R ことはじめ

東芝インフォメーションシステムズ株式会社
横田博史

平成 20 年 8 月 22 日 (金)

目 次

第1章　Rの簡単な紹介

1.1　Rとは

Rは対話的処理による統計計算とデータの可視化を目的としたソフトウエアであり, GNU General Public License Version 2 [1]の下で配布されている.

Rはプログラム言語と言うよりは, 数値や文字列のデータ処理, 計算とデータの可視化に必要な機能, そして再帰が可能な函数が定義出来るプログラミング言語が統合化された環境である.

このRはそもそもAT&Tのベル研究所で開発されたS言語の影響を強く受けたソフトウエアであり, このS言語との互換性が高いという特徴がある. この事もあって, 最近ではS-Plus[2]とRを一緒に紹介する書籍も増えている. 猶,Rは Sのクローンを目指したものではない為, Sでは内部的な構成等で異なる点も多い.

猶,S-Plusとの互換性に関しては, 最近S-Plusに取り入れられた手法やS-Plusのグラフ表示機能をRは全て網羅していない. 但し, 一般的な統計処理であればS-Plusとの互換性に問題が生じる事は少ないだろう.

ここでの解説では,Rを利用する上で必要な事項のみを述べ, 統計的手法に関して詳細を述べる事はしない.

[1] http://www.gnu.org/copyleft/gpl.html 参照

[2] ベル研究所のS言語にInsightfulが独自の拡張を行った商用パッケージ. 処理系はSに準ずる. 国内では数理システムが代理店である.

1.2 Rの起動, デモと終了

Rの起動はxterm等の仮想端末上で R と入力するだけで良い.

以下に実際の起動画面を示しておこう.

```
yokota@tcimc176:~/Works/CAENET/R> R

R : Copyright 2006, The R Foundation for Statistical Computing
Version 2.3.0 (2006-04-24)
ISBN 3-900051-07-0

Rはフリーソフトウェアであり、「完全に無保証」です。
一定の条件に従えば、自由にこれを再配布することができます。
配布条件の詳細に関しては、'license()' あるいは'licence()' と入力してください。

Rは多くの貢献者による共同プロジェクトです。
詳しくは'contributors()' と入力してください。
また、RやRのパッケージを出版物で引用する際の形式については
'citation()' と入力してください。

'demo()' と入力すればデモをみることができます。
'help()' とすればオンラインヘルプが出ます。
'help.start()' でHTMLブラウザによるヘルプがみられます。
'q()' と入力すればRを終了します。

>
```

Rは対話処理で入力したデータの処理を行うシステムである. Rへの入力は入力プロンプト>に続けて入力すれば良い. 入力行は基本的に Enter キーによって入力が中断される迄となる. 猶, 二項演算子で二つ目の被演算子が入力をしていない等, 入力行に不備がある場合には, 入力が継続中である事を示すプロンプト+が表示される事もある. この場合には, 足りない入力を追加して Enter キーを押す事で入力行を終えられる.

では, 最初に demo() と入力してみよう. すると, 以下に示す実行可能なデモの一覧が表示される

```
Demos in package 'base':

is.things               Explore some properties of R objects and
                        is.FOO() functions. Not for newbies!
recursion               Using recursion for adaptive integration
scoping                 An illustration of lexical scoping.

Demos in package 'graphics':

Hershey                 Tables of the characters in the Hershey vector
```

```
                        fonts
Japanese                Tables of the Japanese characters in the
                        Hershey vector fonts
graphics                A show of some of R's graphics capabilities
image                   The image-like graphics builtins of R
persp                   Extended persp() examples
plotmath                Examples of the use of mathematics annotation

Demos in package 'stats':

glm.vr                  Some glm() examples from V&R with several
                        predictors
lm.glm                  Some linear and generalized linear modelling
                        examples from 'An Introduction to Statistical
                        Modelling' by Annette Dobson
nlm                     Nonlinear least-squares using nlm()
smooth                  'Visualize' steps in Tukey's smoothers


Use 'demo(package = .packages(all.available = TRUE))'
to list the demos in all *available* packages.
```

この状態で,q と入力すれば, 上記の表示は消え, 入力プロンプトが現われる. 猶,help 函数を用いてオンラインマニュアルを表示した場合も, 同様にマニュアルが表示される. そして, 読み終えて不要になった場合もq と入力すれば, 通常の入力に戻る.

R の対話処理では, 入力行の編集や以前入力した行を入力行として利用する事も可能である. その場合,emacs と同じキーの割当で入力行や履歴の編集が可能である. そこで,Ctrl+P か上向きの矢印キー ↑ を入力してみよう. すると, 先程入力した demo() がそのまま表示される筈である. 次に,Ctrl+A と入力しよう. すると, カーソルが行の先端に移動する. それから右の矢印キーの → かCtrl+F を入力すると, カーソルが行末へと移動する.

次の表 1.1 に, これらのキーの割当を纏めておこう.

表 1.1: キーの割当

キー	処理内容
Ctrl+A	編集行の先端にカーソルを移動
Ctrl+E	編集行の末尾にカーソルを移動
Ctrl+B	編集行上で一文字カーソルを後退
Ctrl+F	編集行上でカーソルを一文字前進
Ctrl+D	編集行上で一文字消去
Ctrl+K	カーソルの置かれた文字から行末迄を削除
Ctrl+P	一つ前の履歴の呼出
Ctrl+N	一つ後の入力行の呼出

では, カーソルを小括弧 () の中に移動させて,`'Japanese'`と入力しよう. すると, グラフィックス用の画面が現われて, 日本語を含む表を表示するデモが実行される. 猶,demo 函数では, 引数のデモ名を引用符で括っても, 括らなくても構わない.

最後に,R の終了は`quit()`, 或いは`q()`と入力する. すると以下のメッセージが表示される.

```
> q();
Save workspace image? [y/n/c]:
```

ここで`y`と入力すると,R を立ち上げてから定義し, 処理した R のオブジェクトを作業ディレクトリ (基本的には R を立ち上げた時のディレクトリ) 上の.RData ファイルに, 作業履歴を.Rhistory ファイルに各々保存した後に R を終了する. `n`の場合は`y`と入力した時に実行する終了処理を一切行わずに直ちに R を終了する. 最後に`c`の場合は,R を抜けずに R のセッションに復帰する.

R は立ち上げる際に, カレントディレクトリ上に.RData ファイルと.Rhistory ファイルが存在すると, これらのファイルの読込を行い, これらのファイルを生成した時点の状態に復帰する.

ここで.RData ファイルはバイナリデータファイル,.Rhistory ファイルはテキストファイルである. その為,.Rhistory ファイルは R での作業を終えた後に適当なエディッタで変数する事が可能である.

