R のインストールと管理

Version 1.5.0 (2002-04-29)

R Development Core Team

ISBN 3-901167-52-8

日本語訳注:この R-admin.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-admin-jp.v15.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。

[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-admin-jp.v15.texi"、次に索引作成のため "texindex R-admin-jp.*"、そしてもう一度 "ptex R-admin-jp.v15.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(texinfo 付属スタイルファイル), R-defs.texi, version.texi が必要である。

Table of Contents

1 R の入手法

R のソース、バイナリ、そしてドキュメントは CRAN, “Comprehensive R Archive Network” から入手可能である。CRAN に関する情報は R の配布物中のファイル ‘RESOURCES’ を見よ。

1.1 ソースを入手し展開する

最も簡単な方法は最新の ‘R-*x*.*y*.*z*.tgz’ ファイルをダウンロードし、それを GNU tar がインストールされたシステム上で次の命令で解凍・展開することである

```
tar xvfz R-x.y.z.tgz
```

他のシステムでは gzip プログラムがインストールされていることが最低条件となる。そうすると次の命令が使える

```
gzip -dc R-x.y.z.tgz | tar xvf -
```

もしフロッピーディスクからソースファイルを導入する必要があるのならば、‘R-*x*.*y*.*z*.tgz-split.*’ ファイルをダウンロードし、それらを導入箇所で次のように連結する (Unix の場合)

```
cat R-x.y.z-split.* > R-x.y.z.tgz
```

それから先に述べたように作業する。もしあるグループのユーザー全員が使えるように構築したいのならば、目標のグループがファイルを読めるように解凍前に umask を設定する (例えば、全てのユーザーが使えるようにするには umask 022 とする)。

最後に、直近のリリース版に対す公開パッチ (z != 0 である ‘*x*.*y*.*z*’) は ‘R-*x*.*y*.{*z-1*}-*x*.*y*.*z*.diff.gz’ (例えば ‘R-1.2.2-1.2.3.diff.gz’) という名前で入手可能で、これは ‘.tgz’ ファイルよりもかなり小さなファイルである。そうしたファイルを直前のバージョンに適用するには、その先頭ディレクトリに移り次のようにする

```
gzip -dc /path/to/it/R-x.y.{z-1}-x.y.z.diff.gz | patch -E -p1
```

この手順はもしより古いソースが変更されている場合 (例えば、それらのディレクトリで作業する等) は必ずしもうまくいかないかも知れないことを注意しよう。

1.2 rsync を使う

ソースは又匿名 rsync で得ることも出来る。命令

```
rsync -rC rsync.r-project.org::module R
```

を使い、現在のディレクトリのサブディレクトリ ‘R’ 中に *module* という名前のソースツリーのコピーを作る。ここで *module* は R の既存の 4 種類のお好み版の一つを特定し、‘r-release’ (現在のリリース版), ‘r-patched’ (パッチの当たったリリース版)、そして and ‘r-devel’ (開発途上版、より不安定)、そして ‘r-ng’ (時世代、不安定) のいずれかである。rsync のツリーは主 CVS アーカイブから直接作り出され、毎時間更新されている。rsync 命令の ‘-C’ オプションは CVS ディレクトリを避けるようにする。rsync に関する情報は http://rsync.samba.org/rsync/ にある。

2 Unix へ R をインストールする

R は ‘i386-freebsd’, ‘i386-linux-gnu’, ‘i386-sun-solaris’, ‘powerpc-linux-gnu’, ‘powerpc-apple-darwin’, ‘mips-sgi-irix’, ‘alpha-linux-gnu’, ‘alpha-dec-osf4’, ‘rs6000-ibm-aix’, ‘hppa-hp-hpux’, ‘sparc-linux-gnu’ そして ‘sparc-sun-solaris’ を含む、いくつかの Unix タイプのプラットフォームにおいてソースからコンフィギュア・構築可能である。

更に、バイナリの配布が最も普通の Linux ディストリビューション、OSF/Tru64 の動く Compaq Alpha システム、そして X11 を備えた MacOS X (Darwin)、に対して利用できる。これらはプラットフォーム特有の流儀でインストールされる。したがってこの章の残りではソースからの構築についてだけ考える。

2.1 単純なコンパイル

最初に Appendix A [本質的なプログラムと有用なその他のプログラム], page 9 中の本質的、そして有用なツールとライブラリを概観し、必要なもの、もしくは必要なものをインストールしよう。

先ず R ツリーをインストールする場所を選ぶ (R は単なるバイナリではなく、付加的なデータセット、ヘルプファイル、フォント形状情報等を持つ)。この場所を *R_HOME* と呼ぼう。ソースコードを tar で展開する。これは ‘src’, ‘doc’ そしていくつかの他のディレクトリを作り出す。次の命令を実行する：

```
./configure
make
```

(もし使用している make が ‘make’ という名前でなければ Section B.3 [make を使う], page 12 を見よ。)

それから構築したシステムが正しく動作するか、次の命令で検査する

```
make check
```

失敗は関連機能の欠如から引き起こされただけかも知れず、必ずしも問題とはいえない。しかし、報告される矛盾は注意深く吟味すべきである。検査を再実行するには次のようにする必要があるかも知れない

```
make check FORCE=FORCE
```

もしこれらの命令が成功裡に実行されれば、R のバイナリーが ‘*R_HOME*/bin’ ディレクトリにコピーされる。さらに、シェルスクリプトからなる ‘R’ という名前の前処理プログラムが作成され、同じディレクトリにコピーされる。このスクリプトはユーザーがそれを起動できる場所、例えば ‘/usr/local/bin/R’ にコピーすることが出来る。又 man ページ ‘R.1’ を使用している man 命令がそれを見付けることが出来る場所、例えば ‘/usr/local/man/man1’、にコピーしても良い。もし R ツリーの全体を例えば ‘/usr/local/lib/R’ にコピーしたければ Section 2.3 [インストール], page 3 を見よ。注意：R をインストールする必要は無い。それが構築された場所においたまま起動することが可能である。

