A tutorial introduction to Bayesian hierarchical modelling

Darren Wilkinson

darrenjw.github.io

Bayesian statistics

Bayesian models

In Bayesian statistics we combine prior information with data via a model for the data-generating process (using Bayes theorem) to get revised information in the form of a *posterior distribution*.

We assume that the data, X, arises from a process described by a distribution that depends on some parameters, θ. We denote the model, $p(x|\theta)$, so that

$$X \sim p(x|\theta).$$

When this model is considered a function of θ for given fixed x, it is often referred to as the *likelihood* function, and written $L(\theta; x) \equiv p(x|\theta)$.

Bayes theorem

The prior information about the parameters is described using a probability distribution $\pi(\theta)$, so the joint distribution of the parameters and the data can be written

$$\pi(\theta, x) = \pi(\theta)p(x|\theta) = \pi(\theta)L(\theta; x).$$

Note that $\pi()$ is used to denote different distributions - it is supposed to be clear from the arguments which distribution is being referred to.

Given observed data $X = x$, we use Bayes theorem to construct the posterior distribution as

$$\pi(\theta|x) = \frac{\pi(\theta)L(\theta; x)}{\pi(x)} = \frac{\pi(\theta)L(\theta; x)}{\displaystyle\int_\Theta \pi(\theta')L(\theta'; x)\, d\theta'} \propto \pi(\theta)L(\theta; x),$$

since the denominator is clearly just a normalising constant.

Prior to posterior

So,

$$\pi(\theta|x) \propto \pi(\theta)L(\theta; x),$$

and *the posterior is proportional to the prior times the likelihood*. The posterior distribution $\pi(\theta|x)$ is a distribution describing our beliefs about the parameter (vector) θ taking into account the observed data $X = x$, as opposed to the prior distribution, $\pi(\theta)$, which is the distribution describing beliefs about the parameters prior to to observing the data.

Clearly the likelihood is responsible for transforming the prior to the posterior, so the likelihood function contains all information in the data relevant to inference about θ.

Simple example - slope failures

Suppose we are interested in the number of slope failures occurring on a particular line each year. An expert believes that the rate at which slope failures occur can be well described by a Gamma(2,1) distribution - this has a mean and variance of 2. Historical data reveals that there have been 32 failures in the last 20 years. How should the expert revise their belief in the light of this data?

Let the annual rate be θ, so that $\theta \sim \text{Gamma}(2,1)$. The simplest model for the data is that the number of failures is Poisson with mean 20θ, that is $X \sim \text{Poisson}(20\theta)$.

We therefore have

$$\pi(\theta) \propto \theta e^{-\theta} \quad \text{and} \quad L(\theta; x) \propto \theta^{32} e^{-20\theta},$$