例えば,.RData や.Rhistory が無いディレクトリ上で R を立ち上げて以下の処理を実行したとしよう.

```
> a<-1+3;
> a/4;
[1] 1
> q();
Save workspace image? [y/n/c]: y
yokota@tcimc176:~/Works>
```

終了時に y と答えた事で生成される.Rhistory ファイルの内容は, 以下に示す様に,R への入力がそのまま保存される.

.Rhistory の内容

```
a<-1+3;
a/4;
q();
```

この.Rhistory は次に R を立ち上げると自動的に読込まれて,R の入力履歴として利用される. そこで, この履歴ファイルの内容を書換えて R を立ち上げると, R の命令行で履歴の呼出を行うと書換えられた履歴が表示される.

但し, 対象は.RData の内容が直接反映され,.Rhistory の内容を再実行した結果にはならない為に,.Rhistory を書換えても, 変更した履歴に対応する対象が生成されている訳ではない.

この事を上手く利用すれば,R をより効率的に利用出来る.

1.3 簡単な計算

R は統計処理を目的とするソフトウェアである. では数式はどの様に入力すれば良いのだろうか? 再び R を立ち上げて,FORTRAN の書式で数式が入力出来るかどうかを確認してみよう.

```
> 1+2*3-5/6
[1] 6.166667
> 3+9/
+ 3
[1] 6
> 3+3^2/3
[1] 6
> 4**3
[1] 64
```

この例で示す様に R の四則演算は C や FORTRAN と同じ書式で記述可能である. R の入力では *Mathematica* や Maxima といった数式処理システムの様に入力行の末端に; を入力する必要は無い. 猶,`3+9/3` の入力例に示す様に `3+9/`まで入力して途中で Enter キーを押した場合, 商の演算子 `/`が第二の被演算子を必要とする為に, プロンプトが通常の`>` から入力が継続中である事を示す`+`に切替わっている事に注意されたい.

最後に, 冪の演算子は多くの数式処理システムで用いられる`^`や FORTRAN の`**`の両方が使える. 猶, 算術演算子の多くは, 後述の数値のデータ列や行列に対しても利用可能である.

では, 以下に R の持つ主な算術演算子を示しておこう.

R の主な算術演算子

演算	構文	概要
和	`x + y`	$x+y$
差	`x - y`	$x-y$
積	`x * y`	$x \times y$
商	`x / y`	$\frac{x}{y}$
冪	`x ^ y`	x^y
冪	`x ** y`	x^y
剰余	`x %% y`	$x \ mod \ y$
商	`x %/% y`	$\frac{x-(x \ mod \ y)}{y}$

ここで示す様に,R の数値演算子は殆ど C や FORTRAN のものと同じである. 猶,R 独特の演算子は, 剰余の `x %% y` と商の `x %/% y` の二つである. ここで最初の `x %% y` は $x = ay + r$ と記述される場合, その剰余 r を返す. 次の `x %/% y` は a を返す演算子である.

以下に, これらの演算子の計算例を示しておこう.

```
> 2+4
[1] 6
> 4-12
[1] -8
```

```
> 4*6
[1] 24
> 6/9
[1] 0.6666667
> 4^3
[1] 64
> 4.5**2.9
[1] 78.40004
> 10 %% 8
[1] 2
> 10 %/% 8
[1] 1
```

1.4 変数と値の割当

R の変数はアルファベットで開始し, 空行を含まずに引用符で括られていない文字列である. この R の変数に値を割当てる場合, FORTRAN と同様に, 演算子`=`を用いる方法と演算子`<-`,`<<-`,`->`, `->>`を用いる方法がある.

以下に, 変数への値の割当の構文を纏めておこう.

変数への値の割当の構文

演算子	構文
<-	⟨ 変数 ⟩ < −⟨ 値 ⟩
<<-	⟨ 変数 ⟩ << −⟨ 値 ⟩
->	⟨ 値 ⟩− > ⟨ 変数 ⟩
->>	⟨ 値 ⟩− >> ⟨ 変数 ⟩
=	⟨ 変数 ⟩ = ⟨ 値 ⟩

ここで,=と<-は同じ意味であるが,=の方は R の最上層のみでしか使えない制約が入る.

->j は左右の被演算子の順序が<-の逆になる事を除けば, <-と同じ意味である.

```
> a<-1.2
> a
[1] 1.2
> a*3+1->b
> b
[1] 4.6
```

<<-は<-と違った特殊な使い方が出来るが, 通常の利用ではあまり用いる事はない. この演算子の詳細に関しては,R 付属のマニュアルを参照されたい.

1.5 オブジェクト

変数に値を割当てると,R 内部に変数名のオブジェクトが生成される. R 内部にどの様なオブジェクトが存在するかは`ls()` , 或いは`objects()` と入力すると,R 内部に存在するオブジェクトの一覧が表示される.

```
> x<-1
> y<-3
> ls()
[1] "x" "y"
> objects();
[1] "x" "y"
>
```

ここで,ls や objects では末尾の小括弧 () を省略して入力しない様にしなければならない.

又, 不要なオブジェクトは rm 函数で削除する事が可能である.

rm 函数

rm (”⟨ オブジェクト名$_1$⟩, $\cdots$, ⟨ オブジェクト名$_n$⟩”)

では以下に簡単な例で確認してみよう.

```
> word1<-"これはテストです"
> number1<-1+2+3/4
> objects();
[1] "number1" "word1"
> word1
[1] "これはテストです"
> rm("number1")
> objects();
[1] "word1"
>
```

この例では, 最初に word1 と number1 を値を割当てる事で生成している. それから,`objects()` で確認している. 次に,`rm("number1")` で number1 を消去し, 最後に`objects()` で number1 が R から消去された事を確認している.

第2章　型の話

前節では一寸した数値計算を行ったが, 一般の数値では FORTRAN の書式で計算が行える. ところで, 前節では入力した数値の型に関しては全く意識せずに計算を行っている.R は C の様なコンパイラ言語とは異なり, 数値等を直接入力して, 対話的に処理が行え, その上,R が扱えるデータは非常に多岐なものである. この R の特性を生かす事で効率的に問題を解く事が可能となるのである.

ここで R で扱えるデータ型としては, 数値, 文字列, データ列, 行列やリストがある. この節では, これらのデータ型の解説を行う.

2.1　数値

先ず,R で扱える数値の型を天下り的であるが, 以下の表 2.1 に纏めておこう.

表 2.1: R で扱える数値

型	例
`double`	`1,1.0`
`complex`	`1+1i`
`integer`	`1:10`
`numeric`	`1,1.0,NaN,Inf`

R の数値型は double,complex と numerical の境界は非常に緩く,complex が大小関係が複素数に入らない事もあって, 他と比べて特異である. 又, 型が分らない場合には,`mode` 函数を用いて型を調べられる.

```
> mode(1)
[1] "numeric"
> mode(1+1i)
[1] "complex"
>
```

では, 以降の小節で各数値型について簡単に解説しよう.

