoph を使用
> Theoph.1 <- Theoph[ Theoph$Subject == 1, ]      # 第一被験者のデータだけを取り出す
# 経口投与量と経過時間毎の検体のテオフィリン濃度（パラメータ初期値自動推定）
> SSfol(Theoph.1$Dose, Theoph.1$Time, -2.5, 0.5, -3)
 [1] 0.000000 2.214486 3.930988 5.261945 5.659813 5.084852 4.587699 3.916808
 [9] 3.318395 2.579204 0.943593

> lKe <- -2.5; lKa <- 0.5; lCl <- -3              # パラメータ初期値を与える
# 目的変数値とグラディエント
> SSfol( Theoph.1$Dose, Theoph.1$Time, lKe, lKa, lCl )
 [1] 0.000000 2.214486 3.930988 5.261945 5.659813 5.084852 4.587699 3.916808
 [9] 3.318395 2.579204 0.943593
attr(,"gradient")                                 # グラディエント属性
             lKe         lKa       lCl
 [1,]  0.0000000  0.00000000  0.000000
 [2,]  2.1902842  1.78781716 -2.214486
 （途中省略）
[11,] -0.8945410 -0.04944021 -0.943593

# パラメータの初期値を求める
> getInitial(conc ~ SSfol(Dose, Time, lKe, lKa, lCl), data = Theoph.1)
$lKe
[1] -2.994845
$lKa
[1] 0.609169
$lCl
[1] -3.971003
```

```
# SSfol() 関数によるモデルオブジェクトを用いて非線型回帰．初期値は実は nls() による収束値
> fm1 <- nls(conc ~ SSfol(Dose, Time, lKe, lKa, lCl), data = Theoph.1)
> summary(fm1)                                    # その結果要約
Formula: conc ~ SSfol(Dose, Time, lKe, lKa, lCl)
Parameters:                                       # パラメータ推定値，標準誤差，t 値，p 値
    Estimate Std. Error t value Pr(>|t|)
lKe  -2.9196     0.1709 -17.085 1.40e-07 ***
lKa   0.5752     0.1728   3.328   0.0104 *
lCl  -3.9159     0.1273 -30.768 1.35e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.732 on 8 degrees of freedom
Correlation of Parameter Estimates:               # 推定パラメータの相関
        lKe   lKa
```

```
lKa -0.5599
lCl  0.9604 -0.43
```

7.2.8　自己開始型 4 パラメータロジスティックモデル SSfpl()

4 パラメータロジスティック関数とそのグラディエントを評価する．与えられたデータセットに対しパラメータ A, B, xmid そして scal の初期推定値を評価する "initial" 属性を持つ．

書式： SSfpl(input, A, B, xmid, scal)

引数：

A　左端 (input の非常に小さな値) における水平漸近値を表す数値パラメータ

B　右端 (input の非常に大きな値) における水平漸近値を表す数値パラメータ

xmid　曲線の変曲点における input 値を表す数値パラメータ．SSfpl() の値は xmid において A と B の中間になる

scal　input 軸における数値スケールパラメータ

input　そこでモデルを評価する値の数値ベクトル

返り値：　input と同じ長さの数値ベクトル．これは式

$$\mathrm{A} + \frac{\mathrm{B} - \mathrm{A}}{1 + \exp((\mathrm{xmid} - \mathrm{input})/\mathrm{scal})}$$

の input における値である．もし全てのパラメータ A, B, xmid, scal がオブジェクト名なら，これらの名前に関するグラディエントの行列が属性 "gradient" として加えられる．

関連： nls(), selfStart().

```
# 給餌タイプ別のニワトリの体重増データ ChickWeight を使用
> Chick.1 <- ChickWeight[ChickWeight$Chick == 1,] # 第一のニワトリのデータ
> SSfpl( Chick.1$Time, 13, 368, 14, 6 )           # 指定パラメータによるモデル値
 [1]  44.38189  55.31704  69.39853  87.05603 108.47420 133.43149 161.18758
 [8] 190.50000 219.81242 247.56851 272.52580 283.70240

> A <- 13; B <- 368; xmid <- 14; scal <- 6
> SSfpl(Chick.1$Time, A, B, xmid, scal)           # パラメタを変数名で与える
 [1]  44.38189  55.31704  69.39853  87.05603 108.47420 133.43149 161.18758
 [8] 190.50000 219.81242 247.56851 272.52580 283.70240
attr(,"gradient")                                 # グラディエントも計算される
              A          B       xmid       scal
 [1,] 0.9116003 0.08839968  -4.767956  11.125231
 [2,] 0.8807971 0.11920292  -6.212120  12.424241
 (途中省略)
[12,] 0.2374580 0.76254197 -10.713410 -12.498978

# パラメータの初期推定値を計算
> getInitial(weight ~ SSfpl(Time, A, B, xmid, scal), data = Chick.1)
         A          B       xmid       scal
 27.453205 348.971211  19.390530   6.672621       # 初期推定値

# 初期推定値は実際は nls() 関数による最終収束値になっている
> fm1 <- nls(weight ~ SSfpl(Time, A, B, xmid, scal), data = Chick.1)
> summary(fm1)                                    # 非線型モデル当てはめ結果の要約
Formula: weight ~ SSfpl(Time, A, B, xmid, scal)
Parameters:                                       # 4 パラメータの推定値，標準偏差，t 値，p 値
```

```
     Estimate Std. Error t value Pr(>|t|)
A      27.453      6.601   4.159 0.003169 **
B     348.971     57.899   6.027 0.000314 ***
xmid   19.391      2.194   8.836 2.12e-05 ***
scal    6.673      1.002   6.662 0.000159 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 2.351 on 8 degrees of freedom
Correlation of Parameter Estimates:              # 推定パラメータ間の相関
           A      B   xmid
B    -0.8631
xmid -0.8347 0.9976
scal -0.9464 0.9704 0.9598
```

7.2.9 自己開始型ゴンペルツ成長曲線モデル SSgompertz()

この自己開始型モデルはゴンペルツ成長曲線モデルとそのグラディエントを評価する．パラメータ Asym, b2, b3 の初期推定値をつくり出すための "initial" 属性を持つ．

書式： SSgompertz(x, Asym, b2, b3)

引数：

x　そこでモデルを評価する値の数値ベクトル

Asym　漸近値を表す数値パラメータ

b2　関数の x=0 における値に関連した数値パラメータ

b3　x 軸のスケールに関連した数値パラメータ

返り値：　init と同じ長さの数値ベクトル．これは式 $\mathtt{Asym} \times \exp(-\mathtt{b2} \times \mathtt{b3}^x)$ の値である．もし全てのパラメータ Asym, b2, b3 がオブジェクト名ならば，これらの名前に関するグラディエントの行列が "gradient" という名前の属性として付け加えられる

関連：nls(), selfStart().

```
# 酵素免疫吸着測定法によるネズミ血清の組み換え分解酵素定量データ DNase を使用
> DNase1 <- DNase[DNase$Run==1, ]                # 第一回の実験データを取り出す
> Asym <- 4.5; b2 <- 2.3; b3 <- 0.7              # 名前付きパラメータ指定
# ゴンペルツ曲線を当てはめ（グラディエント行列も計算）
> SSgompertz(log(DNase.1$conc), Asym, b2, b3 )   # 当てはめ値とグラディエント
 [1] 0.00525729 0.00525729 0.07323255 0.07323255 0.18049064 0.18049064
 [7] 0.36508763 0.36508763 0.63288772 0.63288772 0.97257180 0.97257180
[13] 1.36033340 1.36033340 1.76786902 1.76786902
attr(,"gradient")
             Asym          b2         b3
 [1,] 0.001168287 -0.01543407  0.1531221
 [2,] 0.001168287 -0.01543407  0.1531221
 （途中省略）
[16,] 0.392859783 -0.71814108 -5.9597255

# パラメータ初期値を推定
> getInitial(density ~ SSgompertz(log(conc), Asym, b2, b3),
             data = DNase.1)
     Asym        b2        b3
4.6033339 2.2713391 0.7164666

# 得られた初期値は nls() 関数による最終収束値と一致
> fm1 <- nls(density ~ SSgompertz(log(conc), Asym, b2, b3), data = DNase.1)
> summary(fm1)                                   # ゴンペルツ曲線当てはめ結果の要約
Formula: density ~ SSgompertz(log(conc), Asym, b2, b3)
Parameters:                                      # パラメータ推定値，標準偏差，t 値，p 値
```

```
     Estimate Std. Error t value Pr(>|t|)
Asym  4.60333    0.65321   7.047 8.71e-06 ***
b2    2.27134    0.14373  15.803 7.24e-10 ***
b3    0.71647    0.02206  32.475 7.85e-14 ***
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.02684 on 13 degrees of freedom
Correlation of Parameter Estimates:               # 推定パラメータ間の相関
     Asym     b2
b2 0.9889
b3 0.9837 0.9517
```

7.2.10　自己開始型ロジスティックモデル SSlogis()

SSlogis() は自己開始型モデルであり，ロジスティック関数とそのグラディエントを評価する．与えられたデータセットに対しパラメータ Asym, xmid, scal の初期推定値を評価する initial 属性を持つ．

書式： SSlogis(input, Asym, xmid, scal)

引数：

Asym 漸近値を表す数値パラメータ

xmid 曲線の変曲点における input 値を表す数値パラメータ．SSlogis() の値は xmid において Asym/2 になる

scal input 軸における数値スケールパラメータ

input そこでモデルを評価する値の数値ベクトル

返り値： 返り値は input と同じ長さの数値ベクトル．これは式

$$\frac{\mathtt{Asym}}{1+\exp((\mathtt{xmid}-\mathtt{input})/\mathtt{scal})}$$

の input における値である．もし全てのパラメータ Asym, xmid, scal がオブジェクト名なら，これらの名前に関するグラディエントの行列が属性名 "gradient" として加えられる

関連：nls(), selfStart().

```
# 給餌タイプ別のニワトリの体重増データ ChickWeight を使用
> Chick.1 <- ChickWeight[ChickWeight$Chick == 1,] # 第一データを取り出す
> SSlogis( Chick.1$Time, 368, 14, 6 )             # 指定パラメータによる各時期のニワトリ体重
 [1]  32.53108  43.86668  58.46383  76.76794  98.97044 124.84166 153.61416
 [8] 184.00000 214.38584 243.15834 269.02956 280.61545

> Asym <- 368; xmid <- 14; scal <- 6              # パラメタを名前で与える
> SSlogis(Chick.1$Time, Asym, xmid, scal)
 [1]  32.53108  43.86668  58.46383  76.76794  98.97044 124.84166 153.61416
 [8] 184.00000 214.38584 243.15834 269.02956 280.61545
attr(,"gradient")                                 # グラディエントも計算される
           Asym       xmid       scal
 [1,] 0.08839968  -4.942557  11.532634
 [2,] 0.11920292  -6.439607  12.879213
 (途中省略)
[12,] 0.76254197 -11.105732 -12.956687

# パラメータの初期推定値を計算
> getInitial(weight ~ SSlogis(Time, Asym, xmid, scal), data = Chick.1)
     Asym      xmid      scal
937.02116  35.22280  11.40519

# 初期推定値は実際は nls() 関数による最終収束値になっている
> fm1 <- nls(weight ~ SSlogis(Time, Asym, xmid, scal), data = Chick.1)
> summary(fm1)                                    # 非線型モデル当てはめ結果の要約
Formula: weight ~ SSlogis(Time, Asym, xmid, scal)
Parameters:                                       # 4 パラメータの推定値，標準偏差，t 値，p 値
     Estimate Std. Error t value Pr(>|t|)
Asym 937.0212   465.8576   2.011  0.07516 .
xmid  35.2228     8.3119   4.238  0.00218 **
scal  11.4052     0.9052  12.599 5.08e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 2.919 on 9 degrees of freedom
Correlation of Parameter Estimates:               # 推定パラメータ間の相関
       Asym   xmid
xmid 0.9991
scal 0.9745 0.9829
```

7.2.11 自己開始型 Michaelis-Menten モデル SSmicmen()

SSmicmen() は Michaelis-Menten モデルとそのグラディエントを評価する．与えられたデータセットに対しパラメータ Vm, K の初期推定値を評価する initial 属性を持つ．

書式： SSmicmen(input, Vm, K)

引数：

input　そこでモデルを評価する値の数値ベクトル

Vm　目的変数の最大値を表す数値パラメータ

K　目的変数の最大値の半分を達成する input 値を表す数値パラメータ．酵素動態学分野ではこれは Michaelis パラメータと呼ばれる

返り値：　input と同じ長さの数値ベクトル．これは式

$$\frac{\mathtt{Vm} \times \mathtt{input}}{(\mathtt{K} + \mathtt{input}) \times \mathtt{A}}$$

の input における値である．もしパラメータ Vm, K の双方がオブジェクト名なら，これらの名前に関するグラディエントの行列が属性 "gradient" として加えられる

関連：nls(), selfStart().

```
# ピューロマイシン処置による酵素反応での基質濃度データ Puromycin を使用
> Treated <- Puromycin[Puromycin$state=="treated", ] # 処理群データ
> SSmicmen(PurTrt$conc, 200, 0.05)                  # 指定パラメータによるモデル値
 [1]  57.14286  57.14286 109.09091 109.09091 137.50000 137.50000 162.96296
 [8] 162.96296 183.60656 183.60656 191.30435 191.30435

> Vm <- 200; K <- 0.05                              # パラメタを変数名で与える
> SSmicmen(PurTrt$conc, Vm, K)
 [1]  57.14286  57.14286 109.09091 109.09091 137.50000 137.50000 162.96296
 [8] 162.96296 183.60656 183.60656 191.30435 191.30435
attr(,"gradient")                                   # グラディエントも計算される
             Vm         K
 [1,] 0.2857143 -816.3265
 [2,] 0.2857143 -816.3265
 (途中省略)
[12,] 0.9565217 -166.3516

> getInitial(rate ~ SSmicmen(conc,Vm,K),data=PurTrt) # パラメータの初期推定値を計算
          Vm           K
212.68370735   0.06412123                          # 初期推定値

# 初期推定値は実際は nls() 関数による最終収束値になっている
> fm1 <- nls(rate ~ SSmicmen(conc, Vm, K), data = PurTrt)
> summary(fm1)                                      # 非線型モデル当てはめ結果の要約
Formula: rate ~ SSmicmen(conc, Vm, K)
Parameters:                                         # 4 パラメータの推定値，標準偏差，t 値，p 値
    Estimate Std. Error t value Pr(>|t|)
Vm 2.127e+02  6.947e+00  30.615 3.24e-11 ***
K  6.412e-02  8.281e-03   7.743 1.57e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 10.93 on 10 degrees of freedom
Correlation of Parameter Estimates:                 # 推定パラメータ間の相関
      Vm
K 0.7651
```

7.2.12 自己開始型ワイブル成長モデル SSweibull()

SSweibull() はワイブル成長モデルとそのグラディエントを評価する自己開始モデルである．与えられたデータセットに対しパラメータ Asym, Drop, lrc そして pwr の初期推定値を評価する "initial" 属性を持つ．このモデルは SSasymp() モデルの拡張であり，pwr が 1 の時は SSasymp() に一致する．

書式： SSweibull(x, Asym, Drop, lrc, pwr)

引数：

input そこでモデルを評価する値の数値ベクトル

Asym 左端 (input の非常に小さな値) における水平漸近値を表す数値パラメータ

Drop Asym から y 軸切片までの変化量を表す数値パラメータ

lrc 比率 (rate) 定数の自然対数を表す数値パラメータ

pwr x の巾指数を表す数値パラメータ

返り値： 式

$$\mathtt{Asym} - \mathtt{Drop} \times \exp(-e^{\mathtt{lrc}} \times x^{\mathtt{pwr}})$$

の値．もし全てのパラメータ Asym, Drop, lrc, pwr がオブジェクト名ならば，これらの名前に関するグラディエントの行列が属性名 "gradient" として付け加えられる

関連：nls(), selfStart(), SSasymp().

```
# 給餌タイプ別のニワトリの体重増データ ChickWeight を使用
# 番号 6 のニワトリの測定時期 0 を除くデータを取り出す
> Chick.6 <- subset(ChickWeight, (Chick == 6) & (Time > 0))
> SSweibull(Chick.6$Time, 160, 115, -5.5, 2.5 )   # 指定パラメータによるモデル値
 [1]  47.62811  59.09743  79.79756 105.12008 128.41818 145.02585 154.25783
 [8] 158.24919 159.58222 159.92314 159.97023

> Asym <- 160; Drop <- 115; lrc <- -5.5; pwr <- 2.5 # パラメタを名前で与える
> SSweibull(Chick.6$Time, Asym, Drop, lrc, pwr)
 [1]  47.62811  59.09743  79.79756 105.12008 128.41818 145.02585 154.25783
 [8] 158.24919 159.58222 159.92314 159.97023
attr(,"gradient")                                  # グラディエントも計算される
      Asym          Drop        lrc        pwr
 [1,]    1 -0.9771469094  2.5978438  1.8006881
 [2,]    1 -0.8774136912 13.1957043 18.2931305
 (途中省略)
[11,]    1 -0.0002589123  0.2459116  0.7486834

# パラメータの初期推定値を計算
> getInitial(weight ~ SSweibull(Time, Asym, Drop, lrc, pwr), data = Chick.6)
      Asym       Drop        lrc        pwr
158.501204 110.997081  -5.993421   2.646141       # 初期推定値

# 初期推定値は実際は nls() 関数による最終収束値になっている
> fm1 <- nls(weight ~ SSweibull(Time, Asym, Drop, lrc, pwr), data = Chick.6)
> summary(fm1)                                     # 非線型モデル当てはめ結果の要約
Formula: weight ~ SSweibull(Time, Asym, Drop, lrc, pwr)
Parameters:                                        # パラメータの推定値，標準偏差，t 値，p 値
     Estimate Std. Error t value Pr(>|t|)
Asym 158.5012     1.1769  134.67 3.28e-13 ***
Drop 110.9971     2.6330   42.16 1.10e-09 ***
lrc   -5.9934     0.3733  -16.05 8.83e-07 ***
```

```
pwr    2.6461     0.1613   16.41 7.62e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.061 on 7 degrees of freedom
Correlation of Parameter Estimates:              # 推定パラメータ間の相関
        Asym    Drop     lrc
Drop  0.6093
lrc   0.3768  0.7682
pwr  -0.4410 -0.7641 -0.9934
```

7.2.13　漸近回帰モデルのその他の補助関数

`sortedXyData(x, y, data)`　x, y 座標データから関数を表す散布図データを計算する．自己開始モデルに対する `initial` 関数の内部で使われる．

`NLSstClosestX(xy, yval)`　関数を表す散布図データ `xy` (`sortedXyData()` 関数の返り値オブジェクト) から，線形補間により値が `yval` となる `x` 値を計算する．

`NLSstLfAsymptote(xy)`　`sortedXyData()` 関数の返り値オブジェクトから，関数の左側水平漸近値の初期推定を行う．自己開始モデルに対する `initial` 関数の内部で使われる．

`NLSstRtAsymptote(xy)`　`sortedXyData()` 関数の返り値オブジェクトから，関数の右側水平漸近値の初期推定を行う．自己開始モデルに対する `initial` 関数の内部で使われる．

7.3　その他，非線形混合モデル

R の推奨パッケージ `nlme` には線形混合モデルと並び，非線形混合効果モデル (nonlinear mixed effects model) 関連の豊富な関数群がある．

以下では，パッケージ `nlme` の中心的関数である `nlme()` のみ[*1] を取り上げる．

書式：

```
nlme(model, data, fixed, random, groups, start, correlation, weights,
     subset, method, na.action, naPattern, control, verbose)
```

引数：

`model`　非線形モデル公式．~ 演算子の左に応答変数，右側にパラメータと共変量を含む表現式か，`nlsList` オブジェクトを含む．もし `data` 引数が与えられれば，公式中に使われた全ての名前はデータフレーム中のパラメータか変数として定義されなければならない．メソッド関数 `nlme.nlsList()` のドキュメントは別個に与えられている

`data`　オプションのデータフレームで，引数 `model`, `fixed`, `random`, `correlation`, `weights`, `subset` そして `naPattern` 中に名前が与えらている変数を含む．既定では，変数は `nlme()` が呼び出された環境中から取られる

`fixed`　形式 `f1+...+fn ~ x1+...+xm` の両側線形公式か，`f1 ~ x1+...+xm` の形式の両側公式のリストで異なったパラメータには異なったモデルが可能．`f1,...,fn` はモデル引数の右側に含まれるパラメータの名前で，表現式 `x1+...+xm` はこれら

[*1] 線形混合効果モデル用関数 `lme()` に付いては 253 頁を参照．

のパラメータの線形モデルを定義する (もし公式の左側が複数のパラメータを含めば，それら全ては公式の右側の表現で指定される同じ線形モデルに従うとされる)．公式の右側中の'1' は対応するパラメータに対する単一の固定効果を意味する

random オプションで以下のどれか；(1) 形式 r1+...+rn~x1+...+xm|g1/.../gQ の両側公式で，r1,...,rn はモデル引数 model の右側に含まれるパラメータ，x1+...+xm はこれらのパラメータに対するランダム効果モデルを指定し，g1/.../gQ はグルーピング構造 (Q は 1 でも良く，その時は分離記号 / は不要)．ランダム効果公式は，もし多重のグルーピング水準があれば，全てのグルーピング水準に対して繰り返し適用される．(2) 両側公式で，形式 r1+...+rn ~ x1+..+xm，形式 r1 ~ x1+...+xm の両側公式のリスト (異なるパラメータに対しては異なるランダム効果モデルを持って良い)，一つの両側公式からなる pdMat オブジェクト，両側公式のリスト (つまり formula(random) が NULL でない値を持つ)，両側公式からなる pdMat オブジェクトのリスト，もしくは両側公式のリスト．この場合，グルーピング構造公式は groups で与えられるか，非線形混合効果モデルを当てはめたデータ (クラス groupedData を継承している必要) から誘導される．(3) 名前付きの公式リスト，公式のリスト，またはグルーピング因子を名前として持つ (2) におけるような pdMat オブジェクト．ネスティングの順序はリスト中の成分の順序と同じとされる．(4) reStruct オブジェクト．利用可能な pdMat クラスについては pdClasses のドキュメントを参照．既定では fixed で，全ての固定効果がランダム効果としても使われる

groups オプションの片側公式で，形式 ~g1 (一重ネスト) または~g1/.../gQ (多重ネスティング) を持ち，ランダム効果が変化するデータの分割を指示する．g1,...,gQ は data 中で因子として評価できる必要がある．多重ネスティングの場合，その順序は左から右 (つまり g1 が最初，g2 が次，等) へとなる

start オプションの数値ベクトル，もしくは固定効果とランダム効果に対する初期推定値のリスト．もし数値ベクトルを与えると，内部的にそれが成分 fixed となるリストに変換される．fixed 成分は，モデル関数がクラス selfStart を継承しない限り必要で，もし継承していれば初期値は関数 nlsList() の呼び出しで計算される．オプションの random 成分はランダム効果に対する初期値を指定するために使われ，行列か，グルーピング水準の長さに等しい行列リストからなるべきである．各行列は対応する水準のグループ数と同じ長さの行と，その水準のランダム効果数と同じ長さの列を持つべきである

correlation オプションの corStruct オブジェクトで級内相関構造を記述する．利用可能な corStruct クラスに付いては corClasses のドキュメントを見よ．既定値は NULL で，級内相関無しに対応する

weights オプションの varFunc オブジェクトか，級内異分散構造を記述する片側公式．公式で与えられた時は，固定分散重みに対応する varFixed への引数として使われる．利用可能な varFunc クラスの説明は varClasses のドキュメントを見よ．既定値は NULL で，等分散誤差に対応する

subset data の行のどの部分集合が当てはめに際して使われるべきかを指定すオプションの表現式．論理ベクトルや，どの番号の観測値を使うかを表す数ベクトルで

も，行ラベルの文字列ベクトルでも良い．既定では全ての観測値が使われる

method 文字列．もし "REML" なら，当てはめは制約付き対数尤度を最大化することにより行われる．もし既定の "ML" なら対数尤度を最大化する

na.action データ中の NA 値を処理する関数．既定である na.fail() はエラーメッセージを表示し，不完全な観測値があれば停止する

naPattern 表現式もしくは公式で，どの返り値が欠損しているとみなすべきかを指示する

control 推定アルゴリズムの制御値のリストで，関数 nlmeControl() が返す既定値を置き換える．既定では空リスト．

返り値： クラス nlme のオブジェクトで，非線形混合効果モデルの当てはめ結果を表す．print(), summary(), plot() といった総称的関数は当てはめ結果に対するメソッド関数を持つ．当てはめ結果の各成分については nlmeObject を見よ．関数 resid(), coef(), fitted(), fixed.effects() そして random.effects() を使って成分を取り出すことができる

関連： nlmeControl(), nlme.nlsList(), nlmeObject(), nlsList(), nlmeStruct(), pdClasses(), reStruct(), varFunc(), corClasses(), varClasses().

以下の example(nlme) の第一例は，組み込みデータセット Loblolly (loblolly pine tree (テーダ松) の成長データ) を使用している．このデータフレームは 3 変数 height (樹高, ft), age (樹齢, year), Seed (種の出処を示す順序付き因子) を持つ．総計 84 個体からなり，Seed 因子で 14 グループにグルーピングされる．当てはめ fm1 は関数 SSasymp() により，自己開始型漸近回帰モデル

$$\mathtt{Asym} + (\mathtt{R0} - \mathtt{Asym}) \exp(-e^{\mathtt{lrc}} \times \mathtt{input})$$

を最尤推定法で当てはめている．パラメータ Asym, R0, lrc のそれぞれに固定効果を想定し start 引数でそれぞれの初期推定値を指定，更に Asym にランダム効果を想定している．つまり各グループ毎の Asym 当てはめパラメータは，グループ間で共通の固定効果と，グループ毎に異なるランダム効果の和になる．

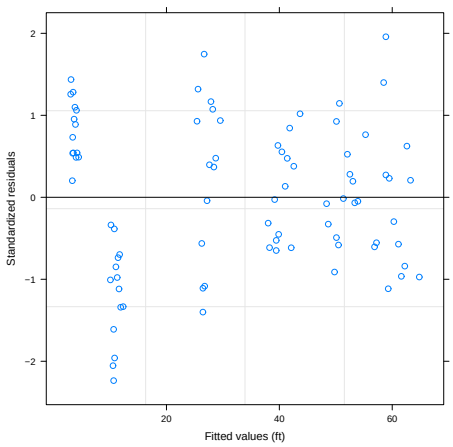
```
# loblolly pine tree(テーダ松) の成長データ Loblolly 使用
> fm1 <- nlme(height ~ SSasymp(age, Asym, R0, lrc), data = Loblolly,
            fixed = Asym + R0 + lrc ~ 1, random = Asym ~ 1,
            start = c(Asym = 103, R0 = -8.5, lrc = -3.3))
> summary(fm1)                                          # 当てはめ結果の要約
Nonlinear mixed-effects model fit by maximum likelihood
  Model: height ~ SSasymp(age, Asym, R0, lrc)      # 当てはめモデル
 Data: Loblolly
       AIC      BIC    logLik                      # AIC,BIC および対数尤度の値
  239.4856 251.6397 -114.7428

Random effects:                                       # Asym パラメータに定数値ランダム効果
 Formula: Asym ~ 1 | Seed                             # 因子 Seed でグルーピング
            Asym  Residual
StdDev: 3.650642 0.7188625

Fixed effects: Asym + R0 + lrc ~ 1             # 各パラメータの固定効果値と推定誤差標準偏差
         Value Std.Error DF   t-value p-value
Asym 101.44960 2.4616951 68  41.21128       0
R0    -8.62733 0.3179505 68 -27.13420       0
lrc   -3.23375 0.0342702 68 -94.36052       0
```

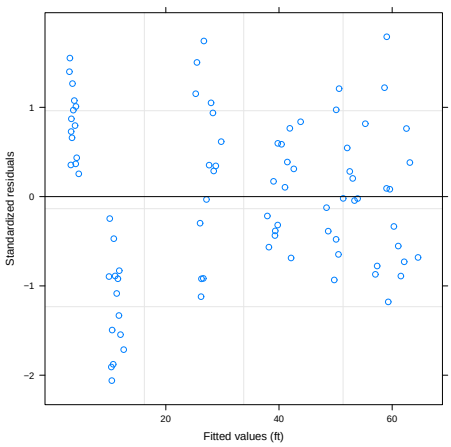
```
 Correlation:
    Asym   R0                                    # パラメータ固定効果間の相関推定値
R0   0.704
lrc -0.908 -0.827

Standardized Within-Group Residuals:             # グループ毎の当てはめ残差の分布
        Min          Q1         Med          Q3         Max
-2.23601930 -0.62380854  0.05917466  0.65727206  1.95794425

Number of Observations: 84
Number of Groups: 14

> fixed.effects(fm1)                             # 固定効果推定値
      Asym         R0        lrc
101.449600  -8.627331  -3.233751
> random.effects(fm1)                       # グループ別の Asym パラメータのランダム効果の値
         Asym
329 -5.5654676
327 -5.0168202
(途中省略)
305  7.0963810

> coef(fm1)                                      # 固定効果とランダム効果を併せた推定値
         Asym       R0       lrc
329  95.88413 -8.62733 -3.233751
327  96.43278 -8.62733 -3.233751
(途中省略)
305 108.54598 -8.62733 -3.233751

> fitted(fm1)                                    # 当てはめ値
      301       301       301       301       301       301       303       303
 3.871331 11.419533 27.881056 41.398469 52.498319 61.612984  4.160977 11.884103
      303       303       303       303       305       305       305       305
28.727108 42.557775 53.914855 63.240745  4.438284 12.328880 29.537115 43.667692
(途中省略)
attr(,"label")
[1] "Fitted values (ft)"
> resid(fm1)                                     # 予測残差
        301         301         301         301         301         303
 0.63866901 -0.52953340  0.83894380  0.34153126  0.20168076 -0.69298379
        303         303         303         303         303         303
 0.38902294 -0.96410276  0.34289237  0.27222506 -0.03485522  0.14925548
(途中省略)
attr(,"label")
[1] "Residuals (ft)"

> plot(fm1)                                 # 当てはめ値と標準化残差のプロット（以下の図を参照）

# 新しいデータでの樹高の予測
> newLoblolly <- data.frame(age = c(5,10,15),Seed = rep(301,6))
# 固定効果のみ使った予測値と，Seed グループ 301 でのランダム効果も含めた予測値
> predict(fm1, newLoblolly, level = 0:1)         # 実際は predict.nlme() を使用
  Seed predict.fixed predict.Seed
1  301      11.05971     11.41953
2  301      27.22576     27.88106
3  301      40.50054     41.39847
```

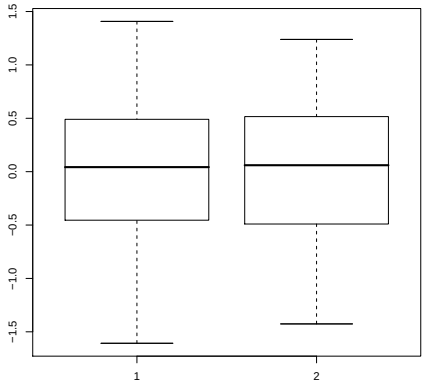
次の example(nlme) の第二例は，更にパラメータ lrc にもランダム効果を加えて，当てはめ結果を更新している．ランダム効果を増やすことによる当てはめの向上の判断は，AIC と BIC では相反する．

```
> fm2 <- update(fm1, random = pdDiag(Asym + lrc ~ 1))
> summary(fm2)                                   # 当てはめ結果の要約
Nonlinear mixed-effects model fit by maximum likelihood
  Model: height ~ SSasymp(age, Asym, R0, lrc)
 Data: Loblolly
       AIC      BIC    logLik
  238.9662 253.5511 -113.4831
```

```
Random effects:
 Formula: list(Asym ~ 1, lrc ~ 1)
 Level: Seed
 Structure: Diagonal
            Asym        lrc  Residual
StdDev: 2.806185 0.03449969 0.6920003

Fixed effects: Asym + R0 + lrc ~ 1
         Value Std.Error DF   t-value p-value
Asym 101.85205 2.3239828 68  43.82651       0
R0    -8.59039 0.3058441 68 -28.08747       0
lrc   -3.24011 0.0345017 68 -93.91167       0
 Correlation:
    Asym   R0
R0   0.727
lrc -0.902 -0.796

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-2.06072906 -0.69785679  0.08721706  0.73687722  1.79015782

Number of Observations: 84
Number of Groups: 14

# 各グループ別の Asym,lrc パラメータのランダム効果の値
> random.effects(fm2)
          Asym          lrc
329 -3.8077108 -0.024236568
327 -3.7521171 -0.017072489
325 -2.2850571  0.009364114
(途中省略)
303  2.8665067  0.023426354
305  3.6826150  0.045555074
> plot(fm2)                                  # 当てはめ値と標準化残差のプロット（以下の図を参照）
# 箱型図による残差の比較．優劣の比較はやはり微妙（以下の図を参照）
> boxplot(resid(fm1), resid(fm2))
```







非線形混合効果モデルの当てはめ

(上左) `fm1` の当てはめ値と標準化残差のプロット
(上右) `fm2` の当てはめ値と標準化残差のプロット
(下左) `fm1` と `fm2` の残差の平行箱型図

第 8 章
平滑化

R は幾つかのデータの平滑化関数を持つ．一部は既に時系列オブジェクトの平滑化関数として紹介された．ここでは特に散布図の平滑化を行う関数を紹介する．

8.1 核関数による平滑化

8.1.1 核関数を用いた平滑化 ksmooth()

ksmooth() は Nadaraya-Watson による核関数を用いた回帰平滑化を行う．(この関数は S との互換性のためだけに移植されている．より良い核関数平滑化関数が他のパッケージ中にある．)

書式：

```
ksmooth(x, y, kernel = c("box", "normal"), bandwidth = 0.5,
        range.x = range(x), n.points = max(100, length(x)), x.points)
```

引数：

x　入力 x 値

y　入力 y 値

kernel　使われる核関数

bandwidth　バンド幅．核関数は (密度関数と見たときの) 上下四分位数が幅 -0.25*bandwidth 位置にあるようにスケール化される

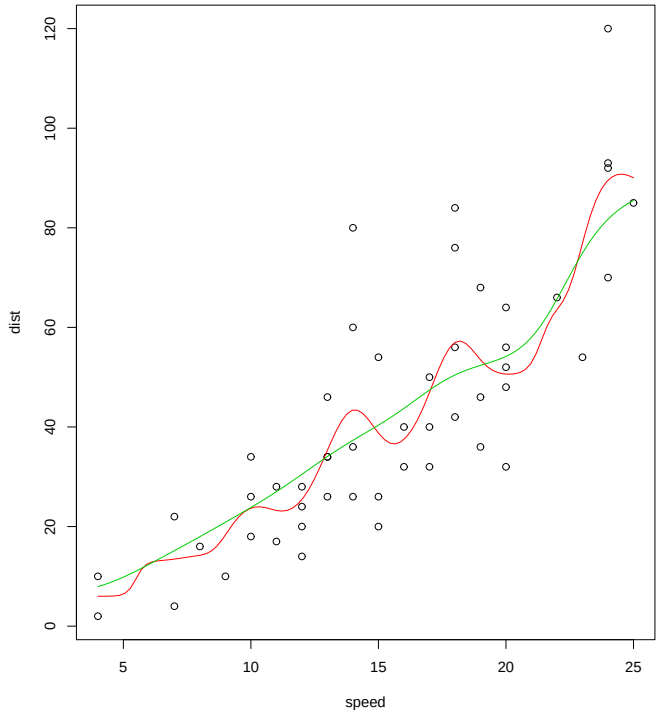
range.x　出力でカバーされる点の範囲

n.points　当てはめを評価する点の数

x.points　滑らかな当てはめを評価する点の数．もし与えられないと，n.points は range.x を一様にカバーするように選ばれる

返り値：　次の成分を持つリスト：x: 平滑化されたあてはめが評価された値．昇順であることが保証される y: x に対応する当てはめ値

```
> with(cars, {    # 車の速度と停止距離データ cars 使用．データ中の変数を展開した環境中で作業
      plot(speed, dist)
      lines(ksmooth(speed, dist, "normal", bandwidth=2), col=2)
      lines(ksmooth(speed, dist, "normal", bandwidth=5), col=3)
     })
```



車の速度と停止距離の散布図の `ksmooth()` による (バンド幅を変えた 2 種類の) 平滑化

8.2　多項式の局所的当てはめによる平滑化

8.2.1　散布図平滑化 `lowess()`

`lowess()` は LOWESS 平滑に対する計算を実行する．`lowess89` は平滑結果の座標である `x` と `y` を成分に持つリストを返す．平滑結果は `lines()` 関数で元の散布図プロットに描き加えることができる．

書式：

```
lowess(x, y=NULL, f=2/3, iter=3, delta=0.01*diff(range(xy$x[o])))
```

引数：

x, y　散布図中の点の座標を与えるベクトル．または，単一のプロット構造を与えることができる

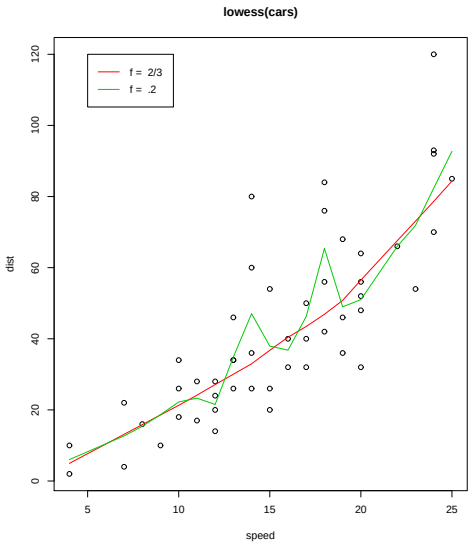
f　平滑幅．これは各点での平滑化に影響を及ぼす点の比率を与える．大きな値はより滑らかな結果を与える

iter　頑健化のために実行される繰り返し数． iter を小さくすると lowess() は高速になる

delta　互いに距離 delta 以内にある x の値は lowess() からの出力では一つの値に置き換えられる．既定値は x の範囲の 1/100

関連：loess() は lowess() のモデル式に基づいた新しいバージョンで，異なった既定動作を持つ．