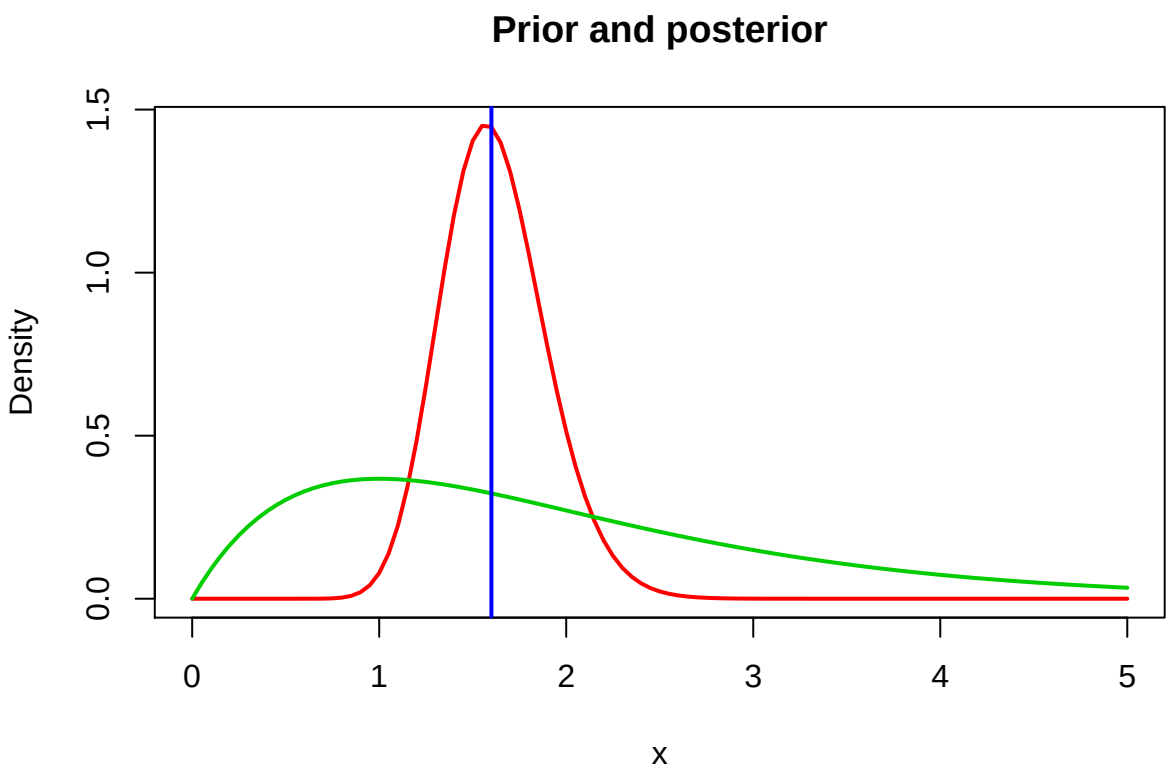
and so Bayes theorem gives

$$\pi(\theta|x) \propto \theta e^{-\theta} \times \theta^{32} e^{-20\theta} = \theta^{33} e^{-21\theta},$$

and so the posterior is Gamma(34,21), which has a mean of 1.62 and a variance of 0.077. Note how the mean has shifted towards the empirical estimate (of 1.6) and the uncertainty has been greatly reduced.

This problem has an analytical solution, since a Gamma prior is *conjugate* to a Poisson likelihood, leading to a Gamma posterior. But most interesting problems are not tractable, and we use computers to generate Monte Carlo samples from the posterior of interest.

```
curve(dgamma(x,34,21), 0,5, col=2,lwd=2,
  ylab="Density", main="Prior and posterior")
curve(dgamma(x,2,1), 0,5, col=3,lwd=2, add=TRUE)
abline(v=32/20, col=4,lwd=2)
```



Bayesian hierarchical models

Introduction

A Bayesian hierarchical model can often be thought of as simply a graphical model where the nodes represent quantitative random variables rather than (just) discrete categorical outcomes. Some nodes represent unobserved variables, and others the data to be observed. The model defines the joint distribution of unobserved model parameters and the data, completely specifying the data generating process, and we are typically interested in the conditional distribution of the model parameters given the data. This is usually intractable, but we typically use Markov chain Monte Carlo (MCMC) methods to generate stochastic samples from this distribution.

JAGS and rjags

JAGS (Just Another Gibbs Sampler) is an example of some general-purpose free software for carrying out Markov chain Monte Carlo (MCMC) simulation for (intractable) Bayesian hierarchical models. Note that JAGS is a standalone piece of software - it is not an R package, and must be installed separately (outside of R) before being used, and may require admin privileges.

Once JAGS is correctly installed on your system, there is an R package on CRAN called `rjags` which is straightforward to install, and makes it easy to use JAGS from within R. It should be possible to install with

```
install.packages("rjags")
```

but note that installation of this package will fail if JAGS is not already correctly installed.

Using rjags

Once JAGS and `rjags` are installed it can be loaded for use in the usual way.

```
library(rjags)
```

```
help(package="rjags")
?"rjags-package"
?jags.model
```

This package does not have any vignettes.

Example: slope failures with rjags

We can reproduce our analytically tractable example using `rjags` as follows.

```
x=32
data=list(x=x)
init=list(theta=1)
modelstring="
  model {
    x~dpois(20*theta)
    theta~dgamma(2,1)
  }
"
```

Note that we put the JAGS model in a string, which we now pass to JAGS for analysis.

```
model=jags.model(textConnection(modelstring),
  data=data, inits=init)
```