2.1.1　double 型

double 型は R で扱う数値の中では最も基本的な数値型である. 何故なら,R に直接入力した数値は全て double 型として扱われる為で, これは小数点を入れようが入れまいが無関係である. 則ち,`1.0` も `1` も同じ double 型の数値になる.

実際にこの事を確認してみよう.

```
[1] TRUE
> 1.0==1
[1] TRUE
> 1.0==0.99999999999999999
[1] TRUE
> 1.0==0.9999999999999999
[1] FALSE
```

この例では比較の演算子==を用いている. この比較の演算子==はC の同名の演算子と同じ働きをする演算子である. この例から分る様に,1.0 と 1 は同じ double 型の数値である. 又,1 と 0.99999999999999999 は一致するが, 0.9999999999999999 は 1 とは異なる. これは前者が $0.99\cdots$ と解釈されたのに対し, 後者は 0.99999999999999990 と解釈された為である.

最後に,R は特殊な数値を含んでいる. これらは欠損値 NA , 不定値 NaN , 無限大 Inf である.

```
> 0/0
[1] NaN
> 1/0
[1] Inf
```

2.1.2 complex 型

R では通常の数値だけではなく複素数も扱える. 例えば, 複素数 $1+2i$ を R では 1+2i と表記する. ここで純虚数は i ではなく 1i と表記する事に注意されたい.

```
> 1+2i
[1] 1+2i
> (2+1i)*(2-1i)
[1] 5+0i
>
```

ここで, 計算の結果, 虚部が 0 になっても R は実数に変換せずに複素数としての表現で表示を行う. この例で,5+0i は R の内部データとしては 5 とは別物である. この事を実際に R で確認してみよう.

```
> 5+0i==5
[1] TRUE
> 5<6
[1] TRUE
> 5+0i<6
以下にエラー 5 + (0+0i) < 6 : 不正な複素値との比較です
> typeof(5+0i)
[1] "complex"
```

```
> typeof(5)
[1] "double"
>
```

この例で示す様に, 後述の等号の演算子==を用いた場合,5+0i は数値の 5 と等しくなる. ところが, 大小関係の演算子では,5+0i が複素数の為に比較が出来ない. 実際, 引数の型が調べられる函数 typeof で調べると 5 が double 型であるのに対し,5+0i は complex 型となり, 全く別物である事が明瞭になる. この様に,1i や 0i の扱いには注意が必要である.

2.1.3 integer 型

integer 型は double 型と非常に紛らわしい型である. 基本的に列の添字として用いられる型でもあるが, 実の所,double 型との境界はあやふやでもある.

この integer 型の数値は, 後述のデータ列を生成する演算子: ,seq 函数 更には,integer 函数で生成された列や as.integer 函数を用いてデータ列を integer 型に変換したものが対応する.

ここでは幾つかの実例を示しておこう.

```
> int1<-1:1
> int1
[1] 1
> is.integer(int1)
[1] TRUE
> x1<-1
> is.integer(x1)
[1] FALSE
> is.double(int1)
[1] FALSE
> is.double(x1)
[1] TRUE
```

この例では, 整数列を生成する演算子:を用いて整数の 1 を生成している. この整数 1 は変数 int1 に割当てられており, 引数が integer 型であるかどうかを判別する函数 is.integer では真 (TRUE) が返却されている. それに対して直接入力した数値の 1 は double 型で,integer 型ではない事に注意されたい. 猶,seq 函数でも integer 型のデータが生成可能であるが, この場合は, 演算子:と併用するか,seq(1,10) の様に始点と終点のみを指定した場合と seq(10) の様に終点のみを指定した場合に限られる. この事に関しては, データ列の節にて詳細を述べる.

2.1.4 numeric 型

numeric 型は integer 型と double 型を包含する型である. 以下に,integer 型であれば TRUE を返す is.integer 函数,double 型であれば TRUE を返す is.double 函数, そして,numeric 型であれば TRUE を返す is.numeric でデータ列の評価をしてみよう.

```
> int1<-1:10;
```

```
> double2<-seq(1,10,length=9)
> int1
 [1]  1  2  3  4  5  6  7  8  9 10
> double2
[1]  1.000  2.125  3.250  4.375  5.500  6.625  7.750  8.875 10.000
> is.integer(int1)
[1] TRUE
> is.integer(double2)
[1] FALSE
> is.double(int1)
[1] FALSE
> is.double(double2)
[1] TRUE
> is.numeric(int1)
[1] TRUE
> is.numeric(double2)
[1] TRUE
```

この様に,numeric 型は integer 型や double 型を包含する事が分る. ところで,numeric 型は complex 型を含まない. 実際, 以下に示す様に `is.numeric(1+1i)` の結果として FALSE が返される事から分る.

```
> is.numeric(1+1i)
[1] FALSE
> mode(1+1i)
[1] "complex"
>
```

因に, この例で用いている函数 `mode` は引数として与えられたオブジェクトの型を返す函数である.

2.2 論理値

R には様々な数値に加えて,`TRUE`(真) ,`FALSE`(偽) の論理値がある. この論理値は logical 型となる.

猶, 論理値は `T`,`F` と略記する事が可能であるが,`T` や `F` は単純に `TRUE` や `FALSE` が割当てられているだけなので, 他の値を割当てる事も可能である. その為, 迂闊に `T` や `F` を変数として利用しない様に注意しなければならない.

ここで論理値を返す演算子としては以下の演算子がある

論理値を返す比較の演算子

演算子	概要
==	等しい
!=	等しくない
>=	以上
<=	以下
>	大
<	小

では, 以下にこれらの演算子を用いた実例を示しておこう.

```
> (1==2)
[1] FALSE
> (sqrt(4)==2)
[1] TRUE
> 1!=2
[1] TRUE
> (1==2)==T
[1] FALSE
> (1==2)==F
[1] TRUE
```

R の論理演算子は以下に示す様に C の論理演算子と同様である.

R の論理演算子

演算子	概要
&&	論理積 (AND)
||	論理和 (OR)
!	否定 (NOT)

```
> (1==2) && (1==1)
[1] FALSE
> (1==2) || (1==1)
[1] TRUE
> !((1==2) && (1==1))
[1] TRUE
>
```

TRUE,FALSE を数値計算に混ぜて使う事も可能である. この場合, 真が 1, 偽が 0 として解釈される.

この特性を上手く利用する事で,MATLAB 風に if 文の様な条件分岐文を用いずに処理を行う事が可能である.

```
> (sin(783)>0)*3+(sin(783)<0)*(-2)
```

```
[1] -2
```

この例では,sin (783) が 0 より大であれば 3, sin (783) が 0 より小であれば-2 を返す処理をさせている. この様に, 一行だけで処理が行えている事に注意されたい.

