R のデータ取り込み／出力 (R Data Import/Export)

Version 1.5.0 (2002-04-29)

R Development Core Team

日本語訳注 : この R-data.texi の日本語訳[1] は、英語原文と全く同じ条件の下で自由に配布、利用、修正可能である。R の開発の早さから、こうした文章の日本語訳は常に"旧式化"していることをお断りしておく。R の最新バージョン付属の文章を適宜参照されたい。R-data-jp.texi は GNU texinfo と呼ばれる計算機マニュアル専用の TEX の方言で書かれており、TEX でコンパイル[2] する。

ISBN 3-901167-53-6

[1] texinfo が日本語対応でないため完全には日本語化されていない。

[2] 日本語版は例えば日本語 TEX の ascii ptex を用いるなら、まず "ptex R-data-jp.texi"、次に索引作成のため "texindex R-data-jp.*"、そしてもう一度 "ptex R-data-jp.texi" と三段階でコンパイルする。もしかすると最後の二段階を複数繰り返す必要があるかも知れない。コンパイルには関連ファイル texindex.tex, R-defs.texi, version.texi が必要である。

Table of Contents

謝辞

このマニュアルのリレーショナルデータベイス部分は Douglas Bates と Saikat DebRoy によって書かれた古いマニュアルに部分的に依存している。このマニュアルの代表著者は Brian Ripley である。

多くのボランティアがここで用いられているパッケージに貢献している。説明されているパッケージの代表作者は以下の方々である。

CORBA	Duncan Temple Lang
e1071	Friedrich Leisch
foreign	Thomas Lumley, Saikat DebRoy, Douglas Bates and Duncan Murdoch
hdf5	Marcus Daniels
Java	John Chambers and Duncan Temple Lang
netCDF	Thomas Lumley
RmSQL	Torsten Hothorn
RMySQL	David James and Saikat DebRoy
RODBC	Michael Lapsley
RSPerl	Duncan Temple Lang
RPgSQL	Timothy Keitt
RSPython	Duncan Temple Lang

Brian Ripley is the author of the support for connections.

1 序

一つの統計解析システムにデータを読み込み、レポートを書くために結果を他のシステムに移し換えることは、統計解析自体よりもはるかに多くの時間を食いかねないいらだたしい作業である。多くの読者に取り解析自体が主要な関心の的なのであるが。

このマニュアルは R 自体、もしくは CRAN から入手可能なパッケージ経由で利用可能なデータの取り込みと移し換えの機能を説明する。説明されているパッケージの幾つかは依然として開発途中 (したがって CRAN) の Devel 領域にある) であるが、既に有用な機能を備えている。

特に説明が無い限り、このマニュアルで説明される全ては R の Unix/Linux そして Windows 版に当てはまる。(古い Macintosh 版についてはまだ多くが利用できない。)

一般に、R のような統計システムは大規模データの操作に特に適しているわけでは無い。幾つかの他のシステムはこの点で R より優れており、このマニュアルの主眼の一部は R で同じような機能を重複して用意するよりは、他のシステムが使えるようにすることにある！(例えば Therneau & Grambsch (2000) はデータの操作を SAS で行い、それから解析のために S の **survival5** を使用することを好むと注釈している。) 幾つかの最近のパッケージは `Java`, `perl` そして `python` といった言語で開発された機能を、直接に R と統合しでこれらの言語中の機能をより適切に使用することを可能にする。(**Java**, **RSPerl** そして **RSPython** パッケージを見よ。)

また S と同様に R は Unix の伝統である再使用可能な小規模ツールから成り立っており、`awk` と `perl` というツールを使って、取り込みと移し換えの前にデータを操作することが有用であることを思い出すのは価値があるであろう。Becker, Chambers & Wilks (1988, Chapter 9) 中の解析例はこの実例であり、そこでは Unix ツールが S に読み込む前にデータを検査し操作するために使われている。R 自体がこのアプローチを利用しており、R 自体よりは `perl` を使ってそのヘルプデータベースを操作し、関数 `read.fwf` は実行時に `perl` を利用しないことが決定されるまで `perl` スクリプトを呼び出す。伝統的な Unix のツールは今では、Windows を始めとし、より広く利用可能になっている。

1.1 取り込み

R にデータを取り込む最も簡単な形式はテキストファイルであり、これはしばしば小量か中程度の大きさの問題に適用できる。テキストファイルからの取り込みのための基本的関数は `scan` であり、これは Chapter 2 [表計算風データ], page 5 で議論されるより便利な関数の基礎である。

しかしながら、すべての統計コンサルタントは顧客から適当なバイナリー形式のデータをフロッピーディスクや CD-R で提供されることが普通であろう。例えば、Execl のスプレッドシートや SPSS ファイルである。しばしば最も簡単な方法は元のアプリケーションを用いてデータをテキストファイルに落す (そしてコンサルタントはそのための最も普通のアプリケーションのコピーを自分の計算機に用意しておく) ことであろう。しかしながら、これはいつでも可能なわけではなく、Chapter 3 [他の統計システムからのデータ取り込み], page 10 はそうしたファイルに R から直接アクセスするどのような機能があるかを論じている。

ある場合には、データはデータ量とアクセス速度のためにバイナリー形式で保管されている。たびたび遭遇する例は画像データであり、普通メモリー中の表現であるバイト列として保管され、ヘッダーが先頭にあるかも知れない。そうしたデータ形式は Chapter 5 [バイナリファイル], page 19 と Section 6.5 [バイナリ接続], page 23 中で議論される。

より大きなデータベースではデータをデータベース管理システム (DBMS) を用いて処理するのが普通である。DBMS を用いて平文ファイルを取り出すオプションがここでもあり得るが、多くのそうした DBMS に対しては R パッケージ See Chapter 4 [リレーショナルデータベース], page 11 か

ら直接抽出操作をすることができる。ネットワーク接続からのデータの取り込みは Chapter 7 [ネットワークインタフェイス], page 25 で議論される。

1.2 テキストファイルへの出力

R から結果を取り出すのは普通より簡単な作業であるが、それでも幾つかの落とし穴がある。目標のアプリケーションが念頭にあるとして、普通テキストファイルが最も便利な交換手段である。(もしバイナリーファイルが必要なら、Chapter 5 [バイナリファイル], page 19 を見よ。)

関数 `cat` がデータの取り出し操作の基礎にある。これは引数 `file` を取り、引数 `append` は一つのテキストファイルに `cat` の連続する呼出で書き込むことを可能にする。

最も普通の作業は行列やデータフレームをテキストに数値の長方形の格子として書き込むことである。行と列ラベルが加わるかも知れない。これは関数 `write.table` と `write` を用いて実行できる。関数 `write` は行列もしくはベクトルを単に指定された数の列 (そして行列の転置) に書き出す。関数 `write.table` はより便利であり、データフレーム (もしくはデータフレームに強制変換できるオブジェクト) を行と列ラベルと共に書き出す。

データフレームをテキストファイルに書き出す際留意すべき幾つかの点がある。

1. 精度

   これらの関数は `print` ではなく `cat` に基づいており、数値が出力される精度は現在の `options(digits)` の設定に支配される。精度を失わないためにはこれを増やす必要があるかも知れない。一層の制御のためには `format` をデータフレームに、ひょっとすると各列毎に、適用しよう。

2. ヘッダー行

   R はヘッダー行が空の行名の項目を持つことを好むので、ファイルは次のような体裁になる

   ```
                   dist    climb   time
   Greenmantle     2.5     650     16.083
      ...
   ```

   他のシステムには行名に対する (場合によると空の) 項目を必要とし、これは `col.names = NA` 引数を指定した `write.table` で行うことができる。Excel はそうしたシステムの一つである。

3. 欄分離記号

   ファイル中で使われる通常の欄分離記号はコンマであり、これは英語圏では如何なる欄にも現れそうもない記号だからである。そうしたファイルは CSV (comma separated values) ファイルと呼ばれる。或る種のロケールではコンマが小数点として使われており (これには `write.table` 中で `dec = ","` とすればよい) 、したがって CSV ファイルは欄分離記号としてセミコロンを使う。

   セミコロンかタブ (`sep = "\t"`) が恐らく一番安全なオプションであろう。

4. 欠損値

   既定では欠損値は `NA`として出力されるが、これは引数 `na` で変更できる。NaNs は `write.table` により `NA` として扱われるが、`cat` や `write` ではそうはならないことを注意しよう。

