

平成 13 年度 北海道大学文学部 行動科学演習 I/総合文化論基礎演習 資料
S-PLUS による線形計算とデータ解析

2001 年 10 月 3 日

大津起夫¹ ・ 行廣隆次²
北海道大学 文学部 行動システム科学講座³

¹ otsu@let.hokudai.ac.jp

² 1999 年 4 月より 京都学園大学 人間文化学部在職

³ 060-0810 札幌市北区北 10 条西 7 丁目

目次

第 I 部 基礎知識	1
1 はじめに	1
2 データ解析に利用可能なソフトウェア	2
2.1 SAS	2
2.2 S-PLUS	2
2.3 Mathematica	2
2.4 MATLAB	2
3 情報メディア教育研究総合センターの使い方	3
3.1 利用できるコンピュータ	3
3.2 コンピュータの起動	3
3.3 コンピュータ利用の終了	3
3.4 UNIX コンピュータへのアクセス	4
3.5 パスワードの変更	4
3.6 用語解説	4
4 S-PLUS の基本機能	5
4.1 起動	5
4.2 終了	5
4.3 入力行の編集	5
4.4 データの代入	6
4.5 オブジェクトの表示	7
4.6 作成済オブジェクトの一覧	8
4.7 オブジェクトの削除	8
4.8 利用説明の表示	8
4.9 Mule 上での S-PLUS の利用	9
5 データの基本操作	10
5.1 ベクトルの算術演算	10
5.2 数学関数と数列・並べ替え	11
5.3 指数と対数	12
5.3.1 指数	12
5.3.2 自然対数の底	12
5.3.3 対数	13
5.3.4 S-PLUS での指数と対数	13
5.4 比較と論理演算	14
5.5 条件分岐	15
5.6 繰り返し	16
5.6.1 for	16
5.6.2 while	17
5.7 数値演算上の注意	17

5.8 欠測値の扱い	18
第 II 部 線形計算	20
6 ベクトルと行列の操作	20
6.1 行列の意味	20
6.2 行列の作成	20
6.3 要素の参照	22
6.4 要素の変更	22
6.5 行列の計算	23
6.6 行和・列和	25
7 ベクトルの計算をグラフで確認	26
7.1 平行 4 辺形を描く	26
7.2 関数の定義と編集	27
7.3 回転をあらわす行列の作成	28
8 ベクトルの直交化	30
8.1 ベクトルの長さ (ノルム)・内積・角度	30
8.2 直交射影	31
9 行列式	31
10 連立 1 次方程式と逆行列	33
11 答えのない連立 1 次方程式	34
11.1 最小 2 乗法 lsfit	34
11.2 直交射影子	37
12 直交行列	38
13 特異値分解	38
14 多変数の 2 次関数	39
14.1 関数の最大・最小と対称行列の固有値	39
14.2 一般行列の固有値と固有ベクトル	41
第 III 部 データ解析	42
15 テキストファイルからの入力	42
15.1 ベクトルの入力	42
15.2 データフレームの作成	43
16 基礎統計機能	44
16.1 1 変数の統計量	44
16.2 分布の比較	45
16.2.1 データフレームの作成	45

16.2.2	枝葉図	46
16.2.3	ヒストグラム	46
16.2.4	箱ヒゲ図	47
16.2.5	分位点プロット	48
16.2.6	位置の差の検定	50
16.3	2変数の関連性	52
16.4	データセット longley	52
16.5	相関係数の検定	53
16.6	Spearman の順位相関係数と Kendall の τ (タウ)	53
16.7	2値変数の関連性	54
17	確率分布と乱数	55
17.1	2項分布	55
17.2	2項分布乱数	56
17.3	ポアソン (Poisson) 分布	57
17.4	正規分布	57
17.5	一様分布	58
18	統計モデルの推定	58
18.1	ロジスティック回帰	58
18.2	多重分割表の入力	59
18.3	ロジスティック回帰の推定	60
第 IV 部 作図と印刷		64
19	S-PLUS による作図	64
19.1	画面に表示する	64
19.2	プリンタへ印刷する	64
19.3	作図機器を変えると図も異なる	65
20	UNIX の印刷環境と PostScript	65
20.1	PostScript	65
20.2	S-PLUS から以外の印刷にも PS ファイル	66
20.3	計算経過の保存	66
20.4	印刷の手続き	67
20.5	印刷イメージを画面で確認する	67
21	2次元プロット	68
21.1	データの表示	68
21.2	複数の折れ線の表示	68
21.3	棒グラフと円グラフ	68
22	矢印や文字を書き込む	69

23 3 次元以上のプロット	69
23.1 persp	69
23.2 pairs	69
23.3 spin	70
23.4 条件付きプロット	71
23.5 XGobi	74
24 グラフ作成の詳細指定	75
第 V 部 演習篇	76
25 基本データ型	76
26 数値演算上の注意	76
27 判断と分岐	77
28 配列と繰り返し	77
28.1 数列と繰り返し	79
28.2 行列をつくる	79
28.3 ベクトルの一部を取り出す	80
28.4 問題の解答例と解説	80
29 リスト構造	82
29.1 サザエさん家族表現	84
29.2 関数の定義	85
29.3 課題の回答例	85
29.4 関数についての追加説明	86
30 多重配列の利用	86
31 乱数とシミュレーション (1)	88
31.1 コイン投げ	88
31.2 トランプゲーム	90
32 乱数とシミュレーション (2)	91
32.1 正規乱数	91
32.2 t 検定の検出力	91
33 乱数とシミュレーション (3)	93
33.1 乱数で面積を求める	93
33.2 相関を持つ変数 (2 変量正規分布)	94
34 Bootstrap 推定	96

35	グラフの作成	97
35.1	散布図	97
35.2	直線を引く	97
35.3	文字を書き入れる	98
35.4	複数の変数を同時に描画する	98
35.5	多角形の描画	99
35.6	円と楕円の描画	99
35.7	2次元正規分布の信頼楕円	100
36	回帰分析の方法	101
37	分散分析の方法	103
37.1	SAS で分散分析を行なう	103
37.2	S-PLUS による分散分析	105
第 VI 部	文献案内	108

第I部

基礎知識

1 はじめに

⁴ 従来、社会科学・行動科学の関連分野において利用されるデータ分析用のソフトウェアは、定型的な入力とそれに応じた定型の帳票形式の出力を備えたものが主流であった。しかしながら、これらの分野において利用される計量的手法の多様化を考えると、従来メインフレーム上で主流であったこのようなソフトウェアでは、新たな研究上の需要に答えることが難しくなっている。つぎにあげるものが、その理由である。

1. 従来の多くの統計ソフトウェアで採用している関係表形式のデータ構造は、定型業務データを表現するのには向いているが、研究データの表現としては必ずしも適切ではない。
2. J.W.Tukey らによる探索的データ解析 (EDA) の方法論の影響により、柔軟で対話的なデータ解析環境への要求が増えている。
3. 統計理論の急速な進展によって手法の多様化が著しいため、固定された機能しか持たないソフトウェアでは対応できない。マクロ記述などによって機能拡張を逐次実現する必要がある。
4. 計量モデルの利用技術が向上するにつれて、利用者がカスタマイズされたモデルを利用することが多くなってきた。
5. 乱数シミュレーションの利用が一般的になった。

従来、これらの要求を満たすには、研究者が独自に FORTRAN や C などのコンパイラ言語を用いて、入出力まで含めたプログラムを自作していた。最近開発されている数学・統計ソフトウェアにおいては、独自の高水準プログラム言語を備えることにより、上記の要求を満たすような設計がなされている。これらを使いこなすためには、独自言語の利用方法を学ぶ必要があるが、プログラムを全て自作するのに比べれば大幅に作業効率が向上する。これらのソフトウェアの商用化は、かなり以前から実現されていたが、従来、社会科学・行動科学系の教育においては広くは利用されてこなかった。原因の一つは価格の問題であり、もう一点はこれらを利用するためには線形代数を始めとする数学知識が必要とされるためと考えられる。

線形代数は基礎数学科目として、理系および社会科学系の大学初年度学生に広く教えられている。数値計算的側面の重視への動きはあるものの、その教育内容は大筋において過去 20 年程の間ほとんど変化していない [19]。しかし、上述のようなソフトウェアを利用することにより、計算アルゴリズムの詳細には立ち入ることなく、応用上重要な性質に重点をおいて学習することができる。

北海道大学では、平成 8 年度から情報処理教育センター (現在、情報メディア教育研究総合センター、以下センターと略す) に S-PLUS と Mathematica が導入され、学部の授業でこれらを利用することが可能になった。また平成 12 年度には MATLAB も導入されている。上の目的のためにどのソフトウェアを選択すべきかは難しい問題であるが、データ解析機能特に EDA 関連のグラフィック表示機能の充実により、この授業では S-PLUS を用いて計算機実習を行なうことにした。

本資料は UNIX 上の S-PLUS を利用した線形計算および記述統計機能および乱数の利用について説明してある。UNIX の基本操作については、最小限しか説明していない。S-PLUS の学習資料としてはテーマの範囲が狭いものに限定されている。より詳しい解説は、利用者マニュアルおよび巻末にあげた文献にある。平成 12 年現在、導入されているのは S-PLUS Ver.3.3J であり、以下の解説もこの版に基く。

⁴ 本稿にあらわれるシステム名・製品名などは一般にそれらの開発元の商標または登録商標です。

2 データ解析に利用可能なソフトウェア

S-PLUS はデータ解析用に開発されたソフトウェアであり、UNIX, Windows など多くのコンピュータ上で利用することができる。商品化は米国の Statistical Sciences Inc. という Seattle にある会社が行っており、日本での販売は (株) 数理システムがおこなっている。この原型になった S と呼ばれるソフトウェアは AT&T の Bell 研究所のデータ解析研究グループによって開発された。S-PLUS とは別に、S 言語に独自の拡張を加えた製品が (株) アイザックによって販売されている。センターでは、データ分析のために S-PLUS 以外に SAS と Mathematica、および MATLAB というソフトウェアが利用できる。

これらの4つのソフトウェアの特徴はつぎのようなものである。

2.1 SAS

数10万件にのぼる大量のデータの処理が可能である。現在、最も多く利用されている統計ソフトウェアの一つであり、北海道大学の大型計算機センターにも導入されている。メインフレームがデータ解析に用いられることが多かった時代に開発が着手され、かなり長い歴史を持っている。各種の統計手法が組み込まれており、データを入力するだけで関連のある各種の値が一覧表として出力される。企業内の業務におけるデータ分析にもしばしば用いられている。S-PLUS や Mathematica が配列およびリストと呼ばれるデータ構造を主な操作の対象としているのに対し、SAS の操作対象はファイル (関係型とよばれる構造を想定している) である。これが、SAS の大量データ処理について強く、逆に複雑な構造を持つデータが扱づらいという特徴の原因になっている。

2.2 S-PLUS

数千件程度の小～中規模の研究データの分析に向いている。対話的なデータ分析をグラフィカル機能を用いて柔軟に行える。各種の分析機能は「関数」として組み込まれているが、関連する値が SAS の様に一覧表の形で表示される訳ではないので、利用者が自分で各種の指定を行わなければならない。ベクトルや行列の数値計算に向いている。特に、探索的データ解析 (Exploratory Data Analysis) [22] と呼ばれる各種の手法やノンパラメトリック回帰関連の機能が充実している。残念ながら初心者向けの日本語の教科書が現状では少ない。英語での教科書は優れたものはいくつか出版されている。S-PLUS の原型となった S 言語はデータ解析用のソフトウェアとして AT&T の Bell 研究所のデータ解析研究グループによって開発された。S-PLUS はこれに機能拡張を行ない、商品化されたものである。なお S-PLUS とほぼ同様の言語仕様を持つフリーソフトウェア R も開発されており、Statlib、会津大学のミラーサイトなどから入手できる。

2.3 Mathematica

高機能な数学ソフトウェアであり、各種の記号処理を行うことができる。因数分解や数式の記号微分・記号積分が実行できるので、また複雑な数式の操作を伴う作業に有益である。数値計算機能も含まれている。また、複雑な図形を表示する機能が優れている。データ解析用の関数はあまり準備されていないので、利用者が自分でプログラムを作成する必要がある。また、大量のデータを扱うのには向いていない。

2.4 MATLAB

おもに線形計算 (行列演算) を中心とした、各種の数値計算を行うためのシステムである。S-PLUS より、高速な線形計算が可能であり、工学分野での利用が多い。信号処理や数値最適化など各種の技術計算

のための拡張モジュールが販売されている。

3 情報メディア教育研究総合センターの使い方

重要な注意

うまく動かなくなってきたからと言ってコンピュータの電源を切ってはいけない。教官、よく分かっている人等に相談すること。

3.1 利用できるコンピュータ

センターに登録された学生は、以下に示すコンピュータを利用することができる。

1. パソコン (日立 FLORA、Windows98)

センター内の実習室および学部端末室にあり、利用できる。実習では使わないが、ワープロや表計算ソフトを始め、各種アプリケーションや言語が使用できるので、各自調べて利用してよい。

2. センターのホストコンピュータ (UNIX)

S-PLUS が利用できるのはホスト名

ap2.ec.hokudai.ac.jp および ap3.ec.hokudai.ac.jp

である (端末パソコンのメニューによる選択時にはこれらの名前は表示されないが、センター外からの接続時には、これらの名前を用いる)。端末パソコンまたは、その他のネットワークにつながったコンピュータから利用する。PC から UNIX を接続する際には、メニューで ' 情報メディア教育研究総合センター S-PLUS ' を指定する。ただし、センターに所属する PC (センター内 PC および文学部 1 階のセンター PC) 以外では、センター側の指定により、X Window System が利用できないので、グラフィック表示が行えない。

3.2 コンピュータの起動

1. 電源スイッチ (本体右下) を入れる。

2. 初期画面が表示されたら、ユーザ ID とパスワード、コース名を入力し、リターンキーを押す (または ボタンをクリックする)。コース名は、行動科学演習 I / 総合文化論初等演習 (大津 後期) は 1-0006 (最初はエル)、行動計量学 (後期) は 1-0007 である。

3. 「 ~さん、こんにちは 」 と出るのでリターンキーを押すか、 をクリックする。

3.3 コンピュータ利用の終了

1. 画面左下の をクリックしてメニューを開く

2. をクリック

3. 電源はそのままにしておく。

3.4 UNIX コンピュータへのアクセス

1. 画面左側のアイコンから **UNIX コンピュータ** をダブルクリック (2 回続けてマウスの左ボタンを押す)
2. メニューから **(splus.ec.hokudai.ac.jp (SPLUS 情報メディア教育研究総合センター))** を選択する。
3. login に対してユーザ ID を入力して **リターンキー** を押す。
4. password に対してパスワードを入力して **リターンキー** を押す (これでホストコンピュータに接続される)。ユーザ ID かパスワードが間違っていた場合にはもう一度入力画面になるので、入力し直す。
5. ap2 21:% などのように、ホストコンピュータ名と番号 (コマンドの通算の数) が表示されたら、UNIX コンピュータが使える。
6. 利用を終えるときには、exit または logout と入力して、ホストコンピュータへの接続を切る。

3.5 パスワードの変更

パスワードは、本人を確認する唯一の手段なので、他人に知られないよう慎重に管理する必要がある。名前そのままや、誕生日、学生番号等、容易に想像が出来るものを使わない、定期的に変更する、等の管理を各自で気をつけること。

1. UNIX コンピュータに接続する。
2. passwd と入力する
3. 現在のパスワードの入力を求められるので、入力する (画面には表示されない)
4. 新しいパスワード入力を求められるので、入力する
5. 新しいパスワードをもう一度入力するよう求められるので、同じものを入力する

3.6 用語解説

マウス 画面上のポインタ (矢印) を移動させ、視覚的に操作するためのデバイス。左右 2 個のボタンがある (機種によってはボタンが 1 個のものや 3 個のものもある)。

クリック マウスボタンのクリック。どちらかのボタンを押す。

ダブルクリック (普通は左の) マウスボタンを 2 回続けてクリックする。

選択 メニュー項目やボタンを、マウスの左ボタンでクリックする。または、メニューの該当項目までカーソルキー (上下左右に向いた 4 つの矢印キー) で反転表示を移動させ、リターンキーを押す。

コントロールキー キーボードの左右の下にある **Ctrl** と書かれたキーのこと。特別な場合を除いて、他のキーと同時に押して利用する。この資料で **Ctrl+A** などと表記してある部分は、コントロールキーを押しながら、**A** のキーを押すことを意味する。^A などと表記されることもある。

4 S-PLUSの基本機能

4.1 起動

コマンドモード (ap2 21 などの文字が表示された状態) で `Splus` と入力する。先頭は大文字、それ以外は小文字である。また、`Splus -e` と入力すると、表 1 に示す方法によって入力行を編集することが可能である。

4.2 終了

> q()

() は必須である。ここで、> は S-PLUS が出力する入力待ちの記号であり、利用者がタイプする必要はない。

4.3 入力行の編集

UNIX のコマンドモードで、つぎのように入力すると、入力コマンドの編集機能が有効になる。

```
ap2 63: Splus -e
```

ここで、対話モードになって、>記号が表示される。この状態で、利用者は関数やデータを作成したり (オブジェクトと総称される)、各種の計算を行なえる。データや関数は、UNIX 版の S-PLUS では利用者のホームディレクトリの下での `.Data` というディレクトリに作成される。UNIX においては、(ピリオド) で始まるファイルやディレクトリは「隠れファイル/ディレクトリ」であり、通常は利用者から見えないようになっている。これらを確認する場合には、`ls -a` と入力する。`.Data` の中にどのようなファイルが含まれているかを確認する場合には、ホームディレクトリで `ls -la .Data` と入力すればよい。矢印キーは、S-PLUS の入力モードでは機能しないことに注意。行編集モードのキー機能は、vi エディタという UNIX 付属のテキストエディタに準じている。man vi で詳細を閲覧できる。

表 1: 行編集モードでの機能

キー	機能
<code>Esc</code>	入力モードからカーソル移動モードに移行
<code>l</code>	一文字前進
<code>h</code>	一文字後退
<code>j</code>	一つ前に入力されたコマンドを表示
<code>k</code>	表示されているコマンドのつぎに入力されたコマンドを表示
<code>x</code>	カーソル位置にある一文字を削除
<code>i</code>	カーソル位置の前に入力 (入力モードに移行)
<code>a</code>	カーソル位置の後に入力 (入力モードに移行)
<code>s</code>	一文字置き換え (入力モードに移行)

また、コマンドモードで `setenv S_CLEditor emacs` と指定してから、`Splus -e` と起動すると、`mule` に類似のキー操作によって入力行の編集をおこなえる。この場合は、入力モードと移動モードを区別する必要がない。キーの移動と編集はコントロールキーを併用することによって行う。

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

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

関数名 (引数 1, 引数 2, ...)

