# Introduction to JAGS

# Setup

## Introduction to `JAGS`

There are several software packages available that will handle the details of MCMC for us. See the supplementary material for a brief overview of options.

The package we will use in this course is `JAGS` (Just Another Gibbs Sampler) by Martyn Plummer. The program is free, and runs on Mac OS, Windows, and Linux. Better yet, the program can be run using `R` with the `rjags` and `R2jags` packages.

In `JAGS`, we can specify models and run MCMC samplers in just a few lines of code; `JAGS` does the rest for us, so we can focus more on the statistical modeling aspect and less on the implementation. It makes powerful Bayesian machinery available to us as we can fit a wide variety of statistical models with relative ease.

## Installation and setup

The starting place for `JAGS` users is mcmc-jags.sourceforge.net. At this site, you can find news about the features of the latest release of `JAGS`, links to program documentation, as well as instructions for installation.

The documentation is particularly important. It is available under the files page link in the Manuals folder.

Also under the files page, you will find the `JAGS` folder where you can download and install the latest version of `JAGS`. Select the version and operating system, and follow the instructions for download and installation.

Once `JAGS` is installed, we can immediately run it from `R` using the `rjags` package. The next segment will show how this is done.

# Modeling in `JAGS`

There are four steps to implementing a model in `JAGS` through `R`:

1. Specify the model.
2. Set up the model.
3. Run the MCMC sampler.
4. Post processing.

We will demonstrate these steps with our running example with the data are the percent change in total personnel from last year to this year for \(n=10\) companies. We used a normal likelihood with known variance and t distribution for the prior on the unknown mean.

## 1. Specify the model

In this step, we give `JAGS` the hierarchical structure of the model, assigning distributions to the data (the likelihood) and parameters (priors). The syntax for this step is very similar to `R`, but there are some key differences.

```
library("rjags")

mod_string = " model {
  for (i in 1:n) {
    y[i] ~ dnorm(mu, 1.0/sig2)
  }
  mu ~ dt(0.0, 1.0/1.0, 1.0) # location, inverse scale, degrees of freedom
  sig2 = 1.0
} "
```

One of the primary differences between the syntax of `JAGS` and `R` is how the distributions are parameterized. Note that the normal distribution uses the mean and precision (instead of variance). When specifying distributions in `JAGS`, it is always a good idea to check the `JAGS` user manual here in the chapter on Distributions.

## 2. Set up the model

```
set.seed(50)
y = c(1.2, 1.4, -0.5, 0.3, 0.9, 2.3, 1.0, 0.1, 1.3, 1.9)
n = length(y)

data_jags = list(y=y, n=n)
params = c("mu")

inits = function() {
  inits = list("mu"=0.0)
} # optional (and fixed)

mod = jags.model(textConnection(mod_string), data=data_jags, inits=inits)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 10
##    Unobserved stochastic nodes: 1
##    Total graph size: 28
##
## Initializing model
```

There are multiple ways to specify initial values here. They can be explicitly set, as we did here, or they can be random, i.e., `list("mu"=rnorm(1))`. Also, we can omit the initial values, and `JAGS` will provide them.

## 3. Run the MCMC sampler

```
update(mod, 500) # burn-in

mod_sim = coda.samples(model=mod,
                       variable.names=params,
                       n.iter=1000)
```

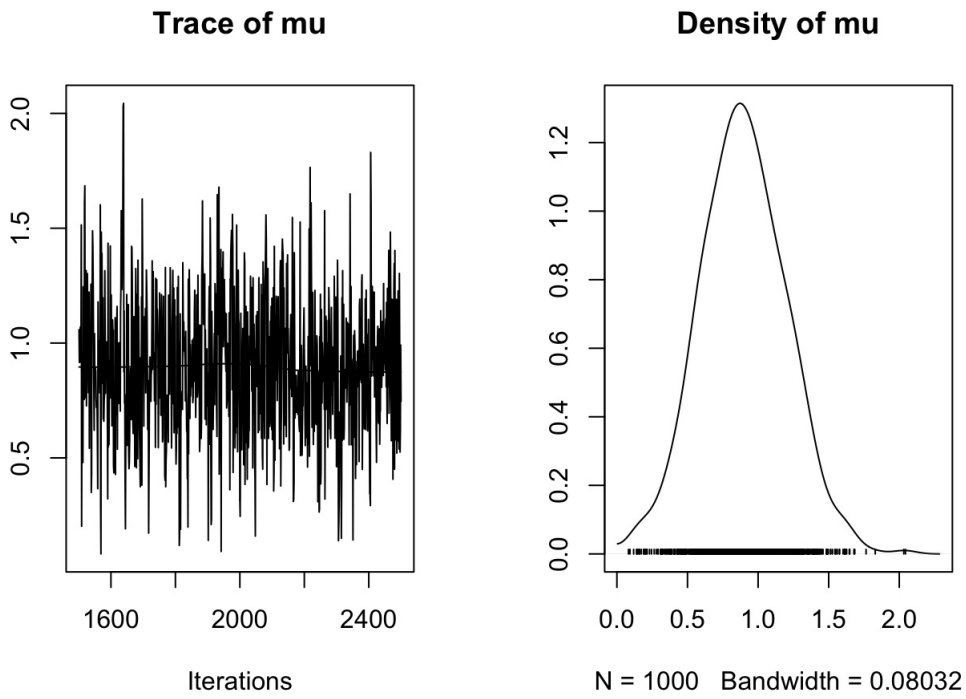
We will discuss more options to the `coda.samples` function in coming examples.

## 4. Post processing

```
summary(mod_sim)
```

```
##
## Iterations = 1501:2500
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##       0.895867       0.301648       0.009539       0.012256
##
## 2. Quantiles for each variable:
##
##   2.5%    25%    50%    75%  97.5%
## 0.3153 0.6854 0.8888 1.0937 1.4977
```

```
library("coda")
plot(mod_sim)
```




We will discuss post processing further, including convergence diagnostics, in a coming lesson.