JAGS Version 2.0.0 installation manual

Martyn Plummer Bill Northcott

April 30, 2010

JAGS is distributed in binary format for Microsoft Windows, Mac OS X, 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, Mac OS X, 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
```

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. Although JAGS contains no Fortran code, you are required to define a Fortran compiler so that JAGS modules can be linked against libraries written in Fortran (such as BLAS and LAPACK)
- Optimization flags for the C and C++ compilers. JAGS is optimized by default if the GNU compiler (gcc) is used. Otherwise you must 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 `--libidr` 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=$HOME/jags #or wherever you want it
./configure --bindir=$JAGS_HOME/bin --libdir=$JAGS_HOME/lib \
 --libexecdir=$JAGS_HOME/bin --includedir=$JAGS_HOME/include
```

You then 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 `LDFLAGS` environment variable to point to this directory at configure time:

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

If your BLAS and LAPACK libraries depend on other libraries that are not on the linker path, you must supply these dependency libraries as additional arguments to `--with-blas` and `--with-lapack`

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
```

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`).

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 the Novell web site. See `http://developer.novell.com/wiki/index.php/SLES_SDK` for more information.

It is quite common for the Netlib implementations of BLAS and LAPACK to break when they are compiled with the latest GNU compilers. Linux distributions that use "bleeding edge" development tools – such as **Fedora** – may ship with a broken version of BLAS and LAPACK. Normally, this problem is quickly identified and fixed. However, you need to take care to use the online updates of the BLAS and LAPACK packages from your Linux Distributor, and not rely on the version that came on the installation disk.

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 `LDFLAGS` variable (see section 1.2)

```
./configure LDFLAGS="-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 file `/etc/ld.so.conf`), 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, *and its depen-*

dencies, as arguments 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+:

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

The library `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. To do this, link to `acml_mp` and add the compiler flag `-fopenmp`:

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

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

For older Linux systems, the last version that supports gcc 3.4 is `acml` 3.6.0. When using gcc 3.4, link against `libg2c`.

```
LDFLAGS="-L/opt/acml3.6.0/gnu64/lib" \
./configure --with-blas="-lacml -lacml_mv -lg2c"
```

1.3.5 Intel Math Kernel Library

The Intel Math Kernel library (MKL) provides optimized BLAS and LAPACK routines for Intel processors. The instructions below are for MKL version 10.0 and above which use 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 library support. 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*`).

I have not been able to link JAGS with MKL using GNU compilers, except which building a static version. To build a static version of JAGS, use the configure option `--disable-shared`. The JAGS library and modules will be linked into the main executable.

JAGS can be linked to a sequential version of MKL by

```
MKL_HOME=/opt/intel/mkl/10.0.3.020/
MKL_LIB_PATH=${MKL_HOME}/lib/em64t/
./configure --disable-shared \
            --with-blas="-L${MKL_LIB_PATH} -lmkl_gf_lp64 -lmkl_sequential -lmkl_core"
```

Threaded MKL may be used with:

```
./configure --disable-shared\
   --with-blas="-L${MKL_LIB_PATH} -lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core -liomp5 -lpthr
```

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`.

1.3.6 Using Intel Compilers

JAGS has been successfully built with the Intel C, C++ and Fortran compilers. The additional configure options required to use the Intel compilers are:

```
source /opt/intel/Compiler/11.1/bin/ifortvars.sh
source /opt/intel/Compiler/11.1/bin/iccvars.sh
CC=icc CXX=icpc F77=ifort ./configure
```

1.4 OpenSolaris

JAGS has been successfully built and tested on the Intel x86 platform under OpenSolaris 2008.05 using the Sun Studio Express 5/08 compilers.

```
./configure CC=cc CXX=CC F77=f95 \
CFLAGS="-xO3 -xarch=sse2" \
FFLAGS="-xO3 -xarch=sse2" \
CXXFLAGS="-xO3 -xarch=sse2"
```

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.

Solaris provides two versions of the C++ standard library: `libCstd`, which is the default, and `libstlport4`, which conforms more closely to the C++ standard. JAGS may be linked to the stlport4 library by adding the options `-library=stlport4 -lCrun` to `CXXFLAGS`.

The configure script automatically detects the Sun Performance library, which implements the BLAS/LAPACK functions. Automatic detection may not work on older versions of Sun Studio, which used a different syntax for specifying this library. In this case, you may need to use the configure option

```
--with-blas="-xlic_lib=sunperf -lsunmath"
```

1.4.1 Using acml

AMD provides a version of their Core Math Library (acml) for Solaris. To use this library instead of the Sun Performance library add the following configure options (changing paths as appropriate):

```
--with-blas="-lacml -lacml_mv -lfsu" \
LDFLAGS="-L/opt/acml4.1.0/sun64/lib \
-R/opt/acml4.1.0/sun64/lib:/opt/SunStudioExpress/lib"
```

The acml library is only available in 64-bit mode, so the option `-m64` must also be added to all the compiler flags.

As with using acml on Linux (section 1.3.4), the configure option `--with-blas` must include not only the acml library, but also its dependencies. The `LDFLAGS` option `-R` hard-codes the paths to these libraries into the JAGS modules that require them.