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

4.4 データの代入

2つの記号 `_` (下線)、および `<-` が値の代入の機能を持っている。どちらも同じ意味である。記号の左辺は、値が代入される変数を表し、右辺は代入する値である。S-PLUS は数値と文字列を扱える。ダブルクォート"またはクォート' で囲まれた文字は、文字列として扱われる。文字列については計算は行なえない (異同判断と大小比較は可能である)。変数および関数の名前は英字、ピリオド (.) または数字 1 文字以上からなるもので、先頭文字は数字であってはならない。S-PLUS が扱うことのできるデータ構造には、ベクトル、行列、リストなどがある。本稿ではベクトルと行列について主に説明する。また実数 (スカラー) は、長さ 1 のベクトルとして扱われる。

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

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

上で用いている `c` という名前の関数は、値をまとめてベクトルにするものである。同名のデータを作成しても、多くの場合問題は生じないが、混乱しないよう注意する必要がある。作成したデータは各自のファイルとして保存される。これらは意図的に利用者が削除しない限り保存され、次回に S-PLUS を起動したときにも利用できる。

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

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

のように、指定された内容の値を表示するだけである。先頭部分の[1]は、表示されている部分の先頭がベクトルの第1番目の要素であることを示している。また、S-PLUSにおいてはデータと関数とが形式上区別されないため(両者ともにオブジェクトと呼ばれる)、システムが用意する関数と同名のデータを作成すると、関数の内容を表示しようとしても、データの内容が表示されてしまう。通常は、あらかじめ準備された関数が保存されているディレクトリと、利用者のデータが保存されるディレクトリは異なるので、関数の内容が消滅してしまうことはない。また、関数を利用すると、引数の指定があることからデータが参照されたのではないことが自動的に判断される。利用者が作成したデータは、UNIXシステムにおいては、通常利用者のホームディレクトリの下に.Dataという名のサブディレクトリに保存される。作業用のディレクトリを変更する場合には、attachおよびdetachという関数を利用するが詳細は省略する。現在の作業ディレクトリおよび関数とデータを参照しているディレクトリの一覧はsearch関数によって表示される。

```
> search()
[1] "/home1/otsu/.Data"           "/opt/splus/33j/splus/.Functions"
[3] "/opt/splus/33j/stat/.Functions" "/opt/splus/33j/s/.Functions"
[5] "/opt/splus/33j/msi/.Functions" "/opt/splus/33j/msi/.Datasets"
[7] "/opt/splus/33j/s/.Datasets"   "/opt/splus/33j/stat/.Datasets"
[9] "/opt/splus/33j/splus/.Datasets"
```

4.5 オブジェクトの表示

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

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

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

```
> c
function(..., recursive = F)
.Internal(list(..., recursive), "S_unlist", T, 1)
```

オブジェクトが多くの要素を含む場合には、表示を行うと画面が流れてしまう場合がある。この際には、`page` 関数を用いて

```
> page(air)
```

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

4.6 作成済オブジェクトの一覧

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

```
> objects()
 [1] ".Last.value" "a"          "ab1"        "aqr"        "asvd"
 [6] "b"           "ca"         "cd2"        "d"          "d1"
[11] "e"           "last.dump" "mat1"       "mat2"
```

4.7 オブジェクトの削除

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

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

4.8 利用説明の表示

関数 `help` は、関数や S-PLUS が用意しているサンプルデータの用法や内容を説明する。現在のところセンターでは英語で表示される。特殊記号についての説明を表示する場合には、記号を `"` で囲んで指定する。

```
> help(help)
> help(c)
> help(ls)
> help("*")
```

`help` 命令による説明画面が表示されたなら、つぎの操作が可能である (UNIX の `less` コマンドと同じである)。

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

また、つぎのように指定することにより、X Window System 上のグラフィカル・インタフェースを用いた説明が利用可能である。

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

このように入力すると、関連テーマ別に関数名などの一覧がウィンドウに表示される。マウスを用いて該当するものをクリックすると、説明が表示される。ウィンドウを閉じるには、左上部分のメニューで **File** を選択し、つぎに **Close** と選択する。

4.9 Mule 上での S-PLUS の利用

S-PLUS(および他のプログラミング言語) を利用するのに、Mule(emacs) の shell モードを用いると便利である。計算を実行過程や表示結果がバッファに保存される。Mule の編集機能を用いて、結果の整理やレポートを作成できる。Mule の操作に慣れるのに多少時間がかかるが、応用が効く。この場合には S-PLUS の起動時に、`-e` オプションをつけない。

1. UNIX のコマンドモードで `mule &` と入力する。
2. Mule のウィンドウが表示されるので、**Esc** x と入力すると画面の左下に M-x と表示され入力待ちになる。ここで `shell` と入力すると、shell モード (ap2 32: などの入力待ちの表示がでる) に移行する。
3. shell モードで `Spplus` と入力すると、S-PLUS が起動する。
4. この状態で、S-PLUS の `help` 関数を用いるとうまく表示がされない。これを回避するには、つぎの 2 つのいずれかを用いる。

上に説明したグラフィカル・インタフェースを利用する。

S-PLUS のコマンドで `options(pager="cat")` と入力する。これにより、ヘルプの表示に `less` が利用されなくなり、ヘルプファイルの内容が一期に表示される。もとに戻す (ヘルプの表示に `less` コマンドが利用されるようにする) には、`options(pager="less")` と入力する。

5. `Spplus` を終了するには、上に説明したように `q()` と入力する。
6. Mule の shell モードを終了するには、UNIX コマンド `exit` を入力する。
7. Mule を終了するには、**Ctrl+X** **Ctrl+C** と続けて入力するか、メニューから **File** → **Exit Emacs** を選択する。

5 データの基本操作

5.1 ベクトルの算術演算

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

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

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

```
> p1 <- c(10,20,30,40,50)
> p2 <- c(1,2)
> p1 + p2
[1] 11 22 31 42 51
Warning messages:
  Length of longer object is not a multiple of the length of the
shorter object
  in: p1 + p2
```

また、欠測値はNAとして表現される。ただし全ての関数が欠測値を持つベクトルの入力を許している訳ではない。また、無限大の大きさを持つ数値はInfで表される。

```

> sqrt(c(3,2,1,0,-1,-2)) # 負の数の平方根は求められない。
[1] 1.732051 1.414214 1.000000 0.000000 NA NA
Warning messages: # 警告が表示される。
1: (-1)^(0.5) DOMAIN error in: x^0.5
2: (-2)^(0.5) DOMAIN error in: x^0.5

> ab3 <- 1/c(3,2,1,0,-1) # 0 の逆数は無限大。
> ab3
[1] 0.3333333 0.5000000 1.0000000 Inf -1.0000000

```

表 2: 主な算術演算子一覧

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

5.2 数学関数と数列・並べ替え

S-PLUS では多くの数学関数が利用可能であるが、これ以降の課題で必要なものを中心に、いくつかを表 3 に示す。ベクトルを引数として数学関数を適用すると、ベクトルの各要素に関数が適用された値が新たなベクトルとして作られる。

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

関数sortは、データの要素を昇順に並べ換える。

```

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

```

表 3: 数学関数一覧

演算子	機能	演算子	機能	演算子	機能
abs	絶対値	exp	指数関数	log	自然対数
log10	常用対数	sqrt	平方根	sin	正弦関数
cos	余弦関数	tan	正接関数		
演算子	機能	演算子	機能	演算子	機能
ceiling	整数への切上げ	floor	整数への切捨て	trunc	絶対値の切捨て
round	四捨五入 (指定精度への丸め)				

5.3 指数と対数

高校の数学で履修済みとは思いますが、指数と対数（統計の関係した文献では頻繁にでてくる）について復習する。

5.3.1 指数

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

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

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

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

5.3.2 自然対数の底

年間 100 パーセントの利子がつく預金を考える (こんなのは普通ないが)。1 年間に利子が元金に加えられる階数が 1 回であれば、1 年後の預金は 2 倍になり、2 年後には 4 倍、 n 年後には 2^n 倍になる。また、半年に 1 回利子が元金に加えられるならば、半年あたりの利子は元金の $1/2$ 倍であるから、半年後には元金は 1.5 倍になり、1 年後には元金は 1.5^2 倍になる。同様に、 n 年後には $(1 + 1/2)^{2n}$ 倍になる。もし、1 年間に利子が元金に加えられる回数が k 回であるならば、1 年後には元金は $(1 + 1/k)^k$ 倍になり、 n 年後には $(1 + 1/k)^{kn}$ 倍になる。 k をどんどん大きくしてゆくと (つまり瞬間複利計算を考えることになる)、

$(1 + 1/k)^k$ は発散せずにある数に収束する。これが自然対数の底と呼ばれるものであり、 $e = 2.71828\dots$ として表される。理論的に e は重要な数であり、多くの数学公式に現れる。 e を底とする指数関数 e^x は $\exp(x)$ とも表記される。つぎの公式が知られている。

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

5.3.3 対数

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

$$\log_2 1 = 0, \log_2 2 = 1, \log_2 4 = 2, \log_2 8 = 3, \log_2 \frac{1}{2} = -1 \quad (1)$$

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

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

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

$$\log_a (bc) = \log_a b + \log_a c \quad (2)$$

$$\log_a \frac{1}{b} = -\log_a b \quad (3)$$

$$\log_a b = \frac{\log_d b}{\log_d a} \quad (4)$$

$$(5)$$

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

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

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

5.3.4 S-PLUS での指数と対数

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

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

課題

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

例えば

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

または

```
> x <- seq(0,5,0.05)
> y <- sqrt(x)
> plot(x,y)
```

5.4 比較と論理演算

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

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

文字列の大小判断は、コード表の順による。

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

論理演算もまたベクトルの要素毎に値が計算される。

```

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

```

表 4: 比較演算子一覧

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

表 5: 論理演算子一覧 (ベクトルを引数とするもの)

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

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

5.5 条件分岐

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

```

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

```

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

```

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

```

また `Splus` にはつぎのような論理値をとるベクトルに対応した関数が用意されている。

```

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

```

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

5.6 繰り返し

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

5.6.1 for

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

```
for(変数名 1 in 表現 1) 表現 2
```

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

```
> for(i in 1:10) cat(i,",")      # 1:10 は 1 から 10 までの数列
1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,10 > # 最後に改行がないので、>が続いている。
>
for(i in 10:1) cat(i," ")      # 今回は 10 から 1 までの数列
10 9 8 7 6 5 4 3 2 1 >
>
> cities <- c("Hakodate","Sapporo","Asahikawa","Obihiro")
> for(City in cities){cat(City);cat(rep(" ",11-nchar(City)));
+   cat("is in Hokkaido.\n");}
Hakodate is in Hokkaido.
Sapporo is in Hokkaido.
Asahikawa is in Hokkaido.
Obihiro is in Hokkaido.
```

5.6.2 while

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

```
while(条件 1) 表現 1
```

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

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

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

5.7 数値演算上の注意

1. 一般的には整数演算の桁あふれには何の警告も生じない。ただし、S-PLUS では通常は浮動小数点演算によって計算されるので、特別な場合 (C、FORTRAN など他言語の呼び出しの利用など) を除いて、あまり心配する必要はない。

```

> as.integer(2000000000)
[1] 2000000000
> as.integer(2000000000)+as.integer(2000000000)
[1] -294967296 # 桁あふれのため負の数になってしまった。

```

2. 浮動小数点による小数の表現には、一般的には誤差が生じる。高精度計算を必要とするのでなければ、実用上それほど気にする必要はないが、等号による判断を計算結果について行う場合には、注意を要する。

```

> options(digits=17)
> 0.1
[1] 0.10000000000000001
> options(digits=7)

```

3. 浮動小数点の精度は絶対値について相対的である .

5.8 欠測値の扱い

S-PLUS で欠測値は NA として扱われるが、欠測値を比較 (==) によって識別するためには、引用符で囲って 'NA' と指定しなければならない。値が欠測値であるか否かを判断するには is.na 関数を用いる。

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

S-PLUS でベクトルから欠測値を除去するには、次のようにする。

```

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

```

下は、上に説明したものの例を参考までに示す。

```
> a <- NA      # 欠測値を代入
> a
[1] NA        # 表示は NA
> a == NA     # 比較の結果が欠測となる
[1] NA
> a == 'NA'   # NA を引用符で囲むと比較が真
[1] T
> is.na(a)    # is.na は機能する。
[1] T
> b <- 0/0    # 非数値になる結果を代入
> b
[1] NA        # 表示は NA
> b == NA     # 等号が成立しない
[1] NA
> b == 'NA'   # 等号が成立しない
[1] F
> b == NaN    # これでも等式が真にならない。
[1] NA
> b == 'NaN'  # これで真。
[1] T
> is.na(b)    # is.na は真となる
[1] T
```

第II部

線形計算

6 ベクトルと行列の操作

6.1 行列の意味

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

m 次元のベクトルを定義域とし、 n 次元のベクトルを値域とする関数 f を考えよう。 m 個の数 $x = (x_1, \dots, x_m)^T$ を一つ決めると、 n 個の数 $y = (y_1, \dots, y_n)^T$ が f によって一指定まる。ここで、記号 T は横ベクトルを縦ベクトルに変換すること (転置, transpose) を意味する。このとき f がつぎのような性質を持っているものとする。

1. $f(ax) = af(x)$ 。ここで a は実定数である。写像 f が作用する前に a 倍するのと、作用したあとで a 倍したものが同じことを意味する。
2. $f(x_1 + x_2) = f(x_1) + f(x_2)$ 。これは、写像 f が作用する前にベクトルの和を求めると、各々のベクトルに写像を作用させた後に和を求めたものが同一であることを意味する。

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

$$y_i = a_{i1}x_1 + \dots + a_{im}x_m \quad (6)$$

という式で表されることが分かる。逆に、ここで現れる $n \times m$ 個の係数 $\{a_{ij}\}$ を 1 セット決めると、線形写像が一指定まることになる。

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

ある行列が与えられたとき、それを特徴づける様々な値があるが、その一つにランク (rank) がある。ランクは階数とも呼ばれる。 n 行 m 列の行列 $A = (a_{ij})$ について、その m 個の列ベクトルを a_1, \dots, a_m と表記することにする。ある m 次元ベクトル x に A を作用させると、

$$y = Ax = x_1a_1 + \dots + x_ma_m$$

となる。 x の値が様々に変化すると y の値もそれにつれて変わる。しかし、もし A が特別な値である場合には y の取り得る値は、ある特定の範囲に限定されたものであるかも知れない。極端な場合、 A の要素が全てゼロであれば y はゼロベクトル以外にはなり得ない。このようにして作られる y の全体を線形写像 A の像 (イメージ) と呼び、 $\text{Im}A$ と表記する。 A のランク ($\text{rank}A$) は、 $\text{Im}A$ の次元 (1 次独立なベクトルを最大何個とれるか) によって定義される。これはまた、 a_1, \dots, a_m の中から選び出される 1 次独立なベクトルの最大の個数に等しい。いささか込み入った証明が必要であるが、 $\text{rank}A = \text{rank}A^T$ であることが示せる。また、 $\text{rank}A = \text{rank}AA^T = \text{rank}A^TA$ も成立する。

6.2 行列の作成

S-PLUS では、1 次元的なベクトルから行列を作る方法が幾つか提供されている。関数 `matrix` は、ベクトルを指定された大きさに区切ることにより、行列を作成する。最初の引数はベクトルであり、2 番目の引

数は行数、3番目の列の数を意味する。関数`rbind`と`cbind`は、ベクトルを束ねることにより行列を作成する。`rbind`はベクトルを行とみなして結合を行なう。また、`cbind`は列方向の結合を行なう。これら2つの関数の引数は、何個であってもよい。また、引数にはベクトルだけでなく行列も許される。

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

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

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

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

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

6.3 要素の参照

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

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

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

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

6.4 要素の変更

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

```

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

```

6.5 行列の計算

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

```

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

```

```

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

```

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

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

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

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

```

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

```

課題

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

6.6 行和・列和

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

```
apply(行列 1, 次元 1, 関数 1)
```

のように指定する。ここで、行列 1 は計算の対象となるものであり、次元 1 は整数値であり計算を適用する次元の指定である。次元 1 が 1 ならば各行に関数 1 を適用し、2 ならば列ごとに関数を適用する。関数 `apply` は、より高次元の配列 (関数 `array` によって定義される) にも適用できる。配列の利用については、本稿では省略した。

```

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

```

apply と類似の機能を持つ関数として lapply (リスト要素への関数の適用)、sapply (lapply と類似の機能を持つが、ベクトルを結果とする)、tapply (指定された変数によって分類を行い、各グループに関数を適用) などの関数がある。

7 ベクトルの計算をグラフで確認

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

7.1 平行4 辺形を描く

2つの2次元ベクトル v_1, v_2 とそれらの和をあらわすベクトル v_3 によって作られる平行4 辺形を描くことを試みる。

下の例で、x11() は X Window System において描画用のウィンドウを起動する命令である。また、センターのシステムでは、x11() と指定するかわりに、motif() と指定することもできる。こちらの報既に描画用のウィンドウが画面上にある場合には、指定する必要はない。また、plot はデータの散布図を表示する関数であり、第1引数のベクトルが横軸の座標、第2引数が縦軸の座標を与える。arrows(p1,p2,q1,q2) は、座標(p1,p2) から座標(q1,q2) に向かう矢印を描く。関数text(x,y,z) は、各*i* について (x[i],y[i]) の座標で示される位置に、文字列 z[i] を重ねて表示する。ただし、plot を実行せずに、この命令だけを用いても描画されない。図1 に描画例を示す。また、複数の文字列や数値をつなげて新たな長い文字列を作るには paste を用いる。

```

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

```

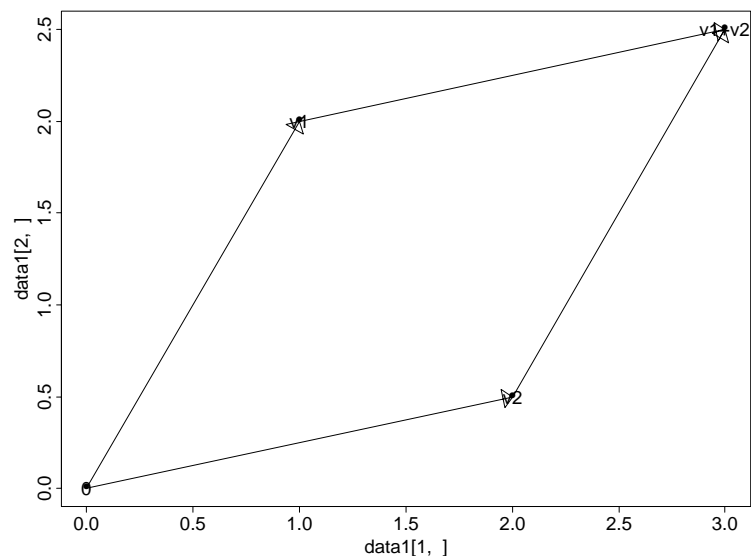


図 1: ベクトル和による平行四辺形の描画

7.2 関数の定義と編集

込み入った操作を何度も繰り返してタイプするのは面倒である。S-PLUS には、幾つかの操作をひとまとめにする機能(関数定義)がある。つぎのように入力すると、 2×2 の行列を指定して、その各列を2次元ベクトルとみなし、平行4辺形を描く関数を定義することができる。関数の中では、命令と命令の間は;(セミコロン)で区切る。行の先頭の+は指定が完結していないことを示すためにS-PLUSが自動的に表示する記号である。

```

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

```

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

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

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

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

エディタとしてviの代わりにmuleを使うためには、つぎのように指定する。

```
para1 <- vi(para1,editor="mule")
```

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

プログラムを編集するもう一つの方法は、あらかじめテキストファイルにmuleなどでS-PLUSのプログラムを作成しておき、それを読み込むことである。プログラムの記述されているテキストファイル名をsprog1とし、これがS-PLUSを起動したディレクトリにあるとすると、S-PLUSでsource("sprog1")と実行することにより、ファイルに記述されたプログラムがS-PLUSに読み込まれ実行される。記述される内容は、多くの場合対話的に実行しているデータの定義であってもよいし、ここで説明した関数の定義であってもよい。

7.3 回転をあらわす行列の作成

つぎのようにして、平面上の30度の回転をあらわす行列をつくる。

```

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

```

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

上と同様にして、60°の回転をあらわす行列を作成することができる。

```

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

```

つぎの行列data4は、左右の反転をあらわす。

```

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

```

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

```

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

```

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

課題

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

例えば、次の関数 test1 では、関数が返す値は $4*a$ であり、それ以前に計算される a や $2*a$ は、関数が返す値には直接には影響しない。

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

8 ベクトルの直交化

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

8.1 ベクトルの長さ (ノルム) ・ 内積 ・ 角度

ベクトル x, y がそれぞれ

$$\mathbf{x}^T = (x_1, x_2, \dots, x_p), \quad \mathbf{y}^T = (y_1, y_2, \dots, y_p) \quad (7)$$

であるとする。 x と y の内積はつぎの式であらわされる。

$$(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y} = x_1 y_1 + x_2 y_2 + \dots + x_p y_p \quad (8)$$

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

ベクトル x の長さ (ノルム) $\|x\|$ はそれ自身との内積の平方根であるので、`sqrt(sum(x*x))` とすればよい。また、関数 `vecnorm(x)` によっても、ノルムを求めることができる。

2 つのベクトルのなす角度 θ はつぎの公式を使って求めることができる。

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

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

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

ラディアンではなく度であらわしたい場合には

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

とすればよい。

課題

適当なベクトルを設定し、それらのなす角度を求めてみなさい。

8.2 直交射影

2つのベクトル x_1, x_2 があるとき、 $y = (x_2, x_1)x_1/\|x_1\|^2$ を x_2 の x_1 で張られる空間への直交射影という。 $x_2 - y$ と x_1 とは直交する。実際、

$$(x_2 - y, x_1) = (x_2, x_1) - (y, x_1) = (x_2, x_1) - (x_2, x_1)(x_1, x_1)/\|x_1\|^2 = 0 \quad (10)$$

である。

これと同様の考えに基づいて、1次独立なベクトルから、次々と互いに直交し長さが1であるベクトルを求めることができる。その手順はつぎのようなものである。まず、1次独立なベクトルの組を x_1, x_2, \dots, x_p とする。

1. $z_1 = x_1/\|x_1\|$ と置くと長さ1である。
2. $y_2 = x_2 - (x_2, x_1)x_1/\|x_1\|^2 = x_2 - (x_2, z_1)z_1$ とし、さらに $z_2 = y_2/\|y_2\|$ とする。
3. $y_3 = x_3 - (x_3, z_1)z_1 - (x_3, z_2)z_2$ とする。 z_2 と同様に $z_3 = y_3/\|y_3\|$ とする。
4. 以下同様に $y_i = x_i - \sum_{j=1}^{i-1} (x_i, z_j)z_j$ とし、 $z_i = y_i/\|y_i\|$ とする。

課題

$x_1^T = (1, 1, 2), x_2^T = (1, -1, 2), x_3^T = (-1, 2, 1)$ とおき、これらから上の手順に基づき、互いに直交する長さ1のベクトルの組を求めなさい。

9 行列式

$p \times p$ の正方行列 A について $\det A$ または $|A|$ という記法によって A の行列式をあらわす。 $A = (a_1, a_2, \dots, a_p)$ としよう。 A の行列式は A の p 個の列ベクトルによって構成される平行(超)多面体の符号付きの体積である。 $p = 2$ のときは、2つのベクトル a_1, a_2 の2つのベクトルによって形づくられる平行四辺形の面積、すなわち $0, a_1, a_1 + a_2$, および a_2 の4点で囲まれた領域の面積である。ただし、ベクトルの位置関係によって符号が変わる。 a_1 から a_2 への変化が逆時計回り(正の角度)のときには正符号であり、時計回り(負の角度)の時には負符号である。但し回転の角度は180度以下とする。 $p = 3$ の時は3つのベクトルによって構成される平行6面体(直方体を歪めたもの)の体積であり、ただしこの場合もベクトルの向きによって符号が変わる。厳密には、行列式はつぎの性質を満たす正方行列の関数として定義される。