R はそのソースの最上位ディレクトリ (例えば ‘*TOP_SRCDIR*’) で構築する必要は無い。‘*BUILDDIR*’ に構築したければ、以下で更に説明されるように

```
cd BUILDDIR
TOP_SRCDIR/configure
make
```

のように実行する、云々。これはソースツリーを常にきれいにしておける利点がある。(このためには GNU make がいるかもしれない。)

Make は又 R オブジェクトの文章の平文のヘルプページだけでなく、HTML と LaTeX 版も作ることが出来る (この三種類はまた別個に `make help`, `make html` そして `make latex` で作成できる)。Perl の第 5 版が必要なことを注意しよう。もしこれが使用システムで利用できなければ CRAN から文章の PDF 版を入手できる。

もし必要なら `rehash` を行い *R* とタイプする。そして the R のマニュアルと R FAQ (ファイル 'FAQ' 又は 'doc/html/faq.html', もしくは常に最新版がある `http://www.ci.tuwien.ac.at/~hornik/R/R-FAQ.html`) を読もう。

2.2 マニュアルを作成する

ソースから構築できる一揃いのマニュアルがある、

'refman' すべてのヘルプページの印刷可能版。

'R-FAQ' R FAQ (既に構築されている)。

'R-intro' "An Introduction to R".

'R-data' "R Data Import/Export".

'R-admin' "R インストール and Administration", このマニュアル。

'R-exts' "Writing R Extensions".

'R-lang' "The R Language Definition".

これらを作るためには、次を実行する

```
make dvi      to create DVI versions
make pdf      to create PDF versions
make info     to create info files (not 'refman').
```

第 4 版、もしくはそれ以上の `makeinfo` がインストールされていなければ info ファイルを構築出来ないかも知れない (ある種の Linux ディストリブーションは 3.12 版を持つ)。

DVI 版は `xdvi` や `dvips` といった標準プログラムを利用してプレビュー・印刷が可能である。PDF 版は Acrobat Reader や (最新版の) ghostscript を使って見ることが出来、Acrobat Reader 内から辿れるハイパーリンクを持つ。info ファイルは Emacs や標準の GNU info プログラムで読むのに適している。

2.3 インストール

命令

```
./configure
make
make check
```

が成功裡に完了したら、完全な R ツリーを自分のシステムに次のようにタイプしてインストールできる

```
make install
```

これは次のディレクトリへインストールを行う：

'*prefix*/bin'

前処理シェルスクリプト

'*prefix*/man/man1'
: マンページ

'*prefix*/lib/R'
: その他すべて (ライブラリ、オンラインヘルプシステム、. . .)

ここで　*prefix* はコンフィグレーションの最中に決定され (典型的には '/usr/local')、configure をオプション '--prefix' 付きで次のように実行することで指定できる

```
./configure --prefix=/where/you/want/R/to/go
```

これは make install に R の実行ファイルを '/where/you/want/R/to/go/bin' にインストールさせる、等。インストールディレクトリの接頭辞は configure の最後に表示されるステータスメッセージ中に見ることが出来る。他のディレクトリにインストールしたければ次のようにする

```
make prefix=/path/to/here install
```

マニュアルの DVI, info そして PDF 版をインストールするには、次の命令から必要なものだけ実行する。

```
make install-dvi
make install-info
make install-pdf
```

インストールされたツリーが適正なユーザーグループから使用できるようにするためには、ソースを展開する前と構築最中に umask を適当 (恐らく '022') に設定する。

3 Windows へ R をインストールする

CRAN サイトの 'bin/windows' ディレクトリには (少なくとも) Windows 95, 98, NT4, 2000 そして ME で動く基本ディストリブーションのバイナリと CRAN からの多数のアドオンパッケージがある。

これらの Windows 版の一つが必要になるかもしれない : Windows 3.11+win32s では動作しないであろう。

使用システムでは長いファイル名の利用ができなければならない (恐らくネットワークにマウントされたシステムを除けばそうなっているであろう)。

最も簡単な方法は 'SetupR.exe' もしくは 'miniR.exe' を使うことである。単にアイコンをダブルクリックし、指示に従えば良い。もし R をこのようにインストールすれば、コントロールパネルやスタートメニューからそれを取り除くことができる (R に対するグループの作成を抑制しない限り)。

より詳しいことは R Windows FAQ (http://www.stats.ox.ac.uk/pub/R/rw-FAQ.html) を見よ。

3.1 ソースからの構築

もしソースから Windows 用に構築したければ、ソース配布物中のファイル 'src/gnuwin32/INSTALL' を見よう。多数のツールのセットを集め、インストールし検査する必要があるであろう: そうしたものの現在の所在に付いては http://www.stats.ox.ac.uk/pub/Rtools/ を見よ。

大・小文字を区別するファイルシステムの下でコンパイルする必要があるかも知れない。samba でマウントされたファイルシステム (全てのファイル名を小文字に変換する) では問題があることが分かっている。命令ウィンドウを パスが空白文字を含まない ディレクトリで開き、次のような命令を実行する

```
tar zxvf R-1.5.0.tgz
cd R-1.5.0\src\gnuwin32
make
```