5. 引用文字列

   既定では文字列 (行と列の名前を含み) は引用符で囲まれる。引数 `quote` は引用符もじと因子引数を制御する。

   文字列が埋め込まれた引用符記号を含む際は少々注意がいる。三つの有用な書式は

```
> df <- data.frame(a = I("a \" quote"))
> write.table(df)
"a"
"1" "a \" quote"
> write.table(df, qmethod = "double")
"a"
"1" "a "" quote"
> write.table(df, quote = FALSE, sep = ",")
a
1,a " quote
```

二番目は表計算ソフトで通例のエスケープ書式である。

write.table はしばしば、過度に大量のメモリを使うので、非常に大きな行列を書き出す最良の方法では無いことを注意しよう。パッケージ **MASS** 中の関数 write.matrix は行列を書きだす特殊なインタフェイスを提供し、メモリ使用を減らすために行列をブロック毎に書き出すオプションを持っている。

sinkを用いて R の標準出力をファイルに落すことが可能であり、それにより print 文 (暗黙のそれを含み) の出力を取り込むことができる。これは常に最も効率的な方法とはいえず、options(width) の設定を増やす必要が有るかもしれない。

1.3 XML

テキストファイルからデータを読み込む際、そのファイルを作成した際の決りを知り、指定することはユーザーの責任である。つまり、@ ref テキストファイルへの出力 } で述べられた注釈文字、ヘッダー行の有無、値の分離規則、欠損値の表現等である。内容だけでなく、ファイルを自己記述的にする内容の構造までも記述することができ、データを読み込むソフトウェアにこれらの詳細を教える必要が無くなる。

拡張可能マークアップ言語 – 単に XML という名前で知られている – は、標準的なデータセットだけでなく、より複雑なデータ構造に対し、そうした構造を提供するのに使うことができる。XML は広く使われつつあり、一般的なデータのマークアップと交換の標準になりつつある。それは幾つかの団体により、地図等の地理データ、グラフィカル表現、数学等に使われている。

XML パッケージは R と S-PLUS の双方で XML 文書を読み書きする一般的な機能を提供し、この発展しつつある新しい技術を容易に使えるようになることを願っている。何人かの人々が、XML を (就中) 異なったアプリケーションを跨って共有可能なデータセットの表現にどのように使えるかを検討している。例えば、R と S-PLUS のオブジェクトを双方のシステムで共有できるように表現する方法、プロットを SVG (Scalable Vector Graphics, XML の方言) で表現する方法、関数のドキュメントを表現する方法、テキスト・データ・コードを含む “ライブ” 解析/報告を生成する方法等である。

XML パッケージの機能の説明はこの文章の枠を超える。詳細と例は ウェッブページ (http://www.omegahat.org/RSXML) を参照されたい。

2 表計算風データ

Section 1.2 [テキストファイルへの出力], page 3 でスプレッドシート風テキストファイルの幾つかの変種を見た。それらではデータは長方形の格子中に表現され、行・列ラベルを持つ可能性がある。この節ではそうしたファイルの R への読み込みを考える。

2.1 read.table の変種

関数 read.table は長方形格子状のデータを読み込む最も便利な方法である。多くの可能性があるため、既定の引数群を変更して read.table を呼び出す幾つかの関数がある。

read.table は非常に大きな数値行列を読み込むには非効率的な方法であることを注意しよう：以下の scan を見よ。

以下にあげるような考慮すべき点がある:

1. ヘッダー行

   header 引数を明確に指定することを推薦する。簡便のためにヘッダー行は行ラベルを持たず、列ラベルにたいする項目だけを持つ。従って、他の行より一つだけ欄が少ない。(もし R がこうした行を見付けたら header = TRUE を設定する。) もしファイルが行ラベルに対する (可能性として空の) ヘッダー欄をもつように表現されているならば、それを以下のように読み込む

   ```
   read.table("file.dat", header = TRUE, row.names = 1)
   ```

   列名は col.namesを持ちいて明確に与えることができる。明確な名前はヘッダー行 (もし存在すれば) を上書きする。

2. 分離記号

   普通ファイルを眺めることにより、使用されている欄分離記号が決定される。しかし、空白文字で分離されたファイルに対しては全ての空白 (空白文字、タブ、そして新行) を用いる sep = "", ある分離記号, sep = " " そして sep = "\t" を選択することができる。分離記号の選択は引用符付き文字列の入力に影響することを注意しよう。

   もし空の欄を含むタブで区切られたファイルの場合は sep = "\t" を必ず使おう。

3. 引用

   既定では文字列は '"' もしくは ''' で囲まれ、いずれの場合でも対応する引用符記号にマッチするまでの全ての文字が文字列の一部分として処理される。適正な引用符記号 (存在しない可能性がある) の集まりは quote 引数で制御できる。sep = "\n" に対しては既定は quote = "" に変更される。

   もし分離記号が指定されていなければ、引用は引用文字列中でそれらの直前に '\' (C-風) を置くことでエスケープシーケンス化できる。

   もし分離記号が指定されていれば、表計算ソフトで慣例のように、引用は引用文字列中でそれらを二重化することでエスケープシーケンス化できる。例えば

   ```
   'One string isn''t two',"one more"
   ```

   は次のように読み込むことができる

   ```
   read.table("testfile", sep = ",")
   ```

   これは既定の分離記号ではうまくいかない。

4. 欠損値

   既定ではファイルは欠損値を表す文字列 NAを含む。しかしこれは引数 na.strings で変更でき、これは欠損値を表す一つもしくはそれ以上の文字のベクトルを表す。

   数値列の空白欄はまた欠損値と見做される。

5. 未充足のライン

   スプレッドシートから読み込まれたファイルが引き続く空の欄 (そしてそれらの分離記号) を持つことは良くあることである。そうしたファイルを読むには `fill = TRUE` と設定する。

6. 文字欄中の空白スペース

   もし分離記号が指定されれば、文字欄の先頭及び引き続く空白スペースは欄の一部と見做される。空白を取り去るには、引数 `strip.white = TRUE` を使う。

7. 空のライン

   既定では `read.table` は空のラインを無視する。これは `blank.lines.skip = FALSE` と設定することで変更できる。これは恐らく `fill = TRUE` と一緒に使うときだけ有用であり、規則的なレイアウト中の欠損したデータを指示する。

8. 変数に対するクラス

   特別な行動を選択しない限り、`read.table` はデータフレーム中の各変数に対し適切なクラスを選択する。それは順に `logical`, `integer`, `numeric` そして `complex` を試み、項目が欠けておらず変換が不可能でも先に進む[1] もしこれらすべてが失敗すると、変数は因子に変換される。

   引数 `colClasses` と `as.is` はより広範囲の制御を提供する。`as.is` は文字ベクトルの因子 (だけ) への変換を抑制する。`colClasses` を使うと入力の各欄に対し好みのクラスをセットする事ができる。

   `colClasses` と `as.is` は変数毎ではなく、欄毎に指定されること、従って行名の欄 (もしあれば) を含むことを注意しよう。

9. 注釈

   既定では `read.table` は '#' を注釈文字として使い、もしこれが登場 (引用符付き文字列中を除き) すると行の残りは無視される。空白文字と注釈だけを含む行は空白行として扱われる。

   もしデータファイル中に注釈が無いことが予めわかるのであれば `comment.char = ""` を使用するのがより安全 (且つ高速) である。

便宜用の関数 `read.csv` と `read.delim` は英語圏のロケールを用いたスプレッドシートから導入された CSV とタブで分離されたファイルに都合の良い `read.table` への引数を提供する。変種 `read.csv2` と `read.delim2` はコンマが小数点として使われる国で使うのが適当である。

もし `read.table` へのオプションが不適切ならば、エラーメッセージは普通次のようになる

```
Error in scan(file = file, what = what, sep = sep, :
        line 1 did not have 5 elements
```

もしくは

```
Error in read.table("files.dat", header = TRUE) :
        more columns than column names
```

これは問題点を見付けるのに十分な情報をあたえるであろう。しかし付属の関数 `count.fields` はさらなる吟味に有用である。

大きなデータグリッドを読み込む際には効率性が重要になることがある。各欄に対し、`colClasses` をアトミックなベクトル型の一つ (logical, integer, numeric, complex or character) として指定し、読みこむべき行数 `nrows` を与える (適度に過剰な評価のほうが全く指定しないより好ましい) のが役にたつ。

