# **R** Installation and Administration

Version 4.5.0 Under development (2025-02-03)

R Core Team

This manual is for R, version 4.5.0 Under development (2025-02-03).

Copyright © 2001–2025 R Core Team

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the R Core Team.

# Table of Contents

1	Obtaining R	1
	1.1 Getting and unpacking the sources	1
	1.2 Getting patched and development versions	
	1.2.1 Using Subversion and rsync	
<b>2</b>	Installing R under Unix-alikes	3
	2.1 Simple compilation	
	2.2 Help options	
	2.3 Making the manuals	
	2.4 Installation	
	2.4       Installation         2.5       Uninstallation	
	2.6 Sub-architectures	
	2.7 Other Options	
	2.7.1 Debugging Symbols	
	2.7.2 OpenMP Support	
	2.7.3  C++ Support	
	2.7.4 C standards	
	2.7.5 Link-Time Optimization	
	2.7.5.1 LTO with GCC	
	2.7.5.2 LTO with LLVM	
	2.7.5.3 LTO for package checking	
	2.8 Testing an Installation	
3	Installing R under Windows	. 16
Ŭ	3.1 Building from source	
	3.1.1 The Windows toolset	10 16
	3.1.1 The windows toolset	
	3.1.2 EAEA	
	3.3 Testing an Installation	
	5.5 Testing an instantion	11
4	Installing R under macOS	10
4	_	
	4.1 Running R under macOS	
	4.2 Uninstalling under macOS	
	4.3 Multiple versions	21
<b>5</b>	Running R	າງ
J	running runnin	. 22
6	Add-on packages	ევ
U		
	6.1 Default packages	
	6.2 Managing libraries	
	6.3 Installing packages	
	6.3.1 Windows	
	$6.3.2 \mod OS$	
	6.3.3 Customizing package compilation	
	6.3.4 Multiple sub-architectures	
	6.3.5 Byte-compilation	29

	6.3.6 External software	29
	6.4 Updating packages	29
	6.5 Removing packages	30
	6.6 Setting up a package repository	30
	6.7 Checking installed source packages	31
7	Internationalization and Localization	32
•	7.1 Locales	
	7.1.1 Locales under Unix-alikes	
	7.1.2 Locales under Windows	
	7.1.2 Locales under macOS	
	7.2 Localization of messages	
	1.2 Localization of messages	55
0	Charain a batan an 20 and C4 bit bailde	<b>٩</b> ٣
8	Choosing between 32- and 64-bit builds	35
9	The standalone Rmath library	36
	9.1 Unix-alikes	
	9.2 Windows	
	0.2 Wildow5	01
٨	prondix A Essential and useful other programs under a	
	ppendix A Essential and useful other programs under a	
	Unix-alike	
	A.1 Essential programs and libraries	39
	A.2 Useful libraries and programs	
	A.2.1 Tcl/Tk	43
	A.2.2 Java support	43
	A.2.3 Other compiled languages	
	A.3 Linear algebra	
	A.3.1 BLAS	-
	A.3.1.1 ATLAS	
	A.3.1.2 OpenBLAS and BLIS	
	A.3.1.3 Intel MKL	
	A.3.1.4 Shared BLAS	10
	A.3.2 LAPACK	
	A.3.3 Caveats	50
Α	ppendix B Configuration on a Unix-alike	52
	B.1 Configuration options	52
	B.2 Internationalization support	53
	B.3 Configuration variables	53
	B.3.1 Setting paper size	53
	B.3.2 Setting the browsers	53
	B.3.3 Compilation flags	54
	B.3.3 Compilation flags B.3.4 Making manuals	
		54
	B.3.4 Making manuals	54 54
	B.3.4 Making manuals B.4 Setting the shell	$54 \\ 54 \\ 54$
	B.3.4 Making manuals         B.4 Setting the shell         B.5 Using make	54 54 54 54

Appendix	x C Platform notes	57
C.1 X11 i	ssues	57
C.2 Linux		58
C.2.1	Clang	60
	lang	
	Intel compilers	
C.3 macC	ps	61
C.3.1	Prerequisites	62
C.3.2	Fortran compiler	64
C.3.3	Cairo graphics	65
C.3.4	Other C/C++ compilers	65
C.3.5	Other libraries	66
C.3.	5.1 Accelerate	67
C.3.	5.2 OpenBLAS	67
C.3.6	Tcl/Tk headers and libraries	67
C.3.7	Java	68
C.3.8	Frameworks	69
C.3.9	Building R.app	69
C.3.10	Building binary packages	70
	Building for Intel on 'arm64'	
C.3.12	Installer	71
C.4 FreeE	SSD	71
*	BSD	
	in	
C.7 New	platforms	71
Function	and variable index	73
Concept	index	74
Environn	nent variable index	75

# 1 Obtaining R

Sources, binaries and documentation for R can be obtained via CRAN, the "Comprehensive R Archive Network" whose current members are listed at https://CRAN.R-project.org/mirrors.html.

# 1.1 Getting and unpacking the sources

The simplest way is to download the most recent R-x.y.z.tar.gz file, and unpack it with

```
tar -xf R-x.y.z.tar.gz
```

on systems that have a suitable<sup>1</sup> tar installed. On other systems you need to have the gzip program installed, when you can use

gzip -dc R-x.y.z.tar.gz | tar -xf -

The pathname of the directory into which the sources are unpacked should not contain spaces, as most make programs (and specifically GNU make) do not expect spaces.

If you want the build to be usable by a group of users, set umask before unpacking so that the files will be readable by the target group (e.g., umask 022 to be usable by all users). Keep this setting of umask whilst building and installing.

If you use a fairly recent GNU version of **tar** and do this as a root account (which on Windows includes accounts with administrator privileges) you may see many warnings about changing ownership. In which case you can use

tar -- no-same-owner -xf R-x.y.z.tar.gz

and perhaps also include the option --no-same-permissions. (These options can also be set in the TAR\_OPTIONS environment variable: if more than one option is included they should be separated by spaces.)

## 1.2 Getting patched and development versions

A patched version of the current release, 'r-patched', and the current development version, 'r-devel', are available as daily tarballs and via access to the R Subversion repository. (For the two weeks prior to the release of a minor (4.x.0) version, 'r-patched' tarballs may refer to beta/release candidates of the upcoming release, the patched version of the current release being available via Subversion.)

The tarballs are available from https://cran.r-project.org/src/base-prerelease/. Download R-patched.tar.gz or R-devel.tar.gz (or the .tar.bz2 versions) and unpack as described in the previous section. They are built in exactly the same way and on the same platform as distributions of R releases. Notice, that you probably want to use the CRAN master site for this, due to propagation delays. An alternative source of daily snapshots is maintained at https://stat.ethz.ch/R/daily/.

### 1.2.1 Using Subversion and rsync

Sources are also available via https://svn.R-project.org/R/, the R Subversion repository. If you have a Subversion client (see https://subversion.apache.org/), you can check out and update the current 'r-devel' from https://svn.r-project.org/R/trunk/ and the current 'r-patched' from 'https://svn.r-project.org/R/branches/R-x-y-branch/' (where x and y are the major and minor number of the current released version of R). E.g., use

svn checkout https://svn.r-project.org/R/trunk/ path

<sup>&</sup>lt;sup>1</sup> e.g. GNU tar version 1.15 or later, or that from the 'libarchive' (as used on macOS) or 'Heirloom Toolchest' distributions.

to check out 'r-devel' into directory path (which will be created if necessary). The alpha, beta and RC versions of an upcoming x.y.0 release are available from 'https://svn.r-project.org/R/branches/R-x-y-branch/' in the four-week period prior to the release.

Note that 'https:' is required<sup>2</sup>, and that the SSL certificate for the Subversion server of the R project should be recognized as from a trusted source.

Note that retrieving the sources by e.g. wget -r or svn export from that URL will not work (and will give a error early in the make process): the Subversion information is needed to build R.

The Subversion repository does not contain the current sources for the recommended packages, which can be obtained by rsync or downloaded from CRAN. To use rsync to install the appropriate sources for the recommended packages, run ./tools/rsync-recommended from the top-level directory of the R sources.

If downloading manually from CRAN, do ensure that you have the correct versions of the recommended packages: if the number in the file VERSION is 'x.y.z' you need to download the contents of 'https://CRAN.R-project.org/src/contrib/dir', where dir is 'x.y.z/Recommended' for r-devel or x.y-patched/Recommended for r-patched, respectively, to directory src/library/Recommended in the sources you have unpacked. After downloading manually you need to execute tools/link-recommended from the top level of the sources to make the requisite links in src/library/Recommended. A suitable incantation from the top level of the R sources using wget might be (for the correct value of dir)

wget -r -l1 --no-parent -A\\*.gz -nd -P src/library/Recommended \
https://CRAN.R-project.org/src/contrib/dir
./tools/link-recommended

 $<sup>^2</sup>$  for some Subversion clients 'http:' may appear to work, but requires continual redirection.

# 2 Installing R under Unix-alikes

R will configure and build under most common Unix and Unix-alike platforms including 'cpu-\*-linux-gnu' for the 'alpha', 'arm64' (also known as 'aarch64', 'ix86', 'mips', 'mipsel'#, 'ppc64', 'riscv64', 's390x', 'sparc64', and 'x86\_64' CPUs, 'aarch64-apple-darwin' and 'x86\_64-apple-darwin' as well as perhaps (it is tested less frequently on these platforms) 'x86\_64-\*-freebsd', 'x86\_64-\*-openbsd' and 'powerpc-ibm-aix6\*'

Only 64-bit platforms are tested regularly, and **configure** will warn if it encounters a 32-bit one.

In addition, binary distributions are available for some common Linux distributions (see the FAQ for current details) and for macOS. These are installed in platform-specific ways, so for the rest of this chapter we consider only building from the sources.

Cross-building is not possible: installing R builds a minimal version of R and then runs many R scripts to complete the build.

## 2.1 Simple compilation

First review the essential and useful tools and libraries in Appendix A [Essential and useful other programs under a Unix-alike], page 39, and install those you want or need. Ensure that either the environment variable TMPDIR is either unset (and /tmp exists and can be written in and scripts can be executed from) or points to the absolute path to a valid temporary directory (one from which execution of scripts is allowed) which does not contain spaces.

Choose a directory to install the R tree (R is not just a binary, but has additional data sets, help files, font metrics etc). Let us call this place  $R_-HOME$ . Untar the source code. This should create directories src, doc, and several more under a top-level directory: change to that top-level directory (At this point North American readers should consult Section B.3.1 [Setting paper size], page 53.) Issue the following commands:

```
./configure
make
```

(See Section B.5 [Using make], page 54, if your make is not called 'make'.) Users of Debian-based 64-bit systems<sup>2</sup> may need

```
./configure LIBnn=lib make
```

Then check the built system works correctly by

make check

Failures are not necessarily problems as they might be caused by missing functionality, but you should look carefully at any reported discrepancies. (Some non-fatal errors are expected in locales that do not support Latin-1, in particular in true C locales and non-UTF-8 non-Western-European locales.) A failure in tests/ok-errors.R may indicate inadequate resource limits (see Chapter 5 [Running R], page 22).

More comprehensive testing can be done by

make check-devel

or

```
make check-all
```

see Section 2.8 [Testing a Unix-alike Installation], page 14, for the possibilities of doing this in parallel. Note that these checks are only run completely if the recommended packages are

 $<sup>^1\,</sup>$ aka 'Apple Silicon', known to some as 'arm64-apple-darwin'.

 $<sup>^2\,</sup>$  which use lib rather than lib64 for their primary 64-bit library directories: attempts are made to detect such systems.

installed. If you have a local CRAN mirror, these checks can be speeded up by either setting environment variable R\_CRAN\_WEB to its URL, or having a file .R/repositories specifying it (see ?setRepositories).

make check-devel checks the included packages' tests directories. For grDevices more complete checks will be run if the environment variable R\_GRDEVICES\_COMPARE\_PS\_PDF is set: those checks are skipped on platforms using musl such as Alpine Linux, and it knows about differences from transliterations in macOS 14 and later.

Parallel make is supported for building R but not for the 'check' targets (as the output is likely to be unreadably interleaved, although where supported<sup>3</sup> GNU make's -0 may help).

If the configure and make commands execute successfully, a shell-script front-end called R will be created and copied to  $R_HOME/bin$ . You can link or copy this script to a place where users can invoke it, for example to /usr/local/bin/R. You could also copy the man page R.1 to a place where your man reader finds it, such as /usr/local/man/man1. If you want to install the complete R tree to, e.g., /usr/local/lib/R, see Section 2.4 [Installation], page 6. Note: you do not *need* to install R: you can run it from where it was built.

You do not necessarily have to build R in the top-level source directory (say, *TOP\_SRCDIR*). To build in *BUILDDIR*, run

### cd BUILDDIR TOP\_SRCDIR/configure make

and so on, as described further below. This has the advantage of always keeping your source tree clean and is particularly recommended when you work with a version of R from Subversion. (You may need GNU make to allow this, and you will need no spaces in the path to the build directory. It is unlikely to work if the source directory has previously been used for a build.)

There are many settings which can be customized when building R and most are described in the file config.site in the top-level source directory. This can be edited, but for an installation using *BUILDDIR* it is better to put the changed settings in a newly-created file config.site in the build directory.

Now rehash if necessary, type R, and read the R manuals and the R FAQ (files FAQ or doc/manual/R-FAQ.html, or https://CRAN.R-project.org/doc/FAQ/R-FAQ.html which always has the version for the latest release of R).

Note: if you already have R installed, check that where you installed R replaces or comes earlier in your path than the previous installation. Some systems are set up to have /usr/bin (the standard place for a system installation) ahead of /usr/local/bin (the default place for installation of R) in their default path, and some do not have /usr/local/bin on the default path.

### 2.2 Help options

R by default provides help pages as plain text displayed in a pager, with the options (see the help for help) of displaying help as HTML or PDF.

By default HTML help pages are created when needed rather than being built at install time.

If you need to disable the server and want HTML help, there is the option to build HTML pages when packages are installed (including those installed with R). This is enabled by the configure option --enable-prebuilt-html. Whether R CMD INSTALL (and hence install.packages) prebuilds HTML pages is determined by looking at the R installation and is reported by R CMD INSTALL --help: it can be overridden by specifying one of the INSTALL options --html or --no-html.

 $<sup>^{3}</sup>$  not by the version supplied by macOS.

The server is disabled by setting the environment variable R\_DISABLE\_HTTPD to a nonempty value, either before R is started or within the R session before HTML help (including help.start) is used. It is also possible that system security measures will prevent the server from being started, for example if the loopback interface has been disabled. See ?tools::startDynamicHelp for more details.

### 2.3 Making the manuals

There is a set of manuals that can be built from the sources,

'fullrefman'

Printed versions of all the help pages for base and recommended packages (around 3900 pages).

'refman' Printed versions of the help pages for selected base packages (around 2300 pages)

'R-FAQ' R FAQ

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

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

'R-admin' "R Installation and Administration", this manual.

'R-exts' "Writing R Extensions".

'R-ints' "R Internals".

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

To make these (with 'fullrefman' rather than 'refman'), use

make pdf	to create PDF versions
make info	to create info files (not 'refman' nor 'fullrefman').

You will not be able to build any of these unless you have texi2any version 6.1 or later installed, and for PDF you must have texi2dvi and texinfo.tex installed (which are part of the GNU Texinfo distribution but are, especially texinfo.tex, often made part of the  $T_EX$  package texinfo in re-distributions). The path to texi2any can be set by macro 'TEXI2ANY' in config.site. NB: texi2any requires perl.

The PDF versions can be viewed using any recent PDF viewer: they have hyperlinks that can be followed. The info files are suitable for reading online with Emacs or the standalone GNU info program. The PDF versions will be created using the paper size selected at configuration (default ISO a4): this can be overridden by setting R\_PAPERSIZE on the make command line, or setting R\_PAPERSIZE in the environment and using make -e. (If re-making the manuals for a different paper size, you should first delete the file doc/manual/version.texi. The usual value for North America would be 'letter'.)

There are some issues with making the PDF reference manual, fullrefman.pdf or refman.pdf. The help files contain both non-ASCII characters (e.g. in text.Rd) and upright quotes, neither of which are contained in the standard LATEX Computer Modern fonts. We have provided the following alternatives:

times (The default.) Using standard PostScript fonts, Times Roman, Helvetica and Courier. This works well both for on-screen viewing and for printing. One disadvantage is that the Usage and Examples sections may come out rather wide: this can be overcome by using *in addition* either of the options inconsolata (on a Unix-alike only if found by configure) or beramono, which replace the Courier monospaced font by Inconsolata or Bera Sans Mono respectively. (You will need the  $\[AT_EX]$  package inconsolata<sup>4</sup> or bera installed.)

Note that in most IATEX installations this will not actually use the standard fonts for PDF, but rather embed the URW clones NimbusRom, NimbusSans and (for Courier, if used) NimbusMon.

This needs LATEX packages times, helvetic and (if used) courier installed.

 $\label{eq:lm} \begin{array}{ll} \mbox{Using the Latin Modern (aka lmodern) fonts, usually installed even in basic $T_EX$ distributions. This uses fonts rather similar to Computer Modern, but is not so good on-screen as times and produces larger PDF files. \\ \end{array}$ 

The default can be overridden by setting the environment variable R\_RD4PDF. (On Unixalikes, this will be picked up at install time and stored in etc/Renviron, but can still be overridden when the manuals are built, using make -e.) The usual<sup>5</sup> default value for R\_RD4PDF is 'times, inconsolata, hyper': omit 'inconsolata' if you do not have LATEX package inconsolata installed. 'hyper' is always enabled (with a fallback if LATEX package hyperref is not installed).

Further options, e.g for hyperref, can be included in a file Rd.cfg somewhere on your  $IAT_EX$  search path. For example, if you prefer to hyperlink the text and not the page number in the table of contents use

\ifthenelse{\boolean{Rd@use@hyper}}{\hypersetup{linktoc=section}}{}

or

#### \ifthenelse{\boolean{Rd@use@hyper}}{\hypersetup{linktoc=all}}{}

to hyperlink both text and page number.

Any generated PDF manuals can be compacted by

make compact-pdf

provided qpdf and gs are available (see ?tools::compactPDF for how to specify them if not on the path).

E-book versions of most of the manuals in one or both of .epub and .mobi formats can be made by running in doc/manual one of

make ebooks make epub make mobi

This requires ebook-convert from Calibre (https://calibre-ebook.com/download), or from most Linux distributions. If necessary the path to ebook-convert can be set as make macro EBOOK by editing doc/manual/Makefile (which contains a commented value suitable for macOS) or using make -e.

### 2.4 Installation

To ensure that the installed tree is usable by the right group of users, set umask appropriately (perhaps to '022') before unpacking the sources and throughout the build process.

After

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

<sup>&</sup>lt;sup>4</sup> Instructions on how to install the latest version are at https://www.ctan.org/tex-archive/fonts/ inconsolata/. TeX Live bundles inconsolata in collection-fontsextra.

<sup>&</sup>lt;sup>5</sup> on a Unix-alike, 'inconsolata' is omitted if not found by configure.

(or, when building outside the source, *TOP\_SRCDIR/configure*, etc) have been completed successfully, you can install the complete R tree to your system by typing

#### make install

A parallel make can be used (but run make before make install). Those using GNU make 4.0 or later may want to use make -j n - 0 to avoid interleaving of output.

This will install to the following directories:

```
prefix/bin or bindir
```

the front-end shell script and other scripts and executables

#### prefix/man/man1 or mandir/man1

the man page

#### prefix/LIBnn/R or libdir/R

all the rest (libraries, on-line help system, ...). Here *LIBnn* is usually 'lib', but may be 'lib64' on some 64-bit Linux systems. This is known as the R home directory.

where *prefix* is determined during configuration (typically /usr/local) and can be set by running configure with the option --prefix, as in

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

where the value should be an absolute path. This causes make install to install the R script to /where/you/want/R/to/go/bin, and so on. The prefix of the installation directories can be seen in the status message that is displayed at the end of configure. The installation may need to be done by the owner of prefix, often a root account.

There is the option of using make install-strip (see Section 2.7.1 [Debugging Symbols], page 10).

You can install into another directory tree by using

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

at least with GNU make (but not some other Unix makes).

More precise control is available at configure time via options: see configure --help for details. (However, most of the 'Fine tuning of the installation directories' options are not used by R.)

Configure options --bindir and --mandir are supported and govern where a copy of the R script and the man page are installed.

The configure option --libdir controls where the main R files are installed: the default is 'eprefix/LIBnn', where eprefix is the prefix used for installing architecture-dependent files, defaults to prefix, and can be set via the configure option --exec-prefix.

Each of bindir, mandir and libdir can also be specified on the make install command line (at least for GNU make).

The configure or make variables rdocdir and rsharedir can be used to install the systemindependent doc and share directories to somewhere other than libdir. The C header files can be installed to the value of rincludedir: note that as the headers are not installed into a subdirectory you probably want something like rincludedir=/usr/local/include/R-4.5.0.

If you want the R home to be something other than *libdir/*R, use **rhome**: for example

make install rhome=/usr/local/lib64/R-4.5.0

will use a version-specific R home on a non-Debian Linux 64-bit system.

If you have made R as a shared/static library you can install it in your system's library directory by

make prefix=/path/to/here install-libR

where **prefix** is optional, and **libdir** will give more precise control.<sup>6</sup> However, you should not install to a directory mentioned in LDPATHS (e.g. /usr/local/lib64) if you intend to work with multiple versions of R, since that directory may be given precedence over the lib directory of other R installations.

make install-strip

will install stripped executables, and on platforms where this is supported, stripped libraries in directories lib and modules and in the standard packages.

Note that installing R into a directory whose path contains spaces is not supported, and some aspects (such as installing source packages) will not work.

To install info and PDF versions of the manuals, use one or both of

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

Once again, it is optional to specify **prefix**, **libdir** or **rhome** (the PDF manuals are installed under the R home directory).

More precise control is possible. For info, the setting used is that of infodir (default *prefix/info*, set by configure option --infodir). The PDF files are installed into the R doc tree, set by the make variable rdocdir.

A staged installation is possible, that it is installing R into a temporary directory in order to move the installed tree to its final destination. In this case prefix (and so on) should reflect the final destination, and DESTDIR should be used: see https://www.gnu.org/prep/standards/html\_node/DESTDIR.html.

You can optionally install the run-time tests that are part of make check-all by

```
make install-tests
```

which populates a tests directory in the installation.

### 2.5 Uninstallation

You can uninstall R by

```
make uninstall
```

optionally specifying **prefix** etc in the same way as specified for installation.

This will also uninstall any installed manuals. There are specific targets to uninstall info and PDF manuals in file doc/manual/Makefile.

Target uninstall-tests will uninstall any installed tests, as well as removing the directory tests containing the test results.

An installed shared/static libR can be uninstalled by

make prefix=/path/to/here uninstall-libR

### 2.6 Sub-architectures

Now 32-bit builds are unsupported, this section is only of historical interest, although in future the mechanisms could be used for different CPU types on the same OS (e.g. 'x86\_64' and 'aarch64').

Some platforms can support closely related builds of R which can share all but the executables and dynamic objects. Examples include builds under Linux for different CPUs or 32- and 64-bit builds.

R supports the idea of architecture-specific builds, specified by adding 'r\_arch=name' to the configure line. Here name can be anything non-empty, and is used to name subdirectories of

<sup>&</sup>lt;sup>6</sup> This will be needed if more than one sub-architecture is to be installed.

lib, etc, include and the package libs subdirectories. Example names from other software are the use of sparcv9 on Sparc Solaris and 32 by gcc on 'x86\_64' Linux.

If you have two or more such builds you can install them over each other (and for 32/64-bit builds on one architecture, one build can be done without 'r\_arch'). The space savings can be considerable: on 'x86\_64' Linux a basic install (without debugging symbols) took 74Mb, and adding a 32-bit build added 6Mb. If you have installed multiple builds you can select which build to run by

R --arch=name

and just running 'R' will run the last build that was installed.