2.3 文字列

R でも文字列が扱える.R の文字列は引用符で文字の羅列を括ったものである. R では character 型として扱われる. 猶, 引用符は二重引用符, 単引用符の何れも利用可能である.

```
> "三毛猫"
[1] "三毛猫"
> ’たま’
[1] "たま"
>
```

猶, 文字列内に引用符等の記号を入力したければ, \ をその記号の直前に置くとよい.

2.4 NULL

R には与えられたデータ列が空である事を示す NULL 型がある. 又, 内部には NULL 型のオブジェクト NULL もある.

```
> c()
NULL
> mode(c())
[1] "NULL"
> c()
NULL
```

この例では, データ列を生成する函数 c に引数を与えなかった為に, 返却値として NULL が返されており, このオブジェクト NULL の型は NULL 型である事を示している.

2.5 データ列

R ではデータ列を柔軟に生成する事が可能である.R のデータ列は, 上述の数値, 論理値や文字列が混在したものも扱える.

ここで R で扱う列の型は, 上述の数値の double 型や integer 型, 論理値の logical 型や文字列の character 型のみの列で構成される vector 型, これらのデータが混在したデータ列の list 型がある. ここで vector と list 型は double,integer,logical や character 型の上位に来る型になるが,vector 型と list 型は包含関係の無い別個の型である. しかし, 成分の取出等の操作に関しては違い無く扱える.

ここでは最初に規則性を持つ数列の生成について述べよう.

2.5.1 数列の生成

R では MATLAB 風に演算子:を用いて integer 型の数列を生成する事が可能である. この場合, 間隔を 1 とする数列が容易に生成出来, この数列は integer 型になる. しかし, 演算子:を用いた場合,1 以外の間隔, 例えば, 間隔を 2 にした数列は生成出来ない.

これに対して sequence 函数やその短縮名の seq 函数を用いれば, より多彩な数列が生成可能である.

では, 演算子:と函数 seq の構文を以下に纏めておこう.

数列生成の構文

⟨ 始点 ⟩ : ⟨ 終点 ⟩
seq(⟨ 始点 ⟩, ⟨ 終点 ⟩)
seq(⟨ 始点 ⟩, ⟨ 終点 ⟩, ⟨ 間隔 ⟩)
seq(⟨ 始点 ⟩, ⟨ 終点 ⟩, by=⟨ 間隔 ⟩)
seq(⟨ 始点 ⟩, ⟨ 終点 ⟩, length=⟨ 数列の長さ ⟩)
seq(along=⟨ 数列 ⟩)
seq(⟨ 数列 ⟩)
seq(⟨ 終点 ⟩)

演算子:を用いる場合, ⟨ 始点 ⟩ から ⟨ 終点 ⟩ までの間隔 1 の数列が生成される. この演算子は MATLAB の同名の演算子に似ているが, この演算子で生成可能な数列は間隔 1 の単調列のみである. 従って,1 以外の一定間隔を持つ数列の生成は seq 函数を用いる必要がある.

この seq 函数では間隔の指定はオプションの by=で行うか, 始点と終点を指定し, 第三の引数に間隔を指定する. 例えば,seq(1,10,by=2) とすると, 間隔 2 の数列 1 3 5 7 9 が生成される. これは seq(1,10,2) と入力しても同値である.

次に,seq 函数に length=オプションを用いると, 数列の長さ, 則ち, 数列に含まれる成分の個数が指定可能である.

seq(along=⟨ 数列 ⟩) や seq(⟨ 数列 ⟩) で,1 から引数の数列の個数までの刻幅 1 の単調草加列を生成する.

最後の seq(⟨ 終点 ⟩) は 1 から開始して, ⟨ 終点 ⟩ で終了する増分 1 の単調増加列を生成する.

猶,seq 函数を用いた場合,seq(⟨ 終点 ⟩), seq(⟨ 始点 ⟩, ⟨ 終点 ⟩) と seq(⟨ 始点 ⟩ : ⟨ 終点 ⟩) で生成した数列のみが integer 型となり, それ以外は double 型となる.

では実際に数列の生成してみよう.

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> 10:1
 [1] 10  9  8  7  6  5  4  3  2  1
> x<-seq(1,10,by=2)
> x
[1] 1 3 5 7 9
> seq(1,10,3)
[1]  1  4  7 10
> seq(10,1,length=6)
```

```
[1] 10.0  8.2  6.4  4.6  2.8  1.0
> seq(along=x)
[1] 1 2 3 4 5
> y<- seq(5)*2
> y
[1]  2  4  6  8 10
> seq(y)
[1] 1 2 3 4 5
>
```

この様にして, 一定の間隔で単調に増大, 減少する数列が生成可能である.

R では seq の他に integer 函数を用いて integer 型の列が生成可能である. 但し,integer 函数で生成した数列の成分は全て 0 である.

integer 函数の構文

integer(⟨ 整数値 ⟩)

この構文で指定した整数値を長さ (成分の個数) とする全ての成分が 0 の integer 型の数列が生成される.

```
> integer(10)
 [1] 0 0 0 0 0 0 0 0 0 0
```

実際は後述の as.integer 函数を用いて integer 型の数列を生成する方が必要な数列を生成する事が容易である.

2.5.2 一般的なデータ列

ところで,R で扱えるデータ列は数値だけではない. より一般的な列は c 函数や list で生成する事が可能である.

先ず,c 函数の構文を以下に示そう.

函数 c による列の生成

c(⟨ データ$_1$⟩, $\cdots$, ⟨ データ$_n$⟩)

猶, ここでの ⟨ データ$_i$⟩ は数値や文字列をコンマで区切ったもので構成されたデータ列, 数列や文字列を自由に混ぜたデータ列を意味する.

以下に, 簡単な列の一例を示そう.

```
> c(1,2,3,4,5)
[1] 1 2 3 4 5
> c("a","b","c","d")
[1] "a" "b" "c" "d"
> c("a","b","c","d",1,2,34,"aa")
[1] "a"  "b"  "c"  "d"  "1"  "2"  "34" "aa"
```

```
> c(1,2,3,TRUE,FALSE)
[1] 1 2 3 1 0
>
```

この c 函数を用いた場合, 成分に文字列が含まれている場合には, 返却されるデータ列の成分は全て文字列に変換される. 同様に, 数値と論理値が混在する場合, 論理値は数値に変換される.

これに対し,list 函数は c と同じ構文が使えるが, その際に, 引数のデータ列の成分の型を変更する事はしない.

list 函数の構文

list(⟨ データ$_1$⟩, $\cdots$ ⟨ データ$_n$⟩)
list(⟨ ラベル$_1$⟩=⟨ データ$_1$⟩, $\cdots$ ⟨ ラベル$_n$⟩=⟨ データ$_n$⟩)

この list 函数で生成するデータ列をリストと呼び, 型は list 型となる. この list 型は double 型を包含しない独立した型である.

最初の構文は vector 型のデータ列を生成する c と似たデータ列を生成する.

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

[[2]]
[1] 2

[[3]]
[1] 3

> z<-list(1,"345","三毛猫",2,TRUE)
> z
[[1]]
[1] 1

[[2]]
[1] "345"

[[3]]
[1] "三毛猫"

[[4]]
[1] 2

[[5]]
[1] TRUE
```

```
>z[2]
[[1]]
[1] "345"

>
```