そして座って待とう (高速ローカルディスクを持つ 1GHz PIII 計算機で約 5 分かかる)。

文章の作成法やクロスコンパイルを含むこれ以上の詳細に付いては 'src/gnuwin32/INSTALL' を見よ。

4 古い MacOS へ R をインストールする

CRAN サイトの ‘bin/macos’ ディレクトリには、MacOS 8.6 ~ MacOS 9.1 又はネイティブ MacOS X で実行可能な基本配布物と大量のアドオンパッケージのアーカイブの bin-hex 化された (‘hqx’) ものと stuffit (‘sit’) 版がある。単にこれらのアーカイブの一つを Aladdin Stuffit Expander (tm) 等の標準機能を使い適当なフォルダーに取り出せば良い。

同様にこの文章では Unix の変種と考えられている MacOS X 用のポートがある。それは CRAN サイトの ‘bin/macosx’ ディレクトリにある。

5 アドオンパッケージ

この章の内容はは R の Unix 風そして Windows 版に当てはまるが、古い MacOS 版には適用できない。

正確な用語を使うことが役に立つ。*package* は *library* から関数 `library()` により読み込まれる。従ってライブラリはインストール済みのパッケージを含むディレクトリの事である。主ライブラリは '*R_HOME*/`library`' であるが、他のも使える。例えば、環境変数 `R_LIBS` を設定するか、R の関数 `.libPaths()` を使う。

5.1 パッケージのインストール

ソースパッケージをインストールするには Perl の 5.005 版もしくはそれ以降がインストールされていることが必要である。

パッケージがインストールされるライブラリを明示的もしくは暗黙のうちに指定する必要があることを注意しよう。これはもちろんもし一つ以上のライブラリを持つ場合にだけ問題になる。

Unix でソースからパッケージをインストールするには次を使う

```
R CMD INSTALL -l /path/to/library pkg1 pkg2 ...
```

'`-l /path/to/library`' の部分は、もし設定されていれば `R_LIBS` 中の最初のライブラリを使うのなら、不要である。さもなければ、主ライブラリ '*R_HOME*/`library`' が使われる。

Windows では[1]

```
Rcmd INSTALL -l /path/to/library pkg1 pkg2 ...
```

または、パッケージは R の内部からダウンロードしインストールできる。最初にオプション `CRAN` を例えば一番近い CRAN のミラーサイトに設定する。

```
> options(CRAN = "http://cran.us.r-project.org/")
```

それからパッケージ **foo** をダウンロードしインストールする

```
> install.packages("foo")
```

ライブラリが指定されていれば (argument `lib`) ライブラリ検索パス中の最初のライブラリが使われる。

これが何をするかは Unix と Windows では異なる。Unix では CRAN にある利用可能なソースパッケージをダウンロードし、最新版の **foo** のソースをダウンロードし、そしてそれをインストールする (`R CMD INSTALL` を使い)。Windows ではパッケージの *binary* 版のリストを眺め、最新の版 (もしあれば) をダウンロードする。

Windows では `install.packages` はまた引数 `CRAN` を `NULL` に設定することにより、ローカルの '`zip`' ファイルからインストールできる。`RGui.exe` は GUI インタフェイスを備えた `install.packages`, `update.packages` そして `library` へのメニュー `Packages` を持っている。

[1] ソースコードパッケージを既にインストールしているならば

5.2 パッケージの更新

命令 `update.packages()` は使用システムにあるすべてのパッケージを更新することを保証する最も簡単な方法である。前節と同様にオプション `CRAN` を設定しよう。`update.packages()` は利用可能なパッケージのリストとそれらの現在バージョンをダウンロードし、それらをインストールされているもの比較し、CRAN にあるより最新の版の取り込みインストールを提案する。

パッケージを最新の物に更新するもう一つの方法が命令 `packageStatus()` で提供される。これは, すべてのインストール済みパッケージと複数の保管場所 (CRAN, ローカルなアーカイブ, . . .) にある利用可能なパッケージに関する情報を持つオブジェクトを返す。`print` と `summary` メッソドはインストール済みと利用可能なパッケージに対する概観を与え、`upgrade` メソッドは古いパッケージの取得とインストールを提案する。これは R に、CRAN のミラーサイトだけでなく、複数の保管場所からパッケージを取得し、それらすべてと同期し、将来の R の版の既定のパッケージマネージャとなることが予定されている。

5.3 パッケージを取り除く

パッケージは幾つかの方法で取り除くことができる。命令プロンプトから次のようにして取り除くことができる

R CMD REMOVE -l /path/to/library *pkg1 pkg2* ...

(Unix) または

Rcmd REMOVE -l /path/to/library *pkg1 pkg2* ...

(Windows)。

実行中の R プロセスからは次のようにして取り除く

```
> remove.packages(c("pkg1", "pkg2"),
                  lib = file.path("path", "to", "library"))
```

最後に、多くのインストールではライブラリから単にパッケージディレクトリを除けばよい。

Note: 現在パッケージを束にして一緒にインストールする事ができるが、それらを上記のように除くことはできない。束中のパッケージは個別に除く必要がある。

Appendix A 本質的なプログラムと有用なその他のプログラム

この付録では Unix 風のプラットフォーム上で R を構築するのに必要なプログラムの詳細を与える。それらが存在すれば `configure` により R が利用する。

A.1 本質的なプログラム

C と FORTRAN 77 (⟨undefined⟩ [Using FORTRAN], page ⟨undefined⟩ を見よ) でコンパイルできる必要がある。幾つかのアドオンパッケージはまた C++ コンパイラーを必要とする。

オンラインドキュメントを構築するには Perl の 5.004 版もしくはそれ以降が必要である。これは `http://www.perl.com/CPAN/` から手にはいる。