R CMD INSTALL will detect if more than one build is installed and try to install packages with the appropriate library objects for each. This will not be done if the package has an executable configure script or a src/Makefile file. In such cases you can install for extra builds by

R --arch=name CMD INSTALL --libs-only pkg1 pkg2 ...

If you want to mix sub-architectures compiled on different platforms (for example 'x86\_64' Linux and 'i686' Linux), it is wise to use explicit names for each, and you may also need to set libdir to ensure that they install into the same place.

When sub-architectures are used the version of Rscript in e.g. /usr/bin will be the last installed, but architecture-specific versions will be available in e.g. /usr/lib64/R/bin/exec\${ $R_$  ARCH}. Normally all installed architectures will run on the platform so the architecture of Rscript itself does not matter. The executable Rscript will run the R script, and at that time the setting of the R\_ARCH environment variable determines the architecture which is run.

When running post-install tests with sub-architectures, use

```
R --arch=name CMD make check[-devel|all]
```

to select a sub-architecture to check.

Sub-architectures were also used on Windows, but by selecting executables within the appropriate bin directory such as  $R_HOME/bin/x64$ . As from R 4.2.0 only the 'x64' subdirectory is used.

### 2.7 Other Options

There are many other installation options, most of which are listed by configure --help. Almost all of those not listed elsewhere in this manual are either standard autoconf options not relevant to R or intended for specialist uses by the R developers.

One that may be useful when working on R itself is the option --disable-byte-compiledpackages, which ensures that the base and recommended packages are not byte-compiled. (Alternatively the (make or environment) variable R\_NO\_BASE\_COMPILE can be set to a non-empty value for the duration of the build.)

Option --with-internal-tzcode makes use of R's own code and copy of the IANA database for managing timezones. This will be preferred where there are issues with the system implementation, usually involving times after 2037 or before 1916. An alternative time-zone directory<sup>7</sup> can be used, pointed to by environment variable TZDIR: this should contain files such as Europe/London. On all tested OSes the system timezone was deduced correctly, but if necessary it can be set as the value of environment variable TZ.

Options --with-internal-iswxxxx, --with-internal-towlower and --with-internalwcwidth control the replacement of the system wide-character classification (such as iswprint), case-changing (wctrans) and width (wcwidth and wcswidth) functions by ones contained in the R sources. Replacement of the classification functions has been done for many years on macOS

<sup>&</sup>lt;sup>7</sup> How to prepare such a directory is described in file src/extra/tzone/Notes in the R sources.

and AIX (and Windows): option --with-internal-iswxxxx allows this to be suppressed on those platforms or used on others. Replacing the case-changing functions is the default on macOS and Windows. Replacement of the width functions has also been done for many years and remains the default. These options will only matter to those working with non-ASCII character data, especially in languages written in a non-Western script<sup>8</sup> (which includes 'symbols' such as emoji). Note that one of those iswxxxxx is iswprint which is used to decide whether to output a character as a glyph or as a '\U{xxxxx}}' escape—for example, try '"\U1f600"', an emoji. The width functions are of most importance in East Asian locale: their values differ between such locales. (Replacing the system functions provides a degree of platform-independence (including to OS updates) but replaces it with a dependence on the R version.)

### 2.7.1 Debugging Symbols

By default, configure adds a flag (usually -g) to the compilation flags for C, Fortran and C++ sources. This will slow down compilation and increase object sizes of both R and packages, so it may be a good idea to change those flags (set 'CFLAGS' etc in config.site before configuring, or edit files Makeconf and etc/Makeconf between running configure and make).

Having debugging symbols available is useful both when running R under a debugger (e.g., R -d gdb) and when using sanitizers and valgrind, all things intended for experts.

Debugging symbols (and some others) can be 'stripped' on installation by using

```
make install-strip
```

How well this is supported depends on the platform: it works best on those using GNU binutils. On 'x86\_64' Linux a typical reduction in overall size was from 92MB to 66MB. On macOS debugging symbols are not by default included in .dylib and .so files, so there is negligible difference.

### 2.7.2 OpenMP Support

By default **configure** searches for suitable flags<sup>9</sup> for OpenMP support for the C, C++ (default standard) and Fortran compilers.

Only the C result is currently used for R itself, and only if MAIN\_LD/DYLIB\_LD were not specified. This can be overridden by specifying

R\_OPENMP\_CFLAGS

Use for packages has similar restrictions (involving SHLIB\_LD and similar: note that as Fortran code is by default linked by the C (or C++) compiler, both need to support OpenMP) and can be overridden by specifying some of

SHLIB\_OPENMP\_CFLAGS SHLIB\_OPENMP\_CXXFLAGS SHLIB\_OPENMP\_FFLAGS

Setting these to an empty value will disable OpenMP for that compiler (and configuring with --disable-openmp will disable all detection<sup>10</sup> of OpenMP). The configure detection test is to compile and link a standalone OpenMP program, which is not the same as compiling a shared object and loading it into the C program of R's executable. Note that overridden values are not tested.

 $<sup>^{8}</sup>$  But on Windows problems have been seen with case-changing functions on accented Latin-1 characters.

<sup>&</sup>lt;sup>9</sup> for example, -fopenmp, -fiopenmp, -xopenmp or -qopenmp. This includes for clang and the Intel and Oracle compilers.

<sup>&</sup>lt;sup>10</sup> This does not necessarily disable *use* of OpenMP – the **configure** code allows for platforms where OpenMP is used without a flag. For the **flang** compiler in late 2017, the Fortran runtime always used OpenMP.

### 2.7.3 C++ Support

C++ is not used by R itself, but support is provided for installing packages with C++ code via make macros defined in file etc/Makeconf (and with explanations in file config.site):

CXX CXXFLAGS CXXPICFLAGS CXXSTD CXX11 CXX11STD CXX11FLAGS CXX11PICFLAGS CXX14 CXX14STD CXX14FLAGS CXX14PICFLAGS CXX17 CXX17STD CXX17FLAGS CXX17PICFLAGS CXX20 CXX20STD CXX20FLAGS CXX20PICFLAGS CXX23 CXX23STD CXX23FLAGS CXX23PICFLAGS

The macros CXX etc are those used by default for C++ code. configure will attempt to set the rest suitably, choosing for CXXSTD and CXX11STD a suitable flag such as -std=gnu++17 for C++17 support (which is required if C++ is to be supported by default). Inferred values can be overridden in file config.site or on the configure command line: user-supplied values will be tested by compiling some C++11/14/17/20/23 code.

It may be that there is no suitable flag for C++14/17/20/23 support with the default compiler, in which case a different compiler could be selected for CXX14/CXX17/CXX20/CXX23 with its corresponding flags.

If no suitable compiler/flag is found for the default C++ compiler, one can be set in file config.site *via* macros CXX and CXXSTD. A user-specified compiler does not need to pass the C++17 tests, so do this at your own risk as some packages may not compile.

The -std flag is supported by the GCC, clang++ and Intel compilers. Currently accepted values are (plus some synonyms)

g++:	c++11 gnu+11 c++14 gnu++14 c++17 gnu++17 c++2a gnu++2a (from 8)
	c++20 gnu++20 (from 10) c++23 gnu++23 c++2b gnu++2b (from 11)
Intel:	c++11 gnu+11 c++14 gnu++14 c++17 gnu++17
	c++20 gnu++20 (from 2021.1) c++2b gnu++2b (from 2022.2)
	c++23 gnu++23 (at least from 2024.0)

(Those for LLVM clang++ are documented at https://clang.llvm.org/cxx\_status.html, and follow g++: -std=c++20 is supported from Clang 10, -std=c++2b from Clang 13 and -std=c++23 from Clang 17. Apple Clang supports -std=c++2b from 13.1.6 and -std=c++23 from 16.0.0.)

'Standards' for g++ starting with 'gnu' enable 'GNU extensions': what those are is hard to track down.

For the use of C++ in R packages, see Writing R Extensions. Prior to R 3.6.0 the default C++ standard was that of the compiler used: currently it is C++17.

https://en.cppreference.com/w/cpp/compiler\_support indicates which versions of common compilers support (parts of) which C++ standards. GCC introduced C++17 support gradually, but version 7 should suffice.

### 2.7.4 C standards

Compiling R requires some POSIX features (such as strdup<sup>11</sup> and the ssize\_t type) not in the C standards. Typically compilers make these available, but **not** if strict C compliance is specified by for example -std=c17. So if you want to specify a non-default standard use something like -std=gnu23.

Compiling R requires C99 or later: C11 and C17 are minor updates, but the substantial update 'C23' (finally published in October 2024) is also supported for current versions of GCC and clang. The upcoming GCC 15 is expected to default to C23.

As from R 4.3.0 there is support for packages to indicate their preferred C version. Macros CC17, C17FLAGS, CC23 and C23FLAGS can be set in config.site (there are examples there). Those for C17 should support C17 or earlier and not allow C23 additions so for example bool, true and false can be used as identifiers. Those for C23 should support the new types such as bool.

Some compilers warn enthusiastically about prototypes. For most, omitting -Wstrictprototypes in C17FLAGS suffices. However, versions 15 and later of LLVM clang and 14.0.3 and later of Apple clang warn by default in all modes if -Wall or -pedantic is used, and may need -Wno-strict-prototypes.

### 2.7.5 Link-Time Optimization

There is support for using link-time optimization (LTO) if the toolchain supports it: configure with flag --enable-lto. When LTO is enabled it is also used for compiled code in add-on packages unless the flag --enable-lto=R is used<sup>12</sup>.

The main benefit seen to date from LTO has been detecting long-standing bugs in the ways packages pass arguments to compiled code and between compilation units. Benchmarking in 2020 with gcc/gfortran 10 showed gains of a few percent in increased performance and reduction in installed size for builds without debug symbols, but large size reductions for some packages<sup>13</sup> with debug symbols. (Performance and size gains are said to be most often seen in complex C++ builds.)

Whether toolchains support LTO is often unclear: all of the C compiler, the Fortran compiler<sup>14</sup> and linker have to support it, and support it by the same mechanism (so mixing compiler families may not work and a non-default linker may be needed). It has been supported by the GCC and LLVM projects for some years with diverging implementations.

 $<sup>^{11}\,</sup>$  this is part of C23, but part of the C library not the compiler.

 $<sup>^{12}</sup>$  Then all add-on packages, including recommended packages are not installed with LTO.

 $<sup>^{13}\,</sup>$  A complete CRAN installation reduced from 50 to 35GB.

<sup>&</sup>lt;sup>14</sup> although there is the possibility to exclude Fortran but that misses some of the benefits.

LTO support was added in 2011 for GCC 4.5 on Linux but was little used before 2019: compiler support has steadily improved over those years and --enable-lto=R is nowadays used for some routine CRAN checking.

Unfortunately --enable-lto may be accepted but silently do nothing useful if some of the toolchain does not support LTO: this is less common than it once was.

Various macros can be set in file config.site to customize how LTO is used. If the Fortran compiler is not of the same family as the C/C++ compilers, set macro 'LTO\_FC' (probably to empty). Macro 'LTO\_LD' can be used to select an alternative linker should that be needed.

### 2.7.5.1 LTO with GCC

This has been tested on Linux with gcc/gfortran 8 and later: that needed setting (e.g. in config.site)

```
AR=gcc-ar
RANLIB=gcc-ranlib
```

For non-system compilers or if those wrappers have not been installed one may need something like

```
AR="ar --plugin=/path/to/liblto_plugin.so"
RANLIB="ranlib --plugin=/path/to/liblto_plugin.so"
```

and NM may be needed to be set analogously. (If using an LTO-enabled build to check packages, set environment variable  $\texttt{UserNM}^{15}$  to 'gcc-nm'.)

With GCC 5 and later it is possible to parallelize parts of the LTO linking process: set the make macro 'LTO' to something like 'LTO=-flto=8' (to use 8 threads), for example in file config.site.

Under some circumstances and for a few packages, the PIC flags have needed overriding on Linux with GCC 9: e.g use in config.site:

```
CPICFLAGS=-fPIC
CXXPICFLAGS=-fPIC
CXX11PICFLAGS=-fPIC
CXX14PICFLAGS=-fPIC
CXX17PICFLAGS=-fPIC
CXX20PICFLAGS=-fPIC
FPICFLAGS=-fPIC
```

We suggest only using these if the problem is encountered (it had not been seen on CRAN with GCC 10–15 at the time of writing).

Note that R will usually need to be re-compiled after even a minor update to the compiler (e.g. from 13.1 to 13.2).

## 2.7.5.2 LTO with LLVM

LLVM supports another type of LTO called 'Thin LTO' as well as a similar implementation to GCC, sometimes called 'Full LTO'. (See https://clang.llvm.org/docs/ThinLTO.html.) Currently the LLVM compilers relevant to R are clang and flang-new for which this can be selected by setting macro 'LTO=-flto=thin'. LLVM has

```
AR=llvm-ar
RANLIB=llvm-ranlib
```

(but macOS does not, and these are not needed there). Where the linker supports a parallel backend for Thin LTO this can be specified *via* the macro 'LTO\_LD': see the URL above for per-linker settings and further linking optimizations.)

 $<sup>^{15}</sup>$  not NM as we found make overriding that.

For example, on macOS one might use

```
LTO=-flto=thin
LTO_FC=
LTO_LD=-Wl,-mllvm,-threads=4
```

to use Thin LTO with 4 threads for C/C++ code, but skip LTO for Fortran code compiled with <code>gfortran</code>.

It is said to be particularly beneficial to use -03 for clang in conjunction with LTO.

It seems that flang-new may in future support LTO.

The 2020s versions of Intel's C/C++ compilers are based on LLVM and as such support LLVM-style LTO, both 'full' and 'thin'. This might use something like

LTO=-flto=thin -flto-jobs=8

### 2.7.5.3 LTO for package checking

LTO effectively compiles all the source code in a package as a single compilation unit and so allows the compiler (with sufficient diagnostic flags such as -Wall) to check consistency between what are normally separate compilation units.

With gcc/gfortran 9.x and later<sup>16</sup> LTO will flag inconsistencies in calls to Fortran subroutines/functions, both between Fortran source files and between Fortran and C/C++. gfortran 8.4, 9.2 and later can help understanding these by extracting C prototypes from Fortran source files with option -fc-prototypes-external, e.g. that (at the time of writing) Fortran LOGICAL corresponds to int\_least32\_t \* in C.

# 2.8 Testing an Installation

Full post-installation testing is possible only if the test files have been installed with

```
make install-tests
```

which populates a tests directory in the installation.

If this has been done, two testing routes are available. The first is to move to the home directory of the R installation (as given by R RHOME or from R as R.home()) and run

```
cd tests
## followed by one of
../bin/R CMD make check
../bin/R CMD make check-devel
../bin/R CMD make check-all
```

and other useful targets are test-BasePackages and test-Recommended to run tests of the standard and recommended packages (if installed) respectively.

This re-runs all the tests relevant to the installed R (including for example the code in the package vignettes), but not for example the ones checking the example code in the manuals nor making the standalone Rmath library. This can occasionally be useful when the operating environment has been changed, for example by OS updates or by substituting the BLAS (see Section A.3.1.4 [Shared BLAS], page 48).

Parallel checking of packages may be possible: set the environment variable TEST\_MC\_CORES to the maximum number of processes to be run in parallel. This affects both checking the package examples (part of make check) and package sources (part of make check-devel and make check-recommended). It does require a make command which supports the make -j n option: most do.

 $<sup>^{16}\,</sup>$  probably also 8.4 and later.

Alternatively, the installed R can be run, preferably with --vanilla. Then

```
pdf("tests.pdf") ## optional, but prevents flashing graphics windows
Sys.setenv(LC_COLLATE = "C", LC_TIME = "C", LANGUAGE = "en")
tools::testInstalledBasic("both")
tools::testInstalledPackages(scope = "base")
tools::testInstalledPackages(scope = "recommended")
```

runs the basic tests and then all the tests on the standard and recommended packages. These tests can be run from anywhere: the basic tests write their results in the tests folder of the R home directory and run fewer tests than the first approach: in particular they do not test things which need Internet access—that can be tested by

tools::testInstalledBasic("internet")

It is possible to test the installed packages (but not their package-specific tests) by testInstalledPackages even if make install-tests was not run. The outputs are written under the current directory unless a different one is specified by outDir.

Note that the results may depend on the language set for times and messages: for maximal similarity to reference results you may want to try setting (before starting the R session)

#### LANGUAGE=en

and use a UTF-8 or Latin-1 locale.

# 3 Installing R under Windows

[The rest of this paragraph is only relevant after release.] The bin/windows directory of a CRAN site contains binaries for a base distribution and a large number of add-on packages from CRAN to run on 64-bit 'x86\_64' Windows.

R is most tested on current versions of Windows 10 and Windows Server 2022 with UTF-8 as the charset encoding. It works also on Windows 11. It runs on older versions of Windows, but normally with other charset encoding and may require manual installation of the Universal C Runtime (UCRT).

Your file system must allow long file names (as is likely except perhaps for some networkmounted systems). If it does not also support conversion to short name equivalents (a.k.a. DOS 8.3 names), then R *must* be installed in a path that does not contain spaces.

Installation is *via* the installer **R-devel-win.exe**. Just double-click on the icon and follow the instructions. You can uninstall R from the Control Panel.

You will be asked to choose a language for installation: that choice applies to both installation and un-installation but not to running R itself.

See the R Windows FAQ (https://CRAN.R-project.org/bin/windows/base/rw-FAQ.html) for more details on the binary installer and for information on use on older Windows systems.

### 3.1 Building from source

It is possible to use other 64-bit toolchains (including 'MSYS2') with UCRT support to build R, but this manual only documents that used for recent binary distributions of R. When using other toolchains, makefiles of R and packages may need to be adapted.

#### 3.1.1 The Windows toolset

The binary distribution of R is currently built with tools from Rtools44 for Windows (https://CRAN.R-project.org/bin/windows/Rtools/rtools44/rtools.html). See Building R-devel and packages on Windows (https://CRAN.R-project.org/bin/windows/base/ howto-R-devel.html) for more details on how to use it.

The toolset includes compilers (currently GCC version 13.2.0 with selected additional patches) and runtime libraries from the 'MinGW-w64' project (http://mingw-w64.org/) and a number of pre-compiled static libraries and headers used by R and R packages, compiled by 'MXE' (https://mxe.cc/) (M cross environment, with updates maintained by Tomas Kalibera). The toolset also includes build tools from the the 'MSYS2' project (https://www.msys2.org/). Additional build tools packaged by 'MSYS2' may be installed via a package manager ('pacman').

There is also an experimental variant of Rtools44 with support for 64-bit ARM CPUs (aarch64) via LLVM 17 toolchain using clang/flang-new compilers, 11d linker, and libc++.

The toolsets used for 64-bit Windows from 2008 to 2022 were based on MinGW-w64. The assistance of Yu Gong at a crucial step in porting R to MinGW-w64 is gratefully acknowledged, as well as help from Kai Tietz, the lead developer of the MinGW-w64 project and from Martin Storsjo.

### 3.1.2 PT<sub>E</sub>X

Both building R and checking packages need a distribution of  $IAT_EX$  installed, with the directory containing pdflatex on the path.

The 'MiKTeX' (https://miktex.org/) distribution of LATEX is that used on CRAN. This can be set up to install extra packages 'on the fly' (without asking), which is the simplest way to

use it. The 'basic' version of 'MiKTeX' will need to add some packages.<sup>1</sup> In any case ensure that the **inconsolata** package is installed—you can check with the 'MiKTeX' Package Manager.

It is also possible to use the TeX Live distribution from https://www.tug.org/texlive/ . (The CRAN package tinytex (https://CRAN.R-project.org/package=tinytex) can install and manage a subset of TeX Live.)

# 3.2 Checking the build

You can test a build by running

make check

The recommended packages can be checked by

make check-recommended

Other levels of checking are

make check-devel

for a more thorough check of the R functionality, and

```
make check-all
```

for both check-devel and check-recommended.

If a test fails, there will almost always be a .Rout.fail file in the directory being checked (often tests/Examples or tests): examine the file to help pinpoint the problem.

Parallel checking of package sources (part of make check-devel and make check-recommended) is possible: see the environment variable TEST\_MC\_CORES to the maximum number of processes to be run in parallel.

### 3.3 Testing an Installation

The Windows installer contains a set of test files used when building R.

The toolset is not needed to run these tests, but more comprehensive analysis of errors will be given if diff is in the path.

Launch either Rgui or Rterm (preferred), preferably with --vanilla. Then run

```
Sys.setenv(LC_COLLATE = "C", LC_TIME="C", LANGUAGE = "en")
tools::testInstalledBasic("both")
tools::testInstalledPackages(scope = "base")
tools::testInstalledPackages(scope = "recommended")
```

runs the basic tests and then all the tests on the standard and recommended packages. These tests can be run from anywhere: testInstalledBasic writes results in the tests folder of the R home directory (as given by R.home()) and testInstalledPackages under the current directory unless a different one is specified by outDir.

For the tests folder to be writeable, one normally needs to install R to a directory other than the default C:\Program Files. The installer also allows to install R without Administrator privileges, see the R Windows FAQ (https://CRAN.R-project.org/bin/windows/base/rw-FAQ.html) for more details.

The results of example(md5sums) when testing tools may differ from the reference output as some files are installed with Windows' CRLF line endings. Also, expect differences in regplot-latin1.pdf.

<sup>&</sup>lt;sup>1</sup> There are reports of segfaults when 'MiKTeX' installs additional packages when making NEWS.pdf: re-running make seems to solve this.

One can also run tests from the toolset shell (e.g. bash) similarly to a Unix-like installation. Move to the home directory of the R installation (as given by R RHOME or from R as R.home()) and run

cd tests ## followed by one of ../bin/R CMD make check ../bin/R CMD make check-devel ../bin/R CMD make check-all

Remember that  ${\rm \ensuremath{\mathbb I}\xspace{-1.5}} T_{E\!X}$  needs to be on the path.

# 4 Installing R under macOS

[The rest of this paragraph is only relevant after release.] The front page of a CRAN site has a link 'Download R for (Mac) OS X' which takes you to a new page. Two files are offered for download, R-4.5.0-arm64.pkg and  $R-4.5.0-x86_64.pkg$ . Both are for macOS 11 or later (Big Sur, Monterey, Ventura, Sonoma, ...).

The first is for 'Apple Silicon' (*aka* 'M1', 'M2', ...) Macs, the second for older Macs with an 'x86\_64' (Intel) CPU.

It is important that if you use a binary installer package that your OS is fully updated: look at 'Software Update' in 'System Preferences' to be sure.

To install, just double-click on the icon of the file you downloaded. At the 'Installation Type' stage, note the option to 'Customize'. This currently shows four components: everyone will need the 'R Framework' component: the remaining components are optional. (The 'Tcl/Tk' component is needed to use package **tcltk**. The 'Texinfo' component is only needed by those installing source packages or R from its sources.)

Note for Ventura users: installation from the Downloads folder may not be allowed or may require additional authorization, so we suggest you download somewhere else such as your desktop or home folder.

These are Apple Installer packages. If you encounter any problem during the installation, please check the Installer log by clicking on the "Window" menu and item "Installer Log". The full output (select "Show All Log") is useful for tracking down problems. Note that the installer is clever enough to try to upgrade the last-installed version of the application where you installed it (which may not be where you want this time . . .).

Various parts of the build require XQuartz to be installed: see https://www.xquartz.org/ releases/.<sup>1</sup> These include the tcltk package and the X11 graphics device: attempting to use these without XQuartz will remind you. This is also needed for some builds of the cairographicsbased devices (which are not often used on macOS) such as png(type = "cairo") and svg() and some third-party packages (e.g. rgl (https://CRAN.R-project.org/package=rgl)).

If you update your macOS version, you should re-install R (and perhaps XQuartz): the installer may tailor the installation to the current version of the OS.