これに対し, 第二の構文はリストの要素にラベルを付ける形になる. 例えば, 以下の様にデータを設定してみよう.

```
> animal <- c('三毛猫','虎猫','狸')
> worth <- c(9,8,4)
> data1<-list(動物=animal, 価値=worth)
> data1
$動物
[1] "三毛猫" "虎猫"   "狸"

$価値
[1] 9 8 4

> data1["動物"]
$動物
[1] "三毛猫" "虎猫"   "狸"

> data1["価値"]
$価値
[1] 9 8 4
>
```

この場合, ラベルの参照は data1['⟨ ラベル ⟩'] の様にラベルを引用符で囲んで行う. 何も指定しない最初の方法では, ラベルではなく数値で成分を指定する.

2.5.3 周期性のあるデータ列

R は MATLAB と違って周期性のある列も容易に生成する事が可能である. ここでの列は数値だではなく, 文字列を含んでいても構わない.

この様な周期性を持った列を構成する場合,rep 函数を用いる.

周期性のある列の生成

```
rep(⟨ 列 ⟩, ⟨ 回数 ⟩)
rep(⟨ 列 ⟩, ⟨ 回数 ⟩, times=⟨ 回数 ⟩)
rep(⟨ 列 ⟩, ⟨ 回数 ⟩, ⟨ 長さ ⟩)
rep(⟨ 列 ⟩, ⟨ 回数 ⟩, length=⟨ 長さ ⟩)
```

この rep 函数は,⟨ 列 ⟩ で与えられるパターンを指定した回数, 或いは反復した列から指定した長さの列を取出す函数である.

反復回数はオプションの times, 出力数列の長さは length オプションで指定可能である.

```
> x<-c("a","b","c",2,34,"aa")
> rep(x,2)
 [1] "a"  "b"  "c"  "2"  "34" "aa" "a"  "b"  "c"  "2"  "34" "aa"
> rep(x,times=3,length=8)
[1] "a"  "b"  "c"  "2"  "34" "aa" "a"  "b"
>
```

2.6 行列

R では行列も扱える. 行列の生成では函数 matrix を用いる.

行列の生成

matrix(⟨ データ列 ⟩, ⟨ 行数 ⟩, ⟨ 列数 ⟩)
matrix(⟨ データ列 ⟩, nrow=⟨ 行数 ⟩, ncol=⟨ 列数 ⟩)
matrix(⟨ データ列 ⟩, nrow=⟨ 行数 ⟩, ncol=⟨ 列数 ⟩, byrow=⟨ 真理値 ⟩)

猶,matrix 函数によって生成される対象は matrix 型である.

この matrix 函数による行列の生成では, 与えられたデータ列を指定した大きさの行列に当て嵌める事を行う. オプションの byrow はデフォルトでは FALSE になっているが, これは数列を 1 列目から順番に列で当て嵌める事を意味する. 又,byrow=TRUE にした場合は,1 行目から順番に数列の内容を入れて行く事を意味する.

この事を数列 $1, 2, \cdots, 9$ から 3×3 の行列を生成する事で確認して見よう.

```
> matrix(1:9,3,3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> matrix(1:9,3,3,byrow=T)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> matrix(1:9,3,3,byrow=F)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

ここで,matrix 函数は与えられたデータ列を指定した大きさの行列に当て嵌める操作を行う. もし, データの大きさが指定した行列よりも大き過ぎる場合には, 指定した行列に適合する範囲内でデータが行列の各成分に置かれ, 逆に少ない場合には, データ列の頭から不足分を補う.

```
> matrix(1:9,3,3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> matrix(1:9,3,2)
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
Warning message:
行列のデータ長 [9] が列数 [2] を整数で割った、もしくは掛けた値ではありません
> matrix(1:9,3,4)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7    1
[2,]    2    5    8    2
[3,]    3    6    9    3
Warning message:
行列のデータ長 [9] が列数 [4] を整数で割った、もしくは掛けた値ではありません
```

行列は数値に限定されない. 文字列や論理式を含む列からも構成可能である. この場合, 列は vector 型でも list 型でも問題は無く, 成分の型を変更する様な事も無い.

```
> mike<-c("a","b","c","d","e","f")
> matrix(mike,3,2)
     [,1] [,2]
[1,] "a"  "d"
[2,] "b"  "e"
[3,] "c"  "f"
> neko<-c(1,2,3,"a","b","c","d","e","f",T,F,T)
> matrix(neko,3,4)
     [,1] [,2] [,3] [,4]
[1,] "1"  "a"  "d"  "TRUE"
[2,] "2"  "b"  "e"  "FALSE"
[3,] "3"  "c"  "f"  "TRUE"
> pochi<-c(1,2,3,T,F,T)
> matrix(pochi,3,4)
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    2    0    2    0
```

```
[3,]    3    1    3    1
> neko2<-list(1,2,3,"a","b","c","d","e","f",T,F,T)
> matrix(neko2,3,4)
     [,1] [,2] [,3] [,4]
[1,] 1    "a"  "d"  TRUE
[2,] 2    "b"  "e"  FALSE
[3,] 3    "c"  "f"  TRUE
>
```

2.7 列や行列から成分の取出し

R ではデータ列や行列から成分の取出が非常に柔軟に行える．データ列や行列の成分の取出しは,FORTRAN の配列の様に成分を指定して取出す事が可能である. 例えば, データ列 d1 から第 i 番目の成分を取出す場合,`d1[i]` と入力する. 行列 m1 から i 行 j 列の成分を取出す場合,`m1[i][j]`, 或いは MATLAB 風に `m1[i,j]` と入力すれば良い. この様に, データの取出しは FORTRAN や C と大差の無い手法もある.

ところで, 行列から成分を取出す場合, 与えられた行列をデータ列と看做して成分を取出す事も可能である. 例えば,$m \times n$ の行列 M で k 番目の成分としては,`M[((k-1) %% m) + 1, ((k-1) %/% m) + 1]` が対応する.

列や行列から指定した成分を取出す標準的な方法

構文	概要
m[⟨ 正数値$_1$⟩]	列, 或いは行列 m から ⟨ 正数値 ⟩ で指定した成分を取出す場合
m[⟨ 正数値$_1$⟩, ⟨ 正数値$_2$⟩]	行列 m から ⟨ 正数値$_1$⟩ 行,⟨ 正数値$_2$⟩ 列の成分を取出す場合

上記の方法は C や FORTRAN の配列の取出しとほぼ同様の手法である．R のユニークな点は, より柔軟にデータ列や行列から成分を抜き出す事が可能な点にある. 先ず, 最初に行や列のみをそのまま行列から取出す方法について述べよう.

行列から行や列の取出し (その 1)

構文	概要
m[⟨ 正数値 ⟩,]	行列 m から ⟨ 正数値 ⟩ 番目の行を取出す場合
m[, ⟨ 正数値 ⟩]	行列 m から ⟨ 正数値 ⟩ 番目の列を取出す場合

この方法は与えられた行列から行や列を取出す MATLAB では非常に一般的な方法である. 以下の例題で具体的に解説しよう.