1. $\det I = 1$ ただし I は単位行列。
2. ある列を c 倍すると、行列式の値も c 倍になる。
3. A の第 j 列 a_j を別のベクトル b_j に置き換えた行列を B とする。また、 $a_j + b_j$ に置き換えた行列を C とする。このとき $\det C = \det A + \det B$ となる。
4. 2つの列を交換すると、行列式は絶対値が同じで符号が逆転する。

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

また、行列式の定義からつぎの性質が導かれる。

1. 2つの列が同じであれば行列式は0である。
2. 1つの列が他の列の定数倍であれば行列式は0である。
3. ある列を定数倍したベクトルを別の列に加えても、行列式の値は変わらない。
4. A を $p \times p$ の行列とする。つぎの4つの条件は全て同値である。このとき A は正則 (regular) であるという。

$\det A \neq 0$ であること

A の列が1次独立であること

$A_{p \times p}$ のランクが p であること

逆行列 A^{-1} が存在すること

5. $\det(AB) = \det(A) \det(B)$

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

7. $\det A = \det A^T$ (T は転置を示す。)

8. 上の定義に示した性質は、列を行と読み換えてもすべて成立する。

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

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

行列式が用いられるのは、多変数の微積分学や座標変換の計算が多い。これらの証明やさらに詳しい性質については、線形代数の教科書を参照のこと。

残念ながら、何故かしら S-PLUS には行列式を求める関数が含まれていない。ただし、次に示す関数を定義することにより、行列式を求めることはできる。実は下に示した中で用いている関数 `eigen` は、行列の固有値と固有ベクトルを求める関数であり、`eigen(x)$values` は行列 x の固有値を成分とするベクトルである。固有値と固有ベクトルについては後述するが、固有値の積は行列式と一致する。行列 A の成分が全て実数であれば行列式の値はつねに実数であるが、 A が対称でない場合、つまり $A \neq A^T$ の場合には A の固有値は実数とは限らない。

```
det <- function(x) prod(eigen(x)$values)
```

行列式を求めるためのより効率的な方法は、次のような関数を定義することによって可能になる。この関数は正式には説明されていない S-PLUS の機能を利用している。ここで `qr(x)` は行列 x の QR 分解と呼ばれる計算を行なう関数である。QR 分解とは、 $n \times m$ の行列 X を長さ n の m 個の直交列ベクトルからなる行列 Q と $m \times m$ の上三角行列 R との積 $X = QR$ として分解する計算方法である。S-PLUS によって得られる内部の計算結果 `qr(x)$qr` の形式はいささか複雑で上の QR と単純には対応しないが、対角成分は R の対角成分に対応する。三角行列の行列式は、対角要素の積として表されるので、`prod(diag(qr(x)$qr))` は $\det R$ に対応する。一方 X が正方行列なら、 Q は $n \times n$ の大きさであり、直交行列 (後述) となる。直交行列の行列式は、その性質から $+1$ か -1 のいずれかである。QR 分解の計算手順から、ピボット選択と

呼ばれる計算手順を用いていなければ、 $\det Q = (-1)^{m-1}$ となる。この2つの性質を用いて行列式を計算している (B.Ripley の snws メーリングリストへの投稿による)。

```
detbyqr <- function(x) prod(diag(qr(x)$qr)) * (-1)^(ncol(x) - 1)
```

課題

関数 `det` および `detbyqr` を自分で定義し、上に示した行列式の性質を確認してみなさい。

10 連立1次方程式と逆行列

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

1. $\det A \neq 0$ すなわち A が正則であれば、 x の値は、常に一通りに定まる。
2. $\det A = 0$ である場合には、解が存在しない場合 (不能) と、解が複数存在する場合 (不定) とがある。 A が正則でないことを特異 (singular) であるという。解が複数存在するのは、 $\text{rank} A = \text{rank}(A|b) < p$ のときであり、解が存在しないのは $\text{rank} A < \text{rank}(A|b)$ の場合である。ここで $(A|b)$ は A の横に縦ベクトル b をならべた $p \times (p+1)$ の行列である。

A が正則であるときには、 $Ax = 0$ となるベクトル x はゼロベクトルのみである。また、このとき逆行列 A^{-1} が存在する。 A の逆行列とは、 $AX = I_p$ となる行列 X のことである。ここで I_p は、 p 次の単位行列を表す。 X の列ベクトルを x_1, \dots, x_p とし、また I_p の j 列を e_j とする。方程式 $AX = I_p$ の両辺の j 列をとると、 $Ax_j = e_j$ である。 A が正則であれば、これらの p 個の連立1次方程式は必ず解を持つので、 x_j が求まり、 X を定めることができる。 $AX = I_p$ が成立すれば、つぎの議論から $XA = I_p$ であることも分かる。

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

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

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

> solve(A,c(5,6,7)) # b^T=(5,6,7) において x を求める。
[1] 6.00000e+00 1.00000e+00 1.35974e-15 # b が横ベクトルであっても OK。
```

```

> a
      [,1] [,2] [,3]
[1,]    1    4   -1
[2,]    2    5    1
[3,]    3    6   -1
> solve(a)
# 引数が一つの場合は逆行列
# を与える。
      [,1]      [,2] [,3]
[1,] -0.9166667 -0.1666667  0.75
[2,]  0.4166667  0.1666667 -0.25
[3,] -0.2500000  0.5000000 -0.25
> solve(a) %% a
# 逆行列になっていることの確認
      [,1]      [,2]      [,3]
[1,]  1.000000e+00 -2.664535e-15  5.551115e-16
[2,]  4.440892e-16  1.000000e+00 -1.387779e-16
[3,] -4.440892e-16 -1.332268e-15  1.000000e+00
> a %% solve(a)
      [,1]      [,2]      [,3]
[1,]  1.000000e+00  0.000000e+00  2.775558e-16
[2,] -3.330669e-16  1.000000e+00 -5.551115e-17
[3,] -3.330669e-16  5.551115e-17  1.000000e+00

```

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

逆行列にはつぎのような性質がある。

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

課題

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

11 答えのない連立 1 次方程式

11.1 最小 2 乗法 `lsfit`

前節で、連立 1 次方程式

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

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

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

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

ここで、 A の次元は $n \times p (n \geq p)$ であり、 A のランクは p であるとする。つまり、 A の各列は 1 次独立なベクトルである。簡単のために $n = 3, p = 2$ の場合を考えてみよう。 $A = (a_1, a_2)$ とすると a_1, a_2 の各々は次元 3 のベクトルであり、3 次元空間の中の同一直線上にはない。また、 $x_1 a_1 + x_2 a_2$ の全体 (x_1, x_2 が様々に変化する場合のベクトル全体) は、原点を通る 1 つの平面をなす。

より具体的に $a_1 = (1, 1, 1)^T$, $a_2 = (0, 1, 2)^T$ としよう。この場合 $b = (0, 2, 3)^T$ とおくと、 $Ax = b$ を満たす x は存在しない。多変数の微分を用いた計算から、実は

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

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

課題

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

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

```
> a
      [,1] [,2]
[1,]    1    0
[2,]    1    1
[3,]    1    2
> b
[1] 0 2 3      # b は横ベクトルでよい。
```

次のように、関数 `lsfit` を使うと、 x の推定を行なえる。

```

> ls1 <- lsfit(a,b,intercept=F)
> ls1

$coef:
      X1  X2          # X の推定値
0.1666667 1.5

$residuals:
[1] -0.1666667  0.3333333 -0.1666667 # 残差 b-Ax

$intercept:
[1] F

$qr:
$qr$qt:
[1] -2.8867513 -2.1213203 -0.4082483

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

$qr$ql:
[1] 1.577350 1.258819

$qr$rank:          # A のランクを計算して求めたもの。
[1] 2

$qr$pivot:
[1] 1 2

$qr$tol:
[1] 1e-07

```

ls1 は、内部にいろいろな要素をもつリスト (list) と呼ばれる構造を持つデータになっている。\$に続く名前が、データの要素をしめしている。このうち、ls1\$coef が推定された \hat{x} である。ls1\$residuals は残差 $b - \hat{y} = b - A\hat{x}$ を示している。つぎのように指定すると、 $\hat{y} = A\hat{x}$ を求めることができる。ここで `matrix(ls1$coef,2,1)` は、求められた係数ベクトル ls1\$coef を 2×1 の行列に変換するものである。

```
> a %*% matrix(ls1$coef,2,1)
      [,1]
[1,] 0.1666667
[2,] 1.6666667
[3,] 3.1666667
```

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

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

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

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

11.2 直交射影子

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

一般に、次の 2 つの性質を満たす正方行列は直交射影子になっている。

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

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

課題

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

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

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

12 直交行列

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

一般に A^{-1} が A の逆行列であるとする $AA^{-1} = A^{-1}A = I$ である。従って $U^T U = U^T U^T = I$ がいえる。これより、 U の各行も長さ1で互いに直交していることがわかる。行列式の性質より、

$$\det U \det U = \det U \det U^T = \det(UU^T) = \det I = 1 \quad (14)$$

である。これより、 $\det U = \pm 1$ である。

課題

1. 次の行列が直交行列であることを、確認しなさい。

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

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

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

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

13 特異値分解

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

$$A = UDV^T$$

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

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

```

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

> mat1svd <- svd(mat1) # 特異値分解の実行し結果を代入
> mat1svd # 特異値分解の中身
$d:
[1] 2.7578164 0.6280515 # 対角要素

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

$u: # U
      [,1] [,2]
[1,] -0.4718579 -0.8816746
[2,] -0.8816746 0.4718579
> mat1svd$u %*% diag(mat1svd$d) %*% t(mat1svd$v) # 復元する。
      [,1] [,2] [,3]
[1,]    1 1.110223e-16    1
[2,]    2 1.000000e+00    1

```

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

課題

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

14 多変数の 2 次関数

14.1 関数の最大・最小と対称行列の固有値

1 変数の 2 次関数は

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

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

まず、2変数の2次関数は一般的につぎのような形式で書きあらわされる。

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

これを行列とベクトルを用いて書き表すとつぎのようになる。

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

ただしここで

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

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

この関数を x の要素毎に微分してみよう。まず、 x_1 で微分すると

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

である。また、 x_2 で微分すると、

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

である。

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

と表記することになると、

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

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

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

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

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

\mathbf{U} の第 j 列を \mathbf{u}_j とすると、

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

が成立する。ここで λ_j は Λ の第 (j, j) 要素である。

先の2次関数 $f(\mathbf{x})$ に戻ると、

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

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

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

となる。

λ_1, λ_2 がともに正の場合は、関数曲面の形はお椀型であり、またともに負の場合は、お椀を伏せた形になる。一方が正で他方が負の場合には、馬の鞍のような形になる。また、片方の固有値が零の場合には、放物線を横に移動した痕跡の形になる。これから、固有値が全て正の場合には、 x_0 は $f(x)$ の最小値であり、また、固有値が全て負の場合には、最大値になることが分かる。

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

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

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

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

14.2 一般行列の固有値と固有ベクトル

実数を要素とする p 次の正方行列 (必ずしも対称ではない) A について、

$$Ax = \lambda x$$

が成立するものとする。ここで、 λ は実数または複素数であり、 x は実数または複素数からなる 0 ではないベクトルである。この条件を満たす λ のことを A の固有値とよび、 x を固有ベクトルとするのが本来の定義である。

A が対称行列である場合には、前節に示した行列分解によって固有値と固有ベクトルを求めることが可能であり、 A の要素が実数ならば、固有値も固有ベクトルも全て実数である。しかし、 A が対称ではない場合には、 A の要素が実数であっても、固有値が実数である保証はない。 λ が A の固有値であれば、 $(A - \lambda I_p)x = 0$ であるので、 $A - \lambda I_p$ の行列式は零でなければならない。 p 次正方行列の行列式は行列要素の p 次式であるので、 λ はこの p 次式の解になっている。非対称行列の固有値は常微分方程式の解を求める場合などに利用される。

第 III 部

データ解析

15 テキストファイルからの入力

15.1 ベクトルの入力

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

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

sample1.dat の内容

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

sample2.dat の内容

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

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

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

Microsoft Windows のテキストファイルを、`ftp` コマンドなどによって転送する場合、バイナリモードを用いると余分な復改コード `^M` がファイルに含まれる。S-PLUS で、このようなファイルを読もうとすると、エラーが生じる。つぎのように `nkf` という UNIX ユティリティを用いると、`file1` から余分なコードを除いたものが、`file2` となる。

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

15.2 データフレームの作成

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

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

これを読み込むためには、次のように実行する。

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

この命令を実行すると、`seisekia1` という名のオブジェクトにデータが保存される。これはデータフレームと呼ばれる型のオブジェクトであり、いくつかのベクトルや行列を要素として含む。この場合には、`name`、`eng`、`math`、`jpn` の 4 つのベクトルが要素である。`row.names=NULL` は、行の名前を数字にする指定である。文字型のベクトルを指定すると (重複がなく、各要素が一意的であるもの)、それが行の名前になる。また特に指定しないと、文字型で重複のない変数の内、最初に現れたものが行の名前になる。`col.names` には列 (変数) の名前を指定する。また、引数に `header=T` を指定すると、データの先頭行を列の名前として入力し、2 行目以降をベクトルの要素とする。

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

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

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

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

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

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

16 基礎統計機能

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

16.1 1 変数の統計量

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

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

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

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

16.2 分布の比較

2群のデータにおける統計量(平均、分散など)を比較することは、最も基本的な統計的な分析の1つである。ここでは、Cleveland(1993)にとりあげられている。原典はFrisby & Clatworthy (1975)によるランダムドットステレオグラムの反応時間の測定実験である。StatlibのData and story library (DASL)より入手したものを/home1/otsu/splus/fusiontime.datにおいてある。

16.2.1 データフレームの作成

まず作業を行なうディレクトリにcdコマンドを用いて移動し、次のようにして、データをコピーする。

```
%xz0000 cp /home1/otsu/splus/fusiontime.dat .
```

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

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

このデータを入力するには、S-PLUSをこのディレクトリにおいて起動し次のように指示する。

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

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

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

⁵ M.FriendlyによるDASLのページにはデータ件数は81となっているが、実際に表示されているデータは78件であった。

16.2.2 枝葉図

分布の特徴を簡単に把握するための関数として `stem` が用意されている。これは Tukey(1977) において枝葉図 (stem-and-leaf plot) と名付けられているグラフである。図 2 では、教示が "NV" であるものについて枝

```
> stem(fusionNV)

N = 43   Median = 6.9
Quartiles = 3.1, 10.3

Decimal point is at the colon

 1 : 779
 2 : 0113347
 3 : 1149
 4 : 2377
 5 : 6
 6 : 139
 7 : 89
 8 : 1499
 9 : 1577
10 : 3
11 :
12 : 23
13 : 04
14 : 7
15 :
16 :
17 : 4
18 :
19 : 7
20 : 4
21 :
22 : 0

High: 47.2
```

図 2: 枝葉図

葉図を描いている。ここで、出力されているのは簡単なヒストグラムである。反応時間の整数部分が図の左側の見出しに描かれており、範囲に該当するデータの個数分の文字が、その行に並べて表示される。表示されるのは小数点 1 桁目の数値である。ほとんどの反応時間は 10 秒以下であるが、中には 47.2 秒かかるものもあり、正方向に裾が重いことが分かる。教示が "VV" であるものについても同様に、`stem(fusionVV)`、または直接 `stem(fusion$time[fusion$inst == "VV"])` とすれば枝葉図が表示される。図の上部に Median と表示されているのは中位数 (メディアン) であり、Quartiles とあるのは、第 1 四分位数と第 3 四分位数である。

16.2.3 ヒストグラム

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

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

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

```
> X11()  
> par(mfrow=c(1,2))  
> hist(fusionNV,xlim=c(0,50),ylim=c(0,30),breaks=5*(0:10))  
> hist(fusionVV,xlim=c(0,50),ylim=c(0,30),breaks=5*(0:10))
```

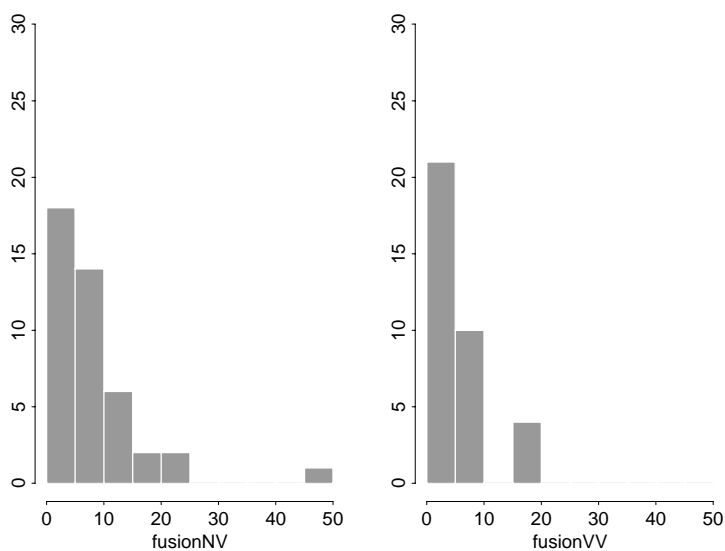


図 3: ヒストグラム

16.2.4 箱ヒゲ図

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

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

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

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

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

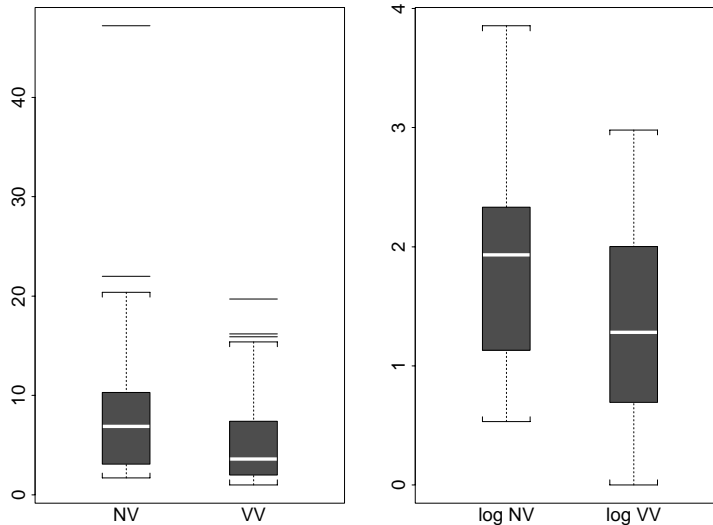


図 4: 両眼融合データの箱ヒゲ図 (左側: 無変換、右側: 対数)

16.2.5 分位点プロット

グラフによって分布の形を検討するための、もう一つの道具が分位点プロット (quantile plot) である。分位点プロットのなかでしばしば利用されるものには、分布の正規性を検討するための正規確率プロット (normal-probability plot) と、2 つの分布の関係をみるための分位点 - 分位点プロット (quantile-quantile plot/ Q-Q plot) とがある。

図 5 の上 2 つは、fusion データ (NV 条件) の反応時間を正規確率プロットによって表示したものである。左は無変換のデータであり、右は対数変換後のデータを示している。正規確率プロットは、データの各点について対応する下側確率を求め、その確率に対応する正規分布の分位点を横軸の値とし、データの値を

縦軸にプロットする。もし、データの分布が正規分布に従っているならば、プロットされた点はほぼ直線上に並ぶはずである。左右の端で傾きが急になっているならば、これは正規分布に比べて裾の重い分布 (自由度の小さい t 分布がこれにあたる) であることを意味する。逆に傾きが緩やかなら、裾のつまった分布 (一様分布など) であることを示す。正規確率プロットに示した直線は関数 `qqline` で表示したものであり、これは第 1 四分位点と第 3 四分位点を結んでいる。変換前のデータはかなり正の方向に裾が重くなっているが、対数変換すると正規分布に近くなっている。ただし、小さい方の分布の裾が短い。

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

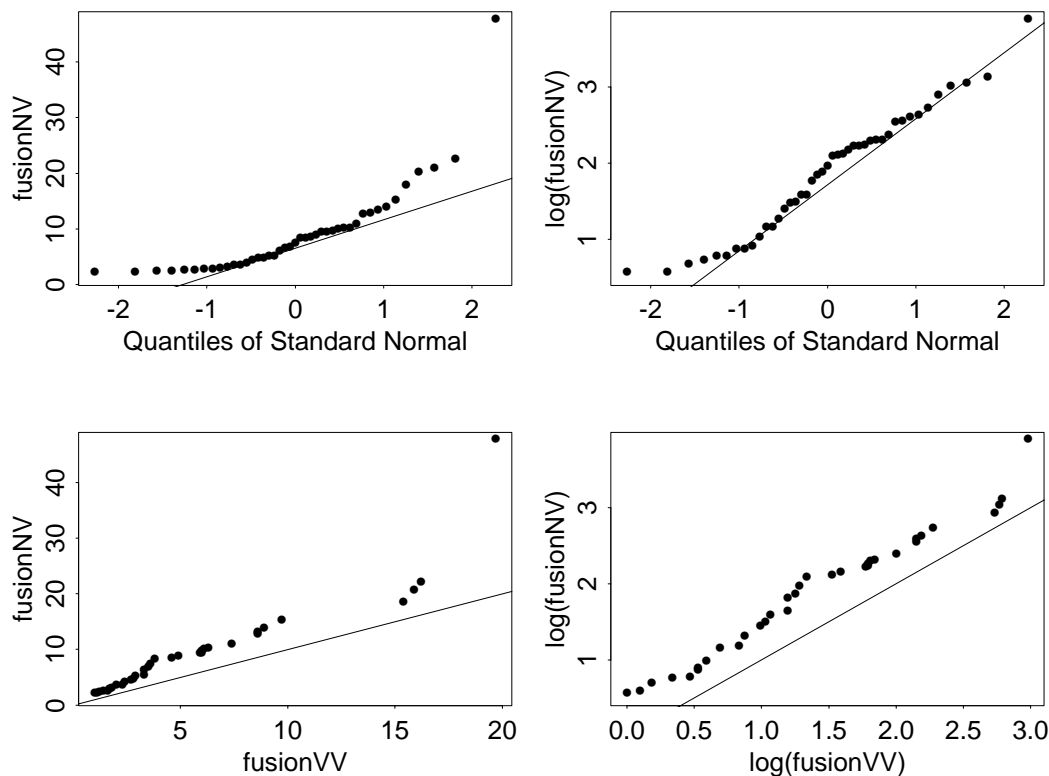


図 5: 正規確率プロットと分位点・分位点プロット

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

分位点 - 分位点プロットを表示するには、`qqplot` を用いる。また、資料の図 5 下段に表示してある直線は $Y = X$ であるが、これを表示するには `abline(0,1)` とする。`abline(a,b)` を実行すると直線 $Y = a + bX$ を画面に追加される。

