Practical Data Analysis with JAGS using R

Department of Biostatistics
Institute of Public Health, University of Copenhagen
Tuesday 1st January, 2013
Computer practicals

Compiled Tuesday 1st January, 2013, 17:54
from: `C:/Bendix/undervis/Bayes/Cph.2012/pracs/pracs.tex`

Lyle Gurrin Senior Lecturer
Centre for MEGA Epidemiology
School of Population Health, University of Melbourne
Carlton, Victoria, Australia
lgurrin@unimelb.edu.au
http://www.sph.unimelb.edu.au/about/contact/allstaff/gurrin

Bendix Carstensen Senior Statistician
Steno Diabetes Center, Gentofte
& Department of Biostatistics, University of Copenhagen
bxc@steno.dk
http://www.biostat.ku.dk/~bxc

Søren Højsgaard Head
Department of Mathematical Sciences
Aalborg University
sorenh@math.aau.dk
http://people.math.aau.dk/~sorenh/

Claus Ekstrøm Professor
Department of Biostatistics, Institute of Public Health,
University of Southern Denmark
cekstrom@health.sdu.dk
http://www.sdu.dk/staff/cekstrom

Contents

Course program

Venue: CSS 1.1.12, Øster Farimagsgade 5, a detailed map of the buildings is here.

If you are in front of the big yellow-brick building with one gate on either side of the spire, choose the *right* gate turn left inside the gate or choose the *left* gate turn right inside the gate, take the stairs up one floor. You are now in building 1, 1st floor, 1.1. Then go to room 12.

Course schedule:

				Present			
Day	Time	Lectures	Practicals	LG	BC	CE	SH
Mon	Morning	1.5	2.0	●	●		
	Afternoon	1.5	2.0	●	●		
Tue	Morning	1.5	2.0	●	●	●	●
	Afternoon	1.0	2.5	●	●	●	●
	Evening	Course dinner					
Wed	Morning	1.5	2.0	●	●	●	●
	Afternoon	1.0	2.5	●	●	●	●
Thu	Morning	1.0	2.5	●	●	●	●
	Afternoon	Free					
Fri	Morning	1.0	2.5	●	●		
	Afternoon	1.0	2.5	●	●		
Total		11.0	20.5				

Monday 13 August 2012

09:00 – 09:30	Registration & coffee.
09:30 – 10:15	**Lecture 1**: Introduction to Bayesian analysis: The binomial model as an example. (LG)
10:15 – 10:30	**Lecture/Practical 0**: Getting R incl.`r-INLA` and `JAGS` running. (BxC)
10:30 – 11:00	**Morning Tea**
11:00 – 12:30	**Practical 1**: Bayesian analysis in R: Discrete prior distribution in the DRUGS example. Illustration of posterior = likelihood × prior. The effect of data and prior variance using beta probability functions in R. (BxC)
12:30 – 13:30	**Lunch**
13:30 – 14:30	**Lecture 2**: Introduction to MCMC and the `BUGS` programming language. (BxC/SH)
14:30 – 16:00	**Practical 2**: Simple analyses in `BUGS` using the binomial distribution, example of restricted uniform or beta prior distribution with narrow prior support for a range of parameter values. (BxC/SH)

Tuesday 14 August 2012

09:00 – 09:30	**Recap of Monday**
09:30 – 10:00	**Lecture 3**: The general philosophical divide between frequentist and Bayesian approaches to models and data analysis (CE)
–	**Practical 3**: BOP (blank on purpose)
10:00 – 10:30	**Lecture 4**: Demonstrating the Gibbs sampler with a multiparameter problem and some data. The role of DAG-able models for the `BUGS` machinery to work. (SH)
10:30 – 11:00	**Morning Tea**
11:00 – 12:30	**Practical 4**: The Gibbs sampler and the Metropolis-Hastings sampler with a bivariate normal example. (SH)
12:30 – 13:30	**Lunch**
13:30 – 14:00	**Lecture 5**: The linear normal model, multiparameter problems and the conceptually simple Bayesian approach. (LG)
14:00 – 14:45	**Practical 5**: Speed of light example showing the use of posterior predictive checking. First introduce a noninformative prior distribution for the mean and then an informative distribution - does this influence our opinion as to whether the lowest observations are outliers? (LG)
14:45 – 15:15	**Lecture 6**: Generalized linear models. (LG)
15:15 – 16:30	**Practical 6**: Airline fatalities and posterior prediction of future fatalities: Several models: 1) Linear in log rate, 2) Linear in rate (problems with prior spec.) [1&2 simple in R.]. 3) Parametric model of rate decay. (BxC/LG)
18:00 – 22:00	Course dinner.

Wednesday 15 August 2012

09:00 – 09:30	**Recap of Tuesday**
09:30 – 10:00	**Lecture 7**: The INLA approach to Bayesian analysis: Advantages and shortcomings relative to MCMC (SH)
10:00 – 11:00	**Practical 7**: A simple normal random effects model fitted with `lme/lmer`, BUGS and INLA (BC)
11:00 – 11:30	**Morning Tea**
11:30 – 12:00	**Lecture 8**: Monitoring convergence and the need to run multiple chains. (LG)
–	**Practical 8**: BOP
12:00 – 13:00	**Lunch**
13:00 – 14:00	**Lecture 9**: Hierarchical models. (LG)
14:00 – 16:00	**Practical 9**: An example of a hierarchical model. Contrasting INLA / BUGS [Fetal growth???] (BC)

Thursday 16 August 2012

09:00 – 09:30	**Recap of Wednesday**
09:30 – 10:00	**Lecture 10**: Generalised linear mixed models (GLMMs) in `BUGS`. (LG)
10:00 – 10:30	**Morning Tea**
10:30 – 12:30	**Practical 10**: Illustration of GLMMs using clustered binary data from GPs, also twin and family data with genetically structured covariance. (LG/CE)
12:30 – 13:30	**Lunch**
–	Afternoon free

Friday 17 August 2012

09:15 – 09:30	**Recap of Thursday**
09:45 – 10:30	**Lecture 11**: Model comparison using DIC. (LG)
10:30 – 11:00	**Morning Tea**
11:00 – 12:30	**Practical 11**: Comparing models in `BUGS` using DIC. (LG)
12:30 – 13:30	**Lunch**
13:30 – 14:15	**Lecture 12**: Comparing methods of measurement in Stata, SAS, `R` and `BUGS`. (BxC)
14:15 – 16:00	**Practical 12**: Comparing methods of measurement using the `MethComp` package — reporting (BxC)
16:00 – 16:15	Wrapping up, closure, evaluation and farewell

Chapter 1

Introduction to computing and practicals

This short course is both theoretical and practical, that is, the aim is to convey a basic understanding of the Bayesian framework for data analysis as well practical computing skills in Bayesian methods. The two components of the course are supposed to support each other.

The practicals during the week will take place in a class room, since the most convenient way to do this part of the course will be to work on your own laptop computer. This will ensure that useful scripts and tricks are readily available for your future exploitation.

The following is a brief overview of the software and other files you must download if you want to use your own computer.

1.1 Software

1.1.1 Overview

In this course, we use the Markov Chain Monte Carlo (MCMC) machinery which is implemented in various guises of `BUGS`. The original purpose of the software `BUGS` was to use it for Bayesian inference, but in many practical circumstances it is used with flat or (almost) non-informative prior distributions, effectively taking a likelihood-based approach to estimation and inference.

The latter type of application is the main content of this course. The use of the software does, however, require a basic knowledge of the Bayesian approach to statistical inference, which is based on full probability modelling.

The data manipulation and report generation is done with `R` in this course, as this is the state of the art in practical statistics. The practical workhorse for the MCMC simulations will be the `JAGS` implementation http://mcmc-jags.sourceforge.net/.

In order to interact with `JAGS` programs, this course will use the `rjags` interface, which basically throws `R` data structures at `JAGS` and sucks the results back into `R`, as suitable objects for further processing. This enables you to maintain a completely reproducible record of your initial data-manipulation (in `R`), estimation (in `JAGS`) and reporting of results (in `R`).

The scripting language in `JAGS` is (almost) the same as for the other implementations of `BUGS`.

In order to be able to write scripts (programs) in R and keep them for future use (and modification for other purposes) a good editor with interface to R is convenient. `Rstudio` is the answer. If you are already a user of ESS in Emacs, just forget about `Rstudio`.

We have decided to try using `INLA` (http://www.r-inla.org/) on an experimental basis. `INLA` uses a fast approximation to get posterior distributions, but it only produces marginal posteriors.

So you need R, `JAGS`, `INLA` and (possibly) `Rstudio`.

1.1.2 What to get

- `Rstudio` is available from http://rstudio.org/.
- R, version 2.15.1, get it from http://mirrors.dotsrc.org/cran/. The relevant packages for this course are easiest installed by firing up R, and then type:

  ```
  > install.packages("rjags","coda","Epi","lme4","pixmap","sp")
  ```

  You will be asked to select a mirror (i.e. a server) from which to download the stuff). `coda` is a package for post-processing and monitoring of MCMC-output, and `Epi` is a package for epidemiology from which we will use a few handy functions.
- `JAGS` from http://mcmc-jags.sourceforge.net/. Download and install the relevant version for your operating system.
- `INLA` from http://www.r-inla.org/download, where you will find the followinng instructions:

  Type the following command line in R:

  ```
  > source("http://www.math.ntnu.no/inla/givemeINLA.R")
  ```

  later on, you can upgrade using:

  ```
  > inla.upgrade()
  ```

1.2 Course material

Datasets and programs for the course will all be collected in the zip file `BDA2012.zip` which soon will be available at the course homepage,
http://BendixCarstensen.com/Bayes/Cph-2012/.

Download this file and unpack it in a separate folder. The resulting folder tree has the following sub-folders:

- data — datasets for use in the practicals.
- R — example R-programs providing solutions to some of the practicals.

At the root level you should find a version of the practicals including solutions to the exercises.

In the next two chapters with Exercises and Solutions, the section numbers (2nd enumeration level) corresponds to each other.

1.3 Simulating data in R

One of the major uses of computers in this course is simulation, so a brief section on how to do this in R is included here.

Start by opening R. In the following, "`>`" is the R-prompt, and "`+`" the continuation prompt, and these should not be typed. The lines starting with "`[1]`", "`[8]`" etc. are output from R, that you can use to check that you got the right output. Since this is about simulation, you will of course not get exactly the same output as shown here.

To simulate binomial variates $Y \sim \text{Bin}(N, p)$, the function to use is `rbinom`. To simulate $n = 1$ observation from one experiment of size $N = 10$ and a probability of success $p = 0.2$, try the following:

```
> rbinom(n=1,size=10,prob=0.4)
```

```
[1] 4
```

In many cases we want to make such simulations several times. To conduct the experiment, say, 15 times we can do:

```
> rbinom(n=15,size=10,prob=0.2)
```

```
 [1] 3 2 3 0 2 0 3 4 1 3 2 3 3 3 1
```

Sampling from a Bernoulli distribution (which is just a $\text{Bin}(1, p)$–distribution) is therefore achieved by

```
> rbinom(n=15,size=1,prob=.2)
```

```
 [1] 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0
```

or simply

```
> rbinom(15,1,.2)
```

```
 [1] 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0
```

For more information on `rbinom` type `?rbinom`. Similarly, random normal and Poisson variates are generated using `rnorm` and `rpois`. For information on these, type `?rnorm` or `?rpois`.

If you want to take a random sample from the elements of a vector you need the function `sample`. First look at the vector from 1 to 10:

```
> 1:10
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
> sample( 1:10, 8, replace=T )
```

```
[1]  7  2  7  2  5  5  8 10
```

Here we took a sample of 8 from the vector $(1, 2, \ldots, 10)$, *with* replacement. If you want a sample *without* replacement, just do:

```
> sample( 1:10, 8 )

[1]  6  4  1 10  8  9  7  2
```

If you omit the second argument, you just get a permutation of the input vector:

```
> sample( 1:10 )

 [1]  6  4  2  7  5  1 10  8  9  3


> sample( letters[1:8] )


[1] "g" "c" "d" "f" "e" "a" "b" "h"
```

1.4 Distributions in R

All the standard distributions are available in R; for example the probability density function for the normal distribution is called by `dnorm`, the cumulative distribution is called `pnorm`, the inverse of this `qnorm`, and a random sample from it generated by `rnorm`.

In general any distribution has the four functions `d`dist, `p`dist, `q`dist and `r`dist, associated with it.

There is a function in the `MASS` library (which is by default included in any R-installation) to generate random samples from a multivariate normal distribution, `mvrnorm`.

1.5 Using the interface to JAGS

This brief "Practice 0" is to get you familiar with the practicalities around running `JAGS` from within R and making sure that the installation on your computer works. It is not a proper exercise but meant for use as a check of your computing installation.

First you must load the `rjags` package, which should automatically find your `JAGS` installation:

```
> library(rjags)
```

Now we choose a model that is so simple that we will know the exact form of the posterior distribution for the parameter of interest. This way we can check that the MCMC machinery gives numerical results that are consistent with the known theoretical posterior distribution for that parameter.

We are going to analyze the annual number of airline fatalities using a simple Poisson model and use this model to predict the future number of fatalities. This corresponds to the first part of exercise 6.

First get the data and take a look at it:

```
> airline <- read.csv( "../data/airline.csv" )
> airline
```

```
   year1975 year fatal  miles  rate
1         1 1976    24  3.863 6.213
2         2 1977    25  4.300 5.814
3         3 1978    31  5.027 6.167
4         4 1979    31  5.481 5.656
5         5 1980    22  5.814 3.784
6         6 1981    21  6.033 3.481
7         7 1982    26  5.877 4.424
8         8 1983    20  6.223 3.214
9         9 1984    16  7.433 2.152
10       10 1985    22  7.107 3.096
11       11 1986    22  9.100 2.418
12       12 1987    25 10.000 2.500
13       13 1988    29 10.600 2.736
14       14 1989    29 10.988 2.639
15       15 1990    27 10.880 2.482
16       16 1991    29 10.633 2.727
17       17 1992    28 11.956 2.342
18       18 1993    33 12.343 2.674
19       19 1994    27 13.011 2.075
20       20 1995    25 14.220 1.758
21       21 1996    24 16.371 1.466
22       22 1997    26 15.483 1.679
23       23 1998    20 18.080 1.106
24       24 1999    21 16.633 1.263
25       25 2000    18 18.875 0.954
26       26 2001    13 19.233 0.676
```

We shall only be interested in the column `fatal` which contains the annual number of fatalities. We use the following simple model to describe the number of fatalities in year i, y_i:

$$y_i|\mu \sim \text{Poisson}(\mu), \quad \mu \sim \Gamma(0,0)$$

The $\Gamma(0.01, 0.01)$ is almost a uniform distribution on $(0, +\infty)$, (so a largely uninformative prior that acts as an approximation to the $\Gamma(0,0)$ prior in the model specification above); the posterior for μ will be $\Gamma(0 + \sum y_i, 0 + n)$ where n is the number of observations, in this case 26, and $\sum y_i = 634$:

```
> nrow( airline )

[1] 26

> sum( airline$fatal )

[1] 634
```

Since we know the posterior distribution, we can compute the mean and median of this by simulating a sample of, say, 1000 from it:

```
> ( mn <- mean( xx <- rgamma( 10000, 634.01, 26.01 ) ) )

[1] 24.35343

> ( md <- median( xx ) )

[1] 24.34909
```

We can also draw the posterior distribution for μ, with indication of the mean and median:

```
> curve( dgamma( x, 634.01, 26.01 ), from=20, to=30, lwd=2 )
> abline( v=mn, col="red", lwd=3 )
> abline( v=md, col="blue" )
```

1.5.1 Using JAGS via rjags

In order to run BUGS we must (i) supply the data; (ii) formulate the model as a BUGS program; and (iii) specify how the sampling from the chains should be done.

Data The first thing to provide to JAGS is the data. This is provided in the form of a named list, one element per data-structure (usually vector or matrix). In this case we provide the vector of fatal airline accidents expanded with a NA for prediction of the number in 2002, as well as the total number of observations:

```
> a.dat <- list( fatal = c(airline$fatal,NA), I=27 )
```

Program specification of model The program specifying the model (BUGS code) must be put in a separate file which is then read by JAGS. When working in R this is most conveniently done using the R-function `cat()` which behaves pretty much like `paste()` with the exception that the result is not a character object but directly written to a file you specify. If you specify `file=""` the output is sent to the screen.

Here is the BUGS code specifying the above model, using `cat` to put it in the file `m1.jag`:

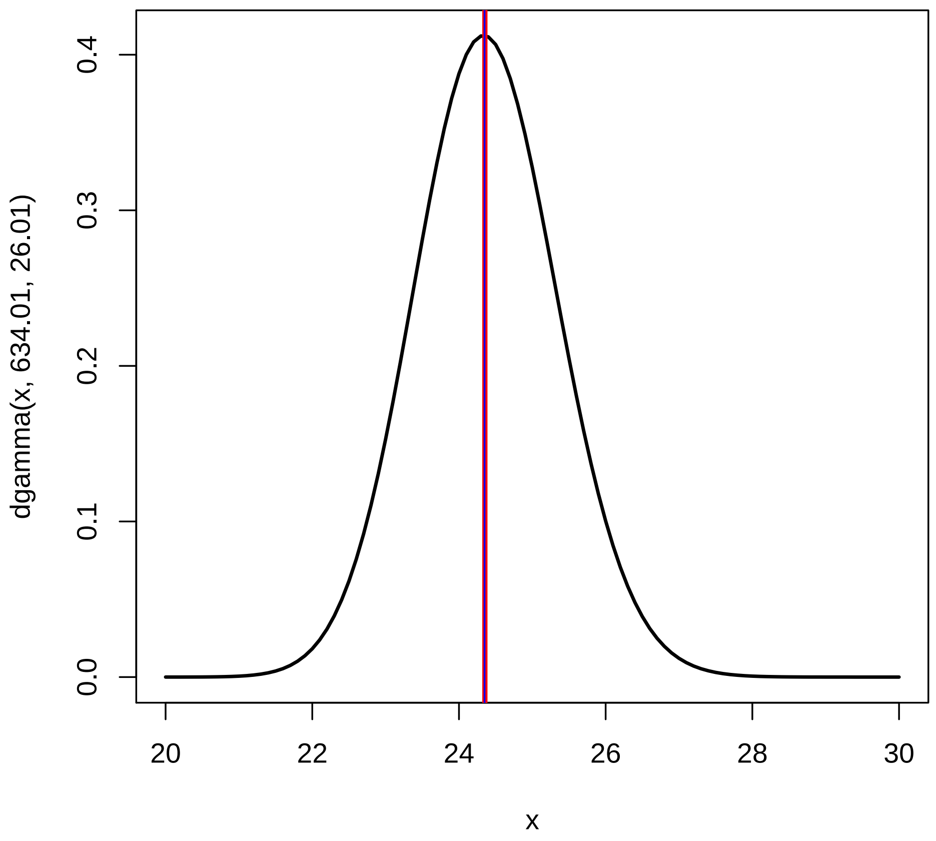


Figure 1.1: *The posterior distribution for mu. Mean is the red line, median the blue.*

```
> cat( "model
+       {
+       for( i in 1:I )
+         {
+         fatal[i] ~ dpois(mu)
+         }
+       mu ~ dgamma(0.1,0.1)
+       }",
+      file="m1.jag" )
```

The code refers to data points in the variable `fatal` which is `I` long. The `BUGS` language is *declarative*, i.e. it is not executed as the program runs. Instead it is a specification of the model structure, and *after* the model is set up `BUGS` will decide how best to go about the MCMC-simulation. So it would not matter if the specification of a prior of `mu` was put before the `for` statement. Also the loop is just a compact way of writing `fatal[1]   dpois(mu)`, `fatal[2]   dpois(mu)`, `fatal[3]   dpois(mu)` etc.

We could have replaced `I` with the number 27 in the code if we wanted. In that case the `I` in the data would have been superfluous. It is, however, good practice to express model quantities as variables rather than fixed values since this makes implementing data updates much easier.

Starting values To start the MCMC simulation we will normally supply some starting values (in most cases `JAGS` will however be able to generate them). In order to be able to monitor convergence we will normally run several chains, so we must supply starting values for each chain. The starting values for one chain is a named list, names are the parameters used in the model. Here we use three chains, hence the initial values is a list of three lists. Each of these list has as elements one named value for each parameter — in this case there is only one parameter μ, called `mu` in the `BUGS` program:

```
> a.ini <- list( list( mu=20 ),
+                list( mu=23 ),
+                list( mu=26 ) )
```

Note that we specify a list with three elements as we intend to run 3 parallel chains.

Compiling and adapting Once these structures have been set up we ask `JAGS` to compile the model and run the chains for a number of cycles (“burn-in”) so that the model is (hopefully) in a stable state, that is, converged to sampling from a stationary process that represents the target distribution, namely the joint posterior distribution for the unobserved quantities (stochastic nodes) in the model. In this case we ask for 3 chains and 2000 cycles of burn-in:

```
> m <- jags.model( file = "m1.jag",
+                  data = a.dat,
+              n.chains = 3,
+                 inits = a.ini,
+               n.adapt = 2000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 30

Initializing model
```

Parameters and simulation parameters Once the model is set up and “burnt in”, we can run the chain using `coda.samples`, which not surprisingly produces an object of class `mcmc.list` that can be manipulated by the functions in the `coda` package.

We must specify:

the variables (nodes) that we want to monitor in the subsequent cycles of the chain. This is done using the argument `variable.names` (which can be abbreviated to `var` if you wish).

how many cycles (iterations) to run the chain (`n.iter`)

how often we sample the parameters specified and retain the results in memory (`thin`)

In this case we run 10,000 cycles of the three chains, and sample every 10th value of μ, so we get 1000 samples from each chain, a total of 3000 samples from the posterior of the parameter(s):

```
> res <- coda.samples( m,
+                     var = "mu",
+                  n.iter = 10000,
+                    thin = 10 )
```

The resulting object is of class `mcmc.list`; in this case a list with 3 elements (one per chain). Each element of the list is a 1000×1 matrix.

1.5.2 Results

First we inspect what type of R-structure was returned by `coda.samples`:

```
> class( res )
```

```
[1] "mcmc.list"
```

```
> str( res )
```

```
List of 3
 $ : mcmc [1:1000, 1] 22.9 24.9 25.5 25.8 23.9 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr "mu"
  ..- attr(*, "mcpar")= num [1:3] 10 10000 10
 $ : mcmc [1:1000, 1] 24.6 25.5 23.4 24.3 25.1 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr "mu"
  ..- attr(*, "mcpar")= num [1:3] 10 10000 10
 $ : mcmc [1:1000, 1] 23 24.8 25.7 24.8 24.7 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr "mu"
  ..- attr(*, "mcpar")= num [1:3] 10 10000 10
 - attr(*, "class")= chr "mcmc.list"
```

The `mcpar` attribute of each of the list members are the first, last and step in the sampling of the chains.

