

JAGS Version 0.75 manual

Martyn Plummer
International Agency for Research on Cancer

November 4, 2004

1 Introduction

JAGS is Just Another Gibbs Sampler. It is a program for analysis of Bayesian hierarchical models using Gibbs sampling that aims for the same functionality as classic BUGS (<http://www.mrc-bsu.cam.ac.uk>). If you want to understand what JAGS does, you need to be familiar with BUGS. For most questions, you should therefore consult the WinBUGS manual. This document is a short description of the differences between WinBUGS and JAGS to allow you to get started.

JAGS was written with three aims in mind: to have a BUGS engine that runs on Unix; to be extensible, allowing users to write their own functions and distributions; and to be a platform for experimentation with ideas in Bayesian modelling. To this last end, JAGS is licensed under the GNU General Public License. You may freely modify and redistribute it under certain conditions (see the file COPYING for details).

JAGS is designed to work closely with the R language and environment for statistical computation and graphics (<http://www.r-project.org>). In particular, you will need R to prepare the input data for JAGS, and you will find it useful to install the CODA package for R to analyze the output.

2 Running JAGS

JAGS has a command line interface. To invoke jags interactively, simply type `jags` at the shell prompt on Unix, or the Windows command prompt on Windows. To invoke JAGS with a script file, type `jags <script file>`

Output from JAGS is printed to the standard output, even when a script file is being used. The output is currently quite verbose. It will be cut down in a future release when JAGS is more stable. The JAGS interface is designed to be forgiving. It will print a warning message if you make a mistake, but otherwise try to keep going. This is useful when you make a syntax error after a long run.

3 Scripting commands

JAGS has a simple set of scripting commands with a syntax loosely based on Stata. Commands are shown below preceded by a dot (.). This is the JAGS prompt. Do not type the dot in when you are entering the commands.

C-style block comments taking the form `/* ... */` can be embedded anywhere in the script file. Additionally, you may use R-style single-line comments starting with `#`.

3.1 SEED statement

```
. seed <n>
```

Sets the random seed of the random number generator (RNG) provided by the Rmath library. The RNG is Marsaglia-Multicarry, which requires two seeds. For user convenience, these will be generated from the single seed supplied by the user. The seed generation algorithm is copied from R.

This statement is optional. Without it, the random seed is determined by the time at startup. The starting seed is printed out so that the run can be reproduced, if necessary.

3.2 MODEL IN statement

```
. model in <file>
```

Checks the syntactic correctness of the model description in `<file>` and reads it into memory. The next compilation statement will compile this model. The file name may be optionally enclosed in quotes, and this is necessary if the name contains spaces, or any character other than alphanumeric characters, and `'_'`, `'-'`, `'.'`, `'/'`, or `'\'`.

The model file should contain a BUGS language description of the model in the form used by WinBUGS.

You may, if you wish, include variable declarations as a comma-separated list in the style of classic BUGS, but these are optional. Here is the standard linear regression example:

```
var x[N], Y[N], mu[N], alpha, beta, tau, sigma, x.bar;
model {
  for (i in 1:N) {
    mu[i] <- alpha + beta*(x[i] - x.bar);
    Y[i] ~ dnorm(mu[i],tau);
  }
  x.bar <- mean(x[]);
  alpha ~ dnorm(0.0,1.0E-4);
  beta ~ dnorm(0.0,1.0E-4);
  tau ~ dgamma(1.0E-3,1.0E-3);
  sigma <- 1.0/sqrt(tau);
}
```

The semi-colons are optional.

3.3 DATA IN statement

```
. data in <file>
```

JAGS keeps an internal data table containing the values of observed nodes. The DATA IN statement reads data from a file into this data table.

The data must be in the form produced by the `dump()` command in the R language. This is unlike the `dput` format used by BUGS. It consists of a series of R assignment statements. For example, here are the data for the line example:

```
"x" <-
c(1, 2, 3, 4, 5)
#R-style comments, like this one, can be embedded in the data file
"Y" <-
c(1, 3, 3, 3, 5)
"N" <-
5
```

N.B. Unlike WinBUGS, there is no need to transpose matrices in R before dumping their values.

If you have an S-style data file for WinBUGS and you wish to convert it for JAGS, then use the command `bugs2jags`, which is supplied with CODA version 0.6 or above.

Several data statements may be used to read in data from more than one file. If two data files contain data for the same variable, the second set of values will overwrite the first, and a warning will be printed.