```

> X11()
X11() # 画面の初期化を行なう。実行済みなら不要。
> par(mfrow=c(2,2)) # 2 × 2 の表示を行なう。
> qqnorm(fusionNV) # 正規確率プロット
> qqline(fusionNV) # 25%点と75%点を結ぶ
NULL # qqline の戻り値 (無視してよい)
> qqnorm(log(fusionNV))
> qqline(log(fusionNV))
NULL
> qqplot(fusionVV,fusionNV) #分位点-分位点プロット
> abline(0,1) # Y=X の直線を描き込む
> qqplot(log(fusionVV),log(fusionNV)) # 対数変換して表示
> abline(0,1)

```

16.2.6 位置の差の検定

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

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

```

> t.test(fusionNV,fusionVV)

Standard Two-Sample t-Test

data: fusionNV and fusionVV
t = 1.9395, df = 76, p-value = 0.0562 # 5%水準では有意にはならない。
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.0809383  6.0990099
sample estimates:
mean of x mean of y
 8.560465  5.551429

```

```
> t.test(fusionNV,fusionVV,var.equal=F) # Welchの検定
```

```
Welch Modified Two-Sample t-Test
```

```
data: fusionNV and fusionVV
```

```
t = 2.0384, df = 70.039, p-value = 0.0453 # 今度は有意
```

```
alternative hypothesis: true difference in means is not equal to 0
```

```
95 percent confidence interval:
```

```
0.06493122 5.95314037
```

```
sample estimates:
```

```
mean of x mean of y
```

```
8.560465 5.551429
```

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

```
Standard Two-Sample t-Test
```

```
data: log(fusionNV) and log(fusionVV)
```

```
t = 2.319, df = 76, p-value = 0.0231 # 等分散の仮定のもとでも有意
```

```
alternative hypothesis: true difference in means is not equal to 0
```

```
95 percent confidence interval:
```

```
0.06077189 0.80034137
```

```
sample estimates:
```

```
mean of x mean of y
```

```
1.820011 1.389454
```

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

```

> wilcox.test(fusionNV,fusionVV)

      Wilcoxon rank-sum test

data: fusionNV and fusionVV
rank-sum normal statistic with correction Z = 2.2107, p-value = 0.0271

alternative hypothesis: true mu is not equal to 0

Warning messages:
  cannot compute exact p-value with ties in: wil.rank.sum(x, y, alternative,
    exact, correct)

```

16.3 2変数の関連性

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

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

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

```

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

```

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

16.4 データセット longley

S-PLUS には各種の関数の他に、サンプルデータが始めから用意されている。longley という名のオブジェクトは、サンプルデータの一つであり、J.W.Longley [13] による被雇用者数と GNP 等のデータであ

る。内容についての解説は `help(longley)` でみることができる。`longley.y` は 1947 年から 1962 年までの被雇用者数であり、`longley.x` は GNP デフレータ等 6 つの指標を含む。

`longley.x` または `longley.y` と S-PLUS のなかで指定するとデータの内容をみることができる。`longley.x` は 6 個の変数 (列) を含む 16 行のデータである。`attributes(longley.x)` と指定すると、データのサイズ、行や列の名前などが表示される。

16.5 相関係数の検定

関数 `cor` は相関係数 (ピアソン積率相関係数)

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

を与える。また `cor` の引数が行列であると、各列を変数とみなし、それらの間の相関係数からなる行列 (相関行列) をつくり出す。上に述べた関数 `var` も、引数が行列の場合には分散共分散行列が結果となる。

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

```
cor(longley.y, longley.x[,1]) # 被雇用者数と GNP(longley.x の 1 列目)
                               # の相関
cor(cbind(longley.x, longley.y)) # 7 × 7 の相関行列。両者を cbind であ
                               # わせ 7 列をもつ行列を引数とする。
var(longley.x, longley.y)      # 6 × 1 の共分散行列
```

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

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

Pearson's product-moment correlation

data: longley.x[, 1] and longley.x[, 2]
t = 28.6668, df = 14, p-value = 0 # 検定のための統計量
alternative hypothesis: true coef is not equal to 0
sample estimates:
      cor
0.9915892 # これが積率相関係数の値
```

16.6 Spearman の順位相関係数と Kendall の τ (タウ)

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

Spearman の順位相関係数は、観測された 2 変量 $x_i, y_i, (i = 1, \dots, N)$ をそれぞれ変数ごとに順位に変換した上で Pearson 積率相関係数を計算する。また、Kendall の τ は同一変数でのサンプル間の $N(N-1)/2$ 個の対、 $(x_i, x_j), (y_i, y_j), (i < j)$ を考え、これらの 2 つの対が同順であるものと逆順であるものの個数を数

える。順対の個数を C とし、逆対の個数を D とおく。ここで、もし同順位があればそれらはいずれにも数えない。 $\{x_i\}$ における同順位の対の個数を T_x 、 $\{y_i\}$ における同順位の対の個数を T_y とする。Kendall の τ (より正確には τ_b) はつぎの式で定義される。

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

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

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

```
Spearman's rank correlation
```

```
data: longley.x[, 1] and longley.x[, 2]
```

```
normal-z = 3.8559, p-value = 1e-04 # 正規近似値と p 値
```

```
alternative hypothesis: true rho is not equal to 0
```

```
sample estimates:
```

```
rho
```

```
0.9970588 # Spearman の 順位相関係数
```

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

```
Kendall's rank correlation tau
```

```
data: longley.x[, 1] and longley.x[, 2]
```

```
normal-z = 5.3127, p-value = 0
```

```
alternative hypothesis: true tau is not equal to 0
```

```
sample estimates:
```

```
tau
```

```
0.9833333 # Kendall のタウ
```

16.7 2 値変数の関連性

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

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

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

```
Pearson's chi-square test with Yates' continuity correction
```

```
data: rbind(c(189, 10845), c(104, 10933))
```

```
X-squared = 24.4291, df = 1, p-value = 0
```

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

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

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

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

```
Fisher's exact test
```

```
data: rbind(c(3, 1), c(1, 3))
```

```
p-value = 0.4857
```

```
alternative hypothesis: two.sided
```

17 確率分布と乱数

17.1 2項分布

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

$$\Pr(X = m) = \binom{n}{m} p^m (1-p)^{n-m}$$

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

```

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

```

`pbinom` は累積確率 $\Pr(X \leq m)$ を与える。

```

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

```

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

```

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

```

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

課題

n と p をいろいろに変化させて、確率分布の形を確認する。

17.2 2 項分布乱数

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

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

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

課題

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

17.3 ポアソン (Poisson) 分布

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

```
> a _ 0:10 # a に 0~10 の数列を代入
> plot(a,dbinom(a,10,0.3)) # Bin(10,0.3) i.e. n=10,p=0.3 の 2 項分布
> plot(a,dbinom(a,100,0.03)) # Bin(100,0.03) n x p は同じ
> plot(a,dbinom(a,200,0.015)) # Bin(200,0.015)
```

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

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

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

課題

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

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

17.4 正規分布

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

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

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

正規分布 $N(\mu, \sigma^2)$ の確率密度関数は

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

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

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

課題

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

17.5 一様分布

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

課題

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

18 統計モデルの推定

18.1 ロジスティック回帰

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

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

$$E(Y|\mathbf{X}) = \mu_X = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p \quad (16)$$

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

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

$$\text{logit}(\pi_X) = \log \frac{\pi_X}{1 - \pi_X} = \eta_X = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p \quad (17)$$

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

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

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

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

ここでは、ロジスティック回帰による簡単な分析例を示す。

18.2 多重分割表の入力

表 6 に示す多重分割表データの分析を考える。これは洗剤の利用についての選好データであり、Cox & Snell (1981) の表 L.1 より引用した。オリジナルの分析は Reis & Smith (1963) による。各被験者は、新製品 X を標準品 M と比較し、いずれを好むかを回答する。表中 Y_j は、該当する条件において X を選好した人数であり、 $n_j - Y_j$ 人が M を選好している。

表 6: 洗剤の選好データ (Cox & Snell, 1981, Table L.1)

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

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

```

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

```

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

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

```

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

```

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

```

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

```

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

18.3 ロジスティック回帰の推定

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

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

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

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

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

実行結果の概略は `summary` という関数で表示される。S-PLUS ではカテゴリーの表示が SAS と比べて親切ではなく、若干結果を読みとりづらい。`water1` とあるのは、`water` の S-PLUS によって生成された第1対比を意味する。これは

```
> contrasts(CoxSnellL$water)  
      [,1] [,2]  
Hard    -1  -1  
Medium   1  -1  
Soft     0   2
```

の様に確認することができる。つまり、`water1` は `Medium-Hard` に相当し、また `water2` は `-Hard-Medium+2Soft` に相当する。`temp` と `use` についても同様に確認すると、

```
> contrasts(CoxSnellL$temp)  
      [,1]  
High   -1  
Low     1  
> contrasts(CoxSnellL$use)  
      [,1]  
NonUser -1  
User     1  
>
```

となっていることが分かる。(S 類似のフリーソフト R-1.2.3 では、SAS などと同様のデザイン行列が指定されるようになっている。)

```

> summary(CSL.log1)
Call: glm(formula = yes/total ~ water + temp + use, family = binomial,
  data = CoxSnellL, weights = total)
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.278475 -0.6381947 -0.1765431  0.5601259  1.573786

Coefficients:
                Value Std. Error  t value
(Intercept) -0.03017770  0.06631407  -0.4550723
  water1 -0.02475827  0.07764524  -0.3188640
  water2  0.01529000  0.04553775   0.3357653
    temp  0.12832631  0.06643141   1.9317113
    use -0.28350798  0.06387307  -4.4386155

(Dispersion Parameter for Binomial family taken to be 1 )

Null Deviance: 32.82562 on 11 degrees of freedom

Residual Deviance: 8.227989 on 7 degrees of freedom

Number of Fisher Scoring Iterations: 3

Correlation of Coefficients:
      (Intercept)   water1   water2   temp
water1  0.0034170
water2  0.0379049  0.0073272
temp -0.2700698  -0.0411785 -0.0660249
use  0.0368681  -0.0301243  0.0023325  0.0187261

```

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

```

> anova(CSL.log1)
Analysis of Deviance Table

Binomial model

Response: yes/total

Terms added sequentially (first to last)
      Df Deviance Resid. Df Resid. Dev
NULL                                11  32.82562
water  2  0.39530                9  32.43032
temp   1  4.30992                8  28.12040
use    1 19.89241                7   8.22799

```

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

```

> predict(CSL.log1)
      1      2      3      4      5      6      7
0.3911249 0.1344722 -0.1758911 -0.4325437 0.3416083 0.0849557 -0.2254076
      8      9     10     11     12
-0.4820602 0.4122366 0.155584 -0.1547794 -0.411432
>
> fitted(CSL.log1)
      1      2      3      4      5      6      7
0.5965535 0.5335675 0.4561402 0.3935191 0.5845812 0.5212262 0.4438855
      8      9     10     11     12
0.3817657 0.601624 0.5388177 0.4613822 0.3985688

```

第IV部

作図と印刷

19 S-PLUSによる作図

S-PLUS でグラフィックを出力する場合には、あらかじめ作図機器を指定しなければならない。作図機器は、どのようなデバイスに出力するか(画面に表示する、プリンタへ出力する、等)によってさまざまな指定ができる。この指定は、一度行えば S-PLUS を終了するか、あるいは別の機器を指定しなおすまで有効である。

19.1 画面に表示する

センターの端末パソコンを使用している場合、既に使っているように X11 デバイスまたは motif デバイスが使用できる。

```
> X11()
```

とすれば、グラフィック表示用のウィンドウが開く。同様なものに、`motif` がある。

```
> motif()
```

とすればウィンドウが開く。X11 よりも `motif` の方が機能が高いのでお勧めである、と X11 のオンラインヘルプには書かれている。

また、ネットワーク経由でセンターを利用している場合などで、X Window System 環境が使えない場合には、`printer` デバイスを指定すると、文字だけを用いた疑似グラフィックを表示してくれる。

```
> printer()
```

として使う。

19.2 プリンタへ印刷する

プリンタへ印刷するには、グラフィック用のウィンドウで対話的に印刷を指示するか、PostScript デバイスを指定して PostScript ファイル(後述)を作成し、それをプリンタへ送る。

対話的な方法は次のようなものである。X11 命令で表示されたグラフィック表示用のウィンドウの上部に、`PostScript Print Comannd` と表示されている部分があり、初期状態では `1p` となっている。そこを `1p-dxxxxxxx` などと変更する。ここで、`xxxxxxx` は実習室にある PostScript プリンタの識別名である。この状態で、左上部にある `Print Graph` と表示されている部分をマウスでクリックすると、表示されている画面がプリンタに印刷される。直接印刷せずに PostScript ファイルとして保存する場合には、PostScript Print Command の部分に `cat > ファイル名と指定すればよい`。⁶ また、グラフィックウィンドウのメニューに `Print Orientation` とあるのは、印刷時の用紙の向き指定である。Landscape は用紙を横長の画面として使うことの指定であり、マウスでクリックすると `Portrait`(縦長)に変更できる。TeX に取り込んで利用する場合には、`Portrait` に指定する。

⁶ ただし、`ps.out.0001.ps` などの名称を持つファイルも作成されてしまう。これらのファイルの内容は同一である。また、ファイル名の番号部分は、一意的なファイル名を指定するよう逐次自動的に増加する。

もう一つの方法は、デバイス名を明示的に指定するものである。

```
> postscript("gfile.ps",horizontal=F,xsize=6,ysize=6)
```

上のように指定し、描画関数 (plot など) を実行すると、ファイル `gfile.ps` に PostScript コードが保存される。ここで、`horizontal=F` は、用紙を縦長 (ポートレート) に使うことと指定である。ワードプロセッサ等に出力結果を取り込む場合にはポートレート方向が使いやすい場合が多い。横長 (ランドスケープ) を指定する場合には `horizontal=T` とする。`xsize` と `ysize` は描画領域の大きさの指定 (インチ) である。大きさを特に指定しないと、用紙全体に描画される。作成した PostScript ファイルをプリンタへ送る方法は後述する。

なお、PostScript を作図機器指定した場合、実際にファイル内容が出力されるのは、その後他の作図機器を指定し直すか、`graphics.off()` を入力するか、S-PLUS を終了した時点である。`graphics.off()` は、`X11()` や `motif()` で開いたウィンドウを閉じるためにも使用できる。

19.3 作図機器を変えると図も異なる

作図機器の指定を変更すると、出力するグラフの細かなところが異なることがある。X11 上でグラフを作成して確認し、満足できる結果が得られたので PostScript を指定して印刷してみると、少し違った出力になる。もっとも、グラフの内容 (分析の結果) が異なるわけではなく、図や文字の大きさ、点をプロットするシンボル等が異なるだけであるが。

20 UNIX の印刷環境と PostScript

20.1 PostScript

センターの UNIX 環境では PostScript プリンタを使用している。これは、印刷すべき内容を PostScript 言語という一種のプログラム言語で受け付け、それを解釈して印刷イメージを作成する。PostScript は Adobe Systems という会社が提案した印刷用の命令の体系であり、現在コンピュータで印刷物を作る場合に広く用いられている。文字のみでなく、線画やハーフトーンの表現までの極めて高度な表現能力を持っている。

UNIX ワークステーション上で文書を作成する場合には、 $\text{T}_{\text{E}}\text{X}$ と呼ばれる文書印刷のプログラム (ワープロのようなもの) を利用することができる。 $\text{T}_{\text{E}}\text{X}$ は文書中に印刷方法の命令を埋め込んだファイルから、DVI 形式と呼ばれる印刷命令を生成する。これを `dvi2ps` という命令で PostScript に変換するとプリンタで印刷することができる。 $\text{T}_{\text{E}}\text{X}$ は多くのフリーソフトウェアのマニュアルや数物系の論文を作成するために用いられている。S-PLUS の PostScript 出力は、Encapsulated PostScript ファイル (EPSF) と呼ばれる形式を取っており、 $\text{T}_{\text{E}}\text{X}$ や他のワードプロセッサに取り込みやすくなっている。

PostScript 言語を記述したファイルを普通 PostScript ファイル (以後 PS ファイルと略記) と呼ぶ。PS ファイルはテキストファイルなので、人間も読むことができる。S-PLUS で適当な PS ファイルを出力し、中身をのぞいてみよう。もっとも、大抵の人はその内容を理解することはできないが、それが PS ファイルであるかどうかを見分けることくらいはすぐにはできるようになる。

```
> postscript("gfile.ps",horizontal=F,xsize=6,ysize=6)
> plot(1:10)
> q()

% less gfile.ps
```

等として、PS ファイルが作成されたことを確認してみよう。

PS ファイルは一般にサイズが大きいため、たまって来るとディスク容量をかなり消費する。印刷が終了と多くの場合 PS ファイルは必要なくなるので、その場合には忘れず削除すること。後に再利用する予定があるならば解りやすいファイル名をつけて保存しておこう。

20.2 S-PLUS から以外の印刷にも PS ファイル

テキストファイルを印刷する場合には、

```
% a2ps textfile | lp -dxxxxxxx
```

のような命令を使用するが、実はここでも a2ps 命令でテキストファイルを PostScript 言語に変換してから、プリンタへ送っている。適当なテキストファイルを指定して、

```
% a2ps textfile | less
```

のようにすれば、a2ps 命令でどのような PostScript プログラムが出力されているかを画面上で見ることができる。

また、UNIX 上の多くのプログラムで PS ファイルを作成することができる。

20.3 計算経過の保存

S-PLUS を実行中に

```
> sink("ファイル名")
```

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

```
> sink()
```

と指定すると、画面に表示されるように戻る。ベクトルの内容のみを出力するには

```
> cat(オブジェクト名,file="ファイル名")
```

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

他には、

1. mule のなかで UNIX のシェルプログラムを起動し、そこで S-PLUS を起動する。
2. X Window System のカット・アンド・ペースト機能を用いて、画面に表示されている内容をエディタ (mule) を用いてファイルに保存する。

という2つの方法が考えられる。

ここでは後者について説明する。

1. S-PLUS を起動する前にコマンドモードで `mule &` と命令を入力し、`mule` のウィンドウを起動しておく。このウィンドウにマウスを移動し、左ボタンを押すと、枠の部分が灰色から緑に変更され、命令を入力することができるようになる。
2. `mule` のメニュー `File` を選び、さらに `Open File` を選ぶ。最下段で、ファイル名の入力を待つ状態になるので、ここで計算経過を保存するファイル名を指定する。もし、不適切な操作を行なってしまったなら、`(Ctrl+G)` を押すとほとんどの場合は、文字を入力する通常の状態に戻る。
3. ここで、`mule` は指定されたファイルの編集を行なっている状態になる。
4. 最初のウィンドウにマウスカーソルを戻し、左ボタンを押すと、枠が緑色に戻る。ファイルに保存したい部分の先頭にマウスカーソルを持って行く。そこで左ボタンを押したまま、保存したい部分の最終部分にマウスカーソルを移動する。指定した部分は黒く反転しているはずである。
5. `mule` のウィンドウにマウスカーソルを移動し、左ボタンを押すと `mule` のウィンドウ枠が緑色になる。ここで、テキストを新たに入力するべき位置にキャラクタカーソルを矢印キーを用いて移動する。
6. マウスの左右のボタンを同時に押すと、黒く反転している部分が `mule` のウィンドウの中に、コピーされる。このとき、`mule` が日本語入力を行なう状態であるとうまくゆかないので、英数字入力状態に戻しておく。
7. メニューで `File` を選択し、さらに `Save Buffer` を選択すると内容が指定したファイルに書き込まれる。`(Ctrl+X)` `(Ctrl+S)` と押しても同様に書き込みが行なわれる。
8. メニューで `File` を選択しさらに `Exit Emacs` を選択するか、`(Ctrl+X)` `(Ctrl+C)` と押すと、`mule` が終了する。

これで、指定したファイルに計算経過の内容がテキストとして保存される。前述したように `a2ps` 命令を用いて、この内容を印刷することができる。

20.4 印刷の手続き

PS プリンタへ PS ファイルを送るには、`lp` 命令を用いる。次のようなコマンドを入力することにより、グラフや文書の印刷が可能である。一般的には、他の UNIX や Linux システムでは、`lpr` を用いる場合も多いので、センター以外の環境では操作が異なる。

```
% lp -dxxxxxxx gfile.ps
```