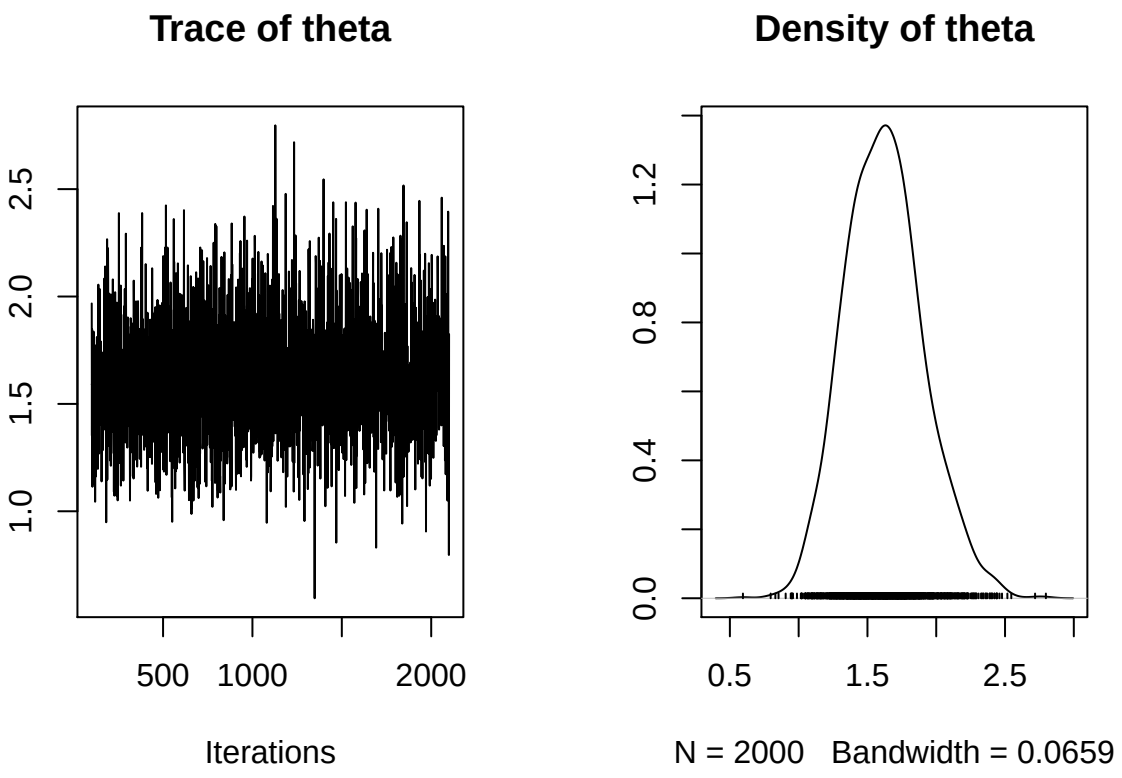
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1
##    Unobserved stochastic nodes: 1
##    Total graph size: 6
##
## Initializing model
```

```
update(model,n.iter=100)
output=coda.samples(model=model,
  variable.names=c("theta"), n.iter=2000, thin=1)
```

```
print(summary(output))
```

```
##
## Iterations = 101:2100
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 2000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##       1.623048       0.284294       0.006357       0.006357
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
## 1.103 1.419 1.615 1.801 2.220
```

```
plot(output)
```



Example - fitting distributions

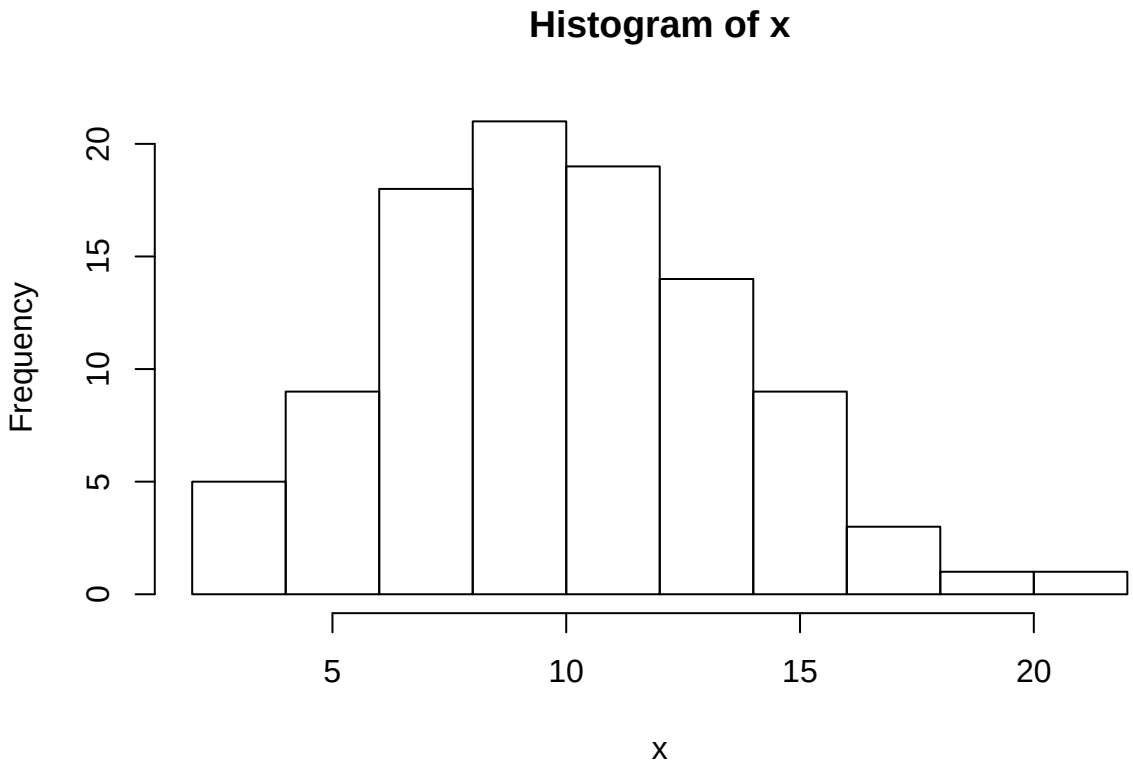
Suppose we want to fit a parametric distribution (eg. Gamma) to some experimental data (eg. soil strength measurements).

Here we'll just simulate some synthetic data:

```
set.seed(1)
n = 100
x = rgamma(n, 5, 0.5)
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.539   7.158   9.608   9.860  12.324  20.578
```

```
hist(x)
```



There are some standard frequentist techniques for fitting distributions to data.

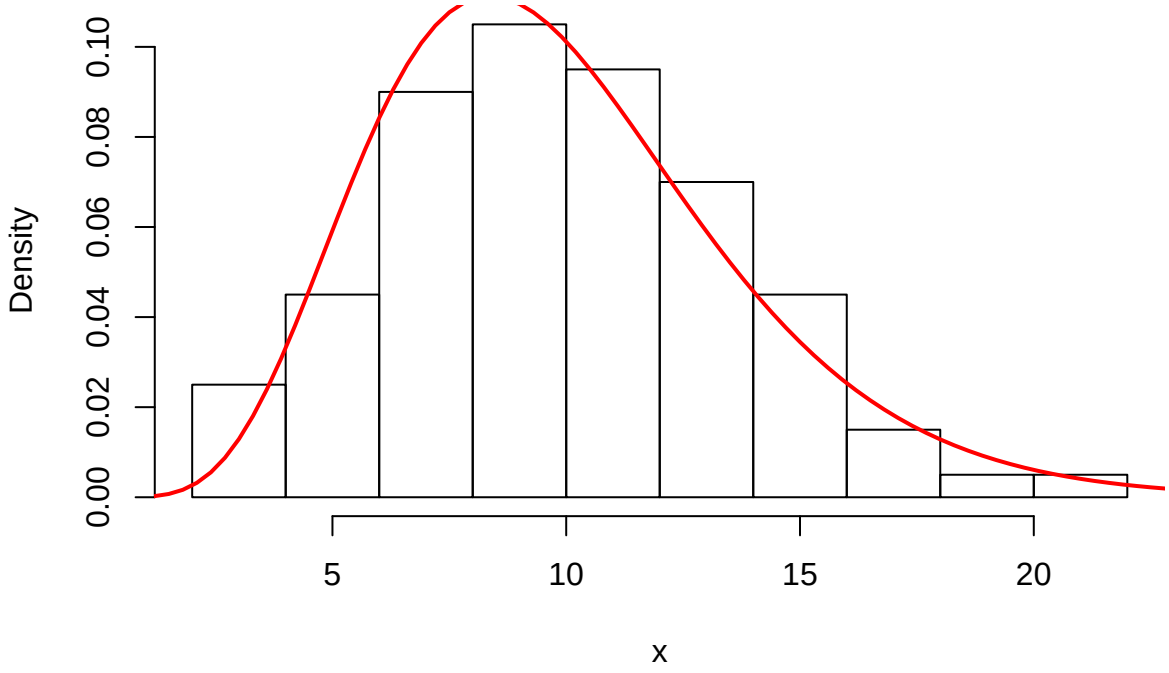
```
library(MASS)
fit = fitdistr(x, "gamma")
```

```
## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced
```

```
fit
```

```
##      shape         rate
##   6.60335788   0.66973638
##  (0.91122771) (0.09602435)
```

```
hist(x,freq=FALSE)
curve(dgamma(x, fit$estimate[1], fit$estimate[2]),
  0, 30, add=TRUE, col=2, lwd=2)
```



We can fit log-normal to the same data

```
fitdistr(x, "log-normal")
```

```
##     meanlog       sdlog
##   2.21082325   0.41257491
##  (0.04125749) (0.02917345)
```

- Frequentist methods don't properly quantify and propagate uncertainty
- Let's do again using JAGS

```
data=list(x=x, n=n)
init=list(a=1, b=1)
modelstring="
  model {
    for (i in 1:n) {
      x[i] ~ dgamma(a, b)
    }
    a ~ dgamma(1, 0.001)
    b ~ dgamma(1, 0.001)
  }
"
```