Installers for R-patched and R-devel are usually available from https://mac.R-project. org. (Some of these packages may be unsigned/not notarized: to install those Control/right/twofinger click, select 'Open With' and 'Installer'.)

For building R from source, see Section C.3 [macOS], page 61.

# 4.1 Running R under macOS

There are two ways to run R on macOS from a CRAN binary distribution.

There is a GUI console normally installed with the R icon in /Applications which you can run by double-clicking (e.g. from Launchpad or Finder). (If you cannot find it there it was possibly installed elsewhere so try searching for it in Spotlight.) This is usually referred to as R.APP to distinguish it from command-line R: its user manual is currently part of the macOS FAQ at https://cran.r-project.org/bin/macosx/RMacOSX-FAQ.html and can be viewed from R.APP's 'Help' menu.

You can run command-line R and Rscript from a Terminal<sup>2</sup> so these can be typed as commands as on any other Unix-alike: see the next chapter of this manual. There are some

<sup>&</sup>lt;sup>1</sup> At the time of writing, version 2.8.5 or later.

<sup>&</sup>lt;sup>2</sup> The installer puts links to R and Rscript in /usr/local/bin. If these are missing or that is not on your path, you can run directly the copies in /Library/Frameworks/R.framework/Resources/bin or link those yourself to somewhere on your path.

small differences which may surprise users of R on other platforms, notably the default location of the personal library directory (under ~/Library/R, e.g. ~/Library/R/arm64/4.4/library), and that warnings, messages and other output to stderr are highlighted in bold.

Those using the zsh shell (the default for new user accounts) might find the command R being masked by the zsh builtin r (which recalls commands). One can use a full path to R in an alias, or add disable r to ~/.zshrc.

It has been reported that running R.APP may fail if no preferences are stored, so if it fails when launched for the very first time, try it again (the first attempt will store some preferences).

Users of R.APP need to be aware of the 'App Nap' feature (https://developer.apple. com/library/archive/releasenotes/MacOSX/WhatsNewInOSX/Articles/MacOSX10\_9. html) which can cause R tasks to appear to run very slowly when not producing output in the console. Here are ways to avoid it:

- Ensure that the console is completely visible (or at least the activity indicator at the top right corner is visible).
- In a Terminal, run

```
defaults write org.R-project.R NSAppSleepDisabled -bool YES
```

(see https://developer.apple.com/library/archive/releasenotes/MacOSX/WhatsNewInOSX/Articles/MacOSX10\_9.html).

Using the X11 graphics device or the X11-based versions of View() and edit() for data frames and matrices (the latter are the default for command-line R but not R.APP) requires XQuartz (https://www.xquartz.org/) to be installed.

Under some rather nebulous circumstances messages have been seen from fontconfig about missing/unreadable configuration files when using cairo-based devices, especially X11(type = "cairo"). With XQuartz installed there are two fontconfig areas from different versions and it can help to set

```
setenv FONTCONFIG_PATH /opt/X11/lib/X11/fontconfig
```

Another symptom has been that italic/oblique fonts are replaced by upright ones.

## 4.2 Uninstalling under macOS

R for macOS consists of two parts: the GUI (R.APP) and the R framework. Un-installation is as simple as removing those folders (e.g. by dragging them onto the Bin<sup>3</sup>). The typical installation will install the GUI into the /Applications/R.app folder and the R framework into the /Library/Frameworks/R.framework folder. The links to R and Rscript in /usr/local/bin should also be removed.

If you want to get rid of R more completely using a Terminal, simply run:

sudo rm -Rf /Library/Frameworks/R.framework /Applications/R.app \
 /usr/local/bin/R /usr/local/bin/Rscript

The installation consists of up to four Apple packages:<sup>4</sup> for the 'Apple Silicon' build, org.R-project.arm64.R.fw.pkg, org.R-project.arm64.R.GUI.pkg, org.r-project.arm64.tcltk and org.r-project.arm64.texinfo. You can use sudo pkgutil --forget if you want the Apple Installer to forget about the package without deleting its files (useful for the R framework when installing multiple R versions in parallel), or after you have deleted the files. NB: the package names are case-sensitive and the R domain is named inconsistently.

 $<sup>^{3}\,</sup>$  Formerly known as the Trash.

<sup>&</sup>lt;sup>4</sup> At the time of writing: use pkgutil --pkgs | grep -i org.r-project to check.

Uninstalling the Tcl/Tk and Texinfo components (which are installed under /opt/R/arm64 on a 'arm64' build and /opt/R/x86\_64 for an 'x86\_64' one) is not as simple. You can list the files they installed in a Terminal by e.g.

pkgutil --files org.r-project.arm64.tcltk
pkgutil --files org.r-project.arm64.texinfo

(For the 'Intel build, replace arm64 by  $x86_64$ .) These are paths relative to /, the root of the file system.

If you are not compiling R nor installing packages from source you could remove all of /opt/R/arm64 or /opt/R/x86\_64.

## 4.3 Multiple versions

The installer will remove any previous version<sup>5</sup> of the R framework which it finds installed. This can be avoided by using pkgutil --forget (see the previous section). However, note that different versions are installed under /Library/Frameworks/R.framework/Versions as 4.4-arm64 (or  $4.4-x86_{64}$ ), 4.3 and so on, so it is not possible to have different '4.x.y' versions installed for the same 'x' and CPU type.

R.APP will always run the 'current' version of R, that is the last installed version.

 $<sup>^5</sup>$  More precisely, of the Apple package of the same name: this means that ARM and Intel versions can be installed together.

# 5 Running R

How to start R and what command-line options are available is discussed in Section "Invoking R" in An Introduction to R.

You should ensure that the shell has set adequate resource limits: R expects a stack size of at least 8MB and to be able to open at least 256 file descriptors. (Any modern OS should have default limits at least as large as these, but apparently NetBSD may not. Use the shell command ulimit (sh/bash) or limit (csh/tcsh) to check.) For some compilers<sup>1</sup> and packages a larger stack size has been needed: 20-25MB has sufficed to date.

R makes use of a number of environment variables, the default values of many of which are set in file  $R_HOME/etc/Renviron$  (there are none set by default on Windows and hence no such file). These are set at configure time, and you would not normally want to change them – a possible exception is  $R_PAPERSIZE$  (see Section B.3.1 [Setting paper size], page 53). The paper size will be deduced from the 'LC\_PAPER' locale category if it exists and  $R_PAPERSIZE$  is unset, and this will normally produce the right choice from 'a4' and 'letter' on modern Unix-alikes (but can always be overridden by setting  $R_PAPERSIZE$ ).

Various environment variables can be set to determine where R creates its per-session temporary directory. The environment variables TMPDIR, TMP and TEMP are searched in turn and the first one which is set and points to a writable area is used. If none do, the final default is /tmp on Unix-alikes and the value of R\_USER on Windows. The path should be an absolute path not containing spaces<sup>2</sup> (and it is best to avoid non-alphanumeric characters such as + or quotes).

Some Unix-alike systems are set up to remove files and directories periodically from /tmp, for example by a cron job running tmpwatch. Set TMPDIR to another directory before starting long-running jobs on such a system.

Note that TMPDIR will be used to execute configure scripts when installing packages, so if /tmp has been mounted as 'noexec', TMPDIR needs to be set to a directory from which execution is allowed.

22

<sup>&</sup>lt;sup>1</sup> Including GCC 9 on Linux.

 $<sup>^2\,</sup>$  On Windows a path containing spaces will be replaced by the 'short path' version if that does not contain spaces.

# 6 Add-on packages

It is helpful to use the correct terminology. A *package* is loaded from a *library* by the function **library()**. Thus a library is a directory containing installed packages; the main library is  $R_HOME/library$ , but others can be used, for example by setting the environment variable  $R_LIBS$  or using the R function .libPaths(). To avoid any confusion you will often see a library directory referred to as a 'library tree'.

# 6.1 Default packages

The set of packages loaded on startup is by default

```
> getOption("defaultPackages")
[1] "datasets" "utils" "grDevices" "graphics" "stats" "methods"
```

(plus, of course, **base**) and this can be changed by setting the option in startup code (e.g. in ~/.Rprofile). It is initially set to the value of the environment variable R\_DEFAULT\_PACKAGES if set (as a comma-separated list). Setting R\_DEFAULT\_PACKAGES=NULL ensures that only package base is loaded.

Changing the set of default packages is normally used to reduce the set for speed when scripting: in particular not using **methods** will reduce the start-up time by a factor of up to two. But it can also be used to customize R, e.g. for class use. Rscript also checks the environment variable R\_SCRIPT\_DEFAULT\_PACKAGES; if set, this takes precedence over R\_DEFAULT\_PACKAGES.

# 6.2 Managing libraries

R packages are installed into *libraries*, which are directories in the file system containing a subdirectory for each package installed there.

R comes with a single library,  $R_HOME/library$  which is the value of the R object '.Library' containing the standard and recommended<sup>1</sup> packages. Both sites and users can create others and make use of them (or not) in an R session. At the lowest level '.libPaths()' can be used to add paths to the collection of libraries or to report the current collection.

R will automatically make use of a site-specific library  $R\_HOME/site-library$  if this exists (it does not in a vanilla R installation). This location can be overridden by setting<sup>2</sup> '.Library.site' in  $R\_HOME/etc/Rprofile.site$ , or (not recommended) by setting the environment variable  $R\_LIBS\_SITE$ .

Users can have one or more libraries, normally specified by the environment variable R\_ LIBS\_USER. This has a default value (to see it, use 'Sys.getenv("R\_LIBS\_USER")' within an R session), but that is only used if the corresponding directory actually exists (which by default it will not).

Both  $R\_LIBS\_USER$  and  $R\_LIBS\_SITE$  can specify multiple library paths, separated by colons (semicolons on Windows).

# 6.3 Installing packages

Packages may be distributed in source form or compiled binary form. Installing source packages which contain C/C++/Fortran code requires that compilers and related tools be installed. Binary packages are platform-specific and generally need no special tools to install, but see the documentation for your platform for details.

<sup>&</sup>lt;sup>1</sup> unless they were excluded in the build.

 $<sup>^2</sup>$  its binding is locked once the startup files have been read, so users cannot easily change it. See ?.libPaths for how to make use of the new value.

Note that you may need to specify implicitly or explicitly the library to which the package is to be installed. This is only an issue if you have more than one library, of course.

Ensure that the environment variable TMPDIR is either unset (and /tmp exists and can be written in and executed from) or is the absolute path to a valid temporary directory, not containing spaces.

For most users it suffices to call 'install.packages(pkgname)' or its GUI equivalent if the intention is to install a CRAN package and internet access is available.<sup>3</sup> On most systems 'install.packages()' will allow packages to be selected from a list box (typically with thousands of items).

To install packages from source on a Unix-alike use in a terminal

R CMD INSTALL -1 /path/to/library pkg1 pkg2 ...

The part '-l /path/to/library' can be omitted, in which case the first library of a normal R session is used (that shown by .libPaths()[1]).

There are a number of options available: use R CMD INSTALL --help to see the current list.

Alternatively, packages can be downloaded and installed from within R. First choose your nearest CRAN mirror using chooseCRANmirror(). Then download and install packages **pkg1** and **pkg2** by

> install.packages(c("pkg1", "pkg2"))

The essential dependencies of the specified packages will also be fetched. Unless the library is specified (argument lib) the first library in the library search path is used: if this is not writable, R will ask the user (in an interactive session) if the default personal library should be created, and if allowed to will install the packages there.

If you want to fetch a package and all those it depends on (in any way) that are not already installed, use e.g.

> install.packages("Rcmdr", dependencies = TRUE)

install.packages can install a source package from a local .tar.gz file (or a URL to such a file) by setting argument repos to NULL: this will be selected automatically if the name given is a single .tar.gz file.

install.packages can look in several repositories, specified as a character vector by the argument repos: these can include a CRAN mirror, Bioconductor, R-forge, rforge.net, local archives, local files, ...). Function setRepositories() can select amongst those repositories that the R installation is aware of.

Something which sometimes puzzles users is that install.packages() may report that a package which they believe should be available is not found. Some possible reasons:

- The package, such as grid or tcltk, is part of R itself and not otherwise available.
- The package is not in the available repositories, so check which have been selected by

getOption("repos")

• The package is available, but not for the current version of R or for the type of OS (Unix/Windows). To retrieve the information on available versions of package *pkg*, use

```
av <- available.packages(filters=list())
av[av[, "Package"] == pkg, ]</pre>
```

in your R session, and look at the 'Depends' and 'OS\_type' fields (there may be more than one matching entry). If the package depends on a version of R later than the one in use, it is possible that an earlier version is available which will work with your version of R: for CRAN look for 'Old sources' on the package's CRAN landing page and manually retrieve an appropriate version (of comparable age to your version of R).

 $<sup>^3</sup>$  If a proxy needs to be set, see <code>?download.file</code>.

Naive users sometimes forget that as well as installing a package, they have to use library to make its functionality available.

### 6.3.1 Windows

What install.packages does by default is different on Unix-alikes (except macOS) and Windows. On Unix-alikes it consults the list of available *source* packages on CRAN (or other repositories), downloads the latest version of the package sources, and installs them (via R CMD INSTALL). On 'x86\_64' Windows it looks (by default) first at the list of *binary* versions of packages available for your version of R and downloads the latest versions (if any). If no binary version is available or the source version is newer, it will install the source versions of packages without compiled C/C++/Fortran code, and offer to do so for those with, if make is available (and this can be tuned by option "install.packages.compile.from.source").

[At present binary packages are not distributed for 'aarch64' Windows, so most of this subsection only applies to 'x86\_64'.]

On Windows install.packages can also install a binary package from a local zip file (or the URL of such a file) by setting argument repos to NULL. Rgui.exe has a menu Packages with a GUI interface to install.packages, update.packages and library.

Windows binary packages for R were distributed as a single binary containing either or both architectures (32- and 64-bit). Prior to R 4.2.0, they might contain only the 32-bit architecture.

R CMD INSTALL works in Windows to install source packages. No additional tools are needed if the package does not contain compiled code, and install.packages(type="source") will work for such packages. Those with compiled code need the tools (see Section 3.1.1 [The Windows toolset], page 16). The tools are found automatically by R when installed by the toolset installer. See Building R-devel and packages on Windows (https://cran.r-project.org/bin/ windows/base/howto-R-devel.html) for more details.

Occasional permission problems after unpacking source packages have been seen on some systems: these have been circumvented by setting the environment variable R\_INSTALL\_TAR to 'tar.exe'.

If you have only a source package that is known to work with current R and just want a binary Windows build of it, you could make use of the building service offered at https://win-builder.r-project.org/.

For almost all packages R CMD INSTALL will attempt to install both 32- and 64-bit builds of a package if run from a 32/64-bit install of R (only 64-bit builds and installs are supported since R 4.2.0). It will report success if the installation of the architecture of the running R succeeded, whether or not the other architecture was successfully installed. The exceptions are packages with a non-empty configure.win script or which make use of src/Makefile.win. If configure.win does something appropriate to both architectures use<sup>4</sup> option --force-biarch: otherwise R CMD INSTALL --merge-multiarch can be applied to a source tarball to merge separate 32- and 64-bit installs. (This can only be applied to a tarball, and will only succeed if both installs succeed.)

If you have a package without compiled code and no Windows-specific help, you can zip up an installation on another OS and install from that zip file on Windows. However, such a package can be installed from the sources on Windows without any additional tools.

### 6.3.2 macOS

On macOS install.packages works as it does on other Unix-alike systems, but there is an additional type binary (available for the CRAN distribution but not when compiling R from

<sup>&</sup>lt;sup>4</sup> for a small number of CRAN packages where this is known to be safe and is needed by the autobuilder this is the default. Look at the source of tools:::.install\_packages for the list. It can also be specified in the package's DESCRIPTION file.

source) which can be passed to install.packages in order to download and install binary packages from a suitable repository. These binary package files for macOS have the extension '.tgz'. The R.APP GUI provides menus for installation of either binary or source packages, from CRAN, other repositories or local files.

On R builds using binary packages, the default is type **both**: this looks first at the list of binary packages available for your version of R and installs the latest versions (if any). If no binary version is available or the source version is newer, it will install the source versions of packages without compiled C/C++/Fortran code and offer to do so for those with, if make is available.

Note that most binary packages which include compiled code are tied to a particular series (e.g. R 4.5.x or 4.4.x) of R.

Installing source packages which do not contain compiled code should work with no additional tools. For others you will need the 'Command Line Tools' for Xcode and compilers which match those used to build R, plus a Fortran compiler for packages which contain Fortran code: see Section C.3 [macOS], page 61. Packages with C/C++ source code which link to Fortran libraries (which include the BLAS and LAPACK libraries) will need either the Fortran compiler or, for CRAN binary distributions of R, to specify

FLIBS = -L/Library/Frameworks/R.framework/Resources/lib -lgfortran -lquadmath in a Makevars file (see the next section) to point to the Fortran libraries in the binary distribution.

Package rJava (https://CRAN.R-project.org/package=rJava) and those which depend on it need a Java runtime installed and several packages need X11 installed, including those using Tk. See Section C.3 [macOS], page 61, and Section C.3.7 [Java (macOS)], page 68. Package rjags (https://CRAN.R-project.org/package=rjags) needs a build of JAGS installed under /usr/local, such as those at https://sourceforge.net/projects/mcmc-jags/files/JAGS/ 4.x/Mac%200S%20X/.

Tcl/Tk extension BWidget used to be distributed with R but no longer is; Tktable has been distributed with recent versions of R.

The default compilers specified are shown in file /Library/Frameworks/R.framework/Resources/etc/Makeconf. At the time of writing those settings assumed that the C, Fortran and C++ compilers were on the path (see Section C.3 [macOS], page 61). The settings can be changed, either by editing that file or in a file such as ~/.R/Makevars (see the next section). Entries which may need to be changed include 'CC', 'CXX', 'FC', 'FLIBS' and the corresponding flags, and perhaps 'CXXCPP', 'DYLIB\_LD', 'MAIN\_LD', 'SHLIB\_CXXLD' and 'SHLIB\_LD', as well as their 'CXX11', 'CXX14', 'CXX17' and 'CXX20' variants.

So for example you could select a specific LLVM clang for both C and C++ with extensive checking by having in  $^{/.R/Makevars}$ 

```
SDK=/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk
CC = /usr/local/clang/bin/clang -isysroot $(SDK)
CXX = /usr/local/clang/bin/clang++ -isysroot $(SDK)
CXX11 = $CXX
CXX14 = $CXX
CXX17 = $CXX
CXX20 = $CXX
CX20 = $CXX
CX22 = $CXX
CX23 = $CXX
CFLAGS = -g -02 -Wall -pedantic -Wconversion -Wno-sign-conversion
CXXFLAGS = -g -02 -Wall -pedantic -Wconversion -Wno-sign-conversion
CXX11FLAGS = $CXXFLAGS
CXX14FLAGS = $CXXFLAGS
CXX14FLAGS = $CXXFLAGS
```

CXX2OFLAGS = \$CXXFLAGS CXX23FLAGS = \$CXXFLAGS

(the current SDK can be found by running xcrun -show-sdk-path) and for the current macOS distribution of gfortran at https://mac.r-project.org/tools/

```
FC = /opt/gfortran/bin/gfortran
(arm64)
FLIBS = -L/opt/gfortran/lib/gcc/aarch64-apple-darwin20.0/12.2.0
    -L/opt/gfortran/lib -lgfortran -lemutls_w -lquadmath
(Intel)
FLIBS = -L/opt/gfortran/lib/gcc/x86_64-apple-darwin20.0/12.2.0
    -L/opt/gfortran/lib -lgfortran -lquadmath
```

(line broken here for the manual only).

If that clang build supports OpenMP, you can add

```
SHLIB_OPENMP_CFLAGS = -fopenmp
SHLIB_OPENMP_CXXFLAGS = -fopenmp
```

to compile OpenMP-using packages. It will also be necessary to arrange for the libomp.dylib library to be found at both install time and run time, for example by copying/linking it somewhere that is searched such as /usr/local/lib.

Apple includes many Open Source libraries in macOS but increasingly without the corresponding headers (not even in Xcode nor the Command Line Tools): they are often rather old versions. If installing packages from source using them it is usually easiest to install a statically-linked up-to-date copy of the Open Source package from its sources or from https:// mac.r-project.org/bin/. But sometimes it is desirable/necessary to use Apple's dynamically linked library, in which case appropriate headers could be extracted from the sources<sup>5</sup> available *via* https://opensource.apple.com/releases – this has been used for iodbc.

Some care may be needed with selecting compilers when installing external software for use with packages. The 'system' compilers as used when building R are clang and clang++, but the Apple toolchain also provides compilers called gcc and g++ which despite their names are based on LLVM and libc++ like the system ones and which behave in almost the same way as the system ones. Most Open Source software has a configure script developed using GNU autoconf and hence will select gcc and g++ as the default compilers: this usually works fine. For consistency one can use

./configure CC=clang CFLAGS=-02 CXX=clang++ CXXFLAGS=-02

(avoiding autoconf's default -g). R CMD INSTALL and install.packages() try to invoke configure with the same compilers and flags used to build R.

### 6.3.3 Customizing package compilation

The R system and package-specific compilation flags can be overridden or added to by setting the appropriate Make variables in the personal file  $HOME/.R/Makevars-R_PLATFORM$  (but HOME/.R/Makevars.win or HOME/.R/Makevars.win64 on Windows), or if that does not exist, HOME/.R/Makevars, where 'R\_PLATFORM' is the platform for which R was built, as available in the platform component of the R variable R.version. The full path to an alternative personal file<sup>6</sup> can be specified *via* the environment variable R\_MAKEVARS\_USER.

Package developers are encouraged to use this mechanism to enable a reasonable amount of diagnostic messaging ("warnings") when compiling, such as e.g. -Wall -pedantic for tools from GCC, the GNU Compiler Collection, and for LLVM (clang and flang-new).

 $<sup>^5\,</sup>$  Note that capitalization and versioning may differ from the Open Source project.

 $<sup>^{6}\,</sup>$  using a path containing spaces is likely to cause problems

Note that this mechanism can also be used when it is necessary to change the optimization level whilst installing a particular package. For example

```
## for C code
CFLAGS = -g -0 -mtune=native
## for C++ code
CXXFLAGS = -g -0 -mtune=native
## for C++11 code
CXX11FLAGS = -g -0 -mtune=native
## for fixed-form Fortran code
FFLAGS = -g -0 -mtune=native
## for C17 code
C17FLAGS = -g -0 -mtune=native -Wno-strict-prototypes
```

Note that if you have specified a non-default C++ or C standard, you need to set the flag(s) appropriate to that standard.

Another use is to override the settings in a binary installation of R. For example, for the current distribution of gfortran at https://mac.r-project.org/tools/

```
FC = /opt/gfortran/bin/gfortran
(arm64)
FLIBS = -L/opt/gfortran/lib/gcc/aarch64-apple-darwin20.0/12.2.0
-L/opt/gfortran/lib -lgfortran -lemutls_w -lquadmath
(Intel)
FLIBS = -L/opt/gfortran/lib/gcc/x86_64-apple-darwin20.0/12.2.0
-L/opt/gfortran/lib -lgfortran -lquadmath
```

(line broken here for the manual only).

There is also provision for a site-wide Makevars.site file under  $R_HOME/etc$  (in a subarchitecture-specific directory if appropriate). This is read immediately after Makeconf, and the path to an alternative file can be specified by environment variable  $R_MAKEVARS_SITE$ .

Note that these mechanisms do not work with packages which fail to pass settings down to sub-makes, perhaps reading etc/Makeconf in makefiles in subdirectories. Fortunately such packages are unusual.

### 6.3.4 Multiple sub-architectures

When installing packages from their sources, there are some extra considerations on installations which use sub-architectures. These were commonly used on Windows prior to R 4.2.0 but can in principle be used on other platforms.

When a source package is installed by a build of R which supports multiple sub-architectures, the normal installation process installs the packages for all sub-architectures. The exceptions are

Unix-alikes

where there is an configure script, or a file src/Makefile.