`makeinfo` の第 4 版もしくはそれ以降がなければインフォファイルを構築できない (そして幾つかの Linux 配布物は第 3.12 版を持つ)。

印刷用のドキュメントは `tex` と `latex`, 又は `pdftex` と `pdflatex` を必要とする。

A.2 有用なライブラリとプログラム

コマンド行編集機能は GNU のいずれのミラーサイトから入手可能な `readline` ライブラリに依存する。かなり新しい版が必要かも知れない。

`gzfile` 接続の使用は `zlib` (第 1.1.3 版もしくはそれ以降) を必要とする。もしインストールされた版がなければ R ソース中のものがコンパイルされ使用される。

ビットマップグラフィックス用のデバイス `jpeg()` と `png()` は適当なライブラリがインストールされていることを必要とする;それぞれ `jpeg` (第 6b 版もしくはそれ以降) もしくは `libpng` (第 1.0.5 版から 1.2.1 版) そして `zlib` (第 1.1.3 版もしくはそれ以降)。`libpng-1.2.2` のヘッダファイルは居場所を変えてしまったので、それを利用するには `CPPFLAGS` に '`-I/usr/local/include/libpng12`' といった指定をする必要があるかも知れない。

`bitmap` そして `dev2bitmap` デバイスは ghostscript (`http://www.cs.wisc.edu/~ghost`) を利用する。`bzfile` connections make use of `libbz2`, part of `bzip2` (`http://sources.redhat.com/bzip2`).

`bzfile` 接続は `bzip2` (`http://sources.redhat.com/bzip2`) の一部である `libbz2` を利用する。

A.2.1 Tcl/Tk

tcltk パッケージは Tcl/Tk がインストールされていることを必要とする。ソースは `http://dev.scriptics.com/` にある。Tcl/Tk ファイルの位置を指定するためには次のようなコンフィギュレーションオプションが必要になるかも知れない

'`--with-tcltk`'
: use Tcl/Tk, or specify its ライブラリ ディレクトリ

'`--with-tcl-config=`*TCL_CONFIG*'
: specify location of '`tclConfig.sh`'

'`--with-tk-config=`*TK_CONFIG*'
: specify location of '`tkConfig.sh`'

もしくはコンフィギュレーション用変数 `TCLTK_LIBS` と `TCLTK_CPPFLAGS` を使って、それぞれ Tcl と Tk ライブラリに対するリンク、'`tcl.h`' と '`tk.h`' ヘッダー位置の指定、を行うのに必要なフラグを指定する。

A.2.2 線形代数

R の線形代数ルーティンは拡張された BLAS (BasicLinear Algebra Subprograms, `http://www.netlib.org/blas/faq.html`) ルーティンを使用できる。幾つかはコンパイラーシステム固有 (Sun Sparc[1]、IBM 用の `libessl`) であるが、ATLAS (`http://math-atlas.sourceforge.net/`) は "最適化" された BLAS で、Unix 風の広範囲のプラットフォームで利用できる。もしこれ例外のライブラリが見付からないと、ライブラリパス中の `libblas` ライブラリが使用される。コンフィギュレーションのオプション '`--with-blas`' を指定することにより、特定の BLAS ライブラリを使い、外部の BLAS ライブラリを使わないように指示できる。

マルチプロセッサシステムに対しては、原理的には ATLAS の multi-threaded 版を使うことが出来る。現在のところこれはサポートされていない。問題は multi-thread の ATLAS 計算を行う際、送られた `SIGINT` シグナルが適正に処理できないことにあり、結果として segmentation fault を引き起こす可能性がある。thread 対応ライブラリをサポートするのには R の内部機構の変更が必要となる。できれば将来の版で対応したい。

BLAS ライブラリは R だけでなく幾つかのアドオンパッケージも使用する。これは共有 BLAS ライブラリを使う事が好ましいことを意味する、なぜなら静的ライブラリは R の実行プログラムと各 BLAS を使うパッケージ中にコンパイルされるからである。

BLAS の倍精度と倍精度複素数版が必要になるが、単精度版と単精度複素数版は必要無い。

LAPACK の最適化版があるが、効率化の程度は小さいと考えられるので R で使用する予定は無い。

すべてのライブラリと同様、R が相互に矛盾の無いコンパイラーとフラグでコンパイルされていることが必要になる。例えば、これは `libsunperf` を使うためには Sun Sparc の固有コンパイラでフラグ '`-dalign`' を使う必要があることを意味する。

'調整済み' の BLAS である ATLAS はまた Windows でも使うことができる：必要な事項については '`src/gnuwin32/INSTALL`' を見よ。

Unix (Windows では異なる) では、もし R が既定でない BLAS に対してコンパイルされると、すべての BLAS 使用パッケージも同じようにコンパイルされる必要があることを注意しよう。従って、もし R が ATLAS のインストール後に再構築されるのなら、**quantreg** といったパッケージも再インストールされる必要がある。

[1] SunPro `cc` と `f95` コンパイラを使用する `libsunperf`

Appendix B Unix に対するコンフィギュレーション

B.1 コンフィギュレーションにおけるオプション

configure は多くのオプションをもつ：次を実行するとその一覧が得られる

```
./configure --help
```

恐らく他の場所で触れられない最も重要なものは以下である (括弧内が既定値)

'--with-x'
: X Window System を使う

'--x-includes=*DIR*'
: X のインクルードファイルは *DIR* にある

'--x-libraries=*DIR*'
: X ライブラリは *DIR* にある

'--with-readline'
: (もし利用可能なら) readline ライブラリを使う [yes]

'--enable-R-profiling'
: Rprof() をサポートするようにコンパイル [yes]

'--enable-R-shlib'
: R を共有ライブラリとして構築する [no]