```
> plot(cars, main = "lowess(cars)")          # 車速と停止距離データ cars 使用（次の図を参照）
> lines(lowess(cars), col = 2)
> lines(lowess(cars, f=.2), col = 3)
> legend(5, 120, c(paste("f = ", c("2/3", ".2"))), lty=1, col=2:3)
```



車の速度と停止距離の散布図の lowess() による (平滑幅 f を変えた 2 種類の) 平滑化

8.2.2　散布図と Loess 平滑化曲線の同時プロット scatter.smooth()

scatter.smooth() は散布図を描き，それに loess() による平滑化曲線を上描きする．

書式：

```
scatter.smooth(x, y, span=2/3, degree=1,
               family=c("symmetric", "gaussian"),
               xlab=deparse(substitute(x)),
               ylab=deparse(substitute(y)),
               ylim=range(y, prediction$y), evaluation=50, ...)
```

```
loess.smooth(x, y, span=2/3, degree=1,
             family=c("symmetric","gaussian"), evaluation=50, ...)
```

引数：

x　散布図の x 座標

y　散布図の y 座標

span　loess() に対する平滑化パラメータ

degree　使われる局所多項式の次数

family　もし "gaussian" なら当てはめは最小自乗法による．もし "symmetric" なら (re-descending)M 推定法が使われる

xlab　x 座標に対するラベル

ylab　y 座標に対するラベル

ylim　プロットの y 座標範囲

evaluation　滑らかな曲線が評価される点の数

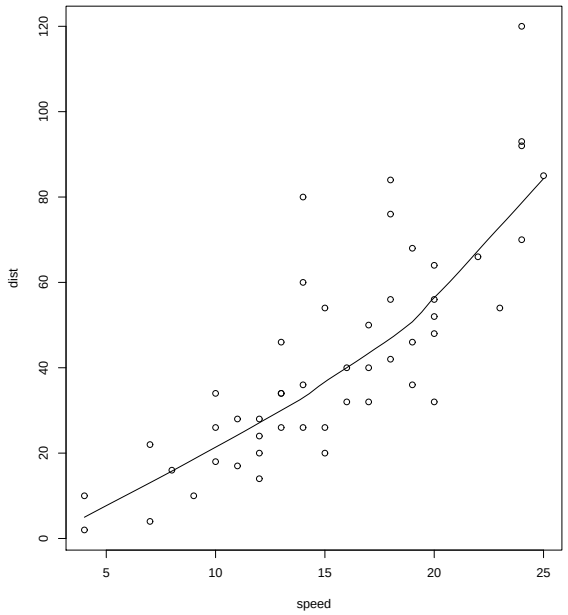
...　作図パラメータ

返り値： 返り値は scatter.smooth() に対しては無い (作図のみ)．loess.smooth() に対しては二つの成分 x (評価点の格子) と y (格子点での平滑化値) を持つリスト

loess.smooth() は x の範囲を覆う evaluation 個の等間隔な loess() 関数で平滑化曲線を評価する補助関数である．

関連： loess()．

```
# cars データ（車の速度と停止時間）中の変数を展開した一次環境中で実行
> with(data = cars, scatter.smooth(speed, dist))
```



scatter.smooth() による散布図と，loess() による平滑化曲線の同時プロット

8.3　スプライン関数当てはめによる平滑化

8.3.1　スプライン関数当てはめによる予測 predict.smooth.spline()

predict.smooth.spline() はスプライン関数当てはめによる新しい点での予測を行う．予測は元のデータ範囲外では直線になる．

書式： # クラス "smooth.spline" に対する S3 メソッド

predict(object, x, deriv=0, ...)

引数：

object　smooth.spline() による当てはめ結果

x　x の新しい値

deriv　整数．要求される導関数の階数

...　他のメソッドへ (から) 引き渡される追加引数

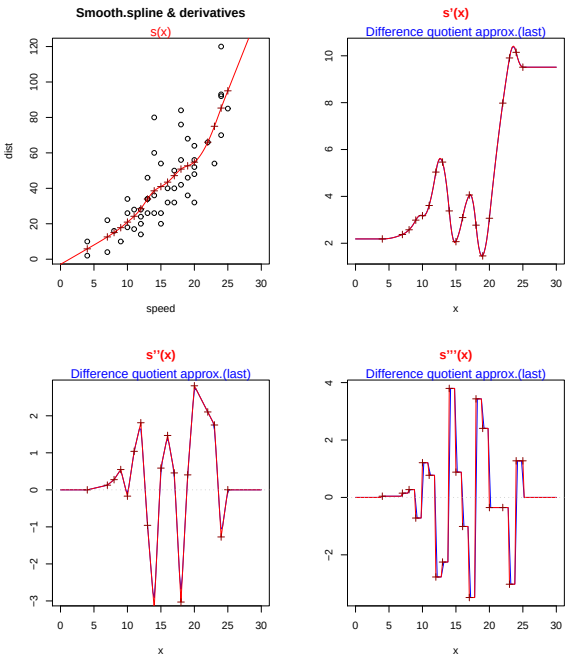
返り値：　次の成分を持つリスト：

x　入力された x

y　x に於ける当てはめ値または導関数値

関連： smooth.spline().

```
# スプライン関数による予測例 (次の図を参照)
> attach(cars)                                    # cars データ中の変数を展開
> cars.spl <- smooth.spline(speed,dist,df=6.4)    # スプライン関数当てはめ

# 階差による導関数値の近似値を計算する関数 (導関数値の正当性をチェック用)
> diff.quot <- function(x,y) {
   n <- length(x); i1 <- 1:2; i2 <- (n-1):n
   c(diff(y[i1]) / diff(x[i1]), (y[-i1] - y[-i2]) / (x[-i1] - x[-i2]),
     diff(y[i2]) / diff(x[i2]))}

# 予測値・導関数値を計算する x 座標
> xx <- unique(sort(c(seq(0,30, by = .2), kn <- unique(speed))))
> i.kn <- match(kn, xx)                           # xx 中のノットの添字
> op <- par(mfrow = c(2,2))                       # 画面を 2x2 分割
# 元データの散布図を丸印で描く
> plot(speed,dist,xlim=range(xx),main="Smooth.spline & derivatives")
# xx でのスプライン予測値を線で上書き
> lines(pp <- predict(cars.spl, xx), col = "red")
# xx でのスプライン予測値を + 印で上書き
> points(kn, pp$y[i.kn], pch = 3, col="dark red")
> mtext("s(x)", col = "red")                      # 赤色で"s(x)"と注釈テキスト書込み

# 当てはめスプライン関数の 3 階までの導関数を求め，それぞれプロット
> for(d in 1:3){
   n <- length(pp$x)
   # 階差近似で求めた導関数値を折れ線で描く
   plot(pp$x,diff.quot(pp$x,pp$y),type='l',xlab="x",ylab="",
        col = "blue", col.main = "red",
        main= paste("s",paste(rep("'",d), collapse=""),"(x)", sep=""))
   # 注釈テキストを青色で書き込む
   mtext("Difference quotient approx.(last)", col = "blue")
   # スプライン当てはめによる導関数値を赤い線で結ぶ
   lines(pp <- predict(cars.spl, xx, deriv = d), col = "black")
   # スプライン当てはめによる導関数値を暗赤色の + 印で描きこむ
   points(kn, pp$y[i.kn], pch = 3, col="dark red")
   abline(h=0, lty = 3, col = "gray")             # x 軸を灰色の点線で加える
 }
> detach(); par(op)                    # 後始末 (データ変数を抹消し，作図パラメータを元に戻す)
```



predict.smooth.spline() による車の速度と停止距離の散布図の平滑化予測値と 3 階までの導関数値

8.3.2 スプライン関数による平滑化 smooth.spline()

smooth.spline() は与えられたデータに 3 次の平滑化スプライン関数を当てはめる.

書式：

```
smooth.spline(x, y = NULL, w = NULL, df, spar = NULL,
              cv = FALSE, all.knots = FALSE, nknots = NULL,
              df.offset = 0, penalty = 1, control.spar = list())
```

引数：

x　説明変数の値を与えるベクトル．もしくは x と y の値を指定するリストか，2 行の行列

y　目的変数．もし y が無ければ， x 中に与えられていると仮定される

w　x と同じ長さの重みのオプションベクトル．既定では全て 1

df　要求される自由度相当数 (平滑化行列のトレース)

spar　平滑化パラメータ．典型的 (しかし必須ではない) には区間 (0,1] 中にある．当てはめ (ペナルティ付き対数尤度) 中の二階微分の 2 乗の積分の λ 係数は spar の単調増加関数である，以下の詳細を見よ

cv　TRUE なら通常の， FALSE ならば「一般化された」クロスバリデーション (GCV)

all.knots　もし TRUE なら， x 中の全ての異なる値が結節点として使われる．もし FALSE なら x[] のある部分集合，特に nknots 個の添字が 1:n 中に均一に位置するような x[j] が結節点として使われる．次の引数 nknots を参照せよ

nknots　all.knots=FALSE の時使われる結節点数を与える整数．既定では，x 中のユニークな値の数 n が 49 以上なら，これは n 未満になる

df.offset　GCV 基準中で自由度を df.offset 分だけ増加することを許す

penalty　GCV 基準中での自由度に対するペナルティ係数

control.spar オプションのリストで，平滑用パラメータ spar を計算する (つまり，これが与えられないか NULL の時) 際の根の解法を制御する名前付き成分からなる:
- low spar に対する下限．既定では -1.5
- high spar に対する上限．既定では +1.5
- tol 使用絶対精度．既定値 1e-4
- eps 使用相対精度．既定値 2e-8
- trace くり返し情報を与えるかどうか指示する論理値
- maxit 最大繰り返し回数を与える整数．既定では 500

返り値： クラス "smooth.spline" のオブジェクトで，次の成分を持つリスト：

x 昇順に並べられた x の異なった値．上の詳細を見よ

y x に対応する当てはめ値

w x のユニークな値に対して使われた重み

yin y のユニークな値に対して使われた y 値

lev 挺子比 (leverages)．平滑化行列の対角成分値

cv.crit クロスバリデーション得点．cv に応じて「一般化」か真値

pen.crit ペナルティ基準

crit 背後にある Fortran ルーチン sslvrg が与えた最小基準値

df 使用された等価自由度 (equivalent degrees of freedom)．真の df が 1 と 2 の間にあるときは極端に不正確になる可能性がある

spar 計算された，もしくは与えられた spar 値

lambda spar に対応する λ 値，上の解説を参照せよ

iparms 名前付きの整数ベクトル．..$ipars["iter"] は spar を計算する際に使われた繰り返し回数

fit predict.smooth.spline() が用いるリストで，以下の成分を持つ：
- knot 結節点列 (繰り返される境界結節点を含む)
- nk 係数の数，もしくは「適正」な結節点数 +2
- coef 使われたスプライン基底に対する係数
- min,range x の対応する量

call マッチした呼び出し式

spar は区間 [low, high] 中でだけ探索されることを注意しよう．

ベクトル x は少なくとも 4 つの異なる点を含む必要がある．ここで異なるとは，有効数字 6 桁に丸めた後で異なるという意味である．つまり x は unique(sort(signif(x, 6))) に変換され，y と w はそれにしたがってプールされる．

用いられる (spar の関数としての λ の計算は式は $\lambda = \mathtt{r} \times 256^{3\times \mathtt{spar}-1}$ であり，ここで $\mathtt{r} = \mathrm{tr}(X'WX)/\mathrm{tr}(\Sigma)$，$\Sigma$ は $\Sigma[i,j] = \int B''[i](t)B''[j](t)dt$ で与えられる行列，X は $X[i,j] = B[j](x[i])$ で与えられ，W は (トレースが元のデータ数 n になるようにスケール化された) 重みの対角行列，そして $B[k](\cdot)$ は k 番目の B-spline である．

これらの定義によれば $f_i = f(x_i)$，B-spline 基底表現は $f = Xc$ (つまり c はスプライン係数のベクトル)，ペナルティ付きの対数尤度は $L = (y-f)'W(y-f) + \lambda c'\Sigma c$，そして従って c は (リッジ回帰) $(X'WX + \lambda\Sigma)c = X'Wy$ の解である．

もし spar が与えられないか NULL なら，df の値が平滑化の次数を決定するのに使われ

る．もしどちらも与えられないと，「一時に一つを取り除く」クロスバリデーション[*1] (cv に応じて通常もしくは「一般化」) が λ を決定するのに使われる．上の関係式から，spar は $= s_0 + 0.0601 \log(\lambda)$ であり，これは S-plus の smooth.spline() での移植 (spar は λ に比例) とは意図的に異なる．R の λ 対数値スケールは，spar を線形に変化させることがより意味があるようにする．

しかしながら，現在のところ，約 -1 や -2 より小さな spar の値に対しては結果が非常に不安定になる可能性があることを注意しよう．同じことは，2 程度の値より大きな場合にも起こり得る．何をしているのか承知していない限り，そうした安全な範囲外に spar, low, high を設定することはしないように．

一般化クロスバリデーション法は x 中に重複する値があるときに正しく動作する．しかしながら，「一時に一つを取り除く」クロスバリデーションが重複値に対して何を意味するのかは曖昧である．内部コードは重複点グループを取り除く近似を利用している．そうした場合 cv=TRUE を避けるのが最善である．

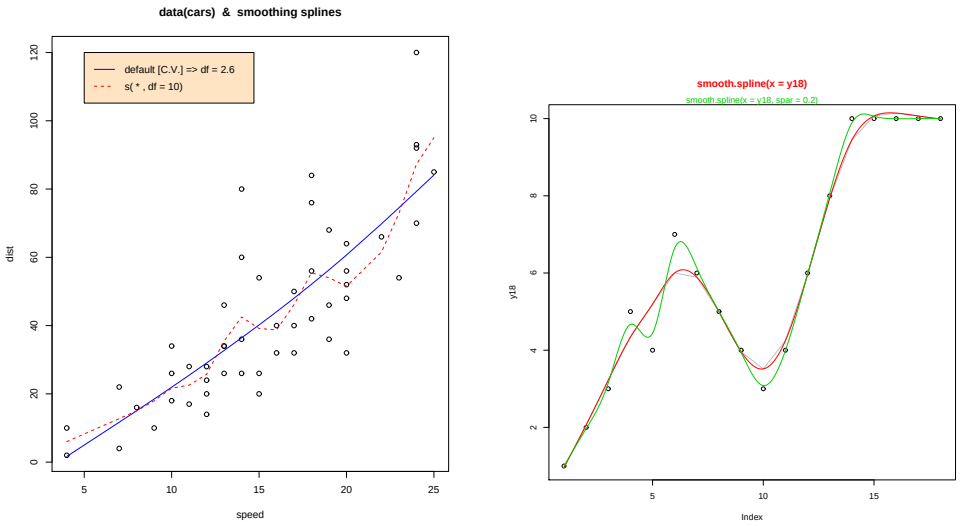
既定の all.knots=FALSE と nknots=NULL は $n > 49$ に対して n 個の結節点の代わりに $O(n^{0.2})$ 個の点だけを使う．これは速度とメモリ使用量を削減するが，R 1.5.1 以降では nk を結節点数として $O(nk) + O(n)$ 個のオーダーになっているため，劇的という程ではない．全てのユニークな x 値が使われないこの場合，結果は狭義の意味で平滑化スプラインではないが，しかし小さな平滑化パラメータ (もしくは大きな df) を使わない限り非常に近い．

関連：スプライン関数とその導関数を評価する predict.smooth.spline() 関数．

```
# cars データを用いた例．2 種類のスプライン平滑化結果の図示 (次の図を参照)
> attach(cars)
> plot(speed, dist, main = "data(cars) & smoothing splines")
> (cars.spl <- smooth.spline(speed, dist))        # スプライン平滑化結果の要約
Call:
smooth.spline(x = speed, y = dist)
Smoothing Parameter  spar= 0.7801305  lambda= 0.1112206 (11 iterations)
Equivalent Degrees of Freedom (Df): 2.635278
Penalized Criterion: 4337.638
GCV: 244.1044

# 元データとスプライン平滑化を折れ線で結ぶ．この例は重複点を持つので cv=TRUE を避けた方が良い
> lines(cars.spl, col = "blue")
> lines(smooth.spline(speed, dist, df=10), lty=2, col = "red")
> legend(5,120,c(paste("default [C.V.] => df =",round(cars.spl$df,1)),
         "s( * , df = 10)"), col=c("blue","red"), lty=1:2, bg='bisque')
> detach(cars)
```

```
# 人工的な例．2 種類のスプライン平滑化結果の図示 (次の図を参照)
> y18 <- c(1:3,5,4,7:3,2*(2:5),rep(10,4))
> xx  <- seq(1,length(y18), len=201)
> (s2  <- smooth.spline(y18))                     # GCV
> (s02 <- smooth.spline(y18, spar = 0.2))
> plot(y18, main=deparse(s2$call), col.main=2)
> lines(s2, col = "gray"); lines(predict(s2, xx), col = 2)
> lines(predict(s02, xx), col=3); mtext(deparse(s02$call), col = 3)
```

[*1] leave-one-out cross validation. n 個のデータから検査用に一つを除き，残りの $n-1$ 個で構成した予測式で予測誤差を求めることを，全てのデータに渡って繰り返し，総合的な平均予測誤差を求めることで，予測方式の善し悪しを判断する．



(左) 車の速度と停止時間データの散布図とスプライン平滑化結果．(右) 人工的な例．二種類のスプライン平滑化結果の図示

8.4　移動直線平滑化，Friedman の supersmoother 法

8.4.1　Friedman の SuperSmoother supsmu()

`supsmu()` は Friedman の super smoother 法を用いて `(x,y)` 値を平滑化する．

書式： `supsmu(x, y, wt, span = "cv", periodic = FALSE, bass = 0)`

引数：

`x`　平滑化される `x` 値

`y`　平滑化される `y` 値

`wt`　重み，既定では全て同じ

`span`　移動直線平滑法に対する間隔中の観測値の割合．`"cv"` ならば「一時に一つを取り去る」クロスバリデーションで割合を決める

`periodic`　もし `TRUE` ならば，`x` 値は `[0,1]` 中にあり，周期 1 とされる

`bass`　当てはめ曲線の滑らかさを制御する．最大 10 まで滑らかさがます

返り値：　次の成分を持つリスト：

`x`　重複値を除き昇順に並べられた入力値

`y`　当てはめ曲線上の対応する y 値

`supsmu()` は移動直線平滑化関数で，直線に対して 3 つの間隔を選ぶ．移動直線平滑化は対称で，予測値の両側に `k/2` 個のデータ点を持つ．ここで，`n` をデータ点の数とすると，`k` の値は `0.5*n`, `0.2*n` そして `0.05*n` である．もし `span` が指定されると，間隔 `span*n` の平滑化法だけが使われる．

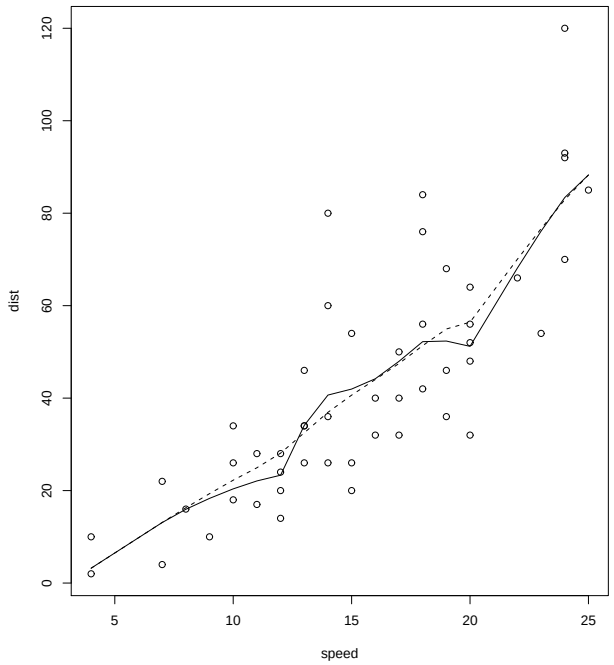
各予測に対して，3 種類の平滑化のうち最良のものがクロスバリデーションで選ばれる．最良の間隔はそれから移動直線平滑化法で平滑化され，最終的な予測値は線形補間で得られる．

使われている FORTRAN コードによれば「小さな標本 (`n < 40`) または `x` 値にかな

りの系列相関を持つ近い値があれば，予め間隔を指定した平滑化 (span > 0) を使うべきである」．常識的な間隔の値は 0.2 から 0.4 である．

関連：ppr().

```
# 2 種類の平滑化結果を散布図に描き加える（次の図を参照）
> data(cars)                                    # 車の速度と停止時間データ
> with(cars, {                                  # cars データ中の変数を展開した環境中で作業
     plot(speed, dist)                          # データの散布図を描く
     lines(supsmu(speed, dist))
     lines(supsmu(speed, dist, bass = 7), lty = 2) } )
```



supsum() による散布図の平滑化

8.5 移動中央値による平滑化

8.5.1 移動中央値平滑化 runmed()

runmed() は奇数スパンの移動中央値 (running median) を計算する．これは考えられる限り「最も頑健」な散布図平滑化である．効率化 (そして歴史的理由) のため，同じ結果を与える二つの方法を選ぶことができる．

書式：

```
runmed(x, k, endrule = c("median","keep","constant"),
       algorithm = NULL, print.level = 0)
```

引数：

x　数値ベクトル．平滑化される「従属」変数

k　中央値ウィンドの整数幅．奇数でなければならない．Turlach は k に対する既定値 1+2*min(floor((n-1)/2),ceiling(0.1n)) を持つ．孤立した外れ値を抹消する「最小の」頑健平滑化には k=3 を使おう

endrule データの先頭と末尾にある値をどう扱うかを指示する以下の文字列：
"keep" 両端の k2 個の値を保持する．ここで k2 はバンド幅の半分 floor(k/2)，つまり j= 1,...,k2 そして j=(n-k2+1),..,n に対して y[j]=x[j]
"constant" median(y[1:k2]) を先頭および末尾の値としてコピーし，平滑化された両端が定数値になるようにする
"median" 既定値．順に小さくなるバンド幅を用いた対称中央値を用いて，両端を平滑化する．しかし，一番端の値では Tukey の頑健な両端ルールが使われる．smoothEnds() を参照せよ
algorithm 文字列 ("Turlach" または "Stuetzle" に部分的にマッチする) か既定値の NULL で，どのアルゴリズムを使うかを指定する．既定の選択は n=length(x) と k に依存し，大規模なデータには"Turlach" が使われる
print.level 整数．アルゴリズムの途中経過報告の程度を指定する．普通のユーザは滅多に変更すべきではない

返り値： x と同じ長さのベクトルで，(奇数化された) k を含む属性 k を持つ

端点を除き，結果 y=runmed(x, k) は非常に効率的に計算された値，k=2*k2+1 とすると y[j]=median(x[(j-k2):(j+k2)]) である．二つのアルゴリズムは内部的に全く異なっている．"Turlach" は Turlach により移植された HNZdle-Steiger アルゴリズムである．木アルゴリズムが使われ，n <- length(x) としたとき計算量 O(n*log(k)) が保証され，漸近的に最良である．"Stuetzle" は (より古い) Stuetzle-Friedman の移植であり，平滑範囲ウィンドに一つデータが出入りする毎に中央値を更新する．これの計算量は O(n*k) であり，漸近的にはより遅いが，小さな k と n に対しては相当早い．

関連：eda パッケージの smooth() はその複合平滑化法として k=3 の移動中央値を使う．Tukey の端点ルールを移植した smoothEnds() は runmed(*, endrule="median") により既定で呼び出される．

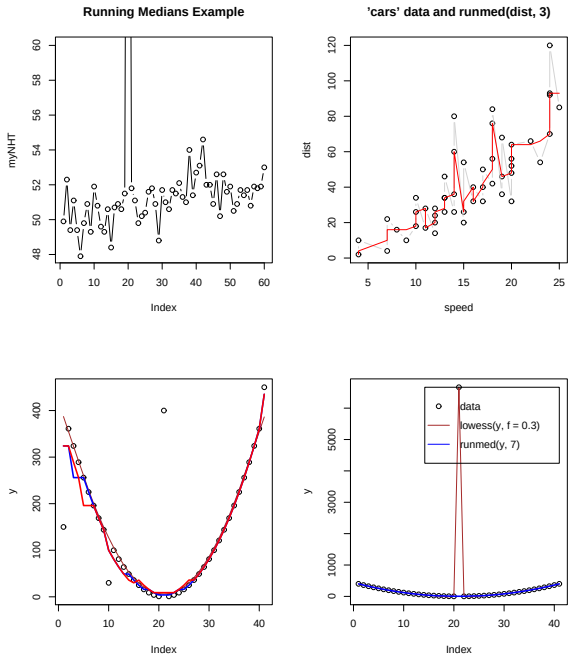
```
# 散布図に移動中央値平滑化結果を上描きする例二つ (次の図を参照)
> data(nhtemp)                                    # New Haven の年平均気温データ使用
> myNHT <- as.vector(nhtemp)                      # 時系列オブジェクトを単なるベクトルに変換
> myNHT[20] <- 2 * nhtemp[20]                     # 20 番目のデータ値を変更 (外れ値化)
# 散布図に移動中央値平滑化結果を上描き
> plot(myNHT, type="b", ylim=c(48,60), main="Running Medians Example")
```

```
> data(cars)                                      # 車の速度と停止時間データ
> plot(cars,main="'cars' data and runmed(dist,3)")# 散布図
> lines(cars, col = "light gray", type = "c")     # データを灰色の線で結ぶ
> with(cars,lines(speed,runmed(dist,k=3),col=2))  # 平滑化結果を上描き
```

```
# 2 次関数に幾つか外れ値を加えた例 (次の図を参照)
> y <- ys <- (-20:20)^2
> y [c(1,10,21,41)] <- c(150, 30, 400, 450)
> plot(y)                                         # 散布図
> lines(lowess(seq(y),y, f = .3), col = "brown")  # lowess 平滑化 (茶色の線)
> lines(runmed(y, 7), lwd=2, col = "blue")        # 幅 7 の runmed 平滑化 (青色の線)
> lines(runmed(y,11), lwd=2, col = "red")         # 幅 11 の runmed 平滑化 (赤色の線)
```

```
# Lowess は頑健でないことを示す例 (次の図を参照)
> y <- ys ; y[21] <- 6666 ; x <- seq(y)           # y[21] は外れ値
> col <- c("black", "brown","blue")               # 色名ベクトル
> plot(y, col=col[1])                             # 散布図
```

```
> lines(lowess(x,y, f = .3), col = col[2])        # lowess は外れ値に追随
> lines(runmed(y, 7), lwd=2, col = col[3])        # 移動中央値による平滑化
> legend(length(y),max(y),c("data","lowess(y,f=0.3)","runmed(y,7)"),
         xjust = 1, col = col, lty = c(0, 1,1), pch = c(1,NA,NA))
```



`runmed()` による散布図の平滑化

(左上) New Haven の年平均気温，外れ値を加えた例，幅 7
(右上) 車の速度と停止時間データ，幅 3
(左下) 2 次関数に幾つか外れ値を加えた例．lowess 平滑化と，幅 7,11 の runmed 平滑化
(右下) Lowess は外れ値に対し頑健でないことを示す例

8.5.2　Tukey の移動中央値平滑化 `smooth()`

`smooth()` は Tukey の移動中央値平滑化 (running median) を行う．

書式：

```
smooth(x, kind = c("3RS3R", "3RSS", "3RSR", "3R", "3", "S"),
       twiceit = FALSE, endrule = "Tukey", do.ends = FALSE)
```

引数：

`x`　ベクトルか時系列

`kind`　使用する手法を指示する文字列．既定値は `"3RS3R"`

`twiceit`　論理値．結果を二重化するかどうかを指示する．平滑化 `S(y)` の二重化とは `S(y)+S(y-S(y))`，つまり平滑化された残差を平滑値に加えることを意味する．これは偏りを減少させる (が分散を増加させる)

`endrule`　境界における平滑化に対する規則を指示する文字列．`"Tukey"` (既定値) か `"copy"`

`do.ends`　論理値．境界においてもタイの `3-splitting` を行うべきかどうかを指示する．これは `kind="S"` に対してだけ使われる

返り値：　クラス `"tukeysmooth"` のオブジェクト (`print` と `summary` メソッドを持つ) であり，追加の属性を持つ平滑値を含むベクトルか，時系列である

3 は長さ 3 の移動中央値に対する Tukey の省略記法である．3R は収束するまで 3 を繰り返すことを意味する．S は長さ 2 および 3 の水平な連続を分割することを意味する．したがって 3RS3R は 3R, S そして 3R を連結したものを意味する．3RSS も同様である．3RSR は最初に 3R，それから S と 3 を交互に収束するまで繰り返すことを意味する (まずい結果になることもある)．

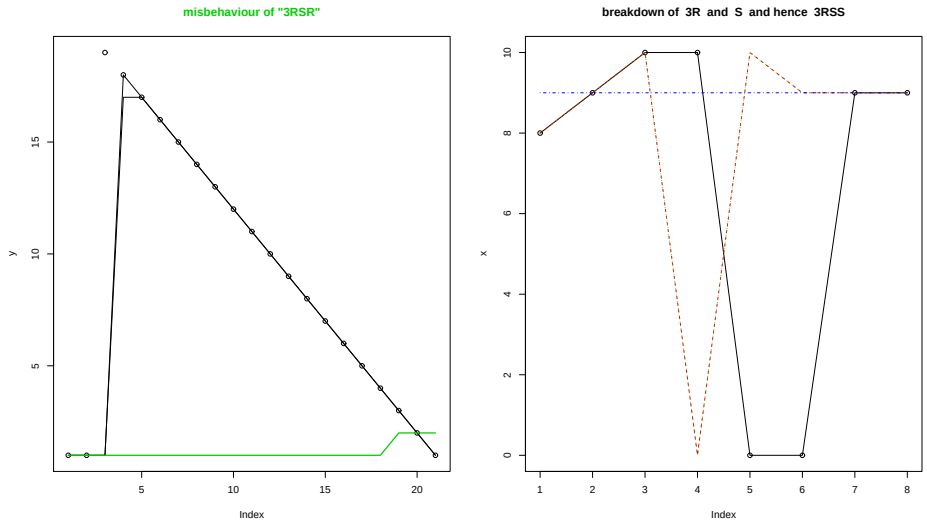
関連：lowess(), loess(), supsmu() そして smooth.spline().

```
# 平滑化機能のデモ demo(smooth) も見よ
> x1 <- c(4, 1, 3, 6, 6, 4, 1, 6, 2, 4, 2)        # とても人工的な例
> (x3R <- smooth(x1, "3R"))                        # "3"を 2 回繰り返す
3R Tukey smoother resulting from  smooth(x = x1, kind = "3R")
 used 2 iterations
 [1] 3 3 3 6 6 4 4 4 2 2 2
> smooth(x3R, kind = "S")
S Tukey smoother resulting from  smooth(x = x3R, kind = "S")
 changed
 [1] 3 3 3 3 4 4 4 4 2 2 2
```

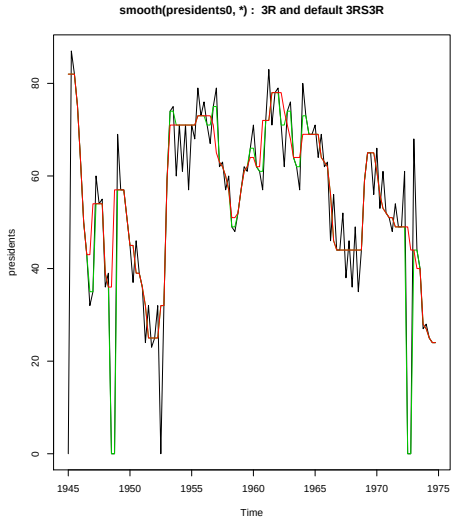
```
# "3RSR"がおかしくなる例 (次の図参照)
> sm.3RS <- function(x, ...)                       # "3R"ついで"S"を行う関数
              smooth(smooth(x, "3R", ...), "S", ...)
> y <- c(1,1, 19:1)
> plot(y, main = "misbehaviour of \"3RSR\"", col.main = 3)
> lines(sm.3RS(y))
> lines(smooth(y))
> lines(smooth(y, "3RSR"), col = 3, lwd = 2) # the horror
```

```
> x <- c(8:10,10, 0,0, 9,9)
> plot(x, main = "breakdown of  3R  and  S  and hence  3RSS")
> matlines(cbind(smooth(x,"3R"),smooth(x,"S"),
           smooth(x,"3RSS"),smooth(x)))
```

```
# 米国大統領指示率データ presidents 使用 (次の図を参照)
> presidents[is.na(presidents)] <- 0               # 欠損値を 0 に置き換えるまずい処置
> summary(sm3 <- smooth(presidents, "3R"))         # "3R"の結果の要約
3R Tukey smoother resulting from
 smooth(x = presidents, kind = "3R") ;  n = 120
 used 4 iterations                                # 収束までの繰り返し回数
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   # 結果の数値要約
    0.0    44.0    57.0    54.2    71.0    82.0

> summary(sm2 <- smooth(presidents,"3RSS"))        # "3RSS"の結果の要約
3RSS Tukey smoother resulting from
 smooth(x = presidents, kind = "3RSS") ;  n = 120
 used 5 iterations
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00   44.00   57.00   55.45   69.00   82.00

> summary(sm  <- smooth(presidents))               # 既定の"3RS3R"の結果の要約
3RS3R Tukey smoother resulting from
 smooth(x = presidents) ;  n = 120
 used 7 iterations
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  24.00   44.00   57.00   55.88   69.00   82.00

> plot(presidents,
       main="smooth(presidents0, *) : 3R and default 3RS3R")
> lines(sm3, col = 3, lwd = 1.5)                   # "3R"の結果の図示
> lines(sm, col = 2, lwd = 1.25)                   # 既定の"3RS3R"の結果の図示
```



(左) "3RSR" がおかしくなる例，(右) "3RSS" がおかしくなる例



米国大統領指示率データを "3R" と既定の "3RS3R" で平滑化

8.5.3　移動中央値に対する端点平滑化 smoothEnds()

smoothEnds() は，最端点で順次小さくなる中央値と Tukey の端点規則を使用し，ベクトル y の端点を平滑化する．

書式： smoothEnds(y, k = 3)

引数：

y　平滑化される従属変数 (ベクトル)
k　最大の中央値ウィンドの幅．奇数である必要

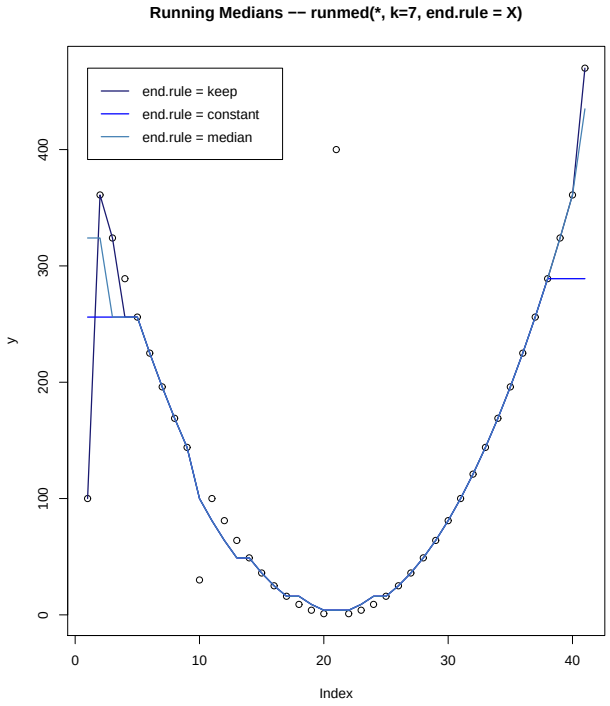
返り値：　平滑化された値のベクトルで，y と同じ長さ

smoothEnds は端点での平滑化だけを行うために使用される．つまり，最大でウィンド幅 k の半分よりも始点と終点に近い観測値だけが変更される．始点値と終点値は「Tukey

の端点規則」を用い計算される．つまり，sm[1]=median(y[1],sm[2],3*sm[2]-2*sm[3]) である．

関連： smoothEnds() を呼び出す runmed(*, end.rule = "median").