Windows

where there is a non-empty configure.win script, or a file src/Makefile.win (with some exceptions where the package is known to have an architecture-independent configure.win, or if --force-biarch or field 'Biarch' in the DESCRIPTION file is used to assert so).

In those cases only the current architecture is installed. Further sub-architectures can be installed by

R CMD INSTALL -- libs-only pkg

using the path to R or R --arch to select the additional sub-architecture. There is also R CMD INSTALL --merge-multiarch to build and merge the two architectures, starting with a source tarball.

### 6.3.5 Byte-compilation

Packages are by default byte-compiled on installation. Byte-compilation can be controlled on a per-package basis by the 'ByteCompile' field in the DESCRIPTION file.

### 6.3.6 External software

Some R packages contain compiled code which links to external software libraries. Unless the external library is statically linked (which is done as much as possible for binary packages on Windows and macOS), the libraries have to be found when the package is loaded and not just when it is installed. How this should be done depends on the OS (and in some cases the version).

For Unix-alikes except macOS the primary mechanism is the ld.so cache controlled by ldconfig: external dynamic libraries recorded in that cache will be found. Standard library locations will be covered by the cache, and well-designed software will add its locations (as for example openmpi does on Fedora). The secondary mechanism is to consult the environment variable LD\_LIBRARY\_PATH. The R script controls that variable, and sets it to the concatenation of R\_LD\_LIBRARY\_PATH, R\_JAVA\_LD\_LIBRARY\_PATH and the environment value of LD\_LIBRARY\_PATH. The first two have defaults which are normally set when R is installed (but can be overridden in the environment) so LD\_LIBRARY\_PATH is the best choice for a user to set.

On macOS the primary mechanism is to embed the absolute path to dependent dynamic libraries into an object when it is compiled. Few R packages arrange to do so, but it can be edited<sup>7</sup> via install\_name\_tool — that only deals with direct dependencies and those would also need to be compiled to include the absolute paths of their dependencies. If the choice of absolute path is to be deferred to load time, how they are resolved is described in man dyld: the role of LD\_LIBRARY\_PATH is replaced on macOS by DYLD\_LIBRARY\_PATH and DYLD\_FALLBACK\_LIBRARY\_PATH. Running R CMD otool -L on the package shared object will show where (if anywhere) its dependencies are resolved. DYLD\_FALLBACK\_LIBRARY\_PATH is preferred (and it is that which is manipulated by the R script), but as from 10.11 ('El Capitan') the default behaviour had been changed for security reasons to discard these environment variables when invoking a shell script (and R is a shell script). That makes the only portable option to set R\_LD\_LIBRARY\_PATH in the environment, something like

```
export R_LD_LIBRARY_PATH="`R RHOME`/lib:/opt/local/lib"
```

The precise rules for where Windows looks for DLLs are complex and depend on the version of Windows. But for present purposes the main solution is to put the directories containing the DLLs the package links to (and any those DLLs link to) on the PATH.

The danger with any of the methods which involve setting environment variables is of inadvertently masking a system library. This is less for DYLD\_FALLBACK\_LIBRARY\_PATH and for *appending* to PATH on Windows (as it should already contain the system library paths).

## 6.4 Updating packages

The command update.packages() is the simplest way to ensure that all the packages on your system are up to date. It downloads the list of available packages and their current versions, compares it with those installed and offers to fetch and install any that have later versions on the repositories.

An alternative interface to keeping packages up-to-date is provided by the command packageStatus(), which returns an object with information on all installed packages and packages available at multiple repositories. The print and summary methods give an overview of

<sup>&</sup>lt;sup>7</sup> They need to have been created using -headerpad\_max\_install\_names, which is the default for an R package.

installed and available packages, the upgrade method offers to fetch and install the latest versions of outdated packages.

One sometimes-useful additional piece of information that packageStatus() returns is the status of a package, as "ok", "upgrade" or "unavailable" (in the currently selected repositories). For example

# 6.5 Removing packages

Packages can be removed in a number of ways. From a command prompt they can be removed by

R CMD REMOVE -1 /path/to/library pkg1 pkg2 ...

From a running R process they can be removed by

Finally, one can just remove the package directory from the library.

## 6.6 Setting up a package repository

Utilities such as install.packages can be pointed at any CRAN-style repository, and R users may want to set up their own. The 'base' of a repository is a URL such as https://www.stats.ox.ac.uk/pub/RWin/: this must be an URL scheme that download.packages supports (which also includes 'https://', 'ftp://' and 'file://'). Under that base URL there should be directory trees for one or more of the following types of package distributions:

- "source": located at src/contrib and containing .tar.gz files. Other forms of compression can be used, e.g. .tar.bz2 or .tar.xz files. Complete repositories contain the sources corresponding to any binary packages, and in any case it is wise to have a src/contrib area with a possibly empty PACKAGES file.
- "binary": binary packages are located under bin/os/build/contrib/x.y for R versions x.y.z, where os is either windows for Windows or macosx for macOS and OS X. The build variable is defined by the third part of the type value after binary and may be empty. For example, in R 4.4.1 "mac.binary.big-sur-arm64" specifies the bin/macosx/big-sur-arm64/4.4 location, while "win.binary" specifies bin/windows/contrib/4.4. The platform and build specification in CRAN R binaries can be obtained from .Platform\$pkgType. Windows binary package files use .zip extension while Mac files use .tgz.

Each terminal directory must also contain a PACKAGES file. This can be a concatenation of the DESCRIPTION files of the packages separated by blank lines, but only a few of the fields are needed. The simplest way to set up such a file is to use function write\_PACKAGES in the tools package, and its help explains which fields are needed. Optionally there can also be PACKAGES.rds and PACKAGES.gz files, downloaded in preference to PACKAGES. (If you have a mis-configured server that does not report correctly non-existent files you may need these files.)

To add your repository to the list offered by **setRepositories()**, see the help file for that function.

Incomplete repositories are better specified *via* a **contriburl** argument than *via* being set as a repository.

A repository can contain subdirectories, when the descriptions in the PACKAGES file of packages in subdirectories must include a line of the form

Path: path/to/subdirectory

-once again write\_PACKAGES is the simplest way to set this up.

### 6.7 Checking installed source packages

It can be convenient to run R CMD check on an installed package, particularly on a platform which uses sub-architectures. The outline of how to do this is, with the source package in directory pkg (or a tarball filename):

R CMD INSTALL -1 libdir pkg > pkg.log 2>&1

R CMD check -1 libdir --install=check:pkg.log pkg

Where sub-architectures are in use the R CMD check line can be repeated with additional architectures by

R --arch arch CMD check -1 libdir --extra-arch --install=check:pkg.log pkg where --extra-arch selects only those checks which depend on the installed code and not those which analyse the sources. (If multiple sub-architectures fail only because they need different settings, e.g. environment variables, --no-multiarch may need to be added to the INSTALL lines.) On Unix-alikes the architecture to run is selected by --arch: this can also be used on Windows with  $R_HOME/bin/R.exe$ , but it is more usual to select the path to the Rcmd.exe of the desired architecture.

So on Windows to install, check and package for distribution a source package from a tarball which has been tested on another platform one might use

.../bin/x64/Rcmd INSTALL -1 libdir tarball --build > pkg.log 2>&1

## 7 Internationalization and Localization

*Internationalization* refers to the process of enabling support for many human languages, and *localization* to adapting to a specific country and language.

Current builds of R support all the character sets that the underlying OS can handle. These are interpreted according to the current locale, a sufficiently complicated topic to merit a separate section. Note though that R has no built-in support for right-to-left languages and bidirectional output, relying on the OS services. For example, how character vectors in UTF-8 containing both English digits and Hebrew characters are printed is OS-dependent (and perhaps locale-dependent).

The other aspect of the internationalization is support for the translation of messages. This is enabled in almost all builds of R.

## 7.1 Locales

A *locale* is a description of the local environment of the user, including the preferred language, the encoding of characters, the currency used and its conventions, and so on. Aspects of the locale are accessed by the R functions Sys.getlocale and Sys.localeconv.

The system of naming locales is OS-specific. There is quite wide agreement on schemes, but not on the details of their implementation. A locale needs to specify

- A human language. These are generally specified by a lower-case two-character abbreviation following ISO 639 (see e.g. https://en.wikipedia.org/wiki/ISO\_639-1).
- A 'territory', used mainly to specify the currency. These are generally specified by an uppercase two-character abbreviation following ISO 3166 (see e.g. https://en.wikipedia.org/ wiki/ISO\_3166).
- A charset encoding, which determines both how a byte stream should be divided into characters, and which characters the subsequences of bytes represent. Sometimes the combination of language and territory is used to specify the encoding, for example to distinguish between traditional and simplified Chinese.
- Optionally, a modifier, for example to indicate that Austria is to be considered pre- or post-Euro. The modifier is also used to indicate the script (@latin, @cyrillic for Serbian, @iqtelif) or language dialect (e.g. @saaho, a dialect of Afar, and @bokmal and @nynorsk, dialects of Norwegian regarded by some OSes as separate languages, no and nn).

R is principally concerned with the first (for translations) and third. Note that the charset may be deducible from the language, as some OSes offer only one charset per language.

### 7.1.1 Locales under Unix-alikes

Modern Linux uses the XPG<sup>1</sup> locale specifications which have the form 'en\_GB', 'en\_GB.UTF-8', 'aa\_ER.UTF-80saaho', 'de\_AT.iso8859150euro', the components being in the order listed above. (See man locale and locale -a for more details.) Similar schemes are used by most Unix-alikes: some (including some distributions of Linux) use '.utf8' rather than '.UTF-8'.

Note that whereas UTF-8 locales are nowadays almost universally used, locales such as 'en\_GB' use 8-bit encodings for backwards compatibility.

#### 7.1.2 Locales under Windows

Windows also uses locales, but specified in a rather less concise way. Most users will encounter locales only via drop-down menus, but more information and lists can be found by searching for 'Windows language country strings').

<sup>&</sup>lt;sup>1</sup> 'X/Open Portability Guide', which has had several versions.

It offers only one encoding per language.

Some care is needed with Windows' locale names. For example, chinese is Traditional Chinese and not Simplified Chinese as used in most of the Chinese-speaking world.

### 7.1.3 Locales under macOS

macOS supports locales in its own particular way, but the R GUI tries to make this easier for users. See https://developer.apple.com/library/archive/documentation/MacOSX/ Conceptual/BPInternational/ for how users can set their locales. End users will generally only see lists of languages/territories. Users of R in a terminal may need to set the locale to something like 'en\_GB.UTF-8' if it defaults to 'C' (as it sometimes does when logging in remotely and for batch jobs: note whether Terminal sets the LANG environment variable is an (advanced) preference, but does so by default).

Internally macOS uses a form similar to Linux: the main difference from other Unix-alikes is that where a character set is not specified it is assumed to be UTF-8.

### 7.2 Localization of messages

The preferred language for messages is by default taken from the locale. This can be overridden first by the setting of the environment variable LANGUAGE and then<sup>2</sup> by the environment variables LC\_ALL, LC\_MESSAGES and LANG. (The last three are normally used to set the locale and so should not be needed, but the first is only used to select the language for messages.) The code tries hard to map locales to languages, but on some systems (notably Windows) the locale names needed for the environment variable LC\_ALL do not all correspond to XPG language names and so LANGUAGE may need to be set. (One example is 'LC\_ALL=es' on Windows which sets the locale to Estonian and the language to Spanish.)

It is usually possible to change the language once R is running *via* (not Windows) Sys.setlocale("LC\_MESSAGES", "new\_locale"), or by setting an environment variable such as LANGUAGE, *provided*<sup>3</sup> the language you are changing to can be output in the current character set. But this is OS-specific, and has been known to stop working on an OS upgrade. Note that translated messages may be cached, so attempting to change the language of an error that has already been output in another language may not work.

Messages are divided into *domains*, and translations may be available for some or all messages in a domain. R makes use of the following domains.

- Domain R for the C-level error and warning messages from the R interpreter.
- Domain R-pkg for the R stop, warning and message messages in each package, including R-base for the base package.
- Domain *pkg* for the C-level messages in each package.
- Domain RGui for the menus etc of the R for Windows GUI front-end.

Dividing up the messages in this way allows R to be extensible: as packages are loaded, their message translation catalogues can be loaded too.

R can be built without support for translations, but it is enabled by default.

R-level and C-level domains are subtly different, for example in the way strings are canonicalized before being passed for translation.

Translations are looked for by domain according to the currently specified language, as specifically as possible, so for example an Austrian ('de\_AT') translation catalogue will be used in preference to a generic German one ('de') for an Austrian user. However, if a specific translation

 $<sup>^2\,</sup>$  On some systems setting LC\_ALL or LC\_MESSAGES to 'C' disables LANGUAGE.

 $<sup>^3</sup>$  If you try changing from French to Russian except in a UTF-8 locale, you may find messages change to English.

catalogue exists but does not contain a translation, the less specific catalogues are consulted. For example, R has catalogues for 'en\_GB' that translate the Americanisms (e.g., 'gray') in the standard messages into English.<sup>4</sup> Two other examples: there are catalogues for 'es', which is Spanish as written in Spain and these will by default also be used in Spanish-speaking Latin American countries, and also for 'pt\_BR', which are used for Brazilian locales but not for locales specifying Portugal.

Translations in the right language but the wrong charset are made use of by on-the-fly reencoding. The LANGUAGE variable (only) can be a colon-separated list, for example 'se:de', giving a set of languages in decreasing order of preference. One special value is 'en@quot', which can be used in a UTF-8 locale to have American error messages with pairs of single quotes translated to Unicode directional quotes.

If no suitable translation catalogue is found or a particular message is not translated in any suitable catalogue, 'English'<sup>5</sup> is used.

See https://developer.r-project.org/Translations30.html for how to prepare and install translation catalogues.

 $<sup>^4</sup>$  the language written in England: some people living in the USA appropriate this name for their language.

 $<sup>^5</sup>$  with Americanisms.

# 8 Choosing between 32- and 64-bit builds

As from R 4.4.0 32-bit builds are unsupported.

## 9 The standalone Rmath library

The routines supporting the distribution and special<sup>1</sup> functions in R and a few others are declared in C header file Rmath.h. These can be compiled into a standalone library for linking to other applications. (Note that they are not a separate library when R is built, and the standalone version differs in several ways.)

The makefiles and other sources needed are in directory src/nmath/standalone, so the following instructions assume that is the current working directory (in the build directory tree on a Unix-alike if that is separate from the sources).

Rmath.h contains 'R\_VERSION\_STRING', which is a character string containing the current R version, for example "4.4.0".

There is full access to R's handling of NaN, Inf and -Inf via special versions of the macros and functions

```
ISNAN, R_FINITE, R_log, R_pow and R_pow_di
```

and (extern) constants R\_PosInf, R\_NegInf and NA\_REAL.

There is no support for R's notion of missing values, in particular not for NA\_INTEGER nor the distinction between NA and NaN for doubles.

A little care is needed to use the random-number routines. You will need to supply the uniform random number generator

```
double unif_rand(void)
```

or use the one supplied (and with a shared library or DLL you may have to use the one supplied, which is the Marsaglia-multicarry with an entry point

```
set_seed(unsigned int, unsigned int)
```

to set its seeds).

The facilities to change the normal random number generator are available through the constant  $NO1_kind$ . This takes values from the enumeration type

```
typedef enum {
    BUGGY_KINDERMAN_RAMAGE,
    AHRENS_DIETER,
    BOX_MULLER,
    USER_NORM,
    INVERSION,
    KINDERMAN_RAMAGE
} NO1+umot
```

```
} N01type;
```

(and 'USER\_NORM' is not available).

## 9.1 Unix-alikes

If R has not already been made in the directory tree, **configure** must be run as described in the main build instructions.

Then (in src/nmath/standalone)

make

will make standalone libraries libRmath.a and libRmath.so (libRmath.dylib on macOS): 'make static' and 'make shared' will create just one of them.

To use the routines in your own C or C++ programs, include

#define MATHLIB\_STANDALONE

<sup>&</sup>lt;sup>1</sup> e.g. Bessel, beta and gamma functions

#### #include <Rmath.h>

and link against '-lRmath' (and '-lm' if needed on your OS). The example file test.c does nothing useful, but is provided to test the process (via make test). Note that you will probably not be able to run it unless you add the directory containing libRmath.so to the LD\_LIBRARY\_PATH environment variable (libRmath.dylib, DYLD\_FALLBACK\_LIBRARY\_PATH on macOS).

The targets

make install make uninstall

will (un)install the header Rmath.h and shared and static libraries (if built). Both prefix= and DESTDIR are supported, together with more precise control as described for the main build.

'make install' installs a file for pkg-config to use by e.g.

```
$(CC) `pkg-config --cflags libRmath` -c test.c
$(CC) `pkg-config --libs libRmath` test.o -o test
```

On some systems 'make install-strip' will install a stripped shared library.

### 9.2 Windows

You need to set  $up^2$  almost all the tools to make R and then run (in a Unix-like shell)

```
(cd ../../gnuwin32; make MkRules)
(cd ../../include; make -f Makefile.win config.h Rconfig.h Rmath.h)
make -f Makefile.win
```

Alternatively, in a cmd.exe shell use

```
cd ../../include
make -f Makefile.win config.h Rconfig.h Rmath.h
cd ../nmath/standalone
make -f Makefile.win
```

This creates a static library libRmath.a and a DLL Rmath.dll. If you want an import library libRmath.dll.a (you don't need one), use

make -f Makefile.win shared implib

To use the routines in your own C or C++ programs using MinGW-w64, include

#define MATHLIB\_STANDALONE
#include <Rmath.h>

and link against '-lRmath'. This will use the first found of libRmath.dll.a, libRmath.a and Rmath.dll in that order, so the result depends on which files are present. You should be able to force static or dynamic linking *via* 

```
-Wl,-Bstatic -lRmath -Wl,Bdynamic
-Wl,-Bdynamic -lRmath
```

or by linking to explicit files (as in the 'test' target in Makefile.win: this makes two executables, test.exe which is dynamically linked, and test-static.exe, which is statically linked).

It is possible to link to Rmath.dll using other compilers, either directly or via an import library: if you make a MinGW-w64 import library as above, you will create a file Rmath.def which can be used (possibly after editing) to create an import library for other systems such as Visual C++.

If you make use of dynamic linking you should use

```
#define MATHLIB_STANDALONE
#define RMATH_DLL
```

 $<sup>^2\,</sup>$  including copying MkRules.dist to MkRule.local and selecting the architecture.

#### #include <Rmath.h>

to ensure that the constants like NA\_REAL are linked correctly. (Auto-import will probably work with MinGW-w64, but it is better to be sure. This is likely to also work with VC++, Borland and similar compilers.)

## Appendix A Essential and useful other programs under a Unix-alike

This appendix gives details of programs you will need to build R on Unix-like platforms, or which will be used by R if found by configure.

Remember that some package management systems (such as RPM and Debian/Ubuntu's) make a distinction between the user version of a package and the development version. The latter usually has the same name but with the extension '-devel' or '-dev': you need both versions installed.

## A.1 Essential programs and libraries

You need a means of compiling C and Fortran 90 (see Section B.6 [Using Fortran], page 54). Your C compiler should be ISO/IEC 60059<sup>1</sup>, POSIX 1003.1 and C99-compliant.<sup>2</sup> R tries to choose suitable flags<sup>3</sup> for the C compilers it knows about, but you may have to set CC or CFLAGS suitably. (Note that options essential to run the compiler even for linking, such as those to set the architecture, should be specified as part of CC rather than in CFLAGS.)

Unless you do not want to view graphs on-screen (or use macOS) you need 'X11' installed, including its headers and client libraries. For recent Fedora/RedHat distributions it means (at least) RPMs 'libX11', 'libX11-devel', 'libXt' and 'libXt-devel'. On Debian/Ubuntu we recommend the meta-package 'xorg-dev'. If you really do not want these you will need to explicitly configure R without X11, using --with-x=no.

The command-line editing (and command completion) depends on the GNU readline library (including its headers): version 6.0 or later is needed for all the features to be enabled. Otherwise you will need to configure with --with-readline=no (or equivalent).

A suitably comprehensive iconv function is essential. The R usage requires iconv to be able to translate between "latin1" and "UTF-8", to recognize "" (as the current encoding) and "ASCII", and to translate to and from the Unicode wide-character formats "UCS-[24] [BL]E" — this is true by default for glibc<sup>4</sup> but not of most commercial Unixes. However, you can make use of GNU libiconv (as used on macOS: see https://www.gnu.org/software/libiconv/).

The OS needs to have enough support<sup>5</sup> for wide-character types: this is checked at configuration. Some C99 functions<sup>6</sup> are required and checked for at configuration. A small number of POSIX functions<sup>7</sup> are essential, and others<sup>8</sup> will be used if available.

Installations of zlib (version 1.2.5 or later), libbz2 (version 1.0.6 or later: called bzip2libs/bzip2-devel or libbz2-1.0/libbz2-dev by some Linux distributions) and liblzma<sup>9</sup> version

 $<sup>^1\,</sup>$  also known as IEEE 754

<sup>&</sup>lt;sup>2</sup> Note that C11 compilers need not be C99-compliant: R requires support for double complex and variable-length arrays which are optional in C11 but are mandatory in C99. C17 (also known as C18 as it was published in 2018) is a 'bugfix release' of C11, clarifying the standard. However, all known recent compilers in C11 or C17 mode are C99-compliant, and most default to C17.

 $<sup>^3</sup>$  Examples are <code>-std=gnu99</code>, <code>-std=c99</code> and <code>-c99</code>.

 $<sup>^4\,</sup>$  However, it is possible to break the default behaviour of glibc by re-specifying the gconv modules to be loaded.

<sup>&</sup>lt;sup>5</sup> specifically, the C99 functionality of headers wchar.h and wctype.h, types wctans\_t and mbstate\_t and functions mbrtowc, mbstowcs, wcrtomb, wcscoll, wcstombs, wctrans, wctype, and iswctype.

<sup>&</sup>lt;sup>6</sup> including expm1, hypot, log1p, nearbyint and va\_copy.

<sup>&</sup>lt;sup>7</sup> including opendir, readdir, closedir, popen, stat, glob, access, getcwd and chdir system calls, select on a Unix-alike, and either putenv or setenv.

<sup>&</sup>lt;sup>8</sup> such as realpath, symlink.

<sup>&</sup>lt;sup>9</sup> most often distributed as part of xz: possible names in Linux distributions include xz-devel/xz-libs and liblzma-dev.

5.0.3 or later are required. Some software distributions<sup>10</sup> replace zlib by  $zlib-ng^{11}$  which has an optional compatibility interface.

Either PCRE1 (version 8.32 or later, formerly known as just PCRE) or PCRE2 is required: PCRE2 is preferred and using PCRE1 requires configure option --with-pcre1. Only the 8bit library and headers are needed if these are packaged separately. JIT support (optional) is desirable for the best performance. For PCRE2 >= 10.30 (which is desirable as matching has been re-written not to use recursion and the Unicode tables were updated to version 10)

```
./configure --enable-jit
```

suffices. If building PCRE1 for use with R a suitable configure command might be

./configure --enable-utf --enable-unicode-properties --enable-jit --disable-cpp The --enable-jit flag is supported for most common CPUs but does not work (well or at all) for 'arm64' macOS.

Some packages require the 'Unicode properties' which are optional for PCRE1: support for this and JIT can be checked at run-time by calling pcre\_config().

Library libcurl (version 7.28.0 or later) is required. Information on libcurl is found from the curl-config script: if that is missing or needs to be overridden<sup>12</sup> there are macros to do so described in file config.site.

A tar program is needed to unpack the sources and packages (including the recommended packages). A version<sup>13</sup> that can automagically detect compressed archives is preferred for use with untar(): the configure script looks for gtar and gnutar before tar – use environment variable TAR to override this. (On NetBSD/OpenBSD systems set this to bsdtar if that is installed.)