[1] これは最初の項目を見さえすれば、ほとんどの可能性を除外するので普通高速である。

2.2 固定幅書式ファイル

データファイルが欄分離記号を持たず、あらかじめ指定された列による欄を持つことがある。これはパンチカードの時代には極めて普通のことであったし、今でもファイルを小さくするためにしばしば使われる。

関数 read.fwf はそうしたファイルを読み込む簡単な方法を提供し、欄の長さのベクトルを指定する。この関数はファイルの各行をまるまるメモリーに読み込み、結果の文字列を分離し、一時的なタブで区切られたファイルに書き出し、それから read.table を用いる。これは小さなファイルには十分であるが、より複雑な場合にはファイルを perl のような言語の機能を使い前処理することが望ましい。

2.3 scan を直接使う

read.table と read.fwf はともにファイルを読み込むために scan を用い、それから scan の結果処理する。これは極めて便利であるが、しばしば scan を直接使用する方が望ましい。

関数 scan は多くの引数を持ち、その多くは既に read.table によりカバーされている。最も重要な引数は what であり、これはファイルから読み込まれるべき変量のモードのリストを指定する。もしリストが名前を持てば、返り値のリストの成分にその名前が使われる。モードは数値、文字、もしくは複素数で良く、普通例で指定される。例えば、0, "" 又は 0i である。たとえば

```
cat("2 3 5 7", "11 13 17 19", file="ex.dat", sep="\n")
scan(file="ex.dat", what=list(x=0, y="", z=0), flush=TRUE)
```

は三つの成分をもつリストを返し、ファイル中の四番目の列は捨てられる。

後の処理のために全ての行を R に読み込みたければ、関数 readLines が用意されている。

scan の一つの通例の使用法は大きな数値行列を読み込むことである。ファイル 'matrix.dat' が丁度 200 x 2000 行列の数値を含むとしよう。そうすると次のようにすることができる

```
A <- matrix(scan("matrix.dat", n = 200*2000), 200, 2000, byrow = TRUE)
```

これに対するあるテストは 2 秒かかったが、一方

```
A <- as.matrix(read.table("matrix.dat"))
```

は 32 秒 (と多くのメモリ) かかった。(comment.char ="" を使用すると 3 秒節約なった。)

2.4 データの形式変更

しばしば表計算風データは各対象に対する共変量とそれに関する全ての観測値をあたえる簡潔な書式で与えられる。R のモデル化関数は観測値が単一の列で与えられることを必要とする。次の MRI による脳の繰り返し観測値の簡単な例を考えよう。

```
  Status   Age    V1     V2     V3    V4
       P 23646 45190  50333  55166 56271
      CC 26174 35535  38227  37911 41184
      CC 27723 25691  25712  26144 26398
      CC 27193 30949  29693  29754 30772
      CC 24370 50542  51966  54341 54273
      CC 28359 58591  58803  59435 61292
      CC 25136 45801  45389  47197 47126
```

二つの共変量と各対象に関する 4 つの観測値がある。このデータは Excel からファイル 'mr.csv' として読み込まれた。

これらのデータを操作して単一の反応ベクトルに変えるために stack を使うことができる。

```
zz <- read.csv("mr.csv", strip.white = TRUE)
zzz <- cbind(zz[gl(nrow(zz), 1, 4*nrow(zz)), 1:2], stack(zz[, 3:6]))
```

結果は次のようになる

```
      Status   Age values ind
X1         P 23646  45190  V1
X2        CC 26174  35535  V1
X3        CC 27723  25691  V1
X4        CC 27193  30949  V1
X5        CC 24370  50542  V1
X6        CC 28359  58591  V1
X7        CC 25136  45801  V1
X11        P 23646  50333  V2
...
```

関数 unstack は逆のことを行い、データを出力するのに使える。

これを行う別の方法は関数 reshape を次のようにつかうことである

```
> reshape(zz, idvar="id",timevar="var",
  varying=list(c("V1","V2","V3","V4")),direction="long")
    Status   Age var    V1 id
1.1      P 23646   1 45190  1
2.1     CC 26174   1 35535  2
3.1     CC 27723   1 25691  3
4.1     CC 27193   1 30949  4
5.1     CC 24370   1 50542  5
6.1     CC 28359   1 58591  6
7.1     CC 25136   1 45801  7
1.2      P 23646   2 50333  1
2.2     CC 26174   2 38227  2
...
```

reshape は stack よりもより複雑な構文を持つが、この例におけるように 'long' フォームが一つ以上の欄を持つ場合に使える。オプション direction="wide" を使うと、reshape はまた反対方向の変換を行うことが出来る。

2.5 フラットな分割表

多次元の分割表を配列形式で表現するのは多くの場合かなり不便である。カテゴリカルなデータの解析では、そうした情報はしばしばセル計数値に対応する因子水準の組合せを指定する説明のための行・列を持つ枠付の二次元配列の形で表現される。これらの行・列は普通ラベルが変わったときだけ表示されるという意味で "不揃い" であり、行は上から下へ、列は左から右へと読み取られるという当然の約束がある。R ではそうした "フラット (flat)" な分割表は ftable を用いて構成でき、適当な表示メソッドを持つクラス "ftable" のオブジェクトを与える。

簡単な例として、R の標準データ UCBAdmissions を考えよう。これは 1973 年度の UC Berkley 大学院の 6 つの主要学部での性別と入学合否による志願者の分類から得られた 3 次元の分割表である。

```
> data(UCBAdmissions)
> ftable(UCBAdmissions)
                Dept   A   B   C   D   E   F
Admit    Gender
```

```
Admitted Male         512 353 120 138  53  22
         Female        89  17 202 131  94  24
Rejected Male         313 207 205 279 138 351
         Female        19   8 391 244 299 317
```

表示された表現は明らかにデータを三次元分割表として表示するよりもより有用である。

ファイルからフラット風な表を読み取るために、もう一つの関数 read.ftable がある。これは、行・列名と水準が本当に持つ情報の違いを処理する追加の引数を持つ。read.ftable に対するヘルプページは幾つかの有用な例を持つ。フラットな表は as.table を用いて配列型の標準的な分割表に変換できる。

フラットな表はその “不揃いな” 行 (そしてひょっとすると列) ラベルの表現で特長づけられることを注意しよう。もし行変量の水準が欠けることなしに完全に与えられているならば、データの読み込みに代わりに read.table を使うべきであり、これから分割表を xtabs を用いて作ることができる。

3 他の統計システムからのデータ取り込み

この章では他の統計システムで作られたバイナリーデータを読み取る問題を考える。これは避けられれば避けるのが一番であるが、元のシステムが使えなければ避けられない。

3.1 EpiInfo, Minitab, S-PLUS, SAS, SPSS, Stata

お勧めのパッケージ **foreign** はこれらの統計システムで作成されたファイルの読み込みと、Stata への移入機能を提供する。ある場合には、これらの関数は `read.table` よりもはるかに少ないメモリしか必要としない。

Stata の '`.dta`' ファイルはバイナリファイル書式である。Stata の 5.0, 6.0 そして 7.0 版は関数 `read.dta` と `write.dta` を用いて読み書きできる。Stata の値ラベル月の変量はオプションで R 因子へ (そして R 因子から) 変換できる。

EpiInfo の 5 と 6 版はデータを自己記述的な固定幅テキスト書式で保管する。`read.epiinfo` はこれらの `.REC` ファイルを R のデータフレームに読み込む。

関数 `read.mtp` は 'Minitab Portable Worksheet' を読み込む。これはワークシートの成分を R リストとして読み込む。

関数 `read.xport` は SAS の Transport (XPORT) 書式のファイルを読み込み、データフレームのリストとして返す。もし SAS が使用中のシステムで使えるならば、関数 `read.ssd` を使って SAS スクリプトを作成・実行し、SAS の permanent データセット (`.ssd` もしくは `.sas7bdat`) を Transport 書式でセーブできる。それから結果のファイルを `read.xport` の呼び出しで読み込む。

関数 `read.spss` は SPSS の 'save' 命令と 'export' 命令で作成されたファイルを読み取ることが出来る。これは保存されたデータセット中の各変量毎に一つの成分を持つリストを返す。ラベルをもつ SPSS の変量はオプションで R の因子に変換できる。

関数 `read.S` は S-PLUS の 3.x, 4.x もしくは (32-bit) Unix または Windows 用の 2000 版で作成されたバイナリオブジェクトを読み込むことが出来る (そしてそれらを異なった OS 上で読むことが出来る)。これにより全部でははないが多くの S オブジェクトを読み込むことが出来る。特にベクトル、行列、そしてこれらを含むデータフレームやリストを読み込むことが出来る。

関数 `data.restore` は (Alpha マシーンからのダンプも読み込めるという点を除けば) 同じ制限の下で S-PLUS のデータダンプ (`data.dump` で作成された) を読み取ることが出来る。オプション `data.dump(oldStyle=T)` で書き込まれた S-PLUS の 5.x と 6.x 版も読み取れるはずである。

3.2 Octave

Octave は数値線形代数システムであり、パッケージ **e1071** 中の関数 `read.octave` は Octave の命令 `save -ascii` で生成された Octave ASCII データファイルからベクトルや行列を読み取ることが出来る。

4 リレーショナルデータベース

4.1 なぜデータベースを使うのか?