```
> m2
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> m2[2,]
[1]  2  6 10
> m2[,2]
[1] 5 6 7 8
>
```

先ず, 行列 m2 が与えられたとする. そこで,m2[2,] で行列 m2 の第 2 行のみを取出す. ここで, 行列の列の部分が空になっているが, こうする事で, 列全てを取出す事を意味する. 同様に,m2[,2] とする事で, 行列 m2 の 2 列目を全て取出す. 猶,R の場合は行列から行を取出そうが, 列を取出そうが, 返却される値はデータ列であり, 行ベクトルや列ベクトルではない.

ところで,R の面白い点は, 指定する添字には, 正の数値ではなく負の数値も扱える事である. この場合, 意味が異なり, 指定した行や列を削除した行列を返す.

この構文を以下に纏めておこう.

行列から行や列の取出 (その 2)

構文	概要
m[⟨ 負数値$_1$⟩, ⟨ 負数値$_2$⟩]	行列 m から ⟨ 負数値$_1$⟩ 行,⟨ 負数値$_2$⟩ 列を除去した小行列を返す
m[⟨ 正数値$_1$⟩, ⟨ 負数値$_2$⟩]	行列 m から ⟨ 正数値$_1$⟩ 行の ⟨ 負数値$_2$⟩ 列を除去した列を返す
m[⟨ 負数値$_1$⟩, ⟨ 正数値$_2$⟩]	行列 m から ⟨ 負数値$_1$⟩ 行を除去した小行列から, ⟨ 正数値$_2$⟩ で指定した列を返す
m[⟨ 負数値$_1$⟩,]	行列 m から ⟨ 負数値$_1$⟩ 行を除去した小行列を返す
m[, ⟨ 負数値$_2$⟩]	行列 m から ⟨ 負数値$_2$⟩ 列を除去した小行列を返す

では先程の行列 m2 を用いて, 負の整数を指定した場合の挙動を確認しておこう.

```
> m2
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> m2[-1,-2]
     [,1] [,2]
[1,]    2   10
[2,]    3   11
[3,]    4   12
> m2[1,-2]
[1] 1 9
> m2[-1,2]
[1] 6 7 8
> m2[-1,]
     [,1] [,2] [,3]
[1,]    2    6   10
[2,]    3    7   11
```

```
[3,]    4    8   12
> m2[,-2]
     [,1] [,2]
[1,]    1    9
[2,]    2   10
[3,]    3   11
[4,]    4   12
>
```

次に, データ列や行列から複数の成分を指定して取出す事も可能である. この場合,MATLAB 風に成分の指定を行えば良い.

複数の行や列の取出

構文	概要
m [⟨ 数列$_1$⟩,⟨ 数列$_2$⟩]	行列 m から ⟨ 数列$_1$⟩ で指定した行と ⟨ 数列$_2$⟩ で指定した列を取出す
m[⟨ 数列$_1$⟩]	m から ⟨ 数列$_1$⟩ で指定した成分を取出す

この方法は, 数列を:で構築した場合には,m2[1:4] の様に MATLAB 風の指定となる.

```
> r1<-seq(1,10,length=72);
> m1<-matrix(r1,3,4)
> m1
         [,1]     [,2]     [,3]     [,4]
[1,] 1.000000 1.380282 1.760563 2.140845
[2,] 1.126761 1.507042 1.887324 2.267606
[3,] 1.253521 1.633803 2.014085 2.394366
> m1[1:3,2:4]
         [,1]     [,2]     [,3]
[1,] 1.380282 1.760563 2.140845
[2,] 1.507042 1.887324 2.267606
[3,] 1.633803 2.014085 2.394366
```

この例では, 行列 m1 の第 1 行目から 3 行目, 第 2 列目から 4 列目を取出している.

猶,R では数列は何も考えずに構築すると double 型となる. 通常の処理系で添字に浮動小数点を用いるとエラーになるが,R の場合, 小数点以下を勝手に切り捨てるので, そのままでも利用可能である.

```
> r1<-seq(1,10,length=72);
>  m1<-matrix(r1,3,4)
> m1
         [,1]     [,2]     [,3]     [,4]
[1,] 1.000000 1.380282 1.760563 2.140845
[2,] 1.126761 1.507042 1.887324 2.267606
[3,] 1.253521 1.633803 2.014085 2.394366
```

```
> m1[seq(1,pi,length=3),seq(2,pi,length=2)]
         [,1]     [,2]
[1,] 1.380282 1.760563
[2,] 1.507042 1.887324
[3,] 1.633803 2.014085
> pi
[1] 3.141593
> seq(1,pi,length=3)
[1] 1.000000 2.070796 3.141593
>
```

この例では, 数列を seq(1,pi,length=3) で生成しているが, この数列は, 1.000000 2.070796 3.141593 となる. この数列を指定した場合, 内部で小数点以下を切り捨てて,1,2,3 を添字として用いている.

R では成分に関する条件を入れて成分の取出しを行う事も可能である. この場合, 与えた条件に対して TRUE となる箇所のみが返却される.

条件に適合する成分の取出し

構文	概要
m[⟨ 条件 ⟩]	m から ⟨ 条件 ⟩ に適合する成分を取出す

次の例では, 行列 m2 からその成分が 5 よりも大きい成分のみを取出している.

```
> m2<-matrix(seq(12),4,3)
> m2
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> m2[m2>5]
[1]  6  7  8  9 10 11 12
```

2.8 型に付随する函数

R には型に付随する二つの函数が存在する. 一つは, 与えられたオブジェクトがある型に属するものであるかどうかを論理値で返す is 函数であり, オブジェクトを指定した型に変換して返す as 函数である.

先ず, 型を確認する函数は型の頭に is. を付けた函数が各型に付随する. その一覧を以下に示しておこう.

型を確認する函数

構文	概要
is.integer(⟨ 対象 ⟩)	(⟨ 対象 ⟩) が integer 型であれば TRUE, それ以外は FALSE
is.double(⟨ 対象 ⟩)	(⟨ 対象 ⟩) が double 型であれば TRUE, それ以外は FALSE
is.complex(⟨ 対象 ⟩)	(⟨ 対象 ⟩) が complex 型であれば TRUE, それ以外は FALSE
is.numeric(⟨ 対象 ⟩)	(⟨ 対象 ⟩) が numeric 型であれば TRUE, それ以外は FALSE

では, 実際に型を確認してみよう.