There need to be suitable versions of the tools grep and sed: the problems are usually with old AT&T and BSD variants. configure will try to find suitable versions (including looking in /usr/xpg4/bin which is used on some commercial Unixes).

You will not be able to build most of the manuals unless you have texi2any version 6.1 or later installed (which requires perl), and if not most of the HTML manuals will be linked to a version on CRAN. To make PDF versions of the manuals you will also need file texinfo.tex installed (which is part of the GNU Texinfo distribution but is often made part of the TEX package texinfo in re-distributions) as well as texi2dvi.<sup>14</sup> Further, the versions of texi2dvi and texinfo.tex need to be compatible: we have seen problems with older TEX distributions.

If you want to build from the R Subversion repository then texi2any is highly recommended as it is used to create files which are in the tarball but not stored in the Subversion repository.

The PDF documentation (including doc/NEWS.pdf) and building vignettes needs pdftex and pdflatex. We require LATEX version 2005/12/01 or later (for UTF-8 support). Building PDF package manuals (including the R reference manual) and vignettes is sensitive to the version of the LATEX package hyperref and we recommend that the TEX distribution used is kept up-to-date. A number of standard LATEX packages are required for the PDF manuals (including url and some of the font packages such as times and helvetic and also amsfonts) and others such as hyperref and inconsolata are desirable (and without them you may need to change R's defaults: see Section 2.3 [Making the manuals], page 5). Note that package

 $<sup>^{10}\,</sup>$  for example, Fedora 40  $\,$ 

<sup>11</sup> https://github.com/zlib-ng/zlib-ng

<sup>&</sup>lt;sup>12</sup> for example to specify static linking with a build which has both shared and static libraries.

<sup>&</sup>lt;sup>13</sup> Such as GNU tar 1.15 or later, bsdtar (from https://github.com/libarchive/libarchive/, used as tar by FreeBSD and macOS 10.6 and later) or tar from the Heirloom Toolchest (https://heirloom.sourceforge. net/tools.html), although the latter does not support xz compression.

<sup>&</sup>lt;sup>14</sup> texi2dvi is normally a shell script. Some of the issues which have been observed with broken versions of texi2dvi can be circumvented by setting the environment variable R\_TEXI2DVICMD to the value emulation.

hyperref (currently) requires packages etoolbox, kvoptions, ltxcmds and refcount, and inconsolata requires xkeyval. Building the base vignettes requires hyperref, fancyvrb, natbib, parskip and listings. For distributions based on TeX Live the simplest approach may be to install collections collection-latex, collection-fontsrecommended, collection-latexrecommended, collectionfontsextra and collection-latexextra (assuming they are not installed by default): Fedora uses names like texlive-collection-fontsextra and Debian/Ubuntu like texlive-fonts-extra.

Programs qpdf and Ghostscript (gs) are desirable as these will be used to compact the installed PDF vignettes and any PDF manuals.

The essential programs should be in your PATH at the time configure is run: this will capture the full paths.

For date-times to work correctly it is essential that the tables defining time zones are installed: these are usually in an OS component named something like tzdata. On most OSes they are required but installations of Alpine Linux have been seen without them. There is a configure check that recent date-times to work correctly in different time zones which catches this when installing from source (but not for binary distributions).

Those distributing binary versions of R may need to be aware of the licences of the external libraries it is linked to (including 'useful' libraries from the next section). The liblzma library is in the public domain and X11, libbzip2, libcurl and zlib have MIT-style licences. PCRE and PCRE2 have a BSD-style licence which requires distribution of the licence (included in R's COPYRIGHTS file) in binary distributions. GNU readline is licensed under GPL (which version(s) of GPL depends on the readline version).

#### A.2 Useful libraries and programs

The ability to use translated messages makes use of gettext and most likely needs GNU gettext: you do need this to work with new translations, but otherwise the version of the gettext runtime contained in the R sources will be used if no suitable external gettext is found.

The 'modern' version of the X11(), jpeg(), png() and tiff() graphics devices uses the Cairo and Pango libraries. Cairo version 1.2.0 or later and Pango version 1.10 or later are required (but much later versions are current). R checks for pkg-config, and uses that to check first that the 'pangocairo' package is installed (and if not, 'cairo') then if suitable code can be compiled. These tests will fail if pkg-config is not installed<sup>15</sup>, and might fail if cairo was built statically unless configure option --with-static-cairo is used. Most systems with Gtk+ 2.8 or later installed will have suitable libraries: for Fedora users the pango-devel RPM and its dependencies suffice. It is possible (but very unusual on a platform with X11) to build Cairo without its cairo-xlib module in which case X11(type = "cairo") will not be available. Pango is optional but highly desirable as it is likely to give much better text rendering, including kerning.

For the best font experience with these devices you need suitable fonts installed: Linux users will want the urw-fonts package. On platforms which have it available, the msttcorefonts package<sup>16</sup> provides TrueType versions of Monotype fonts such as Arial and Times New Roman. Another useful set of fonts is the 'liberation' TrueType fonts available at https://pagure.io/liberation-fonts,<sup>17</sup> which cover the Latin, Greek and Cyrillic alphabets plus a fair range of signs. These share metrics with Arial, Times New Roman and Courier New, and contain fonts rather similar to the first two (https://en.wikipedia.org/wiki/Liberation\_fonts). Then

<sup>&</sup>lt;sup>15</sup> If necessary the path to pkg-config can be specified by setting PKG\_CONFIG in config.site, on the configure command line or in the environment. There is a compatible re-implementation of pkg-config called pkgconf which can be used in the unlikely event that is installed but not linked to pkg-config.

<sup>&</sup>lt;sup>16</sup> also known as ttf-mscorefonts-installer in the Debian/Ubuntu world: see also https://en.wikipedia. org/wiki/Core\_fonts\_for\_the\_Web.

<sup>&</sup>lt;sup>17</sup> ttf-liberation in Debian/Ubuntu.

there is the 'Free UCS Outline Fonts' project (https://www.gnu.org/software/freefont/) which are OpenType/TrueType fonts based on the URW fonts but with extended Unicode coverage. See the R help on X11 on selecting such fonts.

The bitmapped graphics devices jpeg(), png() and tiff() need the appropriate headers and libraries installed: jpeg (version 6b or later, or libjpeg-turbo) or libpng (version 1.2.7 or later) and zlib or libtiff respectively. pkg-config is used if available and so needs the appropriate .pc file (which requires libtiff version 4.x and is not available on all platforms for jpeg before version 9c). They also need support for either X11 or cairo (see above). Should support for these devices **not** be required or broken system libraries need to be avoided there are configure options --without-libpng, --without-jpeglib and --without-libtiff. The TIFF library has many optional features such as jpeg, libz, zstd, lzma, webp, jbig and jpeg12, none of which is required for the tiff() devices but may need to be present to link the library (usually only an issue for static linking). pkg-config can tell you what other libraries are required for linking, for example by pkg-config libtiff-4 --static --libs.

Option --with-system-tre is also available: it needs a recent version of TRE. (The latest sources are in the git repository at https://github.com/laurikari/tre/, but at the time of writing the resulting build did not complete its checks, nor did R built against the version supplied by Fedora.)

An implementation of XDR is required, and the R sources contain one which is likely to suffice (although a system version may have higher performance). XDR is part of RPC and historically has been part of libc on a Unix-alike. (In principle man xdr\_string should tell you which library is needed, but it often does not: on some OSes it is provided by libnsl.) However some builds<sup>18</sup> of glibc omit or hide it with the intention that the TI-RPC library be used, in which case libtirpc (and its development version) should be installed, and its headers<sup>19</sup> need to be on the C include path or under /usr/include/tirpc.

Library libdeflate (https://github.com/ebiggers/libdeflate) is used by memCompress() and memDecompress() if available.

Use of the X11 clipboard selection requires the Xmu headers and libraries. These are normally part of an X11 installation (e.g. the Debian meta-package 'xorg-dev'), but some distributions have split this into smaller parts, so for example recent versions of Fedora require the 'libXmu' and 'libXmu-devel' RPMs.

Some systems (notably macOS and at least some FreeBSD systems) have inadequate support for collation in multibyte locales. It is possible to replace the OS's collation support by that from ICU (International Components for Unicode, https://icu.unicode.org/), and this provides much more precise control over collation on all systems. ICU is available as sources and as binary distributions for (at least) most Linux distributions, FreeBSD, macOS and AIX, usually as libicu or icu4c. It will be used by default where available: should a very old or broken version of ICU be found this can be suppressed by --without-ICU.

The bitmap and dev2bitmap devices and function embedFonts() use Ghostscript (https://www.ghostscript.com/). This should either be in your path when the command is run, or its full path specified by the environment variable R\_GSCMD at that time.

At the time of writing a full installation on Fedora Linux used the following packages and their development versions, and this may provide a useful checklist for other systems:

bzip2 cairo fontconfig freetype fribidi gcc gcc-gfortran gcc-c++ glib2 glibc harfbuzz lapack libX11 libXext libXt libcurl libdeflate libicu libjpeg libpng libtiff libtirpc libxcrypt ncurses pango pkgconf-pkg-config pcre2 readline tcl tk xz zlib

 $<sup>^{18}</sup>$  Including that used by Fedora 28 and later

<sup>&</sup>lt;sup>19</sup> R uses rpc/xdr.h but that includes netconfig.h from the top tirpc directory.

plus, preferably a TeX installation and Java.

#### A.2.1 Tcl/Tk

--with-tcltk

The tcltk package needs  $Tcl/Tk \ge 8.4$  installed: the sources are available at https://www.tcl.tk/. To specify the locations of the Tcl/Tk files you may need the configuration options

```
use Tcl/Tk, or specify its library directory
--with-tcl-config=TCL_CONFIG
specify location of tclConfig.sh
```

--with-tk-config=*TK\_CONFIG* specify location of tkConfig.sh

or use the configure variables TCLTK\_LIBS and TCLTK\_CPPFLAGS to specify the flags needed for linking against the Tcl and Tk libraries and for finding the tcl.h and tk.h headers, respectively. If you have both 32- and 64-bit versions of Tcl/Tk installed, specifying the paths to the correct config files may be necessary to avoid confusion between them.

Versions of Tcl/Tk up to 8.5.19 and 8.6.12 have been tested (including most versions of 8.4.x, but not recently).

Note that the tk.h header includes<sup>20</sup> X11 headers, so you will need X11 and its development files installed.

#### A.2.2 Java support

The build process looks for Java support on the host system, and if it finds it sets some settings which are useful for Java-using packages (such as rJava (https://CRAN.R-project.org/ package=rJava) and JavaGD (https://CRAN.R-project.org/package=JavaGD): installing these from source requires a full JDK). This check can be suppressed by configure option --disable-java. Configure variable JAVA\_HOME can be set to point to a specific JRE/JDK, on the configure command line or in the environment.

Principal amongst these settings are some paths to the Java libraries and JVM, which are stored in environment variable  $R_JAVA_LD_LIBRARY_PATH$  in file  $R_HOME/etc/ldpaths$  (or a sub-architecture-specific version). A typical setting for 'x86\_64' Linux is

JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.322.b06-6.fc34.x86\_64/jre R\_JAVA\_LD\_LIBRARY\_PATH=\${JAVA\_HOME}/lib/amd64/server

Unfortunately this depends on the exact version of the JRE/JDK installed, and so may need updating if the Java installation is updated. This can be done by running R CMD javareconf which updates settings in both  $R_HOME/etc/Makeconf$  and  $R_HOME/etc/ldpaths$ . See R CMD javareconf --help for details: note that this needs to be done by the account owning the R installation.

Another way of overriding those settings is to set the environment variable R\_JAVA\_LD\_ LIBRARY\_PATH (before R is started, hence not in ~/.Renviron), which suffices to run alreadyinstalled Java-using packages. For example

R\_JAVA\_LD\_LIBRARY\_PATH=/usr/lib/jvm/java-1.8.0/jre/lib/amd64/server

It may be possible to avoid this by specifying an invariant link as the path when configuring. For example, on that system any of

JAVA\_HOME=/usr/lib/jvm/java JAVA\_HOME=/usr/lib/jvm/java-1.8.0

 $<sup>^{20}\,</sup>$  This is true even for the 'Aqua' version of Tk on macOS, but distributions of that include a copy of the X11 files needed.

JAVA\_HOME=/usr/lib/jvm/java-1.8.0/jre
JAVA\_HOME=/usr/lib/jvm/jre-1.8.0

worked (since the 'auto' setting of /etc/alternatives chose Java 8 aka 1.8.0).

'Non-server' Oracle distributions of Java as from version 11 are of a full JDK. However, Linux distributions can be confusing: for example Fedora 38 had

```
java-1.8.0-openjdk
java-1.8.0-openjdk-devel
java-11-openjdk
java-11-openjdk-devel
java-17-openjdk-devel
java-17-openjdk-devel
java-latest-openjdk
```

where the -devel RPMs are needed to complete the JDK. Debian/Ubuntu use '-jre' and '-jdk', e.g.

sudo apt install default-jdk

#### A.2.3 Other compiled languages

Some add-on packages need a C++ compiler. This is specified by the configure variables CXX, CXXFLAGS and similar. configure will normally find a suitable compiler. It is possible to specify an alternative C++17 compiler by the configure variables CXX17, CXX17STD, CXX17FLAGS and similar (see Section 2.7.3 [C++ Support], page 11). Again, configure will normally find a suitable value for CXX17STD if the compiler given by CXX is capable of compiling C++17 code, but it is possible that a completely different compiler will be needed. (Similar macros are provided for C++20.)

For source files with extension .f90 or .f95 containing free-form Fortran, the compiler defined by the macro FC is used by R CMD INSTALL. Note that it is detected by the name of the command without a test that it can actually compile Fortran 90 code. Set the configure variable FC to override this if necessary: variables FCFLAGS and FCLIBS\_XTRA might also need to be set.

See file config.site in the R source for more details about these variables.

### A.3 Linear algebra

The linear algebra routines in R make use of BLAS (Basic Linear Algebra Subprograms, https://netlib.org/blas/faq.html) routines, and most make use of routines from LAPACK (Linear Algebra PACKage, https://netlib.org/lapack/). The R sources contain reference (Fortran) implementations of these, but they can be replaced by external libraries, usually those tuned for speed on specific CPUs. These libraries normally contain all of the BLAS routines and some tuned LAPACK routines and perhaps the rest of LAPACK from the reference implementation. Because of the way linking works, using an external BLAS library may necessitate using the version of LAPACK it contains.

Note that the alternative implementations will not give identical numeric results. Some differences may be benign (such the signs of SVDs and eigenvectors), but the optimized routines can be less accurate and (particularly for LAPACK) can be from older versions with fewer corrections. However, R relies on ISO/IEC 60559 compliance. This can be broken if for example the code assumes that terms with a zero factor are always zero and do not need to be computed—whereas x\*0 can be NaN. The internal BLAS has been extensively patched to avoid this whereas MKL's documentation has warned

LAPACK routines assume that input matrices do not contain IEEE 754 special values such as INF or NaN values. Using these special values may cause LAPACK to return unexpected results or become unstable.

Some of the external libraries are multi-threaded. One issue is that R profiling (which uses the SIGPROF signal) may cause problems, and you may want to disable profiling if you use a multi-threaded BLAS. Note that using a multi-threaded BLAS can result in taking more CPU time and even more elapsed time (occasionally dramatically so) than using a similar singlethreaded BLAS. On a machine running other tasks, there can be contention for CPU caches that reduces the effectiveness of the optimization of cache use by a BLAS implementation: some people warn that this is especially problematic for hyper-threaded CPUs.

BLAS and LAPACK routines may be used inside threaded code, for example in OpenMP sections in packages such as mgcv (https://CRAN.R-project.org/package=mgcv). The reference implementations are thread-safe but external ones may not be (even single-threaded ones): this can lead to hard-to-track-down incorrect results or segfaults.

There is a tendency for re-distributors of R to use 'enhanced' linear algebra libraries without explaining their downsides.

#### A.3.1 BLAS

An external BLAS library has to be explicitly requested at configure time.

You can specify a particular BLAS library *via* a value for the configuration option --withblas. If this is given with no =, its value is taken from the environment variable BLAS\_LIBS, set for example in config.site. If neither the option nor the environment variable supply a value, a search is made for a suitable<sup>21</sup> BLAS. If the value is not obviously a linker command (starting with a dash or giving the path to a library), it is prefixed by '-1', so

```
--with-blas="foo"
```

is an instruction to link against '-lfoo' to find an external BLAS (which needs to be found both at link time and run time).

The configure code checks that the external BLAS is complete (as of LAPACK 3.9.1: it must include all double precision and double complex routines, as well as LSAME), and appears to be usable. However, an external BLAS has to be usable from a shared object (so must contain position-independent code), and that is not checked. Also, the BLAS can be switched after configure is run, either as a symbolic link or by the mechanisms mentioned below, and this can defeat the completeness check.

Some enhanced BLASes are compiler-system-specific (Accelerate on macOS, sunperf on Solaris<sup>22</sup>, libessl on IBM). The correct incantation for these is often found *via* --with-blas with no value on the appropriate platforms.

Note that under Unix (but not under Windows) if R is compiled against a non-default BLAS and --enable-BLAS-shlib is not used (it is the default on all platforms except AIX), then all BLAS-using packages must also be. So if R is re-built to use an enhanced BLAS then packages such as quantreg (https://CRAN.R-project.org/package=quantreg) will need to be re-installed.

Debian/Ubuntu systems provide a system-specific way to switch the BLAS in use: Build R with --with-blas to select the OS version of the reference BLAS, and then use updatealternatives to switch between the available BLAS libraries. See https://wiki.debian. org/DebianScience/LinearAlgebraLibraries.

Fedora 33 and later offer 'FlexiBLAS', a similar mechanism for switching the BLAS in use (https://www.mpi-magdeburg.mpg.de/projects/flexiblas). However, rather than overriding libblas, this requires configuring R with option --with-blas=flexiblas. 'Backend' wrappers are available for the reference BLAS, ATLAS and serial, threaded and OpenMP builds of

<sup>&</sup>lt;sup>21</sup> The search order is currently OpenBLAS, BLIS, ATLAS, platform-specific choices (see below) and finally a generic libblas.

 $<sup>^{22}</sup>$  Using the Oracle Developer Studio cc and f95 compilers

OpenBLAS and BLIS, and perhaps others<sup>23</sup>. This can be controlled from a running R session by package flexiblas (https://CRAN.R-project.org/package=flexiblas).

BLAS implementations which use parallel computations can be non-deterministic: this is known for ATLAS.

## A.3.1.1 ATLAS

ATLAS (https://math-atlas.sourceforge.net/) is a "tuned" BLAS that runs on a wide range of Unix-alike platforms. Unfortunately it is built by default as a static library that on some platforms may not be able to be used with shared objects such as are used in R packages. Be careful when using pre-built versions of ATLAS static libraries (they seem to work on 'ix86' platforms, but not always on 'x86\_64' ones).

ATLAS contains replacements for a small number of LAPACK routines, but can be built to merge these with the reference LAPACK sources to include a full LAPACK library.

Recent versions of ATLAS can be built as a single shared library, either libsatlas or libtatlas (serial or threaded respectively): these may even contain a full LAPACK. Such builds can be used by one of

```
--with-blas=satlas
--with-blas=tatlas
```

or, as on 'x86\_64' Fedora where a path needs to be specified,

```
--with-blas="-L/usr/lib64/atlas -lsatlas"
--with-blas="-L/usr/lib64/atlas -ltatlas"
```

Distributed ATLAS libraries cannot be tuned to your machine and so are a compromise: for example Fedora tunes<sup>24</sup> ' $x86_64$ ' RPMs for CPUs with SSE3 extensions, and separate RPMs may be available for specific CPU families.

Note that building R on Linux against distributed shared libraries may need '-devel' or '-dev' packages installed.

Linking against multiple static libraries requires one of

```
--with-blas="-lf77blas -latlas"
--with-blas="-lptf77blas -lpthread -latlas"
--with-blas="-L/path/to/ATLAS/libs -lf77blas -latlas"
--with-blas="-L/path/to/ATLAS/libs -lptf77blas -lpthread -latlas"
```

Consult its installation guide<sup>25</sup> for how to build ATLAS as a shared library or as a static library with position-independent code (on platforms where that matters).

According to the ATLAS FAQ<sup>26</sup> the maximum number of threads used by multi-threaded ATLAS is set at compile time. Also, the author advises against using multi-threaded ATLAS on hyper-threaded CPUs without restricting affinities at compile-time to one virtual core per physical CPU. (For the Fedora libraries the compile-time flag specifies 4 threads.)

#### A.3.1.2 OpenBLAS and BLIS

Dr Kazushige Goto wrote a tuned BLAS for several processors and OSes, which was frozen in 2010. OpenBLAS (https://www.openblas.net/) is a descendant project with support for some later CPUs.

This can be used by configuring R with something like

--with-blas="openblas"

 $<sup>^{23}\,</sup>$  for example, Intel MKL not packaged by Fedora.

<sup>&</sup>lt;sup>24</sup> The only way to see exactly which CPUs the distributed libraries have been tuned for is to read the atlas.spec file.

<sup>&</sup>lt;sup>25</sup> https://math-atlas.sourceforge.net/atlas\_install/

<sup>&</sup>lt;sup>26</sup> https://math-atlas.sourceforge.net/faq.html#tnum

See see Section A.3.1.4 [Shared BLAS], page 48, for an alternative (and in many ways preferable) way to use them.

Some platforms provide multiple builds of OpenBLAS: for example Fedora has RPMs<sup>27</sup>

```
openblas
openblas-threads
openblas-openmp
```

providing shared libraries

libopenblas.so libopenblasp.so libopenblaso.so

respectively, each of which can be used as a shared BLAS. For the second and third the number of threads is controlled by OPENBLAS\_NUM\_THREADS and OMP\_NUM\_THREADS (as usual for OpenMP) respectively.

These and their Debian equivalents contain a complete LAPACK implementation.

Note that building R on Linux against distributed libraries may need '-devel' or '-dev' packages installed.

For 'ix86' and 'x86\_64' CPUs most distributed libraries contain several alternatives for different CPU microarchitectures with the choice being made at run time.

Another descendant project is BLIS (https://github.com/flame/blis). This has (in Fedora) shared libraries

libblis.so libblisp.so libbliso.so

(p for 'threads', o for OpenMP as for OpenBLAS) which can also be used as a shared BLAS. The Fedora builds do not include LAPACK in the BLIS libraries.

### A.3.1.3 Intel MKL

For Intel processors (and perhaps others) and some distributions of Linux, there is Intel's Math Kernel Library<sup>28</sup>. You are encouraged to read the documentation which is installed with the library before attempting to link to MKL. This includes a 'link line advisor' which will suggest appropriate incantations: its use is recommended. Or see https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-link-line-advisor.html#gs.vpt6qp (which at the time of writing selected the Intel library for linking with GCC).

There are also versions of MKL for  $macOS^{29}$  and Windows, but when these have been tried they did not work with the default compilers used for R on those platforms.

The following examples have been used with MKL versions 10.3 to 2023.2.0, for GCC compilers on 'x86\_64' CPUs. (See also Section C.2.3 [Intel compilers], page 60.)

To use a sequential version of MKL we used

MKL\_LIB\_PATH=/path/to/intel\_mkl/mkl/lib/intel64
export LD\_LIBRARY\_PATH=\$MKL\_LIB\_PATH
MKL="-L\${MKL\_LIB\_PATH} -lmkl\_gf\_lp64 -lmkl\_core -lmkl\_sequential"
./configure --with-blas="\$MKL" --with-lapack

The option --with-lapack is used since MKL contains a tuned copy of LAPACK (often older than the current version) as well as the BLAS (see Section A.3.2 [LAPACK], page 49), although this can be omitted.

 $<sup>^{27}\,</sup>$  (and more, e.g. for 64-bit ints and static versions).

 $<sup>^{28}\,</sup>$  Nowadays known as 'Intel one API Math Kernel Library' or even 'one MKL'.

<sup>&</sup>lt;sup>29</sup> The issue for macOS has been the use of double-complex routines.