As always in R, the most useful overview comes from the `summary` function:

```
> summary( res )


Iterations = 10:10000
Thinning interval = 10
Number of chains = 3
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
      24.28784        0.94664        0.01728        0.01721

2. Quantiles for each variable:

 2.5%   25%   50%   75% 97.5%
22.47 23.63 24.29 24.90 26.16
```

If we decide that the set of samples from the 3 chains provides a reasonable representation of the posterior distribution, we can get an overview of the three chains by using the function `plot.mcmc.list`:

```
> par( mfrow=c(1,2) )
> plot( res )
```

Since the model is so simple that we know the theoretical from of the posterior, we can add this curve to the plot in red, say:

```
> curve( dgamma( x, 634.01, 26.01 ), from=20, to=30, lwd=2, col="red", add=TRUE )
```

If we want a simpler structure to work with, we can collect all the posterior samples from the different chains in one matrix:

```
> rmat <- as.matrix( res )
> head( rmat )


           mu
[1,] 22.93417
[2,] 24.93321
[3,] 25.52607
[4,] 25.79572
[5,] 23.93313
[6,] 23.44019
```

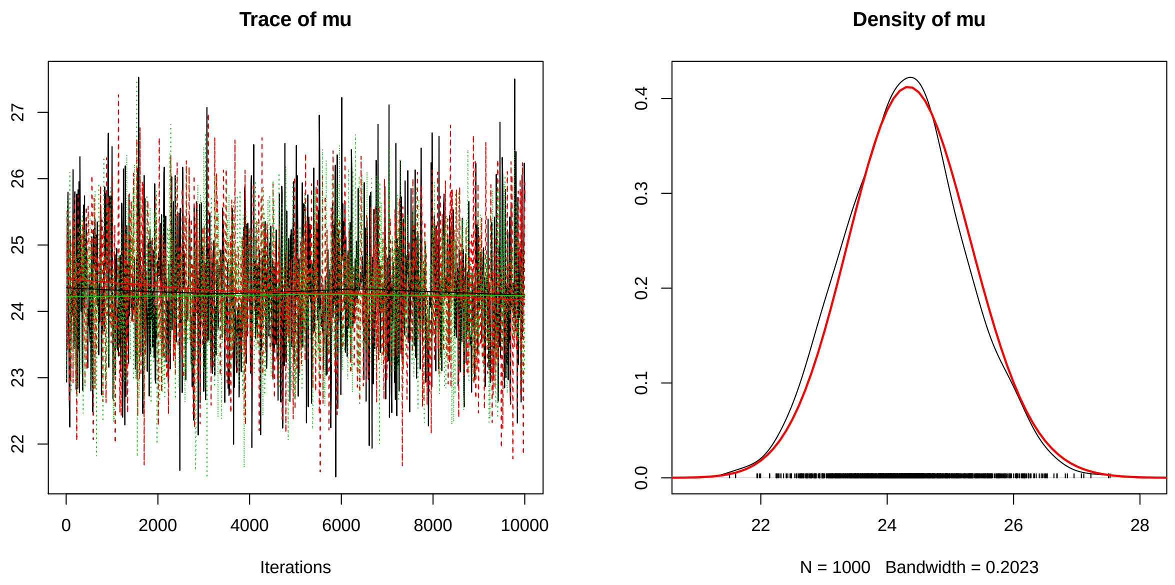


Figure 1.2: *Trace of the chains (left) and density of the posterior overlaid with the theoretical posterior.*

Chapter 2

Exercises

2.1 Bayesian inference in the binomial distribution

This exercise illustrates the prior to posterior calculations in the simple example of to inference about an unknown binomial probability, θ.

1. First, suppose that only a finite number of possible values for the true proportion θ are possible, *e.g.* $(\theta_1, \theta_2, \ldots, \theta_J)$, with prior probabilities $p(\theta_j)$, where $\sum_j p(\theta_j) = 1$. For a single Bernoulli trial $y \in (0, 1)$, the likelihood for each value for θ is given by

   $$p(y|\theta_j) = {\theta_j}^y (1 - \theta_j)^{1-y},$$

   For an outcome y, Bayes' theorem combines the discrete prior distribution with the likelihood to generate posterior probabilities for the θ_j:

   $$p(\theta_j|y) \propto {\theta_j}^y (1 - \theta_j)^{1-y} \times p(\theta_j),$$

   To get the proper posterior distribution, you have to normalize the r.h.s., that is divide by the sum.

   If have a binomial observation, i.e. x events out of n trials, then the posterior will be:

   $$p(\theta_j|x) \propto \theta_j^x (1 - \theta_j)^{n-x} \times p(\theta_j).$$

   (a) Suppose a drug has an unknown true response rate θ, and for simplicity assume that θ can only take one of the values $\theta_1 = 0.2$, $\theta_2 = 0.4$, $\theta_3 = 0.6$ or $\theta_4 = 0.8$, and that we adopt the "neutral" position of assuming each value θ_j is equally likely, i.e. $p(\theta_j) = 0.25$ for each $j = 1, 2, 3, 4$.

   If we observe onle one person with a positive response ($y = 1$). How should our belief in the possible values be revised? Use this table to update from the prior to the posterior:

		Prior	Likelihood	Likelihood $\times$ prior	Posterior
j	θ_j	$p(\theta_j)$	$p(y\|\theta_j)$	$p(y\|\theta_j)p(\theta_j)$	$p(\theta j\|y)$
1	0.2	0.25			
2	0.4	0.25			
3	0.6	0.25			
4	0.8	0.25			
	$\sum_j$	1.0			1.0

(b) If we instead of one patient had observations on $n = 20$ persons out which $x = 15$ had a positive response, how would the posterior look? Use that same table to complete the computations:

j	θ_j	Prior $p(\theta_j)$	Likelihood $p(y\|\theta_j)$	Likelihood $\times$ prior $p(y\|\theta_j)p(\theta_j)$	Posterior $p(\theta j\|y)$
1	0.2	0.25			
2	0.4	0.25			
3	0.6	0.25			
4	0.8	0.25			
	$\sum_j$	1.0			1.0

(c) Suppose we had given non-zero prior probability to the extreme values of $\theta = 0, 1$ (that is, the drug either never or always workes). The prior distribution is then on the six values $\theta_1 = 0$, $\theta_2 = 0.2$, $\theta_3 = 0.4$, $\theta_4 = 0.6$, $\theta_5 = 0.8$ or $\theta_6 = 1.0$, with $p(\theta_j) = 1/6$.

Describe *qualitatively* how the results in the table in part (a) would change if we used this discrete prior distribution on 6 values for θ for the same data, that is, 15 successes out of 20 trials. Uste this table for the calculations:

j	θ_j	Prior $p(\theta_j)$	Likelihood $p(y\|\theta_j)$	likelihood $\times$ prior $p(y\|\theta_j)p(\theta_j)$	Posterior $p(\theta j\|y)$
0	0.0	1/6			
1	0.2	1/6			
2	0.4	1/6			
3	0.6	1/6			
4	0.8	1/6			
5	1.0	1/6			
	$\sum_j$	1.0			1.0

(d) How would the results change if we used the data in the example in the module notes, that is, we had just one success from one trial?

You can use this table for the calculations:

j	θ_j	Prior $p(\theta_j)$	Likelihood $p(y\|\theta_j)$	likelihood $\times$ prior $p(y\|\theta_j)p(\theta_j)$	Posterior $p(\theta j\|y)$
0	0.0	1/6			
1	0.2	1/6			
2	0.4	1/6			
3	0.6	1/6			
4	0.8	1/6			
5	1.0	1/6			
	$\sum_j$	1.0			1.0

(*Hint*: It is not necessary to actually calculate the posterior probabilities explicitly. Try considering the value of the likelihood for each value of θ and the impact that the two new values of the likelihood for $\theta = 0$ and $\theta = 1$ will have on the calculations.

2. In the analysis above, for simplicity, we assumed that θ can could only take one of the values (0), 0.2, 0.4, 0.6, 0.8, (1).

   Now suppose that previous experience with similar compounds has suggested that response rates between 0.2 and 0.6 could be feasible, with an expectation around 0.4. If we want a continuous prior distribution on the interval $(0, 1)$, we should choose one with mean 0.4 and say 95% of the probability mass in the interval (0.2,0.6), or more *ad hoc*, with a standard deviation of 0.1.

   (a) We choose a Beta(a, b) as prior. From the properties of the beta distribution we know that mean m and standard deviation s are:
   $$m = \frac{a}{a+b} \tag{2.1}$$
   $$s = \sqrt{\frac{m(1-m)}{a+b+1}} \tag{2.2}$$
   The expression in equation (2.2) can be rearranged to give $a + b = \big(m(1-m)/s^2\big) - 1$. Now use the target values $m = 0.4$ and $s = 0.1$ to obtain a value for $a + b$, and the formula for m to get separate values for a and b.

   (b) Make a graph of the prior distribution for p, the success probability. The Beta-density is available in R as the function `dbeta`. You would need to type `?dbeta` to get the help function up.

   (*Hint:* You can generate a vector of say 200 equidistantly spaced points between 0 and 1 by `seq(from=0,to=1,length=200)`.

   (c) Suppose we observe $x = 15$ successes out of $n = 20$ trials. Make a graph of the likelihood for this observation. The binomial density is available in R as `dbinom`.

   (d) From the prior distribution for the parameter and the likelihood we can form the posterior by taking the product. We know from lectures that the parameters of the beta distribution are updated to $[a^\star, b^\star]$ where $a^\star = a + x$ and $b^\star = b + (n - x)$.

   Now make a third graph of the posterior for the success probability.

   (e) Plot the three curves in one graph, using `par(mfrow=c(3,1))` before running the three plot statements.

   (f) (*Complicated, but illustrative*) Pack the generation of the three graphs into an R-function that takes m, s (mean and standard deviation of the prior), x and n (the observed data) as arguments, and observe how the posterior changes when changing the prior and the data.

3. The French mathematician Pierre-Simon Laplace (1749–1827) was the first person to show definitively that the proportion of female births in the French population was less then 0.5, in the late 18th century, using a Bayesian analysis based on a uniform prior distribution (see Gelman *et al*; p.34). Suppose you were doing a similar analysis but you had more definite prior beliefs about the ratio of male to female births. In particular, if θ represents the proportion of female births in a given population, you are willing to place a Beta(100,100) prior distribution on θ.

(a) Show that this means you are more than 95% sure that θ is between 0.4 and 0.6, although you are ambivalent as to whether it is greater or less than 0.5.

(b) Now you observe that out of a random sample of 1,000 births, 511 are boys. What is your posterior probability that $\theta > 0.5$?

2.2 Simple linear regression with `JAGS`

The pupose of this exercise is to introduce the use of `JAGS` as a machinery for estimation in standard statistical models. This is done using a simple linear regression example. The model we will use is the simple linear regression model:

$$y_i = \alpha + \beta x_i + e_i, \qquad e_i \sim \mathcal{N}(0, \sigma^2)$$

assuming that the e_is are independent.

1. To make thinge easier for a start, use a set of bogus data for the analysis:

   ```
   > x <- c(1,2,3,4,5,6)
   > y <- c(1,3,3,3,5,7)
   ```

   Plot them and make s standard linear regression using `lm()` from R: What are the estimates of intercept, slope and residual standard deviation in this model?

   Provide confidence intervals for α and β.

2. The next step is to use `JAGS` to estimate in the model. So referring to the section introducing `JAGS`, you should set up the following structures in R before invoking `JAGS`:

   - Data — a list.
   - Initial values — a list of lists.
   - Parameters to monitor — a character vector.
   - A file with the `JAGS` program.

   In the program you must specify the model in terms of the three parameters of the model and the 6 observations of y and x. You should also specify the prior distributions of the parameters α, β of σ. Use uninformative priors for all three; that is normal priors with large variance for α and β, whereas a unform prior on some suitably large interval ([0,100], say) for σ is recommendable.

   Compile and initialize using 10000 cycles as burn-in.

   Run the program for 10000 iterations with 3 chains, sampling say every 10th value.

   Hint: For your convenience we have put up a file with a skeleton for what you need to do when running an analysis with `JAGS` as
   `http://bendixcarstensen.com/Bayes/Cph-2012/pracs/jags.skeleton.rnw`

3. Inspect the posterior using `summary`. Remember to load the `coda` package first. Compare the posterior medians and central 95% posterior intervals with the estimates and confidence intervals derived.

   How well do they agree? Why / why not?

4. Now try to do the same on a real dataset. In the `Epi` package is a datset, `births` which has data on 500 births in London, notably the birthweigst (`bweight`) and gestational age (`gestwks`). We will set up a rather naive regression model with a linear relationship between x, number of gestational weeks and y birthweight.

   Load the data and get the subset where the explanatory variable is non-missing:

```
> library( Epi )
> data( births )
> births <- subset( births, !is.na(gestwks) )
```

You can re-use the set-up from the previous question to get classical regression estimates and estimates from the Bayesian machinery and compare them.

5. How do the classically derived confidence intervals agree with the posterior central intervals?

2.3 Examples of the Gibbs sampler and Metropolis Hastings algorithm

1. Consider a single observation (y_1, y_2) from a bivariate normally distributed population with mean $\theta = (\theta_1, \theta_2)$ and known covariance matrix $\begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$. With a uniform prior distribution on θ, the posterior distribution is

$$\begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} | y \sim \mathrm{N}\left(\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \right).$$

Although it is simple to draw directly from the joint posterior distribution of (θ_1, θ_2), we set up the Gibbs sampler explicitly here for the purpose of illustration. To apply the Gibbs sampler to (θ_1, θ_2), we need the conditional posterior distributions.

(a) Use the properties of the multivariate normal distribution (either (A.1) or (A.2) on page 579 of **BDA**) to show that the relevant conditional distributions are

$$\begin{aligned} \theta_1 | \theta_2, y &\sim \mathrm{N}(y_1 + \rho(\theta_2 - y_2), 1 - \rho^2), \\ \theta_2 | \theta_1, y &\sim \mathrm{N}(y_2 + \rho(\theta_1 - y_1), 1 - \rho^2). \end{aligned}$$

(b) The Gibbs sampler proceeds by alternately sampling from these two normal distributions. In general we would say that the natural way to start the iterations would be with random draws from a normal approximation to the posterior distribution; of course, such draws would eliminate the need for iterative simulation in this trivial example!

Use the conditional distributions for θ_1 and θ_2 with $(y_1, y_2) = (0, 0)$ and $\rho = 0.8$ to set up a simple Gibbs sampler in R. Use two vectors, one for θ_1 called `theta1` and one for θ_2 called `theta2`, and start by setting the all the elements of each of `theta1` and `theta2` to 0:

```
> numsims <- 1000
> rho <- 0.8
> theta1 <- numeric(numsims)
> theta2 <- numeric(numsims)
```

Now amend the first value of `theta1` to -3 and sample a single value from the conditional distribution of θ_2 given θ_1 and set this as the first element of `theta2`:

```
> theta2[1] <- rnorm( 1, mean=rho*theta1[1], sd=sqrt(1 - (rho^2)) )
```

Now use a loop to iterate the process of sampling from the conditional distribution of θ_2 given θ_1 and *vice versa*:

```
> for(i in 2:numsims)
+ {
+ theta1[i] <- rnorm( 1, mean=rho*theta2[i-1], sd=sqrt(1-(rho^2)) )
+ theta2[i] <- rnorm( 1, mean=rho*theta1[i]  , sd=sqrt(1-(rho^2)) )
+ }
```

Generate 1000 values for each of θ_1 and θ_2 using the Gibbs sampling routine from part (b) of the question. Calculate the sample mean and standard deviation of the final 500 realised values for each of θ_1 and θ_2. Show that these empirical values for the mean and standard deviation are close to the theoretical values for the posterior marginal distributions of θ_1 and θ_2 based on the joint posterior distribution displayed above:

```
> mean(theta1[501:1000])
> mean(theta2[501:1000])
> sqrt(var(theta1[501:1000]))
> sqrt(var(theta2[501:1000]))
```

Also check that the correlation between the two sequences is close to the true value of 0.8:

```
> cor( theta1[501:1000], theta2[501:1000] )
```

2. We can also use the Metropolis-Hasting algorithm to sample from the posterior distribution. For the proposal distribution $h()$ we use the uncorrelated bivariate normal distribution. Implement this in R by working through the following.

   Set the correlation to $\rho = 0.7$, the number of simulation `nsim` to 1000, initialise a matrix `ans` with 1000 rows and 2 columns that will hold the results of the simulation and set up the 2×2 correlation matrix `Sigma` and its inverse `SigmaInv`:

```
> rho  <- 0.7
> nsim <- 1000
> ans  <- matrix(NA, nr=nsim, nc=2)
> Sigma     <- matrix(c(1,rho,rho,1), nr=2)
> SigmaInv  <- solve(Sigma)
```

   We start the simulation at `x1` = `x2` = 30 and set up a vector `xcurr` that holds the current values of `x1` and `x2`:

```
> x1 <- x2 <- 30
> xcurr <- c(x1,x2)
```

   Initialise an "acceptance vector" called `accept` to 0 and the standard deviation `sigma` of the proposal distribution to 2. Run `nsim` iterations and at each iteration, generate a proposal called `xprop` by adding a normal random variate with mean 0 and standard deviation 2 to the current value. Calculate the log-likelihood for both the current and proposed values and accept this with the appropriate probability. If the proposal is accepted, the correspondign component of the `accept` vector is set to 1 (in fact "TRUE"), otherwise 0 ("FALSE"):

```
> accept <- numeric(nsim)
> sigma <- 2
> for (ii in 1:nsim){
+   xprop  <- xcurr + rnorm(2, mean = 0, sd = sigma)
+
+   logkxprop <- - t(xprop) %*% SigmaInv %*% xprop /2
+   logkxcurr <- - t(xcurr) %*% SigmaInv %*% xcurr /2
+
+   alpha <- min(1, exp(logkxprop-logkxcurr))
+   u <- rnorm(1)
+
+   if ( accept[ii] <- (u<alpha) ){
+     xaccept <- xprop
+   } else {
+     xaccept <- xcurr
+   }
+
+   ans[ii,] <- xaccept
+   xcurr <- xaccept
+ }
> cat("Accepted proposals: ", sum(accept)/nsim, "\n")
```

Now plot all samples:

```
> pairs(ans)
```

Plot the two series of values (`x1` and `x2`) to determine the number of iterations that we need to use as the burn-in:

```
> matplot(ans, type='l')
```

It looks like it is sufficient to discard the first 100 samples as the burn in:

```
> pairs(ans[-(1:100),])
```

We can check dependencies among each of the series for `x1` and `x2` using the autocorrelation functions `pacf` (for *partial* autocorrelation) and `acf`:

```
> par( mfrow=c(2,2) )
> pacf(ans[,1])
> pacf(ans[,2])
>  acf(ans[,1])
>  acf(ans[,2])
```

You should investigate the effect of changing

(a) The value of the correlation parameters ρ.

(b) The mean of the proposal distribution.

(c) The standard deviation of the proposal distribution.

3. It's instructive to compare the bivariate sampler above to a single component Metropolis–Hastings sampler where the proposal for $h(x_2|x_1^t, x_2^t)$ is $x_2 = x_2^t + \epsilon$ where $\epsilon \sim N(0, \sigma^2)$ for some choice of σ^2 and likewise for x_1. The set up is the same:

```
> rho  <- 0.7
> nsim <- 1000
> ans  <- matrix(NA, nr=nsim, nc=2)
> x1 <- x2 <- 30
> xcurr <- c(x1,x2)
```

We now need two counters, one for each component of the vector containing the values of `x1` and `x2`. We need to calculate the log-likelihood of the conditional distribution of `x1` given `x2` for both the current and proposed value of `x1` and proposal (the quantities `logpx1prop` and `logpx1`, along with the unconditional log-likelihoods `hx1prop` and `hx1`, all of which are used in generating the ratio governing the acceptance probability. We run through the same routine for `x2`.

```
> accept1 <- accept2 <- numeric(nsim)
> sigma <- 5
> for (ii in 1:nsim){
+
+   # Update x1:
+   x1prop <- rnorm(1, mean=x1, sd=sigma)
+
+   logpx1prop  <- -(x1prop-rho*x2)^2/(1-rho^2)
+   logpx1      <- -(x1-rho*x2)^2/(1-rho^2)
+
+   hx1prop <- dnorm(x1prop, mean=x1, sd=sigma)
+   hx1     <- dnorm(x1, mean=x1prop, sd=sigma)
+
+   alpha <- min(1, exp(logpx1prop-logpx1)*(hx1/hx1prop))
+   u     <- rnorm(1)
+
+   if ( accept1[ii] <- (u<alpha) ){
+     x1 <- x1prop
+   }
+
+   # Update x2:
+   x2prop <- rnorm(1, mean=x2, sd=sigma)
+
+   logpx2prop  <- -(x2prop-rho*x1)^2/(1-rho^2)
+   logpx2      <- -(x2-rho*x1)^2/(1-rho^2)
+
+   hx2prop <- dnorm(x2prop, mean=x2, sd=sigma)
+   hx2     <- dnorm(x2, mean=x2prop, sd=sigma)
+
+   alpha <- min(1, exp(logpx2prop-logpx2)*(hx2/hx2prop))
+   u     <- rnorm(1)
+
+   if ( accept2[ii] <- (u<alpha) ){
+     x2 <- x2prop
+   }
+   ans[ii,] <- c(x1,x2)
+ }
> cat("Accepted proposals, x1: ", sum(accept1)/nsim, "x2:", sum(accept2)/nsim, "\n")
```

Once again we can plot all the samples:

```
> pairs(ans)
```

Check the number of iterations that we need to discard as a burn-in:

```
> matplot(ans, type='l')
```

Let's discard the first 100 samples:

```
> pairs(ans[-(1:100),])
```

Have a look at the cumulative acceptance probabilities for `x1` and `x2`:

```
> plot( 1:nsim,cumsum(accept1)/1:nsim, ylim = c(0,1), pch = "",
+       xlab = "Iteration Number", ylab = "Probability")
> lines(1:nsim,cumsum(accept1)/1:nsim, ylim = c(0,1), lwd = 3)
> title(main = "Cumulative acceptance probability", cex = 0.5)


> plot( 1:nsim,cumsum(accept2)/1:nsim, ylim = c(0,1), pch = "",
+       xlab = "Iteration Number", ylab = "Probability")
> lines(1:nsim,cumsum(accept2)/1:nsim, ylim = c(0,1), lwd = 3)
> title(main = "Cumulative acceptance probability", cex = 0.5)
```

Also let's plot the two series `x1` and `x2` against each other (change the value of the standard deviation in the simulations above to see the jumps get bigger or smaller):

```
> plot(ans[,1],ans[,2],ylim = c(-50,50),xlim = c(-50,50), xlab = "x1", ylab = "x2")
> lines(ans[,1],ans[,2],lwd = 1)
> title(main = "Metropolis-Hastings sampler s.d. = 2")
```

Finally check the dependencies within each of the `x1` and `x2` series:

```
> par( mfrow=c(2,2) )
> pacf(ans[,1])
> pacf(ans[,2])
> acf(ans[,1])
> acf(ans[,2])
```

Consider the following questions:

(a) What the is cumulative acceptance probability after 1000 simulations? How many simulations are before the acceptance ratio stabilises?

(b) Explore how changing the standard deviation of the proposal distributions alters

   i. the cumulative acceptance ratio,

   ii. the number of iterations required to achieve convergence and a stable acceptance ratio,

   iii. the visual appearance of the sample path of the bivariate plot.

2.4 Estimating the speed of light

Simon Newcomb set up an experiment in 1882 to measure the speed of light. Newcomb measured the amount of time required for light to travel 7442 metres. The measurements are here (copy-paste from the document):

```
> newcomb <-
+ c(28, 26, 33, 24, 34, -44, 27, 16, 40, -2, 29, 22, 24, 21, 25,
+ 30, 23, 29, 31, 19, 24, 20, 36, 32, 36, 28, 25, 21, 28, 29, 37,
+ 25, 28, 26, 30, 32, 36, 26, 30, 22, 36, 23, 27, 27, 28, 27, 31,
+ 27, 26, 33, 26, 32, 32, 24, 39, 28, 24, 25, 32, 25, 29, 27, 28,
+ 29, 16, 23)
```

The numbers are lifted from Stigler SM. (1977): Do robust estimators work with real data? (with discussion). *Annals of Statistics* **5**, 1055-1098. The data are times for light to travel a fixed distance, recorded as deviations from 24,800 nanoseconds.

1. Make a histogram of the data — use the argument `breaks=50`, in order to get a detailed impression. What do you see?

2. We want to apply the normal model, assuming that all 66 measurements are independent draws from a normal distribution with mean μ and variance σ^2. The main goal is posterior inference for μ as an estimate of the speed of light (suitably transformed).

   First compute the sample mean and standard deviation.

3. If we assume a non-informative prior distribution for $p(\mu, \sigma^2) \propto (\sigma^2)^{-1}$ (which is equivalent to a joint uniform prior distribution on $(\mu, \log \sigma)$), the posterior distribution of μ has the form

$$\left.\frac{\mu - \overline{y}}{s/\sqrt{n}}\right| \sim t_{n-1}. \tag{2.3}$$

   Note that only μ is unknown in the expression above since we are conditioning on the observed values of the sample mean $\overline{y}$, the sample standard deviation s and the sample size n. Use this distributional result to calculate a 95% central posterior interval for μ.

4. The posterior interval can also be obtained by simulation. Following the factorisation of the posterior distribution given in lectures as

$$\begin{aligned} p(\mu|\sigma^2, y) &\sim \mathrm{N}(\overline{y}, \sigma^2/n) \\ p(\sigma^2|y) &\propto (\sigma^2)^{-(n+1)/2} \exp\left(-\frac{(n-1)s^2}{2\sigma^2}\right), \end{aligned}$$

   which is a scaled inverse-χ^2 density:

$$p(\sigma^2|y) \sim \chi^{-2}(n-1, s^2),$$

   First draw a random value of $\sigma^2 \sim \chi^{-2}(65, s^2)$ as $65s^2$ divided by a random draw from the χ^2_{65} distribution. Then given this value of σ^2, we draw μ from its conditional posterior distribution, $\mathcal{N}(26.2, \sigma^2/66)$.