You may not supply the values of logical nodes in the data file. Only stochastic nodes and constant nodes are allowed.

See also: DATA TO (3.12).

3.4 COMPILE statement

```
. compile [, nchains(<n>)]
```

Compiles the model using the information provided in the preceding model and data statements. By default, a single Markov chain is created for the model, but if the `nchains` option is given, then `<n>` chains are created (**NB JAGS only currently supports one chain**)

Following the compilation of the model, further data statements are legal, but have no effect. A new model statement, on the other hand, will replace the current model.

3.5 PARAMETERS IN statement

```
. parameters in <file> [, chain(<n>)]
```

Reads the data values in `<file>` and writes them to the corresponding **unobserved** variables in chain `<n>`. The file has the same format as the data file. The `chain` option may be omitted, in which case $n = 1$ is assumed. (**NB JAGS only supports one chain**)

The PARAMETERS IN statement may be used at any time to overwrite the current parameter values.

You may only supply the values of stochastic nodes in the parameters file. Logical nodes and constant nodes are forbidden.

See also: PARAMETERS TO (3.13)

3.6 INITS statement

```
. inits in <file> [, chain(<n>)]
```

This is a synonym for PARAMETERS IN. It is kept for compatibility with existing script files, but is deprecated. Please use PARAMETERS IN instead.

3.7 INITIALIZE statement

```
. initialize
```

Initializes the model using the data or parameter values supplied for each chain.

3.8 UPDATE statement

```
. update <n> [,by(<m>)]
```

Updates the model by `<n>` iterations. A progress bar is printed on the standard output consisting of 40 asterisks. If the `by` option is supplied, a new asterisk is printed every `<m>` iterations. If this entails more than 40 asterisks, the progress bar will be wrapped over several lines. At the end of each line, the percentage of the total run completed is printed. If `pmc` is zero, the printing of the progress bar is suppressed.

3.9 MONITOR statement

```
. monitor set <varname> [, thin(n)]
```

Sets a monitor for variable `<varname>`. The `thin` option (which may be omitted) sets the thinning interval of the monitor so that it will only record every n th value.

```
. monitor clear <varname>
```

Clears the monitor associated with variable `<varname>`

3.10 CODA statement

```
. coda <varname> [, stem(<filename>)]
```

Dumps the monitored values of variable `<varname>` to files `jags.ind` and `jags.out` in a form that can be read by the CODA package of R. The file name stem may be changed from `jags` to another value by using the `stem()` option. The wild-card character “*” may be used to dump all monitored nodes

3.11 EXIT statement

```
. exit
```

Exits JAGS. JAGS will also exit when it reads an end-of-file character. Note that although JAGS behaves in many ways like classic BUGS, it will not automatically dump the contents of the monitors on exit. You must use the coda statement before exiting.

3.12 DATA TO statement

```
. data to <filename>
```

Writes the data (*i.e.* the values of the observed nodes) to a file in the R dump format. The same file can be used in a DATA IN statement for a subsequent model.

See also: DATA IN (3.3)

3.13 PARAMETERS TO statement

```
. parameters to <file> [, chain(<n>)]
```

Writes the current parameter values (*i.e.* the values of the unobserved stochastic nodes) in chain `<n>` to a file in R dump format. The same file can be used as input in a PARAMETERS IN statement.

See also: PARAMETERS IN (3.5)

4 Errors

There are two kinds of errors in JAGS: runtime errors, which are due to mistakes in the model specification, and logic errors which are internal errors in the JAGS program.

Logic errors are generally created in the lower-level parts of the JAGS library, where it is not possible to give an informative error message. The upper layers of the JAGS program are supposed to catch such errors before they occur, and return a useful error message that will help you diagnose the problem. Inevitably, some errors slip through. Hence, if you get a logic error, there is probably an error your input to JAGS, although it may not be obvious what it is. Please send a bug report (see “Feedback” below) whenever you get a logic error.

Error messages may also be generated when parsing files (model files, data files, command files). The error messages generated in this case are created automatically by the program `yacc`. They generally take the form “syntax error, unexpected FOO, expecting BAR” and are not always abundantly clear.