R がうまく処理できるデータのタイプには制限がある。R により操作される全てのデータはメモリー中に置かれており、関数の実行中にデータの幾つものコピーが作られるため、R は巨大なデータセットには向いていない。サイズが数 (十) メガバイトを超えるようなデータオブジェクトは R をメモリー不足にする可能性がある。

R は簡単なデータへの共有的なアクセスをサポートしていない。つまり、もし複数のユーザーが同じデータにアクセスし、もしかすると更新すれば、一人のユーザーが行った変更は他のユーザーには見えないであろう。

R は、あるセッションからデータオブジェクトやワークシートの全体を保存でき、以後のセッションでそれを読み戻せるという意味で、データの一貫性をサポートしている。しかしながら、保存されたデータの書式は R に固有のものであり、他のシステムでは簡単に処理できない。

データベース処理システム (DBMSs) そして特にリレーショナル DBMSs (RDBMSs) はこれら全てをうまく処理するようにデザインされている。それらの長所は

1. 大きなデータの選ばれた一部分に高速にアクセスすることを可能にする。
2. データベース中の項目を要約し、相互参照する強力な手段。
3. スプレッドシートや R のデータフレームよりもより組織的な方法でデータを保管できる。
4. 複数ホストで作業中の複数使用者からの共有的アクセスを許す一方、データへのアクセスに対するセキュリティー条件を強要できる。
5. 広範囲の使用者にサーバーとして振舞う能力。

DBMS が使われる統計的アプリケーションの種類としては、データの 10% を標本として抽出する、多次元の分割表を作り出すためにデータの相互参照を行う、別々の解析のためにデータベースからグループ毎にデータを取り出す、等が考えられる。

4.2 RDBMS の概観

T これまでに多くの (そして高価な) 商用 RDBMS、(Informix (`http://www.informix.com`); Oracle (`http://www.oracle.com`); Sysbase (`http://www.sysbase.com`); IBM's DB/2; Microsoft SQL Server on Windows) そして学術的で小規模のデータベース (例えば MySQL, PostgreSQL, Microsoft Access, . . .)、が存在しており、前者はデータのセキュリティーにより多くの強調が置かれている。両者の境界は曖昧になりつつあり、Open Source PostgreSQL はますます高機能化し、Linux にはフリー版の Informix, Oracle そして Sysbase がある。

その他にも良く使われているデータソースがあり、スプレッドシート、非リレーショナルデータベイス、そして (もしかすると圧縮された) テキストファイルまでもが含まれる。Open Database Connectivity (ODBC) はこれらのデータソース全てを用いる標準である。これは Windows (`http://www.microsoft.com/data/odbc/` を見よ) に起源を持つが、Linux にも移植されている。

この章の残りで説明される全てのパッケージは使用者間、そして使用者対サーバー型のデータベースを提供する。データベースは同じ機械に置くこともできるし、(よりしばしば) 離れた機械に置くこともできる。これらの DBMS が程度の差こそあれサポートする SQL (Structured Query Language, しばしば 'sequel' と発音される: Bowman *et al.* 1996 を見よ) と呼ばれるインタフェイス用の言語

に対する ISO 標準 (実際は複数: SQL-92 は ISO/IEC 9075 であり、ANSI X3.135-1992 としても知られている) がある。

4.2.1 SQL queries

より包括的な R のインタフェイスは通常の操作の背後で SQL を生成している。しかし、全体としての複雑な操作には SQL の直接的な使用が必要になる。普通 SQL は大文字で書かれるが、R のインタフェイス関数では小文字で書く方が便利であろう。

リレーショナル DBMS はデータを *tables* (又は *relations*) のデータベースとして扱う。これはかなり R のデータフレームに似ており、一つの型 (数値、文字、日時、通貨、. . .) の *columns* や *fields* から成り立っており、*rows* もしくは *records* が一つの項目に対する観測値を含んでいる。

SQL '問合わせ' はリレーショナルデータベースに対する極めて一般的な操作である。古典的な問合わせは次の形式の SELECT 文である。

```
SELECT State, Murder FROM USArrests WHERE rape > 30 ORDER BY Murder

SELECT t.sch, c.meanses, t.sex, t.achieve
  FROM student as t, school as c WHERE t.sch = c.id

SELECT sex, COUNT(*) FROM student GROUP BY sex

SELECT sch, AVG(sestat) FROM student GROUP BY sch LIMIT 10
```

これらの最初の例は R のデータフレーム `USArrests` から、データベースのテーブルを縦断してコピーされた二つの列を選び、第 3 の列を取り出し、結果をソートすることを要求している。第二の例はデータベース *join* の二つのテーブル `student` と `school` を操作し、四つの列を返す。第 3 と第 4 の問合わせはあるクロス作表を行い、数え上げもしくは平均を返す。(五つの集合演算は COUNT(*), SUM, MAX, MIN そして AVG であり、各々単一の列に適用される。)

SELECT 問合わせはテーブルを選択するため FROM を使用し、包合関係を指定するのに WHERE を使用し (もしくは AND や OR で分離された複数の条件)、結果をソートするのに ORDER BY を使用する。データフレームとは異なり、RDBMS テーブルの行は順序付けられていないと考えた方が良く、 ORDER BY 文無しでは順序は確定していない。一つ以上の列に関して、それらをカンマで分離することにより、(辞書式順序で) ソートすることができる。DESC を ORDER BY の後に置くことで逆順にソートすることができる。

SELECT DISTINCT 問合わせは選択されたテーブルから各々異なる行の一つのコピーだけを返す。

GROUP BY 節は基準に従い行のサブグループを選択する。もし複数のコラムが (コンマで区切って) 指定されると、五種類の一括化関数の一つを用いて多重クロス分類を要約することが出来る。一つの HAVING 節は一括値に依存してグループを含めたり、除外したりする選択を許す。

もし SELECT 文が 一意的な順序を生成する ORDER BY 文を含むと、LIMIT 節が出力行の引き続くブロックを (数で) 選択するために付け加えることが出来る。これは行を一時に一ブロックずつ検索するのに役に立つ。(問合わせを最適化するために LIMIT 節が使われるため、これは順序が一意的でないと信頼できないかも知れない。)

表を作る (CREATE TABLE, しかし普通はこれらのインタフェイスにおいてデータフレームをデータベースにコピーするであろう)、データを INSERT したり DELETE したり UPDATE するための問合わせがある。表は DROP TABLE '問合わせ' により破壊される。

Kline and Kline (2001) は SQL Server 2000, Oracle, MySQL そして PostgreSQL における SQL の実現方法に関し詳細に説明している。

4.2.2 データの型

データはデータベイス中に様々なデータタイプで保管できる。データの型は DBMS に依存するが、SQL の標準は多くの型を定義しており、次のような (しばしば SQL 名ではない) 広く使用可能なものを含む。

`float(`p`)` オプションで精度付の、実数。しばしば `real` と呼ばれる。

`integer` 32-ビット整数。しばしば `int` と呼ばれる。

`smallint` 16-ビット整数。

`character(`n`)`
固定長文字列。しばしば `char` と呼ばれる。

`character varying(`n`)`
可変長文字列。しばしば `varchar` と呼ばれる。

`boolean` 真もしくは偽。しばしば `bool` と呼ばれる。

`date` 暦日。

`time` 一日の時間。

`timestamp`
日付と時間。

`time` と `timestamp` には時間帯 `with timezone` を伴う変種がある。

より完備した R のインタフェイスパッケージはユーザーから型変換に関することがらを隠蔽する。

4.3 R のインタフェイスパッケージ

CRAN には R と DBMS との交信を助ける 4 つのパッケージがある。それらは互いに異なるレベルの抽象度を持つ。あるものはすべてのデータフレームをデータベースから、そしてデータベースへとコピーする手段を与える。全部がデータベースから SQL 問合わせによりデータを選択する関数を持っている。そして (**RmSQL** をのぞいて) 結果をデータフレーム全体として、また部分毎に (普通行の集りとして、しかし **RPgSQL** は列を検索できる) 検索できる。**RODBC** を除いて (現在のところ) 単一の DBMS を対象にしている。.

