

# JAGS Version 4.3.1 installation manual

Martyn Plummer

Bill Northcott

Matt Denwood

9 April 2022

JAGS is distributed in binary format for Microsoft Windows, macOS, and most Linux distributions. The following instructions are for those who wish to build JAGS from source. The manual is divided into three sections with instructions for Linux/Unix, macOS, and Windows.

## 1 Linux and UNIX

JAGS follows the usual GNU convention of

```
./configure
make
make install
```

which is described in more detail in the file `INSTALL` in the top-level source directory. On some UNIX platforms, you may be required to use GNU make (`gmake`) instead of the native `make` command. On systems with multiple processors, you may use the option ‘`-j`’ to speed up compilation, *e.g.* for a quad-core PC you may use:

```
make -j4
```

If you have the `cppunit` library installed then you can test the build with

```
make check
```

WARNING. If you already have a copy of the `jags` library installed on your system then the test program created by `make check` will link against the **installed** library and not the one in your build directory. So if the test suite includes a regression test for a bug that was fixed in the version you are building but a previous version of JAGS is already installed then the unit tests will fail. Best practice is to run the tests after `make install` so the build and installed versions are the same.

### 1.1 Configure options

At configure time you also have the option of defining options such as:

- The names of the C, C++, and Fortran compilers.
- Optimization flags for the compilers. JAGS is optimized by default if the GNU compiler (`gcc`) is used. If you are using another compiler then you may need to explicitly supply optimization flags.
- Installation directories. JAGS conforms to the GNU standards for where files are installed. You can control the installation directories in more detail using the flags that are listed when you type `./configure --help`.

#### 1.1.1 Configuration for a 64-bit build

By default, JAGS will install all libraries into ‘`/usr/local/lib`’. If you are building a 64-bit version of JAGS, this may not be appropriate for your system. On Fedora and other RPM-based distributions, for example, 64-bit libraries should be installed in ‘`lib64`’, and on

Solaris, 64-bit libraries are in a subdirectory of 'lib' (*e.g.* 'lib/amd64' if you are using a x86-64 processor), whereas on Debian, and other Linux distributions that conform to the FHS, the correct installation directory is 'lib'.

To ensure that JAGS libraries are installed in the correct directory, you should supply the '--libdir' argument to the configure script, *e.g.*:

```
./configure --libdir=/usr/local/lib64
```

It is important to get the installation directory right when using the `rjags` interface between R and JAGS, otherwise the `rjags` package will not be able to find the JAGS library.

### 1.1.2 Configuration for a private installation

If you do not have administrative privileges, you may wish to install JAGS in your home directory. This can be done with the following configuration options

```
export JAGS_HOME=$HOME/jags #or wherever you want it
./configure --prefix=$JAGS_HOME
```

For more detailed control over the installation directories type

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

and read the section "Fine-tuning of the installation directories."

With a private installation, you need to modify your `PATH` environment variable to include '\$JAGS\_HOME/bin'. You may also need to set `LD_LIBRARY_PATH` to include '\$JAGS\_HOME/lib' (On Linux this is not necessary as the location of `libjags` and `libjrmath` is hard-coded into the JAGS binary).

## 1.2 BLAS and LAPACK

BLAS (Basic Linear Algebra System) and LAPACK (Linear Algebra Pack) are two libraries of routines for linear algebra. They are used by the multivariate functions and distributions in the `bugs` module. Most unix-like operating system vendors supply shared libraries that provide the BLAS and LAPACK functions, although the libraries may not literally be called "blas" and "lapack". During configuration, a default list of these libraries will be checked. If `configure` cannot find a suitable library, it will stop with an error message.