ここで、`gfile.ps` は印刷する PS ファイル名である。`xxxxxxx` はプリンタの名前である。

20.5 印刷イメージを画面で確認する

紙を無駄に消費しないように、実際にプリンタへ出力する前に画面で印刷イメージを確認して、望む出力が本当に得られることを確認するように習慣付けて欲しい。そのためには、`ghostview`(Ghostscript viewer) を利用する。`ghostview` および `gs` は PostScript 言語を解釈してそのイメージを画面や非 PostScript プリンタに出力してくれるプログラムであり、無料で公開されている。UNIX のコマンドモードで、

```
% ghostview gfile.ps &
```

のように使用する。複数ページの出力がある場合にはメニューの `Page` を指定する。プログラムを終了するにはメニューで `File` → `Quit` を指定するか、画面上で `q` と入力する。

21 2次元プロット

21.1 データの表示

2次元の散布図を描くには `plot` を使用する。

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

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

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

4個の点がプロットされる。ここで、`plot(x1, y1)` のかわりに、

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

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

21.2 複数の折れ線の表示

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

1. まず最初に `plot` で1つのグラフをプロットする。つぎに関数 `lines` または `points` を用いて、グラフ上に線分または点を重ね書きする。
2. 関数 `matplot` を利用する。この関数は引数にベクトルだけでなく、行列を指定することができる。`x` が長さ n のベクトルであり、`y` が $n \times m$ の行列であるとする。`matplot(x,y,type="l",lty=1:m)` と指定すると、 m 回 `plot` を繰り返し、重ね書きするのと同様の描画を行なう。`x` にも行列を指定することが可能であり、この場合には `x` の列が順次描画に用いられる。また、描画の線種も各データ毎に変化する (`lty` の指定による)。

`lines`、`points` と同様の機能を持つ `matlines`、`matpoints` も利用できる。描画の詳細を指定のためのオプションである `type`、`lty`、`pch`、`col` などについては、`lines` のオンラインマニュアルを参照せよ。

21.3 棒グラフと円グラフ

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

22 矢印や文字を書き込む

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

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

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

23 3次元以上のプロット

23.1 persp

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

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

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

例えば、つぎのように指定すると平面が表示される。

```
> x2 <- c(1,2,3,4)
> y2 <- c(1,2,3,4)
> z2 <- matrix(c(1,2,3,4,2,3,4,5,3,4,5,6,4,5,6,7),4,4)
> X11() # 指定済みならば不要。
> persp(x2,y2,z2)
```

また、

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

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

23.2 pairs

関数 pairs は多変量のデータを、2変量のペア毎にプロットする。

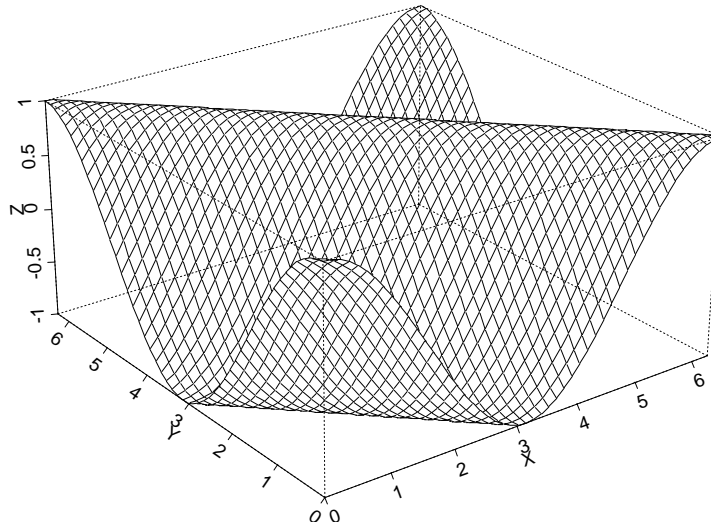


図 6: $\cos(x+y)$ の鳥瞰図

```
pairs(x)
```

x は行列で、各列が変量に対応する。

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

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

```
> pairs(longley.x)
```

23.3 spin

人間は、3次元のプロットを認識できる。しかし、それを画面や紙上に表示するには、3次元空間を2次元平面に投影しなければならない。そうすると、3次元空間の把握が困難になるが、図を回転させる (視点を移動させる) ことにより、配置が認識しやすくなる。

spin は、3変量のデータをプロットし、その図を回転することができる。一般的につぎのように関数を利用する。

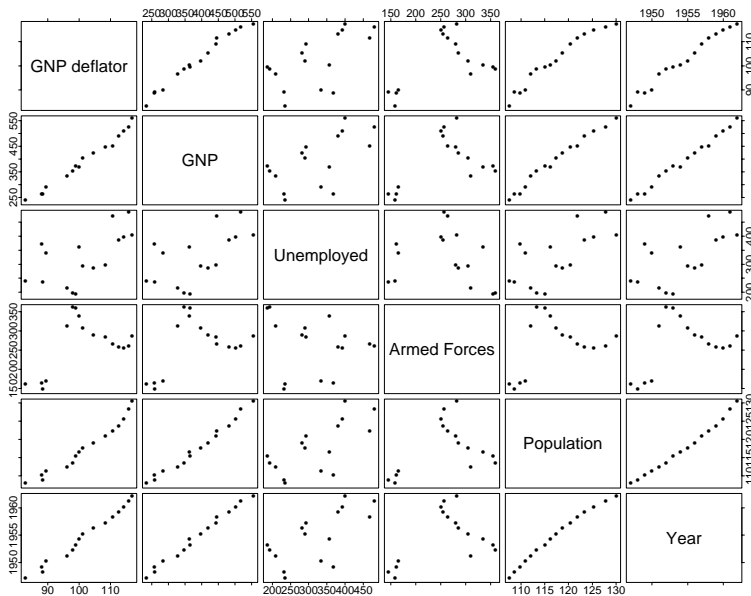


図 7: longley.x の多変量散布図

```
> spin(x)
```

ここで x は行列であり、各列が変量に対応する。上に示したデータを表示するには、

```
> spin(m1)
```

とすればよい。マウスを使って、図を回転させることができる。また、多数の変量のうちどの 3 変量を選んでプロットするかを変えることもできる。

課題

S-PLUS のサンプルデータを用いて、`spin(longley.x)` も試みよ。

23.4 条件付きプロット

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

データフレーム `air` はオゾンについての New York 市で観測された環境データである。`ozone` は地表でのオゾン濃度 (PPM)、`radiation` は日光照射量、`temperaure` は気温 (華氏)、`wind` は風速 (毎時マイル) を示す。

```
> names(air)
[1] "ozone"      "radiation"  "temperature" "wind"      # air の変数名
> dim(air)
[1] 111 4      # 111 サンプル (行)、4 変数 (列) である。
> coplot(ozone ~ radiation|temperature*wind, data=air)
```

分割された画面の各部分は、気温と風速について層別した(サンプルを限定した)データの、オゾン濃度と日照射の関係を示す散布図である。サンプルがどの範囲に限定されているかは、画面の上端と右端のグラフが示している。coplotの最初の引数 `ozone ~ radiation | temperature * wind` は、`ozone` を縦軸に、`radiation` を横軸にした散布図を、`temperature` と `wind` について条件づけて表示することを意味する。

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

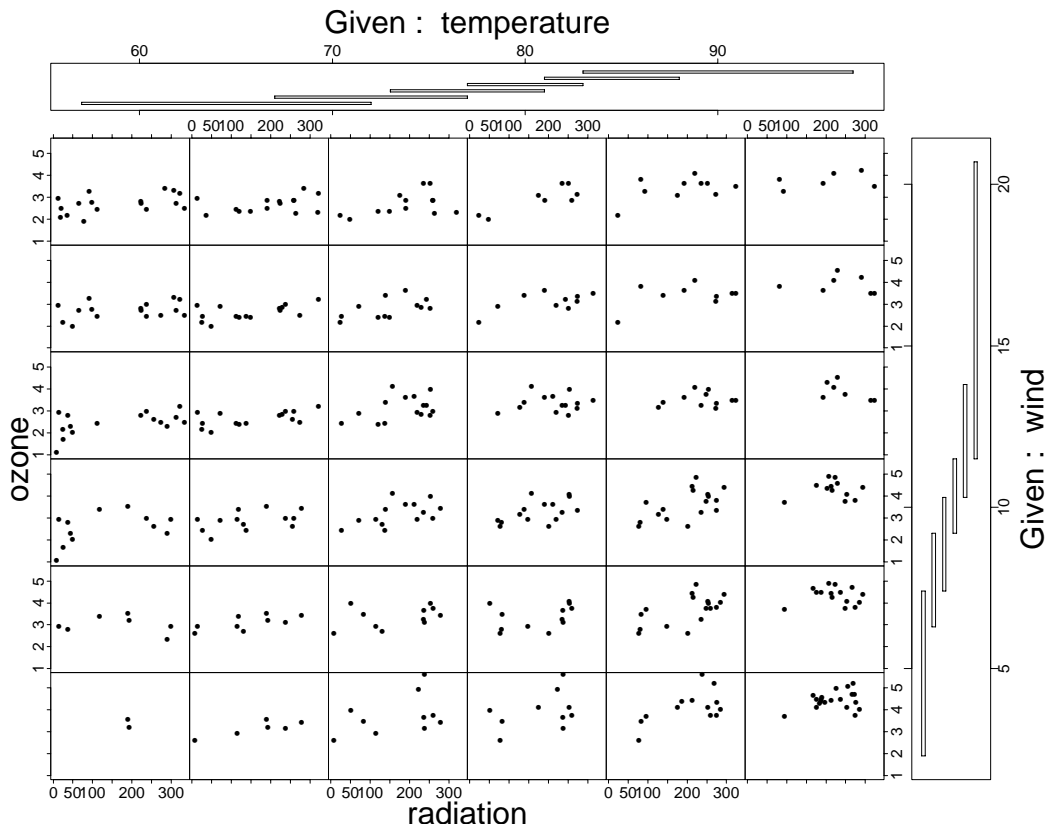


図 8: 条件付きプロット (6 分割)

次は条件の範囲を利用者が指定した例である。given以下が限定する範囲を指定している。関数 `list` は複数のオブジェクトをリストと呼ばれる構造にまとめるためのものである。また、関数 `co.intervals` は指定されたデータから coplot 表示のための範囲を求める関数である。最初の引数に指定されたデータの値の範囲を区分する区間を $k \times 2$ の行列として与える。最初の列は区間の開始点、2列目は区間の終了点である。 k は区間の数であり、`co.intervals` の2番目の引数として与えられる。また、3番目の引数は、隣接する区間が、どれだけ重複するかを指定する。

```
> coplot(ozone ~ radiation | temperature * wind,
  given=list(temperature=co.intervals(air$temperature,4,0.5),
    wind=co.intervals(air$wind,4,0.5)), data=air)
```

210 246 259 217 159 105 55 141 15 MITO 25 31 62 117 162

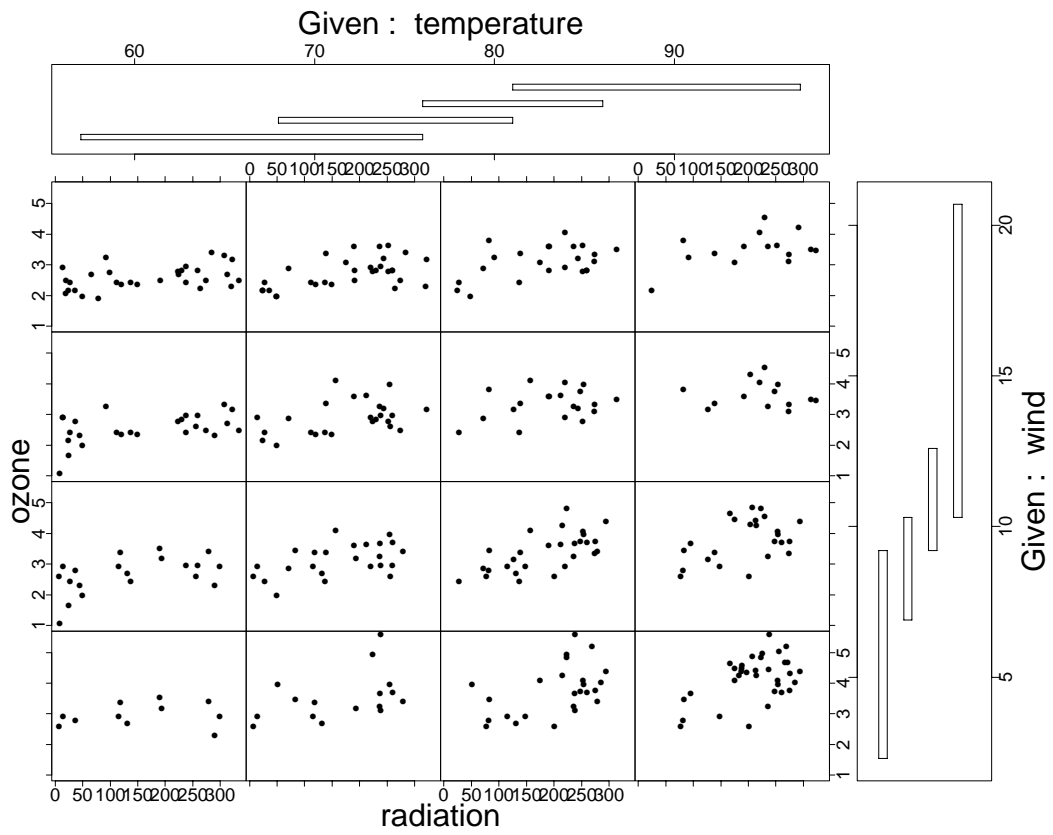


図 9: 条件付きプロット (4 分割)

23.5 XGobi

XGobi は対話的な 3 次元グラフィック表示機能をもつデータ解析のためのソフトウェアである。単体のプログラムとしても利用できるし、S-PLUS から関数として呼び出すことも可能である。まず、S-PLUS から呼び出すためには、次のような手順をとる。

1. S-PLUS を起動する。

2. つぎのように入力する。

```
> source("/home1/otsu/cl14a0/xgobi/Sfunction")
```

これで、関数 `xgobi` が利用可能になる。

3. XGobi を実行するには、

```
> xgobi( 多変量のデータオブジェクト )
```

と指定すればよい。`xgobi(air)` と入力すれば、先の大気データが表示される。

4. XGobi が起動された状態では、2 次元の散布図が表示されている。表示する変数を変更するには、X 軸については、右側の変数名が列挙されている部分にマウスを移動し円の上で左クリックする。Y 軸を変更するには、左右のボタンを同時にクリックする。

5. 3 次元の回転表示を行なうには、画面上側のメニュー部分で `Rotate` を選ぶ。画面が回転している状態で、表示されている青色の部分にマウスカーソルを移動し、左ボタンを押したままの状態でもマウスを移動すると、表示を好みの方向に回転させることができる。

6. メニューから `Identify` を左クリックして選択し、表示データの上にマウスを移動すると、該当するデータの識別記号 (`air` 例では番号) が画面上に表示される。記号が表示されている状態で、左クリックすると記号が画面上に固定して表示される。

7. 終了するには `Exit` を選択する。

8. その他の解説については、`Help` メニューを参照せよ。また、説明を参照したいメニュー上にカーソルを移動し、右クリックすると説明が表示される。また、PostScript ファイルによるマニュアルは、大津の研究室にある。

S-PLUS を用いないで XGobi を起動するには、コマンドモードで

```
%xz0001 xgobi データファイル名
```

と入力すればよい。利用法の概要は `man xgobi` で知ることができる。

つぎのように入力すると、アメリカ合衆国の都市データが表示される。

```
xz0001% xgobi /usr/local/lib/xgobi/data/places
```

ディレクトリ `/usr/local/lib/xgobi/data/` には XGobi 用のサンプルデータが幾つかふくまれている。

同一のデータについて複数の XGobi 画面を表示させることは、S-PLUS の中からでは不可能であり、単体のコマンドとして起動しなければならない。(メニューで `I/O -> Clone XGobi` と指定すると、画面が複数表示され、サンプル毎の色や形の指定が連動する。)

24 グラフ作成の詳細指定

`par` で、グラフ作成のさまざまな指定ができる。例えば、グラフを正方形の領域に描きたければ、`par(pty="s")` とすればよい。また、文字をデフォルトの2倍の大きさに描きたいという場合には、`par(cex=2)` とする。

また、1画面に複数のグラフを表示するには、`mfrow` または `mfcol` を用いる。`par(mfrow=c(2,3))` と指定すると、縦2段横3列にグラフを配置する。描画の順は、上左、上中、上右、下右、下中、下左となる。`mfcol` を指定すると同様に1画面に複数のグラフを配置するが、描画順は、上左、下左、上中、... となる。

`plot` 等に直接指定できるオプションもある。

```
xlim=c(x1,x2), ylim=c(y1,y2)
```

は、x 軸、y 軸の範囲をそれぞれ `[x1,x2]` と `[y1,y2]` に指定するオプションである。

```
> plot( ... , xlim=c(0,10), ylim=c(0,1))
```

のように使用する。

その他に、図の大きさ、目盛の付け方、文字の描画方向、軸のタイトル、等々さまざまなオプションで、きめ細かなグラフ作成指定ができる。詳しくは、`par` のオンラインヘルプか、参考書を見ること。

第 V 部

演習篇

25 基本データ型

現状での殆どのコンピュータでは、つぎのような基本データ型を用いている。

- バイト: 1byte の文字または数値を表す。数値として利用する場合には $0 \sim 255$ または $-128 \sim 127$ のいずれかの範囲である。C および C++ では `char` および `unsigned char` という型が対応する。Java では `byte` がこれにあたる。S-Plus では `character` という名称でバイト文字列を表現している。
- 2バイト整数: 2バイトの領域を用いて整数を表す。C および C++ の `short int` および Java の `short` 型がこれにあたる。古い 16bit パソコン (MS-DOS) の C 言語などでは `int` 型も 2 バイトであった。範囲は $-32,768 \sim 32,767$ 。現在の PC では `int` は 4 バイト整数を表す。
- 4バイト整数: 4バイトの領域を用いる整数。C, C++, Java での `int` 型はこれである。範囲は $-2,147,483,648 \sim 2,147,483,647$ (国家予算は表せない)。
- 4バイト (単精度) 浮動小数点: 4バイトの領域を用いる浮動小数点。表現できる数の絶対値の範囲は $3.4 \times 10^{-38} \sim 3.4 \times 10^{38}$ 。精度は有効数字 7 桁 (電卓より精度は悪い)。C, C++, Java での `float` 型。
- 8バイト (倍精度) 浮動小数点: 8バイトの領域を用いる浮動小数点。表現できる数の絶対値の範囲は $1.7 \times 10^{-308} \sim 1.7 \times 10^{308}$ 。精度は有効数字 15 桁。C, C++, Java での `double` 型。

S-Plus では基本的に数値は倍精度浮動小数点を用いて表されている。整数型に明示的に変換する (他のプログラムと結合する場合などに必要) ためには関数 `as.integer` が用意されている。

26 数値演算上の注意

1. 一般的には整数演算の桁あふれには何の警告も生じない。

```
> as.integer(2000000000)
[1] 2000000000
> as.integer(2000000000)+as.integer(2000000000)
[1] -294967296
>

> as.integer(1.5)
as.integer(1.5)      # 整数型のデータに変換 (単なる切捨てとは少し違う)
[1] 1
> as.integer(2.2)
[1] 2
>
> 2^30
[1] 1073741824
> as.integer(2^30)
```

```
[1] 1073741824      # 正しい
> 2^31
[1] 2147483648
> as.integer(2^31)
[1] 2147483647      # おかしい
```

2. 浮動小数点による少数の表現には，一般的には誤差が生じる．高精度計算を必要とするのでなければ，実用上それほど気にする必要はないが．

```
> options(digits=17)
> 0.1
[1] 0.10000000000000001
> options(digits=7)

> 2^1023
[1] 8.988466e+307
> 2^1024
[1] Inf
> 2^(-1074)
[1] 4.940656e-324
> 2^(-1075)
[1] 0
```

3. 浮動小数点の精度は絶対値について相対的である．

27 判断と分岐

判断と分岐（条件による枝分かれ）については PartI を参照のこと。また S-PLUS にはつぎのような論理値をとるベクトルに対応した関数が用意されている．

```
> a0 <- ifelse(c(T,T,F,F,T,T),"x","y")
> a0
[1] "x" "x" "y" "y" "x" "x"
```

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

28 配列と繰り返し

ベクトルは 1 次元の配列であり，行列は 2 次元の配列である．より一般的な多次元の配列も多くのコンピュータ言語で扱える．

S-PLUS で多次元の配列をつくるには，関数 `array` を用いる．

```
> a1 <- array(1:24,dim=c(4,3,2))
> a1
, , 1
```

```

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

```

```
, , 2
```

```

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

```

```
>
```

```
> a1[1,,]
```

```

      [,1] [,2]
[1,]    1   13
[2,]    5   17
[3,]    9   21

```

```
> a1[, ,1]
```

```

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

```

```
>
```

S-PLUS では最初の次元の添え字が最初に動く (Fortran と同じ)。(C や C++ では最後の次元が最初に動く)。

関数 `apply` は行列だけでなく多次元配列にも適用できる。

```
> apply(a1,3,sum)
```

```
[1] 78 222
```

```
> apply(a1,1,sum)
```

```
[1] 66 72 78 84
```

最初の例では `a[, ,1]` の要素の和と `a[, ,2]` の要素の和を求めベクトルとしている。2 番目の例では `a[1, ,]` の要素和, `a[2, ,]` の要素和, `a[3, ,]` の要素和, `a[4, ,]` の要素和をそれぞれ求めている。