4.3.1 Package RPgSQL

`http://rpgsql.sourceforge.net/` と CRAN にあるパッケージ **RPgSQL** は PostgreSQL (`http://www.postgresql.org`) へのインタフェイスを提供する。

PostgreSQL はその開発者 (Momjian, 2000) によれば '最も進んだオープンソースのデータベースサーバー' である。これはほとんどの Unix 風の OS や Windows (Cygwin もしくは U/Win の使用下で) 稼働するようである。PostgreSQL は商用 RDBMS のほとんどの機能を持っている。

RPgSQL はこれらの RDBMS インタフェイス中最も成熟し完備したパッケージである。

RPgSQL を使うためには、最初にデータベースとの接続を `db.connect` を用いてひらく。(現在のところ一時には唯一つの接続が開ける。) 一旦接続が開かれると R のデータフレームを PostgreSQL テーブルに `db.write.table` を使ってコピー出来、`db.read.table` を使って PostgreSQL テーブルを R のデータフレームにコピー出来る。

RPgSQL は *proxy data frame* という面白い概念を持っている。data frame proxy は "data.frame" クラスを継承する R オブジェクトであるが、データを含まない。proxy data frame への全てのアクセスは適当な SQL 問合わせを発行し、データベースから結果のデータを検索する。proxy data frame は bind.db.proxy の呼び出しで設定される。proxy を取り除くには、単に bind.db.proxy が生成したオブジェクトを抹消すれば良い。

より細かい制御は SQL 問合わせを db.execute を介して PostgreSQL サーバーに送ることにより得られる。これは clear = TRUE (既定) でフラッシュされない限り結果を PostgreSQL の結果キャッシュに留めておく。一旦結果がキャッシュに入ると、db.fetch.result を使って全結果をデータフレームとして取り出すことが出来る。関数 db.result.columns や db.result.rows はキャッシュされたテーブル中のコラムと列を報告し、db.read.column は一つのコラムを (ベクトルとして) 取り出す。結果中の個別のセルは db.result.get.value で読み取ることが出来る。db.clear.result は結果のキャッシュを消去する。

一つの欠点は PostgreSQL は全てのテーブルとコラムの名前を小文字に変換し、従って最大の自由度を得るためには、R の名前として小文字だけを使う必要がある。関数 sql.insert と sql.select は INSERT と SELECT 問合わせに対する便宜用のラッパーを与える。

単純な例でこうした関数を試してみることが出来る。データベース 'testdb' が既に設定されており、PostgreSQL が単独稼働中の機械上で動いているとすると、接続のためにそれ以上の認証は不要のはずである。

```
> library(RPgSQL)
> db.connect(dbname="testdb")   # 必要に応じ認証を与える
Connected to database "testdb" on ""
> data(USArrests)
> usarrests <- USArrests
> names(usarrests) <- tolower(names(USArrests))
> db.write.table(USArrests, write.row.names = TRUE)
> db.write.table(usarrests, write.row.names = TRUE)
> rm(USArrests, usarrests)
## db.ls はデータベース中のテーブルの一覧を与える
> db.ls()
[1] "USArrests"  "usarrests"
> db.read.table("USArrests")
               Murder Assault UrbanPop Rape
Alabama          13.2     236       58 21.2
Alaska           10.0     263       48 44.5
   ...
## proxy data frame を設定する。USArrests は除去されていることを思い出そう
> bind.db.proxy("USArrests")
## USArrests はいまや proxy であり、従って全てのアクセスは database に対してな
される
> USArrests[, "Rape"]
   Rape
1  21.2
2  44.5
   ...
> rm(USArrests) # proxy を取り除く
> db.execute("SELECT rpgsql_row_names, murder FROM usarrests",
             "WHERE rape > 30 ORDER BY murder", clear=FALSE)
```

```
> db.fetch.result()
           murder
Colorado      7.9
Arizona       8.1
California    9.0
Alaska       10.0
New Mexico   11.4
Michigan     12.1
Nevada       12.2
Florida      15.4
> db.rm("USArrests", "usarrests") # 確認をスキップするために ask=FALSE を使
う
Destroy table USArrests? y
Destroy table usarrests? y
> db.ls()
character(0)
> db.disconnect()
```

write.row.names = TRUE を指定すれば、問合わせ中の欄を保存する限り、行名はデータベースのテーブル中の欄 rpgsql_row_names へマップされ、素直に復元されることを注意しよう。

RPgSQL はデータフレーム中の R のクラスと、PostgreSQL の型との間のマッピングを拡張する手段を提供する。

4.3.2 Package RODBC

CRAN にあるパッケージ **RODBC** は ODBC インタフェイスをサポートするデータベースソースへのインタフェイスである。これは非常に広く使われており、異なったデータベースシステムへ同じ R コードでアクセスすることが可能になる。**RODBC** は Linux と Windows 双方で稼働し、多くのデータベースシステムは ODBC のサポートを提供する。例えば、Windows 上のほとんど (例えば Microsoft Access)、MySQL、Oracle そして Unix/Linux 上の PostgreSQL がそうである。

unixODBC (http://www.unixODBC.org) または iOBDC (http://www.iODBC.org) といった Unix/Linux 用の ODBC Driver Manager と、あなたのデータベースシステム用のドライバーをインストールする必要がある。FreeODBC project (http://www.jepstone.net/FreeODBC/) は ODBC に関連した情報の宝庫である。

一群のインタフェイス関数が用意されている。odbc* グループは基本的な ODBC 関数への低水準インタフェイスを提供する。詳細はヘルプページ (?RODBC) を見よ。sql* グループは R のデータフレームと SQL テーブル間のインタフェイスを提供する。

最大で 16 個の接続が同時に可能である。一つの接続はデータベースへのそれ以降のアクセスに用いられるハンドルを返す odbcConnect の呼び出しにより開かれる。接続は odbcClose により閉じられる。接続中のテーブルに関する詳細は sqlTables を用いて得られる。

関数 sqlSave は一つの R のデータフレームをデータベース中のテーブルにコピーし、sqlFetch はデータベース中のテーブルを R のデータフレームにコピーする。

SQL の問合わせは sqlQuery を呼び出すことによりデータベースに送ることができる。これは結果を R のデータフレームとして返す。(sqlCopy は問合わせをデータベースに送り、結果をデータベース中のテーブルとして保管する。) より詳細な制御が、最初に odbcQuery を呼び出し、次ぎに結果を取り出すために sqlGetResults を呼び出すことにより得られる。後者はループ中で限られた数の列を一度に取り出すのに使える。sqlGetResults は規定ではデータフレームを返すが、生の結果を文字行列として得ることもできる。

以下は PostgreSQL を使った例で、ODBC ドライバが列とデータフレーム名を小文字に変換している。先に作ったデータベース testdb を用い、DSN (data source name) は '~/.odbc.ini' 中に unixODBC という名前である。MyODBC を用いた全く同じコードが Linux と Windows NT での MySQL データベースにアクセスするのに使えた (MySQL は同様に名前を小文字に移す)。Windows では DSN はコントロールパネル中の ODBC アプレットに設定される。

```
> library(RODBC)
## 名前を小文字に移すように告げる
> channel <- odbcConnect("testdb", uid="ripley", case="tolower")
## データフレームをデータベースに読み込む
> data(USArrests)
> sqlSave(channel, USArrests, rownames="state")
> rm(USArrests)
## データベース中のテーブルを一覧表示
> sqlTables(channel)
  TABLE_QUALIFIER TABLE_OWNER TABLE_NAME TABLE_TYPE REMARKS
1              NA          NA  usarrests      TABLE      NA
## そのリスト
> sqlFetch(channel, "USArrests", rownames = TRUE)
               murder assault urbanpop rape
Alabama          13.2     236       58 21.2
Alaska           10.0     263       48 44.5
    ...
## 一つの SQL 問合わせ、もともとは一つの行上にあった
> sqlQuery(channel, "select state, murder from USArrests
           where rape > 30 order by murder")
       state murder
1 Colorado      7.9
2 Arizona       8.1
3 California    9.0
4 Alaska       10.0
5 New Mexico   11.4
6 Michigan     12.1
7 Nevada       12.2
8 Florida      15.4
## テーブルを除く
> sqlDrop(channel, "usarrests")
## 接続を閉じる
> odbcClose(channel)
```