5 Differences between JAGS and WinBUGS

Although JAGS aims for the same functionality as BUGS, a number of important differences remain. Most of these differences are limitations of JAGS. In particular, JAGS aims to reproduce the functionality of classic BUGS and until recently, had exactly the same interface. Many improvements implemented in WinBUGS are not yet available in JAGS. For example, JAGS cannot simultaneously run two or more chains.

5.1 Variable declarations

JAGS allows the dimensions of the variables to be defined in the model file. The declarations are modelled on classic BUGS, and consist of the keyword `var` followed by a comma-separated list of variable names, with their dimensions in square brackets. The dimensions may be given in terms of any constant expression, *i.e.* a scalar expression using the operators `'*'`, `'/'`, `'+'`, `'-'` and any scalar data values.

As of JAGS version 0.75, **variable declarations are optional**. If a variable is not declared then JAGS has two methods of determining its size.

1. **Using the data table.** If data values are supplied in the data table (*via* DATA IN statements before compilation) then the dimension of an undeclared variable is inferred from this. Note that you can define the dimensions of *unobserved* variables this way by supplying a vector, matrix or array consisting entirely of missing values.
2. **Using the left hand side of the relations.** The maximal index values on the left hand side of a relation are taken to be the dimensions of the nodes. As with WinBUGS, empty indices are not allowed using this method since, for example, it is impossible to calculate the dimension of a node `X[1,]`. If indices are omitted entirely, the variable is taken to be a scalar.

5.2 Samplers

JAGS has a more limited set of samplers than WinBUGS. The basic workhorse is a univariate slice sampler.

5.3 Functions

The `max` and `min` functions are more general, and behave exactly like the corresponding functions in R. They take a variable number of arguments which may be scalar or vector-valued. The return value is the maximum/minimum value over all supplied arguments.

5.4 Distributions

Structural zeros are allowed in the Dirichlet distribution. If

```
p ~ ddirch(alpha)
```

and some of the elements of `alpha` are zero, then the corresponding elements of `p` will be fixed to zero.

5.5 Decreasing for loops

Versions of JAGS before 0.75 reversed any `for` loop with decreasing indices. JAGS now behaves like BUGS in ignoring such loops.

5.6 Data transformations

As of version 0.75, JAGS allows data transformations, but the syntax is different from BUGS. BUGS allows you to put a stochastic node twice on the left hand side of a relation, as in this example taken from the manual

```
for (i in 1:N) {
  z[i] <- sqrt(y[i])
  z[i] ~ dnorm(mu, tau)
}
```

This is forbidden in JAGS. You must put data transformations in a separate block of relations preceded by the keyword `data`:

```
data {
  for (i in 1:N) {
    z[i] <- sqrt(y[i])
  }
}
model {
  for (i in 1:N) {
    z[i] ~ dnorm(mu, tau)
  }
  ...
}
```

This syntax preserves the declarative nature of the BUGS language. In effect, the `data` block defines a distinct model, which describes how the data is generated. Each node in this model is forward-sampled once, and then the node values are read back into the data table. The `data` block is not limited to logical relations, but may also include stochastic relations. You may therefore use it to generate data from a stochastic model different from the one used to analyse the data in the `model` statement.

This example shows a simple location-scale problem in which the “true” values of the parameters `mu` and `tau` are generated from a given prior in the `data` block, and the generated data is analyzed in the `block`.

```
data {
  for (i in 1:N) {
    y[i] ~ dnorm(mu, tau)
  }
  mu.true ~ dnorm(0,1);
  tau.true ~ dgamma(1,3);
}
model {
  for (i in 1:N) {
    y[i] ~ dnorm(mu, tau)
  }
  mu ~ dnorm(0, 1.0E-3)
  tau ~ dgamma(1.0E-3, 1.0E-3)
}
```

Beware, however, that every node in the `data` statement will be considered as data in the subsequent `model` statement. This example, although superficially similar, has a quite different interpretation.

```
data {
  for (i in 1:N) {
    y[i] ~ dnorm(mu, tau)
  }
  mu ~ dnorm(0,1);
  tau ~ dgamma(1,3);
}
model {
  for (i in 1:N) {
    y[i] ~ dnorm(mu, tau)
  }
  mu ~ dnorm(0, 1.0E-3)
```

```

    tau ~ dgamma(1.0E-3, 1.0E-3)
}

```

Since the names `mu` and `tau` are used in both `data` and `model` blocks, these nodes will be considered as *observed* in the model and their values will be fixed at those values generated in the `data` block.