   Use R to carry out these simulation steps (for 1,000 iterations) and generate a vector of sampled values for the mean μ and the standard deviation σ. What are the 5 and 95% quantiles of these?

5. Check the results in the previous questions by setting up a model in JAGS. Set up data nodes `y`, and choose vague priors for μ and σ. So you set up the whole macinery, for example by suitable modifying the file `jags.skeleton.txt`.

   What are the posterior predictive interval for μ?

6. Based on the currently accepted value of the speed of light, the “true value” for μ in Newcomb’s experiment would be 33.0. How does this conform with the posterior sample?

7. One way to check the suitability of the model is to amend the `JAGS` code from question 3 so that it *generates* a vector `y.pred` of 66 observations from the normal distribution with the current sampled values of μ and σ. We can then ask JAGS to retain the smallest value from the vector `y.pred`, generating a distribution of minimum measurements for a sample of size $N = 66$.

   Extend your JAGS code with a node `smallest`, say, which holds the smallest of the predicted values — you will have to look up the function in the JAGS manual the function that retuns the mininum (have guess!).

8. Amend the model further to sample the two smallest predicted values and compare them with the ones actually present in the data. Is the predictive distribution for the smallest and second smallest observation under the model reasonable in relation to the data?

   See chapter 6 in Gelman *et al.* for an extensive discussion of such “posterior predictive checking”, in particular a more detailed treatment of the problem discussed here in section 6.3 pages 160–161.

2.5 Modelling the rate of airline fatalities 1976 to 2001

This exercise is based on exercises 2.13 and 3.12 from Gelman *et al.*. The original exercise has been extended to include additional data from 1986 to 2001. It is useful to read the partial solution to the original exercise 2.13 that appears in the most recent solutions file on Andrew Gelman's website, which is available as a PDF.

The data is available in the text file `airline.txt` with column names in the first line, aimed a reading into `R`. It is easier to work with distances in units of 10^{11} miles, which is how the passenger miles and accident rate data are presented in both source files (`.odc` and `.txt`).

The file `sol6a.R` contains an R-program that read data, produces all the relevant plots suggested in the following exercise. The R-file also contains specifications of the models used in `BUGS` and calls to `WinBUGS` using the package `R2WinBUGS`.

1. The simplest model: All years look the same.

   (a) Assume that the numbers of fatal accidents in each year are independent with a Poisson(θ) distribution. Set a (noninformative) gamma prior distribution for θ and determine theoretically using the results in lectures the posterior distribution based on the data from 1976 through 2001.

   (b) In this case it is also possible to determine theoretically the predictive distribution for the number of fatal accidents in 2002 - what is it? (See Section 2.7 page 53 of Gelman *et al.*).

   (c) How can we use the posterior distribution for θ and the assumption about the distribution of the number of fatal accidents to construct a two-stage process to draw samples from the predictive distribution for the number of fatal accidents in 2002?

   (d) If we set up a node in `BUGS` for year 2002 (*i.e.* adding an extra component to the data array for years 1976 to 2001 as has been done in the computing code provided) with the number of fatal accidents declared as "`NA`" (missing) will cause `BUGS` to draw from the predictive distribution for this node. What is the 95% predictive interval for the number of fatal accidents in 2002?

2. A model with constant *rate* of fatal airline crashes.

   (a) Now assume that the numbers of fatal accidents in each year follow independent Poisson distributions with a mean proportional to the number of passenger miles flown. Using the same noninformative prior distribution for θ determine the posterior distribution of the rate, i.e. accidents per passenger miles.

   (b) Modify your `BUGS` code from the previous question to accomodate this model, and use it to generate a 95% predictive interval for the number of fatal accidents in 2002 under the assumption that 2×10^{12} passenger miles were flown that year. (*Hint:* Note that you cannot stick an expression in as an argument to a distribution in `BUGS`; an expression as `fatal[i] dpois(lambda*miles[i])` will cause an error, so you will have to construct nodes for the mean, *e.g.* `mu[i] <- lambda * miles[i]; fatal[i]   dpois( mu[i] )`.)

Table 2.1: *Worldwide airline fatalities, 1976–2001. "Passenger miles" are in units of* 10^{11} *and the "Accident rate" is the number of fatal accidents per* 10^{11} *passenger miles. Source: International Civil Aviation Organization, Montreal, Canada (*`www.icao.int`*)*

Year	Fatal accidents	Passenger miles	Accident rate
1976	24	3.863	6.213
1977	25	4.300	5.814
1978	31	5.027	6.167
1979	31	5.481	5.656
1980	22	5.814	3.784
1981	21	6.033	3.481
1982	26	5.877	4.424
1983	20	6.223	3.214
1984	16	7.433	2.152
1985	22	7.107	3.096
1986	22	9.100	2.418
1987	25	10.000	2.500
1988	29	10.600	2.736
1989	29	10.988	2.639
1990	27	10.880	2.482
1991	29	10.633	2.727
1992	28	11.956	2.342
1993	33	12.343	2.674
1994	27	13.011	2.075
1995	25	14.220	1.758
1996	24	16.371	1.466
1997	26	15.483	1.679
1998	20	18.080	1.106
1999	21	16.633	1.263
2000	18	18.875	0.954
2001	13	19.233	0.676

3. We now expand the model by assuming that the number of fatal accidents in year t follows a Poisson distribution with mean $\alpha + \beta t$, *i.e.* independent of passengar miles but merely linearly decreasing by time.

   (a) Plot the number of fatal accidents each year over time to see that this was a dubious assumption even with the original data and is certainly not reasonable in light of the new data - why?

   (b) Moreover, a linear function of time t has the potential to generate negative values unless the parameters α and β are constrained - why is this a problem?

4. It would be more satisfactory to assume that the number of fatal accidents $y(t)$ in year t where $m(t)$ passenger miles were flown follows a Poisson distribution with

mean $\big(\exp(\alpha + \beta t)\big)m(t)$. This is a generalised linear model with canonical (log) link:

$$\begin{aligned} \mathrm{E}\big(y(t)|t, m(t)\big) &= \big(\exp(\alpha + \beta t)\big)m(t) && (2.4) \\ \log\Big(\mathrm{E}\big(y(t)|t, m(t)\big)\Big) &= \alpha + \beta t + \log(m(t)) && (2.5) \end{aligned}$$

(a) Calculate crude estimates and uncertainties for (α, β) using linear regression based on the relationship described above in equation (2.5), i.e. using the log-rates as reponse variable.

(b) Fit the generalized linear model using `glm` in R.

(c) Use the estimates from the maximum likelihood estimation as initial values to run the model in `BUGS` and to generate samples from the posterior distribution of α and β.

(d) Use the `xyplot.mcmc.list` function to check the mixing of the chains for α and β.

(e) Use the `densityplot.mcmc` function to display smoothed marginal posterior densities for α and β based on the sampled values of α and β. Also, make a scatter-plot showing the joint posterior distribution of α and β.

(f) Plot the posterior density for the *expected number* of fatal accidents in 2002, $\big(\exp(\alpha + 2002\beta)\big) \times m(2002)$ where we again assume the number of miles flown in 2002 is 2×10^{12}.

(g) Obtain the 95% predictive distribution interval for the *number* of fatal accidents in 2002.

(h) How would you define and derive the posterior predictive distribution of the number of fatalities in 2002, from the maximum likelihood approach?

2.6 Simple mixed model for fetal growth

The dataset `fetal.csv` contains measurements of head circumference and gestational age, as well as a transformation of gestational age:

```
> fetal <- read.csv("http://BendixCarstensen.com/Bayes/Cph-2012/data/fetal.csv",header=TRUE)
> str( fetal )
> head( fetal, 10 )
```

1. This is a so-called repeated measures dataset, we see that there are typically 4 or 5 measurements on each fetus, a few only have one measurement and some have as much as 7 measurements:

   ```
   > with( fetal, addmargins( table( table(id) ) ) )
   ```

2. We would like a description of the fetal growth as a linear function of time, but this is not a good description; a non-linear transformation of gestational age to make the relationship linear has been estimated: $\texttt{tga} = \texttt{ga} - 0.0116638 \times \texttt{ga}^2$; the transformed gestational age is for convenience put in the variable `tga`:

   ```
   > par( mfrow=c(1,2), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
   > with( fetal, plot( tga, ga-0.0116638*(ga^2), pch=16, cex=0.5 ) )
   > abline(0,1,col="red")
   > with( fetal, plot( ga, tga, pch=16, cex=0.5,
   +                    xlab="Gestational age (GA)", ylab="Transformed GA" ) )
   > abline(0,1,col="red")
   ```

3. The so called spaghetti-plot of a random sample of 100 of the 706 fetuses shows the linearizing effect of the transformation, but also that the square-root transformation of the head circumference makes the relationship more linear and more homogeneous with respect to the variance:

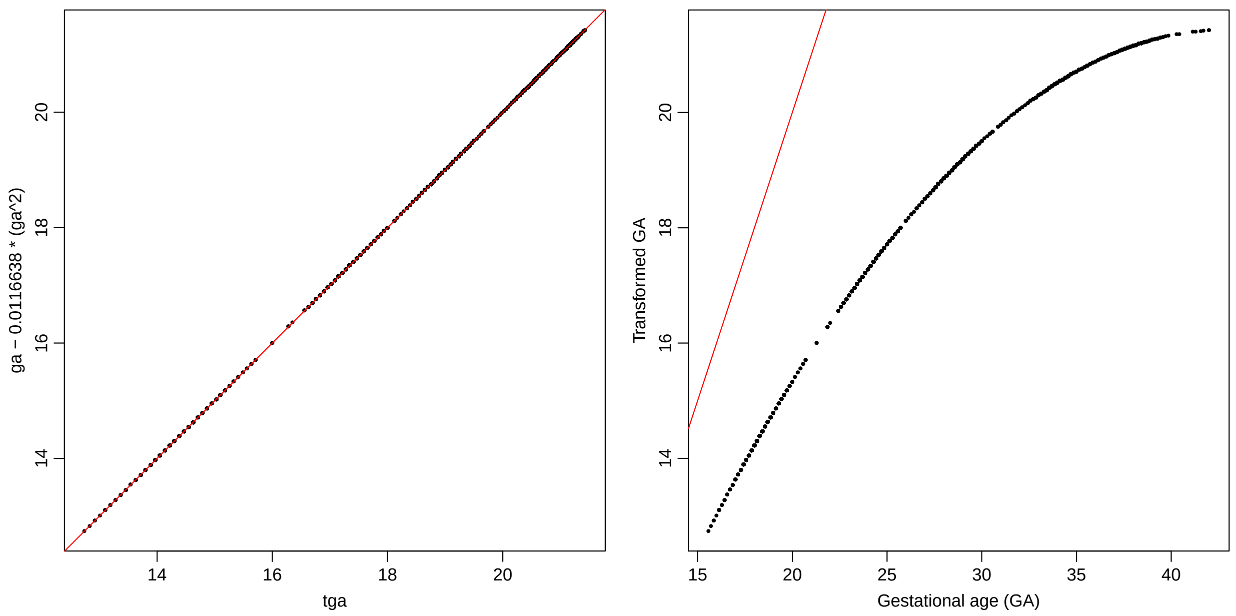


Figure 2.1: *Transformation used for gestational age. The red line is the identity line.*

```
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> id.sub <- sample( unique(fetal$id), 50 )
> with( fetal, plot( ga, hc, type="n" ) )
> for( i in id.sub ) with( subset(fetal,id==i), lines(ga,hc) )
> with( fetal, plot( tga, hc, type="n" ) )
> for( i in id.sub ) with( subset(fetal,id==i), lines(tga,hc) )
> with( fetal, plot( tga, sqrt(hc), type="n" ) )
> for( i in id.sub ) with( subset(fetal,id==i), lines(tga,sqrt(hc)) )
```

Also it appears that the overall variance is stabilized. The particular shape of the transformation is illustrated in figure 3.19

4. As a first attempt at the modelling we set up a simple random effects model for the measurement y_{ft} on fetus f at time t:

$$y_{ft} = \beta_0 + \beta_1 t + u_{0f} + e_{ft}$$
$$u_{0f} \sim \mathcal{N}(0, \tau), \quad e_{ft} \sim \mathcal{N}(0, \sigma)$$

This model can be fitted by REML, using the `lmer` function from the `lme4` pa ckage:

```
> library( lme4 )
> m0 <- lmer( sqrt(hc) ~ tga + (1|id), data=fetal )
> summary(m0)
```

You can extract the estimates and the variances from this using:

```
> fixef( m0 )
> VarCorr( m0 )
```

Note that in order to get the sds out you need (it *is* a little tricky to see where the attributes belong. . .):

```
> attr( VarCorr(m0)$id, "stddev" )
> attr( VarCorr(m0), "sc" )
```

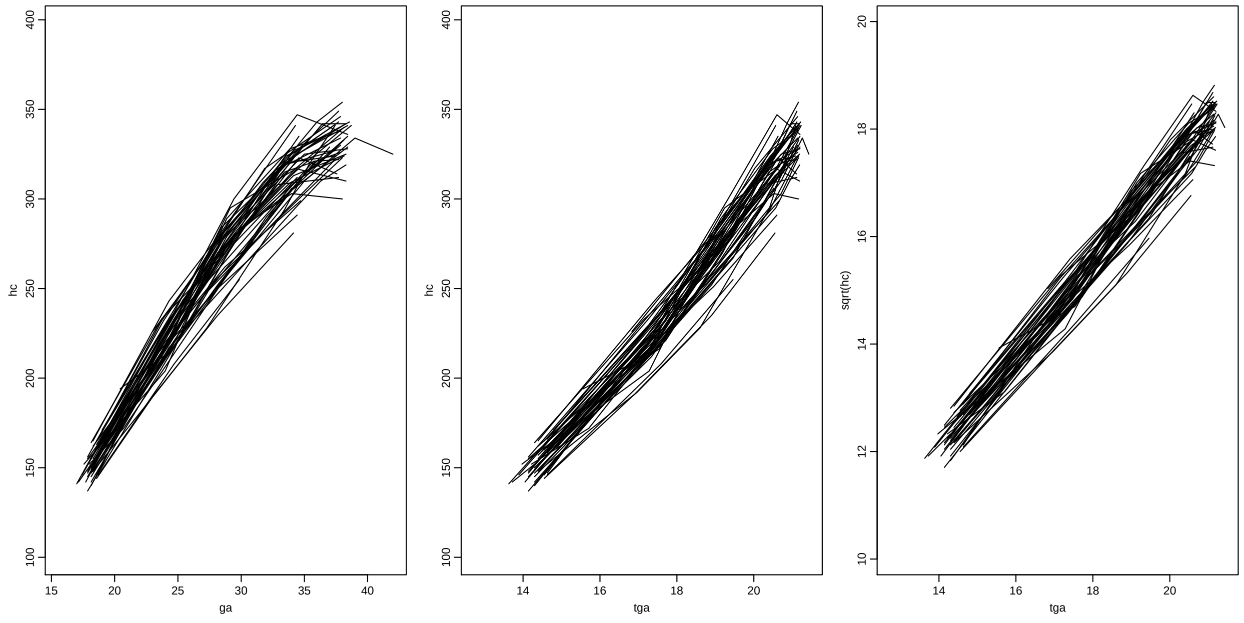


Figure 2.2: *Linearizing transformation of gestational age (quadratic transformation) and head circumference (square root).*

5. How large is the residual variation relative to the between-persons variation?

6. What is the grovt rate of fetuses' head circumference?

7. This model can be specified in JAGS as follows:

```
> cat("
+ # Fixing data to be used in model definition
+ model
+    {
+    # The model for each observational unit
+      for( j in 1:N )
+      {
+      mu[j] <- beta[1] + beta[2] * ( tga[j]-18 )  + u[id[j]]
+      hc[j] ~ dnorm( mu[j], tau.e )
+      }
+
+    # Random effects for each person
+      for( i in 1:I )
+      {
+      u[i] ~ dnorm(0,tau.u)
+      }
+
+    # Priors:
+
+    # Fixed intercept and slope
+      beta[1] ~ dnorm(0.0,1.0E-5)
+      beta[2] ~ dnorm(0.0,1.0E-5)
+
+    # Residual variance
+      tau.e <- pow(sigma.e,-2)
+    sigma.e  ~ dunif(0,100)
+
+    # Between-person variation
+      tau.u <- pow(sigma.u,-2)
+    sigma.u  ~ dunif(0,100)
+    }",
+      file="fetal0.jag" )
```

Set the model up with suitable initial values (derive them from the `lmer` output. Pay particular attention to the required data supplied to JAGS; note from the code that *two* constants are needed, both the number of units in the dataframe (`N`), but also the number of individuals `I`. The latter can be found using for example:

```
> length( unique(fetal$id) )
```

First we need the data. Note the expression `as.integer( factor(fetal$id) )`, which ensures that `id` takes on the values $1, 2, 3, \ldots$, and not just different integer values.

```
> fetal.dat <- list( id = as.integer( factor(fetal$id) ),
+                    hc = fetal$hc,
+                   tga = fetal$tga,
+                     N = nrow(fetal),
+                     I = length( unique(fetal$id) ) )
```

If you inspect the `lmer` object, you can find the estiamtes of the variance componets as follows:

```
> ( sigma.e <- attr(VarCorr(m0),"sc") )
> ( sigma.u <- attr(VarCorr(m0)$id,"stddev") )
> ( beta    <- fixef( m0 ) )
> fetal.ini <- list( list( sigma.e = sigma.e/3,
+                          sigma.u = sigma.u/3,
+                             beta = beta   /3 ),
+                    list( sigma.e = sigma.e*3,
+                          sigma.u = sigma.u*3,
+                             beta = beta   *3 ),
+                    list( sigma.e = sigma.e/3,
+                          sigma.u = sigma.u*3,
+                             beta = beta   /3 ),
+                    list( sigma.e = sigma.e*3,
+                          sigma.u = sigma.u/3,
+                             beta = beta   *3 ) )
```

Once we have set up the model-specification, the data and the starting values, we can initialize the model; that is compile the code, and use the inits and the data to run the sampler for a number of iterations

```
> library( rjags )
> system.time(
+ fetal.mod <- jags.model( file = "fetal0.jag",
+                          data = fetal.dat,
+                      n.chains = 4,
+                         inits = fetal.ini,
+                       n.adapt = 100 )
+             )
```

With the model in place we now can generate samples from the model using `coda.samples`. In this call we specify which nodes we want to sample. In this case we want to see the posterior distribution of the βs and the variance components:

```
> system.time(
+ fetal.res <- coda.samples( fetal.mod,
+                                  var = c("beta","sigma.e","sigma.u"),
+                               n.iter = 500,
+                                 thin = 20 ) )
> str( fetal.res )
> summary( fetal.res )
> dim( as.matrix(fetal.res) )
> colnames( as.matrix(fetal.res) )
```

8. Show the posterior distribution of the between-fetus and the residual standard deviations.

9. How do the estimates for random and fixed effects fit with the `lmer` estimates?

10. Now try to fit the same model with `INLA`, and inspect the object that comes out of it, and compare results with the results from `lmer` and `JAGS`:

```
> fetal <- transform(fetal, tgac=tga)
> library(INLA)
> im0 <- inla( hc ~ tga + f(id), data=fetal )
> summary( imo )
> names( im0 )
```

2.7 Linear mixed models for fetal growth

This is an extension of the fetal growth example from the previous example, based on the same dataset.

1. We are interested in describing how head circumference varies by the transformed gestational age, but also in describing how growth of the head circumference varies between fetuses. The model of choice is therefore a *linear mixed model* with a random intercept and a random slope term for the measurement y_{ft} on fetus f at time t:

$$y_{ft} = (\beta_0 + u_{0f}) + (\beta_1 + u_{1f})t + e_{ft}$$
$$(u_{0f}, u_{1f}) \sim \mathcal{N}(0, \Sigma), \quad e_{ft} \sim \mathcal{N}(0, \sigma)$$

   Now set up and estimate in this model using e.g. `lmer` form the `lme4` package.

```
> library( lme4 )
> m0 <- lmer( hc ~ tga + (tga|id), data=fetal )
> summary(m0)
```

2. Extract the variance-covariance matrix of the random effects, using `VarCorr`. What do you see? Why are they so correlated?

3. Now try to center the gestational age around, say `tga` 18, and refir the model. How is the correlation now?

4. Make a QQ-plot of the residuals from the model o check wheter they are normally distributed. Use `residuals()` to extract them form the model, and `qqplot` and `qqlines` to make a QQ-plot.

   One missing feature of the output from these models is that there is no handle on the uncertainty of the estimated variance components. This of particular interest when making predictions from the model.

2.7.1 Reporting the model

5. There are two main tings of interest to report from this model:

   (a) The estimated *mean* of head circumference as a function of gestational age, with a confidence interval; that is:

$$\hat{y}_{ft} = \beta_0 + \beta_1(t - 18)$$

   The confidence interval would be based on the variance-covariance of the βs only.

   (b) A `prediction` interval, that is an interval where you for a *given value* of gestational age would find, say, 95% of the population. The mean would of course be the same, but the interval would be based not only on the variance-covariance of the βs, but also on the estimate of σ and Σ; the variation between individual *in the current study population.*

When we report prediction intervals we are essentially making calculations `as if` the estimated variance components from the model, *sigma* and Σ were known without error and only the βs had an estimation error. In this sense we will presumably be *under*estimating the width of the prediction interval.

We can make these predictions from the output from *lmer*; the *mean* of the head circumference for a given gestational age (for which the transformed value is g_0, say is:

$$\hat{\beta}_0 + \hat{\beta}_1 g_0$$

and the variance of this is:

$$(1 g_0)\Sigma_\beta(1 g_0)'$$

where Σ_β is the estimated variance-covariance of the βs. The latter formula will even work if $(1 g_0)$ is a two-column matrix with a sequence of prediction points. It is automatically computed in the fuction `ci.lin` from the `Epi` package:

```
> library( Epi )
> tga.pt <- 14:22
> ci.lin( m0, ctr.mat=cbind(1,tga.pt) )
```

Since we are interested in predictions as a function of gestational age, define the function that transforms gestational age to the `tga`. Use this in conjunction with `ci.lin` (from the `Epi` package) to produce predicted values of head circumference as a function of gestational age.

6. However we are also interested in making a *population prediction*, that is an interval that for each value of gestational age captures the middle 95% of the fetuses' head circumference.

   To this end we must use not only the estimation variance of the βs, but also the population variance and the residual variance. So if the estimated variance of (u_0, u_1) is Σ_u, and the residual variance is σ_e^2, then the total variance for transformed gestational age g_0 is:

$$(1 g_0)\Sigma_\beta(1 g_0)' + (1 g_0)\Sigma_u(1 g_0)' + \sigma_e^2 = (1 g_0)(\Sigma_\beta + \Sigma_u)(1 g_0)' + \sigma_e^2$$

   Now extract the two matrices from the model object and use them to construct the relevant standard deviations.

   The quantities are in the `lmer` object, but a bit hidden; you can try to look at `VarCorr(m0)` and `vcoc(m0)`, and fin you that you need:

```
> Sig.u <- as.matrix( VarCorr( m0 )$id )
> Sig.b <- as.matrix( vcov( m0 ) )
> sig.e <- attr( VarCorr(m0), "sc" )
```

   Plot the predicted values of head circumference with prediciton limits.

7. The prediction limist you have just constructed, essentially assumes that the variances are known without error, so we should expect the to be a bit on the small side.

   By using MCMC for estimation we will get a posterior of the joint distribution of β, σ and Σ, meaning that we in the calculation of the prediction interval can use the posterior predictive distribution, which will include the estimation error of the variance components too.

2.7.2 Model using JAGS

8. So now set up a model in JAGS to accomplish this. You might want to use the `jags.skeletion.txt` to make sure that you get everything set up.

2.7.2.1 Model specification

Specify the model that was outlined above, using 18 as the centering point for `tga`. You can use the following as a template for the JAGS code — make sure that you understand what each component of it means.