ODBC を Excel のスプレッドシートと一緒に使う簡単な例として、スプレッドシート 'bdr.xls' に対する DSN がコントロールパネルに設定してあると仮定しよう。そうするとスプレッドシートから次ぎのように読みこむことができる

```
> library(RODBC)
> channel <- odbcConnect("bdr.xls")
## スプレッドシートのリスト
> sqlTables(channel)
  TABLE_CAT TABLE_SCHEM        TABLE_NAME   TABLE_TYPE REMARKS
1 C:\\bdr            NA           Sheet1$ SYSTEM TABLE      NA
2 C:\\bdr            NA           Sheet2$ SYSTEM TABLE      NA
```

```
3 C:\\bdr             NA          Sheet3$ SYSTEM TABLE      NA
4 C:\\bdr             NA Sheet1$Print_Area        TABLE      NA
## シート1の内容を取り出す
> sh1 <- sqlQuery(channel, "select * from [Sheet1$]")
```

テーブルの指定が sqlTables が返すものと異なることを注意しよう。これは sqlFetch の使用を妨げる。

4.3.3 Package RMySQL

CRAN の開発部門にあるパッケージ **RMySQL** は MySQL データベース (http://www.mysql.com と Dubois, 2000 を見よ) へのインタフェイスを提供する。これは R から SQL を用いたリレーショナル DBMS へのアクセスに対する共通の API を提供しようというプロジェクトの一部である。現在のところ、これは **RPgSQL** や **RODBC** よりも低水準の機能を提供する。

MySQL は Unix/Linux と Windows 用がある。(その現在の状況は、ソースは自由に得られるが、Windows のバイナリ版の配布は商業的、であるように見える。) MySQL は '軽くてぜい肉の無い' データベースである。(これは OS が大・小文字を区別すれば名前の大・小文字を区別する。したがって Windows では区別しない。) パッケージ **RMySQL** は Linux と Windows の双方で動く。

関数 MySQL の呼び出しはデータベース接続管理オブジェクトを返し、それから dbConnect の呼び出しがデータベース接続を開き、それは総称的関数 close の呼び出しで閉じられる。

SQL 問合わせは quickSQL または dbExecStatement で行うことが出来る。quickSQL は問合わせを行い、結果をデータフレームとして回収する。dbExecStatement は問い合わせを送り、結果の回収に使えるクラス "MySQLResultSet" のオブジェクトを返し、ついで結果を閉じるのに使える。

関数 fetch は問合わせ結果中の一部もしくは全部の行をリストとして取り出すのに使われる。関数 hasCompleted は全ての行が取り出されたかどうかを指示し、getRowCount は結果中の行数を返す。

RMySQL の第 0.4 版から、データフレームをデータベースに書き込む assignTable、そして表をデータフレームとして回収する getTable という便利な関数がある。assignTable により行名が表に row_names という欄としてコピーされるが、それらは getTable により逆コピーされない。

```
> library(RMySQL)
## MySQL データベースへの接続を開く
> con <- dbConnect(MySQL(), dbname = "test")
## データベース中のテーブルを一覧する
> getTables(con)
## データフレームをデータベース中に読み込み、既存のコピーを削除
> data(USArrests)
> assignTable(con, "arrests", USArrests, overwrite = TRUE)
## 全ての表を得る
> getTable(con, "arrests")
  Murder Assault UrbanPop Rape      row_names
1   13.2     236       58 21.2        Alabama
2   10.0     263       48 44.5         Alaska
3    8.1     294       80 31.0        Arizona
...
## 読み込まれた表を選択する：全て同一行
> quickSQL(con, "select row_names, Murder from arrests
          where Rape > 30 order by Murder")
```

```
    row_names Murder
1   Colorado    7.9
2    Arizona    8.1
3 California    9.0
4     Alaska   10.0
5 New Mexico   11.4
6   Michigan   12.1
7     Nevada   12.2
8    Florida   15.4
> close(con)
```

4.3.4 パッケージ RmSQL

CRAN 中のパッケージ **RmSQL** は Mini SQL データベースシステム (mSQL としても知られている、`http://www.hughes.com.au`, Yarger *et al.*, 1999). へのインタフェイスを提供する。パッケージの付属文章は mSQL を次のように説明している

> mSQL は GPL ライセンスではないが大学と非商用的機関は無料で使用できる。

RmSQL はこの章で解説された最も基本的なインタフェイスを提供し、付加的な機能を持たない mSQL の C-API へのラッパーである。

データベースへの接続は最初ホストを `msqlConnect` で選び、それからデータベースを `msqlSelect` で選ぶ。接続は `msqlClose` の呼び出しで閉じる。SQL 問合わせは `msqlQuery` の呼び出しで発行され、結果は `msqlStoreResult` の呼び出しで保管される。問合わせが修了すると、結果は `msqlFreeResult` で開放される。

問合わせの結果が保管されると、値は `msqlFetchRow` を用い行毎に取り出せる。これは行位置を `msqlDataSeek` でリセットしない限り、行を順番に取り出す。`msqlNumRows` の呼び出しは結果中の全行数を与える。

基本インタフェイスが簡単な例を許さないためここには例をあげないが、パッケージの ‘`Example`’ ディレクトリには一つの例がある。

5 バイナリファイル

バイナリ接続 (Chapter 6 [接続], page 20) が今やバイナリファイルを扱う好ましい方法である。

5.1 バイナリデータ書式

CRAN の開発部門にあるパッケージ **hdf5** と **netCDF** はそれぞれ NASA の HDF5 (Hierarchical Data Format, `http://hdf.ncsa.uiuc.edu/HDF5/` を見よ) と UCAR の netCDF データファイル (network Common Data Form, `http://www.unidata.ucar.edu/packages/netcdf/` を見よ)、への実験的なインタフェイスを提供する。

これらはともに科学的なデータを配列指向の方法で保管するシステムで、既述、ラベル、書式、単位、. . . を含む。 HDF5 はまた配列のグループを許し、R インタフェイスはマップのリストを HDF5 groups に書き込み、数値・文字ベクトルや行列を書き込むことができる。

R インタフェイスは netCDF を読み取るだけで (現在のところ) 書き込みはできない。

6 接続

connection(接続) は R では Chambers (1998) の意味で使われ、ファイル名の使用をファイル類似のオブジェクトへのインタフェイスで置き換える一群の関数を意味する。

6.1 接続の種類

最も馴染のある接続のタイプはファイルであり、ファイル接続は関数 `file` で生成される。ファイル接続は (もし OS が特定のファイルに対しそれを許すならば) テキストもしくはバイナリモードで読み、書き、そして追記するために開くことができる。現在のところ一時には読み、書き、そして追記のいずれかだけのために開くことができ、R は読み書き双方のファイル位置を所持する。

既定では接続はそれが生成されたときには開かれていないことを注意しよう。規則はある接続を使用する関数は、もし接続がまだ開かれていなければ開く必要があり、もし自分が開いたならばその使用後に閉じなければならないということである。簡単にいえば、接続を最初と同じ状態に戻せということである。接続を明示的に開き、閉じるためのメソッドを持つ総称的関数 `open` と `close` がある。

`gzip` で使われるアルゴリズムで圧縮されたファイルは関数 `gzfile` で生成される接続で使うことが出来る。

Unix プログラマーは特殊ファイル `stdin`, `stdout` そして `stderr` を使うことになれている。これらは　R に端末接続 (*terminal connections*) という名前で存在する。これらは通常のファイルかも知れないが、GUI コンソールへの、そしてからの入出力にも使うことができるであろう。(標準的な Unix の R インタフェイスでも、`stdin` はファイルよりも `readline` から発行された行のことを意味する。)