```
# 3 種類の端点処理で移動中央値平滑化実行（次の図を参照）
> y <- ys <- (-20:20)^2                                  # データ値（2 次関数）
> y [c(1,10,21,41)] <-  c(100, 30, 400, 470)      # 一部の値を変更
> s7k <- runmed(y,7, end = "keep")
> s7. <- runmed(y,7, end = "const")
> s7m <- runmed(y,7)                                     # 既定の"end=median"
> col3 <- c("midnightblue","blue","steelblue")
> plot(y, main = "Running Medians -- runmed(*, k=7, end.rule = X)")
> lines(ys, col = "light gray")

# 3 種類の結果を一度に描き，凡例を加える
> matlines(cbind(s7k,s7.,s7m),lwd=1.5,lty=1,col=col3)
> legend(1, 470, paste("end.rule",c("keep","constant","median"),sep=" = "),
         col = col3, lwd = 1.5, lty = 1)
```



移動中央値平滑化に対する
3 種類の端点処理結果

8.6　その他．パッケージ splines 中の関連関数

推奨パッケージ splines には次のようなスプライン関係の関数がある．

backSpline()	単調な逆スプライン関数
bs()	多項式スプライン関数に対する基底を生成する
interpSpline()	補間スプライン関数を生成する
ns89	自然なキュービックスプライン関数に対する基底行列を生成する
periodicSpline()	周期的な補間スプライン関数を生成する
polySpline()	局所的多項式スプライン表現
predict.bSpline()	スプライン関数を新しい x 値で評価する

predict.bs() スプライン基底を評価する
splineDesign() B-スプラインに対する計画行列
splineKnots() スプライン関数の結節点ベクトル
splineOrder() スプライン関数の次数を決定する

第 9 章
最適化

R は幾つかの汎用的最適化関数[*1] を持つ．これらは単独でも用いることができるが，他の R 関数の内部でも用いられている．特に関数 optim() は多変数目的関数の最適化を行う代表的な手法をオプションで選択でき，結果を比較したり，組み合わせて使うことができる．また，こうした最適化関数で必要になるグラディエント (1 階偏微分) 関数ベクトルや，ヘッセ (2 階偏微分) 関数行列を数式的に求めるための deriv() 関数が用意されている．

9.1 多変数目的関数の最適化

9.1.1 汎用的最適化関数 optim()

Nelder-Mead 法，準ニュートン法と共役勾配法アルゴリズムに基づく汎用的最適化プログラム．オプションとして矩形型制約下での最適化とシミュレーテッドアニーリング (SANN) 法も行う．

書式：

```
optim(par, fn, gr=NULL,
      method=c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN"),
      lower=-Inf, upper=Inf, control=list(), hessian=FALSE, ...)
```

引数：

par　最適化されるべきパラメータの初期値

fn　最初の引数として最適化されるパラメータベクトルを持つ，最小化 (または最大化) されるべき関数．スカラー値の返さなければならない

gr　"BFGS", "CG" そして "L-BFGS-B" 法に対するグラディエント (一階偏微分関数ベクトル) を返す関数．もし NULL なら有限階差近似が使われる． "SANN" 法に対してはこれは新しい候補点を生成する関数を指示し，もし NULL なら既定の正規マルコフ関数が使われる

method　使用される手法．詳細を見よ

lower,upper　"L-BFGS-B" 法に対する変数の下界，上界

control　制御パラメータのリスト．詳細を見よ

hessian　論理値．数値微分によるヘッセ行列 (2 階偏微分係数) を返すか？

[*1] R の非標準パッケージ中にも数理計画法を含む最適化関連関数が幾つもある．これらは CRAN にある Task View 中の「Optimization and Mathematical Programming」にまとめられている．

... fn と gr へ引き渡される追加引き数

返り値： 返り値は次の成分を持つリストである：

par 見出された最良のパラメータ値

value par に対応する fn の値

counts それぞれ fn と gr の呼出し回数を与える二つの整数値からなるベクトル．これは，もし必要とされた際の，ヘッセ行列を計算するために必要な呼出しは含まない．また，グラディエントに対する有限階差近似を計算するために必要な fn の呼出し回数を含まない

convergence 整数コード．0 は収束の成功を意味する．エラーコードは

1 最大繰りかえし数 maxit に達した

10 Nelder-Mead 単体が退化

51 "L-BFGS-B"法からの警告．詳細は message を参照

52 "L-BFGS-B"法からの警告．詳細は message を参照

message 最適化手法が返す任意の追加情報を与える文字列，もしくは NULL

hessian 引き数 hessian が真のときだけ存在する．見い出された解に於けるヘッセ行列の推定値を与える対称行列．これは，もし矩形型制約が与えられも，非制約問題に対するヘッセ行列である

既定ではこの関数は最小化を行うが，control$fnscale が負の値なら最大化を行う．既定の手法は Nelder & Mead の方法の移植で，関数値だけを使い頑健であるが，相対的に遅い．微分不可能な関数に対してもそれなりに使える．手法 "BFGS" は準ニュートン法 (variable metric algorithm と呼ばれることもある) であり，特に Broyden, Fletcher, Goldfarb そして Shanno が 1970 年代に独立に提案したものである．この手法は最適化を行う曲面の図を構成するために関数値とグラディエント値を使う．手法 "CG" は Fletcher & Reeves による共役勾配法であるが，Polak-Ribiere もしくは Beale-Sorenson 更新法を持つ．共役勾配法は一般に BFGS 法よりも破綻しやすいが，保管行列を使わないため相当大規模な最適化問題でも使える可能性がある．手法 "L-BFGS-B" は Byrd et al. に基づき，矩形型の拘束条件を許す．つまり，各変数は上界もしくは下界を与えることができる．初期値は制約条件を満たす必要がある．これはメモリ節約型の BFGS 法を使う．もし自明でない限界が与えられると，警告とともに，この方法が使われる．

手法 "SANN" は既定で Belisle が与えたシミュレーテッドアニーリング法の変種である．シミュレーテッドアニーリング法は確率的な大局的最適化法の一つである．これは関数値だけを使うが相対的に遅い．微分不可能な関数にも使える．ここでの移植は採択確率にメトロポリス関数を使う．既定では次の候補点は現在の温度に比例するスケールを持つ正規マルコフ関数である．もし新しい候補点を与える関数が提供されると，"SANN" 法は組合せ的最適化問題も解くことができる．温度パラメータは対数的冷却計画を使う．"SANN" 法は制御パラメータに本質的に依存することを注意しよう．これは汎用的な方法ではないが，極めてラフな曲面上の良い点を得るのに有効である．関数 fn は与えられた点で関数が評価できないときは NA もしくは Inf 値を返しても良いが，初期値では fn は計算可能な有限値を持つ必要がある (値が常に有限となる "L-BFGS-B" 法を除く)．optim() は再帰的に使うことができ，一変数でも多変数関数でも良い．

control 引き数はリストであり，成分として以下のどれを含んでも良い：

trace 非負整数値．もし正なら，最適化の過程に関する情報が報告される．より大きな値はより詳しい情報を与える． L-BFGS-B 法では情報のレベルに 6 段階ある

fnscale 最適化の過程で fn と gr の値に適用される全体的なスケーリング．もし負なら，最大化問題に変える．最適化は fn(par)/fnscale に対して行われる

parscale パラメータに対するスケーリング値．最適化は par/parscale に対して行われる．これらは，任意の要素の単位分の変更が，スケール化された値のほぼ単位分の変更を生じるという意味で比較可能である

ndeps グラディエントに対する有限階差近似に対する par/parscale スケールでのステップサイズのベクトル．既定値は 1e-3

maxit 最大繰りかえし数．既定値はグラディエントに基づく方法に対しては 100 で，Nelder-Mead 法に対しては 500．SANN 法に対しては maxit が総関数評価回数を与える．その他の停止規準は無い．既定値は 10000

abstol 絶対的な収束許容度．ゼロに近付く許容度であるため，非負関数に対してだけ意味がある

reltol 相対的な収束許容度．アルゴリズムはあるステップで値が reltol*(abs(val)+reltol) の割合以下に小さくできなければ停止する．既定値は sqrt(.Machine$double.eps) で，典型的には約 1e-8

alpha,beta,gamma Nelder-Mead 法に対するスケーリングパラメータ． alpha は reflection factor (既定値 1.0)， beta は contraction factor (既定値 0.5)，そして gamma は expansion factor (既定値 2.0)

REPORT control$trace が正のときの BFGS, L-BFGS-B 法の報告頻度．既定値は繰りかえし 10 回毎

type 共役勾配法用．Fletcher-Reeves 更新に対しては値 1，Polak-Ribiere 更新に対しては 2，Beale-Sorenson 更新に対しては 3

lmm L-BFGS-B 法で保たれる BFGS 更新回数を与える数．既定値は 5

factr L-BFGS-B 法の収束の制御を助ける．目的関数の減少が機械イプシロンのこの倍数以下であれば収束と見なされる．既定値は 1e7 で，許容値は約 1e-8

pgtol L-BFGS-B 法の収束の制御を助ける．現在の探索方向に於けるグラディエントの射影に対する許容値である．既定値は 0 で，チェックは行われない

temp SANN 法を制御する．冷却法に対する初期値 (既定値は 10)

tmax SANN 法に対する各温度に於ける関数評価回数 (既定値は 10)

注意： optim() は一次元の par に対しても使えるが，既定の手法はうまく動作しない (そして警告がでる)．代わりに optimize() を使おう．

関連： nlm(), optimize(), constrOptim().

```
# Rosenbrock のバナナ関数の最小値を求める
> fr <- function(x) {x1 <- x[1]; x2 <- x[2]
                     100*(x2-x1*x1)^2+(1-x1)^2 }
> grr <- function(x) {                          # fr のグラディエント関数
          x1 <- x[1]; x2 <- x[2]
          c(-400*x1*(x2-x1*x1)-2*(1-x1), 200*(x2-x1*x1)) }
> optim(c(-1.2,1), fr)                # 初期パラメータ値 c(-1.2,1), 既定の Nelder-Mead 法使用
$par                                            # 収束パラメータ値
[1] 1.000260 1.000506
```

```
$value                                          # そのときの関数値
[1] 8.825241e-08
$counts                                         # 関数とグラディエントの評価回数
function gradient
     195       NA
$convergence                                    # 収束コード（無事収束）
[1] 0
$message                                        # その他のメッセージ（無し）
```

```
> optim(c(-1.2,1), fr, grr, method="BFGS")      # グラディエント関数を与えて"BFGS"法で実行
$par
[1] 1 1
$value
[1] 9.594955e-18
$counts
function gradient                               # 関数とグラディエントの評価回数
     110       43
$convergence
[1] 0
$message
NULL
```

```
# グラディエント関数を与えて"BFGS"法で実行，ヘッセ行列を返す
> optim(c(-1.2,1), fr, NULL, method="BFGS", hessian=TRUE)
$par
[1] 0.9998044 0.9996084
$value
[1] 3.827383e-08
$counts
function gradient
     127       38
$convergence
[1] 0
$message
NULL
$hessian                                        # 収束値でのヘッセ行列（正定値のはず）
          [,1]      [,2]
[1,]  801.6881 -399.9218
[2,] -399.9218  200.0000
```

```
> optim(c(-1.2,1), fr, grr, method="CG")        # 共役勾配法を使う
$par
[1] 1 1
$value
[1] 4.477649e-17
$counts
function gradient
     156       41
$convergence
[1] 0
$message
NULL
```

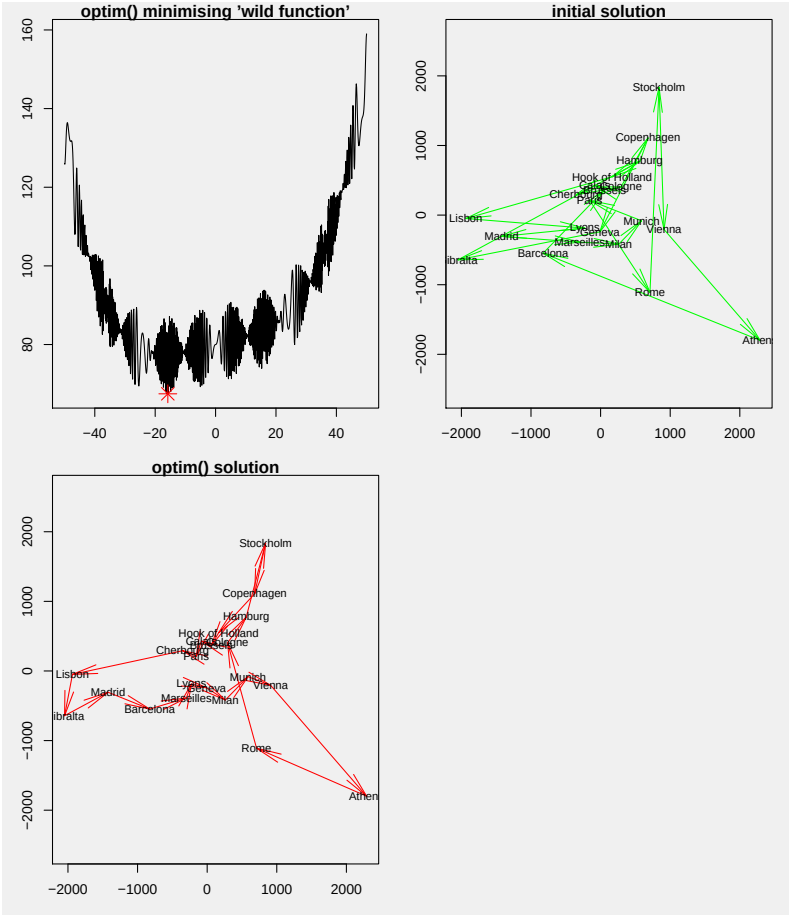
```
# 25 次元の矩形型拘束条件を与え L-BFGS-B 法で最小化
> flb <- function(x)
          {p <- length(x); sum(c(1,rep(4,p-1))*(x-c(1,x[-p])^2)^2)}
> optim(rep(3, 25), flb, NULL, "L-BFGS-B",
        lower=rep(2, 25), upper=rep(4, 25))     # par[24] は境界上には無い
$par
 [1] 2.00000 2.00000 2.00000 2.00000 2.00000 2.00000 2.00000 2.000000
 [9] 2.00000 2.00000 2.00000 2.00000 2.00000 2.00000 2.00000 2.000000
[17] 2.00000 2.00000 2.00000 2.00000 2.00000 2.00000 2.00000 2.109093
[25] 4.00000
$value
[1] 368.1059
$counts
function gradient
       6        6
$convergence
[1] 0
$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
# あるワイルドな関数の最小化（次の図を参照）
> fw <- function (x)
         10*sin(0.3*x)*sin(1.3*x^2) + 0.00001*x^4 + 0.2*x+80
> plot(fw, -50, 50, n=1000, main="optim() minimising 'wild function'") # グラフを描く
> res <- optim(50, fw, method="SANN",             # SANN 法で最小化
              control=list(maxit=20000, temp=20, parscale=20))
> res
$par
[1] -15.81476                                     # 大局的最小解は約-15.81515
$value
[1] 67.46905
$counts
function gradient
   20000       NA
$convergence
[1] 0
$message
NULL

> (r2 <- optim(res$par, fw, method="BFGS")) # 得られた解を初期値として BFGS 法で局所的に改良
$par
[1] -15.81515
$value
[1] 67.46773
$counts
function gradient
      16        3
$convergence
[1] 0
$message
NULL

> points(r2$par, r2$val, pch=8, col="red", cex=2)   # 最適解をグラフに上書き
```

```
# 組合せ的最適化問題（トラベリングセールスマン問題）を SANN 法で解く（次の図を参照）
# 乱数を使うので結果は実行毎に異なる可能性がある
> data(eurodist)                                  # ヨーロッパ主要都市間距離データ
> eurodistmat <- as.matrix(eurodist)              # 行列に変換
> distance <- function(sq) {                      # 目的関数（総旅行距離数）
               sq2 <- embed(sq, 2)
               return(sum(eurodistmat[cbind(sq2[,2],sq2[,1])])) }

# 現在の旅行ルートから新しい旅行ルートを生成する関数（ランダムに二都市を選び交換する関数）
> genseq <- function(sq) {
              idx <- seq(2, NROW(eurodistmat)-1, by=1)
              changepoints <- sample(idx, size=2, replace=FALSE)
              tmp <- sq[changepoints[1]]
              sq[changepoints[1]] <- sq[changepoints[2]]
              sq[changepoints[2]] <- tmp
              return(sq) }

> sq <- c(1,2:NROW(eurodistmat),1)                # 初期旅行ルート
> distance(sq)                                    # その時の総距離数
[1] 29625

> set.seed(2222) # 乱数種．良い解を素早く得るために選ばれた
> res <- optim(sq, distance, genseq, method="SANN",  # SANN 法を実行
              control = list(maxit=6000, temp=2000, trace=TRUE))
sann objective function values
initial       value 29625.000000                # 以下初期値と繰りかえし毎の値
iter     1000 value 14461.000000
(途中省略)
iter     5999 value 13552.000000
final         value 13552.000000
sann stopped after 5999 iterations              # 指定最大回数まで繰りかえして終了
> res
$par                                              # その時の旅行ルート
 [1]  1 19  6 10 20  7 11  3  4 18  5 12  9 14  2 15 13  8 16 17 21  1
$value                                          # 終了時の総距離数（最短総距離 12842 に近い）
[1] 13552
$counts
function gradient
    6000       NA
$convergence
```

```
[1] 0
$message
NULL
```

```
> loc <- cmdscale(eurodist)                          # 旅行ルートを図示
> rx <- range(x <- loc[,1])
> ry <- range(y <- -loc[,2])
> tspinit <- loc[sq,]; tspres <- loc[res$par,]
> s <- seq(NROW(tspres)-1)
> plot(x, y, type="n", asp=1, xlab="", ylab="",    # 最初の旅行ルートのグラフ
       main="initial solution of traveling salesman problem")
> arrows(tspinit[s,1], -tspinit[s,2], tspinit[s+1,1], -tspinit[s+1,2],
       angle=10, col="green")                        # 矢印を上書き
> text(x, y, names(eurodist), cex=0.8)               # 都市名を上書き
> plot(x, y, type="n", asp=1, xlab="", ylab="",    # SANN 法で求められた旅行ルートのグラフ
     main="optim() 'solving' traveling salesman problem")
> arrows(tspres[s,1], -tspres[s,2], tspres[s+1,1], -tspres[s+1,2],
       angle=10, col="red")
> text(x, y, names(eurodist), cex=0.8)
```



(上左) ワイルドな関数とその最小点 (星印). (上右) ヨーロッパ都市間の旅行ルート (初期値). (下左) ヨーロッパ都市間の旅行ルート (SANN 法による解)

9.1.2　汎用的最適化関数 nlm()

この関数はニュートン法タイプのアルゴリズムを用いて関数 f の最小化を行う. 以下の詳細を見よ.

書式：

```
nlm(f, p, hessian = FALSE, typsize=rep(1, length(p)), fscale=1,
    print.level = 0, ndigit=12, gradtol = 1e-6,
    stepmax = max(1000 * sqrt(sum((p/typsize)^2)), 1000),
    steptol = 1e-6, iterlim = 100, check.analyticals = TRUE, ...)
```

引数：

f　最小化されるべき関数．もし関数が属性 gradient，もしくは二つの属性 gradient と hessian の双方を持てば，これらはパラメータ更新の計算で使われる．さもなければ，数値微分が使われる． deriv() は gradient 属性を持つ適切な関数を返す．これは長さ p のベクトル引き数 (... 引き数で指定される他の引き数が続く) 関数でなければならない

p　最小化のための初期パラメータ値

hessian　もし TRUE なら，最小値に於ける f のヘッセ行列が返される

typsize　最小値に於ける各パラメータのサイズの推定値

fscale　最小値に於ける f のサイズの推定値

print.level　この引き数は最小化の過程での出力レベルを決定する．既定値は 0 で出力をしない．値 1 は最初と最後の詳細を出力，値 2 は全ての途中経過を出力

ndigit　関数 f 中の有効桁数

gradtol　正数で，アルゴリズムを終了するために十分なだけスケール化されたグラディエントが 0 に近いかどうかを許容度を与える．スケール化されたグラディエントは， f の各方向 p[i] への相対変化を，p[i] の相対変化で割ったものである

stepmax　スケール化されたステップ長の最大許容値を与える正数． stepmax は目的関数が桁溢れしたり，アルゴリズムがパラメータ空間の興味ある領域を見過ごしたり，またはアルゴリズムが発散するのを防止するために使われ．stepmax はこれらの最初の二つを防ぐのに十分なだけ小さくなるように選ばれなければならないが，予め予想される合理的なステップよりも大きくなるべきである

steptol　許容される最小の相対的ステップを与える正数

iterlim　プログラムが終了するまでに実行されるべき最大の繰りかえし数を与える正の整数

check.analyticals　もしそれらが与えられた際，解析的なグラディエントとヘッセ行列を，初期パラメータ値に於ける数値微分値と比較チェックすべきかどうかを指示する．グラディエントとヘッセ行列の誤った指定を検出するのに役立つ

... f への追加引き数

返り値：　次の成分を持つリスト：

minimum　f の推定最小値の値

estimate　f が最小値を達成する点

gradient　f の推定最小値に於けるグラディエント

hessian　f の推定最小値に於けるヘッセ行列 (必要とされたときだけ)

code　最適化過程がどのように終了したかを示す次の整数値:

1. 相対グラディエントがゼロに近い．おそらく解を与える
2. 許容値範囲内で繰りかえしが成功．おそらく解を与える
3. 最後の大局的なステップは estimate よりも小さな点を見付けるのに

4. 失敗した． estimate は関数の近似的局所最小値を与えるか， steptol が小さすぎた
5. 繰りかえしの限界を越えた
6. 最大のステップサイズ stepmax を引き続く 5 回越えた．関数が下に有界でないか，ある方向で有限の漸近値を持つか， stepmax が小さすぎる

iterations 実行された繰りかえし数

グラディエントとヘッセ行列が提供されるが，不正なモードと長さを与える場合，check.analyticals = TRUE (既定値) ならば無視され，警告がでる．ヘッセ行列はグラディエントが存在し健全性のチェックをパスしなければ，チェックもされない．オリジナルのソース中の三種類の利用可能な手法のうち，線状探索を行う手法 "1" だけが使われる．

関連： optim()．一次元の最適化には optimize()，根を見付けるには uniroot()．数式微分には deriv()．非線形回帰には nls() が勧められる．

```
> f <- function(x) sum((x-1:length(x))^2)          # 目的 (2 次) 関数
# 初期値 c(10,10) で最小化. f(x)=(x1-1)^2+(x2-2)^2 と見なされる
> res <- nlm(f, c(10,10))
> str(res)                                         # 最小化結果を要約形式で得る
List of 5                                          # 結果リスト
 $ minimum   : num 4.3e-26                         # 最小値 (数値的には 0)
 $ estimate  : num [1:2] 1 2                       # 最小値を与える (x1,x2)
 $ gradient  : num [1:2] 2.76e-13 -3.10e-13        # その時のグラディエント値
 $ code      : int 1                               # 収束コード (問題なく収束)
 $ iterations: int 2                       # 繰りかえし回数 2(ニュートン法は収束すれば速い)
```

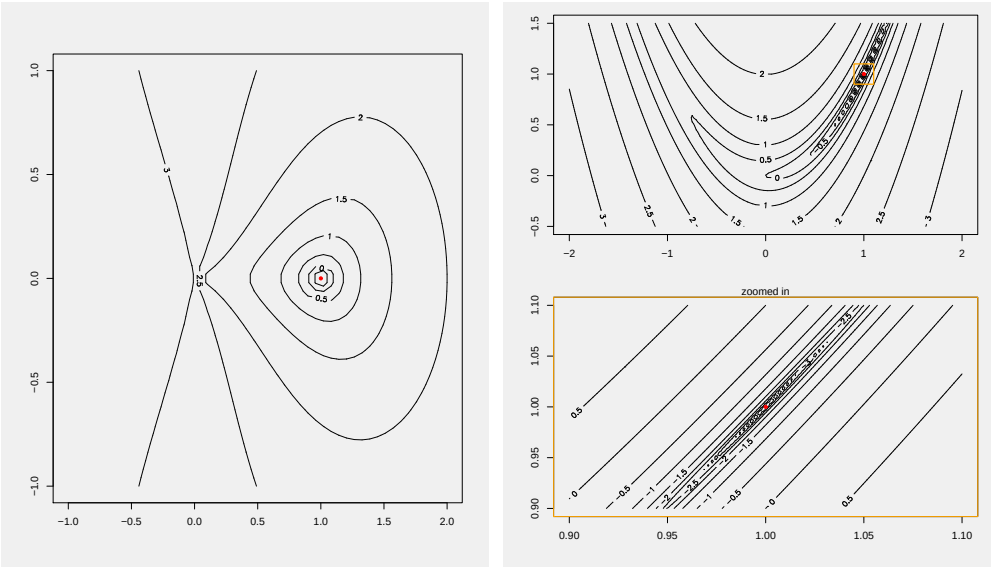
```
# グラディエント関数を与える例 (パラメータ a 付き). グラディエント関数は"gradient"属性として与える
> f <- function(x, a)
         { res <- sum((x-a)^2)
           attr(res, "gradient") <- 2*(x-a)        # グラディエント関数定義
           res }
> res <- nlm(f, c(10,10), a=c(3,5))
> str(res)                                         # 最小化結果を要約形式で得る
List of 5                                          # 結果リスト
 $ minimum   : num 0                               # 最小値は 0(グラディエントを与えた利益)
 $ estimate  : num [1:2] 3 5                       # 最小値を与える (x1,x2)
 $ gradient  : num [1:2] 0 0                       # その時のグラディエント (ぴたり c(0,0))
 $ code      : int 1                               # 収束コード (問題なく収束)
 $ iterations: int 1                               # 繰りかえし回数 1
```

```
# demo(nlm) 中の微分係数を使う例 (次の図を参照)
> theta <- function(x1, x2) atan(x2, x1)/(2 * pi) # 補助関数
> f <- function(x) {                               # 目的関数 (3 変数)
         f1 <- 10 * (x[3] - 10 * theta(x[1], x[2]))
         f2 <- 10 * (sqrt(x[1]^2 + x[2]^2) - 1); f3 <- x[3]
         return(f1^2 + f2^2 + f3^2) }
> x <- seq(-1, 2, len = 50); y <- seq(-1, 1, len = 50)
> z <- apply(as.matrix(expand.grid(x, y)), 1, function(x) f(c(x, 0)))
> contour(x, y, matrix(log10(z), 50, 50))          # 関数値の常用対数の等高線図を描く

# 最適化結果の要約表示
> str(nlm.f <- nlm(f, c(-1, 0, 0), hessian = TRUE, print = 0))
List of 6
 $ minimum   : num 1.24e-14                              # 最小値
 $ estimate  : num [1:3]  1.00e-00 3.07e-09 -6.06e-09 # 最適パラメータ
 $ gradient  : num [1:3] -3.76e-07 3.49e-06 -2.20e-06 # グラディエントベクトル
 $ hessian   : num [1:3, 1:3]  2.00e+02 -4.07e-02  9.77e-07 -4.07e-02  ... # ヘッセ行列
(3x3)
 $ code      : int 2                                    # 収束コード (=2 おそらく成功)
 $ iterations: int 27                                   # 繰りかえし回数 27 回
> points(rbind(nlm.f$estim[1:2]),col="red",pch=20)    # 最適パラメータを上書き
```

```
> fR <- function(x) {
          x1 <- x[1]; x2 <- x[2]
         100 * (x2 - x1 * x1)^2 + (1 - x1)^2}
> x <- seq(-2, 2, len = 100); y <- seq(-0.5, 1.5, len = 100)
> z <- fR(expand.grid(x, y))
> contour(x, y, matrix(log10(z), length(x)))
> str(nlm.f2 <- nlm(fR, c(-1.2, 1), hessian = TRUE))
List of 6
 $ minimum   : num 3.97e-12
 $ estimate  : num [1:2] 1 1
 $ gradient  : num [1:2] -6.54e-07  3.34e-07
 $ hessian   : num [1:2, 1:2]  802 -400 -400  200
 $ code      : int 1
 $ iterations: int 23
> points(rbind(nlm.f2$estim[1:2]), col = "red", pch = 20)
> rect(0.9, 0.9, 1.1, 1.1, border = "orange", lwd = 2)
> x <- y <- seq(0.9, 1.1, len = 100)
> z <- fR(expand.grid(x, y))
> contour(x, y, matrix(log10(z), length(x)))
> mtext("zoomed in")
> box(col = "orange")
> points(rbind(nlm.f2$estim[1:2]), col = "red", pch = 20)
```

```
# 微分演算子を使う例
> fd <- deriv(~100*(x2-x1*x1)^2+(1-x1)^2,c("x1", "x2")) # 数式微分
> fd
expression({                                          # その結果を表示
    .expr2 <- x2 - x1 * x1
    .expr5 <- 1 - x1
    .value <- 100 * .expr2^2 + .expr5^2
    .grad <- array(0, c(length(.value), 2), list(NULL, c("x1",
        "x2")))
    .grad[, "x1"] <- -(2 * .expr5 + 100 * (2 * ((x1 + x1) * .expr2)))
    .grad[, "x2"] <- 100 * (2 * .expr2)
    attr(.value, "gradient") <- .grad
    .value
})
> fgh <- function(x){                                 # グラディエント，ヘッシアン情報を持つ関数
    gr <- function(x1, x2) {                          # グラディエント関数ベクトル
        c(-400*x1*(x2 - x1*x1) - 2*(1-x1), 200*(x2 - x1*x1))
    }
    h <- function(x1, x2) {                           #ヘッシアン関数行列
        a11 <- 2 - 400*(x2 - x1*x1) + 800*x1*x1
        a21 <- -400*x1
        matrix(c(a11,a21,a21,200),2,2)
    }
    x1 <- x[1]; x2 <- x[2]
    res<- 100*(x2 - x1*x1)^2 + (1-x1)^2               # 関数本体
    attr(res, "gradient") <- gr(x1, x2)               # グラディエント属性指定
    attr(res, "hessian") <- h(x1, x2)                 # ヘッシアン属性指定
    return(res)
}
# 初期値 c(-1.2,1) を与えて最小化 (与えられたヘッシアン利用)
> str(nlm(fgh, c(-1.2, 1), hessian = TRUE))
List of 6
 $ minimum   : num 2.83
 $ estimate  : num [1:2] -0.679  0.471
 $ gradient  : num [1:2] -0.491  2.112
 $ hessian   : num [1:2, 1:2] 366 271 271 200
 $ code      : int 4
 $ iterations: int 100
```

```
> fdd <- function(x1, x2) {}                          # 空の関数の定義
> body(fdd) <- fd                                     # 関数本体を fd にする
> res <- nlm(function(x) fdd(x[1], x[2]), c(-1.2, 1), hessian = TRUE)
> str(res)
List of 6
 $ minimum   : num 1.18e-20
 $ estimate  : num [1:2] 1 1
 $ gradient  : num [1:2]  2.58e-09 -1.20e-09
 $ hessian   : num [1:2, 1:2]  802 -400 -400  200
 $ code      : int 1
 $ iterations: int 24
```



(左) 3 番目の例の関数等高線と最小値位置．(右上) 4 番目の例の関数等高線と最小値位置．(右下) その拡大図

9.1.3　汎用的最適化関数 nlminb()

PORT ルーチン[*2] を用いた非制約および制約付き最小化関数．

書式：

```
nlminb(start, objective, gradient = NULL, hessian = NULL, ...,
       scale = 1, control = list(), lower = -Inf, upper = Inf)
```

引数：

`start`　最適化されるべきパラメータの初期値を与える数値ベクトル

`objective`　最小化されるべき関数．スカラ値であり，`NA`, `Inf` 値を取っても良い．第一引数は最適化されるべきパラメータであり，初期値が `start` で与えられる．

`gradient`　`object` と同一の引数を持つオプションの関数で，`object` のグラディエントをその第一引数で評価する．`start` と同じ長さのベクトルを返すべきである

`hessian`　`object` と同一の引数を持つオプションの関数で，`object` のヘッセ行列をその第一引数で評価する．次数が `length(start)` の正方行列を返すべきであり，下三角部分だけが使われる

`...`　`objective` への追加引数で，最適化の途中で同一とされる

`scale`　PORT のドキュメント参照 (さもなければ無視)

`control`　制御パラメータのリスト．以下を参照

`lower, upper`　パラメータの下限と上限を与えるベクトルで，`start` と同じ長さになるまで繰り返される．もし与えられないと，全てのパラメータは制約無しとされる

返り値：　以下の成分を持つリスト：

`par`　見付かった最良のパラメータ値

[*2] URL:`http://netlib.bell-labs.com/cm/cs/cstr/153.pdf`.

objective par に対応する objective の値
convergence 整数コード．0 ならば成功裡に収束
message 追加の情報を与える文字列，もしくは NULL．詳細は PORT のドキュメントを見よ
iterations 必要だった繰り返し回数
evaluations 目的関数とグラディエント関数の評価回数

control 引数リスト中の名前付き成分とその既定値は以下の通り：

eval.max 目的関数の評価回数の最大値．既定では 200
iter.max 繰り返し回数の最大値．既定では 150.
trace 目的関数値とパラメータ値が繰り返し trace 回毎に出力される．既定値は 0 で一切出力されない
abs.tol 絶対許容値． 既定では 1e-20
rel.tol 相対許容値． 既定では 1e-10
x.tol X の許容値 (PORT のドキュメント参照)．既定では 1.5e-8
step.min ステップサイズの最小値．既定では 2.2e-14

```
# 目的関数は負の二項分布の対数分布関数の総和．最大化のためにマイナス化
> x <- rnbinom(100, mu = 10, size = 10)
> hdev <- function(par) {
            -sum(dnbinom(x, mu=par[1], size=par[2], log=TRUE))
          }
> nlminb(c(9, 12), hdev)                          # 初期値 c(9,20) で実行
$par                                              # 最適パラメータ値
[1] 9.720000 7.638535
$objective                                        # そのときの目的関数の値
[1] 292.2223
$convergence                                      # 成功裡に収束
[1] 0
$message
[1] "relative convergence (4)"
$iterations                                       # 反復回数 10 回
[1] 10
$evaluations                                      # 関数評価回数
function gradient                                 # グラディエント関数は数値微分で計算
      11       26
```

```
> nlminb(c(20, 20), hdev, lower = 0, upper = Inf) # 初期値を変えて再実行．同一の結果
$par
[1] 9.720000 7.638536
$objective
[1] 292.2223
$convergence
[1] 0
$message
[1] "relative convergence (4)"                    # PORT ルーチンのメッセージ
$iterations
[1] 16
$evaluations
function gradient
      24       39
```

```
# パラメータの下限値 0.001 を与えて実行．結果は同じだがより時間がかかる
> nlminb(c(20, 20), hdev, lower = 0.001, upper = Inf)
$par
[1] 9.720000 7.638536
$objective
[1] 292.2223
$convergence
```

```
[1] 0
$message
[1] "relative convergence (4)"
$iterations
[1] 16
$evaluations
function gradient
      24       39
```

```
> sumsq <- function(x, y) {sum((x-y)^2)}          # 差の 2 乗和の最小化例．最小値は 0
> y <- rep(1,5)
> x0 <- rnorm(length(y))                          # ランダムな初期値を与える
> nlminb(start = x0, sumsq, y = y)
$par
[1] 1 1 1 1 1
$objective                                        # 実数精度内で 0
[1] 1.043225e-18
$convergence
[1] 0
$message
[1] "X-convergence (3)"
$iterations
[1] 6
$evaluations
function gradient
       7       35
```

```
> y <- c(0, 2, 0, -2, 0)                          # 境界の外側にある y 値を与えた場合
> nlminb(start = x0, sumsq, lower = -1, upper = 1, y = y)
$par
[1]  4.241765e-08  1.000000e+00  2.078284e-07 -1.000000e+00 -3.267133e-07
$objective
[1] 2
$convergence
[1] 0
$message
[1] "relative convergence (4)"
$iterations
[1] 8
$evaluations
function gradient
      10       51
```

```
> sumsq.g <- function(x,y) 2*(x-y)                # グラディエント関数を使う
> nlminb(start=x0, sumsq, sumsq.g, lower=-1, upper=1, y=y)
$par
[1]  3.682221e-08  1.000000e+00 -4.405106e-09 -1.000000e+00 -2.837589e-09
$objective
[1] 2
$convergence
[1] 0
$message
[1] "relative convergence (4)"
$iterations
[1] 8
$evaluations
function gradient
      10        9
```

```
> sumsq.h <- function(x,y) diag(2, nrow = length(x)) # ヘッセ関数行列も使う
> nlminb(start=x0, sumsq, sumsq.g, sumsq.h, lower=-1, upper=1, y=y)
$par
[1] 2.465190e-32 1.000000e+00 1.540744e-33 -1.000000e+00 1.540744e-33
$objective
[1] 2
$convergence
[1] 0
$message
[1] "both X-convergence and relative convergence (5)"
$iterations
[1] 3
$evaluations
function gradient
```

```
       5        3
```

```
> fr <- function(x) {                          # optim 関数の参考例 (Rosenbrock のバナナ関数)
          x1 <- x[1]; x2 <- x[2]
          100*(x2-x1*x1)^2+(1-x1)^2 }
> grr <- function(x) {                               # fr のグラディエント関数
           x1 <- x[1]; x2 <- x[2]
           c(-400*x1*(x2-x1*x1)-2*(1-x1),
             200*(x2-x1*x1)) }
> nlminb(c(-1.2,1), fr)
$par
[1] 1 1
$objective
[1] 1.125615e-19                                   # 実数精度内で 0
$convergence
[1] 0
$message
[1] "X-convergence (3)"
$iterations
[1] 35
$evaluations
function gradient
      43       74

> (c(-1.2,1), fr, grr)                             # グラディエントを与えて再実行
$par
[1] 1 1
$objective
[1] 4.291498e-22
$convergence
[1] 0
$message
[1] "X-convergence (3)"
$iterations
[1] 35
$evaluations
function gradient
      43       36
```

```
# 25-次元矩形型拘束条件下 (下限 2, 上限 4) での最小化例. par[24] は境界上に無いことを注意
> flb <- function(x) { p <- length(x);
           sum(c(1, rep(4,p-1))*(x-c(1,x[-p])^2)^2) }
> nlminb(rep(3, 25), flb, lower=rep(2, 25), upper=rep(4, 25))
$par
 [1] 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000
 [9] 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000
[17] 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.000000 2.109093
[25] 4.000000
$objective
[1] 368.1059
$convergence
[1] 0
$message
[1] "relative convergence (4)"
$iterations
[1] 6
$evaluations
function gradient
      10      177
```

9.1.4 線形制約下で多変数関数の最大，最小値を求める constrOptim()

constrOptim() は適応的バリヤアルゴリズムを用いて，線形制約下での関数最小化を行う．

書式：

```
constrOptim(theta, f, grad, ui, ci, mu = 1e-04, control = list(),
```

```
            method = if(is.null(grad)) "Nelder-Mead" else "BFGS",
            outer.iterations = 100, outer.eps = 1e-05, ...)
```

引数：

theta 初期値．実行可能な領域内にある必要がある

f 最小化すべき関数

grad f のグラディエント

ui 制約 (以下を参照)

ci 制約 (以下を参照)

mu (小さな) 調整用パラメータ

control optim() に引き渡される制御パラメータリスト

method optim() に引き渡される手法名

outer.iterations バリヤアルゴリズムの繰りかえし数

outer.eps バリヤアルゴリズムの相対的収束規準

... optim() に引き渡されるその他のパラメータ

返り値： optim() と同様であるが，次の二つの追加成分を持つ．barrier.value は最適値でのバリヤ関数の値を与え，outer.iterations は外部の繰り返し (optim() 呼び出し) 回数を与える

実行可能な領域は ui %*% theta - ci >= 0 で定義される．初期値は実行可能領域内部になければならないが，最小値は境界上にあるかも知れない．

対数的バリヤが拘束条件を強制するために加えられ，それから optim() が呼び出される．バリヤ関数は目的関数が各繰り返し毎に減少するように選ばれる．実行可能領域の内部にある最小値は典型的に速やかに見出されるが，境界上の最小値には相当の外部繰り返しが必要になる．

調整用パラメータ mu はバリヤ項に掛けられる．その正確な値はしばしば相対的に重要ではない．mu が増加するにつれ，拡大目的関数は本来の目的関数に近付くが，同時に実行可能領域の境界近くではより滑らかでなくなる．

目的関数が無限大の値を取ることを許す optim() の任意の手法を使うことができる (現在 "L-BFGS-B" 法を除く全ての手法)．method="Nelder-Mead" を除けばグラディエント関数を与える必要がある．

optim() と同様に，既定動作は最小化であり，最大化は control$fnscale を 負の値に設定することで可能になる．

関連：optim()，特に矩形型拘束条件下の最的化を行う method="L-BGFS-B".