```
> is.double(1)
[1] TRUE
> is.integer(1)
[1] FALSE
> is.numeric(1)
[1] TRUE
> is.complex(1)
[1] FALSE
> is.complex(1+2i)
[1] TRUE
> is.numeric(1+2i)
[1] FALSE
> is.numeric(Inf)
[1] TRUE
```

型を確認する函数に加えて, その変数に割当てられたデータの型を変換する函数がある. この函数は型の名前の先頭に as. を追加したものである.

型の変換を行う函数

構文
as.integer(⟨ 対象 ⟩)
as.double(⟨ 対象 ⟩)
as.complex(⟨ 対象 ⟩)
as.numeric(⟨ 対象 ⟩)
as.character(⟨ 対象 ⟩)
as.vector(⟨ 対象 ⟩)
as.list(⟨ 対象 ⟩)
as.matrix(⟨ 対象 ⟩)

では,seq(1,pi*2,length=10) で生成した数列を,integer 型,complex 型, list 型, そして,matrix 型に変換する例を以下に示しておく.

```
> x1<-seq(1,pi*2,length=10)
> x1
 [1] 1.000000 1.587021 2.174041 2.761062 3.348082 3.935103 4.522124 5.109144
```

```
 [9] 5.696165 6.283185
> as.integer(x1)
 [1] 1 1 2 2 3 3 4 5 5 6
> as.complex(x1)
 [1] 1.000000+0i 1.587021+0i 2.174041+0i 2.761062+0i 3.348082+0i 3.935103+0i
 [7] 4.522124+0i 5.109144+0i 5.696165+0i 6.283185+0i
> as.list(x1)
[[1]]
[1] 1

[[2]]
[1] 1.587021

[[3]]
[1] 2.174041

[[4]]
[1] 2.761062

[[5]]
[1] 3.348082

[[6]]
[1] 3.935103

[[7]]
[1] 4.522124

[[8]]
[1] 5.109144

[[9]]
[1] 5.696165

[[10]]
[1] 6.283185

> as.matrix(x1)
          [,1]
 [1,] 1.000000
 [2,] 1.587021
 [3,] 2.174041
```

```
 [4,] 2.761062
 [5,] 3.348082
 [6,] 3.935103
 [7,] 4.522124
 [8,] 5.109144
 [9,] 5.696165
[10,] 6.283185
>
```

猶,complex 型から numeric 型に変換する場合, 虚部は自動的に切り捨てられる.

```
> y1<-x1+1i
> y1
 [1] 1.000000+1i 1.587021+1i 2.174041+1i 2.761062+1i 3.348082+1i 3.935103+1i
 [7] 4.522124+1i 5.109144+1i 5.696165+1i 6.283185+1i
> as.integer(y1)
 [1] 1 1 2 2 3 3 4 5 5 6
Warning message:
複素数の虚部は，コネクションで捨てられました
>
```

2.9 列の結合操作

列の結合は, 列に対して c 函数や list 函数を用いる方法の他に, append で新しいデータ列を生成する事も可能である.

append 函数による列の結合

append(⟨ データ列$_1$⟩, ⟨ データ列$_2$⟩)
append(⟨ データ列$_1$⟩, ⟨ データ列$_2$⟩, after=⟨ 挿入位置 ⟩)

ここでは, データ列が vector 型の場合の例を以下に示しておく.

```
> x1<-seq(1,4)
> x1
[1] 1 2 3 4
> y1<-seq(10,7)
> y1
[1] 10  9  8  7
> append(x1,y1)
[1]  1  2  3  4 10  9  8  7
> append(x1,y1,after=2)
[1]  1  2 10  9  8  7  3  4
>
```

この例で示す様に,after=に指定する挿入位置は, ⟨ データ列$_1$⟩ の先頭からの長さで指定する. この例では after=2 とした為に, 第一引数である x1 の 2 番目の成分の後に第二引数の y1 が挿入される形となる.

この append 函数は vector 型だけではなく,list 型や matrix 型に対しても利用可能である. 猶, どちらか一方が list 型であれば返却される列は list 型になり, 一方が matrix 型で, もう一方が vector 型の場合,vector 型が返却される.

2.10 データフレームについて

データフレームは R の data.frame クラスを持つリストで, このリストは数値データ列, 文字列データ列, 因子データ列等の様々な型のデータを含むものである. 猶, このデータフレームの外観は行列に似たものであるが, 行や列にはラベルが設定されている事が特徴である.

このデータフレームの生成は data.frame 函数を用いる. この函数の構文を以下に示しておこう.

data.frame 函数の構文

data.frame(⟨ データ列$_1$⟩, $\cdots$, ⟨ データ列$_n$⟩)
data.frame(⟨ 列名$_1$⟩=⟨ データ列$_1$⟩, $\cdots$, ⟨ 列名$_n$⟩=⟨ データ列$_n$⟩)
data.frame(⟨ 列名$_1$⟩=⟨ データ列$_1$⟩, $\cdots$, ⟨ 列名$_n$⟩=⟨ データ列$_n$⟩, row.names=⟨ 列名 ⟩)

最初の構文では, 列を単純に並べたデータフレームを生成する. この時, 行と列には行列としての数値ラベルが自動的に置かれる.

次の構文の場合, 各列には指定した列名が列のラベルとして設定される. 但し, 行のラベルは最初のものと同様に, 行の数値ラベルが自動的に設定されている.

最後の構文の場合, 行ラベルを row.names で指定している. これらの指定は勿論, 混在していても構わない.

では, 具体的な例で動作を確認しておこう.

```
> f1<-c("猫","猫","狸","犬","犬")
> f2<-c(24,22,42,37,73)
> data1<-data.frame(f1,f2)
> data1
  f1 f2
1 猫 24
2 猫 22
3 狸 42
4 犬 37
5 犬 73
>data2<-data.frame(動物=f1, 月齢=f2)
  動物 月齢
1   猫   24
2   猫   22
3   狸   42
4   犬   37
5   犬   73
> f0<-c("三毛","玉","ぽんぽこ","ポチ","エス")
> data3<-data.frame(動物=f1, 月齢=f2,row.names=f0)
> data3
         動物 月齢
三毛       猫   24
玉         猫   22
ぽんぽこ   狸   42
```

```
ポチ        犬    37
エス        犬    73
>
```

このデータフレームから列データの取出しは, 行列の様に列番号を入れるか, 或いはラベルを指定する事でも行える. 但し, この場合は `row.names` が付随したデータが返却される.