これらの三つの端末接続は常に開いており、独自に開いたり閉じたりすることはできない。`stdout` と `stderr` は普通それぞれ通常の出力とエラーメッセージとして使われる。これらは普通同じ場所に行くが、通常の出力が `sink` の呼び出しでリダイレクト出来る一方、エラー出力は常に `stderr` へ送られる。ここで使われた言い回しに注意して欲しい：接続はリダイレクト出来ないが、出力は他の接続へと送ることが出来る。

テキスト接続 (*Text connections*) はもう一つの入力ソースである。これは R の文字列ベクトルをあたかも各行がテキストファイルから読み込んだかのように読み込ませる。テキスト接続は `textConnection` の呼び出しで生成し開かれ、生成時に文字列ベクトルの現在の内容を内部バッファにコピーする。

テキスト接続はまた R の出力を文字列ベクトルに変換するためにも使える。`textConnection` には新しい文字オブジェクトを作成するか、既存のものに追加するかを問合わせさせることが出来、いずれの場合もユーザーの作業スペースで行われる。この接続は `textConnection` の呼び出しで開かれ、いつでも全体の行出力を R のオブジェクトに行える。接続の閉鎖は全ての残余の出力を文字ベクトルの最終的な要素へと書き込む。

パイプ (*Pipes*) は他のプロセスへつながる特殊なファイル形式であり、全てのパイプ接続は関数 `pipe` (現在 Unix と `Rterm` の上にだけ移植されている) により生成される。パイプ接続を書き出し用に開くと (パイプへの追記は意味をなさない) OS 命令を実行し、その標準出力を R に繋げ、それからその接続に書き込む。逆に、パイプ接続を入力用に開くと、 OS 命令を実行し、その標準出力をその接続からの　R 入力として利用できるようにする。

`http://`, `ftp://` そして `file://` という型の URL は関数 `url` を使って読み取ることが出来る。簡便のために、`file` またこれらをファイル指定として受け入れ `url` を呼び出す。

ソケットは Berkley 風のソケット (ほとんどの Unix システム, Linux 及び Window、しかし現在古い Macintosh には無い) をサポートするプラットホームでは関数 `socketConnection` 経由で

接続として利用できる。ソケットは読み書き双方に使え、クライエントとサーバーの双方のソケットが使える。

6.2 接続への出力

既にファイルへの書き込み、もし引数 `append = TRUE` があればファイルへの追記、を行う関数 `cat`, `write`, `write.table` そして `sink` の説明を行ったが、これは R の 1.2.0 版以前の機能に付いてであった。

現在の動作は同値であるが、実際に行われることは `file` 引数が文字列ならばファイル接続が (書き込みや追記のために) 開かれ、関数呼び出しの終了とともに閉じられる、ということである。もし同じファイルに繰り返し書き込みたいのならば、明示的に接続を宣言して開き、出力関数の各呼び出しへ接続オブジェクトを渡す方がより効率的である。これは同様にパイプへの書き込みも可能にするが、これは前には `file = "|cmd"` という構文 (依然使用可能) により限られた方法でだけ移植されていたものである。

完全なテキスト行を接続に書き込む関数 `writeLines` がある。

幾つかの単純な例をあげる

```
zz <- file("ex.data", "w")   # 出力用接続を開く
cat("TITLE extra line", "2 3 5 7", "", "11 13 17",
    file = zz, sep = "\n")
cat("One more line\n", file = zz)
close(zz)

## 出力中の小数点をパイプを用いコンマに変える (Unix)
zz <- pipe(paste("sed s/\\./,/ >", "outfile"), "w")
cat(format(round(rnorm(100), 4)), sep = "\n", file = zz)
close(zz)
## そして出力ファイルを眺める:
file.show(outfile, delete.file = TRUE)

## R の出力を捉える: help(lm) からの例を使用
zz <- textConnection("ex.lm.out", "w")
sink(zz)
example(lm, prompt.echo = "> ")
sink()
close(zz)
## いまや‘ex.lm.out’は以後の処理用の出力を含む
## それを検査する、例えば、
cat(ex.lm.out, sep = "\n")
```

6.3 接続からの入力

接続から読み出しを行う基本関数は `scan` と `readLines` である。これらは文字列引数を取り、関数呼び出しの間中ファイル接続を開くが、明示的にファイル接続を開くとファイルを順番に異なった書式で読み込むことを可能にする。

`scan` を呼び出す他の関数も接続を利用することができる。特に `read.table` は接続を使用できるが、データを二回読み込まなければならないため中途探索不可能な接続では非効率的になる。(これはデータファイルを R の文字ベクトルに読み込み、それをテキスト接続で操作する。)

簡単な例は

```
## 直近の例で生成されたファイルから読み込む
readLines("ex.data")
unlink("ex.data")

## 現在のディレクトリのリストを読み込む (Unix)
readLines(pipe("ls -1"))

# 入力ファイルから後に続くコンマを取り除く
# Suppose we are given a file ‘data’ containing
450, 390, 467, 654,  30, 542, 334, 432, 421,
357, 497, 493, 550, 549, 467, 575, 578, 342,
446, 547, 534, 495, 979, 479
# そしてこれを次のように読み取る
scan(pipe("sed -e s/,$// data"), sep=",")
```

簡便性のため、もし `file` 引数が FTP もしくは HTTP URL を指定すれば、URL は `url` 経由で読み込みのために開かれる。ファイルを `file://foo.bar` 経由で指定することも許される。

6.3.1 プッシュバック

C プログラマーは恐らくテキスト入力ストリームへの文字のプッシュバックを行う `ungetc` 関数に馴染みがあるであろう。R 接続は同じアイデアを、(本質的に) 任意個数のテキスト行を `pushBack` の呼び出しで接続にプッシュバックすることができるという意味で、より強力な仕方で備えている。(訳注：プッシュバックとはあるストリームから一度読み込んだ文字列を、再度同じストリームから読み込めるように"元に戻す"操作。)

プッシュバックはスタックとして機能し、したがって読み込み要求は最初に最も最近にプッシュバックされたテキストの各行を使用し、それからより以前にプッシュバックされたものを利用し、最後に接続自体から読み込む。プッシュバックされた行が完全に読み込まれるやいなやそれは消去される。プッシュバックされた待機中の行数は `pushBackLength` の呼び出しで見出すことができる。

簡単な例でアイデアを確認できる。

```
> zz <- textConnection(LETTERS)
> readLines(zz, 2)
[1] "A" "B"
> scan(zz, "", 4)
Read 4 items
[1] "C" "D" "E" "F"
> pushBack(c("aa", "bb"), zz)
> scan(zz, "", 4)
Read 4 items
[1] "aa" "bb" "G"  "H"
> close(zz)
```

プッシュバックはテキストモードの入力用に開かれた接続に対してだけ利用可能である。

6.4 接続の一覧と操作

ユーザーが現在使える全ての開かれた接続の要約は `showConnections()` により得ることができ、閉じられた接続や端末接続を含む全ての接続の要約は `showConnections(all= TRUE)` で得ることができる。

総称的関数 `seek` は読みだし・書き込みに対する現在位置を読み取ったり、(いくつかの接続に対しては) リセットすることに使える。不幸にもこれは信頼できないかも知れない OS 機能 (例えば Windows でのテキストファイル) に依存している。関数 `isSeekable` は、もし `seek` がその引数で与えられた接続位置を変更すると報告を行う。

関数 `truncate` は書き込み用に開かれたファイルをその現在位置で切り詰める。これは `file` 接続だけで有効で、全てのプラットフォームに移植されてはいない。

6.5 バイナリ接続

関数 `readBin` と `writeBin` はバイナリ接続に対し読み書きを行う。接続はモード指定に `"b"` を加える、つまり読み込みには `"rb"` モード、書き込みには `"wb"` モード、もしくは (もし適切なら) `"ab"` モード、バイナリモードで開く。関数は次の引数を持つ

```
readBin(con, what, n = 1, size = NA, endian = .Platform$endian)
writeBin(object, con, size = NA, endian = .Platform$endian)
```

どちらの場合にも `con` は必要なら呼び出しの間開かれる接続であり、もし文字列が与えられるとファイル名を指定しているものと仮定される。

書き込みの方が少し簡単なので、先に説明をする。`object` はアトミックなオブジェクトでなければならない。つまり、`numeric`, `integer`, `logical`, `character` もしくは `complex` のベクトルで属性を持たない。既定ではこれはそれがメモリーに表現されているのと全く同じバイトの列として書き込まれる。