1.5 IRIX

JAGS has not been tested on IRIX for some time. Version 1.0.0 was successfully built using the MIPSpro 7.4 compiler on IRIX 6.5. The following configure options were used:

```
./configure CC=cc CXX=CC F77=f77 \
CFLAGS="-O2 -g2 -OPT:IEEE_NaN_inf=ON" \
CXXFLAGS="-O2 -g2 -OPT:IEEE_NaN_inf=ON"
```

and JAGS was built with `gmake` (GNU make).

BLAS and LAPACK functions on IRIX are provided by the Scientific Computing Software library (`scs`). The presence of this library is detected automatically by the configure script.

When using the MIPSpro compiler, optimization flags must be given explicitly at configure time. If this is not done, then JAGS will not be optimized at all and will run slowly.

2 Mac OS X

If trying to build software on Mac OS X you really need to use Leopard (10.5.x) or Snow Leopard (10.6.x). Unless otherwise stated these instruction assume Snow Leopard (10.6.x). The open source support has improved greatly in recent releases. You also need the latest version of Apple's Xcode development tools. The current version is Xcode 3.2.x (Leopard uses 3.1.x). Early versions have serious bugs which affect R and JAGS. Xcode is available as a free download from `http://developer.apple.com`. You need to set up a free login to ADC. The Apple developer tools do not include a Fortran compiler. Without Fortran, you will not be able to build JAGS.

For instructions for building on Tiger or for older versions of R see previous versions of this manual.

The GNU gfortran Fortran compiler is included in the R binary distribution available on CRAN. Install the R binary and select all the optional components in the 'Customize' step of the installer. These instructions assume R-2.7.x.

The default C/C++ compiler for Snow Leopard is gcc-4.2.x. Xcode 3.2 also includes gcc-4.2 and llvm-gcc4.2. The code has been successfully built with these optional compilers but will only run on Leopard. llvm is being actively developed by Apple and may produce better code.

MacOS X 10.2 and onwards include optimised versions of the BLAS and LAPACK libraries. So no extra libraries are is needed for Snow Leopard. Optimisation continues andApple are working on using GPUs for this sort of math. Make sure your OS is up to date.

To ensure the JAGS configure script can find the Fortran compiler for a bash shell

```
export F77=/usr/local/bin/gfortran
```

On 64 bit hardware, which means most recent Macs, there may be a problem with the Fortran compiler. Apple's compilers default to 64 bit builds on 64 bit hardware but the Fortran binaries available default to 32 bit builds. This means you need to add compile and link options.

For instance on 64 bit Intel Macs type

```
export CFLAGS='-arch x86_64'
export CXXFLAGS='-arch x86_64'
export FFLAGS='-arch x86_64'
export LDFLAGS='-arch x86_64'
```

Some Fortran compilers (not the ones from CRAN) do not understand the -arch option. For these you will need something like:

```
export CFLAGS='-arch x86_64'
export CXXFLAGS='-arch x86_64'
export FFLAGS='-m64'
export LDFLAGS='-arch x86_64'
```

To build JAGS unpack the source code and cd into the source directory. Type the following:

```
./configure
make
```

(if you have multiple CPUs try `make -j 4` or `make -j 8`. It may need to be issued more than once)

```
sudo make install
```

You need to ensure `/usr/local/bin` is in your PATH in order for 'jags' to work from a shell prompt.

This will build the default architecture for you Mac: ppc on a G4 or G5 and i386 or `x86_64` on an Intel Mac. If you want to build multiple architecture fat binaries, you will need to ensure that libtool in the JAGS sources is version 1.5.24 or later. Then you can use configure commands like

```
CXXFLAGS="-arch i386 -arch x86_64" ./configure
```

Make will then build fat binaries. See the R Mac developers page `http://r.research.att.com/` for instructions to build fat R packages.

A final note on MacOS X builds: do NOT use `-O3`. It is not optimal and may find compiler bugs. Apple recommends `-Os`.

3 Windows

These instructions use MinGW, The Minimalist GNU system for 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

- MinGW
- MSYS
- NSIS

MinGW (Minimalist GNU for Windows) is a build environment for Windows. There is an official release from `http://www.mingw.org`. However, we used the MinGW distribution that comes as part of the R tools for windows (`http://www.murdoch-sutherland.com/Rtools`), since the compilers in this distribution match the compilers used to build the binary distribution of R for windows, as well as the R packages distributed via CRAN (`http://cran.r-project.org`).

We used version 2.11 of `Rtools`. You only need to install the "MingGW components and tools", not the other components of `Rtools`. The installer will also ask if you wish to modify the Windows PATH. You do not need this.

MSYS (the Minimal SYStem) is part of the MinGW project. It provides a bash shell for you to build Unix software. These instructions were tested with MSYS 1.0.10, the last version of MSYS to be bundled with a Windows installer. The installer can be downloaded from `http://sourceforge.net/projects/mingw/files`. At the end of the installation process, it will launch a post-install script that will allow you to use MSYS in conjunction with MinGW.