否定オプション用に '--without-foo' もしくは '--disable-foo' を使うことができる。

もしプロファイル機能の付いた実行プログラム (例えば '-pg) オプション使用'を持つ R を構築したければ '--disable-R-profiling' を指定する。

フラグ '--enable-R-shlib' は make が R を共有ライブラリ、普通 'libR.so' という名前、として構築するのでコンパイルに長時間かかり、従って R を内部的に使うプログラムを使用したいときにだけ使った方が良い。

B.2 コンフィギュレーションに関する変数

もしあるコンフィギュア用変数をそれらの既定以外の値に設定したい必要があれば、ファイル 'config.site' (設定したいかもしれないすべての変数を説明) を編集するか、コマンド行上で次の様にする

```
./configure VAR=value
```

これらの変数は環境に反映されないと言う意味で 貴重 であり、コマンド行から指示されなくてもキャッシュに保管され、(キャッシュが使われる限り) 引き続くコンフィギュレーション実行間で矛盾が無いように検査され、例えいかなるキャッシュが使われない場合も、コマンド行引数として渡されない場合も自動的な再コンフィギュレーションの間保存される。

これらの変数すべての一覧は configure --help の変数出力部分を見よ。

よく変更される変数は R_PAPERSIZE で、既定値は 'a4' であり 'letter' では無い。

例えばシステム外ディレクトリにある GNU realine のようなライブラリとヘッダファイルを使うなら、その位置を指定するためにそれぞれ変数 LDFLAGS (ライブラリ用、リンカに引き渡すならフラグ '-L' を使う) と CPPFLAGS (ヘッダファイル用、'-I' フラグを使い C/C++ プロセッサーに引

き渡す) を使う。もしライブラリが依然見つからなければ、恐らく使用しているコンパイラ/リンカーがフラグ '-L' と '-l' の再配置をサポートしていないかもしれない (このことは自前 cc を使った HP-UX で報告されている)。この場合は異なったコンパイラを使おう (または再配置を行うフロントエンドシェルスクリプトを使う)。

もしコンフィギュレーション用の変数を変更する必要があることがわかったら、ある種の設定はファイル 'config.cache' にキャッシュされているかもしれないことを留意することが重要であり、再コンフィギュレーション前にそのファイル (もしあれば) を取り除くのは良い考えである。キャッシュ動作は既定で停止されている ; キャッシュ動作を行うようにするにはコマンド行オプション '--config-cache' (または '-C') を使う。

B.3 make を使う

R をコンパイルするには GNU make を使うのが恐らく最も容易であることに気づくであろう。特に Solaris 2.6/7/8 では、GNU make の第 3.77 版以外が必要になる。3.79 版は Sun の make と同様にうまく動く。固有の make は SGI Irix 6.5 では不具合が生じると報告されている。

別のディレクトリ中に構築するには VPATH 変数を使う make が必要になる。例えば GNU make や、Solaris 2.7/8 (それ以前はまずい) の Sun make がその例である。

もし make を別の名前、例えば使用する GNU make が 'gmake' という名前を持てば、例えば次の例のように変数 MAKE をコンフィギュレーション時に設定する必要がある

```
./configure MAKE=gmake
```

B.4 FORTRAN を使う

R をコンパイルするためには FORTRAN コンパイラーもしくは FORTRAN から C への変換プログラム f2c (http://www.netlib.org/f2c) が必要になる。既定では g77, f77, xlf,cf77, cft77, pgf77, fl32,af77, fort77, f90, xlf90,pgf90, epcf90, f95, xlf95,lf95, g95, and fc を (この順で)[1] 探し、それから f2c を探し、どれでも最初に見つかった物を使う。もしどれも見つからなければ R はコンパイルできない。検索機構は、それぞれ FORTRAN 77 と FORTRAN から C への変換プログラムを実行する命令を指定するコンフィギュア用の変数 F77 と F2C を設定することで変えることができる。もし F77 があればそれが FORTRAN のコンパイルに使われる。さもなければ、もし F2C があれば、f2c が別の FORTRAN コンパイラーがあっても使われる。もし使用する FORTRAN コンパイラーが標準的でない場所にあれば、configure を実行する前に環境変数 PATH をそれにあわせて設定しなければならない。さもなければコンフィギュレーション用変数 F77 をその完全なパスを指定するために使う。

もし使用している FORTRAN ライブラリーが少々変わった場所にあれば、全てのライブラリーが見付かるように LD_LIBRARY_PATH もしくは使用しているシステムでの代替変数を注意すべきである。

FORTRAN の integer が C の int ポインターに、FORTRAN の double precision が C の double ポインターに同値になるように、必要な全てのコンパイル用フラグ (もしあれば) を設定する必要がある。これはコンフィギュレーション過程の間にチェックされる。

[1] HPUX 上では fort77 は POSIX は素直な FORTRAN コンパイラーで、検索リストの二番目に現れる。

FORTRAN コードのあるものは COMPLEX*16 変数を使っているかも知れない。これは FORTRAN 90 の拡張仕様である。これはコンフィグレーションの最中[2] にチェックされる。しかし FORTRAN 77 準拠を保証するためにコンパイル用フラグ[3] を使うのは避けた方が良いかもしれない。

パフォーマンスを重視[4] するなら FORTRAN 90/95 コンパイラーを使う。

もし f2c を使うなら、FORTRAN の型 integer が C の型 int に変換されることが保証されることを保証する必要がある。普通 'f2c.h' は 'typedef long int integer;' を含み、これは 32-ビットのプラットフォームでは動作するが、64-ビットのプラットフォームでは動作しない。