`readBin` はファイルからバイトの列を読み込み、`what` で与えられるモードのベクトルと解釈する。これは適当なモード (例えば `what=integer()`) のオブジェクトであるか、モードを記述する文字列 (先のパラグラフで与えられた 5 種類の一つか `"double"` もしくは `"int"`)。引数 `n` は接続から読み込まれるべきベクトル要素の最大数を指定する。もしそれより少なければ、短いベクトルが返される。引数 `signed` は読み込まれる 1 バイトもしくは 2 バイト整数が符号付き (既定) か符号無しの整数であることを指定する。

残る二つの引数はデータを他のプログラムもしくはプラットホームと交換して読み書きするために使われる。既定ではバイナリデータはメモリーから直接に接続に送られ、また送り込まれる。これはもしファイルが異なったアーキテクチャを持つ機械に送られる際には十分ではないが、ほとんど全ての R プラットホーム間では必要な変更はバイトの順序だけである。Intel の `ix86` に基づく機械、Compaq Alpha そして Vaxen は *little-endian* であるが、一方 Sun Sparc, mc680x0 シリーズ, IBM R6000, SGI そして他のほとんどは *big-endian* である。(ネットワークでのバイト順序は (XDR, eXternal Data Representation, を使うため) は big-endian である。)[1] 異なるプログラム間でデータをやりとりするためには更に考慮すべきことがある。例えば、16-ビット整数の読み込みや、単精度数の書き込みである。これは `size` 引数を用いてなされ、これは (普通) 整数に対してはサイズ 1, 2, 4, 8 を許し、実数に対してはサイズ 4, 8 そしてもしかすると 12 や 16 を許す。異なったサイズでの転送は精度を損なうかも知れなく、`NA` を含むベクトルを転送することは避けるべきである。

文字列は C の書式で読み書きされる、つまりゼロバイトで終るバイトの文字列である。関数 `readChar` と `writeChar` を使うとより自由度が得られる。

---

[1] 訳注：little-endian と big-endian は 1 バイト中の 8 ビット列の先頭、末尾どちらを上位と考えるかによる二つの方式。異なる方式の機械で記録されたファイルをそのままバイナリ読みこみすると文字化けする。

6.5.1 特殊値

関数 readBin と writeBin は欠損値と特殊値を通すが、もしサイズの変更が含まれる場合は行うべきでない。

Functions readBin and writeBin will pass missing and special values, although this should not be attempted if a size change is involved.

R の論理型と整数型に対する欠損値は INT_MIN で表され、C のヘッダファイル 'limits.h' で定義された表現可能な最小 int であり、普通ビットパターン 0xffffffff に対応する。

R の数値型と副素数型に対する特殊値の表現は機械依存であり、もしかするとまたコンパイラー依存かも知れない。これらを扱う最も容易な方法は外部アプリケーションを、倍精度定数 NA_REAL, R_PosInf そして R_NegInf を書式出力する標準的な Rmath ライブラリにリンクし、マクロ ISNAN と R_FINITE を定義するヘッダー 'Rmath.h' を取り込むことである。

もしこれが不可能なら、全ての通常のプラットフォームでは IEC 60559 (又の名を IEEE754) 算術演算が使われているので、恭順的な C 機能を用いて Inf, -Inf そして NaN 値を検査したり、設定することが出来る。そうしたプラットフォームでは NA は下位ワード 0x7a2 (十進法で 1954) を持つ NaN 値で表現される。

7 ネットワークインタフェイス

ネットワーク経由でデータの交換を行ういくつかの限られた低水準機能がある。

7.1 ソケットからの読み込み

基本の R は BSD ソケットをサポートするシステム (R の通常の Linux, Unix そして Windows への移植を含む) において、 BSD ソケット経由で交信を行う機能を備えている。ソケットを利用する際の潜在的な問題は、これらの機能がしばしば保安上の理由で阻害されていたり、ウェッブキャッシュの強制使用をしなければならないことで、したがってこれらの関数は外部的に使うよりはインターネット上で使う方が有用であろう。

当初の低水準インタフェイスは関数 `make.socket`, `read.socket`, `write.socket` で `close.socket` 与えられる。.

7.2 `download.file` の使用

関数 `download.file` は FTP や HTTP 経由でウェッブリソースからファイルを読み取るために提供されている。`read.table` と `scan` は URL から、`url` を使って接続を明示的に開くか、または URL を `file` 引数として間接的に与えることにより直接に読み込むことが出来るため、しばしばこれは避けることが出来る。

7.3 DCOM インタフェイス

DCOM は、場合によれば異なった計算機上の、異なったプログラム間で通信を行う Windows のプロトコルである。CRAN の Software->Other->Non-standar から入手できる Thomas Baier の `StatConnector` プログラムはプロキシ DLL に対するインタフェイスであり、Windows 版の R に含まれており DCOM サーバーを作る。これは単純なオブジェクト (ベクトルと行列) を R へ、そして R から送ることや R への命令を発行することが出来る。

このプログラムは Visual Basic のデモとして作られており、CRAN の同じ領域には Erich Neuwirth による Excel 用の plug-in がある。このインタフェイスは、クライエントであるのが他のアプリケーション (Excel や Visual Basic で書かれたもの) であり、R がサーバーであるという意味で、ここで取り上げた多くの物と異なった方向を目指している。

7.4 CORBA インタフェイス

CORBA (Common Object Request Broker Architecture) は DCOM に似ているが、アプリケーションが、もしかすると異なった言語でプログラムされたり異なった計算機で稼働している、他のアプリケーション中で実行中のサーバーオブジェクトのメソッドや操作を呼び出すことを許す。Omegahat Project (`http://www.omegahat.org/RSCORBA`) から入手可能なパッケージ **CORBA** Omegahat Project (`http://www.omegahat.org/RSCORBA`) があり、現在 Unix 版が利用できるが Windows 版は利用できないように見える。

このパッケージは R の命令を利用可能な CORBA サーバーで利用できるようにし、それらが提供するメソッドを問合わせ、そしてこれらのオブジェクト中のメソッドを動的に起動する。これらの呼び出しで引数として与えられた R 値は呼び出し中に同梱され、その操作の起動において利用できるようにする。基本的なデータタイプ (ベクトルとリスト) は既定で同梱されるが、より複雑なオブジェクトは参照により引き渡される。これを用いた例としては Gnumeric (`http://www.gnumeric.org`)

スプレッドシートとの通信や、データ視覚化システム ggobi (http://www.ggobi.org) との交信がある。

CORBA サーバーを R 中に作ることが出来、他のアプリケーションがこれらのメソッドを呼び出すことを可能にする。例えば、特定のデータセットやある R のモデリングソフトウェアへのアクセスを提供出来るかも知れない。これは R のデータオブジェクトと関数を結びつけることにより動的に行われる。R から明示的にデータや機能を輸出することを可能にする。

また **CORBA** パッケージを R における分散、並行計算を実現するために使うことが出来る。一つの R セッションはマネージャーとして機能し、他の実働 R セッション中で稼働している異なったサーバーに仕事を分担させることが出来る。これは R 中で非同期的、又はバックグラウンドでの COBRA 呼び出しを起動する能力を利用する。より多くの情報は Omegahat Project (http://www.omegahat.org/RSCORBA) から得られる。

Appendix A 参考文献

R. A. Becker, J. M. Chambers and A. R. Wilks (1988) *The New S Language. A Programming Environment for Data Analysis and Graphics.* Wadsworth & Brooks/Cole.

J. Bowman, S. Emberson and M. Darnovsky (1996) *The Practical* SQL *Handbook. Using Structured Query Language.* Addison-Wesley.

J. M. Chambers (1998) *Programming with Data. A Guide to the S Language.* Springer-Verlag.

P. Dubois (2000) *MySQL.* New Riders.

M. Henning and S. Vinoski (1999) *Advanced CORBA Programming with C++.* Addison-Wesley.

K. Kline and D. Kline (2001) *SQL in a Nutshell.* O'Reilly.

B. Momjian (2000) *PostgreSQL: Introduction and Concepts.* Addison-Wesley. Also downloadable at `http://www.postgresql.org/docs/awbook.html`.

T. M. Therneau and P. M. Grambsch (2000) *Modeling Survival Data. Extending the Cox Model.* Springer-Verlag.

E. J. Yarger, G. Reese and T. King (1999) *MySQL & mSQL.* O'Reilly.

関数と変数の索引

概念索引