```
> data1
  f1 f2
1 猫 24
2 猫 22
3 狸 42
4 犬 37
5 犬 73
> data1[1]
  f1
1 猫
2 猫
3 狸
4 犬
5 犬
> data3
          動物 月齢
三毛        猫   24
玉          猫   22
ぽんぽこ    狸   42
ポチ        犬   37
エス        犬   73
> data3["月齢"]
          月齢
三毛        24
玉          22
ぽんぽこ    42
ポチ        37
エス        73
>
```

記号$を利用して, 指定したラベルが付いた data.frame の列データそのものを取出す事が可能である.

演算子$

〈データフレーム〉$〈ラベル〉

この事を先程の data1 と data2 を使って確認してみよう.

```
> data1$f1
[1] 猫 猫 狸 犬 犬
Levels: 犬 狸 猫
> data2$月齢
[1] 24 22 42 37 73
>
```

2.11 列や行に対する演算

R では, 成分が numeric である vector 型のデータ列や行列に対して, 和や積といった算術演算子を直接作用させる事が可能である.

先ず, データ列に対する算術演算子の一覧を以下に示しておくが, ここで v1 と v2 の長さは同じ列とする.

行列の演算

演算子	構文	概要
+	v1 + v2	列 v1 と列 v2 の成分毎の和
-	v1 - v2	列 v1 と列 v2 の成分毎の差
*	v1 * v2	列 v1 と列 v2 の成分毎の積 $v1\dot{v}2$
^	v1 ^ n	列 v1 の成分毎の n 乗
**	v1 ** n	列 v1 の成分毎の n 乗
/	v1 / v2	列 v1 と列 v2 の成分毎の商
x %% y	$x\ mod\ y$	
	x %/% y	$\frac{x-(x\ mod\ y)}{y}$
%o%	v1 %o% v2	外積 (outer(v1,v2) と同値
%x%	v1 %x% v2	クロネッカー積

この様に, 和, 差, 積, 冪と商に関しては, 成分毎の処理となる.

行列に対しては以下の演算がある.

行列の演算

演算子	構文	概要
+	m1 + m2	行列 m1 と行列 m2 の和
-	m1 - m2	行列 m1 と行列 m2 の差
*	m1 * m2	行列 m1 と行列 m2 の積 $m1\dot{m}2$
^	m1 ^ n	$m1^n$
/	m1 / m2	$m1\dot{m}2^{-1}$
%o%	m1 %o% m2	外積 (outer(m1,m2) と同値
%x%	m1 %x% m2	クロネッカー積

先ず, 和, 差, 冪と商に関しては,MATLAB と同様である.

```
> m1<-matrix(runif(9),3,3)
> m1
          [,1]       [,2]      [,3]
[1,] 0.1084662 0.05889304 0.6770447
[2,] 0.1024633 0.92267181 0.1747360
[3,] 0.6391715 0.21601933 0.9090187
> m2<-matrix(runif(9),3,3)
> m2
           [,1]      [,2]       [,3]
```

```
[1,] 0.89233081 0.8024568 0.85067247
[2,] 0.00822817 0.2333153 0.01118533
[3,] 0.07514774 0.3114626 0.89859468
> m1+m2
          [,1]      [,2]      [,3]
[1,] 1.0007971 0.8613498 1.5277172
[2,] 0.1106915 1.1559871 0.1859213
[3,] 0.7143192 0.5274819 1.8076134
> m1-m2
            [,1]        [,2]        [,3]
[1,] -0.78386457 -0.74356376 -0.17362778
[2,]  0.09423515  0.68935650  0.16355064
[3,]  0.56402373 -0.09544325  0.01042403
> m1 * m2
             [,1]       [,2]        [,3]
[1,] 0.0967877717 0.04725912 0.575943281
[2,] 0.0008430855 0.21527346 0.001954479
[3,] 0.0480322884 0.06728194 0.816839376
> m1^2
           [,1]        [,2]       [,3]
[1,] 0.01176493 0.003468391 0.45838951
[2,] 0.01049873 0.851323265 0.03053266
[3,] 0.40854017 0.046664353 0.82631502
> m1/m2
           [,1]       [,2]       [,3]
[1,]  0.1215538 0.07339092  0.7958935
[2,] 12.4527477 3.95461318 15.6218895
[3,]  8.5055320 0.69356431  1.0116004
> m1*m2^(-1)
           [,1]       [,2]       [,3]
[1,]  0.1215538 0.07339092  0.7958935
[2,] 12.4527477 3.95461318 15.6218895
[3,]  8.5055320 0.69356431  1.0116004
>
```

R では `1/m1` で行列 `m1` の逆行列 `m1^(-1)` が計算可能であるが,MATLAB で可能な `m1\1` の様な表記は出来ない.

第3章　入出力

3.1　ファイルからのデータ読込

ここでは最も単純な以下の表の読込から始めよう.

表 3.1: dog.data ファイルの内容

名前	性別	月齢	体重
ポチ	♂	6	2
エス	♂	7	5
タロウ	♂	5	2.4
クロミ	♀	6	1.6
マロン	♀	10	4
ナナ	♀	8	4.5

このファイルの内容の様な表は `read.table` 函数を用いて R に取込む.

```
> data1<-read.table("dog.data")
> data1
      V1   V2   V3   V4
1   名前 性別 月齢 体重
2   ポチ   ♂    6    2
3   エス   ♂    7    5
4 タロウ   ♂    5  2.4
5 クロミ   ♀    6  1.6
6 マロン   ♀   10    4
7   ナナ   ♀    8  4.5
>
```

R ではファイルにコメントが含まれている場合には, 読み飛ばす行数の指定や, データの区切が空行でない場合には, 区切文字の指定も行える.

第4章　CRANパッケージの追加

4.1　追加作業

Rには様々なパッケージが用意されている. 新規にRのパッケージを追加する場合,`install.packages`命令を用いる.

`install.packages`命令は, 利用者が計算機環境での権限の違いで, パッケージのインストール先を自動的に変更する. 則ち,Rをインストールしたディレクトリへの書き込みが可能な場合, Rをインストールしたディレクトリにパッケージがインストールされるが, それ以外の環境では利用者のホームディレクトリ上にRという名前のディレクトリを生成し, その中でインストール作業が行われる.

`instatt.packages`函数の構文は, 単純にパッケージ名のみを引数とする. 例えば,`biOps`パッケージをインストールする場合, `install.packages("biOps")`と入力する. するとRは4.1に示す様なTcl/Tkを用いたパッケージのミラー選択用のウィンドウを表示する.

図 4.1: CRANのミラー選択ウィンドウ

そこでミラーを選択すると,`install.packages`函数で指定したパッケージと, このパッケージの

利用に必要とされる他のパッケージの入手とコンパイルを行い, それから所定のディレクトリへのインストールが行われる. ここで root でインストールしなければ, 利用者のホームディレクトリ上に R のパッケージ用のディレクトリが生成され, そこにパッケージのインストールが実行される.

猶, インストール済のパッケージは通常はそのままで利用出来ない. 予め `library` 命令を用いてパッケージの読込を行わなければならない. この時, パッケージ名だけを指定すれば,R はパッケージの読込を行う.

索 引