In particular we need to specify a varince-covariance for the random effects, which is done by specifying a Wishart prior, which takes a 2×2-matrix as input, which we specify in a `data` section of the JAGS program:

```
> cat("
+ # Fixing data to be used in model definition
+ data
+    {
+    zero[1] <- 0
+    zero[2] <- 0
+    R[1,1] <- 0.1
+    R[1,2] <- 0
+    R[2,1] <- 0
+    R[2,2] <- 0.5
+    }
+ # Then define model
+ model
+    {
+    # Intercept and slope for each person, including random effects
+      for( i in 1:I )
+      {
+      u[i,1:2] ~ dmnorm(zero,Omega.u)
+      }
+
+    # Define model for each observational unit
+      for( j in 1:N )
+      {
+      mu[j] <- ( beta[1] + u[id[j],1] ) +
+               ( beta[2] + u[id[j],2] ) * ( tga[j]-18 )
+      hc[j] ~ dnorm( mu[j], tau.e )
+      }
+
+    #------------------------------------------------------------
+    # Priors:
+
+    # Fixed intercept and slope
+      beta[1] ~ dnorm(0.0,1.0E-5)
+      beta[2] ~ dnorm(0.0,1.0E-5)
+
+    # Residual variance
+      tau.e <- pow(sigma.e,-2)
+    sigma.e  ~ dunif(0,100)
+
+    # Define prior for the variance-covariance matrix of the random effects
+      Sigma.u <- inverse(Omega.u)
+      Omega.u  ~ dwish( R, 2 )
+    }",
+      file="fetal.jag" )
```

Now start the model with `jags.model` and sample the relevant quantities from subsequent iterations of the sample using `coda.samples`

Look at the joint distribution of the βs:

```
> plot( fetal.res )
```

For better control of the plotting of the posterior samples you can convert the resulting `mcmc.list` object to a data frame. You would need to doctor the names in order to be able to refer to them without too much fuss.

2.7.3 Predictive distributions

9. One of the features of JAGS is the ability to generate predictive distributions for unobserved quantities by specifying these quantities as nodes in the graphical model used by JAGS to generate the simulations.

   Compare the unconditional predictive distribution of head circumference at 38 weeks gestational age with the corresponding *conditional* distribution given the value of the head circumference at 18 weeks gestational age.

   Take a look at the five observations made on fetus `id = 5` are:

   ```
   > subset( fetal, id==5 )
   ```

   We can get the conditional distribution of head circumference at the final gestational age (38.43 weeks) given the observed measurement at gestational age of 18.43 weeks by creating a new id with identical data for the first gestational age but no observed head circumferences measurements at the final gestational age. Also

10. Finally we want to make population predictions for gestational weeks as defined in the vector `ga.pt`. This can be done in two ways, one by assuming that we look at the same fetus at all times; the other by making separate predictions for each time. Set up extra rows of the data matrix corresponding to these two scenarios, and also revise the JAGS program to catch these predictions. Moreover define nodes that wil monitor not only the *mean* of th predictive distribution, but also predicitons, including the residual error term.

    Initialze and run the model.

    Here are some hints as to how to do it:

    ```
    > x.same <- data.frame( id = max(fetal$id)+3,
    +                        hc = NA,
    +                        ga = ga.pt,
    +                       tga = tr(ga.pt) )
    > x.diff <- data.frame( id = max(fetal$id)+3+1:length(ga.pt),
    +                        hc = NA,
    +                        ga = ga.pt,
    +                       tga = tr(ga.pt) )
    ```

    In order to get the predicted values we simply monitor the relevant nodes after using JAGS on the dataset expanded with these extra records:

```
> fetal.x <- rbind( fetal, xf, x.same, x.diff )
> fetal.x[nrow(fetal)+0:10,]
> tail( fetal.x )
> nrow( fetal.x )
```

However, there is one more snag to this as we are interested in seeing *prediction* intervals, that is predictions for individual measurements, *including* the measurement errors, in the JAGS code those with precision `tau.e`. And this error term is not included in the nodes `mu`, so we must define a set of new prediction nodes, `pr`, say, to give predictions where the residual error term is included. This is done in this piece of code where we only define the `pr` nodes only for the added units where we want the predictions. In turn that requires an extra constant in data, `n`, the index of the first.

```
> cat("
+ # Fixing data to be used in model definition
+ data
+    {
+    zero[1] <- 0
+    zero[2] <- 0
+    R[1,1] <- 0.1
+    R[1,2] <- 0
+    R[2,1] <- 0
+    R[2,2] <- 0.5
+    }
+ # Then define model
+ model
+    {
+    # Intercept and slope for each person, including random effects
+      for( f in 1:F )
+      {
+      u[f,1:2] ~ dmnorm(zero,Omega.u)
+      }
+
+    # Define model for each observational unit
+      for( j in 1:N )
+      {
+      mu[j] <- ( beta[1] + u[id[j],1] ) +
+               ( beta[2] + u[id[j],2] ) * ( tga[j]-18 )
+      hc[j] ~ dnorm( mu[j], tau.e )
+      }
+      for( j in n:N )
+      {
+      pr[j] ~ dnorm( mu[j], tau.e )
+      }
+
+    #------------------------------------------------------------
+    # Priors:
+
+    # Fixed intercept and slope
+      beta[1] ~ dnorm(0.0,1.0E-5)
+      beta[2] ~ dnorm(0.0,1.0E-5)
+
+    # Residual variance
+      tau.e <- pow(sigma.e,-2)
+    sigma.e  ~ dunif(0,100)
+
+    # Define prior for the variance-covariance matrix of the random effects
+      Sigma.u <- inverse(Omega.u)
+      Omega.u  ~ dwish( R, 2 )
+    }",
+      file="fetalp.jag" )
```

Thus we see that the nodes we are interested in monitoring are (refer to the model definition) `mu[*]` with `*` from 3098 and upwards, so we modify the code and supply the relevant parameters to monitor:

```
> fetal.xdat <- list( id = as.integer( factor(fetal.x$id) ),
+                     hc = fetal.x$hc,
+                    tga = fetal.x$tga,
+                      n = nrow(fetal)+1,
+                      N = nrow(fetal.x),
+                      F = length( unique(fetal.x$id) ) )
> system.time(
+ fetal.xmod <- jags.model( file = "fetalp.jag",
+                           data = fetal.xdat,
+                       n.chains = 4,
+                          inits = fetal.ini,
+                        n.adapt = 5000 )
+            )
```

Once the code has been modified, we need to specify the nodes we shall monitor:

```
> rng <- (nrow(fetal)+1):nrow(fetal.x)
> ( mus <- paste("pr[",paste(range(rng),collapse=":"),"]",sep="") )
> system.time(
+ fetal.xres <- coda.samples( fetal.xmod,
+                             var = c("beta","sigma.e","Sigma.u",mus),
+                          n.iter = 5000,
+                            thin = 10 ) )
> fetal.qnt <- summary( fetal.xres )$quantiles
> pr.rows <- rownames(fetal.qnt)[grep( "pr", rownames(fetal.qnt) )]
> wh <- as.numeric( gsub( "\\]","", gsub("pr\\[","", pr.rows ) ) )
> cbind( fetal.x[wh,c("ga","tga")], fetal.qnt[pr.rows,c(1,3,5)] )
```

11. . . . and plot the predictions as a function of gestaional age, both from the `lmer` object and from the JAGS object. Show both the prediction including the residual error and those not, and compare them.

12. Finally, compare the conditional and marginal predictions at gestational age 38 (that is, conditional on the observed value as subject 5 has at `tga`=18). Plot the two posterior densities. Why are they so different?

2.7.3.1 Saving it all

13. For further investigation of the posteriors save the results in a file:

```
> save( fetal.res, fetal.xres, file="../data/fetal.res" )
```

2.9 Generalized linear mixed model in JAGS

1. Pelvic inflammatory disease (PID) and genital warts are conditions that occur commonly among adult women. These conditions are typically diagnosed after referral to and consultation with a sexual health physician or other specialist medical practitioner. A question of relevance to health service providers is the extent to which there is clinically relevant variation between physicians in the frequency with which PID and genital warts are diagnosed. We explore this question using data contributed by 23 sexual health physicians diagnosing patient at the Melbourne Sexual Health Centre. Data on the total number of patient consultations for each physician, and how many of these consultations resulted in the diagnosis of either genital warts or PID are contained in the text file `wartpid.csv`.

2. For each physician, calculate the proportion of patients diagnosed with genital warts and PID, and display these proportions together on the same plot (physician identifier against proportion of patients diagnosed).

3. Use `JAGS` to fit a fixed-effect logit model to the data for genital warts allowing a separate frequency of diagnosis for each physician.

4. Alter the JAGS code to allow the physician-specific parameters to be drawn from a population of normally distributed random effects. What is the posterior mean and 95% credible interval for the standard deviation of the random effects variance? What is the interpretation of this standard deviation?

5. Plot the (posterior means of the) fixed effects and random effects side-by-side on the same graph - is there substantial shrinkage of the random effects from the fixed effects towards the population mean? Does the assumption of a normal distribution for the random effects look reasonable?

6. Repeat the above question for PID...

2.10 Classical twin model in JAGS

2.10.1 Risk factors for mammographic density using twin data

Women with extensive dense breast tissue determined by mammography are known to be at higher risk of breast cancer than women of the same age with lower breast density. We will use data from a study of female monozygous (MZ) and dizygous (DZ) twin-pairs in Australia and North America to analyse the within-pair correlation of breast density, adjusted for age and weight.

The following table describes the variables in the data available as `http://bendixcarstensen.com/Bayes/Cph-2012/data/mgram.csv`:

Table 2.2: *Names of variables in the* `BUGS` *data from the mammographic density example.*

`pdens1`	Percent mammographic density twin 1
`pdens2`	Percent mammographic density twin 2
`weight1`	Weight (kg) twin 1
`weight2`	Weight (kg) twin 2
`mz`	Indicator of MZ pair (1 = MZ, 0 = DZ)
`dz`	Indicator of DZ pair (1 = DZ, 0 = MZ)
`agemgram1`	Age in years of twin 1 at mammogram
`agemgram2`	Age in years of twin 2 at mammogram
`study`	Location indicator (1 = Australia, 0 = North America)

1. Recall the basic hierarchical model for paired data described in lectures:

$$\begin{aligned} y_{i1} &= a_i + \varepsilon_{i1} \\ y_{i2} &= a_i + \varepsilon_{i2} \end{aligned}$$

where

$$\begin{aligned} \varepsilon_{ij} &\sim \mathrm{N}(0, \sigma_e^2) \quad \mathrm{cov}(\varepsilon_{i1}, \varepsilon_{i2}) = 0 \\ a_i &\sim \mathrm{N}(\mu, \sigma_a^2) \end{aligned}$$

Set up this model in JAGS, using the following code (or a variant of it) and also set up the necessary data, inits and nodes to moinitor:

```
> cat( "model
+      {
+      for (i in 1:951)
+ {
+ pdens1[i] ~ dnorm(a[i],tau.e)
+ pdens2[i] ~ dnorm(a[i],tau.e)
+      a[i] ~ dnorm(mu,tau.a)
+ }
+
+    tau.a <- pow(sigma.a,-2)
+  sigma.a  ~ dunif(0,1000)
+
+    tau.e <- pow(sigma.e,-2)
+  sigma.e  ~ dunif(0,1000)
+       mu  ~ dnorm(0,1.0E-6)
+ sigma2.a <- pow(sigma.a,2)
+ sigma2.e <- pow(sigma.e,2)
+         }",
+ file="mgram1.jag" )
```

Note that $\frac{1}{2}(\text{var}(y_{i1}) + \text{var}(y_{i2})) = \sigma_a^2 + \sigma_e^2$ and that $\frac{1}{2}(\text{var}(y_{i1} - y_{i2})) = \sigma_e^2$.

Calculate the empirical values of $\text{var}(y_{i1})$, $\text{var}(y_{i2})$ and $\text{var}(y_{i1} - y_{i2})$, and use these in a "methods of moments" calculation to produce estimates of σ_a^2 and σ_e^2 and hence generate starting values for σ_a and σ_e (since we are placing noninformative prior distributions on the standard deviation rather than the variance). You can use the sample mean of either y_{i1} or y_{i2} as the starting value for μ.

2. Compile the JAGS code and generate 1,000 iterations for summary after a burn-in of 1,000 iterations. What are the posterior means and standard deviations of μ, σ_a^2 and σ_e^2?

3. Use the posterior means of σ_a^2 and σ_e^2 to estimate the within-pair correlation of y_{i1} and y_{i2}.

4. So far we assumed a constant within-pair correlation for y_{i1} and y_{i2}, in particular that this correlation is the same for MZ and DZ pairs. If the outcome is influenced by genetic factors then this is unlikely to be a satisfactory assumption.

   Now modify the code to use an additional parameter `rho` ($\rho_{DZ:MZ}$ from lectures) to represent the ratio of $\text{cov}(y_{i1}, y_{i2})$ in DZ and MZ pairs. Assign `rho` a starting value of 0.5, and use the starting values from question 1 for the remaining parameters. You may use something like this:

```
> cat(
+ "model
+ {
+ for (i in 1:951)
+ {
+ pdens1[i] ~ dnorm(mean.pdens1[i],tau.e)
+ pdens2[i] ~ dnorm(mean.pdens2[i],tau.e)
+ mean.pdens1[i] <- b.int + sqrt(rho)*a1[i] + sqrt(1-rho)*a2[i]
+ mean.pdens2[i] <- b.int + sqrt(rho)*a1[i] + mz[i]*sqrt(1-rho)*a2[i] + dz[i]*sqrt(1-rho)*a3[i
+ a1[i] ~ dnorm(0,tau.a)
+ a2[i] ~ dnorm(0,tau.a)
+ a3[i] ~ dnorm(0,tau.a)
+ }
+
+ rho ~ dunif(0,1)
+
+ b.int ~ dnorm(0,0.0001)
+
+ tau.a <- pow(sigma.a,-2)
+ sigma.a ~ dunif(0,1000)
+
+ tau.e <- pow(sigma.e,-2)
+ sigma.e ~ dunif(0,1000)
+
+ sigma2.a <- pow(sigma.a,2)
+ sigma2.e <- pow(sigma.e,2)
+ }",
+ file="mgram2.jag" )
```

5. Generate a table of posterior summary statistics for the four parameters μ, σ_a^2, σ_e^2 and $\rho_{DZ:MZ}$.

6. How have the posterior means of σ_a^2 and σ_e^2 changed now that DZ and MZ pairs can have distinct within-pair correlations? How should this change be interpreted?

7. Does the posterior mean value for $\rho_{DZ:MZ}$ suggest that there are genetic factors determining the value of mammographic density? Is the posterior estimate of $\rho_{DZ:MZ}$ consistent with an additive genetic model?

8. Previous research has established that age-adjusted mammographic density is a risk factor for breast cancer. Include this adjustment in the model by using an extra parameter (node), `b.age`, say, in the model, and including the terms `b.age*agemgram1` and `b.age*agemgram2` in the mean model for mammographic density `pdens1` and `pdens2` in twins 1 and 2 respectively.

9. Generate a starting value for `b.age` by regressing percent mammographic density on age at mammogram in `R` using data from either twin 1 or twin 2 (or both if you're motivated to concatenate the data vectors).

10. Use the starting value in part (a) to compile and run the `JAGS` model with adjustment for age, and produce a summary table of the posterior distributions for the parameters μ, σ_a^2, σ_e^2, $\rho_{DZ:MZ}$ and $\beta_{\text{age}} =$ `b.age`. Is there evidence for a linear relationship between mammographic density and age at mammogram?

11. Has the adjustment for age changed the posterior mean of $\rho_{DZ:MZ}$? Is the current posterior mean for $\rho_{DZ:MZ}$ consistent with an additive genetic model for mammographic density?

12. The final adjustment is to further include `weight` in the model. Include this variable same way as we did in the previous question for the `agemgram` variable: Use an extra parameter (node) `b.wgt` in the model, and include the terms `b.wgt*weight1` and `b.wgt*weight2` in the mean model for mammographic density `pdens1` and `pdens2` in twins 1 and 2 respectively.

13. Generate a starting value for `b.wgt` by regressing percent mammographic density on weight and age at mammogram in `R` using data from either twin 1 or twin 2 (or both if you're motivated to concatenate the data vectors).

14. Use the starting value in part (a) to compile and run the `BUGS` model with adjustment for weight, and produce a summary table of the posterior distributions for the parameters μ, σ_a^2, σ_e^2, $\rho_{DZ:MZ}$ and $\beta_{\text{age}} =$ `b.age` and $\beta_{\text{weight}} =$ `b.wgt`. Is there evidence for a linear relationship between mammographic density and weight adjusted for age at mammogram?

15. Has the adjustment for age changed the posterior mean of $\rho_{DZ:MZ}$? Is the current posterior mean for $\rho_{DZ:MZ}$ consistent with an additive genetic model for mammographic density?

2.11 Using the DIC in model comparison

In this exercise we work through an example that demonstrates the importance of defining the *focus* (i.e. set of parameters) of a model comparison. This example is courtesy of Bob O'Hara and appears on his website
`deepthoughtsandsilliness.blogspot.com/2007/12/focus-on-dic.html`

Suppose there are $m = 10$ groups of data (indexed by $i = 1, \ldots, m$) each with $n = 50$ observations (indexed by $j = 1, \ldots, n$) that have been generated from the two-level normal-normal hierarchical model:

$$\begin{aligned} Y_{ij}|\theta_i &\sim \mathrm{N}(\theta_i, \sigma^2) \\ \theta_i|\mu_i, \tau &\sim \mathrm{N}(\mu_i, \tau^2) \end{aligned}$$

We consider two models for the group-specific mean parameter μ_i:

$$\begin{aligned} \text{Model 1: } \mu_i &= \mu + \beta(i - 5.5) \\ \text{Model 2: } \mu_i &= \mu \end{aligned}$$

The first model has a covariate (equal to the identity number of the group) but the second has none.

1. Use R to simulate data Y_{ij} according to the two models above, and plot the data in each group along with the observed group specific mean:

```
> m <- 10
> n <- 50
> N <- m*n
> tau <- 5
> sig <- 2
> i <- 1:m
> mu  <- rep(7,m)
> th  <- rnorm(m,mean=mu,sd=tau)
> Y   <- rnorm(m*n,rep(th,n),sd=sig)
> mux <- mu + 2*(i-5.5)
> thx <- rnorm(m,mean=mux,sd=tau)
> Yx  <- rnorm(m*n,rep(thx,n),sd=sig)
```

You should see from the plot that the effect of the covariate is clear, so the DIC should be able to pick it up.

2. Fit each of the models to each of the two simulated data sets, using JAGS. Extract the DIC from each model and compare them. Is the DIC lower for the model that includes the covariate when fitted to the data simulated using the group-specific covariate, compared to fitting the model without the covariate?

You should have found that in both cases the DIC is the same (for most simulations the difference is no higher than the third decimal place). But for the data simulated with a group-specific covariate (Data 1), Model 1 should be better, as suggested by the earlier plots. So what's going on? We can get a clue from plotting the posteriors of μ_i for each of the groups, from the two models.

3. Use R to plot the group-specific means for both datasets, with errors bars (i.e. $\pm$ 1 posterior standard deviation), along with the 1:1 identity line.

Obviously the models are predicting the same means for the groups, and hence we will get the same deviance (recall that we are talking about the plug-in deviance here which depends only on the posterior means of the parameters on which we are focussing). We can see why this is happening from the between-group or group-level standard deviations.

4. Use the output JAGS run to calculate the posterior mean and standard deviation of the between-group or group-level standard deviation parameter τ for both Model 1 and Model 2 applied to Data 1 and Data 2.

You should have found that for the data where there is a trend (Data 1), but none is fitted, the posterior mean of τ is much larger. The lack of the linear trend is compensated by the increase in variance. The difference is not in the model for θ at all, but occurs higher in the hierarchy at the level of the hyperparameter μ where the effect of the group-specific covariate is incorporated into the model.

This is obvious from looking at the models. In order for it to be reflected in a comparison of the DIC between models, we need to change the focus, from θ to μ and β. This then means calculating the *marginal* deviance, marginalising over θ, that is, looking at $p(\mathbf{Y}|\mu,\tau)$ after integrating $p(\mathbf{Y}|\theta)$ over $p(\theta|\mu,\tau)$. This can be done analytically, after which we find that the deviance can be calculated because we know the distribution of the group-specific sample mean $\overline{Y}_{i.} = \sum_{j=1}^{n} Y_{ij}/n$, which is

$$\overline{Y}_{i.} \sim \mathrm{N}(\mu_i, \sigma^2/n + \tau^2). \tag{2.6}$$

5. Recalculate the DIC for each dataset using R.

The results should now make more sense. For the data with a covariate effect for the mean model, the DIC massively favours the correct model. Without the effect in the data, the DIC is pretty similar for the two models. In both cases, also note that p_D is larger by 1 for the model with 1 extra parameter, as expected.

What lessons can we draw from this? Firstly, that DIC is not an automatic panacea - it must be focussed on the right part of the model. If the focus is not at the level immediately above the data (i.e. θ here), then you can't use the DIC given by BUGS. In this example it is more difficult to get at the correctly focussed DIC (in fact you have to calculate it manually yourself, or at least use Bob O'Hara's R function to do so). For more complex models this might be awkward, since if there are no analytical results, then the parameters to be integrated out have to be simulated, for example by Markov chain Monte Carlo.

Some comments from Martyn Plummer:

This example encourages you to think about what DIC is trying to do. It's not about finding the "true" model - both models are true in fact - it's about accurately predicting dropped observations.

In the simulated data, there are 50 observations in each group. If you drop one observation and then tried to predict it, you already have plenty information from the other 49 observations in the same group that share the same mean, and you have 489 degrees of freedom to estimate the variance. The group-level covariate really doesn't add much to your ability to make that prediction.

Changing the focus to the group level, you are dropping a whole group and then trying to predict the 50 observations in it. In this case, the group-level covariate is very useful. Here DIC parts company with the penalized plug-in likelihood since we have around 3 effective parameters and only 10 independent observations! You'd most likely be better off using the "corrected" DIC proposed in the Discussion of Plummer (2008). Although the calculations haven't been done explicitly, the substantive conclusions must surely be the same.

2.12 Measurement comparison in oximetry.

A common problem in medical statistics is assessing the extent to which a new technique for measuring a biological quantity gives results that agree with a more established method of measurement. An important example arises in *oximetry* which is the measurement of the saturation or concentration of oxygen in the blood. Patients who are critically ill are unable to send enough oxygen into the bloodstream and the level of oxygen saturation is monitored as an indicator of the severity of the patient's condition. The traditional method of measurement uses a sample of blood on which a chemical analysis is performed to determine the level of various gases in the blood ("co-oximetry"). A much more convenient, newer, method uses a device called a pulse oximetry, which relies on a small sensor placed on a finger or toe to measure oxygen saturation by measuring the reflectance of light through the blood vessels.

A study was done at the Royal Children's Hospital in Melbourne to examine the agreement between pulse oximetry and co-oximetry in small babies, many of whom were especially sick and therefore had oxygen saturation levels lower than those usually available to test the accuracy of pulse oximetry. The data file contains 5 variables on a total of 61 babies, and is available as the dataset `ox` in the MethComp package.

Each baby contributed 3 samples to the study (so that there are $61 \times 3 = 183$ observations in total from 61 individuals.

The aim of the analyses here will be to use Bayesian methods to draw inferences about the mean and variance of the *difference* between measurements of oxygen saturation made using the pulse oximetry and co-oximetry techniques, as well as producing prediction from measurements by one method to measurements by another method.

1. Load the `MethComp` package, and load the dataset `ox`, and look at the help page for that

```
> library(MethComp)
> data(ox)
> ?ox
```

2. Plot the two types of measurement against each otter, by first making the data set into a `Meth` object:

```
> ox <- Meth(ox)
> BA.plot( ox )
> BA.plot( ox, repl.conn=TRUE )
> BA.plot( ox, repl.conn=TRUE, pl.type="conv" )
```

3. Now try to fit a variance components model that assumes constant difference between methods:

```
> BA.est( ox )
```

4. Fit a proper regression model with all variance components. You should probably use a bit more than 200 iterations:

```
> system.time(
+ Jox <- MCmcmc( ox, program="JAGS", n.iter=200 ) )
> Jox
> MethComp( Jox )
```

5. Fit a model to the transformed data, after looking at them:

```
> BA.plot( ox, Transform="pctlogit", repl.conn=TRUE, axlim=c(0,100), xaxs="i" )
> tJox <- MCmcmc( ox, program="JAGS", n.iter=200, Transform="pctlogit" )
> tJox
> MethComp( tJox )
```

Chapter 3

Solutions

3.1 Bayesian inference in the binomial distribution

1. In the discrete case we just set up a vector if the same length as the prior — we know that the likelihood and posterior only are defined in the points where the prior is positive.