You may use alternative BLAS and LAPACK libraries using the configure options `--with-blas` and `--with-lapack`

```
./configure --with-blas="-lmyblas" --with-lapack="-lmylapack"
```

If the BLAS and LAPACK libraries are in a directory that is not on the default linker path, you must set the `LD_FLAGS` environment variable to point to this directory at configure time:

```
LD_FLAGS="-L/path/to/my/libs" ./configure ...
```

At runtime, if you have linked JAGS against BLAS or LAPACK in a non-standard location, you must supply this location with the environment variable `LD_LIBRARY_PATH`, *e.g.*

```
LD_LIBRARY_PATH="/path/to/my/libs:${LD_LIBRARY_PATH}"
```

Alternatively, you may hard-code the paths to the blas and lapack libraries at compile time. This is compiler and platform-specific, but is typically achieved with

```
LDFLAGS="-L/path/to/my/libs -R/path/to/my/libs
```

JAGS can also be linked to static BLAS and LAPACK if they have both been compiled with the `-fPIC` flag. You will probably need to do a custom build of BLAS and LAPACK if you require this. The configure options for JAGS are then:

```
./configure --with-blas="-L/path/to/my/libs -lmyblas -lgfortran -lquadmath" \  
            --with-lapack="-L/path/to/my/libs -lmylapack"
```

Note that with static linking you must add the dependencies of the BLAS library manually. The LAPACK library will pick up the same dependencies. Note also that `libtool` does not like linking directly to archive files. If you attempt a configuration of the form

```
--with-blas="/path/to/my/libs/myblas.a"
```

then this will pass at configure time but “make” will not correctly build the JAGS modules.

### 1.2.1 Multithreaded BLAS and LAPACK

Some high-performance computing libraries offer multi-threaded versions of the BLAS and LAPACK libraries. Although instructions for linking against some of these libraries are given below, this should not be taken as encouragement to use multithreaded BLAS. Testing shows that using multiple threads in BLAS can lead to significantly *worse* performance while using up substantially more computing resources.

## 1.3 GNU/Linux

GNU/Linux is the development platform for JAGS, and a variety of different build options have been explored, including the use of third-party compilers and linear algebra libraries.

### 1.3.1 Fortran compiler

The GNU FORTRAN compiler changed between gcc 3.x and gcc 4.x from `g77` to `gfortran`. Code produced by the two compilers is binary incompatible. If your BLAS and LAPACK libraries are linked against `libgfortran`, then they were built with `gfortran` and you must also use this to compile JAGS.

Most recent GNU/Linux distributions have moved completely to gcc 4.x. However, some older systems may have both compilers installed. Unfortunately, if `g77` is on your path then the configure script will find it first, and will attempt to use it to build JAGS. This results in a failure to recognize the installed BLAS and LAPACK libraries. In this event, set the `F77` variable at configure time.

```
F77=gfortran ./configure
```

### 1.3.2 BLAS and LAPACK

The **BLAS** and **LAPACK** libraries from Netlib (<http://www.netlib.org>) should be provided as part of your Linux distribution. If your Linux distribution splits packages into “user” and “developer” versions, then you must install the developer package (e.g. `blas-devel` and `lapack-devel`).

On **Red Hat Enterprise Linux (RHEL)** you need to activate an optional repository in order to have access to BLAS and LAPACK. The repository is called `rhel-<v>-<type>-optional-rpms`, where `<v>` is the RHEL release version and `<type>` is the type of installation (server or workstation). Find the corresponding entry in the file `/etc/yum.repos.d/redhat.repo` and change the line `enabled = 0` to `enabled = 1`.

**Suse Linux Enterprise Server (SLES)** does not include BLAS and LAPACK in the main distribution. They are included in the SLES SDK, on a set of CD/DVD images which can be downloaded from <https://download.suse.com/index.jsp> (subscription and login required).

### 1.3.3 ATLAS

On Fedora Linux, pre-compiled atlas libraries are available via the `atlas` and `atlas-devel` RPMs. These RPMs install the atlas libraries in the non-standard directory `/usr/lib/atlas` (or `/usr/lib64/atlas` for 64-bit builds) to avoid conflicts with the standard `blas` and `lapack` RPMs. To use the atlas libraries, you must supply their location using the `LD_FLAGS` variable (see section 1.2)

```
./configure LD_FLAGS="-L/usr/lib/atlas"
```

Runtime linking to the correct libraries is ensured by the automatic addition of `/usr/lib/atlas` to the linker path (see the directory `/etc/ld.so.conf.d`), so you do not need to set the environment variable `LD_LIBRARY_PATH` at run time.