MSYS creates a home directory for you in `c:\msys\<version>\home\<username>`, where `<version>` is the version of MSYS and `<username>` is your user name under Windows. You will need to copy and paste the source files for LAPACK and JAGS into this directory.

The Nullsoft Scriptable Install System (`http:\\nsis.sourceforge.net`) allows you to create a self-extracting executable that installs JAGS on the target PC. These instructions were tested with NSIS 2.33.

3.1.1 Building LAPACK

Download the LAPACK source file from `http://www.netlib.org/lapack`. We used version 3.2.1, which is packaged as `lapack.tgz`. Unpack the file in your home directory.

```
tar xfvz lapack.tgz
cd lapack-3.2.1
```

Copy the file `INSTALL/make.inc.gfortran` to `make.inc` in the top level source directory. Then edit `make.inc`, replacing the line

```
PLAT = _LINUX
```

with something more sensible, like

```
PLAT = _MinGW
```

Edit the file `Makefile` so that it builds the BLAS library. The line that starts `lib:` should read

```
lib: blaslib lapacklib tmglib
```

Type

```
make
```

The compilation process is slow. Eventually, it will create two static libraries `blas_MinGW.a` and `lapack_MingGW.a`. These are insufficient for building JAGS: you need to create dynamic link library (DLL) for each one.

First create a definition file `libblas.def` that exports all the symbols from the BLAS library

```
dlltool -z libblas.def --export-all-symbols blas_MinGW.a
```

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

```
gcc -shared -o libblas.dll -Wl,--out-implib=libblas.dll.a \
libblas.def blas_MinGW.a -lgfortran
```

(If using gcc 3.4, the library should be linked with `-lg2c` instead of `-lgfortran`)

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 lapack_MinGW.a
gcc -shared -o liblapack.dll -Wl,--out-implib=liblapack.dll.a \
liblapack.def lapack_MinGW.a  -L./ -lblas -lgfortran
```

3.2 Compiling JAGS

Unpack the JAGS source

```
tar xfvz JAGS-2.0.0.tar.gz
cd JAGS-2.0.0
```

and configure JAGS

```
./configure LDFLAGS="-L/path/to/import/libs/"
```

where `/path/to/import/libs` is a directory that contains the import libraries (`libblas.dll.a` and `liblapack.dll.a`). This must be an *absolute* path name, and not relative to the JAGS build directory.

Normally you will want to distribute the blas and lapack libraries with JAGS. In this case, put the DLLs and import libraries in the sub-directory `win32/include`. 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:

```
PATH=$PATH:/c/Program\ files/NSIS
```

Then type

```
make win32-installer
```

The self extracting archive will be in the subdirectory `win`.

Note that you must go straight from the configure step to `make win32-installer` without the usual step of typing `make` on its own. The `win32-installer` target resets the installation prefix, and this will cause an error if the source is already compiled.

3.3 Building on 64-bit windows

The build instructions for 64-bit windows are similar to the instructions for 32-bit windows. Instead of the MinGW tools that come with R, we use a 64-bit compiler from the MinGW-w64 project (`http://www.sourceforge.net/projects/mingw-w64`). The distribution used to build JAGS-2.0.0 was `mingw-w64-1.0-bin_i686-mingw_20100322.zip`, which is also the compiler used to build the 64-bit Windows binary distribution of R-2.11.0. This is actually a cross-compiler. It will run under MSYS –which is still 32-bit – but will produce 64-bit code.

Most binary distributions of MinGW-w64 use dynamic linking to runtime libraries. This is inconvenient, as it means that the required runtime libraries must be distributed with JAGS. Failure to do so will either result in an installation that is either completely non-functional or will crash. To force the compiler to use static linking, delete any import libraries (files ending in `.dll.a`). In the case of the `mingw-w64-1.0-bin_i686-mingw_20100322.zip` distribution, we had to remove `libssp.dll.a`. Note that there is currently no way to avoid dynamic linking to the gcc runtime library (see below).

In order to build the BLAS and LAPACK libraries, you need to append the prefix `x86_64-w64-mingw32-` to all the tools. This means you need to edit the entries FORTRAN, LOADER, ARCH, and RANLIB in the Makefile, as well as modifying the name of `dlltool` and `gcc` when building the DLLs for BLAS and LAPACK.[1]

To build JAGS you do not need to edit any makefiles. Just add the configure option

```
--host=x86_64-w64-mingw32
```

to use the cross-compiler. You may also wish to add the linker flag

```
LDFLAGS=-Wl,--enable-auto-import
```

in order to suppress some warnings from the linker, although these seem to be harmless.

The JAGS binary built in this way will be dynamically linked to the C runtime library `libgcc_s_sjlj-1.dll`, which is part of the MinGW-w64 distribution. You will find it in the `bin` directory. You should put a copy of this DLL in the `win/runtime` directory so that it is included in the installer.

Finally, in order to build the NSIS installer, you should use the make target `win64-installer` instead of `win32-installer`.

[1]Although we are using a cross-compiler, you cannot cross-build 64-bit BLAS and LAPACK on a 32-bit Windows system. This is because the build process includes compilation of test programs which must be run.