Threaded MKL may be used by replacing the line defining the variable MKL by

```
MKL="-L${MKL_LIB_PATH} -lmkl_gf_lp64 -lmkl_core \
-lmkl_gnu_thread -dl -fopenmp"
```

R can also be linked against a single shared library, libmkl\_rt.so, for both BLAS and LA-PACK, but the correct OpenMP and MKL interface layer then has to be selected via environment variables. With 64-bit builds and the GCC compilers, we used

```
export MKL_INTERFACE_LAYER=GNU,LP64
export MKL_THREADING_LAYER=GNU
```

On Debian/Ubuntu, MKL is provided by package intel-mkl-full and one can set libmkl\_ rt.so as the system-wide implementation of both BLAS and LAPACK during installation of the package, so that also R installed from Debian/Ubuntu package r-base would use it. It is, however, still essential to set MKL\_INTERFACE\_LAYER and MKL\_THREADING\_LAYER before running R, otherwise MKL computations will produce incorrect results. R does not have to be rebuilt to use MKL, but configure includes tests which may discover some errors such as a failure to set the correct OpenMP and MKL interface layer.

Note that the Debian/Ubuntu distribution can be quite old (for example 2020.4 in mid-2023 when 2023.1 was current): this can be important for the LAPACK version included.

The default number of threads will be chosen by the OpenMP software, but can be controlled by setting OMP\_NUM\_THREADS or MKL\_NUM\_THREADS, and in recent versions seems to default to a sensible value for sole use of the machine. (Parallel MKL has not always passed make check-all, but did with MKL 2019.4 and later.)

MKL includes a partial implementation of FFTW3, which causes trouble for applications that require some of the FFTW3 functionality unsupported in MKL. Please see the MKL manuals for description of these limitations and for instructions on how to create a custom version of MKL which excludes the FFTW3 wrappers.

There is Intel documentation for building R with MKL at https://www.intel.com/ content/www/us/en/developer/articles/technical/using-onemkl-with-r.html: that includes

-Wl, -- no-as-needed

which we have not found necessary.

#### A.3.1.4 Shared BLAS

The BLAS library will be used for many of the add-on packages as well as for R itself. This means that it is better to use a shared/dynamic BLAS library, as most of a static library will be compiled into the R executable and each BLAS-using package.

R offers the option of compiling the BLAS into a dynamic library libRblas stored in  $R_{-}$  HOME/lib and linking both R itself and all the add-on packages against that library.

This is the default on all platforms except AIX unless an external BLAS is specified and found: for the latter it can be used by specifying the option --enable-BLAS-shlib, and it can always be disabled via --disable-BLAS-shlib.

This has both advantages and disadvantages.

- It saves space by having only a single copy of the BLAS routines, which is helpful if there is an external static BLAS (as used to be standard for ATLAS).
- There may be performance disadvantages in using a shared BLAS. Probably the most likely is when R's internal BLAS is used and R is *not* built as a shared library, when it is possible to build the BLAS into R.bin (and libR.a) without using position-independent code. However, experiments showed that in many cases using a shared BLAS was as fast, provided high levels of compiler optimization are used.

• It is easy to change the BLAS without needing to re-install R and all the add-on packages, since all references to the BLAS go through libRblas, and that can be replaced. Note though that any dynamic libraries the replacement links to will need to be found by the linker: this may need the library path to be changed in  $R_HOME/etc/ldpaths$ .

Another option to change the BLAS in use is to symlink a single dynamic BLAS library to  $R_HOME/lib/libRblas.so$ . For example, just

```
mv R_HOME/lib/libRblas.so R_HOME/lib/libRblas.so.keep
ln -s /usr/lib64/libopenblasp.so.0 R_HOME/lib/libRblas.so
```

on 'x86\_64' Fedora will change the BLAS used to multithreaded OpenBLAS. A similar link works for most versions of the OpenBLAS (provided the appropriate lib directory is in the run-time library path or ld.so cache). It can also be used for a single-library ATLAS, so on 'x86\_64' Fedora either of

```
ln -s /usr/lib64/atlas/libsatlas.so.3 R_HOME/lib/libRblas.so
ln -s /usr/lib64/atlas/libtatlas.so.3 R_HOME/lib/libRblas.so
```

can be used with its distributed ATLAS libraries. (If you have the '-devel' RPMs installed you can omit the .0/.3.)

Note that rebuilding or symlinking libRblas.so may not suffice if the intention is to use a modified LAPACK contained in an external BLAS: the latter could even cause conflicts. However, on Fedora where the OpenBLAS distribution contains a copy of LAPACK, it is the latter which is used.

### A.3.2 LAPACK

If when configuring R a system LAPACK library is found of version 3.9.0 or later (and does not contain BLAS routines) it will be used instead of compiling the LAPACK code in the package sources. This can be prevented by configuring R with --without-lapack. Using a static liblapack.a is not supported.

It is assumed that -llapack is the reference LAPACK library but on Debian/Ubuntu it can be switched, including after R is installed. On such a platform it is better to use --withoutlapack or --with-blas --with-lapack (see below) explicitly. The known examples<sup>30</sup> of a non-reference LAPACK library found at installation all contain BLAS routines so are not used by a default configure run.

Provision is made for specifying an external LAPACK library with option --with-lapack, principally to cope with BLAS libraries which contain a copy of LAPACK (such as Accelerate on macOS and some builds of ATLAS, FlexiBLAS, MKL and OpenBLAS on 'ix86'/'x86\_64' Linux). At least LAPACK version 3.2 is required. This can only be done if --with-blas has been used.

However, the likely performance gains are thought to be small (and may be negative). The default is not to search for a suitable LAPACK library, and this is definitely **not** recommended. You can specify a specific LAPACK library or a search for a generic library by the configuration option --with-lapack without a value. The default for --with-lapack is to check the BLAS library (for function DPSTRF) and then look for an external library '-llapack'. Sites searching for the fastest possible linear algebra may want to build a LAPACK library using the ATLAS-optimized subset of LAPACK. Similarly, OpenBLAS can be built to contain an optimized subset of LAPACK (the latter seeming to be the default).

A value for --with-lapack can be set *via* the environment variable LAPACK\_LIBS, but this will only be used if --with-lapack is specified and the BLAS library does not contain LAPACK.

 $<sup>^{30}\,</sup>$  ATLAS, OpenBLAS and Accelerate.

Please bear in mind that using --with-lapack is provided **only** because it is necessary on some platforms and because some users want to experiment with claimed performance improvements. In practice its main uses are without a value,

- with an 'enhanced' BLAS such as ATLAS, FlexiBLAS, MKL or OpenBLAS which contains a full LAPACK (to avoid possible conflicts), or
- on Debian/Ubuntu systems to select the system liblapack which can be switched by the 'alternatives' mechanism.

If building LAPACK from its Netlib sources, be aware that make with its supplied Makefile will make a *static* library and R requires a shared/dynamic one. To get one, use cmake as documented briefly in README.md. Something like (to build only the double and double complex subroutines with 32-bit array indices),

```
mkdir build
cd build
cmake \
-DCMAKE_INSTALL_PREFIX=/where/you/want/to/install \
-DCMAKE_BUILD_TYPE:STRING=Release \
-DBUILD_DEPRECATED=ON -DBUILD_SHARED_LIBS=ON \
-DBUILD_INDEX64_EXT_API:BOOL=OFF \
-DBUILD_SINGLE:BOOL=OFF -DBUILD_COMPLEX:BOOL=OFF \
-DLAPACKE=OFF -DCBLAS=OFF \
-S ..
make -j10
```

This builds the reference BLAS and the reference LAPACK linked to it.

Note that **cmake** files do not provide an **uninstall** target, but **build/install\_manifest.txt** is a list of the files installed, so you can remove them *via* shell commands or from R.

If using --with-lapack to get a generic LAPACK (or allowing the default to select one), consider also using --with-blas (with a path if an enhanced BLAS is installed).

#### A.3.3 Caveats

As with all libraries, you need to ensure that they and R were compiled with compatible compilers and flags. For example, this has meant that on Sun Sparc using the Oracle compilers the flag -dalign is needed if sunperf is to be used.

On some systems it has been necessary that an external BLAS/LAPACK was built with the same Fortran compiler used to build R.

BLAS and LAPACK libraries built with recent versions of gfortran require calls from C/C++ to handle 'hidden' character lengths — R itself does so but many packages used not to and some have segfaulted. This was largely circumvented by using the Fortran flag -fno-optimize-sibling-calls (formerly set by configure if it detected gfortran 7 or later): however use of the R headers which include those character-length arguments is no longer optional in packages.

LAPACK 3.9.0 (and probably earlier) had a bug in which the DCOMBSSQ subroutine may cause NA to be interpreted as zero. This is fixed in the R 3.6.3 and later sources, but if you use an external LAPACK, you may need to fix it there. (The bug was corrected in 3.9.1 and the routine removed in 3.10.1.)

The code (in dlapack.f) should read

```
* ..
* .. Executable Statements ..
*
IF( V1( 1 ).GE.V2( 1 ) ) THEN
```

```
IF( V1( 1 ).NE.ZERO ) THEN
V1( 2 ) = V1( 2 ) + ( V2( 1 ) / V1( 1 ) )**2 * V2( 2 )
ELSE
V1( 2 ) = V1( 2 ) + V2( 2 )
END IF
ELSE
V1( 2 ) = V2( 2 ) + ( V1( 1 ) / V2( 1 ) )**2 * V1( 2 )
V1( 1 ) = V2( 1 )
END IF
RETURN
```

(The inner ELSE clause was missing in LAPACK 3.9.0.)

If you do use an external LAPACK, be aware of potential problems with other bugs in the LAPACK sources (or in the posted corrections to those sources), seen several times in Linux distributions over the years. We have even seen distributions with missing LAPACK routines from their liblapack.

We rely on limited support in LAPACK for matrices with  $2^{31}$  or more elements: it is possible that an external LAPACK will not have that support.

## Appendix B Configuration on a Unix-alike

### **B.1** Configuration options

configure has many options: running

#### ./configure --help

will give a list. Probably the most important ones not covered elsewhere are (defaults in brackets)

--with-x use the X Window System [yes]

```
--x-includes=DIR
X include files are in DIR
```

--x-libraries=DIR X library files are in DIR

```
--with-readline
use readline library (if available) [yes]
```

--enable-R-profiling attempt to compile support for Rprof() [yes]

```
--enable-memory-profiling
```

attempt to compile support for Rprofmem() and tracemem() [no]

--enable-R-shlib

build R as a shared/dynamic library [no]

#### --enable-BLAS-shlib

build the BLAS as a shared/dynamic library [yes, except on AIX]

You can use --without-foo or --disable-foo for the negatives.

You will want to use --disable-R-profiling if you are building a profiled executable of R (e.g. with '-pg)'. Support for R profiling requires OS support for POSIX threads (*aka* 'pthreads'), which are available on all mainstream Unix-alike platforms.

Flag --enable-R-shlib causes the make process to build R as a dynamic (shared) library, typically called libR.so, and link the main R executable R.bin against that library. This can only be done if all the code (including system libraries) can be compiled into a dynamic library, and there may be a performance<sup>1</sup> penalty. So you probably only want this if you will be using an application which embeds R. Note that C code in packages installed on an R system linked with --enable-R-shlib is linked against the dynamic library and so such packages cannot be used from an R system built in the default way. Also, because packages are linked against R they are on some OSes also linked against the dynamic libraries R itself is linked against, and this can lead to symbol conflicts.

For maximally effective use of valgrind, R should be compiled with Valgrind instrumentation. The configure option is --with-valgrind-instrumentation=level, where level is 0, 1 or 2. (Level 0 is the default and does not add anything.) The system headers for valgrind are required: on Linux they may be in a separate package such as valgrind-devel.

If you need to re-configure R with different options you may need to run make clean or even make distclean before doing so.

The configure script has other generic options added by autoconf and which are not supported for R: in particular building for one architecture on a different host is not possible.

 $<sup>^1</sup>$  We have measured 15–20% on 'i686' Linux and around 10% on 'x86\_64' Linux.

### **B.2** Internationalization support

Translation of messages is supported via GNU gettext unless disabled by the configure option --disable-nls. The configure report will show NLS as one of the 'Additional capabilities' if support has been compiled in, and running in an English locale (but not the C locale) will include

Natural language support but running in an English locale

in the greeting on starting R.

#### **B.3** Configuration variables

If you need or want to set certain configure variables to something other than their default, you can do that by either editing the file config.site (which documents many of the variables you might want to set: others can be seen in file etc/Renviron.in) or on the command line as

#### ./configure VAR=value

If you are building in a directory different from the sources, there can be copies of config.site in the source and the build directories, and both will be read (in that order). In addition, if there is a file ~/.R/config, it is read between the config.site files in the source and the build directories.

There is also a general autoconf mechanism for config.site files, which are read before any of those mentioned in the previous paragraph. This looks first at a file specified by the environment variable CONFIG\_SITE, and if not is set at files such as /usr/local/share/config.site and /usr/local/etc/config.site in the area (exemplified by /usr/local) where R would be installed.

These variables are *precious*, implying that they do not have to be exported to the environment, are kept in the cache even if not specified on the command line, checked for consistency between two configure runs (provided that caching is used), and are kept during automatic reconfiguration as if having been passed as command line arguments, even if no cache is used.

See the variable output section of configure --help for a list of all these variables.

If you find you need to alter configure variables, it is worth noting that some settings may be cached in the file config.cache, and it is a good idea to remove that file (if it exists) before re-configuring. Note that caching is turned *off* by default: use the command line option --config-cache (or -C) to enable caching.

#### **B.3.1** Setting paper size

One common variable to change is R\_PAPERSIZE, which defaults to 'a4', not 'letter'. (Valid values are 'a4', 'letter', 'legal' and 'executive'.)

This is used both when configuring R to set the default, and when running R to override the default. It is also used to set the paper size when making PDF manuals.

The configure default will most often be 'a4' if R\_PAPERSIZE is unset. (If the program paperconf is found, present in many Linux distributions, or the environment variable PAPERSIZE is set, these are used to produce the default.)

#### **B.3.2** Setting the browsers

Another precious variable is **R\_BROWSER**, the default HTML browser, which should take a value of an executable in the user's path or specify a full path.

Its counterpart for PDF files is R\_PDFVIEWER.

#### **B.3.3** Compilation flags

If you have libraries and header files, e.g., for GNU readline, in non-system directories, use the variables LDFLAGS (for libraries, using '-L' flags to be passed to the linker) and CPPFLAGS (for header files, using '-I' flags to be passed to the C/C++ preprocessors), respectively, to specify these locations. These default to '-L/usr/local/lib' (LDFLAGS, '-L/usr/local/lib64' on most 64-bit Linux OSes) and '-I/usr/local/include' (CPPFLAGS, but note that on most systems /usr/local/include is regarded as a system include directory and so instances in that macro will be skipped) to catch the most common cases. If libraries are still not found, then maybe your compiler/linker does not support re-ordering of -L and -l flags. In this case, use a different compiler (or a front-end shell script which does the re-ordering).

These flags can also be used to build a faster-running version of R. On most platforms using gcc, having '-O3' in CFLAGS and FFLAGS produces worthwhile performance gains with gcc and gfortran, but may result in a less reliable build (both segfaults and incorrect numeric computations have been seen). On systems using the GNU linker (especially those using R as a shared library), it is likely that including '-W1,-O1' in LDFLAGS is worthwhile, and ''-Bdirect,--hash-style=both,-W1,-O1'' is recommended at https://lwn.net/Articles/192624/. Tuning compilation to a specific CPU family (e.g. '-mtune=native' for gcc) can give worthwhile performance gains, especially on older architectures such as 'ix86'.

#### **B.3.4** Making manuals

The default settings for making the manuals are controlled by R\_RD4PDF and R\_PAPERSIZE.

## **B.4** Setting the shell

By default the shell scripts such as R will be '#!/bin/sh' scripts (or using the SHELL chosen by configure). This is almost always satisfactory, but on a few systems /bin/sh is not a Bourne shell or clone, and the shell to be used can be changed by setting the configure variable R\_SHELL to a suitable value (a full path to a shell, e.g. /usr/local/bin/bash).

### B.5 Using make

To build in a separate directory you need a make that supports the VPATH variable, for example GNU make and dmake.

If you want to use a make by another name, for example if your GNU make is called 'gmake', you need to set the variable MAKE at configure time, for example

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

#### **B.6** Using Fortran

To compile R, you need a Fortran 90 compiler. The current default is to search for gfortran, g95, xlf95 f95, fort, ifort, ifc, efc, pgfortran, pgf95 lf95, ftn, nagfor, xlf90, f90, pgf90, pghpf, epcf90. (Note that these are searched for by name, without checking the standard of Fortran they support.) The command and flags used should support fixed-form Fortran with extension .f: in the unusual case that a specific flag is needed for free-form Fortran with extension .f90 or .f95, this can be specified as part of FCFLAGS.

The search mechanism can be changed using the configure variable FC which specifies the command that runs the Fortran compiler. If your Fortran compiler is in a non-standard location, you should set the environment variable PATH accordingly before running configure, or use the configure variable FC to specify its full path.

If your Fortran libraries are in slightly peculiar places, you should also look at LD\_LIBRARY\_ PATH (or your system's equivalent) to make sure that all libraries are on this path. Note that only Fortran compilers which convert identifiers to lower case are supported.

You must set whatever compilation flags (if any) are needed to ensure that Fortran integer is equivalent to a C int pointer and Fortran double precision is equivalent to a C double pointer. This is checked during the configuration process.

Some of the Fortran code makes use of DOUBLE COMPLEX and COMPLEX\*16 variables. This is checked for at configure time, as well as its equivalence to the Rcomplex C structure defined in  $R_ext/Complex.h$ .

gfortran 10 by default gives a compilation error for the previously widespread practice of passing a Fortran array element where an array is expected, or a scalar instead of a length-one array. See https://gcc.gnu.org/gcc-10/porting\_to.html. gfortran 12 errors in more cases of this.

## B.7 Compile and load flags

A wide range of flags can be set in the file **config.site** or as configure variables on the command line. We have already mentioned

CPPFLAGS	header file search directory $(-\mathtt{I})$ and any other miscellaneous options for the C and C++ preprocessors and compilers
LDFLAGS	path (-L), stripping (-s) and any other miscellaneous options for the linker
and others	include
CFLAGS	debugging and optimization flags, C
MAIN_CFLA	GS ditto, for compiling the main program (e.g. when profiling)
SHLIB_CFL	AGS for shared objects (no known examples)
FFLAGS	debugging and optimization flags, fixed-form Fortran
FCFLAGS	debugging and optimization flags, free-form Fortran
SAFE_FFLA	GS ditto for source files which need exact floating point behaviour
MAIN_FFLA	ditto, for compiling the main program (e.g. when profiling)
SHLIB_FFL	AGS for shared objects (no known examples)
MAIN_LDFL	AGS additional flags for the main link
SHLIB_LDFI	AGS additional flags for linking the shared objects
LIBnn	the primary library directory, lib or lib64
CPICFLAGS	special flags for compiling C code to be turned into a shared object
FPICFLAGS	special flags for compiling Fortran code to be turned into a shared object
CXXPICFLA	special flags for compiling C++ code to be turned into a shared object

#### DEFS defines to be used when compiling C code in R itself

Library paths specified as -L/lib/path in LDFLAGS are collected together and prepended to LD\_LIBRARY\_PATH (or your system's equivalent), so there should be no need for -R or -rpath flags.

Variables such as CPICFLAGS are determined where possible by configure. Some systems allows two types of PIC flags, for example '-fpic' and '-fPIC', and if they differ the first allows only a limited number of symbols in a shared object. Since R as a shared library has about 6200 symbols, if in doubt use the larger version.

Other variables often set by configure include 'MAIN\_LDFLAGS', 'SAFE\_FFLAGS', 'SHLIB\_LDFLAGS' and 'SHLIB\_CXXLDFLAGS': see file config.site in the sources for more documentation on these and others.

To compile a profiling version of R, one might for example want to use 'MAIN\_CFLAGS=-pg', 'MAIN\_FFLAGS=-pg', 'MAIN\_LDFLAGS=-pg' on platforms where '-pg' cannot be used with position-independent code.

Beware: it may be necessary to set CFLAGS and FFLAGS in ways compatible with the libraries to be used: one possible issue is the alignment of doubles, another is the way structures are passed.

On some platforms configure will select additional flags for CFLAGS, CPPFLAGS and LIBS in R\_XTRA\_CFLAGS (and so on). These are for options which are always required, for example to force IEC 60559 compliance.

### B.8 Maintainer mode

There are several files that are part of the R sources but can be re-generated from their own sources by configuring with option --enable-maintainer-mode and then running make in the build directory. This requires other tools to be installed, discussed in the rest of this section.

File configure is created from configure.ac and the files under m4 by autoconf and aclocal (part of the automake package). There is a formal version requirement on autoconf of 2.72 or later, but it is unlikely that anything other than the most recent versions<sup>2</sup> have been thoroughly tested.

File src/include/config.h is created by autoheader (part of autoconf).

Grammar files \*.y are converted to C sources by an implementation of yacc, usually bison -y: these are found in src/main and src/library/tools/src. It is known that earlier versions of bison generate code which reads (and in some cases writes) outside array bounds: bison 3.8.2 is currently used.

The ultimate sources for package **compiler** are in its **noweb** directory. To re-create the sources from **src/library/compiler/noweb/compiler.nw**, the command **notangle** is required. Some Linux distributions include this command in package **noweb**. It can also be installed from the sources at https://www.cs.tufts.edu/~nr/noweb/<sup>3</sup>. The package sources are only re-created even in maintainer mode if **src/library/compiler/noweb/compiler.nw** has been updated.

<sup>&</sup>lt;sup>2</sup> at the time of revision of this para in early 2024, **autoconf-2.72** and **automake-1.16.5**. Previously **autoconf-2.71** was used.

<sup>&</sup>lt;sup>3</sup> The links there have proved difficult to access, in which case grab the copy made available at https:// developer.r-project.org/noweb-2.11b.tgz.

## Appendix C Platform notes

This section provides some notes on building R on different Unix-alike platforms. These notes are based on tests run on one or two systems in each case with particular sets of compilers and support libraries. Success in building R depends on the proper installation and functioning of support software; your results may differ if you have other versions of compilers and support libraries.

Older versions of this manual contain notes on platforms such as HP-UX, IRIX, Alpha/OSF1 (for R < 2.10.0, and support has since been removed for all of these) and AIX (for R < = 3.5.x) for which we have had no recent reports.

C macros to select particular platforms can be tricky to track down (there is a fair amount of misinformation on the Web). The Wiki (currently) at https://sourceforge.net/p/predef/wiki/Home/ can be helpful. The R sources have used (often in included software under src/extra)

```
AIX: _AIX

Cygwin: __CYGWIN__

FreeBSD: __FreeBSD__

HP-UX: __hpux__, __hpux

IRIX: sgi, __sgi

Linux: __linux__

macOS: __APPLE__

NetBSD: __NetBSD__

OpenBSD: __OpenBSD__

Windows: _WIN32, _WIN64

Windows on 64-but ARM: _M_ARM64 or _WIN32 plus __aarch64__
```

Identifying compilers can be very tricky. GCC defines \_\_GNUC\_\_, but so do other compilers claiming conformance with it, notably (LLVM and Apple) clang and Intel compilers. Further, some use the value of \_\_GNUC\_\_ for their version, not the version of GCC they claim to be compatible with.<sup>1</sup> clang-based compilers define \_\_clang\_\_. Both LLVM and Apple clang define \_\_clang\_major\_\_ as a string giving their major version, but for example Apple's 13.x.y is very different from LLVM's 13.x.y. And compilers based on LLVM clang, for example from Intel and IBM, will define these. Some of the included software uses \_\_APPLE\_CC\_\_ to identify an Apple compiler (which used to include Apple builds of GCC), but Apple clang is better identified by the \_\_apple\_build\_version\_\_ macro.