### 1.3.4 AMD Core Math Library

The AMD Core Math Library (`acml`) provides optimized BLAS and LAPACK routines for AMD processors. To link JAGS with `acml`, you must supply the `acml` library as the argument to `--with-blas`. It is not necessary to set the `--with-lapack` argument as `acml` provides both sets of functions. See also section 1.2 for run-time instructions.

For example, to link to the 64-bit `acml` using gcc 4.0+:

```
LD_FLAGS="-L/opt/acml4.3.0/gfortran64/lib" \  
./configure --with-blas="-lacml -lacml_mv"
```

The `acmv_mv` library is a vectorized math library that exists only for the 64-bit version and is omitted when linking against 32-bit `acml`.

On multi-core systems, you may wish to use the threaded `acml` library (See the warning in section 1.2.1 however). To do this, link to `acml_mp` and add the compiler flag `-fopenmp`:

```
LD_FLAGS="-L/opt/acml4.3.0/gfortran64_mp/lib" \  
CXX_FLAGS="-O2 -g -fopenmp" ./configure --with-blas="-lacml_mp -lacml_mv"
```

The number of threads used by multi-threaded `acml` may be controlled with the environment variable `OMP_NUM_THREADS`.

### 1.3.5 Intel Math Kernel Library

The Intel Math Kernel library (MKL) provides optimized BLAS and LAPACK routines for Intel processors. MKL is designed to be linked to executables, not shared libraries. This means that it can only be linked to a static version of JAGS, in which the JAGS library and modules are linked into the main executable. To build a static version of JAGS, use the configure option ‘`--disable-shared`’.

MKL version 10.0 and above uses a “pure layered” model for linking. The layered model gives the user fine-grained control over four different library layers: interface, threading, computation, and run-time. Some examples of linking to MKL using this layered model are given below. These examples are for GCC compilers on `x86_64`. The choice of interface layer is important on `x86_64` since the Intel Fortran compiler returns complex values differently from the GNU Fortran compiler. You must therefore use the interface layer that matches your compiler (`mkl_intel*` or `mkl_gf*`).

For further guidance, consult the MKL Link Line advisor at <http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor>.

Recent versions of MKL include a shell script that sets up the environment variables necessary to build an application with MKL.

```
source /opt/intel/composerxe-2011/mkl/bin/mklvars.sh intel64
```

After calling this script, you can link JAGS with a sequential version of MKL as follows:

```
./configure --disable-shared \  
--with-blas="-lmkl_gf_lp64 -lmkl_sequential -lmkl_core -lpthread"
```

Note that `libpthread` is still required, even when linking to sequential MKL.

Threaded MKL may be used with:

```
./configure --disable-shared \  
--with-blas="-lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core -liomp5 -lpthread"
```

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`. (See the warning in section 1.2.1 however).

### 1.3.6 Using Intel Compilers

JAGS has been successfully built with the Intel Composer XE compilers. To set up the environment for using these compilers call the ‘`compilervars.sh`’ shell script, *e.g.*

```
source /opt/intel/composerxe-2011/bin/compilervars.sh intel64
```

Then call the configure script with the Intel compilers:

```
CC=icc CXX=icpc F77=ifort ./configure
```

### 1.3.7 Using Clang

JAGS has been built with the clang compiler for C and C++ (version 3.1). The configuration was

```
LD="llvm-ld" CC="clang" CXX="clang++" ./configure
```

In this configuration, the gfortran compiler was used for Fortran and the C++ code was linked to the GNU standard C++ library (`libstdc++`) rather than the version supplied by the LLVM project (`libc++`).