```
# optim() の例で使われている Rosenbrock のバナナ関数
> fr <- function(x) { x1 <- x[1]; x2 <- x[2]
                      100 * (x2 - x1 * x1)^2 + (1 - x1)^2 }
> grr <- function(x) {                          # fr のグラディエント
          x1 <- x[1]; x2 <- x[2]
          c(-400*x1*(x2-x1*x1)-2*(1-x1), 200*(x2-x1*x1)) }
> constrOptim(c(-1.2,0.9), fr, grr,             # 矩形型拘束条件，最適値は境界上にある
              ui=rbind(c(-1,0),c(0,-1)), ci=c(-1,-1))
$par
[1] 0.9999761 0.9999522                         # 最適パラメータ値
$value
```

```
[1] 5.708627e-10                                    # その時の目的関数値
$counts
function gradient
       6        1                                   # 目的関数とそのグラディエントの評価回数
$convergence
[1] 0                                               # 収束コード（無事収束）
$message
NULL                                                # 追加メッセージ無し
$outer.iterations
[1] 14                                              # optim() 関数の呼び出し回数
$barrier.value
[1] -0.0001999198                                   # 収束時のバリヤ関数の値
```

```
> constrOptim(c(.5,0), fr, grr,                     # 線形拘束 x<=0.9,y-x>0.1 を与える
              ui=rbind(c(-1,0),c(1,-1)), ci=c(-0.9,0.1))xs
$par
[1] 0.8891335 0.7891335
$value
[1] 0.01249441
$counts
function gradient
      18        1
$convergence
[1] 0
$message
NULL
$outer.iterations
[1] 5
$barrier.value
[1] -7.399944e-05
```

```
# 線形，2 次プログラミング問題を解く（良い初期値が必要）
# ライブラリ'quadprog' 中の example(solve.QP) より（導関数無し）
> fQP <- function(b) {-sum(c(0,5,0)*b)+0.5*sum(b*b)}  # 2 次目的関数
> Amat <- matrix(c(-4, -3, 0, 2, 1, 0, 0, -2, 1), 3, 3)
> bvec <- c(-8, 2, 0)
> constrOptim(c(2,-1,-1), fQP, NULL, ui=t(Amat), ci=bvec)
$par
[1] 0.4761374 1.0477253 2.0954507
$value
[1] -2.380952
$counts
function gradient
     252       NA
$convergence
[1] 0
$message
NULL
$outer.iterations
[1] 5
$barrier.value
[1] -0.0006243786
```

```
> gQP <- function(b) {-c(0,5,0)+b}                  # 導関数を与えて解く
> constrOptim(c(2,-1,-1), fQP, gQP, ui=t(Amat), ci=bvec)
$par
[1] 0.4761908 1.0476188 2.0952376
$value
[1] -2.380952
$counts
function gradient
      21        1
$convergence
[1] 0
$message
NULL
$outer.iterations
[1] 5
$barrier.value
[1] -0.0006243894
```

```
> hQP <- function(b) sum(c(0,5,0)*b)-0.5*sum(b*b) # 最大化の例．目的関数
> constrOptim(c(2,-1,-1), hQP, NULL, ui=t(Amat),
```

```
              ci=bvec, control=list(fnscale=-1))  # 最大化を指示
$par
[1] 0.4763538 1.0477181 2.0954838
$value
[1] 2.380751
$counts
function gradient
     300       NA
$convergence
[1] 0
$message
NULL
$outer.iterations
[1] 1
$barrier.value
[1] -0.001142111
```

9.1.5 一変数関数の最大，最小値を求める optimize()

optimize() は関数 f の第一引数に関する最大・最小値を区間 lower から upper の間で求める．optimise() は optimize() の別名である．

書式：

```
optimize(f = , interval = , lower = min(interval),
         upper = max(interval), maximum = FALSE,
         tol = .Machine$double.eps^0.25, ...)
optimise(f = , interval = , lower = min(interval),
         upper = max(interval), maximum = FALSE,
         tol = .Machine$double.eps^0.25, ...)
```

引数：

f	最適化される関数．この関数は maximum 引数の値に応じて，その第一引数に関して最小化もしくは最大化される
interval	最小値を探す区間の端点を含むベクトル
lower	探索区間の下側端点
upper	探索区間の上側端点
maximum	論理値．最大化するか最小化 (既定値) するか？
tol	要求される精度
...	fに対する追加引数

返り値： 成分 minimum (もしくは成分 maximum) と objective を持つリストで，最小値もしくは最大値の座標とそこでの関数値からなる

使用されている手法は黄金分割探索と逐次放物線補間の組合せである．収束はフィボナッチ探索に比べてかなり遅いということない．もし f が最小値 (lower や upper でないとする) で正の連続 2 階微分値を持てば収束は線形以上であり，普通約 1.324 乗のオーダーである．

関数 f は相互に eps*|x0|+(tol/3) 以下に近い二点では決して評価されない．ここで eps は約 sqrt(.Machine$double.eps) であり，|x0| は最終の座標 optimize()$minimum である．もし f が単峰関数で x の計算された値が

eps*|x|+(tol/3) 以上離れれば常に単峰であれば，x0 は f の区間 [lower, upper] 上での大局的最小値の座標に eps*|x0|+tol 以下の誤差で近付く．もし f が単峰でなければ，optimize() は局所的最小値を同じ誤差で近似するが，おそらく大局的最小値ではない．

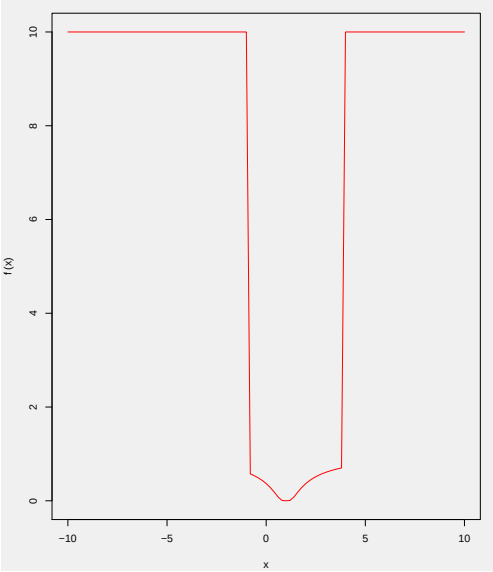
f の最初の評価は常に座標 $x_1 = a + (1 - \phi)(b - a)$ で行われる．ここで $(a, b) = (\mathtt{lower}, \mathtt{upper})$ であり $\phi = (\sqrt{5} - 1)/2 = 0.61803$ は黄金分割比である．ほとんど常に 2 度目の評価は座標 $x_2 = a + \phi(b - a)$ で行われる．区間 $[x_1, x_2]$ 中にある局所的最小値は，たとえ f がそこで定数値でも，解とされることを注意しよう，後の例を参照せよ．

関連：nlm(), uniroot().

```
> f <- function (x,a) (x-a)^2                          # 目的関数 (引数 x に付いて最小化する)
# 探索区間は [0,1], 要求精度は 0.0001, 追加パラメータ a=1/3
> ( xmin <- optimize(f, c(0, 1), tol = 0.0001, a = 1/3) )
$minimum                                               # 求められた最小値を与える x の値
[1] 0.3333333
$objective                                             # そのときの関数の値
[1] 0
```

```
# 関数を評価する度に x の値を出力．探索区間は [0,10]
> optimize(function(x) x^2*(print(x)-1), l=0, u=10)
[1] 3.81966                                            # 以下探索過程に応じた x の値
[1] 6.18034
(途中省略)
[1] 0.6666728
$minimum                                               # 求まった最小値を与える x の値
[1] 0.6666728
$objective
[1] -0.1481481                                         # そのときの目的関数の値
```

```
# 区分的に定数値の関数と不適切な区間を与えたとき誤った解が得られる例 (次の図を参照)
> f  <- function(x) ifelse(x>-1, ifelse(x<4, exp(-1/abs(x-1)), 10), 10)
> fp <- function(x) {print(x); f(x)}                   # f を評価する際 x 引数の値を出力させる
> plot(f, -10,10, col = 2)                             # f のグラフ
> optimize(fp, c(-4, 20))                              # 最小解を求めるのに失敗
[1] 5.167184
(途中省略)
[1] 9.266889
$minimum                                               # 求められた最小値を与える x の値
[1] 9.266889
$objective                                             # そのときの関数値
[1] 10

> optimize(fp, c(-7, 20))                              # 探索区間を広げる
[1] 3.313082
(途中省略)
[1] 0.9996565                                          # 最小解を求めるのに成功
$minimum
[1] 0.9996565
$objective
[1] 0
```



区分的に定数値の関数への optimize() の適用．探索区間 $[-4, 20]$ では失敗，$[-7, 20]$ では成功

9.2　一変数関数の零点を見付ける

9.2.1　一変数関数の零点を見付ける uniroot()

関数 uniroot() は関数 f の第一引数に付いて区間 [lower, upper] 内で根 (零点) を探す．

書式：

```
uniroot(f, interval, lower = min(interval), upper = max(interval),
        tol = .Machine$double.eps^0.25, maxiter = 1000, ...)
```

引数：

f　根を求めるべき関数

interval　根を探す区間の端点を含むベクトル

lower　根を探す区間の下端点

upper　根を探す区間の上端点

tol　要求される精度 (収束許容度)

maxiter　最大繰りかえし数

...　f への追加引数

返り値：　次の 4 つの成分を持つリスト：root と f.root は根の位置とその点での関数値である．iter と estim.prec は実行された繰り返し回数と，root に対する精度の推定値

interval もしくは lower と upper の双方が指定されなければならない．もしアルゴリズムが maxiter ステップ内で収束しなければ，警告とともに現在の近似値が返される．

関連： 多項式の全ての複素根を求める polyroot()．

```
# 3 次多項式の解を区間 [-2,2] で求める．結果の要約を有効数字 10 桁で出力
> str(uniroot(function(x) x*(x^2-1)+0.5, low=-2, up=2, tol=0.0001),
```

```
      dig=10)
List of 4
 $ root      : num -1.191487962                    # 零点
 $ f.root    : num -2.549728179e-07                # 関数値 (文字通り 0 とはならない)
 $ iter      : int 7                               # 繰り返し回数
 $ estim.prec: num 5e-05                           # 根の推定精度

# 精度を上げて再実行
> str(uniroot(function(x) x*(x^2-1)+0.5,low=-2,up=2,tol=1e-10),dig=10)
List of 4
 $ root      : num -1.191487884
 $ f.root    : num 5.66574565e-11
 $ iter      : int 8
 $ estim.prec: num 5.000044823e-11
```

```
# 1e80*exp(x)>1e-300 となる最小の x を区間 [-1000,0] 内で数値的に求める
> r <- uniroot(function(x) 1e80*exp(x) -1e-300,,-1000,0, tol=1e-20)
> str(r, digits= 15)
List of 4
 $ root      : num -745.133219101941               # 求められた零点
 $ f.root    : num -1e-300                         # 関数値
 $ iter      : int 75                              # 繰り返し回数
 $ estim.prec: num 4.54747350886464e-13            # 推定精度
>
> exp(r$r)
[1] 0                                              # exp(根) は実数精度内で 0 となる
> minexp <- r$r * (1 - .Machine$double.eps)        # 0.9999... を掛ける
> exp(minexp)
[1] 4.940656e-324                                  # もはや 0 にはならない
```

9.2.2 実・複素多項式の零点を見付ける polyroot()

`polyroot()` は実・複素多項式の零点を見付ける.

書式： `polyroot(z)`
引数： `z` 昇巾順に並べた多項式係数のベクトル
返り値： 多項式の (複素) 根を与える長さ $n-1$ の複素ベクトル. ここで n は `z` の 0 でない要素の最大位置

次数 $n-1$ の多項式

$$p(x) = z_1 + z_2 x + \ldots + z_n x^{n-1}$$

はその係数ベクトル `z[1:n]` で与えられる. `polyroot()` は $p(x)$ の $n-1$ 個の複素根を Jenkins-Traub のアルゴリズムで求める. もし係数ベクトル `z` の最も高い次数が 0 なら, それらは捨て去られる.

関連： 任意関数の根を求める `uniroot()`.

```
> polyroot(c(1,2,1))                               # 多項式 1+2x+x^2 の根を求める
[1] -1-1.665335e-16i -1+1.665335e-16i              # 二つの共役複素数根
> polyroot(c(1,2,1,0,0))                           # 同じ例
[1] -1-1.665335e-16i -1+1.665335e-16i
> round(polyroot(choose(8, 0:8)), 11)              # (1+x)^8 の 8 重根
[1] -1+0i -1+0i -1+0i -1+0i -1+0i -1+0i -1+0i -1+0i
> polyroot(1)
complex(0)                                         # y=1 の解 (空, つまり解無し)
```

9.3 関連関数

9.3.1 数式微分関数 deriv()

deriv() は簡単な表現式の数式微分を計算する.

書式:

```
D(expr, name)
deriv(expr, namevec, function.arg, tag = ".expr", hessian = FALSE)
deriv3(expr, namevec, function.arg, tag = ".expr", hessian = TRUE)
```

引数:

expr 微分される表現式または呼出し式

name,namevec 文字列ベクトルで，それに関して微分が計算される変数名を与える (D() に対しては一つだけ)

function.arg もし指定されると，関数返り値の引き数の文字ベクトル．もしくは (本体が空の) 関数，または TRUE 最後は引数名に namevec を持つ関数が使われるべきことを意味する

tag 文字列．結果として局所的に作られる変数に付く接頭辞

hessian 論理値．2 階の微分を計算し返り値に含めるべきか？

返り値: D() は関数呼出し式を返し，したがって容易に高階の微分のために繰り返すことができる．deriv() と deriv3() は普通 expression オブジェクトを返し，その評価は "gradient" 属性を持つグラディエント行列を含む関数を返す．もし hessian = TRUE なら，評価はヘッセ配列を含む "hessian" 属性を返す．もし function.arg が指定されると，deriv() と deriv3() は表現式ではなくこれらを引数に持つ関数を返す

D は単純な数式微分を計算する S の同名の関数にならって移植されている．deriv() は総称的関数で，既定動作とモデル式に対するメソッドを持つ．これは expr とその (偏) 微分を同時に計算できる呼出し式を返す．いわゆる *"algorithmic derivatives"* である．もし function.arg が関数なら，その引数は既定値を持つことができる，以下の例の fx 関数を見よ.

現在のところ，deriv.formula() は単に ~ の右にある表現式を取り出した後 deriv.default() を呼び出すだけである．deriv3() とそのメソッドは hessian = TRUE が既定値であることを除けば deriv() に等しい.

関連: 微分関数を使うことができる数値的最小化には nlm() と optim() がある.

```
> dx2x <- deriv(~ x^2, "x")                    # モデル式を使う. x^2 を変数 x に関して一回微分
> dx2x                                         # 結果を表す表現式
expression({
    .value <- x^2                              # 微分前の表現
    .grad <- array(0, c(length(.value),1), list(NULL,c("x")))
    .grad[, "x"] <- 2 * x                      # 微分結果
    attr(.value, "gradient") <- .grad          # .grad に"gradient"属性付与
    .value})
> mode(dx2x)                                   # オブジェクト dx2x のタイプ
```

```
[1] "expression"                                    # 表現式
> x <- -1:2
> eval(dx2x)                                        # 形式的引数 x の値を指定して評価
[1] 1 0 1 4                                         # (微分前の) 関数値
attr(,"gradient")
     x
[1,] -2                                             # 微分された関数による値
[2,]  0
[3,]  2
[4,]  4
```

```
> trig.exp <- expression(sin(cos(x + y^2)))         # より困難な例.  表現式オブジェクト
> ( D.sc <- D(trig.exp, "x") )                      # 変数 x で一階偏微分
-(cos(cos(x + y^2)) * sin(x + y^2))                 # 結果の表現式
> ( dxy <- deriv(trig.exp, c("x", "y")) )           # 変数 x,y で一階偏微分
expression({                                        # 結果の表現式
    .expr2 <- x + y^2
    .expr3 <- cos(.expr2)
    .expr5 <- cos(.expr3)
    .expr6 <- sin(.expr2)
    .value <- sin(.expr3)                           # 微分前の関数表現
    .grad <- array(0, c(length(.value), 2), list(NULL, c("x","y")))
    .grad[, "x"] <- -(.expr5*.expr6)                # x による偏微分結果
    .grad[, "y"] <- -(.expr5*(.expr6*(2*y)))        # y による偏微分結果
    attr(.value, "gradient") <- .grad               # .grad に"gradient"属性付与
    .value})
> x <- -1:2; y <- 1                                 # 形式引数 x,y に値を代入
> eval(dxy)                                         # その時の dxy の値 4 組を評価
[1]  0.8414710  0.5143953 -0.4042392 -0.8360219
attr(,"gradient")
             x         y
[1,]  0.0000000  0.000000                           # グラディエントベクトルの対応する 4 組の値
[2,] -0.7216061 -1.443212
[3,] -0.8316919 -1.663384
[4,] -0.0774320 -0.154864
> eval(D.sc)                                        # x による偏微分の値 4 組
[1]  0.0000000 -0.7216061 -0.8316919 -0.0774320
```

```
# 関数変数 sin(cos(x)*y) を x,y で一階偏微分
> deriv((y ~ sin(cos(x)*y)),c("x","y"),func=TRUE)
function (x, y) {                                   # 返り値は関数
  .expr1 <- cos(x)
  .expr2 <- .expr1 * y
  .expr4 <- cos(.expr2)
  .value <- sin(.expr2)                             # 微分される前の関数
  .grad <- array(0, c(length(.value), 2), list(NULL,c("x","y")))
  .grad[, "x"] <- -(.expr4*(sin(x)*y))              # x に関する一階偏微分関数
  .grad[, "y"] <- .expr4*.expr1                     # y に関する一階偏微分関数
  attr(.value, "gradient") <- .grad                 # .grad に"gradient"属性付与
  .value}
```

```
# 既定値を持つ変数を含む関数 (変数 x は既定値 1:7 を持つ)
> (fx <- deriv(y ~ b0+b1*2^(-x/th), c("b0","b1","th"),
               function(b0, b1, th, x = 1:7){} ) )
function (b0, b1, th, x = 1:7) {
  .expr3 <- 2^(-x/th)
  .value <- b0 + b1 * .expr3
  .grad <- array(0, c(length(.value),3), list(NULL,c("b0","b1","th")))
  .grad[, "b0"] <- 1
  .grad[, "b1"] <- .expr3
  .grad[, "th"] <- b1 * (.expr3 * (log(2) * (x/th^2)))
  attr(.value, "gradient") <- .grad
  .value}

> fx(2,3,4)                                         # 変数 x には既定値が使われる
[1] 4.522689 4.121320 3.783811 3.500000 3.261345 3.060660 2.891905
attr(,"gradient")
     b0        b1        th
[1,]  1 0.8408964 0.1092872                         # b0,b1,th に関するグラディエントの値
[2,]  1 0.7071068 0.1837984                         # x=1:7 に応じて 7 種類の値が計算される
[3,]  1 0.5946036 0.2318331
[4,]  1 0.5000000 0.2599302
[5,]  1 0.4204482 0.2732180
```

```
[6,]  1 0.3535534 0.2756976
[7,]  1 0.2973018 0.2704720
```

```
# 高階微分（グラディエントとヘッセ行列を同時に返す）
# 返り値の関数には，元の関数，グラディエントとヘッセ行列が同時に含まれる
> deriv3(y ~ b0+b1*2^(-x/th),c("b0","b1","th"),c("b0","b1","th","x"))
function (b0, b1, th, x) {
  .expr3 <- 2^(-x/th)
  .expr6 <- log(2)
  .expr7 <- th^2
  .expr9 <- .expr6 * (x/.expr7)
  .expr10 <- .expr3 * .expr9
  .value <- b0 + b1 * .expr3
  .grad <- array(0, c(length(.value), 3), list(NULL, c("b0",
      "b1", "th")))
  .hessian <- array(0, c(length(.value), 3, 3), list(NULL,
      c("b0", "b1", "th"), c("b0", "b1", "th")))
  .grad[, "b0"] <- 1
  .grad[, "b1"] <- .expr3
  .hessian[, "b1", "b1"] <- 0                    # ヘッセ行列成分（b1 に関する 2 階微分）
  .hessian[, "b1", "th"] <- .hessian[, "th", "b1"] <- .expr10
  .grad[, "th"] <- b1 * .expr10
  .hessian[, "th", "th"] <- b1 * (.expr10 * .expr9 - .expr3 *
      (.expr6 * (x * (2 * th)/.expr7^2)))
  attr(.value,"gradient") <- .grad               # .grad に"gradient"属性付与
  attr(.value,"hessian") <- .hessian             # .hessian に"hessian"属性付与
  .value}
```

```
> DD <- function(expr,name, order = 1) {   # 高階微分（DD() は引数階分再帰的に微分する関数）
          if(order < 1) stop("'order' must be >= 1")
          if(order == 1) D(expr,name)
          else DD(D(expr, name), name, order - 1) }

> DD(expression(sin(x^2)), "x", 3)               # x に関して 3 階微分
-(sin(x^2)*(2*x)*2 + ((cos(x^2)*(2*x)*(2*x) + sin(x^2)*
    2)*(2*x) + sin(x^2)*(2*x)*2))

# 表現を単純化する内部関数 simplify() の限界を示す例
> eval(DD(expression(sin(x^2)), "x", 3))         # x=-1:2 で評価
[1]  14.42007   0.00000 -14.42007  59.99645
```

9.3.2 モデル当てはめのプロファイリング profile()

関数 profile() は各種の統計モデル当てはめ結果の妥当性をチェックするための総称的関数[*3] で (非) 線形最小自乗法や最尤推定法用のメソッド関数を持つ.

書式： profile(fitted, ...)

引数：

fitted　当てはめモデルオブジェクト

...　追加引数

オブジェクト fitted に含まれる最適解の近傍での目的関数の挙動を調べる．返り値はプロファイルされたパラメータ値での目的関数の値を成分に持つリスト．詳細は profile.nls() 等の各メソッド関数[*4] を参照．以下は非線形最小自乗法の例 (最尤推定量の例は章末を参照).

[*3] OS 等により関数 Rprof() が有効でない場合は使えない可能性がある.

[*4] 基本パッケージ stats4 には profile.mle(), 推奨パッケージ MASS には profile.nls(), profile.glm() がある.

```
> fm1 <- nls(demand ~ SSasympOrig(Time,A,lrc),data=BOD) # 自己開始型非線形モデル関数を使う
# A パラメータのプロファイリング
> pr1 <- profile(fm1)    # A,lrc パラメータを当てはめ値の近傍で変化させ，目的関数値の変化を見る
> pr1$A
          tau  par.vals.A par.vals.lrc
1  -6.9431473   8.6140764    1.3100919
2  -5.6876147  10.1501074    0.7250247
3  -4.4000315  11.7982472    0.4130025
4  -3.0291032  13.6811484    0.1282910
5  -1.4572773  16.1474757   -0.2204915
6   0.0000000  19.1425781   -0.6328217            # <- 当てはめパラメータ値
7   0.9642882  22.1376805   -0.9896883
(以下省略)
```

9.4 その他，最尤推定 (パッケージ stats4)

最尤推定法や (非) 線形最小自乗法を筆頭に，多くの統計モデルのパラメータの推定量はある目的関数の最大・最小化や，推定方程式の零点を求めることで計算され，本質的にはこの章にまとめた最適化関数が背後で使われる．R の基本パッケージ stats4[*5] は対数尤度関数のマイナス倍を汎用最適化関数 optim() を用いて最小化することにより最尤推定量を計算する関数 mle() を提供する．最適値でのヘッセ行列の逆行列から推定量の近似共分散行列も計算される．

par 見付かった最良のパラメータ値
objective par に対応する objective の値
convergence 整数コード．0 ならば成功裡に収束
message 追加の情報を与える文字列，もしくは NULL．詳細は PORT のドキュメントを見よ
iterations 必要だった繰り返し回数
evaluations 目的関数とグラディエント関数の評価回数

返り値： クラス mle-class のオブジェクトで，これから各種情報を取り出す各種メソッド関数も定義されている

書式： mle(minuslogl, start=formals(minuslogl),
method="BFGS", fixed=list(), ...)

引数：
minuslogl 対数尤度関数の符号を変えたもの
start 最適化関数に対する初期値を与える名前付きリスト
method 使用する最適化手法．optim() を参照
fixed 最適化の中途で固定されるべきパラメータ値を指定する名前付きリスト
... optim() に引き渡される追加引数

以下の例は平均 λ が ymax$/(1 + x/$xhalf$)$ であるポアッソン分布に従う 127 個の独立データが値 $x = 0, 1, \ldots, 10$ を取ったケースがそれぞれ $26, 17, \ldots, 8$ であった場合にパラメータ ymax, xhalf を最尤推定する手順を示している．

[*5] 但し，R を起動した状態では読み込まれておらず，命令 library(stats4) を予め実行する必要がある．

```
> library(stats4)                                    # パッケージ stats4 を読み込む
> x <- 0:10                                          # データ値
> y <- c(26,17,13,12,20,5,9,8,5,4,8)                 # x の各値に対応する観測度数
# ポアッソン分布の確率関数の対数値を加算し，更にマイナス倍
# パラメータ ymax,xhalf には既定値として適切な初期値を与える必要がある
> ll <- function(ymax=15, xhalf=16)
          -sum(dpois(y, lambda=ymax/(1+x/xhalf), log=TRUE))
> (fit <- mle(ll))                                   # 最尤推定値の計算
Call:
mle(minuslogl = ll)
Coefficients:                                        # 推定された値
     ymax     xhalf
24.993092  3.057062
```

```
# 以下幾つかのメソッド関数
> summary(fit)                                       # クラス"mle-class"に対する要約メソッド
Maximum likelihood estimation
Call:
mle(minuslogl = ll)
Coefficients:
       Estimate Std. Error
ymax  24.993092   4.224441                           # 最尤推定量とその標準偏差
xhalf  3.057062   1.034796
-2 log L: 57.20811                                   # 最大対数尤度の-2 倍

> coef(fit)                                          # 最尤推定量の値を取り出す
     ymax     xhalf
24.993092  3.057062
> logLik(fit)                                        # 最大対数尤度の値
'log Lik.' -28.60406 (df=2)
> AIC(fit)                                           # AIC 値
[1] 61.20811
> vcov(fit)                                    # ヘッセ行列からパラメータの近似共分散行列を計算
           ymax     xhalf
ymax  17.845905 -3.720571                            # 対角成分の平方根が推定量の標準偏差
xhalf -3.720571  1.070802
```

```
# 当てはめのプロファイリング結果のプロット（以下の図を参照）
# 最適値の周辺でパラメータ値を僅かに変化させ，目的関数値の変化を見る
> profile(fit)
An object of class “profile.mle"
Slot "profile":
$ymax
           z par.vals.ymax par.vals.xhalf
1  -3.3772856     13.991811       9.991571
2  -2.3413280     16.742131       6.539881
3  -1.4588654     19.492452       4.794190
4  -0.6879678     22.242772       3.748768
5   0.0000000     24.993092       3.057062
6   0.6190600     27.743412       2.568169
(途中省略)
$xhalf
           z par.vals.ymax par.vals.xhalf
1  -3.6918420     41.337984       1.035953
2  -2.6375860     36.077981       1.372804
3  -1.8609289     32.490515       1.709656
4  -1.2541109     29.873028       2.046507
5  -0.7611998     27.870851       2.383359
6   0.0000000     24.993092       3.057062
7   0.5717953     23.016961       3.730765
(以下省略)
> plot(profile(fit), absVal=FALSE)
```

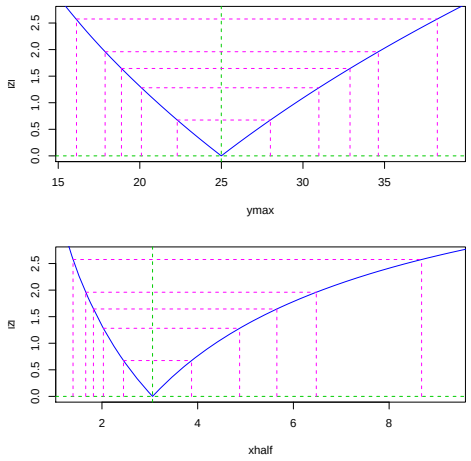
```
# 最尤推定量の漸近正規性を用いたパラメータの近似信頼区間（既定では信頼係数 95%）
> confint(fit)
Profiling...
          2.5 %    97.5 %
ymax  17.885654 34.619763
xhalf  1.663442  6.479056
```

```
> mle(ll, fixed=list(xhalf=6))                       # パラメータ xhalf の値を 6 に固定した場合
Call:
mle(minuslogl = ll, fixed = list(xhalf = 6))
```

```
Coefficients:
    ymax    xhalf
19.28809  6.00000
```

```
# 二つのパラメータ値が非負という制約を加えて最適化
> (fit1 <- mle(ll, method="L-BFGS-B", lower=c(0, 0)))
Call:
mle(minuslogl = ll, method = "L-BFGS-B", lower = c(0, 0))
Coefficients:
     ymax     xhalf
24.999420  3.055779
```

```
# 正値制約を無制約化するパラメトライゼーションを行う
# ymax=exp(lymax) と xhalf=exp(lxhalf) の関係に注意
> ll2 <- function(lymax=log(15), lxhalf=log(6))
          -sum(dpois(y, lambda=exp(lymax)/(1+x/exp(lxhalf)), log=TRUE))
> (fit2 <- mle(ll2))
Call:
mle(minuslogl = ll2)
Coefficients:
   lymax   lxhalf
3.218870 1.117006

> exp(coef(fit2))
    lymax    lxhalf                                  # ymax と xhalf の最尤推定値
24.999855  3.055691
> exp(confint(fit2))                                 # 同様に信頼区間も指数関数で変形する必要
Profiling...
           2.5 %    97.5 %
lymax  17.889802 34.617827
lxhalf  1.661492  6.480038
```



負の対数尤度の最小化のプロファイリング結果

第 10 章
行列分解

R の行列分解関数は多変量解析，線形モデル等の多くの関数の内部で用いられる．QR 分解 (QR decomposition) と特異値分解 (SVD, singular value decomposition) がその中心である．

「QR 分解」は任意の正方行列 A を積 QR として表す．ここで Q は正規直交行列であり，また R は主対角線より下のすべての要素が 0 の上三角行列である．「特異値分解」は任意の (必ずしも正方行列ではない) 行列 A を UDV^t という形式に分解する．ここで U, V は正規直交行列，D は A の固有値を (重複度をこめて) 並べた対角行列である．V^t は V の転置 (複素行列なら転置行列の複素共役行列) を表す．「コレスキ分解 (Cholesky decomposition)」は正定値対称行列 A を $A = B^tB$ と表す．ここで B は上三角行列である．

10.1 行列の QR，SVD 分解

10.1.1 行列の QR 分解，関数 qr()

qr() は行列の QR 分解を計算する．これは LINPACK ルーティン DQRDC または LAPACK ルーティン DGEQP3 (複素行列に対しては ZGEQP3) で用いられている手法へのインタフェイスである．

書式：

```
qr(x, tol = 1e-07, LAPACK = FALSE)
qr.coef(qr, y)
qr.qy(qr, y)
qr.qty(qr, y)
qr.resid(qr, y)
qr.fitted(qr, y, k = qr$rank)
qr.solve(a, b, tol = 1e-7)
# クラス "qr" に対する S3 メソッド
solve(a, b, ...)
is.qr(x)
as.qr(x)
```

引数：

x	QR 分解を計算すべき行列

tol x の列間の線形従属性を検出する際の許容度．LAPACK=FALSE で x が実行列の際だけ意味がある
qr qr() が計算したタイプの QR 分解
y，b 方程式の右辺のベクトルもしくは行列
a QR 分解，もしくは (qr.solve() に対してだけ) 矩形行列
k 実効的なランク
LAPACK 論理値．実行列 x に対しては，真ならば LAPACK，さもなければ LINPACK ルーチンを使う
... 他のメソッドへ (から) 引き渡される追加引数

返り値： 行列の QR 分解は LINPACK もしくは LAPACK ルーティンで計算される．返り値リストの以下の成分は DQRDC/DGEQP3/ZGEQP3 の返り値に直接対応する
qr x と同じ次元の行列．上三角部分は分解の R 部分を，下三角部分は Q 部分を含む (簡潔に保管されている)．DQRDC と DGEQP3 では保管方法が異なる
qraux 長さ ncol(x) のベクトルで，Q に関する追加情報を含む
rank 分解を用いて計算した x のランク．LAPACK ケースでは常にフルランク
pivot 分解の過程で使われたピボット情報

QR 分解は多くの統計手法で重要な役割を果たす．特に，与えられた行列 A とベクトル b に対して方程式 $Ax = b$ を解くのに使われる．回帰係数の計算や Newton-Raphson アルゴリズムでも有用である．関数 qr.coef, qr.resid() そして qr.fitted は y に QR 分解 qr() を当てはめたときの係数，残差，そして当てはめ値を返す．qr.qy() と qr.qty() は Q %*% y と t(Q) %*% y を返す，ここで Q は Q 行列である．上の全ての関数は，もしあれば x と y の dimnames (そして names) を保つ．

solve.qr() は qr オブジェクトに対する solve() 関数のメソッドである．qr.solve() は方程式系を QR 分解で解く．もし a が QR 分解ならそれは solve.qr() と同じであるが，もし a が長方形行列なら QR 分解が最初に計算される．どちらも条件が多すぎる，また少なすぎる系[*1] を扱うことができ，状況に応じて最小長解，もしくは最小 2 乗あてはめを返す．

is.qr() はもし x が名前付き成分 qr, rank そして qraux を持つリストなら TRUE を，さもなければ FALSE を返す．オブジェクトをモード "qr" に強制変換することはできない．オブジェクトは QR 分解であるかそうでないかのどちらかである．

注意：LAPACK で計算された実 QR オブジェクトは値が TRUE の属性 "useLAPACK" を持つ．行列の行列式を計算する (その前に本当に必要かどうか考えよう) には，固有値 eigen を使うより QR 分解経由の方がはるかに効率的である．det() を見よ．(複素行列を含む) LAPACK は列ピボット操作を用い，ランク落ちした行列の検査を行わない．offset で指定されたオフセット項は predict.lm() による予測には含められない．他方で，モデル式のオフセット項で指定されたものは含められる．

関連：行列の再構成には qr.Q(), qr.R(), qr.X(). solve.qr(), lsfit(), eigen(), svd(). qr() を使った行列式の計算には det().

[*1] Moore-Penrose 一般化逆行列 (344 頁) を参照.