5.7 Directed cycles

Directed cycles are forbidden in JAGS. There are two important instances where directed cycles are used in BUGS.

- Defining autoregressive priors
- Defining ordered priors

For the first case, the `GeoBUGS` extension to `WinBUGS` provides some convenient ways of defining autoregressive priors. These should be available in a future version of JAGS.

5.8 Censoring, truncation and prior ordering

These are three, closely related issues that are all handled using the `I(,)` construct in BUGS.

Censoring occurs when a variable X is not observed directly, but is observed only to lie in the range $(L, U]$. Censoring is an *a posteriori* restriction of the data, and is represented in `WinBUGS` by the `I(,)` construct, *e.g.*

```
X ~ dnorm(theta, tau) I(L,U)
```

where L and U are constant nodes.

Truncation occurs when a variable is known *a priori* to lie in a certain range. Under some circumstances, you can also represent truncated nodes with the `I(,)` construct. This works when L, U are constant nodes (You will note that this is syntactically the same as censoring in the BUGS language, even though the underlying concept is quite different).

Prior ordering occurs when a vector of nodes is known *a priori* to be strictly increasing or decreasing. It can be represented in `WinBUGS` with symmetric `I(,)` constructs, *e.g.*

```
X[1] ~ dnorm(0, 1.0E-3) I(,X[2])
X[2] ~ dnorm(0, 1.0E-3) I(X[1],)
```

ensures that $X[1] \leq X[2]$.

JAGS makes an attempt to separate these three concepts.

Censoring is handled in JAGS using the new distribution `dinterval` (interval censored distribution). This has two parameters: t the original continuous variable, and `c[]`, a vector of cut points of length M , say. If $\mathbf{X} \sim \text{dinterval}(\mathbf{t}, \mathbf{c})$ then $X = 0$ if $t < c[1]$; $X = m$ if $c[m] \leq t < c[m+1]$ for $1 \leq m < M$; and $X = M$ if $c[M] \leq t$.

Paradoxically, X is a deterministic function of its parameters, even though it is defined as a stochastic node. The purpose of the `dinterval` distribution is to generate a likelihood for the unobserved variable t . The likelihood is 1 if t is consistent with X and 0 otherwise.

Truncation is represented in JAGS using the `T(,)` construct, which has the same syntax as the `I(,)` construct in `WinBUGS`, but has a different interpretation. If

```
X ~ dfoo(theta) T(L,U)
```

then *a priori* X is known to lie between L and U . This generates a likelihood

$$\frac{p(x | \theta)}{P(L \leq X \leq U | \theta)}$$

if $L \leq X \leq U$ and zero otherwise, where $p(x | \theta)$ is the density of X given θ according to the distribution `foo`. Note that calculation of the denominator may be computationally expensive.

Prior ordering of top-level parameters in the model can be achieved using the `sort` function, which sorts a vector in ascending order.

Symmetric truncation relations like this

```
alpha[1] ~ dnorm(0, 1.0E-3) T(,alpha[2])
alpha[2] ~ dnorm(0, 1.0E-3) T(alpha[1],alpha[3])
alpha[3] ~ dnorm(0, 1.0E-3) T(alpha[2],)
```

Should be replaced by this

```
for (i in 1:3) {
  alpha0[i] ~ dnorm(0, 1.0E-3)
}
alpha <- sort(alpha0)
```

6 Development

At some point there will be a separate manual for JAGS developers. At the moment, if you want to start hacking JAGS then you must rely on the source file documentation. This is written in JavaDoc style and can be processed using `kdock` to generate an on-line reference manual.

The JAGS source is divided into two main directories: `lib` and `terminal`. The `lib` directory contains the JAGS library, which contains all the facilities for defining a Bayesian graphical model in the BUGS language, running the Gibbs sampler and monitoring the sampled values. The JAGS library is divided into several convenience libraries

sarray which defines the basic `SArray` class, modelled on an S language array, and its associated classes.

functions which defines the standard JAGS functions and the `FuncTab` class that allows you to reference them by name.

distributions which defines the standard JAGS distribution and the `DistTab` class that allows you to reference them by name.

matrix which provides a C interface to various LAPACK functions that are used by multivariate functions and distributions in JAGS.

graph which defines the various Node classes used by JAGS when constructing a Bayesian graphical model, as well as the `Graph` class which is a container for nodes.

sampler which defines the various samplers that update stochastic nodes in the graph

model which defines all the classes needed to create a model, including monitor classes.

compiler which contains the `Compiler` class and a number of supporting classes designed for an efficient translation of a BUGS-language description the model into a `Graph`.