## 1.4 Solaris

JAGS has been successfully built and tested on the Intel x86 platform under Solaris 11.3 using the Oracle Developer Studio 12.6 compilers.

I experienced some difficulty with the libtool dynamic linker ltdl on Solaris. This is due to the fact that output from the solaris utility nm does not match what libtool expects. This can be overcome by exporting the environment variable NM:

```
export NM=gnm
```

to use the GNU version of nm.

The C++ library 'libCstd' is not supported. You must therefore add the option '-library=stlport4' to 'CXXFLAGS' to use the alternative STLPort4 library,

```
export LEX=flex
CC=cc CXX="CC -std=sun03" F77=f95 ./configure \
CFLAGS="-O3 -xarch=sse2" \
CXXFLAGS="+w -O3 -xarch=sse2 -library=stlport4 -lCrun"
```

or '-library=--stdc++' for the libstdc++ library.

```
./configure CC=cc CXX="CC -std=c++03" F77=f95 \
CFLAGS="-O3 -xarch=sse2" \
CXXFLAGS="+w -O3 -xarch=sse2 -library=stdc++"
```

The Sun Studio compiler is not optimized by default. Use the option '-xO3' for optimization (NB This is the letter "O" not the number 0) In order to use the optimization flag '-xO3' you must specify the architecture with the '-xarch' flag. The options above are for an Intel processor with SSE2 instructions. This must be adapted to your own platform.

To compile a 64-bit version of JAGS, add the option '-m64' to all the compiler flags. On Solaris, 64-bit files are generally installed in an architecture-specific sub-directory (e.g. 'amd64' on the x86 platform). If you wish to conform to this convention for 64-bit JAGS then you should set the configure options '--libdir', '--libexecdir', and '--bindir' appropriately.

The configure script automatically detects the Sun Performance library, which implements the BLAS/LAPACK functions.

## 2 macOS

A binary distribution of JAGS is provided for Mac OS X versions 10.9 to 10.11 and macOS 10.12 onwards, which is compatible with the current binary distribution of R and the corresponding `rjags` and `runjags` packages that are provided on CRAN. These instructions are only for those users that want to install JAGS from source.

The recommended procedure is to build JAGS using clang and the libc++ standard library, which have been the default since OS 10.9. This provides compatibility with all builds of R available on CRAN from version 3.3.0 onwards, as well as “Mavericks builds” of earlier versions of R. Users needing to build against the libstdc++ library and/or with a version of Mac OS X predating 10.9 (Mavericks) should refer to the installation instructions given in older versions of the JAGS manual.

### 2.1 Required tools

If you wish to build from a released source package i.e. ‘JAGS-4.3.1.tar.gz’, you will need to install command line compilers. The easiest way to do this is using the Terminal application from ‘/Applications/Utilities’ - opening the application gives you a terminal with a UNIX shell. Run the following command on the terminal and follow the instructions:

```
xcode-select --install
```

You will also need to install the gfortran package, which you can download from:

<https://gcc.gnu.org/wiki/GFortranBinaries#MacOS>

This setup should be sufficient to build the JAGS sources and also source packages in R. All the necessary libraries such as BLAS and LAPACK are included within macOS. Additional tools are required to run the optional test suite (see section 2.3).

### 2.2 Basic installation

#### 2.2.1 Prepare the source code

Move the downloaded ‘JAGS-4.3.1.tar.gz’ package to some suitable working space on your disk and double click the file. This will decompress the package to give a folder called ‘JAGS-4.3.1’. You now need to re-open the Terminal and change the working directory to the JAGS source code. In the Terminal window after the \$ prompt type `cd` followed by a space. In the Finder drag the ‘JAGS-4.3.1’ folder into the Terminal window and hit return. If this worked for you, typing `ls` followed by a return will list the contents of the JAGS folder.

#### 2.2.2 Set up the environment

Before configuring JAGS it is first necessary to set a linker flag to include the Accelerate framework (<https://developer.apple.com/documentation/accelerate>). This allows the JAGS installation to use Apple’s implementation of BLAS.