## C.1 X11 issues

The 'X11()' graphics device is the one started automatically on Unix-alikes (except most macOS builds) when plotting. As its name implies, it displays on a (local or remote) X server, and relies on the services provided by the X server.

The 'modern' version of the 'X11()' device is based on 'cairo' graphics and (in most implementations) uses 'fontconfig' to pick and render fonts. This is done on the server, and although there can be selection issues, they are more amenable than the issues with 'X11()' discussed in the rest of this section.

When X11 was designed, most displays were around 75dpi, whereas today they are of the order of 100dpi or more. If you find that X11() is reporting<sup>2</sup> missing font sizes, especially larger ones, it is likely that you are not using scalable fonts and have not installed the 100dpi versions

<sup>&</sup>lt;sup>1</sup> Most clang-based compilers give 4, but not those distributed by FreeBSD. Intel's icx reported 12 in 2023.

 $<sup>^2\,</sup>$  for example, X11 font at size 14 could not be loaded.

of the X11 fonts. The names and details differ by system, but will likely have something like Fedora's

```
xorg-x11-fonts-75dpi
xorg-x11-fonts-100dpi
xorg-x11-fonts-IS08859-2-75dpi
xorg-x11-fonts-Type1
xorg-x11-fonts-cyrillic
```

and you need to ensure that the '-100dpi' versions are installed and on the X11 font path (check via xset -q). The 'X11()' device does try to set a pointsize and not a pixel size: laptop users may find the default setting of 12 too large (although very frequently laptop screens are set to a fictitious dpi to appear like a scaled-down desktop screen).

More complicated problems can occur in non-Western-European locales, so if you are using one, the first thing to check is that things work in the C locale. The likely issues are a failure to find any fonts or glyphs being rendered incorrectly (often as a pair of ASCII characters). X11 works by being asked for a font specification and coming up with its idea of a close match. For text (as distinct from the symbols used by plotmath), the specification is the first element of the option "X11fonts" which defaults to

"-adobe-helvetica-%s-%s-\*-\*-%d-\*-\*-\*-\*-\*"

If you are using a single-byte encoding, for example ISO 8859-2 in Eastern Europe or KOI8-R in Russian, use xlsfonts to find an appropriate family of fonts in your encoding (the last field in the listing). If you find none, it is likely that you need to install further font packages, such as 'xorg-x11-fonts-ISO8859-2-75dpi' and 'xorg-x11-fonts-cyrillic' shown in the listing above.

Multi-byte encodings (most commonly UTF-8) are even more complicated. There are few fonts in 'iso10646-1', the Unicode encoding, and they only contain a subset of the available glyphs (and are often fixed-width designed for use in terminals). In such locales *fontsets* are used, made up of fonts encoded in other encodings. If the locale you are using has an entry in the 'XLC\_LOCALE' directory (typically /usr/share/X11/locale), it is likely that all you need to do is to pick a suitable font specification that has fonts in the encodings specified there. If not, you may have to get hold of a suitable locale entry for X11. This may mean that, for example, Japanese text can be displayed when running in 'ja\_JP.UTF-8' but not when running in 'en\_GB.UTF-8' on the same machine (although on some systems many UTF-8 X11 locales are aliased to 'en\_US.UTF-8' which covers several character sets, e.g. ISO 8859-1 (Western European), JISX0208 (Kanji), KSC5601 (Korean), GB2312 (Chinese Han) and JISX0201 (Kana)).

On some systems scalable fonts are available covering a wide range of glyphs. One source is TrueType/OpenType fonts, and these can provide high coverage. Another is Type 1 fonts: the URW set of Type 1 fonts provides standard typefaces such as Helvetica with a larger coverage of Unicode glyphs than the standard X11 bitmaps, including Cyrillic. These are generally not part of the default install, and the X server may need to be configured to use them. They might be under the X11 fonts directory or elsewhere, for example,

```
/usr/share/fonts/default/Type1
/usr/share/fonts/ja/TrueType
```

## C.2 Linux

Linux is the main development platform for R, so compilation from the sources is normally straightforward with the most common compilers and libraries.<sup>3</sup>

This section is about the GCC compilers: gcc/gfortran/g++.

<sup>&</sup>lt;sup>3</sup> For example, glibc: other C libraries such as musl (as used by Alpine Linux) have been used but are not routinely tested.

Recall that some package management systems (such as RPM and deb) make a distinction between the user version of a package and the developer version. The latter usually has the same name but with the extension '-devel' or '-dev': you need both versions installed. So please check the configure output to see if the expected features are detected: if for example 'readline' is missing add the developer package. (On most systems you will also need 'ncurses' and its developer package, although these should be dependencies of the 'readline' package(s).) You should expect to see in the configure summary

Interfaces supported: X11, tcltk External libraries: pcre2, readline, curl Additional capabilities: PNG, JPEG, TIFF, NLS, cairo, ICU

When R has been installed from a binary distribution there are sometimes problems with missing components such as the Fortran compiler. Searching the 'R-help' archives will normally reveal what is needed.

It seems that 'ix86' Linux accepts non-PIC code in shared libraries, but this is not necessarily so on other platforms, in particular on 64-bit CPUs such as 'x86\_64'. So care can be needed with BLAS libraries and when building R as a shared library to ensure that position-independent code is used in any static libraries (such as the Tcl/Tk libraries, libpng, libjpeg and zlib) which might be linked against. Fortunately these are normally built as shared libraries with the exception of the ATLAS BLAS libraries.

The default optimization settings chosen for CFLAGS etc are conservative. It is likely that using -mtune will result in significant performance improvements on recent CPUs: one possibility is to add -mtune=native for the best possible performance on the machine on which R is being installed. It is also possible to increase the optimization levels to -O3: however for many versions of the compilers this has caused problems in at least one CRAN package.

Do not use -O3 with gcc 11.0 or 11.1: it mis-compiles code resulting in plausible but incorrect results. (This was seen in package MASS (https://CRAN.R-project.org/package=MASS) but has been worked around there as from version 3.1-57.)

For comments on 'ix86' builds (including 32-bit builds on 'x86\_64') see the version of this manual for R 4.3.x.

To build a 64-bit version of R on 'ppc64' (also known as 'powerpc64') with gcc 4.1.1, Ei-ji Nakama used

```
CC="gcc -m64"
CXX="gxx -m64"
FC="gfortran -m64"
CFLAGS="-mminimal-toc -fno-optimize-sibling-calls -g -O2"
FFLAGS="-mminimal-toc -fno-optimize-sibling-calls -g -O2"
```

the additional flags being needed to resolve problems linking against libnmath.a and when linking R as a shared library.

The setting of the macro 'SAFE\_FFLAGS' may need some help. It should not need additional flags on platforms other than '68000' (not likely to be encountered) and 'ix86'. For the latter, if the Fortran compiler is GNU (gfortran or possibly g77) the flags

```
-msse2 -mfpmath=sse
```

are added: earlier versions of R added -ffloat-store and this might still be needed if a 'ix86' CPU is encountered without SSE2 support. Note that it is a *replacement* for 'FFLAGS', so should include all the flags in that macro (except perhaps the optimization level).

Additional compilation flags can be specified for added safety/security checks. For example Fedora adds

```
-Werror=format-security -Wp,-D_FORTIFY_SOURCE=3 -Wp,-D_GLIBCXX_ASSERTIONS
-Fexceptions -fstack-protector-strong -fasynchronous-unwind-tables
```

#### -fstack-clash-protection -fcf-protection

to all the C, C++ and Fortran compiler flags (even though \_GLIBCXX\_ASSERTIONS is only for C++ in current GCC and glibc and none of these are documented for gfortran). Use of \_GLIBCXX\_ASSERTIONS will link abort and printf into almost all C++ code, and R CMD check --as-cran will warn.

### C.2.1 Clang

R has been built with Linux 'ix86' and 'x86\_64' C and C++ compilers (https://clang.llvm. org) based on the Clang front-ends, invoked by CC=clang CXX=clang++, together with gfortran. These take very similar options to the corresponding GCC compilers.

This has to be used in conjunction with a Fortran compiler: the configure code will remove -lgcc from FLIBS, which is needed for some versions of gfortran.

The current out-of-the-box default for clang++ is to use the C++ runtime from the installed g++. Using the runtime from the libc++ (https://libcxx.llvm.org/) project (Fedora RPM libcxx-devel) via -stdlib=libc++ has also been tested.

Recent versions have (optional when built) OpenMP support.<sup>4</sup>

There are problems mixing clang 15.0.0 and later built as default on Linux to produce PIE code and gfortran 11 or later, which does not. One symptom is that configure does not detect FC\_LEN\_T, which can be overcome by setting

FPIEFLAGS=-fPIE

in config.site. (configure tries that value if it is unset.)

#### C.2.2 flang

The name flang has been used for two projects: this is about the sub-project of LLVM which builds a Fortran compiler and runtime libraries. The compiler is currently named flang-new but has been announced to be renamed to flang when more nearly complete.

The version in LLVM 16 and later was able to build R on 'x86\_64' Linux with

```
FC=/path/to/flang-new
```

with the matching clang used as the C compiler, and the build passed make check-all. There is also support for 'aarch64' and 'ppc641e' Linux, but these have not been tested with R.

#### C.2.3 Intel compilers

In late 2020 Intel revamped their C/C++ compilers (and later their Fortran compiler) to use an LLVM back-end (and for the C/C++ compilers, a modified version of clang as the front-end). Those compilers are only for 'x86\_64': the earlier (now called 'Classic') C/C++ compilers were discontinued in late 2023 (and are covered in the version of this manual for R 4.3.x: the Fortran compiler ifort remains part of the Fortran distribution)..

The compilers are now all under Intel's 'oneAPI' brand. The revamped ones are icx, icpx and ifx; they are identified by the C/C++ macro \_\_INTEL\_LLVM\_COMPILER (and do not define \_\_INTEL\_COMPILER: they also define \_\_clang\_\_ and \_\_clang\_major\_\_).

The C++ compiler uses the system's libstdc++ as its runtime library rather than LLVM's libc++.

Standalone installers (which are free-of-charge) are available from https://www.intel. com/content/www/us/en/developer/articles/tool/oneapi-standalone-components. html: they are also part of the oneAPI Base and HPC (for Fortran) toolkits.

 $<sup>^4</sup>$  This also needs the OpenMP runtime which has sometimes been distributed separately.

We tried the compilers in oneAPI 2024.2.1 and 2023.x.y using (the paths do differ by compiler version)

```
IP=/path/to/compilers/bin/
CC=$IP/icx
CXX=$IP/icpx
FC=$IP/ifx
CFLAGS="-03 -fp-model precise -Wall -Wstrict-prototypes"
C17FLAGS="-03 -fp-model precise -Wall -Wno-strict-prototypes"
FFLAGS="-03 -fp-model precise -warn all,noexternals"
FCFLAGS="-free -03 -fp-model precise -warn all,noexternals"
CXXFLAGS="-03 -fp-model precise -Wall"
LDFLAGS="-L/path/to/compilers/compiler/lib -L/usr/local/lib64"
```

but the build segfaulted in the checks (in complex arithmetic in tests/lapack.R).

Intel document building Rwith MKL: for the Intel compilers this needed something like

```
MKL_LIB_PATH=/path/to/intel_mkl/mkl/lib/intel64
export LD_LIBRARY_PATH="$MKL_LIB_PATH"
MKL="-L${MKL_LIB_PATH} -lmkl_intel_lp64 -lmkl_core -lmkl_sequential"
./configure --with-blas="$MKL" --with-lapack
```

and the build passed its checks with MKL 2023.2.0 (but not 2024.x on the hardware tested). It may also be possible to use a compiler option like <code>-qmkl=sequential</code>.

One quirk is that the Intel Fortran compilers do not accept .f95 files, only .f90, for freeformat Fortran. configure adds -Tf which tells the compiler this is indeed a Fortran file (and needs to immediately precede the file name), but -free is needed to say it is free-format. Hence setting the FCFLAGS macro.

The compilers have many options: as the C/C++ and Fortran compilers have different origins for their front-ends, there is little consistency in their options. (The C/C++ compilers support 'all' clang options even if undocumented for icx/icpc, such as -Wno-strict-prototypes above, However it is unclear for which version of clang: the Intel manual suggests checking icx -help.) The C/C++ compilers support clang-style LTO: it is not clear if the Fortran one does.

For some earlier versions (including 2023.2.0) all CPU times in e.g. proc.time() are reported as zero. If you see this, uncomment the 'INTEL\_ICX\_FIX' setting in config.site and re-build.

The preferred Fortran standard for ifx can be set by one of -std90, -std95, -std03, -std08 or -std18 (and variants). However, this is documented to only affect warnings on non-standard features: the default is no such warnings.

Warning to package maintainers: the Intel Fortran compiler interprets comments intended for Visual Fortran<sup>5</sup> like

!DEC\$ ATTRIBUTES DLLEXPORT, C, REFERENCE, ALIAS: 'kdenestmlcvb' :: kdenestmlcvb

The DLLEXPORT gives a warning but the remainder silently generates incorrectly named entry points. Such comment lines need to be removed from code for use with R (even if using Intel Fortran on Windows).

## C.3 macOS

The instructions here are for 'Apple Silicon' ('arm64') or Intel 64-bit ('x86\_64') builds on macOS 11 (Big Sur), 12 (Monterey), 13 (Ventura), 14 (Sonoma) and likely later. (They may well work on Intel macOS 10.14 or 10.15, but are untested there.)

 $<sup>^{5}</sup>$  as the 'Classic' compiler has been known on Windows.

### C.3.1 Prerequisites

The Apple silicon components install into /opt/R/arm64, the Intel ones into  $/opt/R/x86_64$ . That may not exist<sup>6</sup> so it is simplest to first create the directory and adjust its ownership if desired: for example by

sudo mkdir -p /opt/R/arm64
sudo chown -R \$USER /opt/R

Also, add /opt/R/arm64/bin or /opt/R/x86\_64/bin to your path.

Define an appropriate variable in your Terminal:

set LOCAL=/opt/R/arm64 # Apple Silicon
set LOCAL=/opt/R/x86\_64 # Intel

to use the code snippets here.

The following are essential to build R:

• Apple's 'Command Line Tools': these can be (re-)installed by running xcode-select --install in a terminal.

If you have a fresh OS installation, running e.g. make in a terminal will offer the installation of the command-line tools. If you have installed Xcode, this provides the command-line tools. The tools may need to be reinstalled when macOS is upgraded, as upgrading may partially or completely remove them.

The Command Line Tools provide C and C++ compilers derived from LLVM's clang but nowadays known as 'Apple clang' with different versioning (so Apple clang 15 is unrelated to LLVM clang 15).

- A Fortran compiler. See Section C.3.2 [Fortran compilers], page 64.
- Binary components pcre2<sup>7</sup> and xz (for liblzma) from https://mac.r-project.org/bin/ . There is an R script there to help with installing all the needed components. (At the time of writing install.libs("r-base-dev") installed neither readline5 nor those needed to support Pango.)

Intel users want the darwin20 components: the darwin17 ones are for macOS 10.13-10.15.

Or this can be done manually, by for example

```
curl -OL https://mac.r-project.org/bin/darwin20/arm64/pcre2-10.42-darwin.20-arm64.t
sudo tar -xvzf pcre2-10.42-darwin.20-arm64.tar.gz -C /
curl -OL https://mac.r-project.org/bin/darwin20/arm64/xz-5.4.2-darwin.20-arm64.tar.s
sudo tar -xvzf xz-5.4.2-darwin.20-arm64.tar.xz -C /
```

(sudo is not needed if your account owns /opt/R/arm64 or /opt/R/x86\_64 as appropriate.)

Messages like 'opt/R/: Can't restore time' should be ignored.

and desirable

- Component readline5.<sup>8</sup> If readline is not present, the emulation in Apple's version of libedit (aka editline) will be used: if you wish to avoid that, configure with --without-readline.
- Components jpeg, libpng, pkgconfig, tiff and zlib-system-stub from https://mac. r-project.org/bin// for the full range of bitmapped graphics devices. (Some builds of tiff may require libwebp and/or openjpeg.)

 $<sup>^{6}\,</sup>$  it will if R has been installed from CRAN since R 4.3.0.

<sup>&</sup>lt;sup>7</sup> If compiling it from source on 'arm64', pcre2 (at least up to version 10.39) needs to be built without JIT support (the default) as the R build segfaults if that is enabled, so do run make check on your build.

<sup>&</sup>lt;sup>8</sup> For licence reasons this is version 5.2 of readline: for those who want a more recent version it is straightforward to compile it from its sources.

• An X sub-system unless configuring using --without-x: see https://www.xquartz.org/. R's configure script can be told to look for X11 in XQuartz's main location of /opt/X11, e.g. by

```
--x-includes=/opt/X11/include --x-libraries=/opt/X11/lib
```

Be wary of pre-release versions of XQuartz, which may be offered as an update.

• An Objective-C compiler, as provided by clang in the Command Line Tools: this is needed for the quartz() graphics device.

Use --without-aqua if you want a standard Unix-alike build: apart from disabling quartz() and the ability to use the build with R.APP, it also changes the default location of the personal library (see ?.libPaths).

- A Tcl/Tk installation, See (undefined) [Tcl/Tk headers and libraries], page (undefined).
- Support for Cairo-based graphics devices. See Section C.3.3 [Cairo graphics], page 65.
- A TeX installation. See Section C.3.5 [Other libraries], page 66.
- texi2any from a Texinfo distribution, which requires perl (currently a default part of macOS but it has been announced that it may not be in future). A version of texi2any has been included in the binary distribution of R and there is a 'texinfo' component at https://mac.r-project.org/bin/.

To build R itself from the sources with the C/C++ compilers in the Command Line Tools (or Xcode) and gfortran from the installer mentioned below, use a file config.site containing

```
CC=clang
OBJC=$CC
FC="/opt/gfortran/bin/gfortran -mtune=native"
CPPFLAGS='-isystem $LOCAL/include'
CXX=clang++
```

and configure by something like

```
./configure -C \
--enable-R-shlib --enable-memory-profiling \
--x-includes=/opt/X11/include --x-libraries=/opt/X11/lib \
--with-tcl-config=$LOCAL/lib/tclConfig.sh \
--with-tk-config=$LOCAL/lib/tkConfig.sh \
PKG_CONFIG_PATH=$LOCAL/lib/pkgconfig:/usr/lib/pkgconfig
```

(See below for other options for Tcl/Tk.) For an 'arm64' build further flags are desirable in config.site:

```
CFLAGS="-falign-functions=8 -g -02"
```

is needed to inter-work with gfortran without segfaulting in some packages. Some builds of gfortran have targetted the current version of macOS (unlike clang), causing linker warnings: to avoid these use

```
FFLAGS="-g -02 -mmacosx-version-min=11.0"
FCFLAGS="-g -02 -mmacosx-version-min=11.0"
```

or perhaps

```
FFLAGS="-g -O2 -mmacos-version-min=11.0"
FCFLAGS="-g -O2 -mmacos-version-min=11.0"
```

where 11.0 can be replaced by 12.0, 13.0 or 14.0 for macOS 12.x, 13.x and 14.x.

To install packages using compiled code one needs the Command Line Tools (or Xcode) and appropriate compilers, e.g. the C/C++ compilers from those tools and/or gfortran. Some packages have further requirements such as component pkgconfig (and to set PKG\_CONFIG\_PATH= as above).

A subversion client can be obtained from https://mac.r-project.org/tools/, for example by (Apple Silicon)

```
curl -OL https://mac.r-project.org/tools/subversion-1.14.1-darwin.20-arm64.tar.gz
tar xf subversion-1.14.1-darwin.20-arm64.tar.gz
sudo cp subversion-1.14.1-darwin-20-arm64/svn $LOCAL/bin
```

#### or (Intel)

```
curl -OL https://mac.r-project.org/tools/subversion-1.14.0-darwin15.6.tar.gz
tar xf subversion-1.14.0-darwin15.6.tar.gz
sudo cp subversion-1.14.0-darwin15.6/svn $LOCAL/bin
```

If building software or installing source packages with cmake (or a non-Apple make) for 'Apple Silicon' ensure it contains the 'arm64' architecture (use file to be sure). Running Apple compilers from an 'x86\_64' executable will generate 'x86\_64' code ....

Updating an 'arm64' build may fail because of the bug described at https://openradar. appspot.com/FB8914243 but *ab initio* builds work. This has been far rarer since macOS 13.

If you are using the macOS 13 SDK<sup>9</sup>, you may need to add something like -mmacos-versionmin=12.0 to 'CFLAGS'.

Linker warnings like

- ld: warning: could not create compact unwind for \_sort\_:
  - register 26 saved somewhere other than in frame
- ld: warning: ld: warning:
- could not create compact unwind for \_arcoef\_: registers 23 and 24 not saved contigue ld: warning: could not create compact unwind for \_\_\_emutls\_get\_address:
- registers 23 and 24 not saved contiguously in frame

can be ignored. These stem from compiled Fortran code, including its run-time libraries.

The default security settings can make it difficult to install Apple packages which have not been 'notarized'<sup>10</sup> by Apple. And not just packages, as this has been seen for executables contained in tarballs/zipfiles (for example, for pandoc). Usually one can use 'Open With' (Control/right/two-finger-click in Finder), then select 'Installer' and 'Open' if you get a further warning message.

If you run into problems with 'quarantine' for tarballs downloaded in a browser, consider using curl -OL to download (as illustrated above) or xattr -c to remove extended attributes.

configure defaults to --with-internal-tzcode on macOS. The native implementation used to be unusable on earlier versions (with a 32-bit time\_t and/or timezone tables missing information beyond the 32-bit range). As from macOS 12.6, option --without-internaltzcode can be used to override this and R contains sufficient workarounds (for example, the native code fails to recognize dates with a negative tm\_year, that is dates before 1900) for R to pass its checks. However, there are discrepancies, notably in Europe in the 1900s and 1940s, even though the Olson database contains the correct information.

#### C.3.2 Fortran compiler

There is a 'universal' (arm64 and Intel) build of gfortran 12.2 at https://mac.r-project. org/tools/gfortran-12.2-universal.pkg. This installs into /opt/gfortran.

The /opt/gfortran/SDK symlink should point to the desired path to the SDK (defaults to the command line tools SDK). This can be updated by running /opt/gfortran/bin/gfortran-update-sdk or manually. If the symlink is broken, the driver will issue a warning and use xcrun

<sup>&</sup>lt;sup>9</sup> 1s -1 `xcrun -show-sdk-path` in a terminal will show you which SDK is selected.

 $<sup>^{10} {\</sup>rm ~See~https://developer.apple.com/documentation/xcode/notarizing_macos_software_before_distribution.}$ 

-show-sdk-path to try to find an SDK and use its path. (The SDK path is used when using gfortran to link, so not when building R but when installing a few packages.)

Builds of gfortran 13.2 and 14.1 for arm64 macOS 14 are available at https://github.com/fxcoudert/gfortran-for-macOS/releases. These can be built for Intel and older OSes from the sources at https://github.com/iains/gcc-13-branch/ and gcc-14-branch/. To use one of the pre-built compilers with Apple clang needs something like

```
LDFLAGS="-L/opt/R/arm64/lib -rpath /usr/local/gfortran/lib"
```

in config.site to ensure the Fortran run-time libraries are found.

## C.3.3 Cairo graphics

Cairo-based graphics devices such as cairo\_ps, cairo\_pdf, X11(type = "cairo") and the Cairo-based types of devices bmp jpeg, png and tiff are not the default on macOS, and much less used than the Quartz-based devices. However, the only SVG device in the R distribution, svg, is based on Cairo.

Support for Cairo is optional and can be added in several ways, all of which need pkg-config. configure will add Cairo support if pkg-config finds package cairo unless --without-cairo is used.