The `Console` class provides a clean interface to the JAGS library. The member functions of the `Console` class conduct all of the operations one may wish to do on a Bayesian graphical model. They are designed to catch any exceptions thrown by the library and print an informative message to either an output stream or an error stream, depending on the result.

The `terminal` directory contains the source code for a reference front end for the JAGS library, which uses the `Stata`-like syntax discussed above.

Currently, the JAGS library is not installed separately from the client. But in a future version, it will be. This will allow any program to link to the JAGS library, and in particular, a package for R is planned. This development will wait until any outstanding bugs in the library are resolved, and the library is optimized.

7 Feedback

Please send feedback to <jags@iarc.fr>. Since JAGS is currently in beta, I am particularly interested in the following problems:

- Crashes, including both segmentation faults and uncaught exceptions.
- Incomprehensible error messages
- Models that should compile, but don't
- Output that cannot be validated against WinBUGS
- Documentation errors

If you want to send a bug report, it must be reproducible. Send the model file, the data file, the initial value file and a script file that will reproduce the problem. Describe what you think should happen, and what did happen.

8 Acknowledgments

Many thanks to the BUGS development team for creating a piece of software so good I had to copy its functionality. Thanks also to Simon Frost for pioneering JAGS on Windows. Bettina Gruen, Chris Jackson, Greg Ridgeway, Jean-Baptiste Denis and Bill Northcott also provided useful feedback.

A Installation

JAGS is currently distributed in source form only. If you want to use it, you have to compile it. Precompiled binary distributions may be distributed later when JAGS has been more thoroughly tested.

A.1 Unix installation

JAGS is built using the GNU autotools, so in theory it should be portable. It has been successfully built on GNU/Linux, FreeBSD, Windows and MacOS X. If you manage to build JAGS on any other platform, then please let me know at <jags@iarc.fr>.

JAGS depends on two external libraries, LAPACK (<http://www.netlib.org/lapack>) and Rmath (<http://www.r-project.org>). You will need to install these libraries before building JAGS.

LAPACK is a set of linear algebra routines written in FORTRAN that is used by the multivariate functions and distributions in JAGS. LAPACK itself depends on the BLAS library (<http://www.netlib.org/blas>), although there are no explicit calls to BLAS functions from JAGS. Precompiled LAPACK and BLAS libraries should be available for most Unix systems. You may, at your discretion, choose to link LAPACK against the optimized ATLAS library (<http://math-atlas.sourceforge.net>) instead of BLAS. This may give important speed improvements if you make extensive use of multivariate functions and distributions, but otherwise it will not change anything. Bear in mind that JAGS is not currently optimized, so there may be better ways of improving its speed.

If the configure script fails to find the BLAS or LAPACK libraries, you may specify their locations using the flags `-with-blas` and `-with-lapack`, *e.g*

```
./configure --with-blas=/usr/lib/libblas.so.3.0 \  
            --with-lapack=/usr/lib/liblapack.so.3.0
```

The Rmath library is a part of the R distribution that can be compiled as a standalone library. It provides functions to calculate the density, distribution and quantile functions of common distributions as well as providing a random number generator. **IMPORTANT: You must use version R 1.9.0 or later for the standalone math library.** To build it, unpack the R source and follow the directions in the directory `src/nmath/standalone`. After building Rmath, you may wish to copy the header file `Rmath.h` to a place where the C preprocessor will find it (typically `/usr/local/include`) and the libraries `libRmath.a` and `libRmath.so` to a place where your linker will find them (typically `/usr/local/lib`). Don't forget to run `ldconfig` if you want to use dynamic linking.

If you don't want to install the Rmath header and library files, you will have to set the following environment variables before building JAGS

```
export CPPFLAGS="-I<RSRCDIR>/src/include/"  
export LDFLAGS="-L<RSRCDIR>/src/nmath/standalone/"
```

Building JAGS then follows the standard procedure:

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

This installs the JAGS executable in `/usr/local/bin`, along with the JAGS library and header files in `/usr/local/lib` and `/usr/local/include/JAGS` respectively.