   (a) In Rwe just do the computations according to the rules, and the print the vector side by side corresponding to the table in the exercise:

   ```
   > theta <- c(2,4,6,8)/10
   > prior <- c(1,1,1,1)/4
   > x <- 1
   > n <- 1
   > like   <- dbinom( x, n, theta )
   > like.pr <- prior * like
   > post   <- like.pr / sum( like.pr )
   > round( cbind( theta, prior, like, like.pr, post ), 3 )

        theta prior like like.pr post
   [1,]   0.2  0.25  0.2    0.05  0.1
   [2,]   0.4  0.25  0.4    0.10  0.2
   [3,]   0.6  0.25  0.6    0.15  0.3
   [4,]   0.8  0.25  0.8    0.20  0.4
   ```

   Not surprising, the posterior is proportional to the likelihood when we use a uniform prior as in this case. And since the likelihood is maximal for $theta = 1$, we get the maximal posterior probability for $\theta = 0.8$, the largest possible value.

   (b) If we had 20 trials and 15 successes we just change the value of `x` and `n` in the code:

   ```
   > theta <- c(2,4,6,8)/10
   > prior <- c(1,1,1,1)/4
   > x <- 15
   > n <- 20
   > like   <- dbinom( x, n, theta )
   > like.pr <- prior * like
   > post   <- like.pr / sum( like.pr )
   > round( cbind( theta, prior, like, like.pr, post ), 3 )

        theta prior  like like.pr  post
   [1,]   0.2  0.25 0.000   0.000 0.000
   [2,]   0.4  0.25 0.001   0.000 0.005
   [3,]   0.6  0.25 0.075   0.019 0.298
   [4,]   0.8  0.25 0.175   0.044 0.697
   ```

We see the same patterns as before. The 0 posterior for $\theta = 0.2$ is not an exact 0; it is just a consequence of rounding:

```
> round( cbind( theta, prior, like, like.pr, post ), 17 )

     theta prior         like      like.pr         post
[1,]   0.2  0.25 1.664729e-07 4.161823e-08 6.645594e-07
[2,]   0.4  0.25 1.294494e-03 3.236234e-04 5.167614e-03
[3,]   0.6  0.25 7.464702e-02 1.866175e-02 2.979907e-01
[4,]   0.8  0.25 1.745595e-01 4.363988e-02 6.968411e-01
```

(c) If we expand the set of support points for the prior (and hence also for the posterior, should get an expansion of the support for the posterior too. But if $x \neq 0$, then the likelihood at $\theta = 0$ is 0, since this value of θ corresponds to a situation where an event never occurs. Likewise if $x \neq n$ the likelihood at $\theta = 1$ is 0, since this corresponds to a situation where an event always occurs.

If we have $x = 15$ and $n = 20$, the the likelihood at the two outer points will be the same and the posterior will also be the same (because the prior at the "remaining points" is the same as before, bar a constant:

```
> theta <- c(0,2,4,6,8,10)/10
> prior <- c(1,1,1,1,1,1)/6
> x <- 15
> n <- 20
> like  <- dbinom( x, n, theta )
> like.pr <- prior * like
> post  <- like.pr / sum( like.pr )
> round( cbind( theta, prior, like, like.pr, post ), 3 )

     theta prior  like like.pr  post
[1,]   0.0 0.167 0.000   0.000 0.000
[2,]   0.2 0.167 0.000   0.000 0.000
[3,]   0.4 0.167 0.001   0.000 0.005
[4,]   0.6 0.167 0.075   0.012 0.298
[5,]   0.8 0.167 0.175   0.029 0.697
[6,]   1.0 0.167 0.000   0.000 0.000
```

(d) If we only have a singe positive trial, we will however have a positive likelihood at $\theta = 1$:

```
> theta <- c(0,2,4,6,8,10)/10
> prior <- c(1,1,1,1,1,1)/6
> x <- 1
> n <- 1
> like  <- dbinom( x, n, theta )
> like.pr <- prior * like
> post  <- like.pr / sum( like.pr )
> round( cbind( theta, prior, like, like.pr, post ), 3 )

     theta prior like like.pr  post
[1,]   0.0 0.167  0.0   0.000 0.000
[2,]   0.2 0.167  0.2   0.033 0.067
[3,]   0.4 0.167  0.4   0.067 0.133
[4,]   0.6 0.167  0.6   0.100 0.200
[5,]   0.8 0.167  0.8   0.133 0.267
[6,]   1.0 0.167  1.0   0.167 0.333
```

2. In the continuous case we use the Beta-distribution, which is also available in R, so it is straightforward to do the same calculations as above. However we cannot just print the values of the prior, the likelihood and the posterior at the supported values, because the support is now the entire interval $[0, 1]$. Hence we compare by making graphs with an x-axis form 0 to 1.

(a) The formulae given in the exercise immediately lend themselves to implementation in R:

$$\begin{aligned} m &= \frac{a}{a+b} & \Leftrightarrow \qquad a &= m(a+b) \\ s &= \sqrt{\frac{m(1-m)}{a+b+1}} & \Leftrightarrow \quad a+b &= \big(m(1-m)/s^2\big)-1 \end{aligned}$$

The only thing we need to supply are the desired values of m and s:

```
> m <- 0.4
> s <- 0.1
> a.plus.b <- m*(1-m)/s^2 - 1
> a <- m * a.plus.b
> b <- a.plus.b - a
> c(m,s,a,b)

[1]  0.4  0.1  9.2 13.8
```

(b) For these values of a and b we can just use the Beta-density implemented in the `dbeta` function in R to plot the desired prior distribution function:

```
> # Points where we plot:
> p <- seq(from=0,to=1,length=100)
> # Graph of the prior
> plot( p, dbeta( p, a, b ), lwd=4, bty="n", type="l" )
```

(c) For an observation of $x = 15$ out of $n = 20$ we use the `dbinom` function with the probability `p` as the argument to plot the likelihood:

```
> x <- 15
> n <- 20
> plot( p, dbinom( x, n, p ), lwd=4, bty="n", type="l" )
```

(d) We know that the posterior is a Beta-distribution with parameters $a + x$ and $b + n - x$, so this is just as easily implemented in R:

```
> plot( p, dbeta( p, a+x, b+n-x ), lwd=4, bty="n", type="l" )
```

(e) In order to see how the three relate we collect the three plots in one frame:

```
> par( mfcol=c(3,1) )
> plot( p, dbeta( p, a, b ), lwd=4, bty="n", type="l" )
> plot( p, dbinom( x, n, p ), lwd=4, bty="n", type="l" )
> plot( p, dbeta( p, a+x, b+n-x ), lwd=4, bty="n", type="l" )
```

which is slightly primitive; a more beefed-up version would be:

```
> par( mfcol=c(3,1), mar=c(3,3,0,0) )
> plot( p, dbeta( p, a, b ), lwd=4, bty="n", type="l" )
> text( par("usr")[1], par("usr")[4], "\n  Prior", adj=c(0,1) )
> plot( p, dbinom( x, n, p ), lwd=4, bty="n", type="l" )
> text( par("usr")[1], par("usr")[4], "\n  Likelihood", adj=c(0,1) )
> plot( p, dbeta( p, a+x, b+n-x ), lwd=4, bty="n", type="l" )
> text( par("usr")[1], par("usr")[4], "\n  Posterior", adj=c(0,1) )
```

The results of these two approaches are shown side-by-side in figure ??.

(f) In order to illustrate the effect of variations in the prior and the data we wrap the calculations, and the graphing of the three functions in an R-function. The `text`-function draws text on the plot so it is possible to trace the parameters in the various plots.

```
> Bayes.ill <-
+ function( m, s, x, n, ... )
+ {
+ p <- seq(0,1,,1000)
+ a.plus.b <- m*(1-m)/s^2 - 1
+ a <- m * a.plus.b
+ b <- a.plus.b - a
+ plot( p, dbeta( p, a, b ), lwd=4, bty="n", type="l", ... )
+ text( par("usr")[1], par("usr")[4],
+       paste("\n  Prior\n  m=", m, ",s=", s,
+                        "\n  a=", a,", b=", b), adj=c(0,1) )
+ plot( p, dbinom( x, n, p ), lwd=4, bty="n", type="l", ... )
+ text( par("usr")[1], par("usr")[4],
+       paste("\n  Likelihood\n  n=", n,", x=",x), adj=c(0,1) )
+ plot( p, dbeta( p, a+x, b+n-x ), lwd=4, bty="n", type="l", ... )
+ text( par("usr")[1], par("usr")[4],
+       paste("\n  Posterior\n  Beta(", a+x, ",", b+n-x, ")"), adj=c(0,1) )
+ }
```

Note the argument "`...`" which allows us to pass extra parameters on the the plot statements. This function produces three plots, so when using it it will be convenient to set up a layout of plots using for example `par(mfcol=c(3,2)`, which gives a 3 by 2 matrix of graphs, filled column-wise. The `mar=` argument governs the whitespace around the single plot frames, and we use `col=gray(0.5)` to plot the curves in gray so that any text on top of them will be visible:

```
> par( mfcol=c(3,2), mar=c(2,4,0,0) )
> Bayes.ill( 0.4, 0.2, 15, 20, col=gray(0.5) )
> Bayes.ill( 0.4, 0.1, 15, 20, col=gray(0.5) )
```

```
> par( mfcol=c(3,2), mar=c(2,4,0,0) )
> Bayes.ill( 0.4, 0.2, 55, 100, col=gray(0.5) )
> Bayes.ill( 0.4, 0.1, 75, 100, col=gray(0.5) )
```

The results of these statements are shown in figure ??.

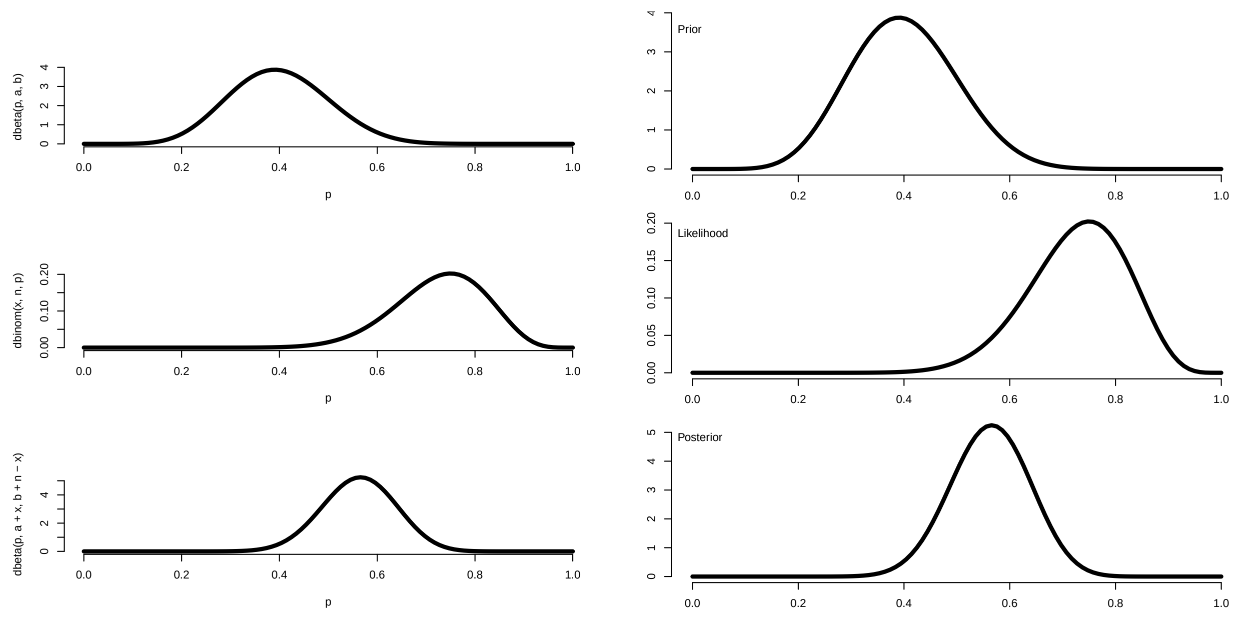


Figure 3.1: *Prior, likelihood and posterior for the binomial model. The right hand side is just the beefed-up version of the plot.*

3. The fraction of female births in most societies is around 48.7%. A reasonable prior would be one that is centered around 50% with a spead that is effectively so large that is will encompass even extreme deviations form the expected mean.

   (a) If we use a Beta(100,100) We can either make a numeric calculation for the probability that a Beta(100,100) variate is between 0.4 and 0.6:

   ```
   > pbeta( 0.6, 100, 100 ) - pbeta( 0.4, 100, 100 )
   ```

   ```
   [1] 0.9956798
   ```

   or do a more brutal computation using a random sample:

   ```
   > zz <- rbeta( 10000, 100, 100 )
   > mean( zz<0.6 & zz>0.4 )
   ```

   ```
   [1] 0.9958
   ```

   So we are indeed more than 95% certain that the true fraction of girls is between 40 and 60%!

   (b) If we see 511 boys out of 1000 births, we can use the previous function to illustrate how the the prior, likelihood and posterior look in this problem. Note that we use the "..." argument to pass on a limitation of the x-axis:

   ```
   > a <- b <- 100
   > m <- a/(a+b)
   > s <- sqrt(m*(1-m)/(a+b+1))
   > par( mfcol=c(3,1), mar=c(4,2,0,0) )
   > Bayes.ill( m, s, 511, 1000, xlim=c(0.4,0.6), xlab="% male births" )
   > abline(v=0.5)
   ```

   (c) The posterior probability that the fraction of female births i larger than 0.5 is the same the probability that the fraction of male births is < 0.5, is just a cumulative probability in the posterior distribution which is Beta(611,589):

   ```
   > pbeta(0.5,611,589)
   ```

   ```
   [1] 0.2626087
   ```

   i.e. the prior and the data translates into a posterior probability of 26%. We see that the prior has a limited influence; a flat prior (Beta(1,1)) would have resulted in a posterior with parameters (511,489), and a smaller posterior probability:

   ```
   > pbeta(0.5,512,490)
   ```

   ```
   [1] 0.2434263
   ```

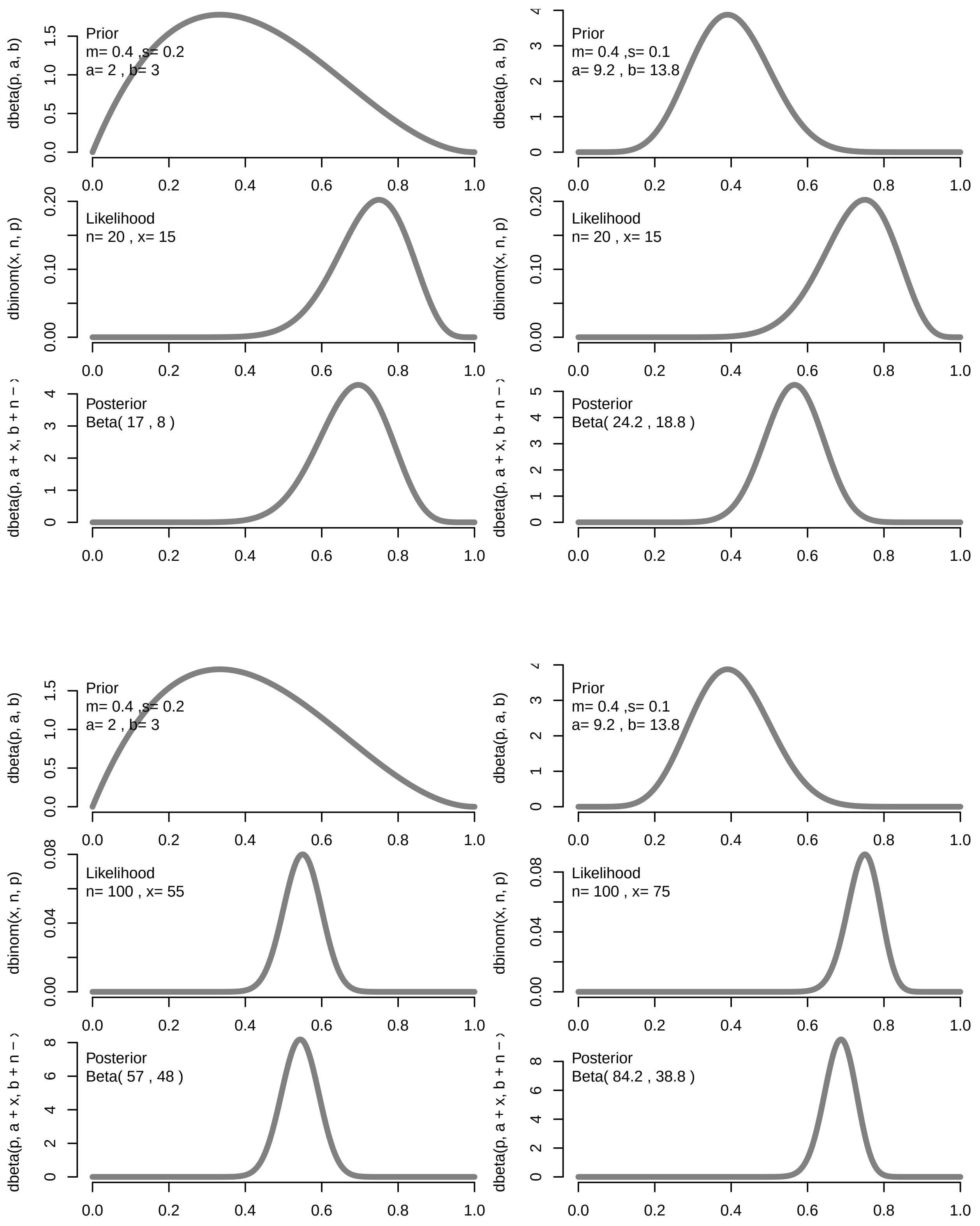


Figure 3.2: *Prior, likelihood and posterior for the binomial model for different combinations of prior information and data. Large amounts of data makes the likelihood the dominant factor; and a narrow prior (strong beliefs!) makes the prior the dominant factor.*

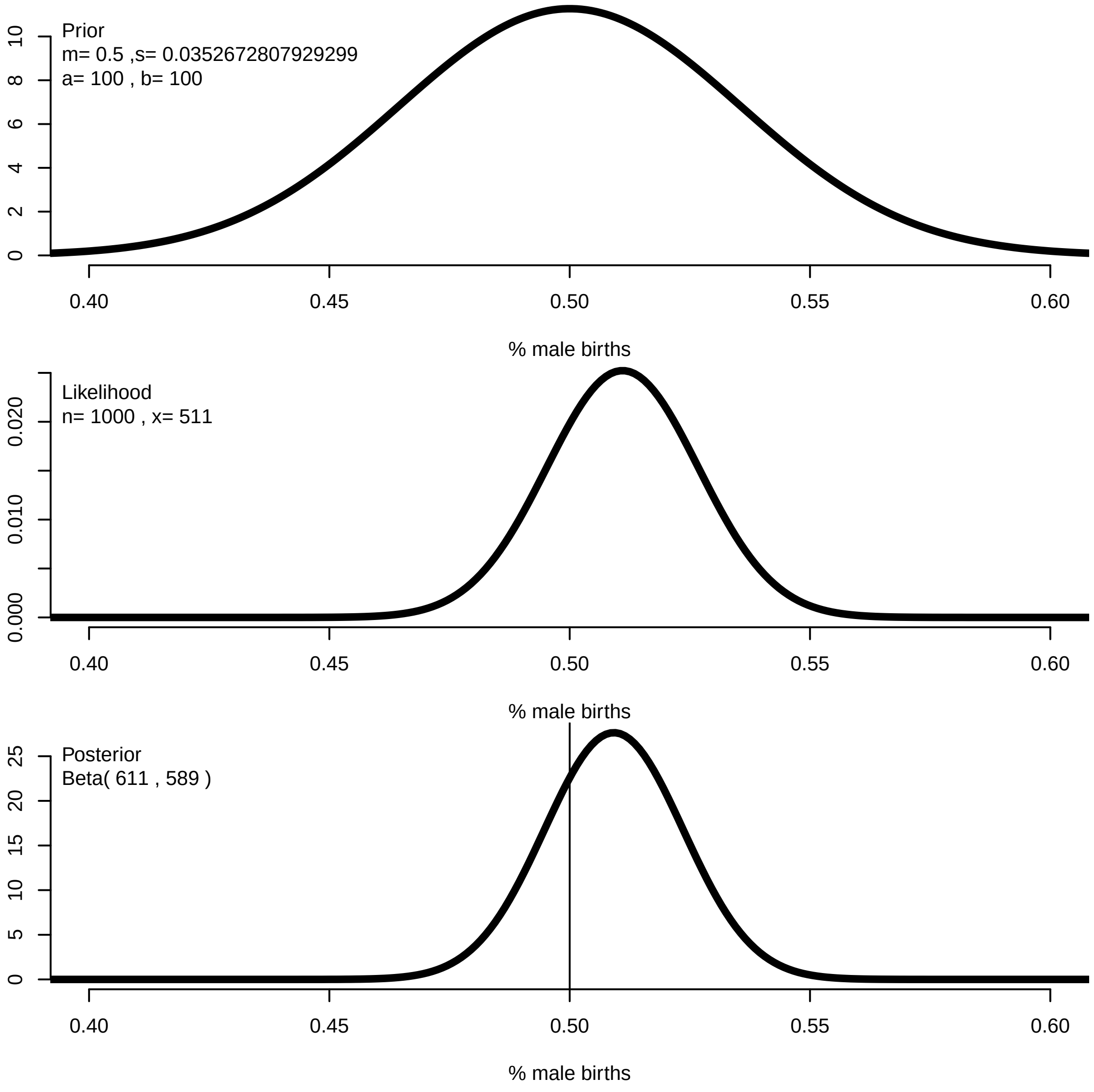


Figure 3.3: *Prior, likelihood and posterior for the binomial model for 511 births out of 100, using a Beta(100,100) prior. It is immediately apparent that the prior has very little influence on the posterior — all the information is in the likelihood, i.e. the data.*

3.2 Simple linear regression with BUGS

First we load all the required packages for this practical:

```
> library( rjags )
> library( Epi )
```

1. Define and plot the bogus data and inspect the output from the linear regression analysis:

   ```
   Call:
   lm(formula = y ~ x)

   Residuals:
          1        2        3        4        5        6 
   -0.09524  0.87619 -0.15238 -1.18095 -0.20952  0.76190 

   Coefficients:
               Estimate Std. Error t value Pr(>|t|)
   (Intercept)  0.06667    0.78153   0.085  0.93612
   x            1.02857    0.20068   5.125  0.00686

   Residual standard error: 0.8395 on 4 degrees of freedom
   Multiple R-squared: 0.8679,        Adjusted R-squared: 0.8348
   F-statistic: 26.27 on 1 and 4 DF,  p-value: 0.00686
   ```

   The estimates of α and β are 0.067 and 1.029, and the estimate of σ is 0.840.

2. In order to use JAGS we set up the data, initial values (for three chains) and the list of parameters to monitor:

   ```
   > reg.dat <- list( x=x, y=y, I=6 )
   > reg.ini <- list( list( alpha=0.05, beta=1.0, sigma=0.9 ),
   +                  list( alpha=0.04, beta=1.1, sigma=1.0 ),
   +                  list( alpha=0.06, beta=0.9, sigma=1.1 ) )
   > reg.par <- c("alpha","beta","sigma" )
   ```

   Finally we need to specify the model in BUGS code, using the names we specified for the data in reg.dat.