```
model=jags.model(textConnection(modelstring),
  data=data, inits=init)
```

```
## Compiling model graph
##    Resolving undeclared variables
```

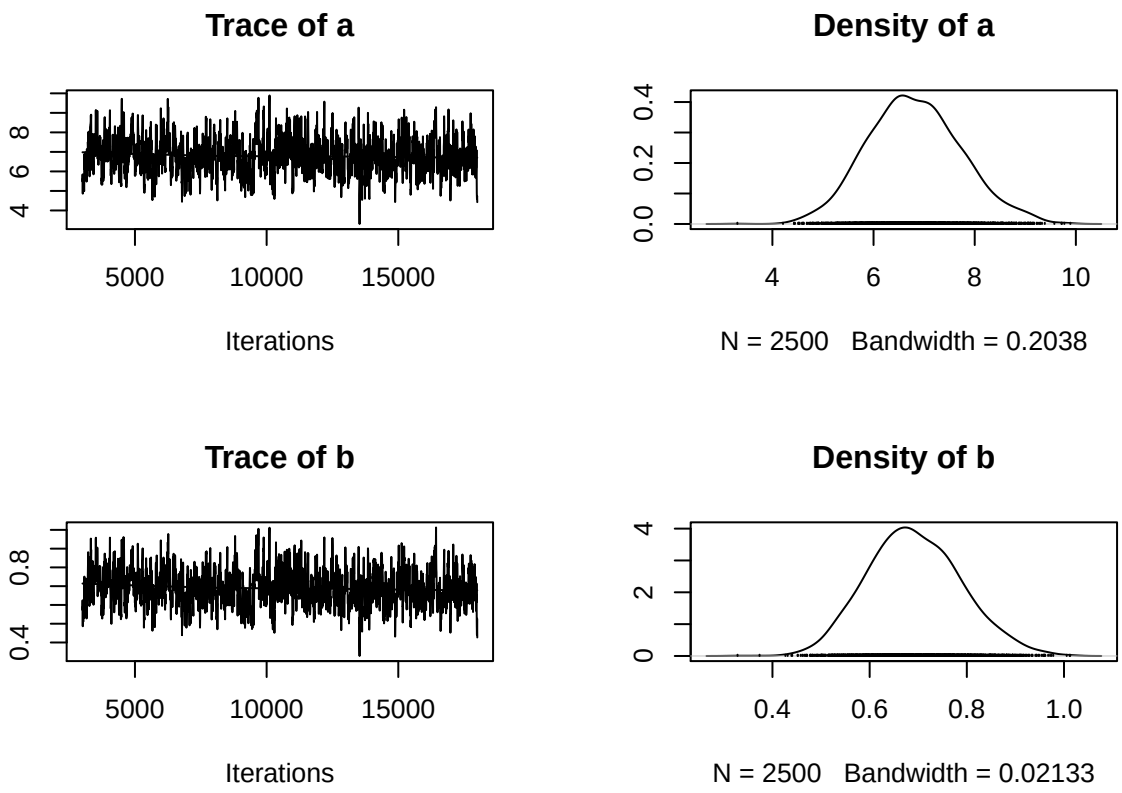
```
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 2
##    Total graph size: 105
##
## Initializing model
```

```
its=2500; thin=6
update(model, n.iter=2000)
output=coda.samples(model=model,
  variable.names=c("a","b"), n.iter=its*thin, thin=thin)
```

```
print(summary(output))
```

```
##
## Iterations = 3006:18000
## Thinning interval = 6
## Number of chains = 1
## Sample size per chain = 2500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##     Mean      SD Naive SE Time-series SE
## a 6.8125 0.91939 0.018388       0.044170
## b 0.6922 0.09621 0.001924       0.004455
##
## 2. Quantiles for each variable:
##
##     2.5%    25%    50%    75%  97.5%
## a 5.0828 6.1692 6.7689 7.4013 8.7314
## b 0.5191 0.6261 0.6871 0.7571 0.8892
```