課題

適当な内容をもつ 5×5 の大きさの数値を要素とする行列を作成し, 対角要素を含む下三角部分の和を求めなさい。

ヒント: 一つの方法は `for` の 2 重ループを用いる。

28.1 数列と繰り返し

```
> 1:5          # 1 から 5 までの数列
[1] 1 2 3 4 5
> 5:1          # 逆順
[1] 5 4 3 2 1
> seq(5)       # 1:5 とおなじ
[1] 1 2 3 4 5
> seq(1,5)     # 同上
[1] 1 2 3 4 5
> seq(5,1)     # 5:1 とおなじ
[1] 5 4 3 2 1
> for(i in 1:5) print(i) # i が 1 から 5 まで変化する。毎回 i の値を表示
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
> for(i in 1:5) cat(i)   # 上と同じだが、数値のみ表示
12345>
> # 都市の名前のベクトルをつくる。
> city <- c("Sapporo","Otaru","Hakodate","Asahikawa")
> city
[1] "Sapporo" "Otaru" "Hakodate" "Asahikawa"
> for(k in city) print(k) # k には毎回都市の名前が代入される
[1] "Sapporo"
[1] "Otaru"
[1] "Hakodate"
[1] "Asahikawa"
>
```

28.2 行列をつくる

```
> matrix(1:9,3,3) # 1 から 9 までの数列を 3 × 3 の行列にする。
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
>
> matrix(c(10,20,30,40,50,60),2,3) # 2 × 3 の行列をつくる。
     [,1] [,2] [,3]
[1,]   10   30   50
[2,]   20   40   60
>
```

```

> # 文字を要素とする行列もできる。
> matrix(c("a","b","c","d"),2,2)

      [,1] [,2]
[1,] "a"  "c"
[2,] "b"  "d"
> dim(matrix(c(10,20,30,40,50,60),2,3)) # 行列の次元を求める
[1] 2 3 # 最初が行の大きさ、2番目が列の大きさ
>
> a <- matrix(c(2,3,4,5,6,1),2,3)
> a
      [,1] [,2] [,3]
[1,]    2    4    6
[2,]    3    5    1
> dim(a)
[1] 2 3
>

```

28.3 ベクトルの一部を取り出す

```

> v1 <- c(10,20,30,40,50) # v1 を定義
> v1
[1] 10 20 30 40 50
> v1[1] # 1番目の要素
[1] 10
> v1[c(1,2)] # 1番目の要素と2番目の要素をとりだす。
[1] 10 20
> v1[3:5] # 3番目から5番目の要素をとりだす。
[1] 30 40 50
> v1[c(T,F,T,F,T)] # 真値をもつ部分のみをとりだす。
[1] 10 30 50
> v1[1:5 < 3] # 1から5の数値が3より小さい部分(つまり1と2)を選択
[1] 10 20

```

28.4 問題の解答例と解説

```

> b <- matrix(1:25,5,5) # 5 x 5の行列をつくる
> b
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25

```

```

> # 以下に指定する部分を足し合わせたい。
> b[1,1:1]      # 1行目
[1] 1
> b[2,1:2]      # 2行目 (以下同)
[1] 2 7
> b[3,1:3]
[1] 3 8 13
> b[4,1:4]
[1] 4 9 14 19
> b[5,1:5]
[1] 5 10 15 20 25
>
> sum(b[5,1:5]) # sumは要素の合計値を求める
[1] 75
>
> b0 <- 0      # b0をゼロに設定
> b0
[1] 0
> for(i in 1:5) b0 <- b0 + sum(b[i,1:i]) # 上の部分を足し合わせる
> b0
[1] 155      # これが下三角部分の和
>
> # 途中経過を確認
> b0 <- 0
> for(i in 1:5) {b0 <- b0 + sum(b[i,1:i]);print(b0);}
[1] 1
[1] 10
[1] 34
[1] 80
[1] 155

```

追加課題:

- 行列があたえられたとき行を逆順にしない。

この行列を

```

      [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25

```

つぎのようになる。

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	5	10	15	20	25
[2,]	4	9	14	19	24
[3,]	3	8	13	18	23
[4,]	2	7	12	17	22
[5,]	1	6	11	16	21

- 行列の上三角部分 (対角部分を含まない) の和を求めなさい。

29 リスト構造

資料の前半で説明しなかったものの一つに、リスト (List) と呼ばれるデータ構造がある。ベクトルや配列と同様に、多くの要素を列挙するのに利用される。ベクトルなどと異なり、各要素が別の型を持っていてもよい。また、リストの要素自体がまたリストであることを許すので、どんどん構造を複雑にすることができる。

```
> b <- list(1,2.2,"ccc")
> b
[[1]]
[1] 1

[[2]]
[1] 2.2

[[3]]
[1] "ccc"
```

S-PLUS ではリストの要素を参照するのに、記号 `[[i]]` を用いる。また S-PLUS ではリストを要素名付きにすることができる。要素名を `$` の後に指定することによって、リストの要素を参照できる。

```
> b1 <- list(height=163,weight=65.2, address="Sapporo")
> b1
$height
[1] 163
$weight
[1] 65.2
$address
[1] "Sapporo"
> b1[[1]]
[1] 163
> b1$height
[1] 163
>
```

リストの要素はベクトルや行列、配列、リストであっても良い。

```
> b2 <- list(c(1,2,3),matrix(c(10,20,30,40),2,2),
```

```

+ list("a", "bb", "ccc"))
> b2
[[1]]
[1] 1 2 3

[[2]]
      [,1] [,2]
[1,]   10   30
[2,]   20   40

[[3]]
[[3]][[1]]
[1] "a"

[[3]][[2]]
[1] "bb"

[[3]][[3]]
[1] "ccc"
>
> b2[[1]][1]    #リストの1番目の要素(ベクトル)の第1要素
[1] 1
> b2[[2]][2,2] #リストの第2要素(行列)の(2,2)要素
[1] 40
> b2[[3]][[3]] #リストの第3要素(リスト)の第3要素
[1] "ccc"
>

```

リストの各要素に関数を適用するには `lapply` を用いる .

```

> b3 <- list(c(1,2),c(3,4,5),c(6,7,8,9))
> lapply(b3,sum)
[[1]]
[1] 3      #第1要素のベクトルに sum を適用

[[2]]
[1] 12     #第2要素のベクトルに適用

[[3]]
[1] 30     #第3要素のベクトルに適用

```

各セッションで試行数の異なる実験結果などを容易に表現できる .

課題

サザエさんの家族関係を、リスト構造を用いて表現してみなさい。方法は幾通りも考え得る。例をあげると、個人を一つのリストとして表現し、関係(父、妻など)を指定する要素を定義する。もう一つは家族を一つのリストとして表現し、その中での役割(世帯主、妻...)を要素として表現する。また、一つの婚姻をリストと

して表現し、それとの関係で個人を関係づけることもできる。

29.1 サザエさん家族表現

下のは1例。ファイル/home1/otsu/cl01k0/zyoho99/sazae.splusにコードがある。

```
> source("/home1/otsu/cl01k0/zyoho99/sazae.splus")
```

で以下の命令を実行し、各データが登録される。

```
namihei <- list(gender="male", spouse="fune")
```

```
fune <- list(gender="female", spouse="namihei")
```

```
sazae <- list(gender="female", spouse="masuo",  
             father="namihei", mother="fune")
```

```
masuo <- list(gender="male", spouse="sazae")
```

```
katuo <- list(gender="male", father="namihei", mother="fune")
```

```
wakame <- list(gender="female", father="namihei", mother="fune")
```

```
tara <- list(gender="male", father="masuo", mother="sazae")
```

```
isonofamily <- c("namihei", "fune", "sazae", "masuo", "katuo",  
                "wakame", "tara")
```

家族の配偶者を表示する。

```
> namihei$spouse # 波平の配偶者
```

```
[1] "fune"
```

```
> sazae$spouse # サザエの配偶者
```

```
[1] "masuo"
```

```
>
```

```
> namihei # 波平の属性表示
```

```
$gender:
```

```
[1] "male"
```

```
~M
```

```
$spouse:
```

```
[1] "fune"
```

```
>
```

```
> isonofamily[1] # 波平のオブジェクト名が含まれている。
```

```
[1] "namihei"
```

```
>
```

```
> # 文字列に get を適用すると、その名のオブジェクトを参照
```

```
> get("namihei")
```

```

$gender:
[1] "male"

$spouse:
[1] "fune"
>
> # isonofamily[1] は"naihei"であるので、波平の属性を表示
> get(isonofamily[1])
$gender:
[1] "male"

$spouse:
[1] "fune"
>
> # 文字列を指定してその名のオブジェクトに代入するときには
> # assign を使う。
>

```

課題: 磯野家の各人の配偶者を列挙して表示するプログラムを書け。
 ヒント `for(x in isonofamily)`

29.2 関数の定義

ベクトルの和と2乗和を長さ2のベクトルとして返す関数をつくる。

```

sumandsqsum <- function(x) { c(sum(x), sum(x*x)) }
> sumandsqsum(c(1,2,3))
[1] 6 14      #ここで 6=1+2+3; 14=1*1 + 2*2 + 3*3
>

```

課題: 個人(オブジェクト)を指定すると、その父母名を文字列ベクトルとして返す関数を書け。

課題: 個人(オブジェクト)を指定すると、その祖父母名を文字列ベクトルとして返す関数を書け。

29.3 課題の回答例

1. 両親を表示する。

```

> parents <- function(x) { c(x$father, x$mother) }
> parents(sazae)
[1] "namihei" "fune"
> parents(tara)
[1] "masuo" "sazae"

```

2. 祖父母を表示する。

```

> grandparents <- function(x) {

```

```

+ c(parents(get(x$father)), parents(get(x$mother)))}
>
> grandparents(tara)
[1] "namihei" "fune" # masuo の父母は未定義なので表示されない。

```

29.4 関数についての追加説明

関数の中で定義される（新たに作成される）オブジェクトは永続的なものではない。関数の実行終了後に値は残らない。また、もともと同名前のオブジェクトがあったとしても別のものとみなされる。

```

> y <- c(10,20,30,40) # y を作成しておく。
> # rev はベクトルの要素を逆順にする関数
> xAndRevx <- function(x) { y <- rev(x); c(x,y); }
>
> xAndRevx(c(1,2,3)) #関数を実行。関数の内部で y (別物) への代入。
[1] 1 2 3 3 2 1
> y
[1] 10 20 30 40 # y は変わらない。
>

```

ただし、関数中で新たに同名のオブジェクトが定義されていなければ、始めから定義されているオブジェクトが参照される。以下で定義している関数 `printy` ではオブジェクト `y` を参照しているが、関数内部で定義されていないので、始めから存在している `y` が使われる。

```

> printy <- function() { cat("This is y. \n"); print(y); }
> printy()
This is y.
[1] 10 20 30 40

```

30 多重配列の利用

つぎのような 4 重配列データを考える。これは、米国 Maine 州での 1991 年の自動車事故における乗員の負傷データである。このような多重配列データの表現と集計方法を示す。

自動車事故における乗員の負傷				
性別	場所	シートベルト	負傷	
			無し	有り
女性	市街	無し	7287	996
		有り	11587	759
	郊外	無し	3246	973
		有り	6134	757
男性	市街	無し	10381	812
		有り	10969	380
	郊外	無し	6123	1084
		有り	6693	513

Agresti,A. (1996) *An Introduction to Categorical Data Analysis*, Wiley. 表.6.8 より . Source: Dr
Cristanna Cook, Medical Care Development, Augusta,Maine.

関数 array で定義される多重配列は、最初の次元の添字が最初に動く。このため、上述の表の順にデータを指定すると、分類変数 (アイテム) の指定順が表とは逆になる。

多重配列にアイテムやカテゴリーの名前をつけるには、array を用いて配列を作成するときにオプションで指定するか、または配列を作成したあとで、下に示すようにして定義することができる。

```
> tb68 <-array(c(7287, 996, 11587, 759, 3246, 973, 6134, 757, 10381, 812,  
+ 10969, 380, 6123, 1084, 6693, 513), c(2,2,2,2))
```

```
>
```

```
> dimnames(tb68) <- list(Injury=c("no","yes"),SeatBelt=c("No","Yes"),  
+ Location=c("Urban","Rural"), Gender=c("Female","Male"))
```

```
>
```

```
> tb68  
, , Urban, Female  
      No  Yes  
no 7287 11587  
yes 996   759
```

```
, , Rural, Female  
      No  Yes  
no 3246 6134  
yes 973  757
```

```
, , Urban, Male  
      No  Yes  
no 10381 10969  
yes  812  380
```

```
, , Rural, Male  
      No  Yes  
no 6123 6693  
yes 1084 513
```

```
>
```

```
> tb68[,,"Urban","Female"] # カテゴリーを記号で指定できる。
```

```
      No  Yes  
no 7287 11587  
yes 996   759
```

```
>
```

```
> tb68[, ,1,1] # 数字で指定することもできる。
```

```
      No  Yes  
no 7287 11587  
yes 996   759
```

```
>
```

```
> tb68[,,"Urban","Female"] + tb68[,,"Urban","Male"] # 部分の合算
      No   Yes
no 17668 22556
yes  1808  1139
```

```
> apply(tb68,1,sum) # 負傷の有無の合計を求める。
```

```
      no  yes
62420 6274
```

```
> apply(tb68,c(1,2),sum) # 性別、地域を合算。負傷×ベルト着用
```

```
      No   Yes
no 27037 35383
yes  3865  2409
```

課題：男性について、負傷の有無とベルト着用のクロス集計表を求めなさい。

31 乱数とシミュレーション (1)

シミュレーション (simulation) とは元来何物かのふりをするという意味だが、現在では小型の模型やコンピュータを用いての模擬実験のことを意味する場合が多い。コンピュータを用いたシミュレーションには、決定論的なもの (何度実行しても初期条件が同じなら、同一の結果がえられるもの) と確率論的 (統計的, stochastic) なもの (毎回確率的に結果が変動するもの) とに大別される。行動科学や社会科学の研究ではどちらも使われるが、最初の条件が同じでも結果は変動すると考えられるもの多いため、確率論的なシミュレーションが多用される。このような研究をコンピュータなしで行うのは、事実上極めて難しい。ここでは主に確率論的なシミュレーションについて説明する。

31.1 コイン投げ

表と裏とが確率的に現れるコイン投げを、コンピュータを用いてあらわす。S-PLUS には何種類かの乱数 (random number) を生成する関数が準備されている。最も基本的なものは `runif` であり、一様分布 (uniform distribution) に (近似的に) 基づく確率変数を生成する。一様分布とは 0 から 1 までの区間の数がどれも同等の確率で生じる確率分布のことである。

```
> runif(5) # 5 個の一様乱数を生成する。
```

```
[1] 0.5342416 0.3867699 0.4801704 0.3670870 0.7017096
```

```
> runif(5) # 2 回目は異なる値が生成される。
```

```
[1] 0.7121886 0.7853770 0.2701691 0.1714956 0.5935449
```

一様乱数を用いてコイン投げのシミュレーションを行うには、値が 0.5 以上であれば表 (おもて) とし、それより小さければ裏をあらわすことにすればよい。

```
> runif(5) >= 0.5
```

```
[1] F T T T F
```

表の出る回数を求めるには、上の例について次のような方法をとればよい。(1) 関数 `table` を用いる。TRUE の回数が表の回数の合計である。(2) 関数 `sum` を用いる。論理値を持つベクトルについて計算を行うと T は 1, F は 0 として扱われるので、1 の個数が集計される。

```
> table(runif(10)>=0.5)
FALSE TRUE
   6    4
> sum(runif(20)>=0.5)
[1] 10
```

生成される乱数の系列を再現したい場合 (同一の乱数を求める場合) には、関数 `set.seed` を用いる。`set.seed` の引数には 0 から 1000 までの整数を指定する。

```
> set.seed(123) # 乱数系列の指定
> runif(5)
[1] 0.8756982 0.5321866 0.6700785 0.9921576 0.7029280
> set.seed(123)
> runif(5) # 今度も同じ乱数
[1] 0.8756982 0.5321866 0.6700785 0.9921576 0.7029280
```

課題： 確率 0.1 で Daikiti、0.3 で Tyukiti、0.4 で Syokiti、0.15 で Kyo、0.05 で Daikyo と表示するプログラムを書け。

S-PLUS には上のようなコイン投げの乱数 (2 項分布乱数) を生成する関数 `rbinom` が用意されている。次の例は、表がでる確率が 0.5 (第 3 引数) のコインを 10 回投げたときの表の回数を 1 回分示す。

```
> rbinom(1,10,0.5)
[1] 9
```

一般的に「表がでる確率が p のコイン」を m 回投げたときの表の出る回数をあらわす確率変数を 2 項分布 $Bin(m, p)$ で表す。S-PLUS の関数 `binom(n,m,p)` は $Bin(m, p)$ に従う乱数を n 個生成する。

2 項分布についてつぎのような性質が分かっている (初等統計の教科書に書いてある)。

X が $Bin(n, p)$ に従うとすると、 X の平均は np であり、分散は $np(1-p)$ である。

また、つぎのような確率変数の一般的な性質が知られている (これも初等統計の教科書に書いてある)。

1. 確率変数 X の平均が a であり、分散が b とする。 c を定数とすると cX の平均は ca であり、分散は c^2b である。
2. 確率変数 X の平均が a_X であり、分散が b_X であるとする。また、 Y の平均が a_Y であり、分散が b_Y であるとする。 $X+Y$ の平均は a_X+a_Y である。もし、 X と Y とが統計的に独立 (independent) であれば $X+Y$ の分散は b_X+b_Y である。
3. 確率変数 X_1, \dots, X_n の各々の平均と分散が a と b であるとする。 $Y = (X_1 + \dots + X_n)/n$ (n 個の平均) の平均は a である。また、各々の確率変数が互いに独立であるなら、 Y の分散は $nb/n^2 = b/n$ である。

上の最初の性質 1 を使うと、 m 回コインを投げた時の表の出る回数の比率 ($X \sim Bin(m, p)$ のとき X/m である) の平均は p 、分散は $p(1-p)/m$ となることが分かる。

実際に 2 項分布に従う乱数を発生し、それらの平均と分散がどのようなものになるか確かめてみる。

```
> x <- rbinom(1000,10,0.5) # Bin(10,0.5) の乱数を 1000 個生成
> mean(x) # 個数が大きければ、ほぼ 5 になる。
[1] 4.942
```

```
> var(x) # 個数が大きければ、ほぼ 10*0.25=2.5 になる。
[1] 2.348985
> table(x) # 分布を確かめる。
 1  2  3  4  5  6  7  8  9 10
11 34 132 211 254 217 89 42 8 2
>
```

31.2 トランプゲーム

トランプは4つのスーツ (spade、heart、diamond、club) があり、各スーツには13枚のカードがある。この組み合わせの中から5枚カードをとると、どのような手札になるだろうか？

まず52枚のカードを1から52までの数で表すことにする。あるカードの番号を k とする。 $k-1$ を13で割って、商が0なら spade(1から13までのカード)、1なら heart(14から26まで)、2なら diamond(27から39まで)、3なら club(40から52まで) とする。また $k-1$ を13で割った剰余に1加えたものをカードの数とする (1はAce、11はJack、12はQueen、13はKing)。

関数 `sample` は、指定されたベクトルの要素を無作為にサンプリングする関数である。特に指定しなければ、重複を許さずにサンプリングを行う。この機能を用いてトランプ手札のシミュレーションを行う。

カードの数とスーツを求める関数を定義する。`%/`は除算の剰余(余り)を求める演算子であり、`%/%`は整数の除算の商を求める演算子である。

```
> cardnumber <- function(k) { (k-1)%%13+1 ; }
> cardsuite <- function(k) { j <- (k-1)%/%13+1;
+ if(j==1) "spade" else if(j==2) "heart"
+ else if(j==3) "diamond" else "club"
+ }
```

スーツを求める関数はつぎのようにもかける。

```
> cardsuite1 <- function(k) { j <- (k-1) %/%13 +1;
+ switch(j,"spade","heart","diamond","club");
+ }
```

ここで `switch` は最初の引数の値が j (自然数) であるときに $j+1$ 番目の引数を値として返す関数である。

```
> mycards <- sample(1:52,5)
> mycards
[1] 40 16 45 9 46
> for(i in mycards) cat(cardsuite(i),cardnumber(i)," ");
club 1 heart 3 club 6 spade 9 club 7 >
```

課題: 5枚の手札がフルハウス (ワンペアとスリーカードの組み合わせ) であるかどうかを確かめる関数を書け。

ヒント: 5枚のカードの `cardnumber` を求めベクトルに代入する。ついで関数 `table` を利用し、頻度が (2,3) または (3,2) になっていればフルハウス。

課題: 5枚の手札を2000回生成し、そのうちフルハウスになった回数確かめなさい。

32 乱数とシミュレーション (2)

32.1 正規乱数

正規分布に従う乱数は `rnorm` で生成することができる。つぎの例は平均ゼロ、分散 1 の正規乱数を 5 個生成する例である。

```
> rnorm(5)
[1] -0.8683298 -0.4512669 -2.5522623  0.1605283 -0.7108070
```