   ```
   > cat( "model
   +       {
   +       for( i in 1:I )
   +          {
   +          y[i] ~ dnorm(mu[i],tau)
   +          mu[i] <- alpha + beta*x[i]
   +          }
   +       alpha ~ dnorm(0, 1.0E-6)
   +       beta  ~ dnorm(0, 1.0E-6)
   +       sigma ~ dunif(0,100)
   +       tau <- 1/pow(sigma,2)
   +       }",
   +       file="reg.jag" )
   ```

   With these specifications we can now use JAGS to first compile and initialize the model and then run the model for some 10000 iterations (and hopefully get to a steady state of the chain):

   ```
   > reg.mod <- jags.model( file = "reg.jag",
   +                        data = reg.dat,
   +                    n.chains = 3,
   +                       inits = reg.ini,
   +                     n.adapt = 10000 )
   ```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 35

Initializing model
```

SAfter that we run the chain further while monotoring the parameters of interest:

```
> reg.res <- coda.samples( reg.mod,
+                               var = reg.par,
+                            n.iter = 10000,
+                              thin = 10 )
```

3. The summary of the posterior distributions of the parameters can now be obtained by the `summary` function and compared to the parameter estimates from the standard regression model:

```
> summary( reg.res )


Iterations = 10010:20000
Thinning interval = 10
Number of chains = 3
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
alpha 0.09658 1.6408 0.029957       0.038482
beta  1.02186 0.4126 0.007533       0.008915
sigma 1.36959 1.1588 0.021156       0.055039

2. Quantiles for each variable:

         2.5%     25%     50%    75% 97.5%
alpha -2.7649 -0.6261 0.08741 0.7623 3.100
beta   0.2471  0.8479 1.02317 1.2081 1.761
sigma  0.5481  0.8183 1.07674 1.5263 3.706


> ci.lin( m0 )


              Estimate    StdErr          z            P       2.5%    97.5%
(Intercept) 0.06666667 0.7815329 0.08530245 9.320209e-01 -1.4651096 1.598443
x           1.02857143 0.2006791 5.12545318 2.968229e-07  0.6352476 1.421895


> summary( m0 )$sigma


[1] 0.839501
```

It is seen that the ML estimates and the posterior means / medians are in fairly good agreement whereas the estimate of σ is pretty far away from the posterior mean / median. This is partly due to the fact that the dataset have 6 observations and hence virtually no information about the residual standard deviation.

4. If we try to do the parallel analysis of a real dataset with some 500 obeservations we must make sure that there are no missing values in the x-variable.

   From the `births` dataset we will use $y =$`bweight` and $x =$ `gestwks` $- 35$. We can use almost the same code as for the small bogus dataset:

```
> data( births )
> births <- subset( births, !is.na(gestwks) )
> dim( births )

[1] 490   8


> mb <- lm( bweight ~ I(gestwks-35), data=births )
> summary( mb )


Call:
lm(formula = bweight ~ I(gestwks - 35), data = births)

Residuals:
     Min       1Q   Median       3Q      Max 
-1698.40  -280.14    -3.64   287.61  1382.24 

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     2404.902     38.504   62.46   <2e-16
I(gestwks - 35)  196.973      8.788   22.41   <2e-16

Residual standard error: 449.7 on 488 degrees of freedom
Multiple R-squared: 0.5073,	Adjusted R-squared: 0.5062 
F-statistic: 502.4 on 1 and 488 DF,  p-value: < 2.2e-16


> bth.dat <- list( x=births$gestwks-35,
+                  y=births$bweight,
+                  I=nrow(births) )
> bth.ini <- list( list( alpha=2400, beta=200, sigma=400 ),
+                  list( alpha=2300, beta=150, sigma=450 ),
+                  list( alpha=2500, beta=250, sigma=500 ) )
> bth.par <- c("alpha","beta","sigma" )
> cat( "model
+      {
+      for( i in 1:I )
+         {
+         y[i] ~ dnorm(mu[i],tau)
+         mu[i] <- alpha + beta*x[i]
+         }
+      alpha ~ dnorm(0, 1.0E-6)
+      beta  ~ dnorm(0, 1.0E-6)
+      sigma ~ dunif(0,10000)
+      tau <- 1/pow(sigma,2)
+      }",
+      file="bth.jag" )
> bth.mod <- jags.model( file = "bth.jag",
+                        data = bth.dat,
+                    n.chains = 3,
+                       inits = bth.ini,
+                     n.adapt = 2000 )


Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 1661

Initializing model
```

```
> bth.res <- coda.samples( bth.mod,
+                               var = bth.par,
+                            n.iter = 10000,
+                              thin = 10 )
> summary( mb )$sigma

[1] 449.7237
```

We now get a much better accordance between the regression estimates and the posterior means / medians and also for the confidence intervals. The latter is of course because the residual standard deviation is now much more precisely determined. The moral is of course that with more data you get more precision.

5. The classically derived confidence intervals are now much better in agreement with the posterior central intervals:

```
> summary( bth.res )$quan[,c(3,1,5)]

            50%      2.5%     97.5%
alpha 2401.4021 2323.0067 2477.9097
beta   197.7579  180.2511  214.9954
sigma  450.4190  423.9036  480.4147

> ci.lin( mb )[,c(1,5,6)]

                 Estimate      2.5%     97.5%
(Intercept)     2404.9021 2329.4351 2480.3692
I(gestwks - 35)  196.9726  179.7482  214.1971
```

For comparison of the posterior for the standard deviation, we can use the χ^2/f approximation to the estimate of the residual variance, which yields confidence limits for the standard deviation as the estimate muliplied by $\sqrt{f/\chi^2_{0.975}(f)}$ and $\sqrt{f/\chi^2_{0.025}(f)}$ (derive that!):

```
> cim <- sqrt( c( 1,
+                 mb$df/qchisq(0.975,mb$df),
+                 mb$df/qchisq(0.025,mb$df) ) )
> summary(mb)$sigma * cim

[1] 449.7237 423.1938 479.8294

> summary( bth.res )$quan["sigma",c(3,1,5)]

     50%      2.5%     97.5%
450.4190 423.9036 480.4147

> summary(mb)$sigma * cim /
+ summary( bth.res )$quan["sigma",c(3,1,5)]

      50%      2.5%     97.5%
0.9984564 0.9983257 0.9987816
```

The agreement with the posterior is impressive…

3.3 Examples of the Gibbs sampler and Metropolis Hastings algorithm

1. (a) Let $\theta = (\theta_1, \theta_2)$ be the mean vector, which we know has a multivariate normal posterior distribution with mean $\mathbf{y} = (y_1, y_2)$ and covariance matrix $\begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$. If we let $U = \theta_1$ and $V = \theta_2$ then we can use result (A.1) on page 579 of **BDA**, which states that $p(U|V)$ is univariate normal with

$$\begin{aligned} \mathrm{E}(U|V) &= \mathrm{E}(U) + \mathrm{cov}(V,U)\mathrm{var}(V)^{-1}(V - \mathrm{E}(V)) \\ \mathrm{var}(U|V) &= \mathrm{var}(U) - \mathrm{cov}(V,U)\mathrm{var}(V)^{-1}\mathrm{cov}(U,V)) \end{aligned}$$

Substituting in the expectations, variances and covariances *conditional on* $\mathbf{y}$ into the right hand sides of these expressions gives the following results:

$$\begin{aligned} \mathrm{E}(\theta_1|\theta_2, y) &= \mathrm{E}(\theta_1|y) + \mathrm{cov}(\theta_2, \theta_1|y)\mathrm{var}(\theta_2|y)^{-1}(\theta_2 - \mathrm{E}(\theta_2|y)) \\ &= y_1 + \rho \times 1 \times (\theta_2 - y_2) \\ &= y_1 + \rho(\theta_2 - y_2) \\ \mathrm{var}(\theta_1|\theta_2, y) &= \mathrm{var}(\theta_1|y) - \rho \times \mathrm{var}(\theta_2|y)^{-1} \times \rho \\ &= 1 - \rho \times 1 \times \rho \\ &= 1 - \rho^2. \end{aligned}$$

The result for θ_2 follows by symmetry.

(b) Gibbs Sampler.

2. For the Metropolis-Hastings bivariate proposal distribution example, here's some summary plots of the sample paths.

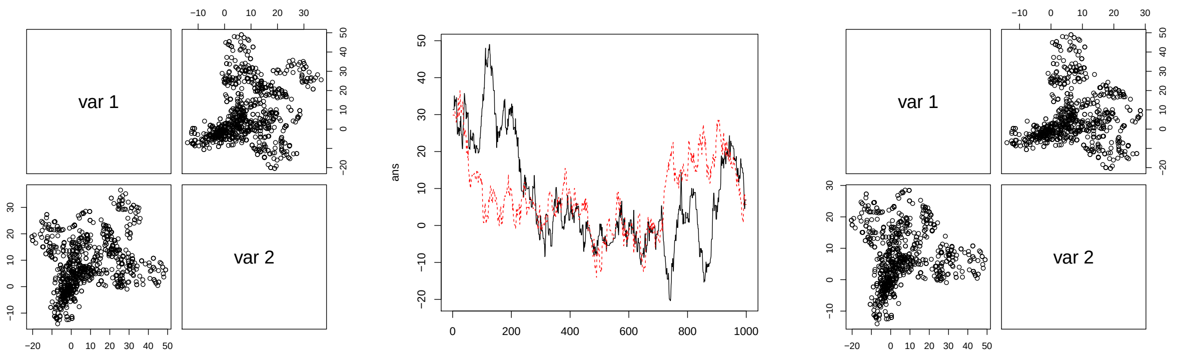


Figure 3.4: *Metropolis-Hastings sample paths*

A plot of the dependencies using the `pacf` and `acf` functions:

The acceptance probability increases slightly as the correlation parameter decreases since the proposal distribution is getting closer to the target distribution.

3. For the single component Metropolis–Hastings sampler, here's some summary plots of the sample paths.

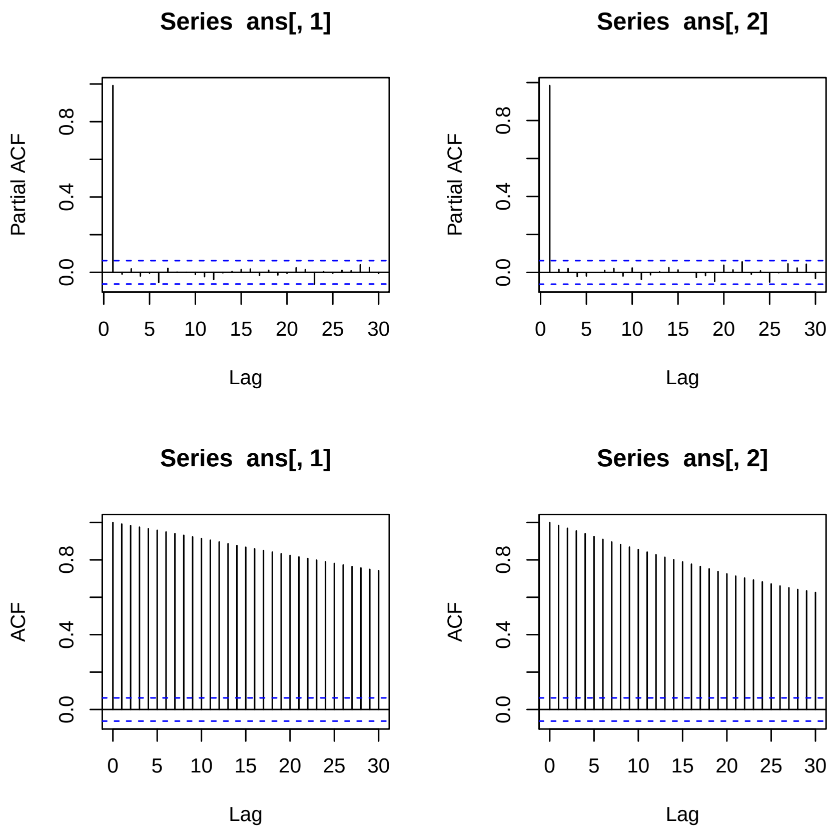


Figure 3.5: *Metropolis-Hastings — autocorrelations*

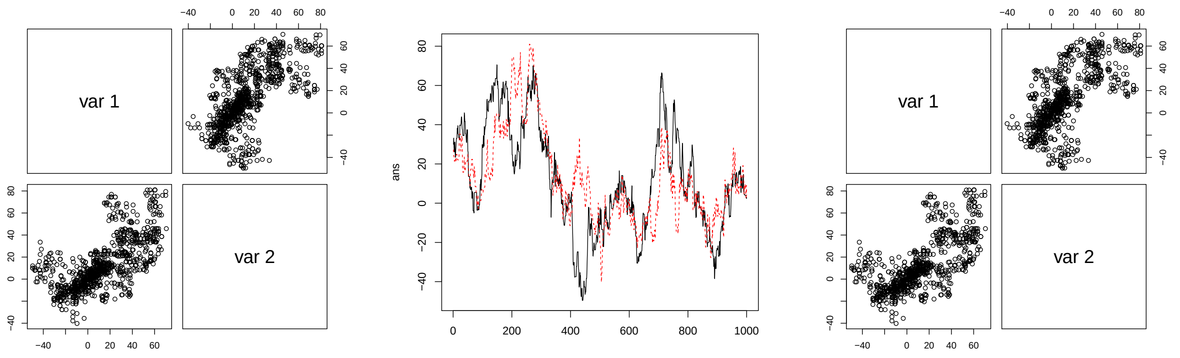


Figure 3.6: *Single component Metropolis-Hastings — sample paths*

And a plot of the acceptance probabilities:

Plotting the two series `x1` and `x2` against each other in a scatter plot is a good way to see how the length of the jumps depends on the standard deviation of the proposal distribution. The jumps get longer when the standard deviation of the proposal distribution increases.

Finally we check the dependencies within each of the `x1` and `x2` series by using the `pacf` and `acf` functions.

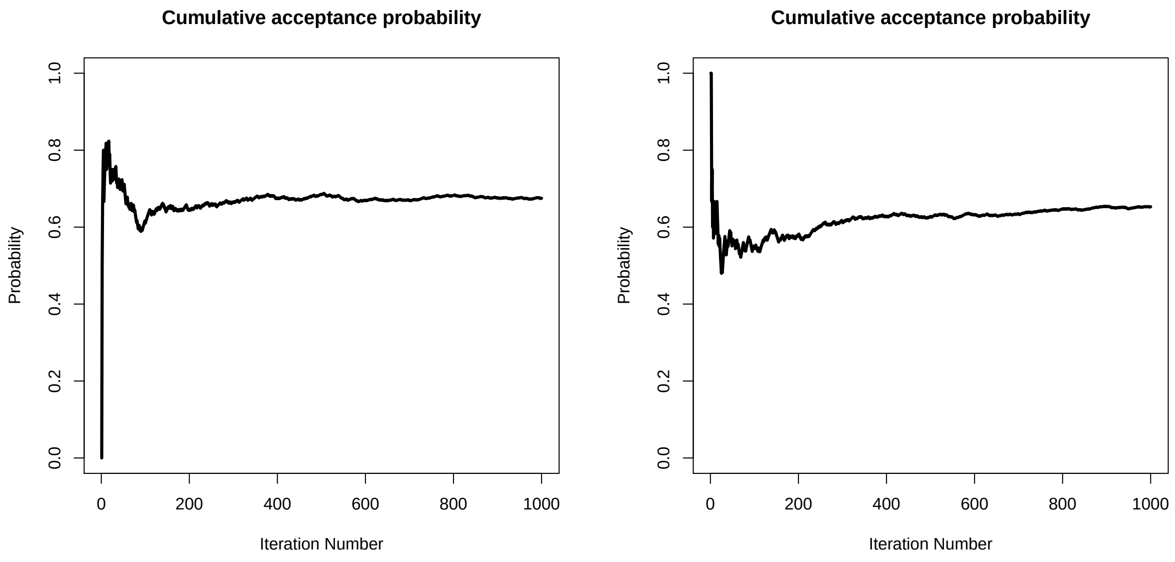


Figure 3.7: *Metropolis-Hastings acceptance probabilities*

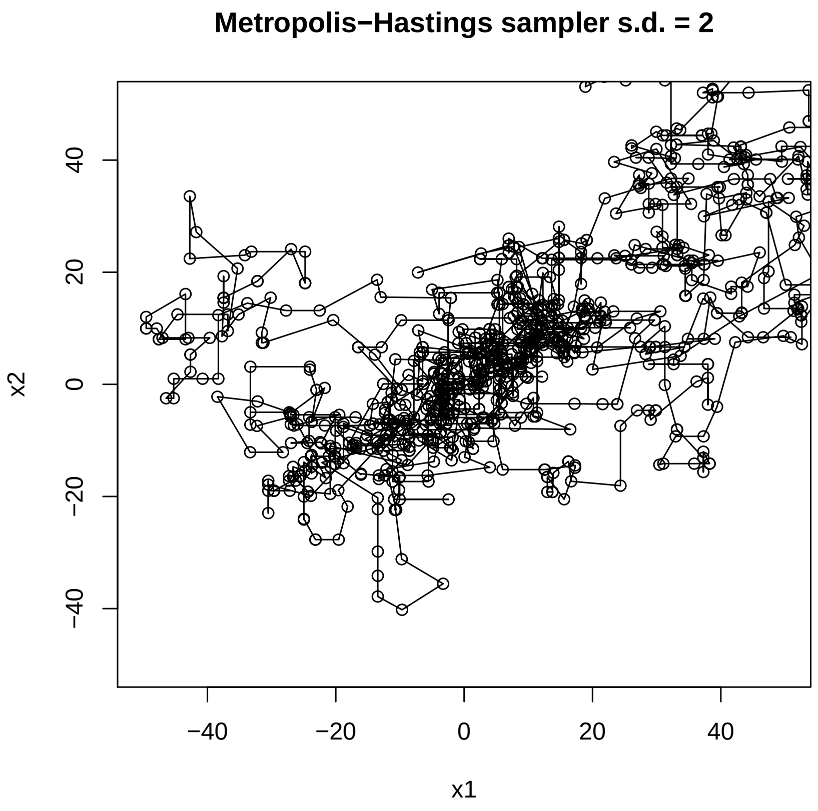


Figure 3.8: *Scatter plot of x1 and x2.*

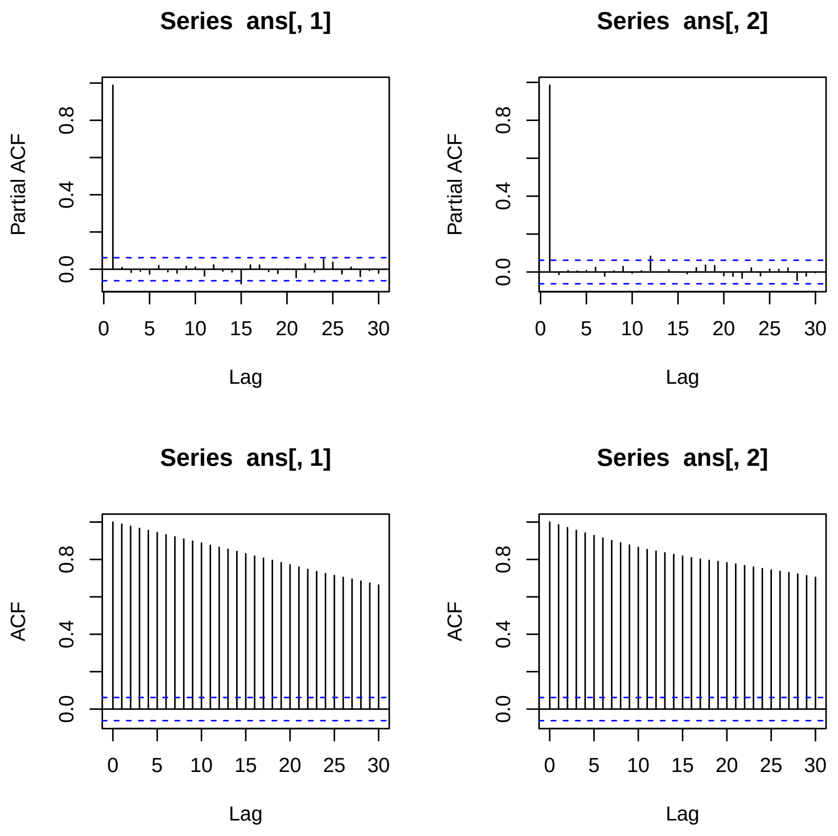


Figure 3.9: *Single component Metropolis-Hastings — autocorrelations*

3.4 Estimating the speed of light

Simon Newcomb set up an experiment in 1882 to measure the speed of light. Newcomb measured the amount of time required for light to travel 7442 metres. The measurements are given in the practicals text:

```
> newcomb <-
+ c(28, 26, 33, 24, 34, -44, 27, 16, 40, -2, 29, 22, 24, 21, 25,
+ 30, 23, 29, 31, 19, 24, 20, 36, 32, 36, 28, 25, 21, 28, 29, 37,
+ 25, 28, 26, 30, 32, 36, 26, 30, 22, 36, 23, 27, 27, 28, 27, 31,
+ 27, 26, 33, 26, 32, 32, 24, 39, 28, 24, 25, 32, 25, 29, 27, 28,
+ 29, 16, 23)
```

1. We first make a histogram of data:

   ```
   > hist( newcomb, breaks=50, col="gray" )
   ```

   A histogram of Newcomb's 66 measured is shown in figure **??**.

   There are two unusually low measurements and then a cluster of measurements that seems to be approximately symmetrically distributed.

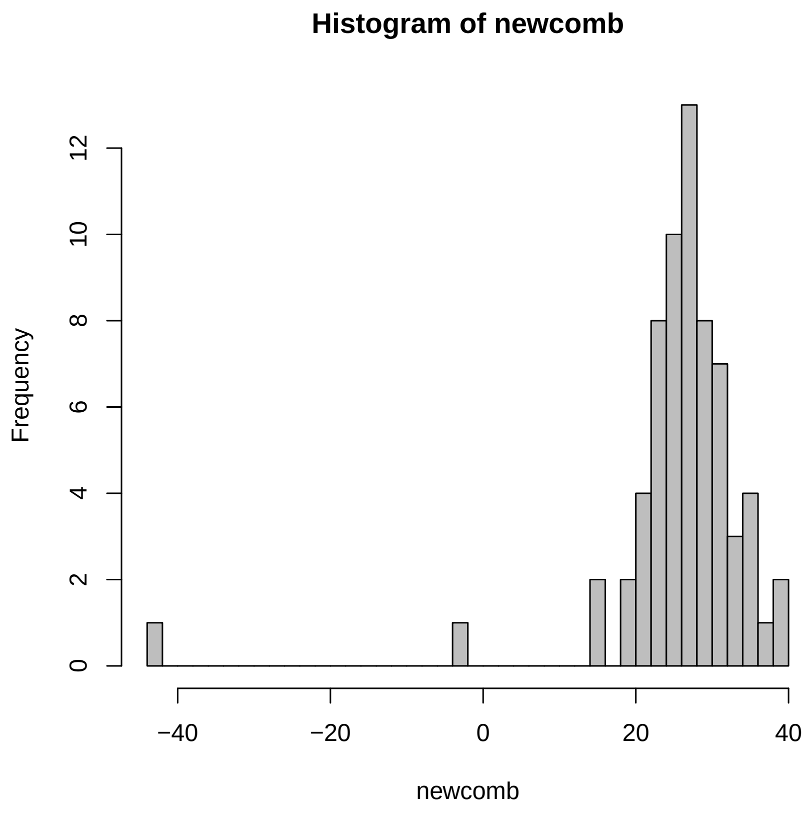


Figure 3.10: *Histogram of Simon Newcomb's measurements for estimating the speed of light, from Stigler SM. (1977). Do robust estimators work with real data? (with discussion).* Annals of Statistics **5**, *1055-1098. The data are times for light to travel a fixed distance, recorded as deviations from 24,800 nanoseconds.*

2. We then (inappropriately!) apply the normal model, assuming that all 66 measurements are independent draws from a normal distribution with mean μ and variance σ^2. The main goal is posterior inference for μ as an estimate of the speed of light (suitablu transformed).

   The sample mean of the $N = 66$ measurements is $\overline{y} = 26.2$, and the sample standard deviation is $s = 10.7$:

```
> mean( newcomb )

[1] 26.21212

> sd( newcomb )

[1] 10.74532
```

3. Assuming the non-informative prior distribution $p(\mu, \sigma^2) \propto (\sigma^2)^{-1}$ (which is equivalent to a joint uniform prior distribution on $(\mu, \log\sigma)$), the posterior distribution of μ has the form

$$\left.\frac{\mu - \overline{y}}{s/\sqrt{n}}\right| \sim t_{n-1}. \tag{3.1}$$

   Note that only μ is unknown in the expression above since we are conditioning on the observed values of the sample mean $\overline{y}$, the sample standard deviation s and the sample size n.

   The 95% posterior credible interval is therefore obtained from the t-distribution:

```
> dev <- qt(0.975,65) * sd(newcomb)/sqrt(length(newcomb))
> mean(newcomb) + c(0,-dev,dev)

[1] 26.21212 23.57059 28.85365
```

4. The posterior interval can also be obtained by simulation. Following the factorisation of the posterior distribution is a scaled inverse-χ^2:

$$p(\sigma^2|y) \sim \chi^{-2}(n-1, s^2),$$

   In order to simulat eform this we first draw a random value of $\sigma^2 \sim \chi^{-2}(65, s^2)$ as $65s^2$ divided by a random draw from the χ^2_{65} distribution, and then draw μ from its conditional posterior distribution:

```
> ybar <- mean(newcomb)
> s     <-    sd(newcomb)
> n   <- length(newcomb)
> numsims <- 1000
> sigma <- sqrt( ((n-1)*(s^2))/( rchisq(numsims,n-1,ncp=0)) )
> mu     <- rnorm( numsims, mean = ybar, sd = sigma/sqrt(n) )
> quantile(    mu, probs=c(5,50,95)/100 )

      5%      50%      95%
24.21086 26.25371 28.40835
```

```
> quantile( sigma, probs=c(5,50,95)/100 )


       5%       50%       95%
 9.385029 10.771879 12.490222
```

5. We can now check the results in questions 1 and 2 by setting up a model in JAGS. We also set up prediction nodes `y.pred`. (The following piece of code also contains a few other things that we will use later.)