```
plot(output)
```



```
## convert MCMC output to a matrix
mat = as.matrix(output)
dim(mat)
```

```
## [1] 2500    2
```

```
## select 5 iterations at random
samples = sample(1:its, 5)
samples
```

```
## [1] 1536  504 1481  358  785
```

```
hist(x, freq=FALSE, ylim=c(0,0.15), main="Sample fits")
for (i in samples) curve(dgamma(x, mat[i,1],
  mat[i,2]), 0, 30, add=TRUE, col=2)
```

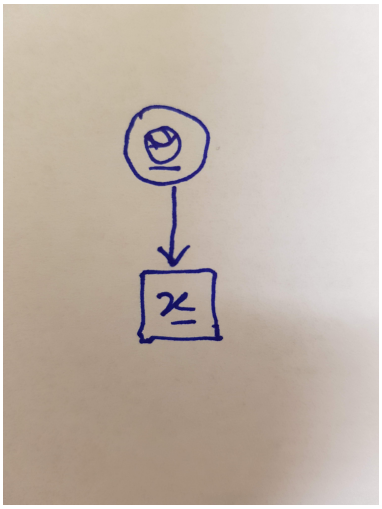
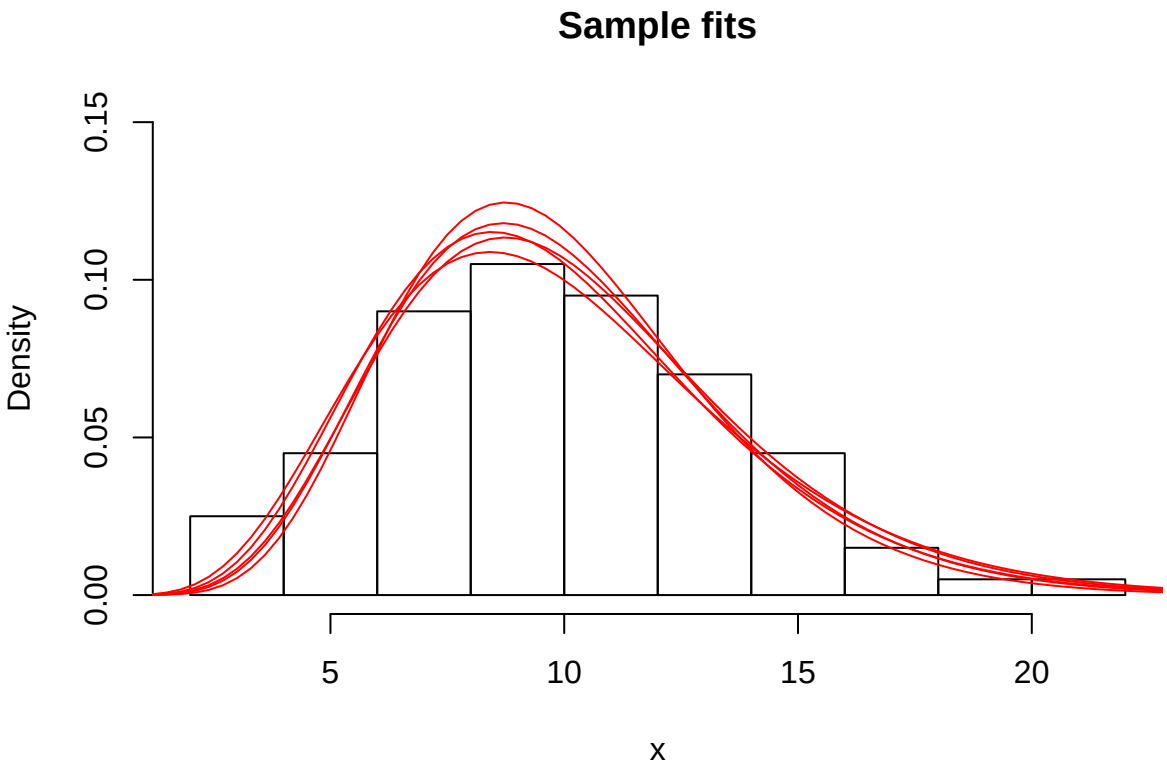


Figure 1: 2-layer DAG model



Hierarchical structure and latent variables

Minimal 2-layer DAG model structure

(in Bayesian modelling, we often use rectangular nodes for data)

Typical 3-layer structure with latent variables, z

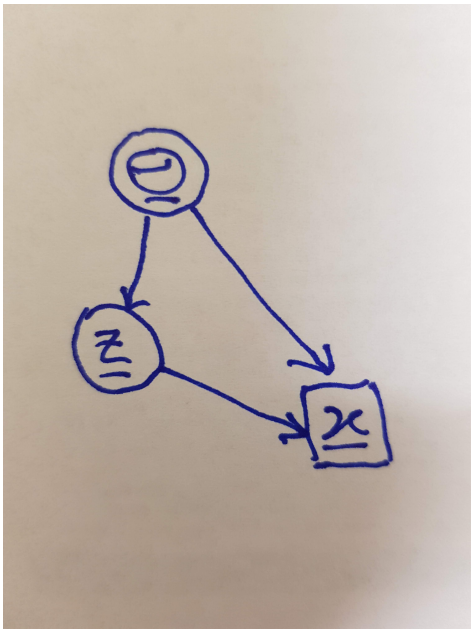


Figure 2: 3-layer DAG model

Factorisation: $\pi(\theta, z, x) = \pi(\theta)\pi(z|\theta)p(x|z, \theta)$

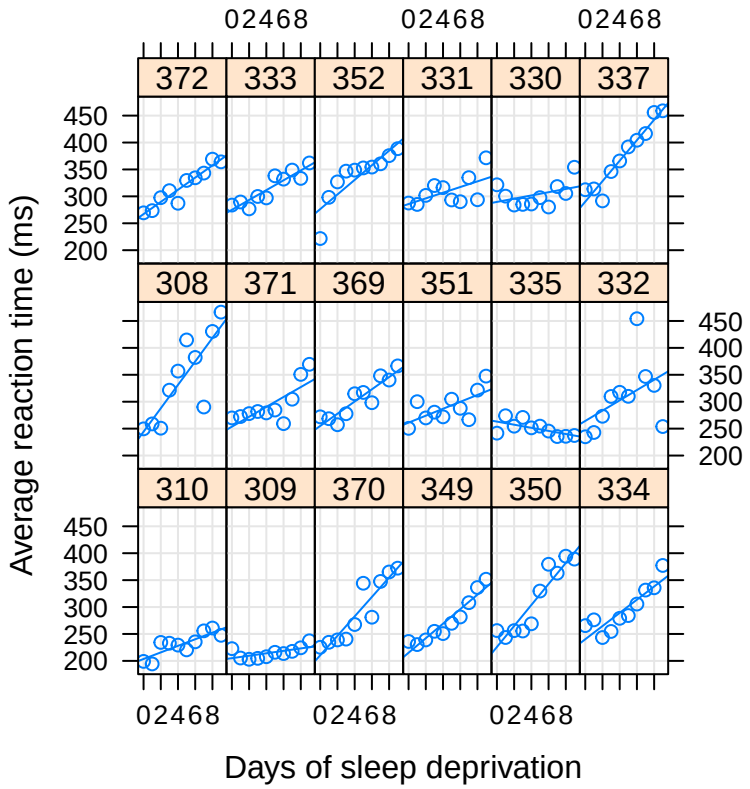
Example - random effects