```
# 行列のランクの計算は数値的に困難であることを示す例
# Hilbert 行列を計算する関数
> hilbert <- function(n) {i <- 1:n; 1/outer(i - 1, i, "+")}
> h9 <- hilbert(9)             # 9 次の Hilbert 行列
> h9
         [,1]     [,2]     [,3]      [,4]      [,5]      [,6]      [,7]
[1,] 1.000000 0.500000 0.333333 0.2500000 0.2000000 0.1666667 0.1428571
[2,] 0.500000 0.333333 0.250000 0.2000000 0.1666667 0.1428571 0.1250000
(以下省略)
          [,8]      [,9]
[1,] 0.1250000 0.1111111
[2,] 0.1111111 0.1000000
(以下省略)
> qr(h9)$rank                                    # ランク (7 しかない)
[1] 7
> qrh9 <- qr(h9, tol = 1e-10)                    # 数値計算精度を上げて再計算
> qrh9$rank                                      # フルランクという結果
[1] 9

> y <- 1:9/10
> x <- qr.solve(h9, y, tol = 1e-10)              # 線型方程式を解く
> x <- qr.coef(qrh9, y)                          # その解
> all.equal(as.vector(h9%*%x), y)                # 解をチェック (h9%*%x は行列に注意)
[1] TRUE                                         # 正しい

# qr.Q(qrh9) と qr.R(qrh9) から qrh9 を復元するにはピボット情報を考慮する必要
> all.equal(qr.Q(qrh9) %*% qr.R(qrh9)[,order(qrh9$pivot)], h9)
[1] TRUE                                         # 数値精度範囲内で復元できている

# qr.Q(qrh9) は直交行列か？
> all.equal(t(qr.Q(qrh9)) %*% qr.Q(qrh9), diag(9))
[1] TRUE                                         # 数値精度範囲内で OK
```

10.1.2 QR 分解の補助関数

これらの関数は QR オブジェクトから元の行列 X もしくは分解成分行列 Q, R を返す．

書式：

```
qr.X(qr, complete = FALSE, ncol =)
qr.Q(qr, complete = FALSE, Dvec =)
qr.R(qr, complete = FALSE)
```

引数：

`qr` QR 分解を表現するオブジェクト．これは典型的には `qr()` もしくは `lsfit()` の呼出し結果である

`complete` 論理値．`Q` もしくは `X` 行列の任意の直交補完を求めるか，それとも R 行列を正方上三角行列の下に 0 からなる行を付け加えて補完するかを指示する

`ncol` 範囲 `1:nrow(qr$qr)` 内の整数で，再構成された `X` の列数．`complete=FALSE` の時の既定値は `qr` オブジェクトが計算された元の `X` の最初の `min(ncol(X), nrow(X))` 列．`complete=TRUE` の時の既定値は，元の `X` が最初の `ncol(X)` 列で，残りの列は任意の直交補完 (複素ケースではユニタリ補完) である正方行列

`Dvec` 対角成分のベクトル (行列ではない)．返り値の Q の各列には，これの対応する対角成分値がかけられる．既定では全て 1 のベクトル

`qr.X()` は `ncol(X) <= nrow(X)` である限り，qr オブジェクトを計算した元の行列を

返す．もし complete = TRUE であるか，ncol が ncol(X) より大きければ，X を任意に直交 (ユニタリ) 補完した追加の列が返される．

qr.Q() は Q 行列を返す．これは qr により表現される次数 nrow(X) の直交 (ユニタリ) 変換である．もし complete = TRUE ならば Q は nrow(X) 個の列を持つ．もし complete = FALSE なら Q は ncol(X) 個の列を持つ．Dvec が指定されたときは，Q の各列には Dvec 中の対応する値がかけられる．qr.R() は X == Q %*% R となるような上三角行列 R を返す．R の行数は complete が TRUE か FALSE に応じて，nrow(X) か ncol(X) である．

関連： qr(), qr.qy().

```
# 家計貯蓄率データを使う．x は総個人貯蓄変数を除いた 50x4 行列
> p <- ncol(x <- LifeCycleSavings[,-1])
> qrstr <- qr(x)                                    # QR 分解
> qrstr$rank                                        # ランク
[1] 4
> Q <- qr.Q(qrstr)                                  # Q 行列，dim(Q)==dim(x)
> R <- qr.R(qrstr)                                  # R 行列，dim(R)==ncol(x)
> X <- qr.X(qrstr)                                  # X は元の行列 x に等しい

# X と x が数値精度内で一致することを確認．x はデータフレームなので行列に変換する必要
> range(X - as.matrix(x))                           # 食い違いのレンジ
[1] -1.818989e-12  1.136868e-12                     # 数値誤差範囲内で 0 と見做せる
> Q %*% R                                           # X は Q%*%R で計算される
        pop15 pop75     dpi  ddpi                   # x の列ラベルを引き継いでいる
   [1,] 29.35  2.87 2329.68  2.87
   [2,] 23.32  4.41 1507.99  3.93
   (途中省略)
  [50,] 47.20  0.66  242.69  5.08
```

10.2 行列の特異値分解

10.2.1 行列の特異値分解，svd()

svd() は矩形行列の特異値分解を計算する．

書式：

```
svd(x, nu = min(n, p), nv = min(n, p), LINPACK = FALSE)
La.svd(x, nu=min(n, p), nv=min(n, p), method=c("dgesdd","dgesvd"))
```

引数：

x　SVD 分解を求めたい行列

nu　計算されるべき左特異値ベクトルの数．これは method="dgesdd" の場合を除き 0, nrow(x) そして ncol(x) のどれかでなければならない

nv　計算されるべき左特異値ベクトルの数で 0 か ncol(x)

LINPACK　論理値．LINPACK を使うべきか？ (R1.7.0 以前との互換性のために存在)

method　実ケースで使うべき LAPACK ルーティン

返り値：　次の成分を持つリスト (La.svd() に対しては返り値は v は その転置 (複素行列なら共役転置) 行列である t(v) に置き換えられる)：

d　x の特異値を含むベクトル

u　x の左特異値ベクトルを列として持つ行列．nu>0 の時だけ与えられる

v　x の右特異値ベクトルを列として持つ行列．nv>0 の時だけ与えられる

特異値分解は多くの統計手法で重要な役割を果たす．svd() と La.svd() は二つの少し異なるインタフェイスを与える．主に使われる関数は LAPACK のルーチン DGESDD と ZGESVD である．svd(LINPACK=TRUE) は LINPACK ルーチン DSVDC へのインタフェイスを与えるが，単に過去との一貫性のためだけにある．La.svd() は LAPACK のルーチン DGESVD と DGESDD の双方へのインタフェイスを与える．後者は特異値ベクトルが必要なら普通かなり早い．

<URL:http://www.cs.berkeley.edu/~demmel/DOE2000/Report0100.html>

を見よ．ほとんどの利点は最適化 BLAS システム[*2] と併用することにより得られる．method="dgesdd" の使用は IEEE754 算術演算を必要とする．もしこれが使用中のプラットフォームで利用不可能なら警告とともに method="dgesvd"が使われる．特異値ベクトルの計算は大きな行列に対しては低速である．基本にある LAPACK コードからの失敗情報は正のエラーコードを与えるエラーになる．これらは FORTRAN コードを詳しく検討しない限り説明不可能である．

LINPACK により計算される SVD 分解は次の形式を持つ：

$$X = UDV^T$$

ここで U と V は直交行列，V^T は V の転置，そして D は特異値 D_{ij} からなる対角行列である．同じことだが，以下の実例で分かるように $D = U^T XV$ である．

関連：eigen(), qr().

```
# Hilbert 行列の非正方副行列の SVD 分解
> hilbert <- function(n) { i <- 1:n; 1/outer(i - 1, i, "+") }
> X <- hilbert(9)[,1:6]                          # 非正方な部分行列を取り出す
> (s <- svd(X))                                  # SVD 分解実行
$d                                               # 特異値ベクトル
[1] 1.668433e+00 2.773727e-01 2.223722e-02 1.084693e-03 3.243788e-05
[6] 5.234864e-07
$u
           [,1]      [,2]      [,3]       [,4]      [,5]        [,6]
 [1,] -0.724410  0.626562 0.2735000 -0.0852690 0.0207412 -0.00402455
 (途中省略)
 [9,] -0.124894 -0.257125 0.3654254  0.4388465 0.4649671  0.43459954
$v
          [,1]      [,2]     [,3]       [,4]      [,5]        [,6]
[1,] -0.736493  0.622500 0.255002 -0.0697629 0.0132823 -0.00158815
 (途中省略)
[6,] -0.190442 -0.383187 0.511065  0.5385635 0.4466363  0.25724891

> D <- diag(s$d)                                 # 特異値を対角成分とする対角行列
> s$u %*% D %*% t(s$v)                           # UDV' で X を復元
           [,1]      [,2]       [,3]       [,4]       [,5]       [,6]
 [1,] 1.0000000 0.5000000 0.33333333 0.25000000 0.20000000 0.16666667
 (途中省略)
 [9,] 0.1111111 0.1000000 0.09090909 0.08333333 0.07692308 0.07142857

> t(s$u) %*% X %*% s$v                           # D=U'XV を計算
              [,1]          [,2]          [,3]          [,4]          [,5]
```

[*2] LAPACK(Linear Algebra PACKage) は netlib で公開されているフォートラン数値計算ライブラリ．BLAS(Basic Linear Albegra Subprograms) はその線形演算ライブラリ．特に後藤茂和氏による最適化 BLAS (libGoto) は効果が大きい．

```
[1,]  1.668433e+00 3.026618e-16 -1.266755e-16 -1.201025e-16 1.198586e-16
（途中省略）
[6,]  2.260618e-17 6.278690e-18  1.090743e-17 -1.076960e-18 3.338436e-18
             [,6]
[1,] -8.178950e-17
（途中省略）
[6,]  5.234864e-07
```

10.3 行列のコレスキ分解

10.3.1 行列のコレスキ分解，chol()

chol() は実正定値対称行列のコレスキ分解 (Cholesky decomposition) を計算する．

書式：

```
chol(x, pivot = FALSE,  LINPACK = pivot)
La.chol(x)
```

引数：

x　実正定値対称行列

pivot　ピボット操作を行うべきか？

LINPACK　論理値．LINPACK ルーティンを使うか？ (バージョン 1.7.0 未満の R との整合性のため用意されている)

返り値： コレスキ分解の上三角部分．つまり $R^TR = x$ となる上三角行列 R である，実行例を見よ．もしピボット操作が使われると，二つの追加属性 "pivot" と "rank" が返される

オプション chol(pivot = TRUE) は LINPACK ルーティン DCHDC へのインタフェイスを与える．La.chol() は LINPACK ルーティン DPOTRF へのインタフェイスを与える．

x の上三角部分だけが使われ，したがって x が対称なときだけ $R^TR = x$ となることを注意しよう．もし pivot = FALSE で x が非負定値でなければ，エラーが生じる．もし x が非負定値 (つまり 0 の固有値がある) であれば，数値的精度の観点からやはりエラーが生じる．もし pivot = TRUE なら，非負定値行列 x の Choleski 分解が計算できる．x のランクは attr(Q, "rank") で得られるが，数値的な誤差を含む．ピボットは attr(Q, "pivot") で得られる．こうした場合もはや t(Q) %*% Q は x には一致しない．しかしながら，pivot <- attr(Q,"pivot") そして oo <- order(pivot) と置くと t(Q[, oo])%*% Q[, oo] は x に一致する，もしくは t(Q) %*% Q は x[pivot, pivot] と一致する．以下の参考例を見よ．

注意： 対称性のチェックはされない．もし pivot=TRUE で x が非負定値でなければ，警告は発せられず，無意味な結果が得られるかも知れない．したがって x がその構成から非負定値であることが確かな場合だけオプション pivot=TRUE を使おう．

関連： (ピボット操作無しの) 逆行列を求めるには chol2inv()．線型方程式系を上三角部分を使って解く backsolve()．関連する行列分解には qr(), svd()．

```
> ( m <- matrix(c(5,1,1,3),2,2) )                    # 例示用対称行列
     [,1] [,2]
[1,]    5    1
[2,]    1    3
> ( cm <- chol(m) )                                  # そのコレスキ分解
         [,1]      [,2]
[1,] 2.236068 0.4472136
[2,] 0.000000 1.6733201

> t(cm) %*% cm                                       # m を復元
     [,1] [,2]
[1,]    5    1
[2,]    1    3

> crossprod(cm)                                      # 同じこと
     [,1] [,2]
[1,]    5    1
[2,]    1    3

# 非負定値行列なら (分解の正当性は必ずしも保証されない)
> x <- matrix(c(1:5, (1:5)^2), 5, 2)
> x <- cbind(x, x[, 1] + 3*x[, 2])
> m <- crossprod(x)                                  # 例示用非負定値行列
> eigen(m)                                           # 固有値は？
$values
[1]  1.124685e+04  3.149326e+00 -1.413945e-13      # 復元値は (数値精度内で) ランク落ち
> all.equal(t(chol(m)) %*% chol(m), m)               # 一応 m を復元できている
[1] TRUE
```

```
# chol() は qr() と異なり計算精度を指定できず，丸め誤差により失敗することがある
> try(chol(m))
         [,1]     [,2]         [,3]
[1,] 7.416198 30.33899 9.843318e+01
[2,] 0.000000  7.65150 2.295450e+01
[3,] 0.000000  0.00000 2.980232e-08
> (Q <- chol(m, pivot = TRUE))                       # m はランク落ちしていることを注意
         [,1]     [,2]          [,3]
[1,] 101.0742 7.222415  3.128394e+01
[2,]   0.0000 1.684259 -5.614195e-01
[3,]   0.0000 0.000000  2.092056e-07
attr(,"pivot")
[1] 3 1 2
attr(,"rank")                                        # 不思議なことにフルランクとされる
[1] 3

# こうした場合，例えば次のようにすることが考えられる
> pivot <- attr(Q, "pivot")
> oo <- order(pivot)
> t(Q[, oo]) %*% Q[, oo]                             # m を復元できた
     [,1] [,2]  [,3]
[1,]   55  225   730
[2,]  225  979  3162
[3,]  730 3162 10216
```

10.3.2 コレスキ分解による逆行列計算，chol2inv()

chol2inv() は正定値対称行列の逆行列をコレスキ分解から求める．

書式：

```
chol2inv(x, size = ncol(x), LINPACK = FALSE)
La.chol2inv(x, size = ncol(x))
```

引数：

x　行列．上三角部分の最初の nc 個の列は逆行列を求めるべき行列のコレスキ分解を含む

size コレスキ分解を含む x の列数

LINPACK 論理値．LINPACK を使うべきか？ (R1.7.0 以前との互換性のために存在する)

返り値： 分解された行列の逆行列

chol2inv(LINPACK=TRUE) は LINPACK のルーチン DPODI へのインタフェイスである．La.chol2inv() は LINPACK のルーチン DPOTRI へのインタフェイスである．

関連：chol(), solve().

```
> cma <- chol(ma  <- cbind(1, 1:3, c(1,3,7)))     # 人工的な例のコレスキ分解
> ma %*% chol2inv(cma)                            # 逆行列かどうか確認
              [,1]         [,2] [,3]
[1,]  1.000000e+00 0.000000e+00    0              # 数値精度範囲内で単位行列
[2,]  1.110223e-16 1.000000e+00    0
[3,] -1.110223e-16 4.440892e-16    1
```

10.4 関連関数

10.4.1 三角行列係数の線型方程式を解く backsolve()

これらの関数は係数行列が上・下三角行列である線型方程式を解く．

書式：

backsolve(r, x, k= ncol(r), upper.tri = TRUE, transpose = FALSE)

forwardsolve(l, x, k= ncol(l), upper.tri = FALSE, transpose = FALSE)

引数：

r, l 方程式の係数である上 (下) 三角行列．下 (上) 側の値は無視される

x その列が方程式の「右辺」を与える行列

k r の列数であり，x の行数

upper.tri 論理値．もし TRUE (既定値) ならば r の上側三角部分が使われる．さもなければ下側三角部分が使われる

transpose もし TRUE ならば r'*y=x を y について解く，つまり r が転置される

返り値： 三角システムの解．結果は，もし x がベクトルならベクトル，行列なら行列となる

関連： chol(), qr(), solve().

```
> r <- rbind(c(1,2,3), c(0,1,1), c(0,0,2))        # 例示用上三角行列
> ( y <- backsolve(r, x <- c(8,4,2)) )            # r%*%y=x を解く
[1] -1  3  1
> r %*% y                                         # x=(8,4,2) を確認
     [,1]
[1,]    8
[2,]    4
[3,]    2

> backsolve(r, x, transpose = TRUE)               # t(r)%*%y==x を解く
[1]   8 -12  -5
```

10.4.2 行列の固有値と固有ベクトル eigen()

eigen() は行列のスペクトル分解 (固有値と固有ベクトル) を計算する.

書式: eigen(x, symmetric, only.values = FALSE, EISPACK = FALSE)

引数:

x スペクトル分解を求める行列

symmetric もし TRUE なら行列は対称 (複素行列ならエルミート行列) とされ, 下三角部分だけが使われる. さもなければ対称かどうかチェックされる

only.values もし TRUE なら固有値だけが計算される. さもなければ固有値と固有ベクトルの双方が計算される

EISPACK (バージョン 1.7.0 未満の R との互換性のためだけにある)

返り値: 行列 x のスペクトル分解を与える以下の成分を持つリスト:

values x の固有値のベクトルで, (複素) 絶対値の減少順に並べられる.

vectors 列が x の固有ベクトルからなる行列. もし only.values=TRUE なら NULL. 固有ベクトルは乗法定数を除いてしか定まらないことを注意しよう. ベクトルのノルムの値を指定しても, 複素ケースでは絶対値 1 の乗法定数, 実ケースでは符号分の曖昧さが依然として残る

既定では LAPACK ライブラリが使われ, EISPACK=TRUE なら EISPACK ライブラリが使われる. もし symmetric 引数を指定しなければ, 実数精度内で対称かどうかのチェックがされるので, 指定する方が確実で早い. 大きな行列に対しては固有ベクトルの計算は時間がかかる. また, スペクトル分解は実数演算の観点で精度に問題が起きる可能性がある (例えば実固有値が複素固有値になる等).

関連: eigen() の拡張である svd() および関連行列分解関数 qr(), chol().

```
> eigen(cbind(c(1,-1),c(-1,1)))                    # 2 次行列のスペクトル分解
$values
[1] 2 0
$vectors
           [,1]       [,2]
[1,] -0.7071068 -0.7071068
[2,]  0.7071068 -0.7071068

> eigen(cbind(1,c(1,-1)), only.values = TRUE)      # 固有値だけ計算
$values
[1]  1.414214 -1.414214
$vectors
NULL

> eigen(cbind(-1,2:1))                             # 複素数行列の例
$values
[1] 0+1i 0-1i
$vectors
                   [,1]                  [,2]
[1,] 0.8164966+0.0000000i 0.8164966+0.0000000i
[2,] 0.4082483+0.4082483i 0.4082483-0.4082483i
> eigen(cbind( 1,3:1,1:3))                         # 3 次行列の例
$values
[1]  5.000000e+00  1.000000e+00 -5.189805e-16
```

```
$vectors
           [,1]       [,2]       [,3]
[1,] -0.5773503 -0.8451543 -0.9428090
[2,] -0.5773503 -0.1690309  0.2357023
[3,] -0.5773503  0.5070926  0.2357023
```

10.4.3 行列式 det(), determinat()

det() は行列の行列式を計算する．determinant() は総称的関数で，行列式の絶対値 (オプションでその対数値) と符号を別個に返す．

書式：

det(x, ...), determinant(x, logarithm = TRUE, ...)

引数：

x 数値行列

logarithm もし既定の TRUE なら行列式の絶対値の対数値を返す

... オプション引数．現在無視される

返り値： det() に対しては x の行列式の値．determinant() に対しては以下の成分を持つリスト：

modulus logarithm=FLASE ならば行列式の絶対値．さもなければその対数値

sign 行列式の符号．+1 か -1

関数 determinant() は行列の LU 分解を使う．det() は単にそのラッパ関数である．普通，与えられた問題を解く際に行列式を計算することは本質的ではないことが多い．

```
> x                                                    # 2x2 行列
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> det(x)                                               # 行列式は-2
[1] -2

> determinant(x)
$modulus
[1] 0.6931472                                          # 行列式の値-2 の絶対値の対数
attr(,"logarithm")
[1] TRUE
$sign                                                  # 行列式の符号
[1] -1
attr(,"class")
[1] "det"
> unlist(determinant(x))
   modulus       sign
 0.6931472 -1.0000000

> determinant(x, logarithm=FALSE)
$modulus
[1] 2                                                  # 行列式の絶対値
attr(,"logarithm")
[1] FALSE
$sign
[1] -1
attr(,"class")
[1] "det"
```

10.4.4 行列の条件数 kappa(), rcond()

kappa() は行列の条件数 (conditional number) を計算する．rcond() は行列の逆条件数 (reciprocal conditional number) を計算する．

書式：

```
kappa(z, ...)
kappa(z, exact=FALSE,
      norm=NULL, method=c("qr","direct"), ...) # 既定の S3 メソッド
kappa(z, ...)                                  # クラス lm に対する S3 メソッド
kappa(z, ...)                                  # クラス qr に対する S3 メソッド
rcond(x, norm = c("O","I","1"), triangular = FALSE, ...)
```

引数：

z,x　行列，qr() 関数の結果，もしくはクラス "lm" を継承する当てはめオブジェクト

exact　論理値．FALSE なら近似式により計算する

norm　文字列で，条件数を計算する際に使われる行列ノルムを指定．既定の"O" は L^1-ノルム．現在可能なもう一つの選択肢 "I" は最大値ノルム L^∞

method　文字列で，使用する手法を指定する．主として過去との一貫性のため "qr" が既定値

triangular　論理値．もし TRUE なら行列の下三角部分だけが使われる

LINPACK　論理値．もし TRUE で z が実数なら Linpack ライブラリが，さもなければ Lapack ライブラリが使われる

...　他のメソッドへ (から) 引き渡される追加引数

条件数は行列とその (一般化) 逆行列の行列ノルムの積で，行列ノルムの定義に依存する．kappa() は L^2 ノルム を用いて，行列もしくは QR 分解 (線形モデル当てはめ結果に伴うそれを含む) の R 部分の条件数を計算する．これは，行列の最大特異値と最小特異値の比になる．

もし exact=FALSE ならば，kappa() は既定で L^2 ノルム に対する条件数の容易に計算できる近似値を計算するが，特異値分解を用いた正確な計算も十分速い．L^1 もしくは L^∞ ノルムに対する条件数は相当速く計算でき，rcond() はそれらの逆数 (reciprocal conditional number) を計算する．

注意： 行列 A の条件数は線形方程式 $y = Ax$ を解く際，y が変化すると解 x がどの程度変化するかを測る目安であり，大きな条件数は解の不安定性を示唆し，数値的にも悪条件になると考えられる．条件数は単位ノルムのベクトル x, y に対する $\|A^{-1}x\|/\|A^{-1}y\|$ の最大値であり，定義から 1 以上の値 (直交行列なら丁度 1 に等しい) になる．A が正則正方行列なら条件数は $\kappa(A) = \|A^{-1}\| \cdot \|A\|$ に一致する．L^2 ノルムに対しては $\kappa(A)$ は行列の最大特異値と最小特異値の比になる．

```
> kappa(x1 <- cbind(1,1:10))                    # 近似による値
[1] 15.70590
```

```
> kappa(x1, exact = TRUE)                          # 近似によらない値
[1] 13.67903

> kappa(x2 <- cbind(x1,2:11))                      # x2 は非正則であり，大きな条件数を持つ
[1] 8.351867e+16

# n 次の Hilbert 行列を計算する関数
> hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
> sv9 <- svd(h9 <- hilbert(9))$d                   # 9 次の Hilbert 行列に対する特異値分解
> kappa(h9)     # n が少し大きくなると hilbert(n) の行列式はほとんど 0 であり，条件数は相当大きい
[1] 728289149562

# 2 乗ノルムに対する条件数は最大・最小特異値の比として計算可能
> kappa(h9, exact = TRUE) == max(sv9)/min(sv9)
[1] TRUE
> kappa(h9, exact = TRUE) / kappa(h9)              # kappa() の近似値と正確な値の比較
[1] 0.6771398
```

10.5 その他

10.5.1 その他の行列分解，パッケージ Matrix

貢献パッケージ Matrix は疎行列 (大部分の要素が 0 であるような行列) 用のパッケージであるが, QR 分解, コレスキ分解以外に, Shur 分解, LU 分解, そして Bunch-Kaufman 分解用の関数を提供する．

必ずしも対称でない正方行列 A の Shur 分解は $A = QTQ^t$ の形の分解で，Q は直交行列であり，T はサイズが 1 か 2 の対角ブロックを持つブロック対角行列である．サイズが 2 のブロック対角副行列は A の複素固有値に対応する．A の固有値は T のそれと一致する．

正方行列 A の LU 分解は $A = PLUQ$ の形を持ち，P と Q は置換行列，L と U はそれぞれ下 (上) 三角行列である．

正方行列 A の Bunch-Kaufman 分解は $A = PLDL^tP^t$ の形を持ち，P と L は置換行列，D はサイズが 1 か 2 の対角ブロックを持つブロック対角行列である．

10.5.2 一般化逆行列，パッケージ MASS

最小自乗法ではベクトル y と矩形行列 A に対しノルム $\|y - Ax\|$ を最小化するベクトル x を求める必要がある．もし A が可逆な正方行列ならば当然 $x = A^{-1}y$ であるが，A が可逆でないか，正方行列で無い場合には解 x は一意には定まらないが，A の一般化逆行列 (generalized inverse, pseudoinverse) と呼ばれる行列 A^+ を用いて一つの解を $x = A^+y$ として求めることができる．一般化逆行列の定義は色々あるが，推奨パッケージ MASS[*3] は Moore-Penrose 一般化逆行列を計算する関数 ginv() を提供する．

サイズ $n \times m$ の行列 A に対し Moore-Penrose 一般化逆行列 A^+ は以下の性質を満たすサイズ $m \times n$ の行列で常に一意的に定まる：

$$AA^+A = A, \quad A^+AA^+ = A^+,$$
$$(AA^+)^* = AA^+, \quad (A^+A)^* = A^+A.$$

ここで B^* は B の共役転置 (実行列なら単に転置) である．もし A が可逆な正方行列なら $A^+ = A^{-1}$ である．A が正方行列なら QR 分解 $A = QR$ を用いて $A^+ = (R^*R)^+A^*$

[*3] パッケージバンドル VR の一部で，VR をインストールすれば使用可能になる．

と計算できる．ここで R^*R は対角行列であるから，$(R^*R)^+$ は R^*R の 0 以外の対角成分を逆数に変えたものになる．関係 $A^*A = R^*R$ より R^*R は行列 A^*A のコレスキ分解でもある．より一般的には A の特異値分解 $A = UDV^*$ を用いると，D は矩形対角行列であり，D^+ は D の 0 以外の対角成分を逆数に置き換えたものになり，$A^+ = VD^+U^*$ となる．

書式： `ginv(X, tol = sqrt(.Machine$double.eps))`
引数：
`X`　数値行列 `tol`　特異値が実数精度内で 0 かどうかを判定する許容値
返り値：　`X` の Moore-Penrose 一般化逆行列

関連: `solve()`, `svd()`, `eigen()`.
注意: 一般化逆行列は `solve(A, diag(1, ncol(A)))` でも計算できるが，これは Moore-Penrose 一般化逆行列ではない (上の性質の 4 番目が不成立)．正定値行列ならば逆行列は `chol2inv(A)` で計算できる．

```
> library(MASS)                                    # パッケージ MASS を読み込む
> ( x <- cbind(c(1,1,1),c(1,1,1), c(1,2,3)) )      # 非正則正方行列
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    2
[3,]    1    1    3
> ( y <- ginv(x) )                                 # その Moore-Penrose 一般化逆行列
          [,1]          [,2]       [,3]
[1,]  0.6666667  1.666667e-01 -0.3333333
[2,]  0.6666667  1.666667e-01 -0.3333333
[3,] -0.5000000 -1.243377e-16  0.5000000

> max(abs(x %*% y %*% x  - x))                     # 実数精度内で x%*%y%*%x は x に一致
[1] 3.330669e-16
> max(abs(y %*% x %*% y  - y))                     # 実数精度内で y%*%x%*%y は y に一致
[1] 5.551115e-16

> max(abs(t(y %*% x ) - y %*% x))                  # 実数精度内で y%*%x は対称
[1] 7.21645e-16
> max(abs(t(x %*% y ) - x %*% y))                  # 実数精度内で x%*%y は対称
[1] 5.684472e-16
```

```
> ( xx <- cbind(c(1,1,1),c(1,2,3), c(1,0,0), c(3,2,1)) ) # 矩形行列の場合
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    3
[2,]    1    2    0    2
[3,]    1    3    0    1
> ( yy <- ginv(xx) )                               # その Moore-Penrose 一般化逆行列
             [,1]       [,2]          [,3]
[1,] -6.606959e-16  0.1111111 -6.642432e-16
[2,]  9.706320e-17 -0.2777778  5.000000e-01
[3,]  1.000000e+00 -2.0000000  1.000000e+00
[4,]  3.203275e-16  0.7222222 -5.000000e-01

> max(abs(yy %*% xx %*% yy  - yy))
[1] 8.881784e-16
> max(abs(xx %*% yy %*% xx  - xx))
[1] 2.220446e-15

> max(abs(t(yy %*% xx ) - yy %*% xx))
[1] 9.937628e-16
> max(abs(t(xx %*% yy ) - xx %*% yy))
[1] 6.106227e-16
```

```
> ( yyy <- qr.solve(xx, diag(1, nrow(xx))) )      # qr.solve() を用いた一般化逆行列
     [,1] [,2] [,3]
[1,]    0    3   -2
[2,]    0   -1    1
[3,]    1   -2    1
[4,]    0    0    0

> max(abs(yyy %*% xx %*% yyy  - yyy))
[1] 1.332268e-15
> max(abs(xx %*% yyy %*% xx  - xx))
[1] 3.552714e-15
> max(abs(t(xx %*% yyy ) - xx %*% yyy))
[1] 2.220446e-15

# yyy%*%xx は対称ではなく，yyy は Moore-Penrose 一般化逆行列ではない
> max(abs(t(yyy %*% xx ) - yyy %*% xx))
[1] 4
```

例え可逆であっても逆行列を求めることは数値的に困難なことがある．例えば，線形方程式 $y = Ax$ は関数 `solve()` を用いて解くべきであり，逆行列 A^{-1} を先ず求めてから $x = A^{-1}y$ と解くことは数値的には勧められない．以下の 6 次の Hilbert 行列[*4] を用いた例が参考になる．

```
> X                                               # 6 次の Hilbert 行列
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 1.0000000 0.5000000 0.3333333 0.2500000 0.2000000 0.1666667
[2,] 0.5000000 0.3333333 0.2500000 0.2000000 0.1666667 0.1428571
[3,] 0.3333333 0.2500000 0.2000000 0.1666667 0.1428571 0.1250000
[4,] 0.2500000 0.2000000 0.1666667 0.1428571 0.1250000 0.1111111
[5,] 0.2000000 0.1666667 0.1428571 0.1250000 0.1111111 0.1000000
[6,] 0.1666667 0.1428571 0.1250000 0.1111111 0.1000000 0.0909091
> det(X)                                          # 行列式はほとんど 0
[1] 5.3673e-18

> eigen(X)$values                                 # 固有値は全て正で，正定値
[1] 1.618900e+00 2.423609e-01 1.632152e-02 6.157484e-04 1.257076e-05
[6] 1.082799e-07

# 以下 X の逆行列を 3 通りの方法で計算
> Y1 <- ginv(X)                                   # 一般化逆行列として計算
> Y2 <- solve(X, diag(1,6))                       # QR 分解を用いる標準的方法
> Y3 <- chol2inv(X)                               # 正定値対称行列であるから使える方法

> max(abs(X %*% Y1 - diag(1,6)))                  # Y1 の逆行列としての数値的精度
[1] 1.164153e-10
> max(abs(Y1 %*% X -diag(1,6)))                   # つまり X%*%Y1 は Y1%*%X と一致しない
[1] 1.164153e-10

> max(abs(X %*% Y2 - diag(1,6)))                  # Y2 の逆行列としての数値的精度
[1] 5.820766e-11
> max(abs(Y2 %*% X -diag(1,6)))
[1] 1.325589e-10

> max(abs(X %*% Y3 - diag(1,6)))                  # Y3 の逆行列としての数値的精度，悲惨な結果
[1] 0.8999073
> max(abs(Y3 %*% X -diag(1,6)))
[1] 0.8999073

> max(abs(Y1-t(Y1)))                              # Y1,Y2 は「数値的には」対称行列でない！
[1] 5.881302e-07
> max(abs(Y2-t(Y2)))
[1] 5.369075e-07
> max(abs(Y3-t(Y3)))
[1] 0
```

[*4] i, j 成分が $1/(i+j-1)$ である正方行列．正定値対称行列であるが，次数が少しでも高くなると行列式が 0 に近く，数値的困難さを持つことで知られている．

第 11 章
確率分布関数，疑似乱数

R は多くの確率分布[*1] に対する疑似乱数発生関数，分布関数，密度 (確率) 関数，クォンタイル関数を持ち，従来不可欠であった様々な統計数値表はもはや不要である．これらは，疑似乱数は rxxx()，分布関数は pxxx()，密度 (確率) 関数は dxxx()，クォンタイル関数は qxxx() といった統一的な名称を持つ．

11.1 疑似乱数

11.1.1 疑似乱数発生関連関数

.Random.seed は整数ベクトルで，R の乱数生成機構に対する乱数生成器 (RNG) の状態を含んでいる．これは保存し，元に戻すことができるが，ユーザが変更すべきではない．RNGkind() は使用中の RNG の種類を確認したり，設定したりするためのより分かりやすいインタフェイスである．set.seed() は乱数種を指定するための推薦できる方法である．

書式：

```
.Random.seed <- c(rng.kind, n1, n2, ...)
save.seed <- .Random.seed
RNGkind(kind=NULL, normal.kind=NULL)
set.seed(seed, kind=NULL)
```

引数：

kind　文字または NULL．もし kind が文字列なら，R の RNG を希望するものに設定できる．もしこれが NULL なら現在使用中の RNG を返す． default を指定すると R の既定手法に戻る

normal.kind　文字または NULL．もしこれが文字列なら正規乱数生成の手法を設定する． default を指定すると R の既定手法に戻る

seed　乱数種を設定する単一の値で，整数と解釈される

rng.kind　上の kind に対する範囲． 0:k 内の整数コード

n1,n2,...　整数．いくつ必要か (rng.kind に依存する) は以下の説明を見よ

現在使用できる RNG は 6 種類ある (詳細がヘルプ文章で得られる) が既定の "Mersenne-Twister" 及び R1.7 以前の既定手法であった"Marsaglia-Multicarry"

[*1] 多変量分布を含む多くのその他の確率分布用の関数が R の貢献パッケージにある．

以外は実際上無意味と思われる．

- "Marsaglia-Multicarry"：George Marsaglia が ML sci.stat.math に投稿した記事 (Sep. 29, 1997) で推薦した multiply-with-carry RNG が使われる．周期 2^60 以上を持ち，(Marsaglia によれば) 全てのテストをパスする ．乱数種は二つの整数 (なんでも良い)
- "Mersenne-Twister": Matsumoto & Nishimura[*2] による．周期 $2^{19937}-1 = 4.3e6001$ を持つ twisted GFSR 法であり，(全周期に渡り) 引き続く 623 次元空間中に一様分布する．乱数種は 32 ビット整数の 624 個のセットと，セット中の現在位置を表す整数である
- "user-supplied": ユーザ定義の発生法を使う．詳細は Random.user() を見よ

normal.kind 引数は文字列 "Kinderman-Ramage", "Buggy Kinderman-Ramage", "Ahrens-Dieter", "Box-Muller", "Inversion" (既定，qnorm() の解説を見よ)，もしくは"user-supplied" のどれかである．

set.seed() はその単一の整数引数を元に，必要なだけの乱数種を設定できる．これは小さな整数引数を指定して全く異なる乱数種を得る簡単な方法であり，より複雑な手法 (特に "Mersenne-Twister") に対する適正な乱数種を得ることができる．

.Random.seed は整数ベクトルで，その最初の要素は RNG と正規乱数生成法をコーディングしている．下二桁は k を利用可能な RNG の種類として範囲 0:(k-1) にある．(0 で始まる) 百の桁は正規乱数生成法の種類を表す．根底にある C コードでは .Random.seed[-1] は unsigned である．従って R では，符号無し整数の符号付き整数による表現のせいで，.Random.seed[-1] は負になり得る．RNGkind() は呼び出し前に使われていた RNG と正規乱数のタイプを表す要素 2 の文字ベクトルであり，もしどちらも NULL でなければ返り値は不可視である．RNGversion() は同じ情報を返す．set.seed() は NULL を返すが，不可視でコンソールに出力されない．

注意： 最初如何なる乱数種も存在せず，必要になった時に現在の時間から新しいものがつくり出される．従って既定では，異なったセッションは異なったシミュレーション結果を与える．.Random.seed は，少なくともシステム生成器に対する一様乱数疑似乱数に対する乱数種を保存する．他の生成器の状態は必ずしも保存しない．特に Box-Muller 正規乱数生成器の状態は保存しない．もし後で作業を再現したかったら .Random.seed を設定する代わりに set.seed() を呼び出そう．.Random.seed はユーザの作業スペースの中だけで探される．

注意： 全ての一様疑似乱数は 32 ビット整数を実数に変換したものであり，実際の周期とは違って $2^{32} = 4,294,967,296$ 通りの異なった値しか取らず， 周期内でもいつか同じ値を返す可能性がある．

```
> runif(1); .Random.seed[1:6]                    # 既定の乱数種は長さ 626 の整数 (一部だけ表示)
[1] 0.2397553
[1]   400 12109 21780  3671    NA    NA
> runif(1); .Random.seed[1:6]
```

[*2] 日本語版 Wikipedia の記事「メルセンヌ・ツイスタ」に詳しい解説がある．

```
[1] 0.5957438
[1]   400 12347 18399 17610    NA    NA

# もし乱数種がなければ，新しいものが「ランダム」に作られる
> rm(.Random.seed); runif(1); .Random.seed[1:6]
[1] 0.8409342
[1]   400 13719 17376 24694    NA    NA