Copy and paste the following command into the Terminal window:

```
export LDFLAGS="-framework Accelerate"
```

Other compiler options such as optimisation flags can also be set at this stage if desired, for example:

```
export CFLAGS="-Os"  
export CXXFLAGS="-Os"  
export FFLAGS="-Os"
```

Note that JAGS is usually compiled using `-O2` by default, but it may be necessary to specify this explicitly depending on the version of the compiler being used.

### 2.2.3 Configuration

To configure the package type:

```
./configure
```

This instruction should complete without reporting an error.

### 2.2.4 Compile

To compile the code type:

```
make -j 8
```

The number ‘8’ indicates the number of parallel build threads that should be used (this will speed up the build process). In general this is best as twice the number of CPU cores in the computer - you may want to change the number in the instruction to match your machine. Again, this instruction should complete without errors.

### 2.2.5 Install

Finally to install JAGS you need to be using an account with administrator privileges. Type:

```
sudo make install
```

This will ask for your account password and install the code ready to run as described in the User Manual. You need to ensure that `/usr/local/bin` is in your `PATH` in order for the command `jags` to work from a shell prompt.

## 2.3 Running the test suite

### 2.3.1 Installing CppUnit

As of JAGS version 4, a test suite is included with the source code that can be run to ensure that the compiled code produces the expected results. To run this code on your installation, you will need to download the CppUnit framework either using Homebrew (see section 2.4.1) or from:

<http://freedesktop.org/wiki/Software/cppunit/>

For the latter, download the source code under “Release Versions” corresponding to the latest release (currently Cppunit 1.14.0), unarchive the file, and then navigate a terminal window to the working directory inside the resulting folder. Then follow the usual terminal commands (as given on the website) to install CppUnit.

### 2.3.2 Running the tests

The test suite is run following the instructions given in section 2.2.5, using the following additional command:

```
make check
```

If successful, a summary of the checks will be given. If compiler errors are encountered, you may need to add the following compiler flag (and subsequent reconfiguration) in order to force the compiler to build with C++11, as required by CppUnit 1.14.0 and later:

```
export CXXFLAGS="-std=c++11 $CXXFLAGS"
./configure
make check
```

Note that the configuration step may also need to be repeated if CppUnit was not installed the first time this was run. In this case, you may also need to clean the existing compiled code before running `make check` using:

```
make clean
```

## 2.4 Tips for developers and advanced users

### 2.4.1 Additional tools

Some additional tools are required to work with code from the JAGS repository. The easiest way of obtaining the necessary utilities is using Homebrew, which can be installed by following the instructions at <http://brew.sh>

The following instructions have been verified to work with both Mac OS X 10.9 (Mavericks) and macOS 10.12 (Sierra), and should also work with other supported versions of OS X / macOS. If problems are encountered with these instructions on OS X 10.8 or earlier, then an alternative method of installing the required tools using e.g. MacPorts (as given in version 4.1.0 of the JAGS manual) may be more successful.

### 2.4.2 Working with the development code

If you want to work on code from the JAGS repository, you will need to build and install the auxillary GNU tools (autoconf, automake and libtool), as well as mercurial, bison, and flex as follows:

```
brew install mercurial
brew install autoconf
brew install automake
brew install libtool
brew install pkg-config
brew install bison
brew install flex
```

Note that CppUnit can also be installed using the same method:

```
brew install cppunit
```

The following sequence should then retrieve a clone of the current development branch of JAGS, and prepare the source code:

```
hg clone -r release-4_patched http://hg.code.sf.net/p/mcmc-jags/code-0
cd code-0
autoreconf -fis
```

The following modification to the `PATH` is also currently required to find the Homebrew versions of bison and flex:

```
export PATH="/usr/local/opt/bison/bin:/usr/local/opt/flex/bin:$PATH"
```

For more information see `brew info bison` and `brew info flex`

Once these commands have been run successfully, you should be able to follow the configuration and installation instructions from section 2.2.2 onwards.