```
library(lme4)
```

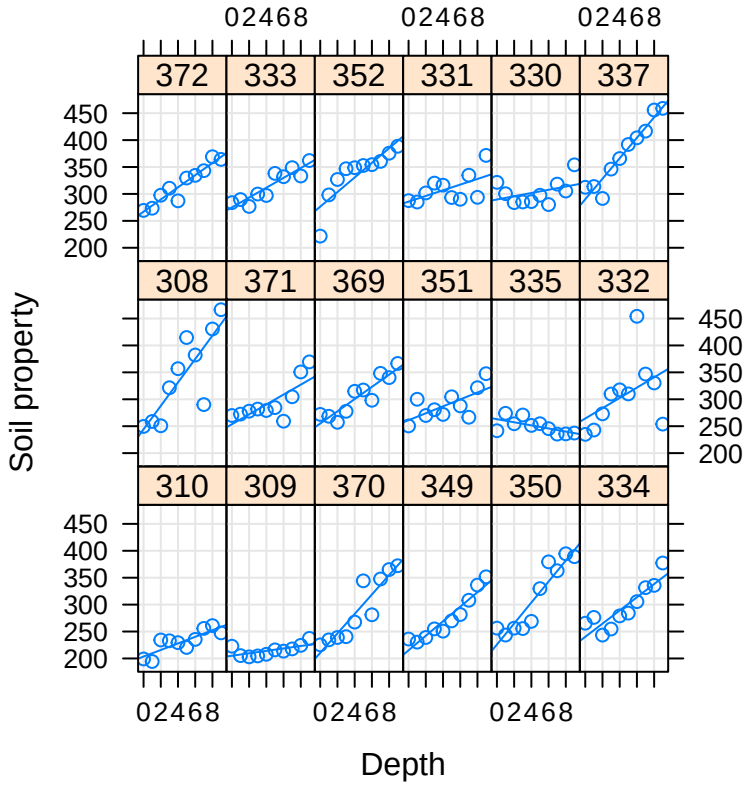
```
## Loading required package: Matrix
```

```
library(lattice)
```

```
?sleepstudy
example(sleepstudy)
```



Reaction time with days of sleep deprivation in 18 subjects



Soil property varying with depth in 18 London clay slopes

We want to treat the `Days` coefficient (the gradient of the regression line) as a *random effect*, which can vary from slope to slope, but drawn from a population of values from a distribution.

Here we will treat the intercept of each regression line as a *fixed effect* (but still a latent variable).

```
data=list(y=sleepstudy$Reaction, x=sleepstudy$Days,
  index=as.numeric(sleepstudy$Subject),
  n=dim(sleepstudy)[1])
init=list(mu=1, tau=1, tau.s=1)
```

```
modelstring="
  model {
    for (i in 1:18) {
      slope[i] ~ dnorm(mu, tau.s)
      inter[i] ~ dnorm(0, 0.0001)
    }
    for (j in 1:n) {
      y[j] ~ dnorm(inter[index[j]] +
        slope[index[j]]*x[j], tau)
    }
    tau ~ dgamma(1, 0.001)
    tau.s ~ dgamma(1, 0.001)
    mu ~ dnorm(0, 0.0001)
  }
"
```