# 乱数種を初期化，再現するには set.seed() 関数を使う
> set.seed(111111); runif(5)
[1] 0.4738884 0.7371098 0.3933957 0.8484342 0.2935077
> set.seed(222222); runif(5)
[1] 0.9043933 0.5916865 0.1318843 0.4878007 0.2776533
> set.seed(111111); runif(5)                        # 元に戻る
[1] 0.4738884 0.7371098 0.3933957 0.8484342 0.2935077
```

11.1.2 ユーザ定義の疑似乱数発生器

関数 `RNGkind()` はユーザがコードした一様疑似乱数と正規疑似乱数生成器の使用を許す．

ユーザー指定の一様 RNG は動的にロードされたコンパイル済みコードのエントリ点から呼び出される．ユーザはエントリ点 `user unif rand` を提供する必要があり，これは引数はなく，倍精度実数へのポインタを返す．

オプションとして，ユーザは `RNGkind` (もしくは `set.seed`) が呼び出されたとき引数 `unsigned int` で呼び出されるエントリ点 `user unif init` を提供することができ，ユーザの RNG コードを初期化するために使うことが想定されている．2 個の引数は「乱数種」を設定するのに使われることを想定している．それは `set.seed` への `seed` 引数であるか，もし `RNGkind` が呼び出されるならば本質的にランダムな乱数種である．

もしこれらの関数だけが提供されると，生成器に対する如何なる情報も `.Random.seed` に記録されない．オプションとして，引数無しに呼び出されると，乱数種の数と乱数種の整数配列へのポインタを返す関数 `user unif nseed` と `user unif seedloc` を与えることができる．`GetRNGstate` と `PutRNGstate` への呼び出しは，この配列を `.Random.seed` へ，そして `.Random.seed` からコピーする．ユーザ定義の正規 RNG は単一のエントリ点 `user norm rand` で指定され，これは引数を持たず，倍精度実数へのポインタを返す．

注意： 全てのコンパイルコードと同様に，これらの関数指定のミスは R をクラッシュさせる．タイプチェックのためにヘッダファイル `R ext/Random.h` をインクルードすること．

11.1.3 その他の疑似乱数発生法

CRAN にある R の非標準パッケージにも幾つかの疑似乱数発生関数がある．特に注目すべきものとして，パッケージ `SuppDists` 中にある G. Marsaglia による周期が $10e^{9824}$ を越える疑似乱数発生関数 `rMWC1019()` がある．これは現在の R の既定である Mersenne-Twister 法に比べ，コードも簡略で発生速度が 2 倍程度早い．また，標準の疑似乱数検査を全てパスし，$2, 3, \ldots, 1018$ 次元空間に「一様に分布」するとされる．またパッケージ `rlecuyer`, `rsprng` は並列処理用に疑似乱数を並列生成する機能を提供する．

```
# 一億個の一様疑似乱数の発生速度の比較
> library(SuppDists)                          # パッケージ SuppDists 読み込み
> system.time(runif(1e8))                     # 既定の Mersenne-Twister 法
   ユーザ     システム        経過
    6.432       0.480       6.914
> system.time(rMWC1019(1e8))                  # rMWC1019 関数，約 2 倍高速
   ユーザ     システム        経過
    3.040       1.952      11.232
```

11.2　連続分布

11.2.1　一様分布

これらの関数は区間 min から max 上の一様分布に関する情報を与える．dunif() は密度関数，punif() は分布関数，qunif() はクォンタイル関数，そして runif() は乱数を与える．

書式：

```
dunif(x, min=0, max=1, log=FALSE)
punif(q, min=0, max=1, lower.tail=TRUE, log.p=FALSE)
qunif(p, min=0, max=1, lower.tail=TRUE, log.p=FALSE)
runif(n, min=0, max=1)
```

引数：

x,q　クォンタイルのベクトル

p　確率のベクトル

n　必要な乱数の数

min，max　分布の下限と上限 (既定値はそれぞれ 0 と 1)

log，log.p　論理値．もし TRUE なら確率 p は対数値 log(p) とされる

lower.tail　論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

もし min や max が指定されないと，それらはそれぞれ既定値の 0 や 1 とされる．一様分布は密度関数

$$f(x) = \frac{1}{\mathtt{max} - \mathtt{min}}, \qquad \mathtt{min} \le x \le \mathtt{max}$$

を持つ．もし min == max なら極限例として X == u とされる．

```
> u <- runif(20)                              # 区間 [0,1) 上の一様疑似乱数 20 個
> all.equal(punif(u), u)                      # punif(u) は常に u に等しいはず
[1] TRUE
> all.equal(dunif(u), rep(1,20))              # dunif(u) は常に 1 に等しいはず
[1] TRUE
> var(runif(10000))                           # 理論分散 (1/12=.08333) に近いはず
[1] 0.08385671
```

```
> A <- 1; B <- 1; runif(10, min=A, max=B)     # min と max が等しいと一点分布とされる
 [1] 1 1 1 1 1 1 1 1 1 1

> A <- 1; B <- 0; runif(10, min=A, max=B)     # min > max なら NaN が生成され警告がでる
 [1] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
```

```
Warning message:
In runif(10, min = A, max = B) :  NA が生成されました
```

11.2.2 正規分布

dnorm() は平均 mean，標準偏差 sd の正規分布の密度関数，pnorm() は分布関数，qnorm() はクォンタイル関数，そして rnorm() は乱数を与える．

書式：

```
dnorm(x, mean=0, sd=1, log=FALSE)
pnorm(q, mean=0, sd=1, lower.tail=TRUE, log.p=FALSE)
qnorm(p, mean=0, sd=1, lower.tail=TRUE, log.p=FALSE)
rnorm(n, mean=0, sd=1)
```

引数：

x,q　クォンタイルのベクトル

p　確率のベクトル

n　必要な乱数の数

mean　平均のベクトル (既定値 0)

sd　標準偏差のベクトル (既定値 1)

log,log.p　論理値．もし TRUE なら確率 p は対数値 log(p) とされる

lower.tail　論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

もし mean または sd が指定されないと，それぞれ既定値 0 と 1 が使われる．正規分布は密度関数

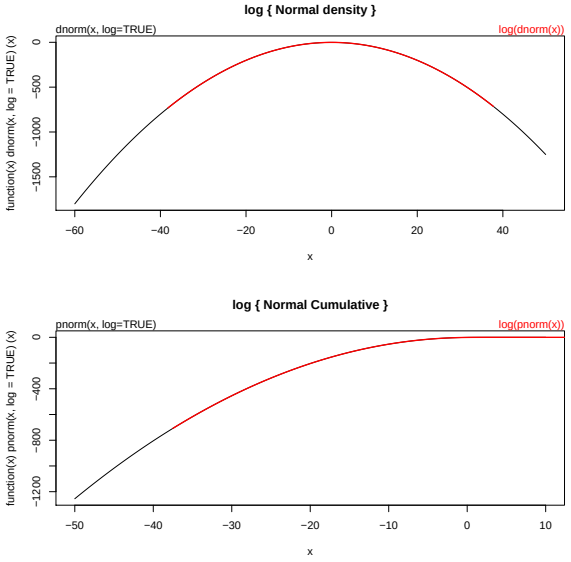
$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)}$$

を持つ，ここで μ は平均，σ は標準偏差である．qnorm() は Wichura のアルゴリズム AS 241 に基づき，16 桁まで正確な値を与える．

```
> all.equal(dnorm(0), 1/ sqrt(2*pi))              # 幾つかの点での密度関数の値をチェック
[1] TRUE
> all.equal(dnorm(1), exp(-1/2)/ sqrt(2*pi))
[1] TRUE
> all.equal(dnorm(1), 1/ sqrt(2*pi*exp(1)))
[1] TRUE
```

```
# 分布，密度関数のプロット（次の図を参照）．オプション log=TRUE で対数値を使用
> par(mfrow=c(2,1))
> plot(function(x) dnorm(x, log=TRUE), -60, 50,
                        main = "log { Normal density }")
> curve(log(dnorm(x)), add=TRUE, col="red",lwd=2)
> mtext("dnorm(x, log=TRUE)", adj=0);
        mtext("log(dnorm(x))", col="red", adj=1)

> plot(function(x) pnorm(x, log=TRUE), -50, 10,
                        main = "log { Normal Cumulative }")
> curve(log(pnorm(x)), add=TRUE, col="red",lwd=2)
> mtext("pnorm(x, log=TRUE)", adj=0);
        mtext("log(pnorm(x))", col="red", adj=1)
```

```
# いわゆる error function は次のように定義される
> erf <- function(x) 2 * pnorm(x * sqrt(2)) - 1

# いわゆる complementary error function は次のように定義される
> erfc <- function(x) 2 * pnorm(x * sqrt(2), lower=FALSE)
```



対数正規分布の密度関数 (上) と分布関数 (下) のプロット．関数値は対数尺度．

11.2.3　対数正規分布

dlnorm は対数正規分布の密度関数，plnorm は分布関数，qlnorm はクォンタイル関数，そして rlnorm は乱数を与える．平均は meanlog，標準偏差は sdlog となる．

書式：

```
dlnorm(x, meanlog=0, sdlog=1, log=FALSE)
plnorm(q, meanlog=0, sdlog=1, lower.tail=TRUE, log.p=FALSE)
qlnorm(p, meanlog=0, sdlog=1, lower.tail=TRUE, log.p=FALSE)
rlnorm(n, meanlog=0, sdlog=1)
```

引数：

x，q　クォンタイルのベクトル

p　確率のベクトル

n　必要な乱数の数

meanlog,sdlog　対数尺度での平均と標準偏差．既定値はそれぞれ 0,1

log，log.p　論理値．もし TRUE なら確率 p は対数値 log(p) とされる

lower.tail　論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

対数正規分布は密度関数

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

を持ち，μ と σ は対数値の平均と標準偏差である．平均は $\exp(\mu + 1/2\sigma^2)$，分散は

$\exp(2\mu+\sigma^2)(\exp(\sigma^2)-1)$ となり，従って変動係数は $(\exp(\sigma^2)-1)^{1/2}$ であり，σ が小さければ (例えば $\sigma < 1/2$)，ほぼ σ に等しい．

注意：累積ハザード関数 $H(t) = -\log(1-F(t))$ は `-plnorm(t, r, lower = FALSE, log = TRUE)` で計算できる．

```
> all.equal(dlnorm(1),dnorm(0))                    # 正規分布と対数正規分布の関係
[1] TRUE
```

11.2.4 ガンマ分布

`dgamma()` はパラメータ `shape` と `scale` のガンマ分布の密度関数，`pgamma()` は分布関数，`qgamma()` はクォンタイル関数，そして `rgamma()` は乱数を与える．

書式：

```
dgamma(x, shape, rate=1, scale=1/rate, log=FALSE)
pgamma(q, shape, rate=1, scale=1/rate, lower.tail=TRUE, log.p=FALSE)
qgamma(p, shape, rate=1, scale=1/rate, lower.tail=TRUE, log.p=FALSE)
rgamma(n, shape, rate=1, scale=1/rate)
```

引数：

`x, q` クォンタイルのベクトル
`p` 確率のベクトル
`n` 必要な乱数の数
`rate` スケール指定のもう一つの方法
`shape,scale` 形状およびスケールパラメータ
`log, log.p` 論理値．もし `TRUE` なら確率 `p` は対数値 `log(p)` とされる
`lower.tail` 論理値．もし `TRUE` (既定値) なら確率は下側確率 `P(X <= x)`，さもなければ上側確率 `P(X > x)` とされる

もし `scale` が省略されると，既定値の `1` とされる．パラメータ `shape` $= a > 0$ で `scale` $= s > 0$ のガンマ分布は密度関数

$$f(x) = \frac{1}{s^a \Gamma(a)} x^{a-1} e^{-x/s}, \quad x > 0,$$

平均と分散はそれぞれ as と as^2 である．`pgamma()` はアルゴリズム AS 39 を用いる．

注意：S のパラメータ化は `shape` と d `rate` により，`scale` パラメータは持たない．累積ハザード関数 $H(t) = -\log(1-F(t))$ は `-pgamma(t, ..., lower = FALSE, log = TRUE)` で計算できる．`pgamma()` は不完全ガンマ関数と密接な関係がある．Abramowitz & Stegun 6.5.1 によれば不完全ガンマ関数は

$$P(a,x) = \frac{1}{\Gamma(a)} \int_0^x t^{a-1} e^{-t} dt$$

と定義され，$P(a,x)$ は `pgamma(x, a)` と等しい．他の文献 (例えば Karl Pearson の 1922 年の数表) は正規化定数を省略しており，不完全ガンマ関数は `pgamma(x,a)*gamma(a)`

とされる．

関連：ガンマ関数は gamma()，ベータ分布は dbeta()，ガンマ分布の特殊例であるカイ 2 乗分布は dchisq().

```
> -log(dgamma(1:4, shape=1))
[1] 1 2 3 4
> p <- (1:9)/10
> pgamma(qgamma(p,shape=2), shape=2)
[1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
> 1 - 1/exp(qgamma(p, shape=1))
[1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

11.2.5　ベータ分布

dbeta() はパラメータ shape1, shape2 (そしてオプションの非心度パラメータ ncp) を持つベータ分布の密度関数，pbeta() は分布関数，qbeta() はクォンタイル関数，そして rbeta() は乱数を与える．

書式：

```
dbeta(x, shape1, shape2, ncp=0, log=FALSE)
pbeta(q, shape1, shape2, ncp=0, lower.tail=TRUE, log.p=FALSE)
qbeta(p, shape1, shape2, lower.tail=TRUE, log.p=FALSE)
rbeta(n, shape1, shape2)
```

引数：

x, q　クォンタイルのベクトル

p　確率のベクトル

n　必要な乱数の数

shape1,shape2　ベータ分布の正パラメータ

ncp　非心度パラメータ

log, log.p　論理値．もし TRUE なら確率 p は対数値 log(p) とされる

lower.tail　論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

パラメータ shape1 $= a > 0$, shape2 $= b > 0$ のベータ分布の密度関数は

$$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}, \quad 0 \le x \le 1$$

であり，$x = 0, 1$ に於ける境界値は連続性から (極限として) 定義される．pbeta() は不完全ベータ関数と密接な関係がある．Abramowitz & Stegun 6.6.1, 6.6.2 によれば不完全ベータ関数は

$$I_x(a,b) = \frac{B_x(a,b)}{B(a,b)}, \quad \text{ここで} \quad B_x(a,b) = \int_0^x t^{a-1}(1-t)^{b-1} dt$$

と定義され，$B(a,b) = B_1(a,b)$ はベータ関数 beta() である．$I_x(a,b)$ は pbeta(x,a,b) に等しい．

関連：ベータ関数は beta()，ガンマ分布は dgamma().

```
> x <- seq(0, 1, length=21)
> dbeta(x, 1, 1)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
> pbeta(x, 1, 1)
 [1] 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65
[15] 0.70 0.75 0.80 0.85 0.90 0.95 1.00
```

11.2.6 カイ 2 乗分布

dchisq() は自由度 df，そしてオプションの非心パラメータ ncf を持つカイ 2 乗 (χ^2) 分布の密度関数，pchisq() は分布関数，qchisq() はクォンタイル関数，そして rchisq() は乱数を与える．

書式：

```
dchisq(x, df, ncp=0, log=FALSE)
pchisq(q, df, ncp=0, lower.tail=TRUE, log.p=FALSE)
qchisq(p, df, ncp=0, lower.tail=TRUE, log.p=FALSE)
rchisq(n, df, ncp=0)
```

引数：

x, q　クォンタイルのベクトル

p　確率のベクトル

n　必要な乱数の数

df　自由度 (非負であるが，非整数であっても良い)

ncp　非心 度パラメータ (非負)．約 1417 以上の ncp 値は現在のところ pchisq() と qchisq() では許されない

log,log.p　論理値．もし TRUE なら確率 p は対数値 log(p) とされる

lower.tail　論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

自由度 df $= n$ のカイ 2 乗分布は密度関数

$$f_n(x) = \frac{1}{2^{n/2}\Gamma(n/2)} x^{n/2-1} e^{-x/2}, \quad x > 0$$

を持つ．平均と分散はそれぞれ n と 2n である．自由度 df，非心度パラメータ ncp $= \lambda$ の非心カイ 2 乗分布は密度関数

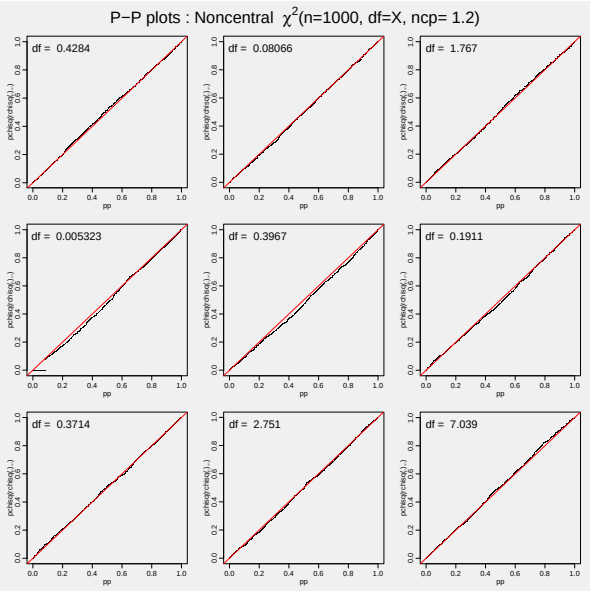
$$f(x) = \exp(-\lambda/2) \sum_{r=0}^{\infty} \frac{(\lambda/2)^r}{r!} \ \mathtt{dchisq}(x, df + 2r), \quad x \geq 0$$

を持つ．整数の n に対しては，これは分散 1 の n 個の独立な正規変数の和の分布であり，λ は正規分布の平均の和である．更に平均は $n + \lambda$，分散は $2(n + 2\lambda)$，3 次の中心化モーメントは $8(n + 3\lambda)$ である．自由度 df $= n$ は非整数であっても良く，非心度 λ が正ならば $n = 0$ すら許される．

関連： 自由度 n のカイ 2 乗分布は shape $= n/2$ で scale $= 2$ のガンマ分布と同一である．ガンマ分布 dgamma() を見よ．

```
> dchisq(1, df=1:3)
[1] 0.2419707 0.3032653 0.2419707
> pchisq(1, df= 3)
[1] 0.1987480
> pchisq(1, df= 3, ncp = 0:4)# includes the above
[1] 0.19874804 0.13229855 0.08787311 0.05824691 0.03853592

> x <- 1:10
> all.equal(dchisq(x,df=2),dexp(x,1/2))     # 自由度 2 のカイ 2 乗分布は特別な指数分布である
[1] TRUE
> all.equal(pchisq(x, df=2), pexp(x, 1/2))
[1] TRUE

# 非心度パラメータ ncp>0 なら df=0 が可能．Z0 は原点にマスを持つ!
> Z0 <- rchisq(100, df = 0, ncp = 2.)
> stem(Z0)                                          # 幹葉表示
 The decimal point is at the |
  0 | 00000000000000000000000000000000000000000011234666788123366778899
  2 | 2334445555901113788
  4 | 067890004488
  6 | 349
  8 | 3
 10 | 5
 12 |
 14 |
 16 | 8
```

```
# 様々な自由度で 1000 個の乱数の P-P プロットを描く（次の図を参照）
> L <- 1.2; n <- 1000; pp <- ppoints(n)
> op <- par(mfrow = c(3,3), mar= c(3,3,1,1)+.1, mgp= c(1.5,.6,0),
            oma = c(0,0,3,0))
> for(df in 2^(4*rnorm(9))) {
    plot(pp, sort(pchisq(rr <- rchisq(n,df=df, ncp=L), df=df, ncp=L)),
         ylab="pchisq(rchisq(.),.)", pch=".")
    mtext(paste("df = ",formatC(df, digits = 4)), line= -2, adj=0.05)
    abline(0,1,col=2)
  }
> mtext(expression("P-P plots : Noncentral  "*
                   chi^2 *"(n=1000, df=X, ncp= 1.2)"),
                   cex = 1.5, font = 2, outer=TRUE)
> par(op)
```



様々な自由度での非心カイ 2 乗乱数の P-P(確率・確率) プロット

11.2.7 t 分布

dt() は自由度 df (そして非心度パラメータ ncp の) t 分布の密度関数，pt() は分布関数，qt() はクォンタイル関数，そして rt() は乱数を与える．

書式：

```
dt(x, df, ncp=0, log=FALSE)
pt(q, df, ncp=0, lower.tail=TRUE, log.p=FALSE)
qt(p, df, lower.tail=TRUE, log.p=FALSE)
rt(n, df)
```

引数：

x，q　クォンタイルのベクトル

p　確率のベクトル

n　必要な乱数の数

df　自由度 (正だが，整数でなくても良い)

ncp　非心度パラメータ δ．現在 pt() と dt() でのみ有効で，37.62 以下という制限

log，log.p　論理値．もし TRUE なら確率 p は対数値 log(p) とされる

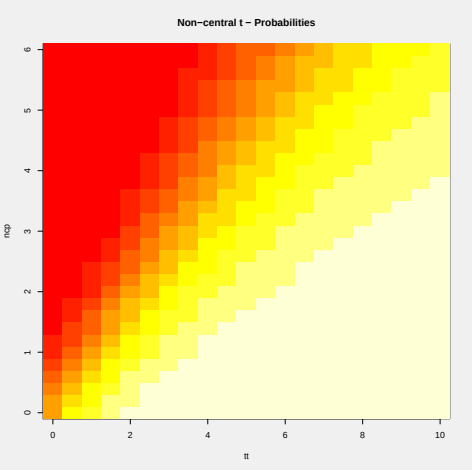
lower.tail　論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

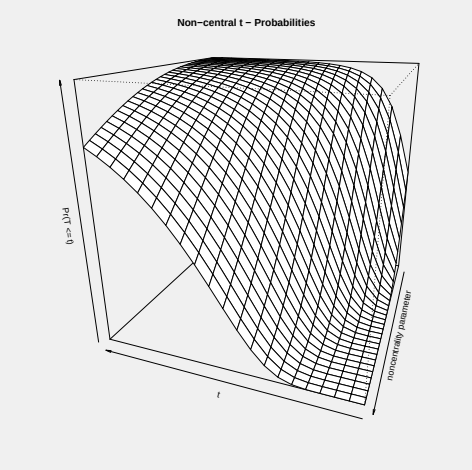
自由度 df $= n$ の t 分布は密度関数

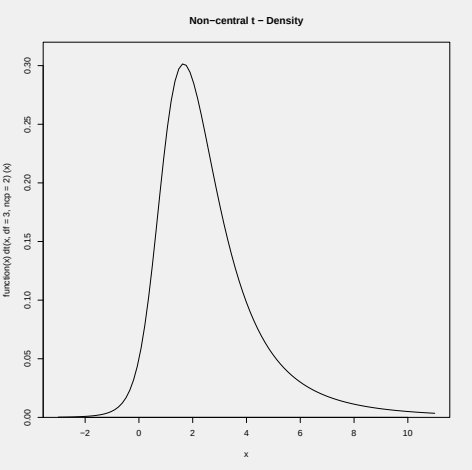
$$f(x) = \frac{\Gamma((n+1)/2)}{\sqrt{(n\pi)}\Gamma(n/2)} \left(1 + \frac{x^2}{n}\right)^{-(n+1)/2}, \quad -\infty < x < \infty$$

を持つ．平均は 0 $(n > 1)$ であり，分散は $n/(n-2)$ $(n > 2)$ である．パラメータ $(df, \Delta) = (\mathtt{df}, \mathtt{ncp})$ を持つ一般の t 分布は $T(df, \Delta) = (U + \Delta)/(\Xi(df)/\sqrt{(df)})$ の分布である．ここで U と $\Xi(df)$ は独立な確率変数で，U は標準正規分布に従い，$\Xi(df)^2$ は自由度 df のカイ 2 乗分布に従うとする．これの最大の用途は t 検定の検出力の計算である．正規分布 $\mathrm{N}(\mu, \sigma^2)$ に従う独立同分布データ $X_1, X_2, \ldots, X_n$ の標本平均を m_X，標本標準偏差を S とし，$T = (m_X - \mu_0)/(S/\sqrt{(n)})$ と置く．すると T は自由度 $n-1$，非心度パラメータ ncp $= (\mu - \mu_0) \times \sqrt{(n)}/\sigma$ の非心 t 分布に従う．

```
> 1 - pt(1:5, df = 1)                             # 非心 t 分布 (次の図を参照)
[1] 0.25000000 0.14758362 0.10241638 0.07797913 0.06283296
> qt(.975, df = c(1:10,20,50,100,1000))
 [1] 12.706205 4.302653 3.182446 2.776445 2.570582 2.446912 2.364624
 [8]  2.306004 2.262157 2.228139 2.085963 2.008559 1.983972 1.962339
> tt <- seq(0,10, len=21)
> ncp <- seq(0,6, len=31)
> ptn <- outer(tt,ncp, function(t,d) pt(t, df = 3, ncp=d))
> image(tt,ncp,ptn,zlim=c(0,1),
        main=t.tit<-"Non-central t - Probabilities")
> persp(tt,ncp,ptn, zlim=0:1, r=2, phi=20, theta=200, main=t.tit,
        xlab = "t", ylab = "noncentrality parameter",
        zlab = "Pr(T <= t)")
> op <- par(yaxs="i")
> plot(function(x) dt(x, df = 3, ncp = 2), -3, 11, ylim = c(0, 0.32),
       main="Non-central t - Density")
> par(op)
```







(上左) 非心度を変えたときの自由度 3 の t 分布の分布関数のイメージ図，
(上右) その鳥瞰図，
(下左) 自由度 3，非心度 2 の t 分布の密度関数

11.2.8　F 分布

`df()` は自由度 `df1`, `df2` (そしてオプションの非心度パラメータ `ncp`) を持つ F 分布の密度関数，`pf()` は分布関数，`qf()` はクォンタイル関数，そして `rf()` は乱数を与える．

書式：

```
df(x, df1, df2, log=FALSE)
pf(q, df1, df2, ncp=0, lower.tail=TRUE, log.p=FALSE)
qf(p, df1, df2, lower.tail=TRUE, log.p=FALSE)
rf(n, df1, df2)
```

引数：

`x, q`　クォンタイルのベクトル
`p`　確率のベクトル
`n`　必要な乱数の数
`df1, df2`　自由度
`ncp`　非心度パラメータ
`log, log.p`　論理値．もし `TRUE` なら確率 `p` は対数値 `log(p)` とされる
`lower.tail`　論理値．もし `TRUE` (既定値) なら確率は下側確率 `P(X <= x)`，さもなければ上側確率 `P(X > x)` とされる

自由度 df1 $= n_1$, df2 $= n_2$ の F 分布は密度関数

$$f(x) = \frac{\Gamma((n_1+n_2)/2)}{\Gamma(n_1/2)\Gamma(n_2/2)} \left(\frac{n_1}{n_2}\right)^{n_1/2} x^{n_1/2-1} \left(1+\frac{n_1}{n_2}x\right)^{-(n_1+n_2)/2}, \quad x>0$$

を持つ．これはそれぞれ n_1, n_2 個の独立標準正規変数の 2 乗平均の比の分布であり，従って対応自由度で除した 2 つのカイ 2 乗変数の比である．標準正規変数と m 個の独立標準正規変数の 2 乗平均の平方根の比はスチューデントの $\mathtt{t}_m$ 分布に従うので，その 2 乗は自由度 $(1, m)$ の F 分布に従う．非心 F 分布は再び分散 1 の独立正規変数の 2 乗平均の比であるが，分子の正規変数は 0 でない平均を持っても良く，ncp はこの平均の 2 乗和である．非心分布に付いての詳細はカイ 2 乗分布を参照せよ．

```
> x <- seq(0.001, 5, len = 100)                    # F 分布と t 分布の関係
> all.equal(df(x^2, 1, 5), dt(x, 5)/x)
[1] TRUE
> p <- seq(1/2, 0.99, length = 50)
> df <- 10
> rel.err <- function(x, y)
              ifelse(x==y, 0, abs(x-y)/mean(abs(c(x,y))))
> quantile(rel.err(qf(2*p-1, df1 = 1, df2 = df), qt(p, df)^2), 0.9)
         90%
8.700353e-16
```

11.2.9 コーシー分布

dcauchy() は位置パラメータ location，スケールパラメータ scale のコーシー分布の密度関数，pcauchy() は分布関数，qcauchy() はクォンタイル関数，そして rcauchy() は乱数を与える．

書式：

```
dcauchy(x, location=0, scale=1, log=FALSE)
pcauchy(q, location=0, scale=1, lower.tail=TRUE, log.p=FALSE)
qcauchy(p, location=0, scale=1, lower.tail=TRUE, log.p=FALSE)
rcauchy(n, location=0, scale=1)
```

引数：

x，q	クォンタイルのベクトル
p	確率のベクトル
n	必要な乱数の数
location,scale	位置とスケールパラメータ
log，log.p	論理値．もし TRUE なら確率 p は対数値 log(p) とされる
lower.tail	論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

もし location もしくは scale が指定されないと，それぞれ既定値の 0 と 1 が使われる．位置 ℓ，スケール s のコーシー分布の密度関数は以下で与えられる：

$$f(x) = \frac{1}{\pi s(1+((x-l)/s)^2)}, \quad -\infty < x < \infty.$$

関連：dcauchy(*, l=0, s=1) を一般化する dt().

```
> dcauchy(-1:4)
[1] 0.15915494 0.31830989 0.15915494 0.06366198 0.03183099 0.01872411
```

11.2.10　指数分布

dexp() は割合 rate (つまり平均が 1/rate) の指数分布の密度関数，pexp() は分布関数，qexp() はクォンタイル関数，そして rexp() は乱数を与える．

書式：

```
dexp(x, rate=1, log=FALSE)
pexp(q, rate=1, lower.tail=TRUE, log.p=FALSE)
qexp(p, rate=1, lower.tail=TRUE, log.p=FALSE)
rexp(n, rate=1)
```

引数：

x，q　クォンタイルのベクトル
p　確率のベクトル
n　観測値の数
rate　割合のベクトル
log，log.p　論理値．もし TRUE なら確率 p は対数値 log(p) とされる
lower.tail　論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

もし rate が指定されないと既定値 1 とされる．割合 λ の指数分布の密度関数は以下の式で与えられる：

$$f(x) = \lambda e^{-\lambda x}, \quad x \geq 0.$$

注意：累積ハザード関数 $H(t) = -\log(1 - F(t))$ は -pexp(t, r, lower = FALSE, log = TRUE) で計算できる．
関連：指数関数は exp()．ガンマ分布 dgamma()，ワイブル分布 dweibull() は指数分布を一般化する．

```
> dexp(1) - exp(-1)                                    # 0 になるはず
[1] 0
```

11.2.11　ロジスティック分布

dlogis() は位置パラメータ location とスケールパラメータ scale のロジスティック分布の密度関数，plogis() は分布関数，qlogis() はクォンタイル関数，そして rlogis() は乱数を与える．

書式：

```
dlogis(x, location=0, scale=1, log=FALSE)
plogis(q, location=0, scale=1, lower.tail=TRUE, log.p=FALSE)
qlogis(p, location=0, scale=1, lower.tail=TRUE, log.p=FALSE)
rlogis(n, location=0, scale=1)
```

引数：

x, q　クォンタイルのベクトル
p　確率のベクトル
n　必要な乱数の数
location　位置パラメータ
scale　スケールパラメータ
log,log.p　論理値．もし TRUE なら確率 p は対数値 log(p) とされる
lower.tail　論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

もし location または scale が省略されると，それぞれ既定値 0 と 1 が使われる．location $= m$ で scale $= s$ のロジスティック分布の分布関数は

$$F(x) = \frac{1}{1+\exp(-(x-m)/s)}, \quad -\infty < x < \infty$$

であり，密度関数は

$$f(x) = \frac{1}{s}\,\frac{\exp((x-m)/s)}{(1+\exp((x-m)/s))^2}, \quad -\infty < x < \infty$$

である．これは裾の長い分布であり，平均 m，分散 $(\pi s)^2/3$ を持つ．

```
> var(rlogis(4000, 0, s = 5))                    # 近似的に pi^2/3*5^2(絶対誤差 3 以下)
[1] 81.21852
```

11.2.12 ワイブル分布

dweibull() はパラメータ shape と scale のワイブル分布の密度関数，pweibull() は分布関数，qweibull() はクォンタイル関数，そして rweibull() は乱数を与える．

書式：

```
dweibull(x, shape, scale=1, log=FALSE)
pweibull(q, shape, scale=1, lower.tail=TRUE, log.p=FALSE)
qweibull(p, shape, scale=1, lower.tail=TRUE, log.p=FALSE)
rweibull(n, shape, scale=1)
```

引数：

x, q　クォンタイルのベクトル
p　確率のベクトル
n　必要な乱数の数
shape　形状パラメータ

scale スケールパラメータ．既定値 1
log，log.p 論理値．もし TRUE なら確率 p は対数値 log(p) とされる
lower.tail 論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

shape $= a$, scale $= b$ のワイブル分布の密度関数は次式で与えられる：

$$f(x) = \frac{a}{b}\left(\frac{x}{b}\right)^{a-1}\exp(-(x/b)^a), \quad x > 0$$

分布関数は $F(x) = 1 - \exp(-(x/b)^a)$，平均は $b\Gamma(1 + 1/a)$，そして分散は $b^2(\gamma(1 + 2/a) - (\gamma(1 + 1/a))^2)$ である．

注意：累積ハザード関数 $H(t) = -\log(1 - F(t))$ は -pweibull(t, a, b, lower = FALSE, log = TRUE) で計算され，これは単に $H(t) = (t/b)^a$ となる．
関連：ワイブル分布の特殊例である指数分布は dexp()．

```
> x <- c(0,rlnorm(50))                    # 指数分布はワイブル分布の特殊例であることを示す例
> all.equal(dweibull(x, shape = 1), dexp(x))
[1] TRUE
> all.equal(pweibull(x, shape = 1, scale = pi), pexp(x, rate = 1/pi))
[1] TRUE

> all.equal(pweibull(x,2.5,pi,lower=FALSE,log=TRUE), # 累積ハザード関数 H()
            -(x/pi)^2.5, tol=1e-15)
[1] TRUE
> all.equal(qweibull(x/11, shape = 1, scale = pi),
            qexp(x/11, rate=1/pi))
[1] TRUE
```

11.2.13 スチューデント化範囲分布

スチューデント化範囲 (Studentized range) 分布とは，R を n 個の標準正規標本の範囲，s^2 をそれと独立な自由度 df のカイ 2 乗分布に従う確率変数とすると，R/s の確率分布である．ptukey() は分布関数，qtukey() はクォンタイル関数を与える．

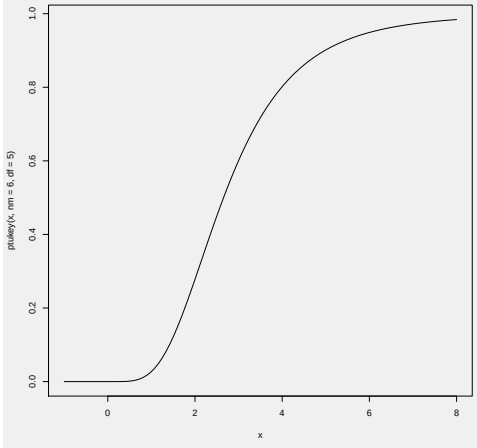
書式：
ptukey(q, nmeans, df, nranges=1, lower.tail=TRUE, log.p=FALSE)
qtukey(p, nmeans, df, nranges=1, lower.tail=TRUE, log.p=FALSE)

引数：
q クォンタイルのベクトル
p 確率のベクトル
nmeans 範囲に対する標本サイズ (各グループで同一)
df s に対する自由度
nranges その最大範囲を考慮するグループ数
log，log.p 論理値．もし TRUE なら確率 p は対数値 log(p) とされる
lower.tail 論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

もし nranges が 1 より大きければ，R は各々 nmeans 個の観測値からなる nranges グループの最大値である．

注意：ptukey() を求める数値積分に対してはルジャンドルの 16 点公式が使われる．計算はかなり時間を要し，特に ptykey() の逆関数を単純な割線法で求める qtukey() に対してそうである．qtukey() の精度は 4 桁であろう．

```
> if(interactive())
     curve(ptukey(x, nm=6, df=5), from=-1, to=8, n=101)

> (ptt <- ptukey(0:10, 2, df= 5))         # スチューデント化範囲分布の確率分布（次の図を参照）
 [1] 0.000000 0.488915 0.783562 0.912640 0.963257 0.983358 0.991851
 [8] 0.995714 0.997601 0.998583 0.999124
> (qtt <- qtukey(.95, 2, df= 2:11))
 [1] 6.07963 4.50065 3.92650 3.63535 3.46045 3.34408 3.26118 3.19917
 [9] 3.15106 3.11266

> summary(abs(.95 - ptukey(qtt,2, df = 2:11)))    # 精度は 8 桁より大きくないかも知れない
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
1.023e-13 3.101e-12 7.766e-11 1.419e-09 6.330e-10 1.175e-08
```



標本サイズ 6，自由度 5 のスチューデント化範囲分布の確率分布．

11.3 離散分布

11.3.1 無作為抽出とランダムな置換，sample()

sample() は x の要素から指定したサイズの標本を復元，もしくは非復元抽出する．

書式： sample(x, size, replace=FALSE, prob=NULL)

引数：

x　その中から抽出を行う少なくとも一つの要素を持つ (数値，複素数，文字列もしくは論理値) ベクトルであるか，正の整数

size　抽出される項目の数を与える非負整数

replace　復元抽出を行うか？

prob　抽出されるベクトルの要素に対する確率重みのベクトル

もし x が長さ 1 なら，抽出は 1:x から行われる．この便利な特性は x の長さが変化する場合は望ましくない結果を与える可能性がある．以下の resample() 関数例を見よ．