B.5 コンパイルとロード用のフラグ

多義に渉るフラグをファイル 'config.site' やコンフィギュレーション変数、そして命令行として設定出来る。既に次のものを解説した

CPPFLAGS
: ヘッダーファイル検索ディレクトリ ('-I') とその他の雑多な C と C++ プリプロセッサーやコンパイラー用のオプション

LDFLAGS
: パス ('-L'), ストリッピング ('-s') そしてその他の雑多なリンカー用オプション

その他として

CFLAGS
: C 用のデバッグ、最適化フラグ

MAIN_CFLAGS
: 主プログラムのコンパイル用の同様のもの

SHLIB_CFLAGS
: 共有ライブラリ用

FFLAGS
: FORTRAN 用のデバッグ、最適化フラグ

MAIN_FFLAGS
: 主プログラムのコンパイル用の同様のもの

SHLIB_FFLAGS
: 共有ライブラリ用

MAIN_LDFLAGS
: 主リンク用の追加フラグ

SHLIB_LDFLAGS
: 共有ライブラリのリンク用の追加フラグ

LDFLAGS 中の '-L/lib/path' として指定されるライブラリパスは一緒にされ、LD_LIBRARY_PATH (もしくは使用しているシステムの同値物) の前に追加され、したがって '-R' や '-rpath' フラグの必要は無くなる。

プロファイル機能付きの R のコンパイルには、'-pg' が位置独立なコードと一緒に使えないプラットフォームでは例えば 'MAIN_CFLAGS=-pg', 'MAIN_FFLAGS=-pg', 'MAIN_LDFLAGS=-pg' を使いたくなるかも知れない。

注意: 使用するライブラリと両立するように CFLAGS と FFLAGS を設定する必要があるかも知れない。一つの可能性のある問題点は倍精度実数の並べ方であり、今一つは構造体が引き渡される仕方である。

---

[2] 'R_ext/Complex.h' 中で定義されたその同値物 Rcomplex 構造と同様に

[3] 特に g77 の '-pedantic' は避けよう。これは理解困難なエラーメーッセージを与える。

[4] 例えば Sun/Sparc 上で最適化された BLAS を使うため

B.6 GNOME へのインタフェイスの構築

このインタフェイスは実験的であり不完全である。これはコンソールと `gtk()` と `gnome()` という名前のグラフィックスデバイスを提供する。コンソールは基本的な行命令編集と履歴機構とともに、ある種の R 命令へのマウスを使ったインタフェイスを提供する。コンソールの多くの機能は現在のところ競合する。`gtk()` グラフィックスデバイスは GDK (the GIMP Drawing Kit) への `x11()` の入口である。`gnome()` デバイスは GNOME のキャンバスを使う。

実験的な性格から R への GNOME インタフェイスは時動的には構築されない。オプション '`--with-gnome`' を用いてコンフィギュレーションプログラムを実行するように指定する必要がある。例えば次のようにする

```
./configure --with-gnome
```

しかしながら全ての要求を最初にチェックする必要がある。R への GNOME インタフェイスは現在 GNOME 1.4 版に対して開発中であり、それ以前の版では動く可能性は保証されない。したがって、少なくとも次のライブラリがインストールされている必要がある。

```
audiofile-0.2.1
esound-0.2.23
glib-1.2.10
gtk+-1.2.10
imlib-1.9.10
ORBit-0.5.12
gnome-libs-1.4.1.2
libxml-1.8.16
libglade-0.17
```

GNOME のデスクトップ環境が完全にインストールされていることが望ましい。もし Linux を使っているのなら、これらは使用している配布物中に提供されているはずである。加えて、ほとんどの通常の Linux 配布と Solaris 用の GNOEM のバイナリー配布が `http://www.ximian.com` から入手可能である。

幾つかのパッケージ管理システム (RPM や deb) はユーザー版のパッケージと開発者版のパッケージを区別することを覚えておこう。後者は同じ名前だが、拡張子 '`-devel`' を持つ。もし GNOME のパッケージ前のバージョンを使うのなら、R-GNOME インタフェイスをコンパイルするためには開発者版を使うべきである。

コンフィグレーション用の GNOME 用オプションの完全なリストは次の通りである

'`--with-gnome`'
: GNOME を使用、またはそのプレフィックスを指定する [no]

'`--with-gnome-includes=`*DIR*'
: GNOME ヘッダーの位置を指定

'`--with-gnome-libs=`*DIR*'
: GNOME ライブラリの位置を指定

'`--with-libglade-config=`*LIBGLADE_CONFIG*'
: `libglade-config` の位置を指定

B.7 プラットフォームに関する注意

この節では異なった Unix 風のプラットフォーム上での R の構築に関する幾つかの注意を与える。これらの注意はそれぞれ特定のコンパイラーと支援ライブラリの組を用いて一つもしくは二つのシス

テム上で行ったテスト用実行に基づくものである。R の構築の成功は適切なインストールと支援ソフトウェアの機能に依存するので、もし他の版のコンパイラと支援ライブラリを用いると結果は異なるかも知れない。

B.7.1 MacOS X

MacOS X 上に R を Unix アップリケーションとして構築することが出来る。DevTools, `f2c` もしくは g77、そして `dlcompat` ライブラリが必要である。同様に X サブシステムがインストールされているか、オプション '`--without-x`' を用いてコンフィギュレーションする必要がある。

f2c, g77, the `dlcompat` ライブラリ、そして X サーバーと支援ライブラリは Fink プロジェクト (`http://fink.sourceforge.net`) から得られる。これを書いている時点では Fink バイナリー配布には `f2c` と g77 が無く、直接インストールする必要がある。例えば g77 に対しては次のようにする

```
fink install g77
```