```
model=jags.model(textConnection(modelstring),
  data=data, inits=init)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 180
##    Unobserved stochastic nodes: 39
##    Total graph size: 944
##
## Initializing model
```

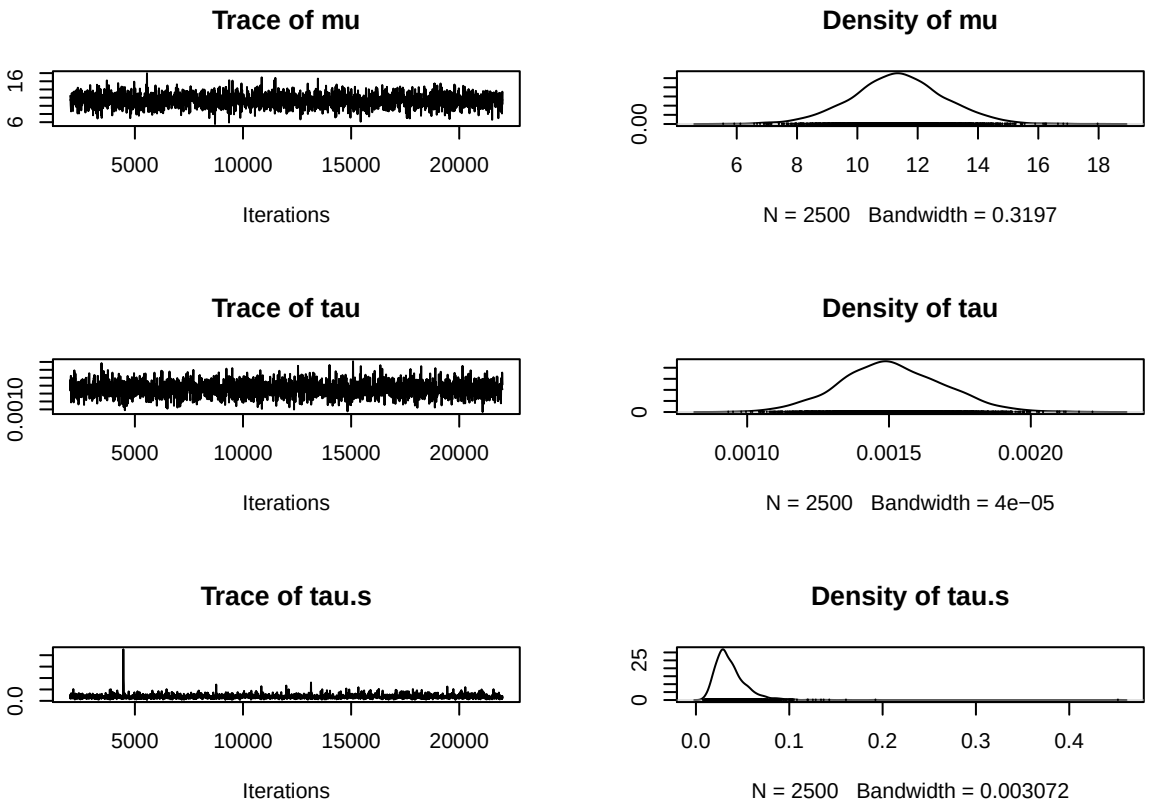
```
its=2500; thin=8
update(model, n.iter=2000)
output=coda.samples(model=model,
  variable.names=c("mu","tau","tau.s"),
  n.iter=its*thin, thin=thin)
```

```
print(summary(output))
```

```
##
## Iterations = 2008:22000
## Thinning interval = 8
```

```
## Number of chains = 1
## Sample size per chain = 2500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean        SD  Naive SE Time-series SE
## mu    11.355605 1.5436958 3.087e-02      3.087e-02
## tau    0.001512 0.0001815 3.630e-06      3.630e-06
## tau.s  0.037143 0.0189516 3.790e-04      4.237e-04
##
## 2. Quantiles for each variable:
##
##           2.5%       25%       50%       75%    97.5%
## mu    8.219464 10.385895 11.349899 12.318698 14.41239
## tau   0.001167  0.001389  0.001501  0.001631  0.00188
## tau.s 0.014741  0.025764  0.033494  0.044332  0.07970
```

```
plot(output)
```



- The slope is drawn from a distribution with mean around 11.5 and standard deviation around 1.5
- This random effects distribution allows "borrowing of strength" across different slopes, including any with sparse observations
- It also allows shared learning of the observational error without having to assume a common slope
- We could use this distribution of random effects to draw values for slopes not observed or measured in our data set

Further reading

On-line material

Wikipedia

- Bayesian statistics
- Bayesian hierarchical modelling
- MCMC

Software

- JAGS
- Stan
- WinBUGS
- CRAN Task views
  - Bayesian
- R packages
  - rjags
  - rstan
  - brms

Books

- BDA3: Bayesian data analysis, third edition (Gelman et al) - ISBN 1439840954
- Statistical rethinking (McElreath) - ISBN 1315362619
- Bayesian reasoning and machine learning (Barber) - ISBN 0521518148 - **free PDF available**