### 2.4.3 Using ATLAS

Rather than using the versions of BLAS and LAPACK provided within OS X, it is possible to use ATLAS, which is available from <http://math-atlas.sourceforge.net>. This can be either be installed by following the instructions given at [http://math-atlas.sourceforge.net/atlas\\_install/](http://math-atlas.sourceforge.net/atlas_install/), or by using MacPorts (<https://www.macports.org/>) with the following Terminal command:

```
sudo port install atlas
```

Once ATLAS is successfully installed, the `-framework Accelerate` flag can be omitted from the instructions given in section 2.2.2.

## 3 Windows

These instructions use MSYS (Minimal System), which is a software development environment to help build and deploy Unix code on Windows. You need some familiarity with Unix in order to follow the build instructions but, once built, JAGS can be installed on any PC running windows, where it can be run from the Windows command prompt.

### 3.1 Preparing the build environment

You need to install the following packages

- The Rtools42 compiler suite for Windows
- NSIS, including the AccessControl plug-in

#### 3.1.1 Rtools

Rtools is a set of compilers and utilities used for compiling R on Windows. Rtools can be downloaded from your nearest CRAN mirror (<https://cran.r-project.org/bin/windows/Rtools/>). We have to use the same compiler for R and JAGS to maintain binary compatibility. For R 4.2.0 and above, the Rtools42 toolchain is used.

Rtools42 comes with its own version of MSYS, a minimal unix shell that we will use to build JAGS. At the time of writing, the MSYS shell does not include the Rtools42 compiler tools on the PATH, so this must be added manually with

```
export PATH=/x86_64-w64-mingw32.static.posix/bin:$PATH
```

#### 3.1.2 NSIS

The Nullsoft Scriptable Install System (<https://nsis.sourceforge.io>) allows you to create a self-extracting executable that installs JAGS on the target PC. These instructions were tested with NSIS 3.08. You must also install the AccessControl plug-in for NSIS, which is available from [http://nsis.sourceforge.io/AccessControl\\_plug-in](http://nsis.sourceforge.io/AccessControl_plug-in). The plug-in is distributed as a ZIP file which is unpacked into the installation directory of NSIS.

At the time of writing, the ZIP file for the AccessControl plug-in is not correctly configured. Within the Plugins sub-folder of the NSIS installation directory, you will need to copy `i386-ansi/AccessControl.dll` into `x86-ansi` and `i386-unicode/AccessControl.dll` into `x86-unicode`.

## 3.2 Building LAPACK

Rtools42 contains static libraries for BLAS and LAPACK. In order to use these with JAGS we need to build dynamic libraries for them.

First identify the location of the static libraries

```
TLIB='which gcc | sed -e 's!bin/gcc$!!g'"lib"
```

Then create a directory where we will build the shared BLAS and LAPACK libraries

```
mkdir sharedlb
cd sharedlb
SHAREDLB='pwd'
```

Create a definition file 'libblas.def' that exports all the symbols from the BLAS library

```
dlltool -z libblas.def --export-all-symbols $TLIB/libblas.a
```

Then link this with the static library to create a DLL ('libblas.dll') and an import library ('libblas.dll.a')

```
gfortran -shared -o libblas.dll -Wl,--out-implib=libblas.dll.a \
libblas.def $TLIB/libblas.a
```

Repeat the same steps for the LAPACK library, creating an import library ('liblapack.dll.a') and DLL ('liblapack.dll')

```
dlltool -z liblapack.def --export-all-symbols $TLIB/liblapack.a
gfortran -shared -o liblapack.dll -Wl,--out-implib=liblapack.dll.a \
liblapack.def $TLIB/liblapack.a -L. -lblas
```

### 3.3 Compiling JAGS

Unpack the JAGS source

```
tar xfvz JAGS-4.3.1.tar.gz
cd JAGS-4.3.1
```

and configure JAGS

```
./configure --host=x86_64-w64-mingw32.static.posix \
--with-blas="-L$SHAREDLB -lblas" --with-lapack="-L$SHAREDLB -llapack"
```

After the configure step, type

```
make win64-install
```

This will install JAGS into the subdirectory 'win/inst64'. Note that you must go straight from the configure step to `make win64-install` without the usual step of typing `make` on its own. The `win64-install` target resets the installation prefix, and this will cause an error if the source is already compiled.