Fink は '`/sw`' 中にインストールされることを好むので、`CPPFLAGS` と `LDFLAGS` はそのように設定される必要がある。もし、やはり Fink から入手できる Tcl/Tk を使いたければ、Tcl と Tk の位置を指定するため `configure` オプションも使う必要があるであろう。

B.7.2 Sparc 上の Solaris

R は gcc/g77 そして SunPro WorkShop 6 を用いて Solaris 2.7 (別名、Solaris 7、SunOS 5.7) 上で構築できることが確認されている。2.7 版以前ではソースツリー中以外の場所 (恐らくその場所ですら) に構築するには GNU `make` が必要である。

もし `gcc` を使うならば、コンパイラーが現在使用中の Solaris 用にコンパイルされていることを確認しよう。(これは `gcc -v` で確認出来る。) `gcc` はある種のヘッダファイルの修正されたバージョンを作成し、従って (例えば) Solaris 2.6 でコンパイルされた `gcc` は Solaris 2.7 では R をコンパイル出来ない。

もし SunPro コンパイラーを使うのならば、オプション '`-fast`' を指定してはならない。なぜならこれは IEEE 算術規則を無効にし、`make check` を失敗させる。適用可能と知られている最適化オプションは次のものである。

```
-xlibmil -xO5 -dalign
```

`gcc` と `cc` の間には効率性にはほとんど差が無いが SunPro Fortran コンパイラを使うと相当の効果がある。`gcc/f77` の組合せはうまく働く。

Solaris 上で 64-ビット版でコンパイルする (これには UltraSparc チップとそれを利用可能にする OS が必要になる) には '`config.site`' 中に次を置く

```
CC="cc -xarch=v9"
CFLAGS="-xO5 -xlibmil -dalign"
F77="f95 -xarch=v9"
FFLAGS="-xO5 -xlibmil -dalign"
CXX=CC
CXXFLAGS="-xO5 -xlibmil -dalign -xarch=v9"
```

`f95` を使うと Sun performance ライブラリを選ぶことが可能になることを注意しよう。これは `f77` や g77 とでは動作しない。

`configure` が見出したライブラリが R の実行プログラムやモヂュールと両立することを保証するにはいくらかの注意がいる。なぜなら、テストの過程は可能な問題の多くを検出しないであろうか

らである。cc を用いた 32-ビット版ではフラグ '-dalign' が幾つかの Sun ライブラリに対して必要になる。幸いにも gcc に対する同値なフラグ '-munaligned-doubles' が既定値である。理論的には libpng, libjpeg, zlib そして ATLAS ライブラリといったライブラリはフラグ pic または PIC とともに構築される必要があり、これは静的なライブラリが使われているときは問題を引き起こしかねない。実際にはこれは 32-ビット用の構築にはほとんど影響を与えないが、64-ビット用構築では本質的に見える。

64-ビット用構築には 64-ビット版のライブラリが必要になる。コンフィギュレーションの過程は既定では LDFLAGS を '-L/usr/local/lib' に設定するので、32-ビット用のアドオンライブラリを探してしまうことを避けるにはそれをリセットする必要があるかも知れない。

B.7.3 HPUX

R は固有のコンパイラーと gcc の双方を用いて HPUX 10.2 版と HPUX 11.0 版へと構築されている。しかしながら 10.2 版は R 1.4.0 版以来テストされていない。既定では、R は (もし利用可能ならば) HPUX 用の gcc と g77 を用いてコンフィギュレーションされる。g77 によるインストールの幾つかは、それのファイルが位置独立コード (PIC) を生成するための適当なフラグとともにコンパイルされていないので、共有ライブラリにリンク出来ない g2c ライブラリの静的バージョンだけをインストールする。このため make はリンカーエラーを起こし、固有の cc PIC フラグである '+z' もしくは '+Z' を用いてコンパイルすべきだと文句を言う。この場合は g77 のインストールを修正するか、または以下のようにコンフィギュレーションを行い

```
F77=fort77
```

POSIX に従順な固有の FORTRAN 77 コンパイラーを使うことを指示する。

もしかすると configure が R が共有ライブラリとして使うが静的ライブラリとしてしか利用できない BLAS のような他のライブラリを検出するかも知れない。もし共有版をインストール出来なければ、これらのライブラリを使わないよう configure に指示する必要がある。

gcc の幾つかのバージョンは最適化レベル '-O2' が引き起こすバグを含んでいる可能性があり、次のようになり

```
> 2 %/% 2
[1] 1
> 1:2 %/% 2
[1] 0 0      # wrong!!
```

make check が失敗する。このときは使用する最適化レベル '-O' を指定するために CFLAGS を使用すべきである。

HPUX 11.0 版を使用しているある種のシステムは HPUX 10.2 版でインストールされた gcc を持っているかも知れない。バージョン 10.2 と 11.0 の HPUX 間では IEEE 算術に対する支援関数が IEEE 標準の推薦関数から、C9x ドラフト標準のそれへと変更されている。特に、これは finite が isfinite に変更されていることを意味する。11.0 上で動いている HPUX 10.2 版に対してコンフィグレーションされている gcc は isfinite を検出できず、結果として configure は計算機が IEEE 算術を完全にサポートしていると認識できず、C コードをコンパイルするときに IEEE_754 を定義できない。このため make check が失敗する。最良の解決策は適切にコンフィギュレーションされた gcc をインストールすることである。もう一つの便法は '-DIEEE_754' を CFLAGS 変数に加えることである。