既定では size は length(x) に等しく，従って sample(x) は x (もしくは 1:x) のランダム置換を生成する．

オプションの prob 引数はベクトルの要素を得る重みのベクトルを与えるのに使うことができる．これらは和が 1 である必要はないが，非負で，全てが 0 であってはならない．もし replace が FALSE なら，これらの確率は逐次的に適用される．つまり，次の要素を選ぶ確率が残りの項目の確率に比例する．この場合，0 でない重みの数は少なくとも size 個なければならない．

```
> x <- 1:12
> sample(x)                                   # ランダムな置換
 [1] 12 11  8  7  3  4  5 10  6  9  2  1
> sample(x, replace=TRUE)                     # ブートストラップ抽出 (length(x)>1 の時だけ)
 [1]  9  1  1  6  4  9 12 12  4  2 10  8
> sample(c(0,1), 10, replace = TRUE)          # 10 回のベルヌイ試行
[1] 1 0 0 1 1 1 0 0 0 1
```

```
# より注意深いブートストラップ (プログラム中で sample() を使うときは注意). sample() 関数の落し穴
> x <- 1:10
> sample(x[x >  8])                                 # 長さ 2
[1]  9 10
> sample(x[x >  9])                                 # 長さ 10，つまり sample(1:10) とされる
 [1]  9  4  8  1  3  6  5  2 10  7
> try(sample(x[x > 10]))                       # エラーになる！ sample(numeric(0)) とされる
Error in sample(length(x), size, replace, prob) :
        invalid first argument

> resample <- function(x, size, ...)                # より安全な方法
   if(length(x) <= 1) { if(!missing(size) && size==0) x[FALSE] else x
   } else sample(x, size, ...)
> resample(x[x > 8])                                # 長さ 2
[1] 10  9
> resample(x[x > 9])                                # 長さ 1
[1] 10
> resample(x[x > 10])                               # 長さ 0
numeric(0)
```

11.3.2 二項分布

dbinom() はパラメータ size と prob の二項分布の確率関数，pbinom() は分布関数，qbinom() はクォンタイル関数，そして rbinom() は乱数を与える．

書式：

```
dbinom(x, size, prob, log=FALSE)
pbinom(q, size, prob, lower.tail=TRUE, log.p=FALSE)
qbinom(p, size, prob, lower.tail=TRUE, log.p=FALSE)
rbinom(n, size, prob)
```

引数：

x, q	クォンタイルのベクトル
p	確率のベクトル
n	必要な乱数の数
size	試行数
prob	各試行の成功確率
log, log.p	論理値．もし TRUE なら確率 p は対数値 log(p) とされる

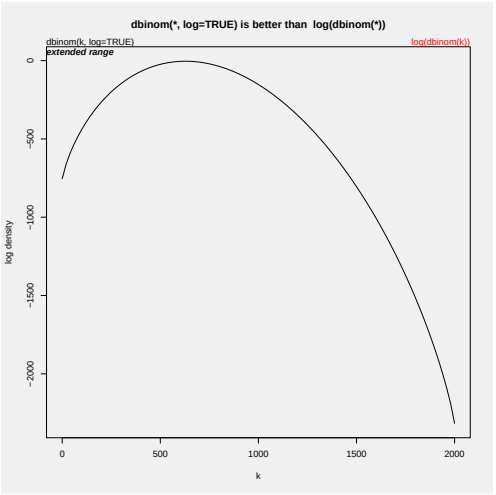
lower.tail 論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

size $= n$ で prob $= p$ の二項分布の確率関数は次式で与えられる：

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad x = 0, \ldots, n.$$

もし x のある要素が整数でなければ dbinom() の結果は 0 とされ，警告が出される．$p(x)$ は Loader のアルゴリズムを用いて計算される．クォンタイルは F を分布関数として $F(x) \geq p$ となる最小の値 x と定義される．もし size が整数でなければ，NaN が返される．

```
# Binomial(100,0.5) に対する確率 P(45<X<55) の計算 (次の図を参照)
> sum(dbinom(46:54, 100, 0.5))
[1] 0.6317984

# 拡大領域には log=TRUE を使う
> n <- 2000; k <- seq(0, n, by = 20)
> plot(k, dbinom(k,n,pi/10,log=TRUE), type='l', ylab="log density",
       main="dbinom(*, log=TRUE) is better than log(dbinom(*))")
> lines(k, log(dbinom(k, n, pi/10)), col='red', lwd=2)

# 極端な点は dbinom = 0 となるので省略する
> mtext("dbinom(k, log=TRUE)", adj=0)
> mtext("extended range", adj=0, line = -1, font=4)
> mtext("log(dbinom(k))", col="red", adj=1)
```



Binomial(2000, pi/10) の密度関数の対数値のグラフ．

11.3.3 負の二項分布

dnbinom() はパラメータ size と prob の負の二項分布の密度関数，pnbinom() は分布関数，qnbinom() はクォンタイル関数，そして rnbinom() は乱数を与える．

書式：

```
dnbinom(x, size, prob, mu, log=FALSE)
pnbinom(q, size, prob, mu, lower.tail=TRUE, log.p=FALSE)
qnbinom(p, size, prob, mu, lower.tail=TRUE, log.p=FALSE)
rnbinom(n, size, prob, mu)
```

引数：

x (非負整数の) クォンタイルのベクトル

q クォンタイルのベクトル

p 確率のベクトル

n 必要な乱数の数

size 成功の試行の目標数．もしくは拡散パラメータ (ガンマ混合分布の形状パラメータ)

prob 各試行に於ける成功確率

mu 平均によるもう一つのパラメータ化，以下を参照

log，log.p 論理値．もし TRUE なら確率 p は対数値 log(p) とされる

lower.tail 論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

パラメータ size $= n$ と prob $= p$ の負の二項分布の確率関数は次式で与えられる：

$$p(x) = \frac{\Gamma(x+n)}{\Gamma(n)x!}\, p^n(1-p)^x, \quad x = 0, 1, 2, \ldots$$

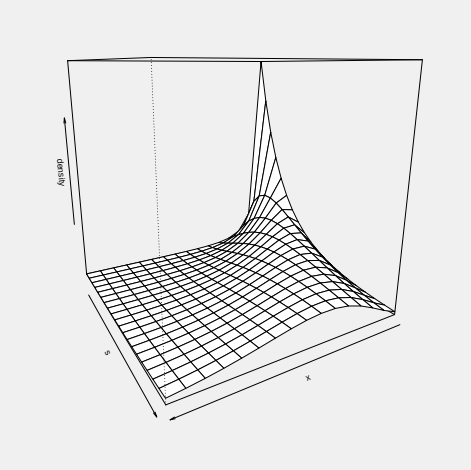
これは目標成功数が得られるまでのベルヌイ試行列中の失敗数を表す．

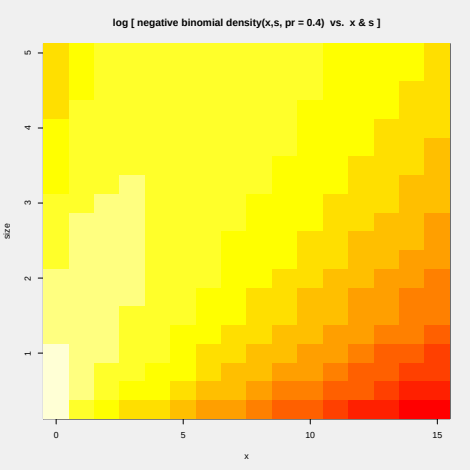
負の二項分布の確率関数は，平均がガンマ分布 (スケールパラメータ $(1 - \texttt{prob})/\texttt{prob}$ と形状パラメータ size を持つ pgamma) に従う混合ポアッソン分布として構成できる．(この定義は負の size でも意味を持つ．) このモデルでは prob $= \texttt{scale}/(1 + \texttt{scale})$ であり，平均は $\texttt{size}(1 - \texttt{prob})/\texttt{prob}$ となる．もう一つのパラメータ化 (生態学でしばしば用いられる) は，平均 mu, size そして拡散パラメータを用いる．ここで prob $= \texttt{size}/(\texttt{size} + \texttt{mu})$ である．このパラメータ化では分散は $\texttt{mu} + \texttt{mu}^2/\texttt{size}$ となる．もし x のある要素が整数でなければ，dnbinom() の結果は 0 とされ，警告がでる．クォンタイルは F を分布関数として $F(x) \geq p$ となる最小の値 x と定義される．

関連：二項分布は dbinom()，ポアッソン分布は dpois()，負の二項分布の特殊例である幾何分布は dgeom()．

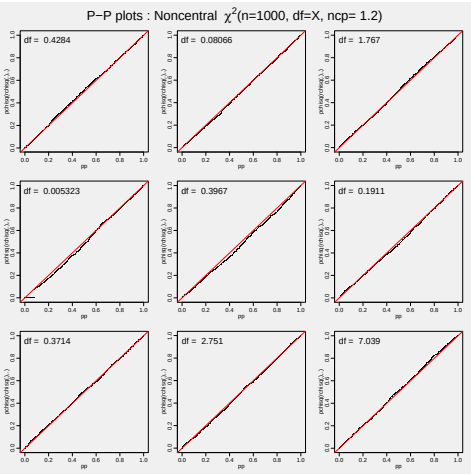
```
# prob=0.4 に対する負の二項分布の確率関数 (次の図を参照)
> x <- 0:15; size <- (1:20)/4                    # クォンタイルとサイズの組合せ
> persp(x,size,dnb <- outer(x,size,function(x,s)dnbinom(x,s,pr=0.4)),
        xlab = "x", ylab = "s", zlab="density", theta = 150)
> title(tit <- "negative binomial density(x,s, pr = 0.4) vs. x & s")
> image  (x,size, log10(dnb), main= paste("log [",tit,"]")) # その対数値のイメージ図
> contour(x,size, log10(dnb),add=TRUE)
```

```
# 平均によるパラメータ化の例 (size=1,10,100 に対する乱数のヒストグラム)
> x1 <- rnbinom(500, mu = 4, size = 1)
> x2 <- rnbinom(500, mu = 4, size = 10)
> x3 <- rnbinom(500, mu = 4, size = 100)
> h1 <- hist(x1, breaks = 20, plot = FALSE)
> h2 <- hist(x2, breaks = h1$breaks, plot = FALSE)
> h3 <- hist(x3, breaks = h1$breaks, plot = FALSE)
> barplot(rbind(h1$counts, h2$counts, h3$counts),
          beside = TRUE, col = c("red","blue","cyan"),
          names.arg = round(h1$breaks[-length(h1$breaks)]))
```





(左) prob=0.4 に対する負の二項分布の確率関数．(右) その対数値のイメージ図



prob=0.4 に対する負の二項分布の確率関数の平均によるパラメータ化の例 (size=1,10,100 に対する乱数のヒストグラム)

11.3.4 ポアッソン分布

dpois() はパラメータ lambda のポアッソン分布の (対数) 確率関数，ppois() は (対数) 分布関数，qpois() はクォンタイル関数，そして rpois() は乱数を計算する．

書式：

```
dpois(x, lambda, log=FALSE)
ppois(q, lambda, lower.tail=TRUE, log.p=FALSE)
qpois(p, lambda, lower.tail=TRUE, log.p=FALSE)
rpois(n, lambda)
```

引数：

x	(非負整数の) クォンタイルのベクトル
q	クォンタイルのベクトル
p	確率のベクトル
n	返される乱数の数
lambda	正の平均のベクトル
log, log.p	論理値．もし TRUE なら確率 p は対数値 log(p) とされる

lower.tail 論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

平均 lambda $= \lambda$ のポアッソン分布の確率関数は次式で与えられる：

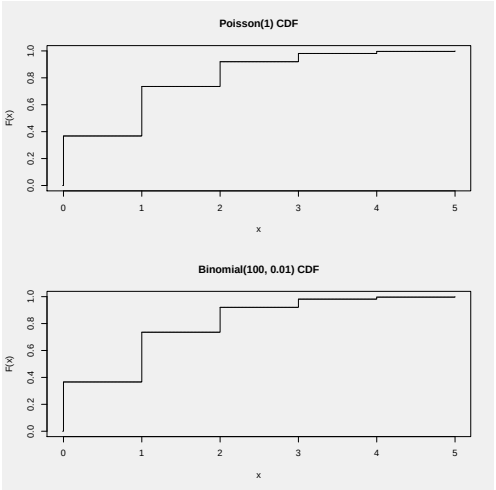
$$p(x) = e^{-\lambda}\,\frac{\lambda^x}{x!}, \quad x = 0, 1, 2, \ldots$$

もし x のある要素が整数でなければ，dpois() の結果は 0 とされ，警告がでる．$p(x)$ は Loader のアルゴリズムで計算される，dbinom() を参照せよ．クォンタイルは左連続である．qgeom(q, prob) は $\mathbf{P}(X \le x) < q$ となる最大の整数 x である．lower.tail = FALSE と置くと，既定では lower.tail = TRUE では 1 が返される場合でも，相当より正確な結果を得ることができる，下の参考例を見よ．

関連：二項分布は dbinom()，負の二項分布は dnbinom()．

```
> -log(dpois(0:7, lambda=1) * gamma(1+ 0:7))      # 1 になるはず
[1] 1 1 1 1 1 1 1 1
# 平均 4 のポアッソン乱数 50 個の度数分布表
> Ni <- rpois(50, lam= 4); table(factor(Ni, 0:max(Ni)))
 0  1  2  3  4  5  6  7  8
 1  1  7  9 12  9  8  2  1
> 1 - ppois(10*(15:25), lambda=100)               # キャンセルにより 0 となる
 [1] 1.233094e-06 1.261664e-08 7.085799e-11 2.252643e-13 4.440892e-16
 [6] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
[11] 0.000000e+00
> ppois(10*(15:25), lambda=100, lower=FALSE)      # キャンセルされない
 [1] 1.233094e-06 1.261664e-08 7.085800e-11 2.253110e-13 4.174239e-16
 [6] 4.626179e-19 3.142097e-22 1.337219e-25 3.639328e-29 6.453883e-33
[11] 7.587807e-37
```

```
> par(mfrow = c(2, 1))
> x <- seq(-0.01, 5, 0.01)
> plot(x, ppois(x, 1), type="s", ylab="F(x)", main="Poisson(1) CDF")
> plot(x, pbinom(x, 100, 0.01),type="s", ylab="F(x)",
       main="Binomial(100, 0.01) CDF")
```



二項分布のポアッソン近似の例．ppois(x,1) (上) と pbinom(x,100,0.01) (下) の分布関数の比較．

11.3.5 幾何分布

dgeom() はパラメータ prob の幾何分布の密度関数，pgeom() は分布関数，qgeom() はクォンタイル関数，そして rgeom() は乱数を与える．

書式：
dgeom(x, prob, log=FALSE)
pgeom(q, prob, lower.tail=TRUE, log.p=FALSE)
qgeom(p, prob, lower.tail=TRUE, log.p=FALSE)
rgeom(n, prob)

引数：
x, q　ベルヌイ試行列で，成功が起きるまでに必要だった失敗の数
p　確率のベクトル
n　必要な乱数の数
prob　各試行に於ける成功確率
log, log.p　論理値．もし TRUE なら確率 p は対数値 log(p) とされる
lower.tail　論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

prob $= p$ の幾何分布の確率関数は次式で与えられる：

$$p(x) = p(1-p)^x, \quad x = 0, 1, 2, \ldots$$

もし x 中のある要素が整数でなければ 0 が返され，警告がでる．クォンタイルは F を分布関数として $F(x) \geq p$ となる最大の x と定義される．

関連：幾何分布の一般化である負の二項分布 dnbinom().

```
> qgeom((1:9)/10, prob = .2)
[1]  0  0  1  2  3  4  5  7 10
# 幾何分布乱数 20 個の度数分布表を作る
> Ni <- rgeom(20, prob = 1/4); table(factor(Ni, 0:max(Ni)))
 0  1  2  3  4  5  6  7  8  9 10 11 12
 3  4  5  2  2  0  1  1  1  0  0  0  1
```

11.3.6 超幾何分布

dhyper() は超幾何分布の密度関数，phyper() は分布関数，qhyper() はクォンタイル関数，そして rhyper() は乱数を与える．

書式：
dhyper(x, m, n, k, log=FALSE)
phyper(q, m, n, k, lower.tail=TRUE, log.p=FALSE)
qhyper(p, m, n, k, lower.tail=TRUE, log.p=FALSE)
rhyper(nn, m, n, k)

引数：
x, q　白と黒の球を含む壺から非復元で取り出した白い球の数を表すクォンタイルのベクトル
m　壺の中の白い球の数
n　壺の中の黒い球の数

k 壺から取り出した球の数
p 確率．0 と 1 の間の数である必要
nn 必要な乱数の数
log，log.p 論理値．もし TRUE なら確率 p は対数値 log(p) とされる
lower.tail 論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

超幾何分布は非復元抽出で使われる．パラメータ m, n そして k (以下ではそれぞれ Np, $N-Np$, n と表す) の超幾何分布の確率関数は次式で与えられる：

$$p(x) = \frac{\binom{m}{x}\binom{n}{k-x}}{\binom{m+n}{k}}, \quad x = 0, \ldots, k.$$

```
> m <- 10; n <- 7; k <- 8
> x <- 0:(k+1)
> rbind(phyper(x, m, n, k), dhyper(x, m, n, k))
     [,1]        [,2]      [,3]     [,4]     [,5]     [,6]     [,7]
[1,]    0 0.000411353 0.0133689 0.117030 0.419374 0.782188 0.963595
[2,]    0 0.000411353 0.0129576 0.103661 0.302344 0.362813 0.181406
          [,8]      [,9] [,10]
[1,] 0.9981489 1.0000000     1
[2,] 0.0345536 0.0018510     0

# 分布関数は確率関数の累積和にはならない
> all(phyper(x, m, n, k) == cumsum(dhyper(x, m, n, k)))
[1] FALSE
> signif(phyper(x, m, n, k) - cumsum(dhyper(x, m, n, k)), digit=3)   # しかし誤差は僅か
[1] 0.00e+00  1.73e-18  5.38e-17 4.02e-16 1.55e-15 2.78e-15 3.22e-15
[8] 3.33e-15 -2.22e-16 -2.22e-16
```

11.3.7 多項分布

多項分布に従う乱数ベクトルを発生し，確率関数を計算する．

書式：
rmultinom(n, size, prob)
dmultinom(x, size=NULL, prob, log=FALSE)

引数：
x 0:size 中の整数からなる長さ K のベクトル
n 生成する乱数ベクトルの数
size 典型的な多項試行で K 個の箱に入れられるオブジェクトの総数 (N とする)．dmultinom() に対しては既定値は sum(x)
prob 長さ K の数値非負ベクトルで， K 種類のクラスに対する確率を指定し，内部的に総和 1 に正規化される
log 論理値．もし TRUE なら確率の対数値が計算される

もし x が K 成分のベクトルなら dmultinom(x, prob) は確率

$$\mathbf{P}(X_1 = x_1, \ldots, X_K = x_k) = C \times \prod_{j=1}^{K} p_j^{x_j}$$

となる，ここで C は多項係数 $C = N!/(x_1! \times \ldots \times x_K!)$ であり $N = \sum_{j=1}^{K} x_j$．定義から，各 $j = 1, \ldots, K$ で，成分 X_j は二項分布 `Bin(size, prob[j])` に従う．`rmultinom()` は二項乱数 $\mathrm{Bin}(n_j, P_j)$ を順次生成する，ここで $n_1 = N(=$ `size`$)$, $P_1 = p_1$ (p は総和 1 に正規化された `prob`) であり，$j \geq 2$ に対しては再帰的に $n_j = N - \sum_{k=1}^{j-1} n_k$ と $P_j = p_j/(1 - \sum_{k=1}^{j-1} p_k)$ とされる．

`rmultinom()` の返り値は，各列が要求された多項分布に従って発生された (従って総和が `size` となる) ランダムベクトルである $K \times n$ 整数行列となる．一見結果を転置したほうがより自然に見えるが，行列は列順に保管されるので，この方がより効率的である．

注意： `dmultinom()` は現在全くベクトル化されておらず，C へのインタフェイスを持たない．これは将来修正されるかも知れない．
関連： 特殊ケースと考えられる `rbinom()`.

```
> rmultinom(10, size = 12, prob=c(0.1,0.2,0.8))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    2    2    1    1    0    0    1    1    0     0
[2,]    2    0    2    2    2    4    2    1    4     1
[3,]    8   10    9    9   10    8    9   10    8    11

> pr <- c(1,3,6,10)                                  # 乱数発生には確率の正規化は不要
> rmultinom(10, 20, prob = pr)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    0    1    1    1    0    0    0    0     0
[2,]    1    3    5    2    5    2    5    2    3     3
[3,]    9    4    7    9    6    9    5   11    9     7
[4,]    9   13    7    8    8    9   10    7    8    10

# Multinom(N=3, K=3) の全ての可能な出力
> X <- t(as.matrix(expand.grid(0:3, 0:3))); X <- X[, colSums(X) <= 3]
> X <- rbind(X, 3:3 - colSums(X))
> dimnames(X) <- list(letters[1:3], NULL)
> X
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
a    0    1    2    3    0    1    2    0    1     0
b    0    0    0    0    1    1    1    2    2     3
c    3    2    1    0    2    1    0    1    0     0
> round(apply(X, 2, function(x) dmultinom(x, prob = c(1,2,5))), 3)
[1] 0.244 0.146 0.029 0.002 0.293 0.117 0.012 0.117 0.023 0.016
```

11.3.8 Wilcoxon ランク和統計量分布

`dwilcox()` はサイズがそれぞれ `m`, `n` の標本から得られた Wilcoxon のランク和統計量の密度関数, `pwilcox()` は分布関数, `qwilcox()` はクォンタイル関数, そして `rwilcox()` は乱数を与える．

書式：

```
dwilcox(x, m, n, log=FALSE)
pwilcox(q, m, n, lower.tail=TRUE, log.p=FALSE)
qwilcox(p, m, n, lower.tail=TRUE, log.p=FALSE)
rwilcox(nn, m, n)
```

引数：

`x`, `q` クォンタイルのベクトル

p 確率のベクトル
n 必要な乱数の数
m，n それぞれ最初と 2 番目の標本中の観測値の数．正整数のベクトルで良い
log，log.p 論理値．もし TRUE なら確率 p は対数値 log(p) とされる
lower.tail 論理値．もし TRUE (既定値) なら確率は下側確率 P(X <= x)，さもなければ上側確率 P(X > x) とされる

この分布は次のように得られる．x と y をサイズが m と n の二つの独立な標本とする．Wilcoxon のランク和統計量は $y_j \le x_i$ である全ての対 (x_i, y_j) の数である．この統計量は 0 と mn の間の値を取り，その平均と分散はそれぞれ $mn/2$ と $mn(m+n+1)/12$ である．もし最初の 3 つの引数のどれかがベクトルなら，最長のベクトルの長さに対する 3 つの引数の全ての組合せに対して計算を行うために，リサイクリング規則が適用される．

注意： S-PLUS は異なった (しかし同値な) Wilcoxon 統計量の定義を用いる．
関連： データから 統計量を計算したり，p 値等を見出したりするには wilcox.test()．一標本 Wilcoxon 符号付きランク統計量の分布に付いては dsignrank() 等．

```
> x <- -1:(4*6 + 1)
> fx <- dwilcox(x, 4, 6)
> Fx <- pwilcox(x, 4, 6)
> dwilcox(x, 4, 6)
 [1] 0.000000000 0.004761905 0.004761905 0.009523810 0.014285714 0.023809524
(途中省略)
[25] 0.004761905 0.004761905 0.000000000
> pwilcox(x, 4, 6)
 [1] 0.000000000 0.004761905 0.009523810 0.019047619 0.033333333 0.057142857
(途中省略)
[25] 0.995238095 1.000000000 1.000000000
```

11.3.9 符号付きランク統計量分布

サイズ n の標本に対する Wilcoxon 符号付きランク統計量の分布関数に関する情報を得る．dsignrank() は確率関数，psignrank() は分布関数，qsignrank() はクォンタイル関数，そして rsignrank() は乱数を与える．

書式：

```
dsignrank(x, n, log=FALSE)
psignrank(q, n, lower.tail=TRUE, log.p=FALSE)
qsignrank(p, n, lower.tail=TRUE, log.p=FALSE)
rsignrank(nn, n)
```

引数：

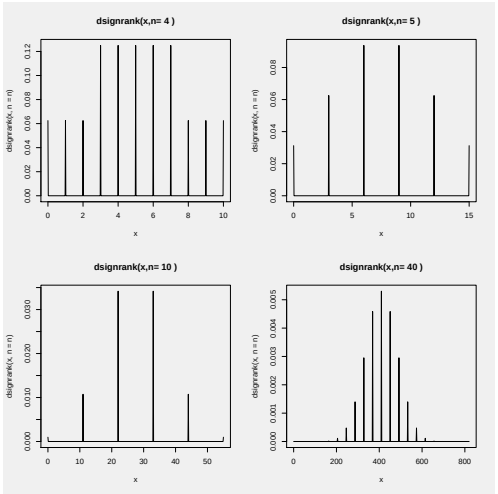
x，q クォンタイルのベクトル
p 確率のベクトル
nn 必要な乱数の数
n 一標本中の観測値の数．正整数か，そうした整数のベクトル
log，log.p 論理値．もし TRUE なら確率 p は対数値 log(p) とされる

`lower.tail` 論理値．もし `TRUE` (既定値) なら確率は下側確率 `P(X <= x)`，さもなければ上側確率 `P(X > x)` とされる

この分布は以下のように得られる．`x` を原点に関して対称な連続分布からのサイズ `n` の標本とする．すると Wilcoxon の符号付きランク統計量は，x_i の絶対値のランクのうち，$x_i > 0$ であるようなものの総和である．この統計量は 0 と $n(n+1)/2$ の間の値を取り，その平均と分散はそれぞれ $n(n+1)/4$ と $n(n+1)(2n+1)/24$ である．もし最初の 2 つの引数の一方がベクトルなら，リサイクル規則を用い，長い方のベクトルの長さに対する全ての 2 種類の組合せに対して計算が行われる．

関連：データから統計量や p 値を計算するには `wilcox.test()`．2 標本 Wilcoxon ランク和統計量は `dwilcox()` 等．

```
# n=4,5,10,40 に対する確率関数のプロットを描く（次の図を参照）
> par(mfrow=c(2,2))
> for(n in c(4:5,10,40)) {
    x <- seq(0, n*(n+1)/2, length=501)
    plot(x, dsignrank(x,n=n), type='l',
    main=paste("dsignrank(x,n=",n,")")) }
```



Wilcoxon の符号付きランク統計量分布の確率関数，$n = 4, 5, 10, 40$.

11.3.10 ランダムな 2 元配置

`r2dtable()` は Patefield のアルゴリズムを用い，周辺和を与えたランダムな 2 元配置を生成する．

書式： `r2dtable(n, r, c)`

引数：

`n` 生成する表の数を与える非負数

`r` 少なくとも長さ 2 の非負ベクトルで，行和を与え，整数に強制変換される．総和は `c` のそれと同じにならねばならない

`c` 少なくとも長さ 2 の列和を与える非負ベクトルで，整数に強制変換される

返り値： 長さ `n` のリストで，生成された表を成分に持つ

```
# Fisher の喫茶データ（検定関数 fisher.test() の例を参照）
> TeaTasting
    <- matrix(c(3,1,1,3), nr=2,
              dimnames=list(Guess=c("Milk","Tea"), Truth=c("Milk","Tea")))
# 最大 Pearson 残差（その和でなく）に基づく独立性の置換検定のシミュレーション
> rowTotals <- rowSums(TeaTasting)
> colTotals <- colSums(TeaTasting)
> nOfCases <- sum(rowTotals)
> expected <- outer(rowTotals, colTotals, "*") / nOfCases
> maxSqResid <- function(x) max((x - expected)^ 2 / expected)
> simMaxSqResid <-
     sapply(r2dtable(1000, rowTotals, colTotals), maxSqResid)
> sum(simMaxSqResid >= maxSqResid(TeaTasting)) / 1000
[1] 0.453
# Fisher の正確検定は p=0.4857... を与える
```

11.3.11 一致確率

一般化された「誕生日のパラドックス」問題への近似解を与える．pbirthday() は一致確率，qbirthday() は指定された一致確率に必要な観測数を計算する．

書式：

```
qbirthday(prob = 0.5, classes = 365, coincident = 2)
pbirthday(n, classes = 365, coincident = 2)
```

引数：

classes　人々をいくつのカテゴリに分類するか

prob　必要な一致確率

n　人数

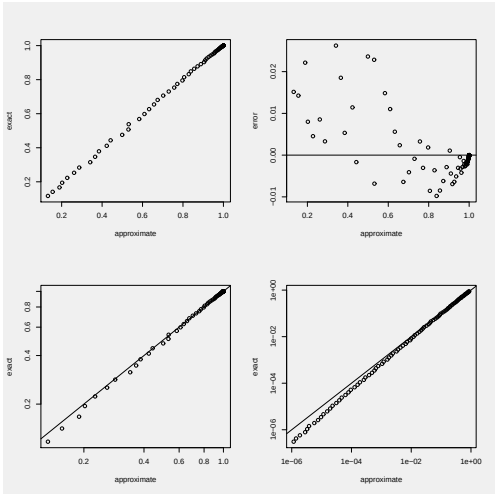
coincident　同じカテゴリに入る人数

返り値：　qbirthday() は k 人が classes 個の同様に確からしいラベルの内の同じものを持つ確率が prob となるのに必要な人数を与える．pbirthday() は指定された一致が起きる確率を与える

誕生日のパラドックスとは，誕生日が一致するペアがいる確率が 50% を越えるには 23 人という非常に少ない人数で十分というものである．この関数は 50% 以外の確率の計算，二人以上の一致ケース，そして 365 以外のクラスにも一般化されている．この公式は以下の例が示すように近似である．coincident=2 では正確な計算が容易であり，おそらく好ましい．

```
# 標準版（誕生日が一致するペアがいる確率が 50% を越えるために必要な人数）
> qbirthday()
[1] 22                                          # 実際は 23 人必要

# 無作為に選んだ 4 桁の PIN 番号中の一致ペア確率が 50% を越えるために必要な数
> qbirthday(classes=10^4)
[1] 118

> qbirthday(coincident=3,prob=0.9)  # 3 人の誕生日が一致する確率が 90% を越えるために必要な人数
[1] 123

> pbirthday(150,coincident=4)                   # 150 人中に誕生日の同じ 4 人がいる確率
[1] 0.351437
```

```
> x1<- sapply(10:100, pbirthday)                    # 正確な計算との比較
> x2<-1-sapply(10:100, function(n) prod((365:(365-n+1))/rep(365,n)))
> par(mfrow=c(2,2))
> plot(x1,x2,xlab="approximate",ylab="exact")
> plot(x1,x1-x2,xlab="approximate",ylab="error")
> abline(h=0)
> plot(x1,x2,log="xy",xlab="approximate",ylab="exact")
> abline(0,1)
> plot(1-x1,1-x2,log="xy",xlab="approximate",ylab="exact")
> abline(0,1)
```



誕生日確率．近似値と正確な値との比較
(上左) 近似値と正確な値．
(上右) 誤差．
(下左) 対数軸表示．
(下右) (対数値の) 非一致確率比較．

11.4 その他

11.4.1 混合分布

密度 (確率) 関数 $f_1, f_2, \ldots, f_k$ と重み $p_1, p_2, \ldots, p_k > 0$ $(p_1 + p_2 + \cdots + p_k = 1)$ から

$$f(x) = p_1 f_1(x) + p_2 f_2(x) + \cdots + p_k f_k(x)$$

で定義される密度関数を，$\{f_i\}$ の離散混合分布と呼ぶ．離散混合分布に従う乱数を発生するには $f_i(x)$ に従う乱数 r_i と，確率 $p_1, p_2, \ldots, p_k$ でそれぞれ値 $1, 2, \ldots, k$ に従う乱数 I を用い $r = r_I$ とすれば良い．

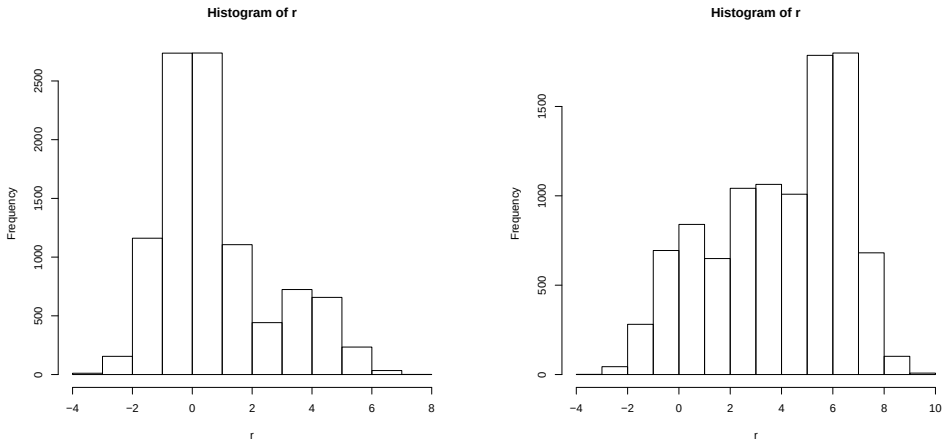
```
# N(0,1),N(4,1) を確率 0.2,0.8 で混合した分布に従う乱数を発生する
> n <- 1e4                                          # 必要乱数数
> I <- sample(0:1, n, prob=c(0.2,0.8), rep=T)       # 確率 0.2,0.8 で値 0,1 を取る乱数
> r <- I*rnorm(n,0,1)+(1-I)*rnorm(n,4,1)            # 論理判断を掛け算で実現
> hist(r)                                           # 以下の図を参照

# N(0,1),N(3,1),N(6,1) を確率 0.2,0.3,0.5 で混合した分布に従う乱数を発生する
> I <- sample(1:3, 1e4, prob=c(0.2,0.3,0.5), replace=TRUE)
> I1 <- I==1;I2 <- I==2;I3 <- I==3 # 1,2,3 をそれぞれの確率で選択する論理ベクトル
> r <- I1*rnorm(1e4,0,1) + I2*rnorm(1e4,3,1) + I3*rnorm(1e4,6,1)
> hist(r)                                           # 以下の図を参照
```

パラメータ θ を持つ密度関数族 $f(x \mid \theta)$ を密度関数 $g(\theta)$ で (連続) 混合した分布の密度関数は

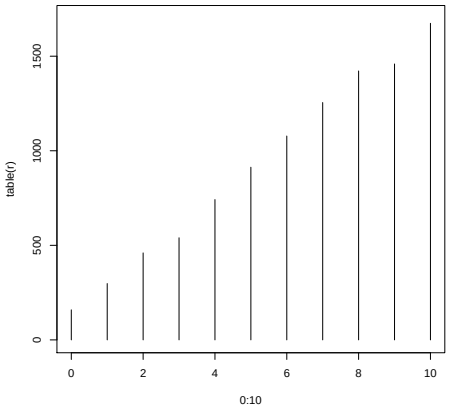
$$f(x) = \int f(x \mid \theta) g(\theta) d\theta$$

となる．f に従う乱数を発生するには g に従う乱数 Θ に対し，密度関数 $f(x \mid \Theta)$ に従う乱数を発生すれば良い．しかしながら，この方法はベクトル化できないため時間がかかる．

```
# 二項分布の確率パラメータをβ分布で混合した分布に従う乱数を発生
> P <- rbeta(1e4, shape1=2, shape2=1)   # β乱数 1e4 個

# 発生確率を変えながらサイズ 10 の二項分布乱数を発生
> r <- sapply(seq(1e4), function(i) rbinom(1,10,P[i]))
> table(r)
r
   0    1    2    3    4    5    6    7    8    9   10
 166  305  470  621  800  896 1047 1234 1311 1508 1642
> plot(0:10, table(r), ylim=c(0,max(table(r))))   # 以下の図を参照
```



(左) 二つの正規分布の混合分布．(右) 三つの正規分布の混合分布



連続混合分布

11.4.2 棄却法

より一般の確率分布用の乱数を発生する方法は，分布固有の方法[*3] を別にしても色々あるが，代表的なものに棄却法 (rejection sampling)[*4] がある．確率密度関数 $f(x)$ に対し，乱数の発生が容易な確率密度関数 $g(x)$ があるとし，$f(x) \leq Mg(x)$ となる定数 M があるとする．$g(x)$ に従う乱数 X と単位区間上の一様乱数 U に対し，$U \leq f(X)/Mg(X)$ となれば X は $f(x)$ に従う乱数となる．実際 $h(x)$ を任意関数とすれば

$$\begin{aligned}
\boldsymbol{E}\{h(X) \mid U \leq f(X)/Mg(X)\} &= \frac{\boldsymbol{E}\{I[U \leq f(X)/Mg(X)]h(X)\}}{\boldsymbol{E}\{I[U \leq< f(X)/Mg(X)]\}} \\
&= \frac{\boldsymbol{E}\{f(X)h(X)/Mg(X)\}}{\boldsymbol{E}\{f(X)/Mg(X)\}} \\
&= \frac{\int f(x)h(x)/Mg(x) \cdot g(x)dx}{\int f(x)/Mg(x) \cdot g(x)dx} \\
&= \int h(x)f(x)dx.
\end{aligned}$$