A.2 Mac OS X installation

Instructions from Chris Jackson

This has only been tested on Panther (10.3), but this should also work on Jaguar (10.2)

To build JAGS on Mac OS X you must have the Apple Developer Tools installed. Since Mac OS 10.2 these include an optimized version of the LAPACK and BLAS libraries, so you should not need to obtain these separately.

To build the standalone math library libRmath.a and the corresponding header Rmath.h, you must use R-1.9.0 with the current patch set applied, or 1.9.1 when it is released.

Apple do not provide any Fortran development libraries, which are required to statically link JAGS with Rmath and LAPACK. Instead you can obtain the g77 package from the Fink repository of free software for MacOS X. After installing Fink (from <http://fink.sourceforge.net>), do

```
apt-get install g77
```

Then to build JAGS, follow the usual procedure from a Terminal window:

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

A.3 Windows installation

The following instructions describe how to build a statically linked version of JAGS for Windows. These instructions use MinGW, The Minimalist GNU system for Windows, which is available from <http://www.mingw.org>. JAGS has also been successfully built using the CygWin POSIX emulation layer for Windows (<http://www.cygwin.com>). You need some familiarity with Unix in order to follow the build instructions but, once built, the JAGS executable can be copied to any PC running Windows and can be run from the Windows command prompt.

A.3.1 Preparing the build environment

You need to download the following

- The MinGW compiler
- MSYS
- Any updates to MinGW

The MinGW compiler is distributed as a self extracting executable. Click on it and follow the on-screen instructions. By default, it will install into `c:\mingw`.

MSYS (the Minimal SYStem) is also a self-extracting executable. Once installed, it will launch a post-install script that will allow you to use MSYS in conjunction with MinGW. Note that you do *not* need the MSYS developer toolkit (DTK) to compile JAGS. 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, R and JAGS into this directory.

If there are any updates to MinGW (*i.e.* `mingw-runtime` and `win32api`), these will be distributed as gzipped tarballs. You should copy these to `c:\mingw`, and then unpack them from within the MSYS shell by changing directory to `/mingw` and then typing

```
tar xfvz mingw-runtime-<version>.tar.gz  
tar xfvz win32api-<version>.tar.gz
```

A.3.2 Building the required libraries

LAPACK

Download the LAPACK source file from <http://www.netlib.org/lapack> and unpack it in your home directory.

```
tar xfvz lapack.tgz
cd LAPACK
```

Replace the file `make.inc` with `INSTALL/make.inc.LINUX`. Then edit `make.inc`, replacing the line `PLAT = _LINUX`

with something more sensible, like

```
PLAT = _WinNT
```

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

```
lib: blaslib lapacklib
```

It is not necessary to build `tmglib`. Type

```
make lib
```

Copy the resulting file `blas_WinNT.a` to `/mingw/lib/libblas.a` and `lapack_WinNT.a` to `/mingw/lib/liblapack.a`.

Rmath

Get the latest R source from <http://cran.r-project.org>. This is a gzipped tar file, like the others mentioned previously, but there is a difference: the R tarball contains symbolic links. Many compression utilities can't create symlinks, including WinZIP and the version of tar that comes with MSYS. You have three options

1. If you have CygWin installed, use the CygWin tar command to unpack the R source
2. Use the tar command provided by Brian Ripley in the file `tools.zip` available from <http://www.stats.ox.ac.uk/Rtools>.
3. Use another tool to unpack the sources and hope for the best

Option 3 is not recommended, and above all should not be the subject of a bug report.

Once you have unpacked the R sources, use the following sequence of commands to build the Rmath library

```
cd R-<version>
./configure
cd src/install
make
cd ../nmath/standalone
make static
```

This will create a file `libRmath.a` which you should copy to `/mingw/lib`. You should also copy the file `R-<version>/src/include/Rmath.h` to `/mingw/include`

Note that you cannot do a complete build of R in this way. These instructions are only for building the standalone Rmath library. There is extensive documentation on building R for Windows.

A.3.3 Compiling JAGS

To build JAGS, unpack the JAGS source and type the following commands from within MSYS

```
./configure --disable-shared  
make
```

The `jags.exe` executable may then be found in the file `JAGS-0.75/src/terminal`. Copy it to somewhere on your Windows PATH. You can modify the PATH environment variable from the system dialogue in the control panel.

By default, the JAGS executable contains a lot of debugging information and is consequently quite large. If you are not going to do any debugging then you may wish to strip it:

```
strip jags.exe
```

This considerably reduces the file size.