R は固有の cc と fort77 の双方を使って次のようにコンフィギュレーション出来る

```
./configure CC=cc F77=fort77
```

f90 は普通非標準的なディレクトリ (例えば '/opt/fortran90/lib') に存在する静的な 'libF90.a' に対するリンクに固執する。したがって f90 を使うためにはこのディレクトリをコンフィギュレーション用変数 LDFLAGS を用いてリンカーのパスに加える必要がある (例えば ./configure F77=f90 LDFLAGS=/opt/fortran90/lib)。

B.7.4 IRIX

R を 32-ビット実行ファイルに対する gcc/f77 や cc/f77、64-ビット実行ファイルに対する固有のコンパイラーを用いて IRIX64 6.5 版上で構築することに成功している。次の命令

```
./configure CC="cc -64" F77="f77 -64" --with-tcltk=no
```

が 64-ビット実行ファイルを作成するのに使われた。configure が 32-ビット版を見付けだし、それが 64-ビット版と不整合であることを検出しないため、明示的に Tcl/Tk の不使用を宣言する必要があった。

gcc/g77 を用いた 32-ビット用構築は make check をパスするが、make test-all-extras は複素数用 LAPACK で失敗した。

B.7.5 Alpha/OSF1

R は gcc/g77 と cc/f77 を用い OSF1 V4.0 が動いている Alpha 機で構築に成功している。cc と g77 の混用はコンフィギュレーションに失敗した。固有の blas は SIGFPE を抑制するために必要なフラグとともに構築されていない陽なので、 configure のオプション '--without-blas' を使用した。現在のところ R は IEEE 算術をサポートするプラットフォーム上では SIGFPE に対する信号ハンドラーを設定しないので、これらは致命的になる。

make check passes with no problems.

B.7.6 Alpha/FreeBSD

R を FreeBSD 4.3 版の仮想する Alpha 上で構築することは部分的にしか成功していない。CFLAGS と FFLAGS の双方に '-mieee' を加えたコンフィギュレーションは成功するが、検査は SIGFPE で失敗する。'-mieee' はこれらを完全に抑制する代わりに延期させるだけのように見える。このポートを如何にして完成されるかに付いてのアドバイスは大いに歓迎されるであろう。

B.7.7 AIX

AIX 4.3.3 と AIX 5.1 では "実行時リンク" (普通の AIX 風のリンクに対し) が必要であった。このために、R の主プログラムは '-brtl' リンカーオプションを用いて実行時リンカーにリンクされなければならなく、共有オブジェクトはリンカーオプション '-G' を用いて実行時リンクが出来るようにしなければならない。これらのオプション無しでは AIX リンカーは '.so' 拡張子をもつ如何なる共有オブジェクトとも自動的にはリンクされないであろう。同様に、R の主プログラムは dlopen 呼び出しを持つロードモジュール (X11 のような) を動的に読み込み出来ないであろう。

MAIN_LDFLAGS と SHLIB_LDFLAGS をそれに応じて設定する際は、リンカーフラグは gcc をリンク用に用いた際は '-Wl,' を使ってエスケープ化される必要があることを注意しよう。この際は 'MAIN_LDFLAGS="-Wl,brtl"' と 'SHLIB_LDFLAGS="-Wl,-G"' を使おう。

Appendix C 新しいプラットフォーム

R を新しいハードウェアや OS にインストールする際に起こる幾つかの問題がある。それらは

浮動小数点算術: R は浮動小数点算術に対する POSIX, SVID そして IEEE モデルをサポートする。POSIX そして SVID モデルは何らの問題も引き起こさない。しかしながら IEEE モデルはトラブルを起こす。問題はシグナルの挙動を如何に設定するかに付いて何らの合意も無いことによる。Sun/Sparc, SGI/IRIX そして ix86 Linux は特別な行動を要求しないが、FreeBSD はマクロ `fpsetmask(0)` の呼び出しを要求し、そして OSF1 は '`-ieee_with_inexact`' フラグを用いて計算が行われることを要求する、等。新しいプラットフォームでは魔法の処方箋を発見し、それが動作するように幾つかのコードを加える必要がある。これはしばしば最上位ディレクトリにあるファイル '`config.site`' を通じて行われる。

高度の最適化を使うのは、少なくとも最初は注意しよう。多くのコンパイラではこれは IEEE モデルへの従順さを減ずる。例えば Solaris で '`-fast`' を使うと、SunPro コンパイラーは R の `NaN` を不正確に設定してしまう。

共有ライブラリ: 共有ライブラリを構築するために何をなすべきかに付いて各種プラットフォーム間での合意はほとんど無いように見える。コンパイラーとローダーに対するフラグには多くの異なった組合せがある。GNU libtool は現在のところ完全には FORTRAN をサポートしておらず (恐らく決して `f2c` をサポートしないであろう: このためにはシェルのラッパーが必要になるかも知れない)(まだ) 使用できない。我々が持ちいているテクニックは、最初に X ウィンドウシステムにそれが何をするか問い合わせ (`xmkmf` を用いて)、それからもっと良いことが分かっている状況 (GNU Compiler Collection からのツールや我々が知っているプラットフォームに対して) へとこれを書き換える。これは普通うまく動作するが、結果を手作業で書き換える必要があるかも知れない。`cc` や `ld` のマニュアルを調べると正しい呪文を明らかにする。一旦処方箋が分かれば、ファイル file '`config.site`' を (そこにある指示に従い) 構築がこれらのオプションを参照するように書き換えることが出来る。

もし R を新しいプラットフォームで稼働させせることに成功したならば、このコンフィギュレーション手順にそのプラットフォームを加えられるように知らせて欲しい。

もし使用しているプラットフォームで稼働する R を得るのに問題があれば、遠慮無く我々に質問して欲しい。我々は R を新しいプラットフォームに移植する作業に関する相当の経験を持っている。

関数と変数の索引

概念の索引