```
> cat( "model
+       {
+       for (i in 1:N)
+           {
+                y[i] ~ dnorm(mu,tau)
+           y.pred[i] ~ dnorm(mu,tau)
+           }
+       ord    <- sort( y.pred[] )
+       small1 <- ord[1]
+       small2 <- ord[2]
+       # Priors
+       mu     ~ dnorm(0,0.0001)
+       tau   <- pow(sigma,-2)
+       sigma ~ dunif(0,1000)
+       }",
+       file = "light.jag" )
> light.dat <- list( y=newcomb, N=length(newcomb) )
> light.ini <- list( list( mu=0, sigma=1 ),
+                    list( mu=0, sigma=2 ),
+                    list( mu=0, sigma=3 ) )
> light.par <- c("mu","sigma","small1","small2")
> library( rjags )
> # Model compilation and burn-in
> light.mod <- jags.model( file = "light.jag",
+                          data = light.dat,
+                      n.chains = length( light.ini ),
+                         inits = light.ini,
+                       n.adapt = 5000 )


Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 145

Initializing model


> # Sampling from the posterior
> light.res <- coda.samples( light.mod,
+                                   var = light.par,
+                                n.iter = 10000,
+                                  thin = 30 )
> summary( light.res )


Iterations = 5030:15020
Thinning interval = 30
Number of chains = 3
Sample size per chain = 334

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

         Mean     SD Naive SE Time-series SE
```

```
mu      26.189 1.3680  0.04322        0.04186
sigma   10.957 0.9185  0.02902        0.03087
small1   0.493 5.4949  0.17359        0.17623
small2   4.553 4.2228  0.13340        0.13691

2. Quantiles for each variable:

          2.5%    25%     50%    75%  97.5%
mu      23.530 25.228 26.1753 27.112 28.839
sigma    9.347 10.344 10.8786 11.500 12.877
small1 -12.282 -2.840  0.9742  4.357  9.582
small2  -4.466  2.004  4.8112  7.565 12.210
```

6. From the posterior sample we see that the median is 26.2 and the 95% predictive interval is (23.6;29.0) quite some distance from the actual value (!) of the speed of light.

7. One way to check the suitability of the model is to amend the `JAGS` code from question 3 so that it not only a vector `y.pred` of 66 observations from the normal distribution with the current sampled values of μ and σ, but also retains the two smallest value from the vector `y.pred`, generating a distribution of minimum measurements for a sample of size $N = 66$.

   This was already done in the code, and from the summary we see that the lower bound for the smallest observation is way above the observed smallest value, wheres the second smallest has a quantile not too far from the second smallest observation.

   The conclusion seems to be that the smallest value seen by Newcomb is not consonant with the model chosen. Whether the observation or the model is the culprit is however an open question.

3.5 Modelling the rate of airline fatalities 1976 to 2001

1. (a) The model for the data is:

$$y_i|\theta \sim \text{Poisson}(\theta)$$

where θ is the expected number of fatal accidents in a year.
If the prior distribution for θ is $(\Gamma(\alpha, \beta)$ then the posterior distribution is $\Gamma(\alpha + n\overline{y}, \beta + n)$, where in this case $n = 26$ and $n\overline{y} = \sum_{i=1}^{26} y_i = 634$:

```
> airline <- read.csv( "../data/airline.csv" )
> str( airline )
```

```
'data.frame':        26 obs. of  5 variables:
 $ year1975: int  1 2 3 4 5 6 7 8 9 10 ...
 $ year    : int  1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 ...
 $ fatal   : int  24 25 31 31 22 21 26 20 16 22 ...
 $ miles   : num  3.86 4.3 5.03 5.48 5.81 ...
 $ rate    : num  6.21 5.81 6.17 5.66 3.78 ...
```

```
> sum( airline$fatal )
```

```
[1] 634
```

```
> dim( airline )
```

```
[1] 26  5
```

A noninformative gamma prior distribution has $(\alpha, \beta) = (0, 0)$. This is not a proper distribution — the Γ-density is:

$$f(\theta) = \frac{\beta^\alpha}{\Gamma(\alpha)}\theta^{\alpha-1}\mathrm{e}^{-\beta x}$$

so setting $(\alpha, \beta) = (0, 0)$ specifies a density proportional to $1/\theta$, which is really not possible since $\int_0^{+\infty} 1/\theta \,\mathrm{d}\theta = +\infty$. A density proportional to $1/\theta$ corresponds to a flat prior on $1/\theta$.
However, *provided* the product of the prior and the likelihood results in a proper posterior distribution for θ, (which it does in this case) we can use it.
The posterior distribution is:

$$\theta|y \sim \Gamma(634, 26)$$

and thus the posterior mean for θ is $(\alpha + n\overline{y})/(\beta + n) = 634/26 = 24.385$.

(b) Let $\tilde{y}$ be the number of fatal accidents in 2002. Given θ, the predictive distribution for $\tilde{y}$ is Poisson(θ). The derivation on pages 52 and 53 of *Bayesian Data Analysis* show that the *prior* predictive distribution for y is:

$$\begin{aligned}
p(y) &= \frac{p(y|\theta)p(\theta)}{p(\theta|y)} \\
&= \frac{\text{Poisson}(y|\theta)\text{gamma}(\theta|\alpha, \beta)}{\text{gamma}(\theta|\alpha + y, \beta + 1)} \\
&= \frac{\Gamma(\alpha + y)\beta^\alpha}{\Gamma(\alpha)y!(1 + \beta)^{\alpha+y}} \\
&= \begin{pmatrix} \alpha + y + 1 \\ y \end{pmatrix} \left(\frac{\beta}{\beta + 1}\right)^\alpha \left(\frac{1}{\beta + 1}\right)^y
\end{aligned}$$

which is the *negative binomial* density:

$$y \sim \text{Neg-bin}(\alpha, \beta)$$

For the uninformative prior (*i.e.* with $(\alpha, \beta) = 0, 0)$, this is actually not a distribution, but what we actually want is the *posterior* predictive distribution for the number of fatal accidents in 2002, that is, the predictive distribution conditioning on the available data from 1976 to 2001. This has the same form as $p(y)$ presented above but we must replace α and β with the posterior quantities $\alpha^\star = \alpha + n\overline{y} = 0 + 634 = 634$ and $\beta^\star = \beta + n = 0 + 26 = 26$.

(c) The posterior distribution for θ is $\theta|y \sim \text{Gamma}(634, 26)$, and the conditional distribution of $\tilde{y}$ (the number of fatal accidents in 2002) is Poisson(θ). So to simulate values of $\tilde{y}$ all we need to do is first generate a realized value from the posterior distribution of θ and secondly sample a value from the Poisson distribution using the realized value of θ as the mean. Iterating this process will generate values of $\tilde{y}$ from the posterior predictive distribution. What we are doing here is integrating numerically, using simulation, over the posterior distribution of θ.

This can actually be accomplished in R:

```
> theta <- rgamma(1000, 634, 26 )
> y.2002 <- rpois(1000,theta)
> hist( y.2002 )
```

The default histogram is not impressive; it's actually better to explicitly plot the table of the realized values for y_{2002}:

```
> plot( table(y.2002), type="h", lwd=5, lend=2, col=gray(0.5), bty="n", ylab="" )
```

(d) The model can also be specified in BUGS, and run using the `bugs()` function from `R2WinBUGS`. Besides the model we need starting values and a specification of data:

```
> library(rjags)
> cat( "model
+       {
+       for( i in 1:I )
+         {
+         fatal[i] ~ dpois(mu)
+         }
```

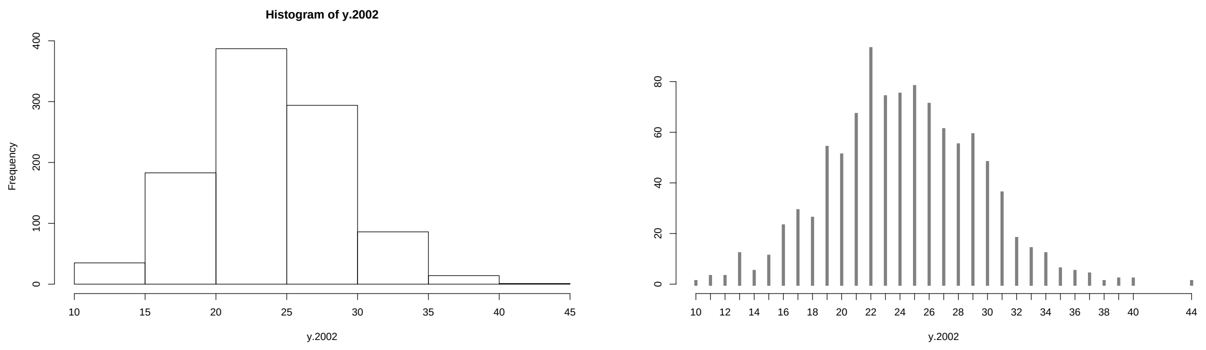


Figure 3.11: *Posterior predictive distribution of y_{2002} — the number of fatal airline crashes in 2002. Left panel the default* `hist()` *and right panel the result of* `plot( ..., type="h")`.

```
+       mu ~ dgamma(0.01,0.01)
+       }",
+      file="a1.jag" )
> a1.par <- c("mu","fatal[27]")
> a1.ini <- list( list( mu=22 ),
+                 list( mu=23 ),
+                 list( mu=24 ) )
> a1.dat <- list( fatal = c(airline$fatal,NA), I=27 )
> # Model compilation and burn-in
> a1.mod <- jags.model( file = "a1.jag",
+                       data = a1.dat,
+                      inits = a1.ini,
+                   n.chains = 3,
+                    n.adapt = 1000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 30

Initializing model

> # Sampling from the posterior
> a1.res <- coda.samples( a1.mod,
+                            var = a1.par,
+                         n.iter = 10000,
+                           thin = 10 )
> summary( a1.res )

Iterations = 10:10000
Thinning interval = 10
Number of chains = 3
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
fatal[27] 24.28 4.9636  0.09062        0.08122
mu        24.39 0.9655  0.01763        0.01897

2. Quantiles for each variable:

           2.5%   25%   50%   75% 97.5%
fatal[27] 15.00 21.00 24.00 27.00 34.03
mu        22.56 23.73 24.38 25.02 26.32
```

The summary of the resulting object shows that the posterior mean and median of the μ is about 24.37. This is also the posterior expectation of the predictive distribution for the number of fatal accidents in 2002, represented by the node `fatal[27]`.

The posterior predictive distribution for the number of fatal accidents in 2002 has median 24 and 95% posterior interval [15,35]. Recall that the posterior predictive distribution is a discrete distribution. We can compare this with the one we simulated directly before:

```
> theta <- rgamma(6000, 634, 26 )
> y.2002 <- rpois(6000,theta)
> plot( table(y.2002), type="h", lwd=5, lend=2, col=gray(0.2), bty="n",
+                       ylab="", xlim=c(5,50) )
> tpr <- table( as.matrix( a1.res[,"fatal[27]"] ) )
> points( as.numeric(names(tpr))+0.4, tpr, type="h", col="red", lwd=4 )
```

2. (a) Let m_i = number of passenger miles flown in year i and λ = accident rate per

passenger mile. The model for the data is $y_i|m_i, \lambda \sim \text{Poisson}(m_i\lambda)$. We use the noninformative $\Gamma(0, 0)$ prior distribution for λ as we did for μ previously.

The posterior distribution for λ is $\lambda|y, m \sim \Gamma(n\bar{y}, n\bar{m}) = \Gamma(634, 275.56)$ where $n\bar{m} = \sum_{i=1}^{26} m_i$:

```
> sum( airline$miles )
[1] 275.564
```

Note that the model is invariant under scaling of m in the sense that if the ms are divided by a factor K then λ is multiplied by K. In this exercise we have used the ms in the units of 10^{11}miles as they are given in the file `airline.csv`.

(b) Given λ, the predictive distribution for $\tilde{y_{2002}}$ is $\text{Poisson}(\lambda m_{2002}) = \text{Poisson}(2 \times 10^{12}\lambda)$. The posterior predictive distribution for $\tilde{y}$ will be (related to the) negative binomial but the algebra is more complex due to the presence of the 2×10^{12} scale factor based on the number of miles flown. SO we let `BUGS` do the hard work — you can see that the change to the `BUGS` code is rather minimal.

Note that we as before add an extra `NA` value to the vector of fatalities, and in order to get a predictive distribution for this an anticipated value for the number of miles flown, in this case 20 ($\times 10^{11}$).

Also note that you cannot stick an expression in as an argument to a distribution; an expression as `fatal[i] dpois(lambda*miles[i])` will cause an error.

```
> cat( "model
+       {
+       for( i in 1:I )
+         {
+         mu[i] <- lambda * miles[i]
+         fatal[i] ~ dpois( mu[i] )
+         }
+       lambda ~ dgamma(0.01,0.01)
+       }",
+      file="a2.jag" )
> a2.ini <- list( list( lambda=10 ),
+                 list( lambda=20 ),
+                 list( lambda=30 ) )
> a2.dat <- list( fatal=c(airline$fatal,NA),
+                 miles=c(airline$miles,20), I=27 )
> a2.par <- c("mu","fatal[27]")
```

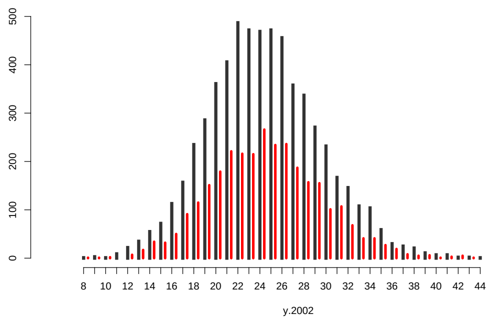


Figure 3.12: *Posterior predictive distribution of y_{2002} — the number of fatal airline crashes in 2002. Gray bars are directly simulated, red bars are the posterior from* `BUGS` *output.*

```
> # Model compilation and burn-in
> a2.mod <- jags.model( file = "a2.jag",
+                       data = a2.dat,
+                      inits = a2.ini,
+                   n.chains = 3,
+                    n.adapt = 1000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 84

Initializing model

> # Sampling from the posterior
> a2.res <- coda.samples( a2.mod,
+                             var = a2.par,
+                          n.iter = 10000,
+                            thin = 10 )
> summary( a2.res )

Iterations = 10:10000
Thinning interval = 10
Number of chains = 3
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
fatal[27] 46.067 7.1790 0.131069       0.126857
mu[1]      8.894 0.3506 0.006402       0.006060
mu[2]      9.900 0.3903 0.007126       0.006746
mu[3]     11.574 0.4563 0.008331       0.007887
mu[4]     12.619 0.4975 0.009083       0.008599
mu[5]     13.386 0.5277 0.009635       0.009121
mu[6]     13.890 0.5476 0.009998       0.009465
mu[7]     13.531 0.5335 0.009740       0.009220
mu[8]     14.327 0.5649 0.010313       0.009763
mu[9]     17.113 0.6747 0.012318       0.011661
mu[10]    16.362 0.6451 0.011778       0.011150
mu[11]    20.951 0.8260 0.015081       0.014277
mu[12]    23.023 0.9077 0.016572       0.015689
mu[13]    24.404 0.9622 0.017567       0.016630
mu[14]    25.298 0.9974 0.018210       0.017239
mu[15]    25.049 0.9876 0.018031       0.017069
mu[16]    24.480 0.9652 0.017621       0.016682
mu[17]    27.526 1.0852 0.019814       0.018757
mu[18]    28.417 1.1204 0.020455       0.019364
mu[19]    29.955 1.1810 0.021562       0.020412
mu[20]    32.739 1.2908 0.023566       0.022309
mu[21]    37.691 1.4860 0.027131       0.025684
mu[22]    35.646 1.4054 0.025659       0.024291
mu[23]    41.625 1.6411 0.029963       0.028365
mu[24]    38.294 1.5098 0.027565       0.026095
mu[25]    43.456 1.7133 0.031280       0.029612
mu[26]    44.280 1.7458 0.031873       0.030174
mu[27]    46.046 1.8154 0.033145       0.031377

2. Quantiles for each variable:

            2.5%    25%    50%    75%  97.5%
fatal[27] 32.000 41.000 46.000 51.000 60.000
mu[1]      8.222  8.644  8.887  9.134  9.604
mu[2]      9.152  9.622  9.892 10.167 10.691
mu[3]     10.700 11.248 11.565 11.886 12.498
mu[4]     11.666 12.264 12.609 12.960 13.627
```

```
mu[5]      12.375 13.009 13.375 13.747 14.455
mu[6]      12.841 13.500 13.879 14.265 14.999
mu[7]      12.509 13.150 13.520 13.896 14.611
mu[8]      13.246 13.925 14.316 14.714 15.472
mu[9]      15.821 16.632 17.100 17.575 18.480
mu[10]     15.127 15.903 16.350 16.804 17.669
mu[11]     19.369 20.362 20.935 21.517 22.624
mu[12]     21.285 22.376 23.005 23.645 24.862
mu[13]     22.562 23.719 24.385 25.063 26.354
mu[14]     23.388 24.587 25.278 25.981 27.318
mu[15]     23.158 24.345 25.029 25.725 27.050
mu[16]     22.632 23.793 24.461 25.141 26.436
mu[17]     25.448 26.753 27.505 28.269 29.725
mu[18]     26.272 27.619 28.395 29.185 30.687
mu[19]     27.694 29.114 29.932 30.764 32.348
mu[20]     30.267 31.819 32.713 33.623 35.354
mu[21]     34.845 36.632 37.661 38.709 40.701
mu[22]     32.955 34.645 35.619 36.609 38.494
mu[23]     38.483 40.456 41.593 42.749 44.950
mu[24]     35.403 37.218 38.264 39.328 41.353
mu[25]     40.175 42.235 43.422 44.629 46.927
mu[26]     40.937 43.036 44.245 45.476 47.817
mu[27]     42.570 44.752 46.010 47.289 49.724
```

The posterior expectation of the predictive distribution for the number of fatal accidents in 2002 is 46 and the 95% posterior interval is [33,60].

3. (a) A closer inspection of the number of fatal airline crashes can be dome by:

```
> par(mfrow=c(1,2))
> with(airline, plot( year, fatal, pch=16, type="b", ylim=c(0,32), bty="n" ) )
> with(airline, plot( year,  rate, pch=16, type="b", ylim=c(0,7), bty="n" ) )
```

There is a decrease *on average* over the ten year period 1976 to 1985. The fatal accident *rate* per mile flown over the 26 year period shows a more consistently decreasing trend that looks amenable to modelling using a (possibly exponentially transformed) simple first order function of time.

(b) The mean of a Poisson random variable must be positive, so modelling the mean as a linear function of time, that is, $\mathrm{E}(y|\mu) = \mu = \alpha + \beta(t - 1990)$ has the potential to generate negative values for μ and thus a mean for our sampling distribution that is outside the parameter space.

In this case it seems to work, however, because the chains never get to generate a negative value of any of the `mu[i]`s:

```
> cat( "model
+       {
+       for( i in 1:I )
+         {
+         mu[i] <- (alpha + beta*(i-10)) * miles[i]
+         fatal[i] ~ dpois( mu[i] )
+         }
+       alpha ~ dnorm(0,0.000001)
+        beta ~ dnorm(0,0.000001)
+       }",
+      file="a3.jag" )
> a3.ini <- list( list( alpha=10, beta=-0.5 ),
+                 list( alpha=20, beta=-0.6 ),
+                 list( alpha=30, beta=-0.4 ) )
> a3.dat <- list( fatal=c(airline$fatal,NA),
+                 miles=c(airline$miles,20), I=27 )
> a3.par <- c("alpha","beta","fatal[27]")
> # Model compilation and burn-in
> a3.mod <- jags.model( file = "a3.jag",
+                       data = a3.dat,
+                      inits = a3.ini,
+                   n.chains = 3,
+                    n.adapt = 1000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 194

Initializing model

> # Sampling from the posterior
> a3.res <- coda.samples( a3.mod,
+                              var = a3.par,
+                           n.iter = 10000,
+                             thin = 10 )
> summary( a3.res )

Iterations = 1010:11000
Thinning interval = 10
Number of chains = 3
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
alpha      3.4252 0.15878 0.0028989      0.0030423
beta      -0.1656 0.01368 0.0002497      0.0002866
fatal[27] 12.1977 4.29838 0.0784774      0.0851265

2. Quantiles for each variable:
```

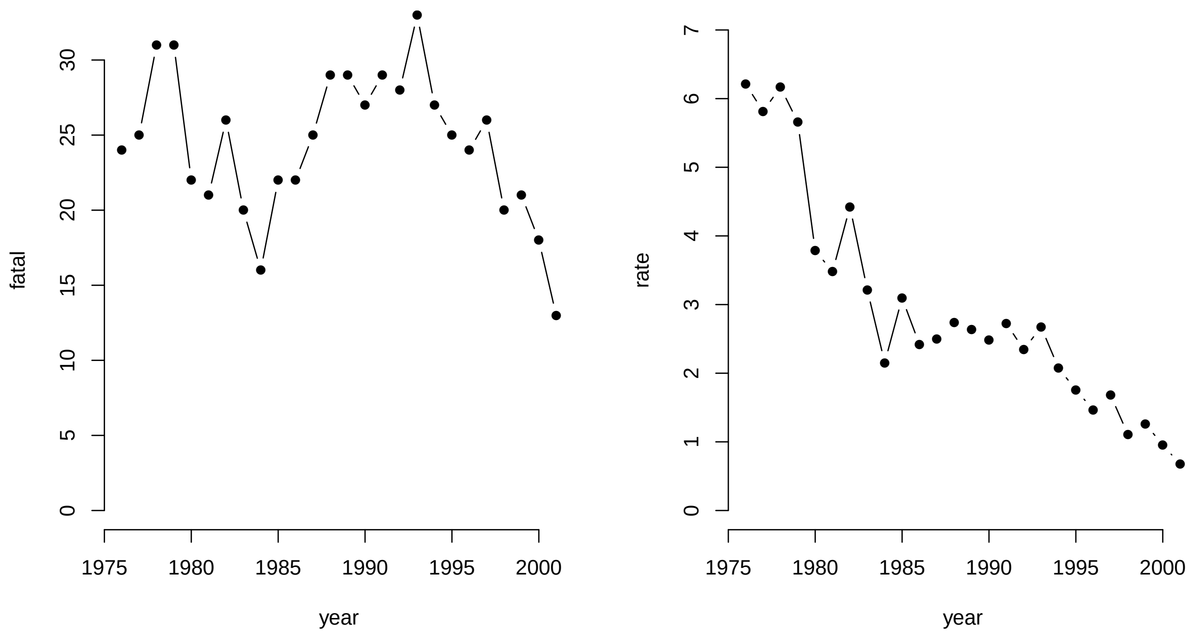


Figure 3.13: *The numbers (left) and rates (right) of fatal airline accidents.*

```
               2.5%     25%     50%     75%   97.5%
alpha        3.1217  3.3188  3.4209  3.5299  3.7559
beta        -0.1929 -0.1746 -0.1658 -0.1564 -0.1385
fatal[27]    5.0000  9.0000 12.0000 15.0000 21.0000
```

Finally we can take a look at traces of the three chains used in this analysis (see figure ??):

```
> print( xyplot( a3.res[,1:2] ) )
```

4. A more natural model is the multiplicative one

$$\log\Big(\mathrm{E}\big(y(t)|t, m(t)\big)\Big) = \alpha + \beta t + \log(m(t)) \tag{3.2}$$

   (a) The simple linear regression approach to the model is to regress the log-rate on the year:

```
> summary( lm( log( fatal/miles ) ~ I(year-1985), data=airline ) )

Call:
lm(formula = log(fatal/miles) ~ I(year - 1985), data = airline)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.46628 -0.14912  0.04327  0.14137  0.37938 

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     1.163059   0.044640   26.05  < 2e-16
I(year - 1985) -0.069878   0.005394  -12.96 2.52e-12