まず、正規乱数をたくさん生成し、基準点を超えるものが実際に何個あるか調べる。以下で用いている関数 `pnorm` は正規分布の確率関数 (下側確率を与える) である。また `qnorm` は、下側確率を引数に指定すると、対応する分位点を与える。

```
> x <- rnorm(1000) # 正規乱数を 1000 個生成
> table(x>1.96)    # 1.96 より大きいものの個数
FALSE TRUE
  975    25      # ちょうど 25 個あった。
> pnorm(1.96)     # 1.96 はほぼ 97.5%点
[1] 0.9750021
> qnorm(0.99)     # 99 %点は 約 2.33
[1] 2.326348
> pnorm(2.33)     # 2.33 はほぼ 99 % (上側 1 %) 点
[1] 0.9900969
> table(x>2.33)   # 2.33 より大きいデータは？
FALSE TRUE
  991     9      # 9 個あった
> X11()
> hist(x)         # ヒストグラムの表示
> qqnorm(x)       # 正規確率プロット
> qqline(x)       # 25 %点と 75 %点を結ぶ
```

32.2 t 検定の検出力

t 検定は、2 群のデータの平均が同一と見做し得るかを検討するための統計的方法である。第 1 群のデータが x_i ($i = 1, \dots, m$) で与えられており、第 2 群のデータが y_i ($i = 1, \dots, n$) であるとする。通常の t 検定が前提にしていることはつぎのようなものである。

1. 各 x_i 、 y_j は独立である。
2. 第 1 群と第 2 群は各々正規分布に従い、それらの分散は同一である。

2 群の等分散性を仮定しない方法は、Welch の検定とよばれる。Welch の検定については、統計の教科書を見よ。関数 `t.test` でもオプションで実行できる。

t 検定の手順はつぎのようなものである。

1. (x_i) の平均 \bar{x} と (y_i) の平均 \bar{y} を求める。

2. 2群共通の分散をつぎの式で求める。

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^m (x_i - \bar{x})^2 + \sum_{i=1}^n (y_i - \bar{y})^2}{m + n - 2} \quad (19)$$

3. t 値をつぎの式で求める。

$$t = \frac{\bar{x} - \bar{y}}{\sqrt{(1/m + 1/n)\hat{\sigma}^2}} \quad (20)$$

4. もし、第1群と第2群の平均が等しければ t 値は自由度 $df = m + n - 2$ の t 分布に従う。 t 値を自由度 $m + n - 2$ の t 分布の分位点と比較し、それより外側であれば (片側検定の場合はより大、またはより小ならば)、帰無仮説を棄却する。

統計検定を考える際、つぎの2つの値、有意水準 (significance level) と検定力 (または検出力) (power of a test) が重要である。検定を行なうときに生じる事態はつぎの4つである。

1. 帰無仮説が正しく、かつ棄却されない。
2. × 帰無仮説が正しいが、誤って棄却される。(有意水準 = 第1種の誤り)
3. × 帰無仮説が成立していないが、帰無仮説が棄却されない。(第2種の誤り)
4. 帰無仮説が成立しておらず、かつ棄却される。(検出力)

をつけたものは望ましい事態であり、×をつけたものは望ましくない。有意水準 (p 値) は2番目の確率を表す値である。つまり、帰無仮説が正しいという仮定のもとで、それが誤って棄却される可能性がどれだけあるかを示す。しかし、これだけでは片手落ちであって、4番目の値にも注意を払う必要がある。4番目の事態が生じる確率のことを検出力という。これがどのような値になるかを、乱数シミュレーションで確かめてみる。

ここで、第1群の母平均 (真の平均) が $\mu_x = 0$ であり、第2群の母平均が $\mu_y = 0.5$ であるとする。サンプル数は $m = n = 5$ とする。このとき、両側5%水準の t 検定を行なうと、実際に仮説が棄却されるのは、どれくらいになるだろうか。

つぎのような関数をつくる。

```
tpower <- function(m,n,dmu,p,k) {  
  count <- 0;  
  for(i in 1:k) { x <- rnorm(m); y <- rnorm(n)+dmu;  
    testxy <- t.test(x,y);  
    if(testxy$p.value < p) count <- count+1;  
  }  
  return(count);  
}
```

ここで、引数 m と n はそれぞれ、第1群と第2群のサンプル数を表す。 dmu は第1群と第2群の平均の差である。 p は t 検定の p 値 (両側確率) を識別するための値であり、 k は t 検定の繰り返し数である。上の関数は指定された回数分、それぞれの群から乱数を生成し、等分散で対応のない場合の t 検定を行なう。 k 回の検定のうち、 p 値が指定された値 p より小さい回数値として返す。

課題1: 関数 `t.test` を用いず、 `mean` と `var` を用いて t 値を計算し、上の `tpower` と同じ働きをする関数を作りなさい。ここで `var(x)` によって得られるのは、つぎの値である。

$$\frac{1}{m-1} \sum_{i=1}^m (x_i - \bar{x})^2 \quad (21)$$

また $qt(p,df)$ によって自由度 df の t 分布の下側確率 p に対応する分位点 ($Pr(t \leq x) = p$ となる x) を求められる。

課題 2: m 、 n 、 $d\mu$ を色々変化させて検出力がどの程度のものであるか検討しなさい。「 t 検定によって有意差がみられなかった」ということと、「2 群の平均に差がない」ということの間にはどのような食い違いがあるだろうか？

33 乱数とシミュレーション (3)

33.1 乱数で面積を求める

乱数 (特に一様乱数) を利用すると、直接には求めることが難しい部分の面積や体積を求めることができる。ここでは、乱数を用いて円周率 π の推定を試みる。

一様乱数 (0 から 1 の値をとる) を 2 個生成する。これらを X, Y とする。ここで $X^2 + Y^2 \leq 1$ である確率は、全体の面積に対する扇形 (4 分の 1 円) の面積の比率になっている。扇形の面積は円の面積 ($1^2 \times \pi$) の 4 分の 1 であるので、この部分に該当する乱数の比率を 4 倍すればその期待値は π になる。乱数の個数が十分に大きければ、かなり正確に π を推定できるはずである。

つぎのような関数を作ってみよう。

```
randompi <- function(n) {  
    sum(runif(n)^2 + runif(n)^2 <= 1)  
}
```

この関数は n 個の一様乱数の対を求め、それらの 2 乗和が 1 より小さいものの個数を与える。引数 n を 10000 に設定し、この関数の値に $4/10000 = 1/2500$ を掛けると π の値に近くなるはずである。

```
> randompi(10000)/2500  
[1] 3.1212  
> randompi(10000)/2500  
[1] 3.1628  
>
```

このようにして得られた値が、どの程度信用できるかは 2 項分布を用いて推定できる。上の `randompi` の値は $Bin(10000, \pi/4)$ の 2 項分布に従う。この分散は $10000\pi(4-\pi)/16$ である。これを 25000 で割った値の分散は $10000\pi(4-\pi)/(16 \times 2500^2)$ である。仮に $\pi = 3.1$ と仮定してみると、この分散は 0.000279 であり、標準偏差は 0.01670329 となる。少数点 2 桁の精度を得るには、標準偏差を 1 桁小さくする必要 (つまり 10 分の 1 にする) があり、このためにはサンプル数を 100 倍しなければならない。このプログラムでは乱数をベクトルに保存しているので $n=1000000$ とおくと、長さ 100 万のベクトルを 2 本確保することになり領域が不足する。大量の乱数を用いた計算を高速に行なう場合には、専用のプログラムを作成するなどの工夫が必要である。

参考: つぎは C++ 言語を用いたプログラムの例である。(SPlus で 100 万個の乱数を生成するのは非効率的である。)

```
#include <iostream.h>  
#include <stdlib.h>  
  
void main(void) {  
    int i,k,n,ise;
```

```

double x,y;
cout << "input generation number\n" ;
cin >> n;
cout << "input random seed\n";
cin >> ise;
srand48(ise);
k=0;
for(i=1;i<=n;i++) {
    x = drand48(); y = drand48();
    if(x*x + y*y <= 1.0) k++;
}
cout << "In the area: " << k << '\n' ;
cout << "Pi estimation " << (4.0* k)/(double) n << '\n' ;
}

```

以下は上のプログラム (randompi.cpp) の実行例である。

```

xz0000% CC -o randompi randompi.cpp # プログラムのコンパイル
xz0000% ./randompi # 実行
input generation number
1000000 # 1000000 回を指定。
input random seed
12345 # 乱数の初期値に使う数の指定
In the area: 784577
Pi estimation 3.13831 # 四捨五入すると 3.14
xz0000% ./randompi
input generation number
1000000
input random seed
567890
In the area: 785774
Pi estimation 3.1431 # 四捨五入すると 3.14

```

33.2 相関を持つ変数 (2 変量正規分布)

正規分布にもとづく 2 つの変数が互いに相関を持つようにするためには、つぎのような手続きを考える。

正規分布 $N(0, 1)$ に独立に従う 2 つの乱数 X と Y とを考える。適当な $[-1, 1]$ 区間の数 ρ を定め、 $Z = \rho X + \sqrt{1 - \rho^2} Y$ を作る。このとき

$$\text{Var}(Z) = \rho^2 \text{Var}(X) + (1 - \rho^2) \text{Var}(Y) = 1 \quad (22)$$

であるので、 Z の分散は 1 である。また、

$$\text{Cov}(X, Z) = \rho \text{Cov}(X, X) = \rho \quad (23)$$

である。これより、 X と Z との相関は

$$r_{XZ} = \frac{\text{Cov}(X, Z)}{\sqrt{\text{Var}(X)\text{Var}(Z)}} = \rho \quad (24)$$

であることが分かる。

実際に相関を持つ乱数を作ってみる。

```
> x <- rnorm(1000)
> y <- rnorm(1000)
> r <- 0.5
> z <- r*x + sqrt(1-r*r)*y
> var(x)          # x の分散
[1] 1.008933
> var(y)          # y の分散
[1] 1.040834
> var(z)          # z の分散
[1] 1.017291
> var(x,y)        # x と y の共分散. ほぼゼロ.
[1] -0.01797576
> var(x,z)        # x と z の共分散. ほぼ 0.5
[1] 0.4888988
> cor(x,z)        # x と z の相関. これも約 0.5
[1] 0.4825755
> X11()
> plot(x,y)       # 散布図を表示してみる.
> plot(x,z)
```

上の X と Z からなる 2 変量の同時分布を、2 変量正規分布と呼ぶ。

では、つぎに 2 変量正規分布に基づくデータから計算された相関係数はどれくらい変動するかをみてみよう。まず、2 変量正規乱数から相関を求める関数をつくる。

```
> normalcor <- function(n,r) {
+   x <- rnorm(n); y<- rnorm(n); z <- r*x + sqrt(1-r*r)*y;
+   cor(x,z);
+ }
```

この関数を多数回実行してみる。

```
rs <- rep(0,200)
> for(i in 1:200) rs[i] <- normalcor(100,0.5) # 200 個の相関係数
> mean(rs)
[1] 0.498998    # 平均は 0.5
> var(rs)
[1] 0.00575466
> sqrt(var(rs)) # 標準偏差は約 0.08
[1] 0.07585948
>
> mean(0.5*log((1+rs)/(1-rs))) # Fisher の z 変換の平均
[1] 0.5529719
> 0.5*log(1.5/0.5)
[1] 0.5493061
```

```
> (100-3)* var( 0.5*log((1+rs)/(1-rs)))
[1] 0.988218
```

最後の部分の計算は、Fisher の z 変換と呼ばれるものである。2 変量正規分布 (母集団の相関が ρ とする) から n 個の 2 変数の組をとりだして相関を計算し、その値を r とする。さらに

$$z = \frac{1}{2} \log \frac{1+r}{1-r} \quad (25)$$

すると、 n が大きいときつぎの式が近似的に成り立つことが知られている。

$$E(z) = \frac{1}{2} \log \frac{1+\rho}{1-\rho}, \quad Var(z) = \frac{1}{n-3} \quad (26)$$

34 Bootstrap 推定

2 変量正規分布の相関係数がどのような平均と分散を持つかは、理論的に知られており、また上のような近似式を利用できる。しかし、より一般的に実際に得られたデータについて、計算された相関係数はどの程度のばらつきを持つものなのだろうか。もし、分布の形が理論的に分かっているものならば、2 変量正規分布の場合のように、その分布に従う 2 変量の乱数を生成して、 r の分布を知ることは可能である。しかし、そうでない場合にはどのように推定すればよいかは、一般的には分からない。

このような場合に対処するための一つの方法がブートストラップ法 (bootstrap method) である。データから得られた統計量 (平均、分散、相関など) がどのような分布をするかをシミュレーションによって知るためには、データを生み出す元になっている母集団からデータを再び得て、それについて統計量を計算し直すことを繰り返せばよい。ところが、実際には母集団から再びデータを得ることは多くの場合不可能である。そこで、実際に得られているデータを母集団に見立て、それから疑似的なサンプリングを行ない、統計量の計算を繰り返すことを考える。データは母集団から生じたものであるから、母集団を近似していると考えられる。そこで、データから疑似的にサンプリングされたデータ (ブートストラップサンプルと呼ばれる) は、母集団からのサンプルに類似した性質を示すはずである。

ブートストラップ推定はつぎのような手順で行なう。

1. データを x_i ($i = 1, \dots, n$) とする。(ベクトルの場合も考える。2 変量分布の場合は、各サンプルは 2 次元のベクトル。)
2. ブートストラップサンプリングの回数 B を決める。
3. データから重複を許して n 個のサンプルをランダムに選び、統計量を求める。(S-PLUS の関数 `sample` でオプション `replace=T` と指定すると重複を許してサンプリングする。)
4. 上の疑似的なサンプリングを B 回繰り返し、統計量の分布を求める。

この手法の発案者の一人である B.Efron らによれば、統計量の分散を求める場合には、100 回以上の繰り返しが必要であり、また更に統計量の信頼区間を求めるには片側 5% 点の推定の場合で、500 ~ 1000 回以上とする必要があると述べている。

```
> xz <- matrix(rnorm(200),100,2) # 100 行 2 列の正規乱数
> dim(xz)
[1] 100 2
> xz[,2] <- 0.5 * xz[,1] + sqrt(1-0.25) * xz[,2]
> var(xz) # 1 列目と 2 列目の母相関は 0.5
```

```

      [,1]      [,2]
[1,] 1.2240338 0.5877325
[2,] 0.5877325 1.1369694
> cor(xz)
      [,1]      [,2]
[1,] 1.0000001 0.4982056 # サンプルの相関は0.498
[2,] 0.4982056 0.9999999
> bootr <- rep(0,300) # 最初にベクトルを確保
> for(i in 1:300) bootr[i] <- cor(xz[sample(100,replace=T),])[1,2]
> mean(bootr)
[1] 0.4935265 # 相関係数のブートストラップ平均は約0.494
> var(bootr)
[1] 0.006422318
> sqrt(var(bootr))
[1] 0.08013937 # 相関係数のブートストラップ標準偏差は約0.08
> mean(0.5*log((1+bootr)/(1-bootr)))
[1] 0.5463519 # Z変換の平均値は約0.55
> var(0.5*log((1+bootr)/(1-bootr)))*(100-3)
[1] 1.118238 # Z変換の分散を(100-3)倍すると、約1.12
# 本当は1であって欲しい。

```

35 グラフの作成

35.1 散布図

散布図を描画するには、x-軸とy-軸を指定するベクトルを引数とし、関数 `plot` を用いる。座標指定の詳細については `help(axis)` をみよ。

```

> x <- rnorm(50)
> y <- 0.5*x + sqrt(0.75)*rnorm(50)
> plot(x,y)

```

プロットの範囲を指定するには、オプションで `xlim,ylim` を指定する。

```

> plot(x,y,xlim=c(-4,4),ylim=c(-3,3))

```

35.2 直線を引く

直線を引くには `plot` につづけて、関数 `abline` を用いる。 `abline(a,b)` と指定すると、直線 $Y = a + bX$ を描画する。オプション `lty` で線種を指定できる。

```

> abline(0,0,lty=1)
> abline(0,0.5,lty=2)
> abline(v=c(-3,0,3),lty=3) # x= -3,0,3 の箇所に垂直な線を引く。

```

`lines` は指定された座標を次々と直線で結ぶ。 `segments` と `arrows` は第1引数と第2引数で指定された点を第3引数と第4引数で指定された対応する点と結ぶことを繰り返す。

```

> plot(x,y)
> lines(x,y) # 座標を次々と直線で結ぶ
> plot(x,y)
> lines(x,y,lty=2) # 線種を変えて描画
> plot(x,y)
> segments(x[1:25],y[1:25],x[26:50],y[26:50])
> plot(x,y)
> arrows(x[1:25],y[1:25],x[26:50],y[26:50])

```

35.3 文字を書き入れる

つぎのようにオプションで `type="n"` と指定すると、必要な範囲の枠のみを描画し、各々のポイントのマークを描画しない。関数 `text` はベクトルで指定された各々の箇所に、3番目の引数で指定された文字列(数値も可)を描画する。`cex` は文字の大きさの指定。

```

> plot(x,y,type="n")
> text(x,y,1:50) # 1 から 50 までの数値を描画
> plot(x,y,type="n")
> text(x[1:4],y[1:4],c("First","Second","Third","Fourth"),cex=4)

```

対話的に場所を指定することも可能である。つぎの関数を実行し、3箇所グラフィック画面をクリックしてみる。関数 `locator` は利用者がクリックした画面の座標を指定した個数分だけ取得する。

```

> plot(0,0,xlim=c(-4,4),ylim=c(-3,3),type="n")
> text(locator(3), c("Here","There","Third"))

```

35.4 複数の変数を同時に描画する

共通の X 座標を持つ、複数の変数の散布図を同時に描画するには、関数 `matplot` を用いる。第2引数には、第1番目の引数の長さと同じ行を持つ行列を指定する。行列の第1列が最初の変数、第2列が2番目の変数として扱われる。

```

> z <- 0.8*x + sqrt(1-0.64)*rnorm(50) # zをつくる。
> plot(x,z) # x との相関は高い
> matplot(x,cbind(y,z))
> matplot(x,cbind(y,z),pch="AB")

> d <- (0:100)*2*pi/100 # 0 から 2Pi まで 101 ポイント
> e <- sin(d) # Sin カーブ
> f <- cos(d) # Cos カーブ
> matplot(d,cbind(e,f)) # 点を描画
> matplot(d,cbind(e,f),type="n") # 枠だけ描画
> matlines(d,cbind(e,f)) # 線を追加
> matpoints(d,cbind(e,f)) # データ点を追加
> matplot(d,cbind(e,f),type="l",lty=c(2,1)) # 同じ図(線種変更)

```

35.5 多角形の描画

```
> plot(c(0,10),c(0,10),type="n") # xの範囲0,10; yの範囲0,10
> polygon(c(2,4,6,4),c(1,1,2,2)) # 多角形を塗りつぶす
> plot(c(0,10),c(0,10),type="n")
> polygon(c(2,4,6,4),c(1,1,2,2),density=0) # 塗りつぶさない
> plot(c(0,10),c(0,10),type="n")
> polygon(c(2,4,6,4),c(1,1,2,2),density=20)
```

35.6 円と楕円の描画

円を描画する関数はS-PLUSにないので、自分でつくる。(x,y)を中心とし半径rの円周を描く。引数のdensとsegは、暗黙の指定値の指定。このような書き方をした引数は、関数の利用時に省略できる。値を指定する場合には、circle(5,5,2,dens=50,seg=32)などのようにする。

```
circle <- function(x,y,r,dens=0,seg=64)
{
  xcir <- x + r * cos(2*pi/seg*(0:seg));
  ycir <- y + r * sin(2*pi/seg*(0:seg));
  polygon(xcir,ycir,density=dens)
}
```

中心が(x,y)で傾いていない楕円の式はつぎのように書ける。

$$(X - x)^2/a^2 + (Y - y)^2/b^2 = 1 \quad (27)$$

これは $Y = y$ の時は、 $X = x \pm a$ を通り、 $X = x$ の時には $Y = y \pm b$ を通る。この楕円を描くにはつぎのような関数をつくれれば良い。

```
ellipse0 <- function(x,y,a,b,dens=0,seg=64)
{
  xcir <- x + a * cos(2*pi/seg*(0:seg));
  ycir <- y + b * sin(2*pi/seg*(0:seg));
  polygon(xcir,ycir,density=dens)
}
```

さらにこの楕円を d 度(ラジアンでは $= d/360 \times 2\pi$)反時計周りに傾けるには、つぎのようにする。

```
ellipse <- function(x,y,a,b,deg,dens=0,seg=64)
{
  theta <- deg*pi/180;
  xcir <- a * cos(2*pi/seg*(0:seg));
  ycir <- b * sin(2*pi/seg*(0:seg));
  xe <- x + cos(theta)*xcir - sin(theta)*ycir;
  ye <- y + sin(theta)*xcir + cos(theta)*ycir;
  polygon(xe,ye,density=dens)
}
```

上の関数を入力し、つぎのように実行してみなさい。

```

> plot(c(0,10),c(0,10),type="n")
> ellipse(5,5,2,1,0)
> ellipse(5,5,2,1,30)
> ellipse(5,5,2,1,60)
> ellipse(5,5,2,1,90)
> ellipse(5,5,2,1,120)
> ellipse(5,5,2,1,150)

```

さらに楕円の式は一般につきのようにも書ける。

$$a(X - x_0)^2 + b(Y - y_0)^2 + 2c(X - x_0)(Y - y_0) = 1 \quad (28)$$

ここで

$$\mathbf{A} = \begin{pmatrix} a & c \\ c & b \end{pmatrix}, \quad \mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \quad (29)$$