A way to statically link Cairo is by downloading and unpacking components cairo, fontconfig, freetype, pixman and zlib-system-stub (and do not have /opt/X11/lib/pkgconfig in PKG\_CONFIG\_PATH). Some static builds of fontconfig need libxml2 (from component xml2) and others expat, supplied by macOS but needing a file \$LOCAL/lib/pkgconfig/expat.pc along the lines of

```
Name: expat
Version: 2.2.8
Description: expat XML parser
URL: http://www.libexpat.org
Libs: -lexpat
Cflags:
```

Note that the list of components is liable to change: running pkg-config cairo --exists --print-errors should tell you if any others are required.

The best font experience of Cairo graphics will be to use it in combination with Pango which will match that supported on most other Unix-alikes. configure uses pkg-config to determine if all the external software required by both Pango and Cairo is available: running pkg-config pangocairo --exists --print-errors should show if the installation suffices and if not, what is missing. At the time of writing using pre-built components cairo, fontconfig, freetype, ffi, fribidi, gettext, icu, glib, harfbuzz, pango, pcre, pixman and xml2 sufficed.

## C.3.4 Other C/C++ compilers

Other pre-compiled distributions of clang may be available from https://github.com/llvm/ llvm-project/releases/ (recently only for arm64 and usually unsigned/not notarized which makes them hard to use). In particular, these include support for OpenMP which Apple clang does not. Some of these have included support for the ASan and UBSan sanitizers.

Suppose one of these distributions is installed under LOCAL/llvm. Use a file config.site containing

```
SDK=`xcrun -show-sdk-path`
CC="$LOCAL/llvm/bin/clang -isysroot $SDK"
CXX="$LOCAL/llvm/bin/clang++ -isysroot $SDK"
OBJC=$CC
FC=/opt/gfortran/bin/gfortran
```

```
LDFLAGS="-L$LOCAL/llvm/lib -L$LOCAL/lib"
R_LD_LIBRARY_PATH=$LOCAL/llvm/lib:$LOCAL/lib
```

The care to specify library paths is to ensure that the OpenMP runtime library, here \$LOCAL/llvm/lib/libomp.dylib, is found when needed. If this works, you should see the line

checking whether OpenMP SIMD reduction is supported... yes

in the configure output. Also, 'R\_LD\_LIBRARY\_PATH' needs to be set to find the latest version of the C++ run-time libraries rather than the system ones.

It is normally possible to build R with GCC (built from the sources, from a gfortran distribution, from Homebrew, ...). When last tested it was not possible to use gcc to build the quartz() device, so configure --without-aqua may be required. R was built and tested with the GCC 14.1 compilers in the arm64 gfortran distribution mentioned above using a config.site containing

```
CC=/usr/local/gfortran/bin/gcc

CXX=/usr/local/gfortran/bin/g++

FC=/usr/local/gfortran/bin/gfortran

CFLAGS="-g -02 -Wall -pedantic -Wstrict-prototypes"

C17FLAGS="-g -02 -Wall -pedantic -Wno-strict-prototypes"

C90FLAGS=$C17FLAGS

C99FLAGS=$C17FLAGS

CXXFLAGS="-g -02 -Wall -pedantic"

CPPFLAGS='-isystem /opt/R/arm64/include'

LDFLAGS=-L/opt/R/arm64/lib
```

It is usually possible to add some OpenMP support to the Apple clang compilers: see https://mac.r-project.org/openmp/. Note that that approach is somewhat fragile as it needs a libomp.dylib library matching the version of the compiler used—and for example at the time of writing none was offered for the current compilers in Xcode CLT 14.3 nor 15.

### C.3.5 Other libraries

Pre-compiled versions of many of the Section A.2 [Useful libraries and programs], page 41, are available from https://mac.r-project.org/bin//.

Looking at the top of /Library/Frameworks/R.framework/Resources/etc/Makeconf will show the compilers and configuration options used for the CRAN binary package for R: at the time of writing the non-default options

```
--enable-memory-profiling --enable-R-framework
--x-libraries=/opt/X11/lib --x-includes=/opt/X11/include
```

were used. (--enable-R-framework implies --enable-R-shlib.)

The main T<sub>E</sub>X implementation used by the developers is MacTeX<sup>11</sup> (https://www.tug. org/mactex/): the full installation is about 8.5GB, but a much smaller version ('Basic TeX') is available at https://www.tug.org/mactex/morepackages.html to which you will need to add some packages to build R, e.g. for the 2022 version we needed to add<sup>12</sup> helvetic, inconsolata and texinfo which brought this to about 310MB.<sup>13</sup> 'TeX Live Utility' (available *via* the MacTeX front page) provides a graphical means to manage T<sub>E</sub>X packages. These contain executables which run natively on both 'arm64' and 'x86\_64'.

Checking packages thoroughly requires Ghostscript (part of the full MacTeX distribution or separately from https://www.tug.org/mactex/morepackages.html) and qpdf (from

 $<sup>^{11}</sup>$  An essentially equivalent TeX installation can be obtained by the Unix TeX Live installation scripts.

 $<sup>^{12}\,</sup>$  E.g. via tlmgr install helvetic inconsolata texinfo .

<sup>&</sup>lt;sup>13</sup> Adding all the packages needed to check CRAN increased this to about 600MB.

https://mac.r-project.org/bin//, a version of which is in the bin directory of a binary installation of R, usually /Library/Frameworks/R.framework/Resources/bin/qpdf).

R CMD check --as-cran makes use of 'HTML Tidy'. macOS at the time of writing has a version in /usr/bin/tidy dating from 2006 which is far too old. Up-to-date versions can be installed from http://binaries.html-tidy.org/.

One macOS quirk is that the default path has /usr/local/bin after /usr/bin, contrary to common practice on Unix-alikes. This means that if you install tools from the sources they will by default be installed under /usr/local and not supersede the system versions.

Parallel installation of packages will make use of the utility timeout if available. A 'universal' build can be downloaded from https://www.stats.ox.ac.uk/pub/bdr/timeout: make it executable (chmod 755 timeout) and put it somewhere on your path.

### C.3.5.1 Accelerate

The Accelerate library<sup>14</sup> can be used *via* the configuration option

### --with-blas="-framework Accelerate"

to provide potentially higher-performance versions of the BLAS and LAPACK routines.<sup>15</sup> This includes a full LAPACK which can be used *via* --with-lapack: however, the version of LA-PACK it contains has often been seriously old (and is not used unless --with-lapack is specified). Some CRAN builds of R can be switched<sup>16</sup> to use Accelerate's BLAS.

As from macOS 13.3, the BLAS and LAPACK libraries under the Accelerate framework are 'now inline with reference version 3.9.1'.<sup>17</sup> However, this has been done by naming new entry points and so only accessible *via* their C headers. That version can be used for BLAS calls *via* configure option --with-newAccelerate: it requires at least macOS 13.3 and SDK 13.3 (from Xcode CLT 14.3). To use it for both BLAS and LAPACK calls, configure with --with-newAccelerate=lapack. These options cannot be used with others such as --with-blas and --with-lapack.

Threading in Accelerate is controlled by 'Grand Central Dispatch'<sup>18</sup> and is said not to need user control. Test nls.R in package stats has often failed with the Accelerate BLAS on Intel macOS. All versions of Accelerate show differences from the reference BLAS (and most others) in the use of NA vs NaN and a substantial number of R packages fail their checks.

#### C.3.5.2 OpenBLAS

R has been built on 'arm64' using OpenBLAS 0.3.24 (sources from https://github.com/ OpenMathLib/OpenBLAS/releases) by symlinking /opt/OpenBLAS/lib/libopenblas.dylib to lib/libRblas.dylib (see Section A.3.1.4 [Shared BLAS], page 48).

On macOS, a default build of OpenBLAS uses pthreads (as macOS does not have OpenMP) with the number of threads controlled by environment variable OPENBLAS\_NUM\_THREADS. On an M1 Pro this defaulted to 10 threads (there are 8 'performance' cores and 2 'efficiency cores') and we saw a 9x speedup over the reference BLAS on a large SVD (which was slightly faster than Accelerate).

#### C.3.6 Tcl/Tk headers and libraries

If you plan to use the tcltk package for R, you will need to install a distribution of Tcl/Tk. There are two alternatives. If you use R.APP you will want to use X11-based Tcl/Tk (as used

 $<sup>^{14}</sup>$  https://developer.apple.com/documentation/accelerate.

<sup>&</sup>lt;sup>15</sup> It has been reported that for some non-Apple toolchains CPPFLAGS needed to contain -D\_\_ACCELERATE\_\_: not needed for clang from LLVM.

 $<sup>^{16} \ \</sup>texttt{https://cran.r-project.org/bin/macosx/RMacOSX-FAQ.html\#Which-BLAS-is-used-and-how-can-it-be-changed_003f}$ 

 $<sup>^{17}</sup>$  Released 2021-04-01.

<sup>&</sup>lt;sup>18</sup> E.g., https://en.wikipedia.org/wiki/Grand\_Central\_Dispatch .

on other Unix-alikes), which is installed under \$LOCAL/lib as part of the CRAN binary for R.<sup>19</sup> This may need configure options

```
--with-tcltk=$LOCAL/lib
```

or

```
--with-tcl-config=$LOCAL/lib/tclConfig.sh
--with-tk-config=$LOCAL/lib/tkConfig.sh
```

Note that this requires a matching XQuartz installation.

There is also a native ('Aqua') version of Tcl/Tk which produces widgets in the native macOS style: this will not work with R.APP because of conflicts over the macOS menu, but for those only using command-line R this provides a much more intuitive interface to Tk for experienced Mac users. Earlier versions of macOS came with an Aqua Tcl/Tk distribution but these were often not at all recent versions of Tcl/Tk. It is better to install Tcl/Tk 8.6.x from the sources<sup>20</sup> or a binary distribution from https://www.activestate.com/platform/supported-languages/tcl/. For the latter, configure R with

```
--with-tcl-config=/Library/Frameworks/Tcl.framework/tclConfig.sh
--with-tk-config=/Library/Frameworks/Tk.framework/tkConfig.sh
```

If you need to find out which distribution of Tk is in use at run time, use

```
library(tcltk)
```

tclvalue(.Tcl("tk windowingsystem")) # "x11" or "aqua"

Note that some Tcl/Tk extensions only support the X11 interface: this includes Tktable and the CRAN package tkrplot (https://CRAN.R-project.org/package=tkrplot).

### C.3.7 Java

macOS does not comes with an installed Java runtime (JRE) and a macOS upgrade may remove one if already installed: it is intended to be installed at first use. Check if a JRE is installed by running java -version in a Terminal window: if Java is not installed this may prompt you to install it from Oracle<sup>21</sup> (but see the next paragraph). We recommend you install a version with long-term support, e.g. 17 or  $21^{22}$  but not 18–20, 22–24 with a 6-month lifetime.

The currently simplest way to install Java is from Adoptium (https://adoptium.net)<sup>23</sup>: this installs into an Apple-standard location and so works with /usr/bin/java. Other builds of OpenJDK are available from https://www.azul.com/downloads/zulu-community/?os=macos&architecture=arm-64-bit&package=jdk and from OpenJDK at https://jdk. java.net/, for which JAVA\_HOME may need to be set both when configuring R and at run-time. Note that Java distribution sites may use unusual designations for macOS CPUs such as AArch64, x64 or x86 64-bit.

Binary distributions of R are built against a specific version (e.g. 11.0.18 or 17.0.1) of Java so

sudo R CMD javareconf

will likely be needed to be run before using Java-using packages.

To see what compatible versions of Java are currently installed, run the appropriate one of

/usr/libexec/java\_home -V -a arm64

 $<sup>^{19}\,</sup>$  Just that component can be selected from the installer for R: at the 'Installation Type' screen select 'Customise' and then just the 'Tcl/Tk 8.6.11' component.

 $<sup>^{20}\,</sup>$  Configure Tk with --enable-aqua.

 $<sup>^{21}\,</sup>$  Oracle Java has a restrictive licence, unlike distributions based on OpenJDK.

 $<sup>^{22}\,</sup>$  The planned nextLTS release is 25 in September 2025. Java 8 aka 1.8.0 is still LTS but some packages require 11 or later.

 $<sup>^{23}\,</sup>$  which website works with Safari but not some other browsers.

#### /usr/libexec/java\_home -V -a x86\_64

If needed, set the environment variable  $JAVA\_HOME$  to choose between these, both when R is built from the sources and when R CMD javareconf is run.

Configuring and building R both looks for a JRE and for support for compiling JNI programs (used to install packages rJava (https://CRAN.R-project.org/package=rJava) and JavaGD (https://CRAN.R-project.org/package=JavaGD)); the latter requires a JDK (Java SDK). Most distributions of Java 11 or later are of a full JDK.

The build process tries to fathom out what JRE/JDK to use, but it may need some help, e.g. by setting environment variable JAVA\_HOME. To select a build from Adoptium (https://adoptium.net) set e.g.

```
JAVA_HOME=/Library/Java/JavaVirtualMachines/temurin-21.jdk/Contents/Home
```

in config.site. For Java 21 from https://jdk.java.net/ (which might no longer be available), use

#### JAVA\_HOME=/path/to/jdk-21.jdk/Contents/Home

Note that it is necessary to set the environment variable NOAWT to 1 to install many of the Java-using packages.

#### C.3.8 Frameworks

The CRAN build of R is installed as a framework, which is selected by the option

```
./configure --enable-R-framework
```

(This is intended to be used with an Apple toolchain: others may not support frameworks correctly but those from LLVM have done so.)

It is only needed if you want to build R for use with the R.APP console, and implies --enable-R-shlib to build R as a dynamic library. This option configures R to be built and installed as a framework called R.framework. The default installation path for R.framework is /Library/Frameworks but this can be changed at configure time by specifying the flag --enable-R-framework[=DIR] (or --prefix) or at install time *via* 

```
make prefix=/where/you/want/R.framework/to/go install
```

Note that installation as a framework is non-standard (especially to a non-standard location) and Unix utilities may not support it (e.g. the pkg-config file libR.pc will be put somewhere unknown to pkg-config).

### C.3.9 Building R.app

Building the R.APP GUI console is a separate project, using Xcode. Before compiling R.APP make sure the current version of R is installed in /Library/Frameworks/R.framework and is working at the command-line (this can be a binary install).

The current sources can be checked out by

```
svn co https://svn.r-project.org/R-packages/trunk/Mac-GUI
```

and built by loading the R.xcodeproj project (select the R target and a suitable configuration), or from the command-line by e.g.

```
xcodebuild -target R -configuration Release
```

See also the INSTALL file in the checkout or directly at https://svn.r-project.org/ R-packages/trunk/Mac-GUI/INSTALL.

R.APP does not need to be installed in any specific way. Building R.APP results in the R.APP bundle which appears as one R icon. This application bundle can be run from anywhere and it is customary to place it in the /Applications folder.

## C.3.10 Building binary packages

CRAN macOS binary packages are distributed as tarballs with suffix .tgz to distinguish them from source tarballs. One can tar an existing installed package, or use R CMD INSTALL --build.

However, there are some important details.

• Current CRAN macOS distributions are targeted at Big Sur so it is wise to ensure that the compilers generate code that will run on Big Sur or later. With the recommended compilers we can use

```
CC="clang -mmacos-version-min=11.0"
CXX="clang++ -mmacos-version-min=11.0"
FC="/opt//gfortran/bin/gfortran -mmacosx-version-min=11.0"
```

or set the environment variable

```
export MACOSX_DEPLOYMENT_TARGET=11.0
```

- Using the flag -Werror=partial-availability can help trigger compilation errors on functionality not in Big Sur.
- Check that any compiled code is not dynamically linked to libraries only on your machine, for example by using otool -L or objdump -macho -dylibs-used. This can include C++ and Fortran run-time libraries under /opt/R/x86\_64/lib or /opt/R/arm64/lib: one can use install\_name\_tool to point these at system versions or those shipped with R, for example

```
install_name_tool -change /usr/local/llvm/lib/libc++.1.dylib \
/usr/lib/libc++.1.dylib \
pkg.so
```

```
install_name_tool -change
/opt/gfortran/lib/gcc/aarch64-apple-darwin20.0/12.2.0/libgfortran.5.dylib \
/Library/Frameworks/R.framework/Resources/lib/libgfortran.5.dylib \
pkg.so
```

```
install_name_tool -change
/opt/gfortran/lib/gcc/aarch64-apple-darwin20.0/12.2.0/libquadmath.0.dylib \
/Library/Frameworks/R.framework/Resources/lib/libquadmath.0.dylib \
pkg.so
```

(where the details depend on the compilers and CRAN macOS R release).

• For C++ code there is the possibility that calls will be generated to entry points not in the system /usr/lib/libc++.1.dylib. The previous step allows this to be tested against the system library on the build OS, but not against earlier ones. It may be possible to circumvent that by static linking to libc++.a and libc++abi.a by something like

```
SHLIB_CXXLD = /usr/local/llvm/bin/clang
PKG_LIBS = /usr/local/llvm/lib/libc++.a /usr/local/llvm/lib/libc++abi.a
in src/Makevars. It would also be possible to static link the Fortran runtime libraries
libgfortran.a and libquadmath.a should the Fortran compiler have later versions (but
gfortran 8-14 all have version 5).
```

The CRAN binary packages are built with the Apple compiler on the oldest supported version of macOS, which avoids the first two and any issues with C++ libraries.

## C.3.11 Building for Intel on 'arm64'

Should one want to build R for Intel on an 'arm64' Big Sur Mac, add the target for the compilers:

```
CC="clang -arch x86_64
```

OBJC=\$CC CXX="clang++ -arch x86\_64"

FC="/opt//gfortran/bin/gfortran -arch x86\_64 -mtune=native -mmacosx-version-min=11"

and install the Fortran compiler and external software described above for Intel builds (and have  $/opt/Rx86_64/bin$  before /opt/R/arm64/bin in your path).

To set the correct architecture (which will be auto-detected as **aarch64**), use something like

/path/to/configure --build=x86\_64-apple-darwin20

## C.3.12 Installer

The scripts for the CRAN packaging of R can be found under https://svn.r-project.org/ R-dev-web/trunk/QA/Simon/R4/: start with the README file in that directory.

## C.4 FreeBSD

There have been few recent reports on FreeBSD: there is a 'port' at https://svnweb.freebsd. org/ports/head/math/R, currently last updated for R 4.0.4. Recent versions of FreeBSD use Clang and the libc++ C++ headers and runtime, but the 'port' has been configured to use GCC.

Use of ICU for collation and the configure option --with-internal-tzcode are desirable workarounds.

## C.5 OpenBSD

Ingo Feinerer installed R version 3.2.2 on OpenBSD 5.8 arch 'amd64' (their name for 'x86\_64'). Details of the build (and patches applied) are at https://cvsweb.openbsd.org/cgi-bin/ cvsweb/ports/math/R/, currently updated for R 4.2.3.

## C.6 Cygwin

The 32-bit version never worked well enough to pass R's make check, and residual support from earlier experiments was removed in R 3.3.0.

The 64-bit version was never supported.

## C.7 New platforms

There are a number of sources of problems when installing R on a new hardware/OS platform. These include

Floating Point Arithmetic: R requires arithmetic compliant with IEC 60559, also known as IEEE 754. This mandates the use of plus and minus infinity and NaN (not a number) as well as specific details of rounding. Although almost all current FPUs can support this, selecting such support can be a pain. The problem is that there is no agreement on how to set the signalling behaviour; Sun/Sparc, SGI/IRIX and 'ix86' Linux require no special action, FreeBSD requires a call to (the macro) fpsetmask(0) and OSF1 required that computation be done with a -ieee\_with\_inexact flag etc. With Intel compilers on 32-bit and 64-bit Intel machines, one has to explicitly disable flush-to-zero and denormals-are-zero modes. Some ARM processors including A12Z and M1 (Apple Silicon) by default use runfast mode, which includes flush-to-zero and default-nan and hence has to be disabled. With default-nan mode, the NaN payload used for representation of numeric NA values is lost even on simple operations with finite values. On a new platform you must find out the magic recipe and add some code to make it work. This can often be done via the file config.site which resides in the top level directory.

Beware of using high levels of optimization, at least initially. On many compilers these reduce the degree of compliance to the IEEE model. For example, using **-fast** on the Oracle compilers has caused R's NaN to be set incorrectly, and gcc's -ffast-math and clang's -Ofast have given incorrect results.

Shared Objects: There seems to be very little agreement across platforms on what needs to be done to build shared objects. there are many different combinations of flags for the compilers and loaders. GNU libtool cannot be used (yet), as it currently does not fully support Fortran: one would need a shell wrapper for this). The technique we use is to first interrogate the X window system about what it does (using xmkmf), and then override this in situations where we know better (for tools from the GNU Compiler Collection and/or platforms we know about). This typically works, but you may have to manually override the results. Scanning the manual entries for cc and ld usually reveals the correct incantation. Once you know the recipe you can modify the file config.site (following the instructions therein) so that the build will use these options.

It seems that gcc 3.4.x and later on 'ix86' Linux defeat attempts by the LA-PACK code to avoid computations entirely in extended-precision registers, so file src/modules/lapack/dlamc.f may need to be compiled without optimization or with additional flags. Set the configure variable SAFE\_FFLAGS to the flags to be used for this file.

If you do manage to get R running on a new platform please let us know about it so we can modify the configuration procedures to include that platform.

If you are having trouble getting R to work on your platform please feel free to use the 'R-devel' mailing list to ask questions. We have had a fair amount of practice at porting R to new platforms . . .

One thing you might want to add for a new platform is the mapping of C/C++/Fortran calls to entry point names used for R CMD check. See https://svn.r-project.org/R-dev-web/trunk/sotools.txt for how to do so.

# Function and variable index

С	R
configure	R_HOME         3           remove.packages         30
I install.packages24	<b>U</b> update.packages
$\mathbf{M}$	
make	

# Concept index

# В

BLAS library	BLAS	library.					44, 67
--------------	------	----------	--	--	--	--	--------

# $\mathbf{C}$

Cairo	 	19,41,57,65

# $\mathbf{F}$

Fortran	54
FreeBSD	71

# Ι

Installation
Installing under Unix-alikes
Installing under Windows16
Internationalization

## $\mathbf{L}$

LAPACK library
Libraries
Libraries, managing 23
Libraries, site
Libraries, user
Linux
Locale
Localization

# $\mathbf{M}$

macOS3, 19, 6	31
Manuals	5
Manuals, installing	8

# 0

Obtaining	$\mathbf{R}\ldots$	 	 		1
OpenBSD		 • • • •	 	••••	71

## $\mathbf{P}$

De alas margina de la composición de la compos	9
Packages	3
Packages, default 2	3
Packages, installing 2	3
Packages, removing	0
Packages, updating 2	9
Pango 41, 6	5

## $\mathbf{R}$

Repositories																											3	30
--------------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	---	----

# $\mathbf{S}$

Site libraries
Sources for R 1
Subversion

# $\mathbf{U}$

User	libraries	3
------	-----------	---

## $\mathbf{V}$

Vignettes	40
-----------	----

# Environment variable index

## В

BLAS_LIBS	 	 

## $\mathbf{C}$

•				
CONFIG_SIT	Έ	•••••	 	53

## D

## J

## $\mathbf{L}$

LANG 33
LANGUAGE
LAPACK_LIBS 49
LC_ALL
LC_COLLATE
LC_MESSAGES
LD_LIBRARY_PATH

## Ρ

PAPERSIZE	53
PATH	54

## $\mathbf{R}$

R_ARCH
<b>R_BROWSER</b>
R_DEFAULT_PACKAGES
R_DISABLE_HTTPD
<b>R_GSCMD</b>
<b>R_INSTALL_TAR</b>
R_JAVA_LD_LIBRARY_PATH
<b>R_LIBS</b>
<b>R_LIBS_SITE</b>
<b>R_LIBS_USER</b>
<b>R_PAPERSIZE</b>
<b>R_PDFVIEWER</b>
<b>R_RD4PDF</b>
R_SCRIPT_DEFAULT_PACKAGES
<b>R_USER</b>

## $\mathbf{T}$

TAR	0
TAR_OPTIONS	1
ТЕМР 22	2
TMP	2
TMPDIR	4