You can now create the installer. Normally you will want to distribute the BLAS and LAPACK libraries with JAGS. In this case, put the DLLs and import libraries in the subdirectory 'win/runtime64'. They will be detected and included with the distribution.

Make sure that the file 'makensis.exe', provided by NSIS, is in your PATH. For a typical installation of NSIS, on 64-bit windows:

```
PATH=$PATH:/c/Program\ Files\ \((x86\))/NSIS
```

Then type

```
make installer
```

After the build process finishes, the self extracting archive will be in the subdirectory 'win'.

### 3.4 Running the unit tests

In order to run the unit tests on Windows you must first install the cppunit library from source. Download the file ‘cppunit-1.12.1.tar.gz’ from Sourceforge (<http://sourceforge.net/projects/cppunit/files/cppunit/1.12.1/>) and unpack it:

```
tar xfvz cppunit-1.12.1.tar.gz
cd cppunit-1.12.1
```

Then compile and install as follows:

```
./configure --prefix=/opt --build=x86_64-w64-mingw32.static.posix --disable-shared
make
make install
```

The configure option `--disable-shared` prevents the creation of the DLL ‘libcppunit.dll’ and builds only the static library ‘libcppunit.a’. Without this option, the unit tests will fail. One of the major limitations of static linking to the C++ runtime is that you cannot throw exceptions across a DLL boundary. Linking the test programs against ‘libcppunit.dll’ will result in uncaught exceptions and apparent failures for some of the tests, so it must be disabled.

JAGS relies on `pkg-config` to detect `libcppunit`, but `pkg-config` is not installed automatically with Rtools42. You can install it with

```
pacman -Sy pkg-config
```

Before configuring JAGS, set the environment variable `PKG_CONFIG_PATH` so that `pkg-config` can find the configuration information for `libcppunit`.<sup>1</sup>

```
PKG_CONFIG_PATH=/opt/lib/pkgconfig
```

Then run `make check` after `make win64-install`.

### 3.5 Installing rjags

To install the `rjags` package from source you need to let R know where JAGS is installed. This is done via the `JAGS_ROOT` make variable, which is set in the file `.R/Makevars.win`, e.g.

```
JAGS_ROOT=c:\Program~1\JAGS\JAGS-4.3.1
```

Note that this is a Make variable, not an environment variable. Setting `JAGS_ROOT` as an environment variable will have no effect.

---

<sup>1</sup>You may wonder why we are using a non-standard prefix instead of installing `libcppunit` into the default prefix `/usr`. At the time of writing, `pkg-config` does not play nicely with the compiler toolchain. The toolchain does not look for headers in `/usr/include` and `pkg-config` will not add this directory to the header search path because it assumes that it is already there by default. Hence compilation of the unit tests will fail with an error message about missing headers.

### 3.6 Running the examples

The BUGS classic examples (file ‘classic-bugs.tar.gz’ from the JAGS Sourceforge site) can be run from the MSYS shell using the `make` command provided by Rtools. You must have R installed, along with the packages `rjags` and `coda`, both of which are available from CRAN ([cran.r-project.org](http://cran.r-project.org)).

It is necessary to add R to the search path.

```
PATH=$PATH:/c/Program\ Files/R/R-4.2.0/bin
```

where ‘username’ is your Windows user name. Then

```
tar xfvz classic-bugs.tar.gz
cd classic-bugs
cd vol1
make Rcheck
```

will check all examples in volume 1 using the `rjags` package. Repeat for ‘vol2’ to complete the checks.