とする。ここで $z_1 = X - x_0, z_2 = Y - y_0$ である。

このようにおくと、楕円の式は

$$\mathbf{z}^t \mathbf{A} \mathbf{z} = 1 \quad (30)$$

となる。

これを実行する関数はつきのようにかける (固有値計算を用いているので難しい)。

```

ellipse2 <- function(x,y,A,dens=0,seg=64)
{
  eig <- eigen(A);
  a <- 1/sqrt(eig$values[1]);
  b <- 1/sqrt(eig$values[2]);
  xcir <- a * cos(2*pi/seg*(0:seg));
  ycir <- b * sin(2*pi/seg*(0:seg));
  xy <- eig$vectors %*% rbind(xcir,ycir);
  polygon(x+xy[1,],y+xy[2,],density=dens)
}

```

35.7 2次元正規分布の信頼楕円

ここで2次元の正規分布の信頼区間 (信頼楕円) を描く方法を示す。 n 個の2次元のデータ (X_i, Y_i) , $i = 1, \dots, n$ が与えられているものとする。これらのデータは2次元の正規分布から得られたものと仮定する。ここでつきのように仮定する。

$$a = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{x})^2 \quad (31)$$

$$b = \frac{1}{n-1} \sum_{i=1}^n (Y_i - \bar{y})^2 \quad (32)$$

$$c = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{x})(Y_i - \bar{y}) \quad (33)$$

$$(34)$$

また、つぎのように 2×2 の行列を定める。

$$S = \begin{pmatrix} a & c \\ c & b \end{pmatrix} \quad (35)$$

また

$$d = \begin{pmatrix} X - \bar{x} \\ Y - \bar{y} \end{pmatrix} \quad (36)$$

とする。Hotelling の T^2 統計量の性質を使うと、新たに得られるであろう (X, Y) の α -信頼区間は、つぎの式で表される。ただし p は分布の次元であり、ここでは 2 である。

$$d^T S^{-1} d \leq \frac{(n^2 - 1)p}{n(n - p)} F_{\alpha}(p, n - p) \quad (37)$$

ここで $F_{\alpha}(p, n - p)$ は自由度 $(p, n - p)$ の F 分布の α 分位点である (S-PLUS の関数では `qf` によって求められる)。

これらを用いて 2 次元の信頼区間を求める関数をつくる。

```
cfellipse <- function(xdata, ydata, alpha, dens=0, seg=64)
{
  p<-2;
  ndata <- length(xdata);
  xme <- mean(xdata); yme <- mean(ydata);
  A <- solve(var(cbind(xdata, ydata)));
  fval <- ((ndata*ndata-1)*p)/((ndata-p)*ndata)
          *qf(alpha, p, ndata-p);
  ellipse2(xme, yme, A/fval, dens=dens, seg=seg)
}
```

これを用いて信頼区間を描いてみる。

```
> x <- rnorm(100)
> y <- 0.5*x + sqrt(0.75)*rnorm(100)
> plot(x, y)
> cfellipse(x, y, 0.95)
> cfellipse(x, y, 0.90)
```

36 回帰分析の方法

ここでは、単純な 1 変量回帰分析を S-PLUS で行なう方法を示す。SAS での分析方法は、教科書 (市川他) に記述があるので省略する。

人工データを例にとって説明する。以下で x と y とは

```
> x <- rnorm(50)
> y <- 1 + 0.5*x + sqrt(1-0.5*0.5)*rnorm(50)
```

これによって $E(Y) = 1 + 0.5X$ であり、残差の分散が $\sqrt{0.75}$ のデータが得られる。

このデータを回帰分析によって推定してみよう。2 つの変数の関係を確認するために散布図を表示するには、`x11()` コマンドでグラフィックウインドウを表示したあと、`plot(x, y)` と表示すればよい。

回帰分析を行なうには、関数 `lm` を用いる。

```
> reg1 <- lm(y ~ x) # y を x で回帰する。
> reg1
Call:
lm(formula = y ~ x) # 指定された式 (重回帰もできる)
```

```
Coefficients: # 推定された係数
(Intercept)      x
  1.001176 0.390611
```

```
Degrees of freedom: 50 total; 48 residual
Residual standard error: 0.8189085
```

もし x と y とが一つのデータフレーム `data1` に含まれる変数である場合には、

```
> reg1 <- lm(y ~ x, data=data1)
```

の様に指定する (直接 `data1$y ~ data1$x` と指定することも可)。
追加情報を表示するには、関数 `summary` を用いる。

```
> summary(reg1)
```

```
Call: lm(formula = y ~ x)
```

```
Residuals: # 残差の特徴
  Min      1Q  Median      3Q     Max
-2.526 -0.5218 -0.02381 0.5199 2.231
```

```
Coefficients: # 推定された係数とその標準誤差
              Value Std. Error t value Pr(>|t|)
(Intercept)  1.0012 0.1158      8.6449 0.0000 # 定数項
              x  0.3906 0.1228      3.1798 0.0026 # x の係数
```

```
Residual standard error: 0.8189 on 48 degrees of freedom
Multiple R-Squared: 0.174 # 重相関係数の2乗 (決定係数)
F-statistic: 10.11 on 1 and 48 degrees of freedom, the p-value is
0.002582 # F 統計量 (F 値が大きいと、モデルが説明力を持つこと)
```

```
Correlation of Coefficients:
(Intercept)
x 0.0016 # 推定されたパラメータ間の相関
```

関数 `plot` の引数に回帰分析の結果を指定し、`plot(reg1)` の様に入力すると、関連するグラフ (1. 散布図に回帰直線を引いたもの, 2. 推定値と残差の散布図) を自動的に出力する。

推定された切片と回帰係数を得るには、関数 `coef` を用いる。また、推定値 \hat{y}_i と、残差 $y_i - \hat{y}_i$ は、各々 `fitted.values` および `residuals` によって求められる。

```
> coef(reg1)
(Intercept)      x
```

```
1.001176 0.390611
> coef(reg1)[1]
(Intercept)
1.001176
> coef(reg1)[2]
x
0.390611
> coef(reg1)["(Intercept)"]
(Intercept)
1.001176
> coef(reg1)["x"]
x
0.390611
>
> fitted.values(reg1) # 出力は省略
> residuals(reg1) #
> qqnorm(residuals(reg1)) # 残差の正規性を確認
```

37 分散分析の方法

ここでは SAS と S-PLUS の両者による分散分析の例を示す。

37.1 SAS で分散分析を行なう

SAS で分散分析を行なうには、幾つかのプロシジャが利用できるが、ここでは GLM プロシジャを使う方法を示す。(以下は行動計量学 (1) での実習資料と重複する。)

つぎのような 2 元配置のデータを考える (Dobson, 表 7.4)。

2 元配置のデータ (Dobson 表 7.4)

要因 A	要因 B			
	B ₁		B ₂	
A ₁	6.8	6.6	5.3	6.1
A ₂	7.5	7.4	7.2	6.5
A ₃	7.8	9.1	8.8	9.1

このようなデータはつぎのようにしてファイルに記入する。

```
a1 b1 6.8
a1 b1 6.6
a1 b2 5.3
a1 b2 6.1
a2 b1 7.5
a2 b1 7.4
a2 b2 7.2
a2 b2 6.5
a3 b1 7.8
a3 b1 9.1
a3 b2 8.8
a3 b2 9.1
```

交互作用まで含めたモデルを当てはめる場合には、つぎのような SAS プログラムで分析すればよい。ここで tb7_4.dat は上記のデータを入力したファイル名である。(交互作用を考慮しないモデルの当てはめを行なう場合には、model ステートメントの a*b の部分を除く。

```
options linesize=80;
data tb74;
    infile 'tb7_4.dat';
    input a$ b$ y;
proc print data=tb74;
proc glm data=tb74 ;
    class a b;
    model y = a b a*b /solution ss1 ss2 ss3;
run;
```

A のみを要因とする 1 元配置を行なう場合には、model ステートメントの部分を

```
model y = a /solution ss1 ss2 ss3;
```

のように指定する。

出力についての注意

1. この例では Type I の SS と Type II, Type III の SS とがすべて同一になっているが、これは各セルのデータ数が同一でバランスしているためである。一般的には同じにはならない。
2. Type II の SS は、その要因を削除した場合の残差 2 乗和の増加量である。

3. 重回帰分析の場合には、Type III の SS と Type II の SS は同一のものである。
4. 分散分析 (データの個数がバランスしない場合) には Type II の SS と Type III の SS とは必ずしも一致しない。
5. Type III の SS の意味は、マニュアルの解説によるといささか複雑であり、単に該当する要因を除いた場合の残差 2 乗和の減少分ではない。該当する要因に関わるより高次の交互作用項への影響も考慮した統計量となっている。SAS の GLM プロシジャでは、Type I、II、III、IV の 4 種の検定統計量が出力できるが、これらの用語が一般的にどれだけ用いられているものであるかは、良く分からない。SPSS (別の統計ソフトウェア、大型計算機センターには導入されている。このセンターにはなし) にも、同様のオプションが用意されている。

37.2 S-PLUS による分散分析

以下で利用しているデータファイル `tb7_4.dat` はつぎの箇所にある。

```
/home1/otsu/cl01k0/datalib/tb7_4.dat
```

```
# データの読み込み
```

```
> tb74 <- read.table("tb7_4.dat", col.names=c("A", "B", "Y"))
> tb74
  A B  Y
1 a1 b1 6.8
2 a1 b1 6.6
3 a1 b2 5.3
4 a1 b2 6.1
5 a2 b1 7.5
6 a2 b1 7.4
7 a2 b2 7.2
8 a2 b2 6.5
9 a3 b1 7.8
10 a3 b1 9.1
11 a3 b2 8.8
12 a3 b2 9.1
> is.factor(tb74$A) # 文字データは要因として扱われる。
[1] T
> is.factor(tb74$B)
[1] T
```

数値変数を離散値をとる要因として取扱う場合には、

```
> tb74$A <- as.factor(tb74$A)
```

のように、関数 `as.factor` によって明示的に変数の属性を変更する。(上の例では `read.table` で入力された時点で要因となっているので、変化はしない)。

各要因について水準別の平均をグラフで確認することができる。

```
> X11()
> plot.design(tb74) # 水準毎の平均を表示
> plot.factor(tb74) # 各要因 (離散変数) 毎に水準別の箱ひげ図
```

分散分析を実行し、その結果を保存するには関数 `aov` を用いる。分散分析表を出力するには `summary` を用いる。

```
> tb74.aov <- aov(Y ~ A*B,data=tb74) # 主効果+交互作用
> tb74.aov
Call:
  aov(formula = Y ~ A * B, data = tb74)
```

```
Terms:
          A          B          A:B Residuals
Sum of Squares 12.74000  0.40333  1.20667  1.48000
Deg. of Freedom      2          1          2          6
```

Residual standard error: 0.4966555

Estimated effects are balanced

```
> summary(tb74.aov)
      Df Sum of Sq Mean Sq F Value Pr(F)
A       2  12.74000  6.370000  25.82432 0.0011274
B       1   0.40333  0.403333   1.63514 0.2482245
A:B     2   1.20667  0.603333   2.44595 0.1671644
Residuals 6   1.48000  0.246667
>
```

もし主効果のみのモデルを当てはめるのなら、

```
tb74.aov <- aov(Y ~ A + B,data=tb74)
```

と記述する。また、A のみの主効果を考えるのなら

```
tb74.aov <- aov(Y ~ A,data=tb74)
```

と指定する。

分散分析の結果得られる各水準 (および交互作用項) の対比 (`contrast`) に対応する係数は、関数 `coef` によって表示される。

```
> coef(tb74.aov)
(Intercept)  A1  A2          B A1B      A2B
      7.35 0.475 0.675 -0.1833333 0.1 0.2166667
>
> coef(tb74.aov)["A1"] # 水準名または番号を指定できる。
  A1
0.475
```

対比の内容は計算結果の `contranst`s というリスト要素に含まれている。

```
> tb74.aov$contrasts
```

```
$A:
```

```
  [,1] [,2]
a1  -1  -1
a2   1  -1
a3   0   2
```

```
$B:
```

```
  [,1]
b1  -1
b2   1
```

関数 `lm` の場合と同様に、`plot(tb74.aov)` と指定すると、適切なグラフを表示する。また、関数 `fitted.values`、`residuals` を適用できる。

第 VI 部

文献案内

線形代数学の教科書は多く出版されている。ここでは標準的な教科書を 3 冊あげる [18],[19],[20]。(特に、[18] は簡潔で分かりやすい教科書。[20] はかなり専門的な内容を含んでいる。)また、線形代数の数値計算的側面については [14] をあげておく。原著は 1992 年に第 2 版が出版され、かなり内容が増えている。行列計算だけでなく、数値最適化、スペクトル解析、乱数生成などについての解説と、C 言語によるプログラム例がある。ただし、掲載されている線形計算のプログラムコードの頑健性(ランク落ちへの対処など)は必ずしも高くない。

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

文献 [2] は S-PLUS の原型になった S 言語の最新マニュアルである。現在(1999 年 7 月)センターにある S-Plus が、この機能をすべて実現している訳ではない。旧版については翻訳 [3] があるが、拡張・変更された部分についての記述はない。

S-PLUS の入門書としては、[21],[8] がある。S および S-PLUS で利用可能な各種の統計分析法についての解説書としては、[4],[16] がある。S-PLUS の分析法の特徴ともいえる、データの平滑化と非線形な変換を利用する分析法についての教科書には、[11] がある。

また、米国カーネギー・メロン大学の統計学科の Mike Meyer 氏によって構築された StatLib アーカイブ <http://lib.stat.cmu.edu/> には S 言語で記述された各種の関数が登録されており、オンラインでこれらをコピーして利用することができる。StatLib には S, S-PLUS 用以外にも各種の統計分析用の関数が数多く集められている。また、S と類似の言語仕様を持つフリーソフトウェアである R については、<http://www.r-project.org> より、UNIX(Linux) 用、および Windows 用のソフトウェアをダウンロードできる。また、Statlib および CRAN(The Comprehensive R Archive Network) と呼ばれるサービスサイトからダウンロードできる。国内では会津大学 <ftp://ftp.u-aizu.ac.jp/pub/lang/R/CRAN/contents.html> がミラーサイトを提供している。

参考文献

- [1] Anderson,E. et al. (1995). *LAPACK Users' Guide, 2nd. ed.*. Philadelphia: S.I.A.M.
- [2] Becker,R.A.(1998) *The New S Language: A Programming Environment for Data Analysis and Graphics*, CRC.Pr.
- [3] Becker, Chambers & Wilks (1988). (渋谷・柴田 訳, 1991), *S 言語 I,II*. 東京 : 共立出版.
- [4] Chambers,J.M. & Hastie,T.J. eds. (1992).(柴田里程訳, 1994). *S と統計モデル:データ科学の新しい波*. 東京 : 共立出版.
- [5] Cleveland,W.S. (1993). *Visualizing Data*. Murray Hill, New Jersey: AT&T Bell Laboratories.
- [6] Cox,D.R. & Snell,E.J. (1981). *Applied Statistics: Principles and Examples*, London: Chapman& Hall. 医学統計研究会訳 (1985). *応用統計実践教本*. 東京:MPC.
- [7] Dongarra,J.J. et al. (1979). *LINPACK : Users' Guide*. Philadelphia: S.I.A.M.

- [8] Everitt,B.S. (1994) *A Handbook of Statistical Analyses using S-PLUS*. London: Chapman and Hall.
- [9] Frisby,J.P. & Clatworthy,J.L., (1975). Learning to see complex random-dot stereograms,*Perception*,**4**, 173-178.
- [10] Gnanadesikan,R. (1997). *Method for Statistical Data Analysis of Multivariate Observations 2nd ed.*, Wiley. 邦訳 (初版) 丘本正・磯貝恭史訳. (1979) 統計的多変量データ解析, 東京:日科技連.
- [11] Hastie,T.J. & Tibshirani, R.J. (1990). *Generalized Additive Models*, London : Chapman and Hall.
- [12] Hoaglin,D.C., Mosteller,F. & Tukey,J.W. (1983). *Understanging Robust and Exploratory Data Analysis*. New Yourk: Wiley.
- [13] Longley,J.W.(1967). An appraisal of least-squares programs from the point of view of the user,*Journal of the American Statistical Association*, **62**, 819-841.
- [14] Press,W.H. et al. (1988). (丹慶勝市他訳, 1993) *Numerical Recipes in C:C 言語による数値計算のレシピ*. 東京 : 技術評論社.
- [15] Ries,P.N. & Smith,H. (1963). The use of chi-square for preference testing in multidimensional problems. *Chem. Eng. Progress*,**59**, 39-43.
- [16] Venables,W.N. & Ripley,B.D. (1999) *Modern Applied Statistics with S-Plus, 3rd ed.*, New York: Springer-Verlag.
- [17] Venables,W.N. & Ripley,B.D. (2000) *S Programming*, New York: Springer-Verlag.
- [18] 三宅敏恒 (1991). 入門線形代数学. 東京:培風館.
- [19] 齋藤正彦 (1966) 線型代数入門. 東京:東京大学出版会.
- [20] 佐武一郎 (1974) 線型代数学 (増補改題). 東京:裳華房.
- [21] 渋谷政昭・柴田里程 (1992), S によるデータ解析. 東京 : 共立出版.
- [22] Tukey,J.W. (1977) *Exploratory Data Analysis*. Reading,MA:Addison-Wesley.

謝辞

本稿作成にあたり多くの有益なコメントを寄せてくれた、北海道大学文学部の中島晃氏に感謝します。

索引

`*`, 61
`:`, 61
`:`, 11
2 次関数, 39

`a2ps`, 66
`abline`, 49
`acos`, 30
`anova`, 62
`apply`, 25
`array`, 25
`arrows`, 26, 69
`attach`, 7

`barplot`, 68
冪 (べき) 等, 37
`binomial`, 60
`boxplot`, 47
`breaks`, 47
分位点, 56
分位点プロット, 48
分散分析表, 62

`c`, 6, 21
`cat`, 16
`cbind`, 20
`cex`, 75
`chisq.test`, 54
`col`, 68
`col.name`, 45
`contrasts`, 61
`cor`, 52, 53

代入, 6
DASL, 45
`data.frame`, 43, 59
`dbinom`, 55, 56
`det`, 31
`detach`, 7
データフレーム, 43, 45
`detbyqr`, 33
`diag`, 24, 29
`dpois`, 57

`e`, 34
枝葉図, 46
`editor`, 28
`eigen`, 32, 41
`else`, 15
円周率, 58
EPSF, 65
`exit`, 4
`exp`, 12

F, 14, 22
`factor`, 48
`family`, 60
`fisher.test`, 54
Fisher の正確検定, 54
`fitted`, 63
`for`, 16
`function`, 27

`gaussian`, 61
`ghostscript`, 67
`ghostview`, 67
`glm`, 60
`graphics.off`, 65
`gs`, 67
`gui`, 9
逆行列, 33
行列積, 24
行列式, 31
行編集, 5

箱ヒゲ図, 47
`help`, 8
`help.start`, 9
変更, 22
比較演算子, 15
`hist`, 46
ヒストグラム, 46
表示, 7

`if`, 15
`ifelse`, 16
イメージ, 20

Inf, 10
 intercept, 35
 一覧, 8
 カイ 2 乗検定, 54
 交互作用項, 61
 固有値, 41
 Landscape, 64
 lapply, 26, 83
 less, 9, 66
 lines, 68
 リスト, 36
 lm, 61
 log, 12
 login, 4
 logistic, 59
 logit, 59
 longley, 52, 71
 lp, 66, 67
 lpr, 67
 lp, 64
 lsfit, 34
 lty, 68
 matlines, 68
 matplot, 68
 matpoints, 68
 matrix, 20
 max, 44
 mean, 44
 mfc01, 47, 75
 mfrow, 47, 75
 min, 44
 motif, 26, 64, 65
 mule, 9
 NA, 10
 nchar, 17
 nkf, 42
 objects, 8
 options, 9
 ordered, 60
 pager, 9
 pairs, 69
 par, 47, 75
 passwd, 4
 パスワード, 4
 paste, 26
 pbinom, 56
 pch, 68
 persp, 69
 pi, 28, 58
 pie, 68
 plot, 26, 48, 65, 68
 pnorm, 58
 points, 68
 Portrait, 64
 PostScript, 64
 postscript, 65
 ppois, 57
 predict, 63
 printer, 64
 prod, 32
 pty, 75
 qnorm, 58
 qqline, 49
 qqnorm, 49
 qqplot, 49
 qr, 33
 ランク, 20
 rbind, 20
 rbinom, 56
 read.table, 43
 レコード, 43
 rep, 17, 59
 rm, 8
 rnorm, 58
 論理演算子, 15
 ロジスティック回帰, 58
 rpois, 57
 runif, 58
 最尤法, 59
 削除, 8
 算術演算子, 11
 sapply, 26

scan, 42
search, 7
正規確率プロット, 48
線形写像, 20
seq, 11
説明, 8
shell モード, 9
sink, 66
指数, 12
solve, 33
sort, 11
相関係数, 53
source, 28
spin, 70
sqrt, 10, 30
stem, 46
sum, 30
summary, 62
数学関数, 12
数列, 11
svd, 38

T, 14, 22
t, 24
t.test, 50
table, 45
対数, 12
tapply, 26
 t 分布, 48
転置, 20, 24
TEX, 65
text, 27, 69
 t 検定, 50
特異値分解, 38
直交化, 30
直交行列, 38
直交射影, 31
type, 68
注釈, 6

var, 44, 52, 53
vecnorm, 30
vi, 5, 28

weights, 60
Welch の検定, 50
while, 17
whisker, 47
Wilcoxon 順位和検定, 51

X11, 26, 64
xlim, 47, 75

ylim, 47, 75
要素, 22
要因, 48

順位相関係数, 54