$f(x)$,$g(x)$ は多次元密度関数でも良い．以下の例では $f(x,y) = I[x^2 + y^2 \leq 1]/\pi$, $g(x,y) = I[|x|, |y| \leq 1]/4$ はそれぞれ 2 次元単位円，2 次元矩形 $[-1,1] \times [-1,1]$ 上の一様分布の密度関数で，$M = 1$ である．

```
# 単位円上に一様分布する二次元乱数の生成
> X <- runif(1e4,-1,1)   # まず 2 次元矩形 [-1,1]x[-1,1] 上に一様分布する乱数の x,y 座標を生成
> Y <- runif(1e4,-1,1)
> I <- (X^2+Y^2 <= 1)                          # (X,Y) が単位円に入るかどうかを表す論理ベクトル
> sum(I)                                       # 7859 個が単位円に入った
[1] 7859
> X <- X[I]; Y <- Y[I]                         # 単位円内に入るものだけ残す
```

11.4.3 打ち切り分布

密度 (確率) 関数 $f(x)$ に対し，その値の範囲を集合 G に制限した確率分布を打ち切り分布 (truncated distribution) と呼ぶ．連続分布であれば，その密度関数 $f^*(x)$ は

$$f^*(x) = \begin{cases} \frac{f(x)}{\int_G f(y)dy} & x \in G \\ 0 & x \notin G \end{cases}$$

となる．応用上しばしば登場する打ち切り分布に，離散値 $0, 1, 2, \ldots$ を取る確率変数 X に対し，$X > 0$ という条件を付けた零打ち切り分布 (zero-truncated distribution) がある．その確率関数は

$$\boldsymbol{E}\{X = k \mid X > 0\} = \frac{\boldsymbol{p}(X = k)}{1 - \boldsymbol{P}(X = 0)} \qquad k = 1, 2, \ldots$$

となる．打ち切り分布に従う乱数を発生するには棄却法で $g(x) = f(x)$ と置けば良い．具体的には $f(x)$ に従う乱数 X の内 $X \in G$ であるものだけを採用する．

[*3] 例えば "Non-Uniform Random Variate Generation", Luc Devroye, Springer-Verlag(1986) が詳しい．

[*4] より洗練され効率的な方法として，単調減少もしくは対称単峰な密度関数に従う分布に従う乱数を発生する ziggurat アルゴリズムがある．

```
# 標準正規分布 N(0,1) を値 0.5 以上に打ち切った乱数を発生
> x <- rnorm(1e4)                                  # 1 万個の正規乱数を発生
> xx <- x[x >= 0.5]
> length(xx)                                       # 3026 個の乱数を得た
[1] 3026
> 1e4*(1-pnorm(0.5))                               # 理論的には約 3085 個の乱数を得るはず
[1] 3085.375

# 平均 2 のポアッソン分布の零打ち切り分布に従う乱数を発生
> x <- rpois(1e4, lambda=2)                        # 1 万個のポアッソン乱数を発生
> xx <- x[x > 0]
> length(xx)                                       # 8652 個の乱数を得た
[1] 8652
> 1e4*(1-ppois(0, lambda=2))                       # 理論的には約 8647 個の乱数を得るはず
[1] 8646.647

# 一万個の打ちきり乱数が確実に欲しければ
> ( n <- 1e4/(1-ppois(0, lambda=2)) )              # 理論的にはこれだけの元乱数が必要
[1] 11565.18
> x <- rpois(1.5*n, lambda=2)                      # 元乱数を余裕を見て多めに発生
> xx <- (x[x > 0])[1:1e4]                          # 打ちきり乱数の最初の一万個
```

11.4.4 準乱数

準乱数 (semi-random number)[*5] とは疑似乱数よりは規則的であるが，数値積分に使用するとはるかに効率的な結果[*6] を与える特殊な数列である．準乱数の発生アルゴリズムは幾つも提案されており，R のパッケージには準乱数を発生する関数を提供するもの[*7] がある．以下ではパッケージ `randtoolbox` 中の関数 `torus()` を紹介する．`torus()` 関数は Torus アルゴリズムを用い，d 次元準乱数の k 番目の値は

$$u_k = (\mathtt{frac}(k\sqrt{p_1}), \ldots, \mathtt{frac}(k\sqrt{p_d}))$$

で与えられる．ここで $p_1, p_2, \ldots, p_d$ は d 個の相異なる素数で，既定では $2, 3, 5, \ldots$ である．`usetime=FALSE` ならば系列は $k = 1$ から開始され，さもなければ内蔵時計から決まるある k から開始される．

書式：`torus(n, dim = 1, prime, mixed = FALSE, usetime = FALSE)`

引数：

`n` 発生する (ベクトル) 準乱数の数

`dim` 準乱数の次元 (10,000 以下)

`prime` オプションの素数もしくは素数のベクトル

`mixed` mixed Torus algorithm を使うか？

```
> torus(3, dim = 5)                                # 5 次元準乱数
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.41421356 0.7320508 0.2360680 0.6457513 0.3166248
[2,] 0.82842712 0.4641016 0.4721360 0.2915026 0.6332496
[3,] 0.24264069 0.1961524 0.7082039 0.9372539 0.9498744
```

[*5] 超一様乱数もしくは低食い違い数列 (low-discrepancy sequence) と呼ばれることもある．

[*6] 疑似乱数による数値積分の精度が，乱数の数 n に対し一般に $O(\sqrt{\log\log n/n})$ であるのに対し，準乱数によるそれは次元を d とすると，一般に $O((\log n)^d/n)$ となる．但しある程度高次元になると有効性は失われる．

[*7] パッケージ `fOptions` は Halton, Sobol 準乱数用の関数を持つ．またパッケージ `gsl` (GSL, GNU Scientific Library, へのインタフェイス関数群パッケージ) も順乱数用の関数を持つ．

```
> torus(3, dim = 5, c(3,5,7,11,13))              # 上と同じこと
          [,1]      [,2]      [,3]      [,4]       [,5]
 [1,] 0.7320508 0.2360680 0.6457513 0.3166248 0.60555128
 [2,] 0.4641016 0.4721360 0.2915026 0.6332496 0.21110255
 [3,] 0.1961524 0.7082039 0.9372539 0.9498744 0.81665383
```

11.4.5 一般パッケージ中の分布関連関数

CRAN の貢献パッケージには R 本体には無い分布関数，および R 本体のそれを代替・拡張する関数がある．参考までに幾つか紹介 (2008.7 現在) する．

`bs` Birnbaum-Saunders 分布

`BiasedUrn` 偏りのある壺モデル分布

`benchden` カーネル密度関数推定のベンチマーク用に 28 種類の密度関数関連関数とその特性のまとめがある

`compoisson` Conway-Maxwell-Poisson 分布

`degreenet` skewed count 分布

`evd` (2 次元) 極値分布

`exactRankTests` 正確な rank, permutation テストの分布

`gbs` 一般化 Birnbaum-Saunders 分布

`ghyp` 一般化超幾何分布とその特殊例

`gld` 一般化 (Tukey) λ 分布

`gsl` gsl(Gnu scientific library) へのラッパ関数群で，準乱数発生関数がある

]`HyperbolicDist`] hyperbolic 分布

`ig` 逆正規分布

`mnormt` 多変量正規分布と t 分布

`modeest` Chernoff 分布

`mvtnorm` 多変量正規分布と t 分布

`mvtnormpcs` 多変量正規分布と t 分布

`normalp` exponential power 分布

`POT` 一般化 Pareto 分布

`poilog` Poisson lognormal, 二変量 Poisson lognormal 分布

`Runuran` 広範囲の離散・連続分布に従う乱数発生関数ライブラリである UNU.RAN(library for Universal Non-RANdom variate generators への R インタフェイス

`SuppDists` 一般化超幾何分布，逆正規分布，Johnson システム分布，ノンパラメトリック統計量分布関連の関数

`skewt` skewed Student t 分布

`tdist` 独立な Student t 変数の線形和の分布

`triangle` triangle 分布

`zipfR` Zipf(単語頻度) 分布

第 12 章
組み込みデータセット

R の魅力の一つが大量の組み込みデータセットを持つことである．これらは，`example()` 関数による参考実行コードで頻繁に使われ，説明のための説明に終らない迫真性を与えている．またこれらのデータには統計的手法のベンチマークデータとして多くの文献[*1] で使われている古典的なものがある．また千以上に及ぶ R の貢献パッケージの多くも例示用の組み込みデータセット[*2] を備えている．この章では R 本体に含まれる 86 組の組み込みデータセット[*3] を紹介する．これらのデータセットにはより詳しい解説とともに，例示用コードが付随[*4] しており `example(iris)` 等で実行結果を見ることができる．

以下のデータセットの分類はあくまで便宜上のものであり，二つ以上の分類に跨るものもある．データセットの解説に含まれる科学・技術用語はかなり特殊なものも多く，訳語が適切かどうかは保証の限りでない．詳しくは英語ヘルプドキュメントを参照されたい．

12.1 組み込みデータセット 時系列データ

- `AirPassengers`： 古典的な Box-Jenkins のデータ．1949–1960 年の月別国際航空路線総顧客数 (単位千人)．144 観測値.
- `airmiles`： 米国の商業航空路線の有償旅客マイル数 (ある期間中の運賃支払乗客数をその乗客の飛行距離でかけたもの)．1937–1960 年の年別時系列．24 観測値.
- `austres`： 1971 年 3 月から 1994 年 5 月に至るオーストラリアの四半期毎の居住者数 (単位千人)．89 観測値.
- `BJsales`： Box-Jenkins の売上高データ `BJsales` と，景気の先行指標データ `BJsales.lead` を含む．各々 150 観測値.
- `co2`： ハワイのマウナ・ロア山で観測された大気中の CO2 濃度 (単位 *ppm*)．1997 年度の ISO モル分率圧力計スケールによる．1959–1997 年度の月別データ．1964 年

[*1] 一般的に単なる統計データには著作権の保護は及ばず，論文や書物に一旦公表されたデータは自由に再利用できる．その際データのソースを明記することが好ましいことはいうまでもない．但しそれから「創造的に」派生した二次的解析結果等 (例えば解析結果の図表，データベースとして整備されたもの等) は著作権の対象となり得るので注意が必要である．ネットで公開されているデータについては微妙な問題が起き得るようであり，利用に関する注意を考慮した方が良い．R 及びその貢献パッケージ収載のデータセットに付いても特に著作権が明記されている例は知らない.

[*2] 貢献パッケージにはコンパニオンパッケージと呼ばれる，ある本に記載された R コードとデータセットを一まとめに提供するものが幾つもある.

[*3] 現在は基本パッケージ `datasets` にまとめて入っており，命令 `libarary(help=datasets)` で簡易説明とともに一覧できる.

[*4] Linux システムでは端末から `grep -r "iris" /usr/lib/R/* | grep "/help"` 等の命令を実行すれば，あるデータセットを参考用に参照している R 関数の一覧を得ることができる．また中間栄治氏のサイト R code search by gonzui (URL `http://rgonzui.nakama.ne.jp/`) でデータセットをキーワードを含む R コードを検索できる.

の 2,3,4 月分は欠損しており，1964 年の 1,5 月のデータから線形補間で補われている．468 観測値.

- `EuStockMarkets`： 多次元時系列データ．ヨーロッパの主要株式指標 (Germany DAX (Ibis), Switzerland SMI, France CAC, そして UK FTSE) の毎営業日 (週末と祭日を除く) の引け値．4 変量．1,860 観測値.
- `JohnsonJohnson`： 1960–1980 年の Johnson-Johnson 社の一株当たり四半期毎利益 (単位ドル)．84 観測値.
- `LakeHuron`： 1875–1972 年のヒューロン湖の年別水位 (単位 $feet$)．98 観測値.
- `lh`： 一人の女性の 10 分間隔の血液中の黄体形成ホルモン (luteinizing hormone) 量の時系列．48 観測値.
- `lynx`： 1821–1934 年にカナダで罠で捕獲された大山猫 (lynx) の年別数．114 観測値.
- `Nile`： 1871–1970 年のアスワンに於けるナイル川の年流量測定値．100 観測値.
- `nhtemp`： コネチカット州 New Haven の年平均気温 (華氏)．1912–1971 年の年別データ．60 観測値.
- `nottem`： 英国ノッティンガム城で 1920–1939 年に測定された月別平均気温 (華氏)．240 観測値.
- `presidents`： 米国の (ほぼ) 四半期毎の大統領支持率 (ギャラップ社調査)．1945 年第 1 四半期から 1974 年第 4 四半期．実際はかなりでっち上げのデータらしい．120 観測値.
- `sunspots`： 月別の平均相対太陽黒点数．1960 年まではチューリッヒのスイス連邦観測所，それ以降は東京天文台による．1749–1983 年の月別データ．データセット `sunspot.month` はより長期で少し異なる黒点数データである．2,820 観測値.
- `sunspot.month`： 月別の平均相対太陽黒点数．2,988 観測値.
- `sunspot.year`： 年別の平均相対太陽黒点数．289 観測値.
- `treering`： (次元無しの) 正規化されたカリフォルニア州の Bristlecone pine の年輪の幅．Bristlecone pine は樹齢 4 千年を越える個体が存在する，最長級の寿命を持つ植物．7,981 観測値.
- `UKDriverDeaths`： データセット `Seatbelts` 参照.
- `Seatbelts`： 多変量時系列．1969 年 1 月から 1984 年 12 月までの英国の，死亡もしくは重傷を負った車の運転者の月別数．シートベルトの着用が 1983 年 1 月 31 日に義務付けられた．変数 `DriversKilled` は運転者死亡数，変数 `drivers` はデータセット `UKDriverDeaths` と同一，変数 `front` は前部席搭乗者による死亡もしくは重傷者数，変数 `rear` は後部席搭乗者による死亡もしくは重傷者数，変数 `kms` は走行距離，変数 `PetrolPrice` は石油価格，変数 `VanKilled` は軽量車運転者数，変数 `law` は法律が施行された後かどうかを示す 0,1 データ．192 観測値.
- `UKLungDeaths`： 呼吸器系疾患 (肺癌，肺気腫，喘息) による 1974–1979 年の英国の月別死亡者数．両性 `ldeaths`，男性 `mdeaths`，女性 `fdeaths` の 3 時系列からなる．72 観測値.
- `UKgas`： 1960-1968 年の英国の四半期毎のガス消費量 (単位 ガス熱量単位 $therm$)．108 観測値.
- `USAccDeaths`： 1973–1978 年の米国の月別事故死亡者数．72 観測値.

- `uspop`： 10 年毎の国勢調査による 1790–1970 年の米国人口 (単位百万人)．19 観測値．
- `WWWusage`： あるサーバー経由でインターネットに接続したユーザの分毎の数．100 観測値．
- `discoveries`： 1860 年から 1959 年の各年における “偉大な” 発明と科学的発見の数．100 観測値．
- `freeny.y`： 1962 年第 2 四半期から第 4 四半期までの国家歳入額．関連データセット `freeny`, `freeny.x`．39 観測値．

12.2　組み込みデータセット データフレーム

データフレームとして表現されたデータセットは公式によるシンボリックな操作が可能で，R のもっとも基本的なデータ形式である．多変量データとみなせる場合が多いが，グルーピングのための因子変数[*5] を含むものも多い．

- `BOD`： 水質検査における生化学的酸素要求度のデータ．数値変数 `Time` (日付) と数値ベクトル `demand` (単位 mg/l) を持つ 6 ケースのデータフレーム．2 変数 6 ケース．
- `ChickWeight`： 鶏の体重を誕生から 20 日目まで二日置きに観測．21 日目にも観測．蛋白質成分が異なる 4 つのグループからなる．変数 `weight` は鶏体重の数値データ (単位 g)，変数 `Time` は誕生からの日数，変数 `Chick` は個体を識別する順序付き因子，変数 `Diet` は給餌種類を表す因子 $1, 2, 3, 4$．4 変数 578 ケース．
- `CO2`： 草 (Echinochloa crus-galli) の耐寒性実験で得られたデータ．Quebec と Mississippi 産のそれぞれ 6 個体．半数を一晩低温下においた．変数 `Plant` は各個体を識別する順序付き因子，`Type` は原産地を表す因子，`Treatement` は処理を表す因子，`conc` は環境二酸化炭素濃度 (ml/L) の数値ベクトル，`uptake` は二酸化炭素摂取量 $(umol/m^2)$ の数値ベクトル．5 変数 84 ケース．
- `DNase`： 鼠血清中の組換えタンパク質 DNA 分解酵素 (DNase, deoxyribonuclease) の酵素免疫吸着測定法 (ELISA, enzyme-linked immunosorben assay) 開発中に得られたデータ．変数 `Run` は測定実行順を表す順序付き因子，`conc` は既知のタンパク質濃度の数値ベクトル，`density` は検査における光学的濃度 (次元無し) の測定値の数値ベクトル．3 変数 176 ケース．
- `Formaldehyde`： ホルムアルデヒドの定量用の標準曲線の作成のために行われた化学実験のデータ．クロマトグラフィー酸と濃縮硫酸を炭水化物試量に加えて生じる紫色の濃さを分光光度計で読み取る．変数 `carb` は炭水化物量の数値ベクトル，`optden` は光学的濃度の数値ベクトル．2 変数 6 ケース．
- `Indometh`： インドメタシン (indomethicin，非ステロイド性抗炎症薬) の薬物動態学的データ．6 人の被験者はインドメタシンの静脈注射を受けた．変数 `Subject` は試料コードの順序付き因子，`time` は血液試料が採取された時間 (単位 時間)，`conc` はインドメタシンの血漿濃度 (単位 mcg/ml)．3 変数 66 ケース．
- `InsectSprays`： 異なる殺虫剤スプレーの効果を比較するため実施された複数農業実験区でのおそらく死んだ昆虫の計数データ．変数 `count` は昆虫の数，`spray` はスプレーの種類で 6 水準の因子．2 変数 72 ケース．

[*5] データフレーム中の文字列変数は原則因子 (整数コード) として処理される．

- LifeCycleSavings： 変数 sr は総個人貯蓄額，pop15 は 15 歳以下人口比，pop75 は 75 歳以上人口比，dpi は一人あたり実可処分所得，ddpi は dpi の増加率．地域毎の貯蓄率 (可処分所得で割った個人総貯蓄) に関する (life-cycle savings) 仮説を実証するためのデータ．5 変数 50 ケース．
- Loblolly：テーダ松 (loblolly pine) の成長データ．変数 height は樹高 (単位 feet)，age は樹齢 (単位 年)，Seed は 14 水準の順序付き因子で，種子グループ．3 変数 84 ケース．
- Orange：オレンジの木の成長データ．変数 Tree は測定された 5 本の木を識別する 5 水準の順序付き因子, 変数 age は樹齢 (1968.12.31 以来の日数), 変数 circumference は (おそらく胸高) 周囲長 (単位 mm)．3 変数 35 ケース．
- OrchardSprays： 果樹園への散布農薬の蜜蜂に対する忌避性を調査するためにラテン方画法による実験計画を用いて行われた実験データ．変数 rowpos は実験計画の行，変数 colpos は実験計画の列，変数 treatment は 8 水準の処理因子，変数 decrease は応答値．4 変数 64 ケース．
- PlantGrowth： 対照群と二つの異なる処理条件下で得られた収穫 (乾燥重量) を比較する実験で得られたデータ．変数 weight は重量，group は 3 水準の因子．2 変数 30 ケース．
- Puromycin： 未処理細胞と抗生物質ピューロマイシン (puromycin) で処理された細胞に対する酵素反応での反応速度と基質濃度．変数 conc は基質濃度 (単位 ppm)，rate は瞬間反応率 (単位 $counts/min/min$)，state は 2 水準 treated, untreated の因子．3 変数 23 ケース．
- Theoph： 喘息治療薬テオフィリン (theophylline，茶葉に含まれる苦み成分アルカロイド) の薬物動態学的実験で得られたデータ．変数 Subject は各被実験者を表す 12 水準の順序付き因子，Wt は被実験者の体重 (単位 Kg)，Dose はテオフィリンの経口投与量 (単位 mg/Kg)，Time は経口投与時から試量が得られるまでの時間 (単位 時)，conc は試量中のテオフィリン濃度 (単位 mg/L)．5 変数 132 ケース．
- ToothGrowth： ビタミン C をオレンジジュースとアスコルビン酸の形で投与 (0.5,1,2mg) した各々 10 匹のギニアピッグ (モルモット) の歯の象牙芽細胞の長さのデータ．変数 len は歯の長さ，supp は投与法を示す 2 水準 "OJ","VC" の因子，dose は投与量 (単位 mg)．3 変数 60 ケース．
- USArrests： 1973 年の米国 50 州の人口 10 万人当たりの暴行，殺人，強姦による逮捕者数と都市部住民比率のデータ．変数 Murder は人口 10 万人当たりの殺人による逮捕者数，変数 Assault は人口 10 万人当たりの暴行による逮捕者数，変数 UrbanPo は都市部住民比率，変数 Rape は人口 10 万人当たりの強姦による逮捕者数．米国 50 州情報のデータセット state も参照．4 変数 50 ケース
- USJudgeRatings： 弁護士による米国最高裁判事の評価データ (恐らく 10 点満点での主観評価)．変数 CONT は弁護士と判事との接触回数，INTG は法曹家としての高潔さ，DMNR は態度，DILG は勤勉さ，CFMG は裁判処理能力，DECI は決定の迅速さ，PREP は審理への準備，FAMI は法知識，ORAL は適切な判決口述，WRIT は適切な判決文，PHYS は身体能力，RTEN は記憶に値するかどうか．12 変数 43 ケース．
- airquality： ニューヨークにおける 1973 年 5 月から 9 月までの毎日の大気状態観測データ．変数 Ozone は Roosevelt 島における 13:00 時から 15:00 時までの平均オ

ゾン量 (単位 *ppb*)，Solar.R はセントラルパークにおける 08:00 時から 12:00 時の周波数 4000–7700 オングストロームの日射量 (単位 Langleys, cal/cm^2)．Wind は La Guardia 空港における毎日の 07:00 時から 10:00 時の平均風速 (単位 マイル / 時)，Temp は華氏温度，Month は月，Day は日にち．6 変数 154 ケース．

- anscombe： 線形単回帰に対する Anscombe の四つ組．同じ統計的性質 (平均，分散，相関，回帰直線) を持つが，全く異なる 4 組の x,y 値のデータセット．変数 x1,x2,x3 は同一で特別に並べられた数列 4:14，x4 は c(8,19)，y1 から y4 は平均 7.5，分散 2.03 の区間 (3,12.5) 中の値からなるベクトル．8 変数 11 ケース．
- attenu： Joyner-Boore の地震波の減衰データ．このデータはカリフォルニア州の 23 の地震のピーク時加速度を，様々な観測基地で測定したデータを与える．このデータは多くの研究者により，基本加速度に対する距離による減衰効果を推定するために用いられてきた．変数 event は事象番号，mag はモーメント・マグニチュード，station は観測基地番号の因子，dist は基地と震央との距離 (単位 km)，accel はピーク時加速度 (単位 g)．5 変数 182 ケース．
- attitude： Chatterjee-Price の管理者態度データ．大規模ファイナンス企業の無作為に選ばれた 30 の部署の各々から選ばれたほぼ 35 人の事務従業員への調査から得られた．数値は各部署における 7 つの上司に関する質問への好意的な解答の割合を与える．変数 rating は全般的評価，complaints は雇用者の苦情の処理，privileges は特別な特権の付与，learning は学習機会の付与，raises は業績による昇進機会，critical は批判的態度，advancel は進歩．7 変数 30 ケース．
- beavers： 2 組のデータフレームからなる．ウィスコンシン州北部中地域の 4 匹の雌のビーバ (Castor canadensis) の体温を 10 分毎にテレメータで測定したデータ．データフレーム beaver1 は 4 変数 114 ケース (12 月 12,13 日，22:20 分のデータは欠損)，beaver2 は 4 変数 100 ケース (10 月 3,4 日)．変数 day は観測日．1990 年始めからの日数で beaver1 は 12 月 12,13 日， beaver2 は 11 月 3,4 日．time は観測時間で 3:30am を 0330 等と表記，temp は体温 (摂氏)， activ は避難所外での活動指数．
- cars： 車が停車するまでに必要な距離のデータ．データは 1920 年代に得られたことを注意せよ．変数 speed は速度，dist は停車距離．2 変数 50 ケース．
- chickwts： 補助食品タイプによる鶏の成長速度のデータ．生まれたばかりの鶏を 6 群に分け，それぞれ異なった補助食品入りの飼量を与え，6 週間後体重 (単位 g) を測定した．変数 weight は鶏の体重，feed は補助食品タイプを与える因子．2 変数 71 ケース．
- esoph： フランスの Ile-et-Vilaine における食道ガン (esophageal cancer) の類別研究のデータ．変数 agegp は年齢グループ，alcgp はアルコール摂取量 (単位 g/day)，変数 tobgp は 喫煙量 (単位 g/day)，ncases は症例数，変数 ncontrols は対照群数．5 変数 88 ケース．
- faithful： ワイオミング州イエローストーン国立公園にある間欠泉 Old Faithful Geyser の噴出間隔と噴出継続時間のデータ．変数 eruption は噴出継続時間 (単位 分)，waiting は次の噴出までの時間 (単位 分)．2 変数 272 ケース．
- freeny： 1962 年第 2 四半期から第 4 四半期までの国家歳入額．データセット freeny.x, freeny.y を一まとめにしたもの．5 変数 39 ケース．

- infert： 自然流産と人工流産後の不妊症を研究するために行われた対応対照群研究データ．それまでにそれぞれ 2 回の自然流産と人工流産を経験している被験者一名は除外されている．変数 Education は学歴 (3 水準因子)，age は年齢，parity は過去の出産回数，age は過去の人工流産回数 (3 段階)，case は case(1), control(0)，spontaneous は過去の自然流産回数 (3 段階)，stratum は matched set number(1–83)，pooled.stratum は stratum number(1–63 段階)．8 変数 248 ケース．
- iris： この有名な (Fiher もしくは Anderson の) あやめのデータセットは，3 種類のあやめの品種のそれぞれからの 50 の花の，センチメートル単位の蕚 (がく) 片の長さと幅，花弁の長さと幅の計測結果を与える．品種は Iris setosa, versicolor そして virginica である．変数 Sepal.Length は蕚 (がく) 片の長さ，Sepal.Width は蕚 (がく) 片の長さ，Petal.Length は花弁の長さ，Petal.Width 花弁の幅，Species は品種 (3 水準因子)．データセット iris3 は同じデータを $50 \times 4 \times 3$ の 3 次元配列で表現したもの．5 変数 150 ケース．
- longley： Longley の 7 組のマクロ経済データ (1947–1962)．高度な共線性を示すことで有名．GNP.deflator は GNP デフレーター (1945 年を 100 とする，個々の財・サービスの価格上昇率で加重平均して算出する総合的物価上昇率)，GNP は GNP(国民総生産)，Unemployed は失業者数，Armed.Forces は軍人数，Population は特定の組織に属さない (14 歳未満) 人口，Year は年代，employed は雇用者数．7 変数 16 ケース．
- morley： Michaelson-Morley による古典的な光速測定実験データ．各々引き続く 20 回の測定からなる 5 組の実験からなる．変数 Expt は実験番号，Run は各実験での測定番号，Speed は適当にコード化された光速．3 変数 100 ケース．
- mtcars： 1973–74 年タイプの 32 台の車の燃料消費量とデザインや性能に関する 10 の変数からなる．変数 mpg は燃料消費量 (単位 $Miles/(US)gallon$)，cyl はシリンダー数，disp はエンジン排気量 (単位 cubic inch)，hp は総馬力，drat は後部車軸比，wt は重量 (単位 $lb/1000$)，qsec は $1/4mile$ 走行時間，vs は燃料圧縮比 V/S，am はトランスミッションタイプ (オートマティック 0，マニュアル 1)，gear は前進ギヤ数，carb はキャブレター数．11 変数 32 ケース．
- pressure： 温度と水銀の蒸気圧の関係のデータ．変数 temperature は摂氏温度，pressure は水銀の蒸気圧 (単位 mm)．2 変数 19 ケース．
- quakes： 1964 年以降のフィジー諸島近くの立方体状区域での 4MB 以上の震度の地震の震央位置と震度．変数 lat は震央位置の緯度，long は震央位置の経度，depth は深度，mag は震度 (単位 グーテンベルグ・リヒターの実体波マグニチュード MB)，stations は観測所数．5 変数 1000 ケース．
- randu： 計算機 VAX の Fortran の乗算合同法による組み込み一様疑似乱数発生関数で得られた，引き続く 5 個の乱数の最初の 3 個からなる 400 組のデータ．この発生法は初期の計算機言語で広く使われていた代表的な疑似乱数発生法であったが，引き続く 3 組の乱数を単位立方体中の点と考えると，15 の平面に全て載ってしまうことが後に発見され大問題となった．変数 x, y, z はそれぞれ 3 次元座標値．3 変数 400 ケース．
- rock： 石油貯蓄層から採取された岩石に関するデータ．試料断面の 256×256 ピクセル画像の計測から得られた．変数 area はピクセル数で計った空隙面積，peri

はピクセル数で周長，shape は周長と面積の平方根の比，perm 固有浸透係数 (単位 $mili - Darcy$)．4 変数 48 ケース．

- sleep： Student(Gosset) の睡眠薬の効果のデータ．2 種類の睡眠薬を 10 人の被験者に与え，睡眠時間の増加を対象群と比較した．変数 extra は睡眠時間増加量 (単位 時間)，group は薬の種類を表す 2 水準の因子．2 変数 71 ケース．
- stackloss： Brownlee の Stack Loss データ．アンモニアを酸化して硝酸を製造する化学プラントの 22 日分の吸収塔損失データ．生産された硝酸は向流吸収塔で吸収される．変数 Air Flow は冷却空気の流量で工場の稼働率を表す，Water.Temp は吸収塔中のコイルを環流する冷却水の注入口温度，Acid.Conc．は環流する硝酸の濃度 (ppm) から 500 を引いた値，stack.loss はプラントに注入されるアンモニアの内吸収塔から吸収されない量の割合で，プラント全体の非効率性の目安．同じデータを行列とベクトルで表現した stack.x と stack.loss が別にある．4 変数 21 ケース．
- swiss： 1888 年頃のスイスのフランス語圏行政区域 47 の標準化された出生率と社会経済的指標のデータ．単位はいずれも % で，Fertility 以外は人口に対する比率．Fertility は標準化出生率，Agriculture は農業従事男性，Examination は軍隊試験で最高得点を得た招集兵，Education は招集兵に対する小学校以上の教育歴，Catholic は (プロテスタントと比較した) カソリック信者，Infant.Mortality は生後 1 年以内の死亡者．6 変数 47 ケース．
- trees： 31 本のアメリカ桜 (black cherry) の倒木の測定データ．変数 Girth は (地面から 3 フィート 6 インチの高さの) 幹周り長 (単位 インチ)，Height は樹高 (単位 フィート)，Volume は体積 (単位 立方フィート)．3 変数 31 ケース．
- warpbreaks： 織機あたり (一定量の糸長に対応) の縦糸の切断数データ．変数 breaks は切断回数，wool は毛糸の種類を表す 2 水準 (A, B) の因子，tension は糸の張力を表す 3 水準 (L,M,H) の因子．組み合わせ AL, AM, AH, BL, BM, BH に対してそれぞれ 9 回の測定がされている．3 変数 54 ケース．
- women： 年齢 30–39 歳の米国女性の平均身長 (靴ばき時) と体重 (通常の室内着着用時) のデータ．変数 height は身長 (単位 インチ)，weight は体重 (単位 ポンド)．2 変数 15 ケース．

12.3　組み込みデータセット ベクトル・行列・配列

- HairEyeColor： Delaware 大学の 592 人の学生を髭色 ("Black" "Brown" "Red" "Blond")，瞳色 ("Brown" "Blue" "Hazel" "Green")，性別 ("Male", "Female") で分類した 3 元分類表．次元 $4 \times 4 \times 2$ の配列．
- Harman23.cor]: 7 歳から 17 歳までの女子 305 人の 8 項目の身体測定値から計算された相関行列．Harman の例 2.3．8 行 8 列の行列．
- Harman74.cor： 相関行列．シカゴの第 7,8 学年の児童 145 人の 24 項目の心理検査測定値から計算された 24×24 の相関行列．Harman の例 7.4．
- USPersonalExpenditure： 米国の個人支出データ (単位 百万ドル)．各行は支出項目 (食品とタバコ，家事，医薬と健康，私的支出，教育)，各列は年度 (1940,1945,1950,1955,1960)．5 行 5 列．
- VADeaths： バージニア州の 1940 年の千人当たりの死亡数データ．各行は年齢層

(50–54,55–59,60–64,65–69,70–74)，各列は性別と居住地 (都市，田舎) の組合せ．5 行 4 列．

- `WorldPhones`： 世界の各地での電話数 (単位 千)．各行は年度 (1951,1956–1961)，各列は地域 (北米，欧州，アジア，南米，オセアニア，アフリカ，中米)．7 行 7 列．
- `ability.cov`： 112 人に対する 6 種類の知能検査結果から得られた．成分 `cov` は共分散行列，`center` は中心 `c(0,0,0,0,0,0)`，`n.obs` は被験者数 112．6 行 6 列の共分散行列を含む 3 成分のリスト．
- `euro`： 1998 年 12 月 31 日で固定された EU 当初加盟国の旧通貨の真通貨ユーロへの交換比率．名前付きの長さ 11 のベクトル．
- `euro.cross`： 様々なヨーロッパの通貨間の交換比率．データセット `euro` から `outer(1/euro,euro)` で得られる．名前付きの 11 行 11 列の行列．
- `freeny.x`： 1962 年第 2 四半期から第 4 四半期までの国家歳入額 `freeny` 関連の説明変数．各列は `freeny.y` はラグ 1 の `freeny.y`，物価指数，所得水準，市況見通し．39 行 4 列．
- `iris3`：
  同じデータのデータフレーム表現である `iris` を参照せよ．$50 \times 4 \times 3$ の 3 次元配列．

- `islands`： 1 万平方マイルを越える陸地の千平方マイル単位の面積．長さ 48 の名前付きベクトル．
- `precip`： 1975 年の 70 の米国都市 (とプエルトリコ) の年平均降雨量 (単位 *inch*)．都市名の名前ラベルが付く．長さ 70 の名前付きベクトル．
- `rivers`： 北米の主要河川 141 の長さ (単位 マイル)．長さ 141 のベクトル．
- `stack.x`： データフレーム `stack` の最初の 3 変数を行列で表現したもの．21 行 3 列の行列．
- `stack.loss`： データフレーム `stack` の第 4 変数．長さ 21 のベクトル．
- `state.abb`： 米国州名の 2 文字による省略．長さ 50 の文字列ベクトル．
- `state.area`： 米国各州の面積 (単位 平方マイル)．長さ 50 のベクトル．
- `state.division`： 米国各州の区域名 (New England, Middle Atlantic, South Atlantic, East South Central, West South Central, East North Central, West North Central, Mountain, Pacific)．長さ 50 の 10 水準因子．
- `state.name`： 米国の完全な州名．長さ 50 の文字列ベクトル．
- `state.region`： 米国各州の所属する地域名 (Northeast, South, North Central, West)．長さ 50 の 4 水準因子ベクトル．
- `state.x77`： 米国各州の統計情報．列 `Population` は 1975 年元旦の推定人口，`Income` は 1974 年の一人あたり年収額，`Illiteracy` は 1970 年の非識字率 (人口あたりの % 値)，`Life Exp` は 1969–71 年の平均余命 (単位 歳)，`Murder` は 1976 年の人口 1 万人あたりの殺人と過失致死数，`HS Grad` は 1970 年の高校卒業学歴者の割合, `Frost` は 1931–1960 年の首都もしくは大都市における最低気温が零度以下の平均日数，`Area` は面積 (単位 平方マイル)．列名ラベル付きの 50 行 8 列の行列．
- `volcano`： ニュージーランドのオークランド火山帯の Maunga Whau 火山 (Mt. Eden) の $10m \times 10m$ 刻みの高度情報．行は東から西方向，列は南から北方向．Ross Ihaka が地形図をデジタル化して得たものであり，正確とは限らない．87 行 61 列．

12.4　組み込みデータセット 特殊なクラスオブジェクト

- `Titanic`： 2,201 観測値を 4 変数でクロス集計した 4 次元配列．タイタニック号の乗船客の運命を，乗船クラス，性別，年齢，そして生死で分類した情報．変数 `Class` は 4 水準 (一等，二等，三等船室，船員)，`Sex` は 2 水準 (男女)，`Age` は 2 水準 (子供と大人)，`Survive` は 2 水準 (生存，死亡)．
- `UCBAdmissions`： 4,526 個のデータを 3 変数でクロス集計した 3 次元文字列配列．カリフォルニア大学バークレイ分校大学院の 6 主要研究科の 1973 年の受験生を合否と性別で分類したデータ．変数 `Admit` は合否を表す文字列 `"Admitted"`, `"Rejected"`，変数 `Gnder` は性別を表す文字列 `"Male"`, `"Female"`，変数 `Dept` は研究科を表す文字列 `"A"`,...,`"F"`．
- `crimtab`： 42 行 22 列の計数値からなる表形式データ．Student(Gosset) の犯罪者データ．指の長さと身長の相関を調べるために用いられた．イングランドとウェールズの上級監獄の 20 歳以上の男性服役者のデータ．42 の行は指の長さ範囲の中点を表すラベル `"9.4","9.5",...` を持つ．22 の列は身長を表すラベル `"142.24","144.78",...` を持つ．
- `eurodist`： 長さ 210 のクラス `dist` のオブジェクト．ヨーロッパの 21 の都市間の道路距離 (単位 km) を与える．
- `state.center`： 2 成分のリストで米国各州の中心位置．長さ 50 のベクトルである成分 `x` は (負の) 経度，成分 `y` は緯度．

索引