Residual standard error: 0.2063 on 24 degrees of freedom
Multiple R-squared: 0.8749,	Adjusted R-squared: 0.8697 
F-statistic: 167.8 on 1 and 24 DF,  p-value: 2.518e-12
```

   which shows that rates decrease about 7% per year ($\exp(\hat{\beta}) - 1$).

   This model puts equal weight on all observations regardless of the number of fatalities seen, so a proper Poisson-model would presumably be more appropriate.

   (b) The relevant Poisson model is one where the log of the mean is linear, as indicated in the formula (3.2) above. The log of the miles is a regression variable, but with no coefficient, *i.e.* with a regression coefficient fixed at 1. This is a so-called offset-variable:

```
> summary( glm4 <- glm( fatal ~ I(year-1985) + offset(log(miles)),
+                        family=poisson, data=airline ) )

Call:
glm(formula = fatal ~ I(year - 1985) + offset(log(miles)), family = poisson, 
    data = airline)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-2.0782  -0.7953   0.1626   0.7190   1.9369  

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)     1.176111   0.043200   27.23   <2e-16
I(year - 1985) -0.068742   0.005394  -12.74   <2e-16

(Dispersion parameter for poisson family taken to be 1)
```

```
    Null deviance: 182.628  on 25  degrees of freedom
Residual deviance:  22.545  on 24  degrees of freedom
AIC: 157.02

Number of Fisher Scoring iterations: 4
```

This is pretty much the same results as those from the linear regression of the log-rates.

(c) We can now fit the same model using JAGS, by a suitable modification of the code from before:

```
> cat( "model
+        {
+        for( i in 1:I )
+          {
+          mu[i] <- exp( alpha + beta*(i-10) ) * miles[i]
+          fatal[i] ~ dpois( mu[i] )
+          }
+        alpha ~ dnorm(0,0.000001)
+         beta ~ dnorm(0,0.000001)
+        }",
+       file="a4.jag" )
> a4.ini <- list( list( alpha=1.0, beta=-0.05 ),
+                 list( alpha=1.5, beta=-0.06 ),
+                 list( alpha=0.5, beta=-0.04 ) )
> a4.dat <- list( fatal=c(airline$fatal,NA),
+                 miles=c(airline$miles,20), I=27 )
> a4.par <- c("alpha","beta","fatal[27]")
> # Model compilation and burn-in
> a4.mod <- jags.model( file = "a4.jag",
+                       data = a4.dat,
+                      inits = a4.ini,
+                   n.chains = 3,
+                    n.adapt = 1000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 221

Initializing model

> # Sampling from the posterior
> a4.res <- coda.samples( a4.mod,
+                              var = a4.par,
+                           n.iter = 10000,
+                             thin = 10 )
> summary( a4.res )

Iterations = 1010:11000
Thinning interval = 10
Number of chains = 3
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
alpha      1.17519 0.043270 7.900e-04      7.619e-04
beta      -0.06883 0.005381 9.824e-05      8.955e-05
fatal[27] 20.07233 4.777451 8.722e-02      8.337e-02

2. Quantiles for each variable:

              2.5%      25%      50%      75%    97.5%
alpha      1.08755  1.14706  1.17593  1.20350  1.25793
```

```
beta      -0.07961 -0.07244 -0.06881 -0.06519 -0.05833
fatal[27] 11.00000 17.00000 20.00000 23.00000 30.00000
```

If we compare the results with those from the generalized linear model:

```
> library( Epi )
> ci.lin( glm4 )
                   Estimate      StdErr         z P        2.5%       97.5%
(Intercept)      1.17611148 0.043199710  27.22499 0  1.09144161  1.26078136
I(year - 1985) -0.06874189 0.005393721 -12.74480 0 -0.07931339 -0.05817039
```

we see that the asymptotic 95% c.i.s from this model are virtually identical to the 95% posterior interval from the BUGS simulation.

(d) The mixing of the chains for α and β is checked using xyplot on the resulting mcmc.list object. This is placed alongside the corresponding plot for the model with linear trend in the rates:

```
> print( xyplot( a4.res[,1:2] ) )
```

(e) The mixing of the chains for α and β can also be checked by checking whether the densities based on each of the chains look similar:

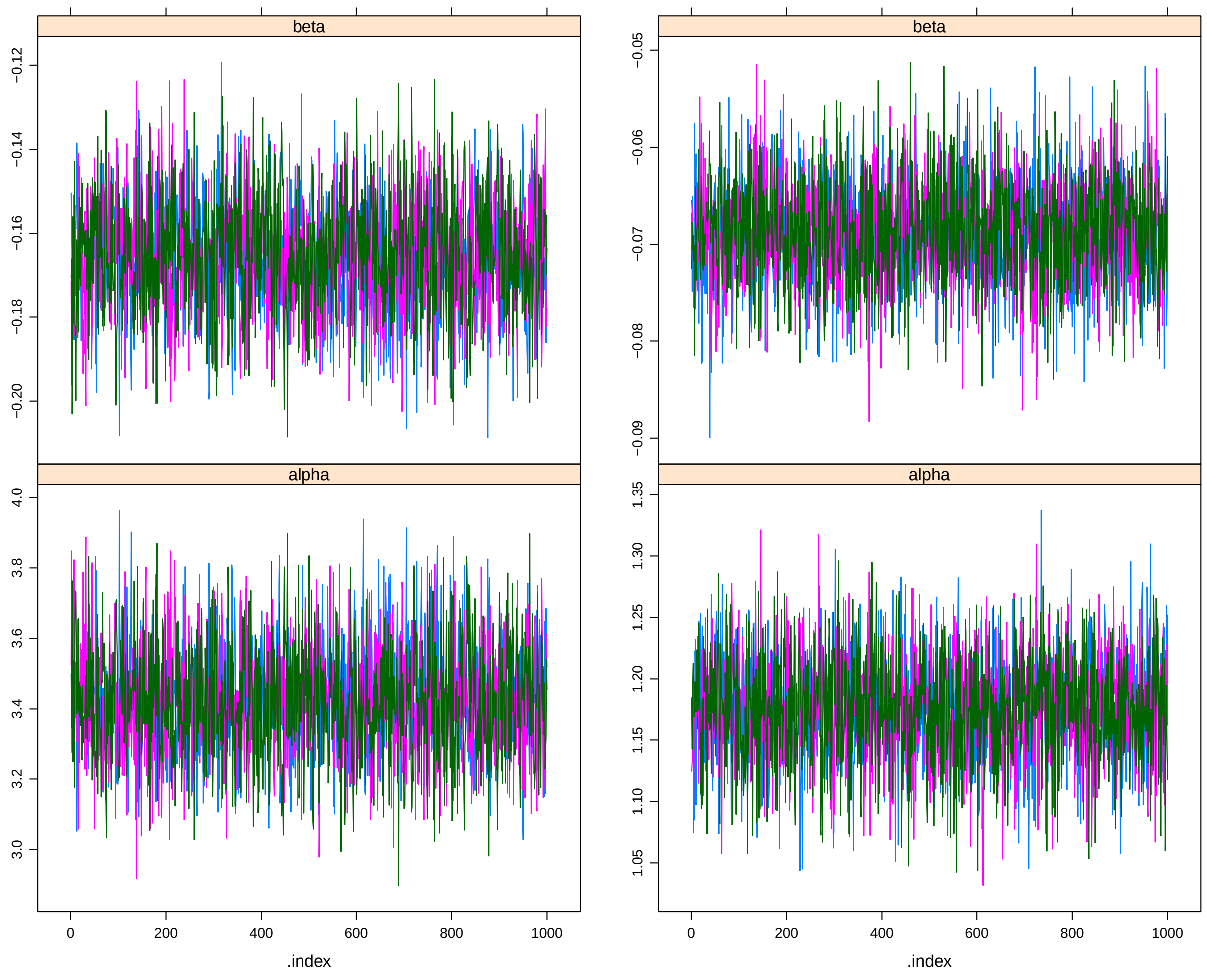


Figure 3.14: *Traceplots of chains from the linear model (left) and the log-linear model (right). For two of the chains in the linear model there is clearly some kind of boundary problems, as two of the chains stay in the same state for longer periods of time.*

```
> print( densityplot( a4.res[,1:2], aspect="fill" ) )
```

Likewise, we may simply plot the simulated values for α and *beta* against each other with different colors:

```
> mat4 <- as.matrix( a4.res, chains=TRUE )
> # permute the rows to get the colors better mixed in the plot
> mat4 <- mat4[sample(1:nrow(mat4)),]
> plot( mat4[,"alpha"], mat4[,"beta"],
+       pch=16, cex=0.3, col=rainbow(3)[mat4[,"CHAIN"]] )
```

(f) If we want the posterior of the *expected* number of airline fatalities in 2002 (assuming the the amount of flown miles is 20×10^{12}), we are asking for the posterior of $exp(\alpha + \beta \times (2002 - 1985)) \times 20$:

```
> a4.m <- as.matrix(a4.res)
> enum.2002 <- exp(a4.m[,"alpha"] + a4.m[,"beta"]*17)*20
> summary( enum.2002 )

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  14.24   18.97   20.09   20.17   21.30   27.45

> ( e2002.qnt <- quantile( enum.2002, probs=c(50,2.5,97.5)/100 ) )

     50%     2.5%    97.5%
20.08800 16.94916 23.76928
```

A plot of the posterior density of this can be obtained using the `density` function:

```
> plot( density(enum.2002), type="l", lwd=3 )
> abline( v=e2002.qnt )
```

(g) The node `fatal[27]` contains the predictive distribution for the number of fatal accidents in 2002. Its posterior mean is 20.04 (similar to that for the expected number of fatal accidents in 2002) with a standard deviation of 4.864 and 95% interval [11,30]. We can plot the distribution of this by:

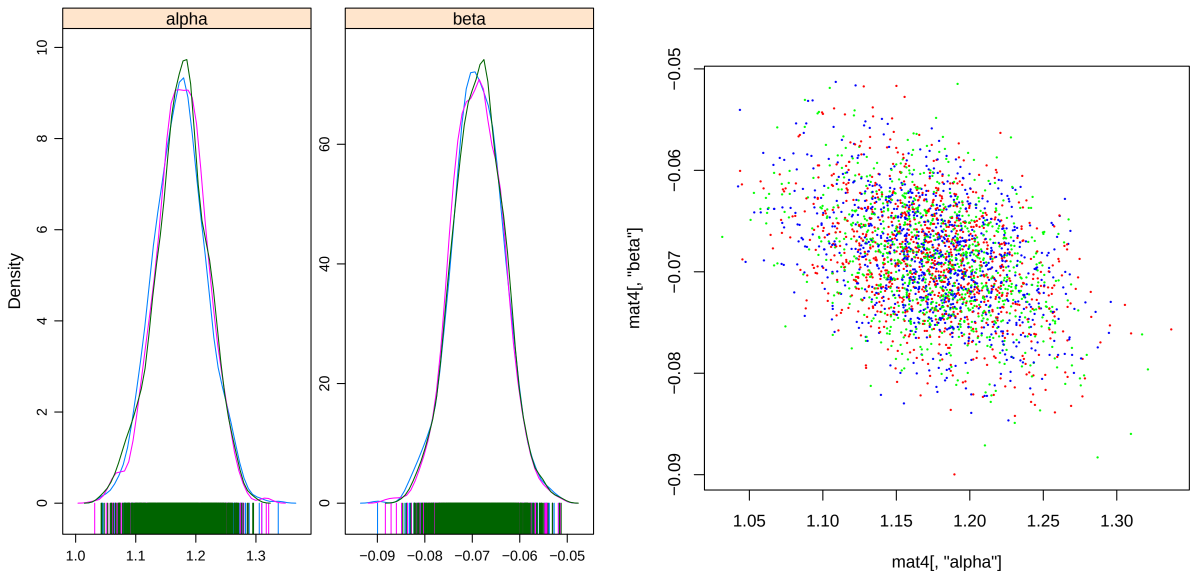


Figure 3.15: *Marginal densities (left) and joint distribution (right) for α and β from the multiplicative model. Results from different chains have different colours.*

```
> plot( table(a4.m[,"fatal[27]"]),
+       type="h", lwd=5, lend=2, col=gray(0.5), bty="n", ylab=""  )
```

As an aside, the actual figures for 2002, 2003 and 2004 are shown in table 3.1. Note that the guess that 20×10^{11} miles would be flown in 2002 was almost spot on! Secondly, the actual number of fatal accidents was 14, less than the 20 predicted from our final model in question 3, but well within the prediction interval of (11,30). Finally, the rate in 2002 (0.708) was similar to that in 2001 (0.676, which was the lowest rate for the series up to that time), but the rates in the final two year 2003 and 2004 (0.3004 and 0.4433 respectively) are about half as great as those in the previous two years. Since 1976, the rate of fatal accidents per air mile flown has decreased by an order of magnitude, that is, it is ten times lower.

(h) To produce the posterior predictive distribution of the number of fatalities in 2002, based on the maximum likelihood estimates from the generalized liner model above, we would simulate the log-rate based on an assumption of multivariate normality of the estimates, or rather based on normality of the parameter function $\alpha + \beta(2002 - 1985)$. Then we simulate a random number from this, take the exponential and multiply by 20 to get a random sample from the posterior mean. Finally we would simulate a Poisson variate with this mean:

```
> # ci.lin gives the estimate and its sd. for a linear combination of parameters
> mn.sd <- ci.lin( glm4, ctr.mat=rbind(c(1,2002-1985)) )[1:2]
> N <- 1000
```

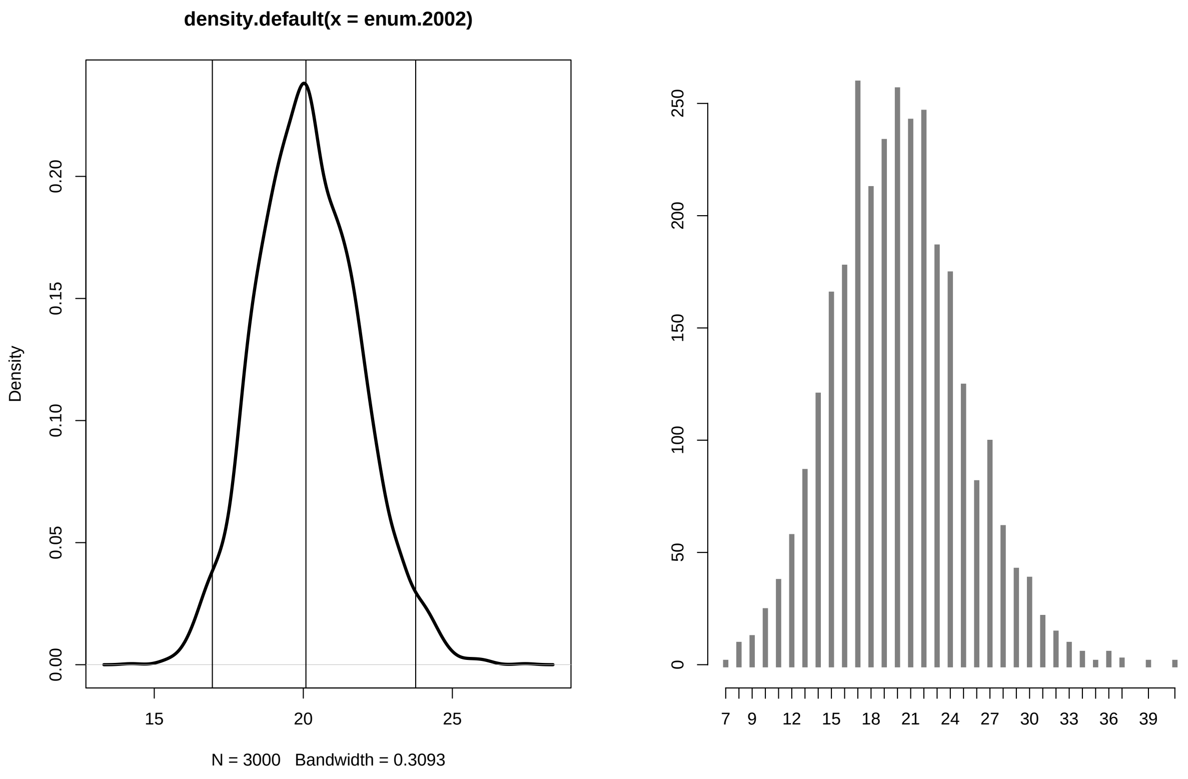


Figure 3.16: *Posterior density of the* expected *number of airline fatalities in 2002 (left) and the posterior predicted number of fatalities in 2002.*

Table 3.1: *Worldwide airline fatalities, 2002–2004. "Passenger miles" are in units of* 10^{11} *and the "Accident rate" is the number of fatal accidents per* 10^{11} *passenger miles. Source: International Civil Aviation Organization, Montreal, Canada (*`www.icao.int`*)*

Year	Fatal accidents	Passenger miles	Accident rate
2002	14	19.775	0.7080
2003	7	23.300	0.3004
2004	9	20.300	0.4433

```
> log.rate <- rnorm( N, mean=mn.sd[1], sd=mn.sd[2] )
> e.num <- exp( log.rate ) * 20
> p.num <- rpois( N, e.num )
> summary( p.num )

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   8.00   17.00   20.00   20.17   23.00   36.00

> quantile( p.num, probs=c(50,2.5,97.5)/100 )

  50%  2.5% 97.5%
   20    11    30

> # For comparison we make the same summary for the posterior sample
> quantile( a4.m[,"fatal[27]"], probs=c(50,2.5,97.5)/100 )

  50%  2.5% 97.5%
   20    11    30
```

3.6 Simple mixed model for fetal growth

The dataset `fetal.csv` contains measurements of head circumference and gestational age, as well as a transformation of gestational age:

```
> fetal <- read.csv("http://BendixCarstensen.com/Bayes/Cph-2012/data/fetal.csv",header=TRUE)
> str( fetal )
```

```
'data.frame':        3097 obs. of  4 variables:
 $ id : int  1 1 1 1 2 2 2 2 2 2 ...
 $ hc : int  211 274 314 330 141 199 266 297 313 321 ...
 $ ga : num  23 28.4 33.4 38.4 17.7 ...
 $ tga: num  16.8 19 20.4 21.2 14.1 ...
```

```
> head( fetal, 10 )
```

```
   id  hc    ga   tga
1   1 211 23.00 16.83
2   1 274 28.43 19.00
3   1 314 33.43 20.39
4   1 330 38.43 21.20
5   2 141 17.71 14.05
6   2 199 22.86 16.76
7   2 266 27.86 18.81
8   2 297 31.29 19.87
9   2 313 34.57 20.63
10  2 321 36.57 20.97
```

1. This is a so-called repeated measures dataset, we see that there are typically 4 or 5 measurements on each fetus, a few only have one measurement and some have as much as 7 measurements:

   ```
   > with( fetal, addmargins( table( table(id) ) ) )
   ```

   ```
     1   2   3   4   5   6    7 Sum
    11  21  82 206 350  28    8 706
   ```

2. We would like a description of the fetal growth as a linear function of time, but this is not a good description; a non-linear transformation of gestational age to make the relationship linear has been estimated: $\texttt{tga} = \texttt{ga} - 0.0116638 \times \texttt{ga}^2$; the transformed gestational age is for convenience put in the variable `tga`:

   ```
   > par( mfrow=c(1,2), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
   > with( fetal, plot( tga, ga-0.0116638*(ga^2), pch=16, cex=0.5 ) )
   > abline(0,1,col="red")
   > with( fetal, plot( ga, tga, pch=16, cex=0.5,
   +                    xlab="Gestational age (GA)", ylab="Transformed GA" ) )
   > abline(0,1,col="red")
   ```

3. The so called spaghetti-plot of a random sample of 100 of the 706 fetuses shows the linearizing effect of the transformation, but also that the square-root transformation of the head circumference makes the relationship more linear and more homogeneous with respect to the variance:

```
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> id.sub <- sample( unique(fetal$id), 50 )
> with( fetal, plot( ga, hc, type="n" ) )
> for( i in id.sub ) with( subset(fetal,id==i), lines(ga,hc) )
> with( fetal, plot( tga, hc, type="n" ) )
> for( i in id.sub ) with( subset(fetal,id==i), lines(tga,hc) )
> with( fetal, plot( tga, sqrt(hc), type="n" ) )
> for( i in id.sub ) with( subset(fetal,id==i), lines(tga,sqrt(hc)) )
```

Also it appears that the overall variance is stabilized. The particular shape of the transformation is illustrated in figure 3.19

4. As a first attempt at the modelling we set up a simple random effects model for the measurement y_{ft} on fetus f at time t:

$$y_{ft} = \beta_0 + \beta_1 t + u_{0f} + e_{ft}$$
$$u_{0f} \sim \mathcal{N}(0, \tau), \quad e_{ft} \sim \mathcal{N}(0, \sigma)$$

This model can be fitted by REML, using the `lmer` function from the `lme4` pa ckage:

```
> library( lme4 )
> m0 <- lmer( sqrt(hc) ~ tga + (1|id), data=fetal )
> summary(m0)
```

```
Linear mixed model fit by REML
Formula: sqrt(hc) ~ tga + (1 | id)
   Data: fetal
  AIC  BIC logLik deviance REMLdev
 1381 1405 -686.6     1355    1373
Random effects:
 Groups   Name        Variance Std.Dev.
 id       (Intercept) 0.059715 0.24437
 Residual             0.062665 0.25033
Number of obs: 3097, groups: id, 706
```

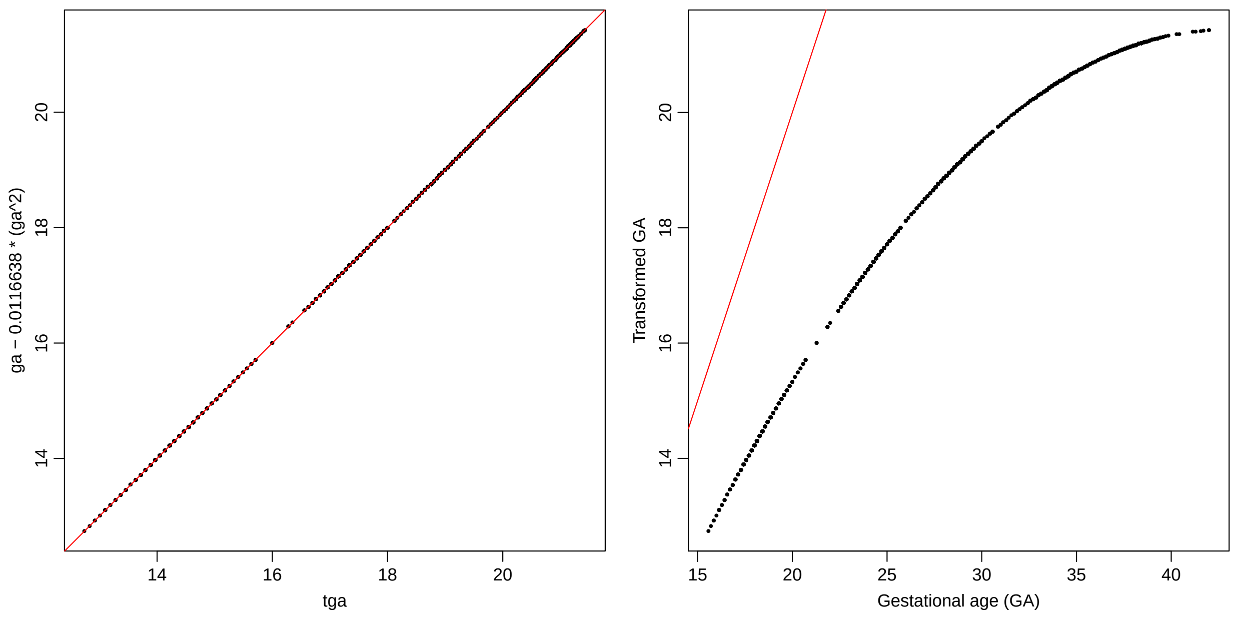


Figure 3.17: *Transformation used for gestational age. The red line is the identity line.*

```
Fixed effects:
             Estimate Std. Error t value
(Intercept) -0.080663   0.036084    -2.2
tga          0.868333   0.001889   459.7

Correlation of Fixed Effects:
    (Intr)
tga -0.958
```

You can extract the estimates and the variances from this using:

```
> fixef( m0 )


(Intercept)         tga
-0.08066286  0.86833250


> VarCorr( m0 )


$id
            (Intercept)
(Intercept)  0.05971482
attr(,"stddev")
(Intercept)
  0.2443662
attr(,"correlation")
            (Intercept)
(Intercept)           1

attr(,"sc")
[1] 0.2503288
```

Note that in order to get the sds out you need (it *is* a little tricky to see where the attributes belong. . .):

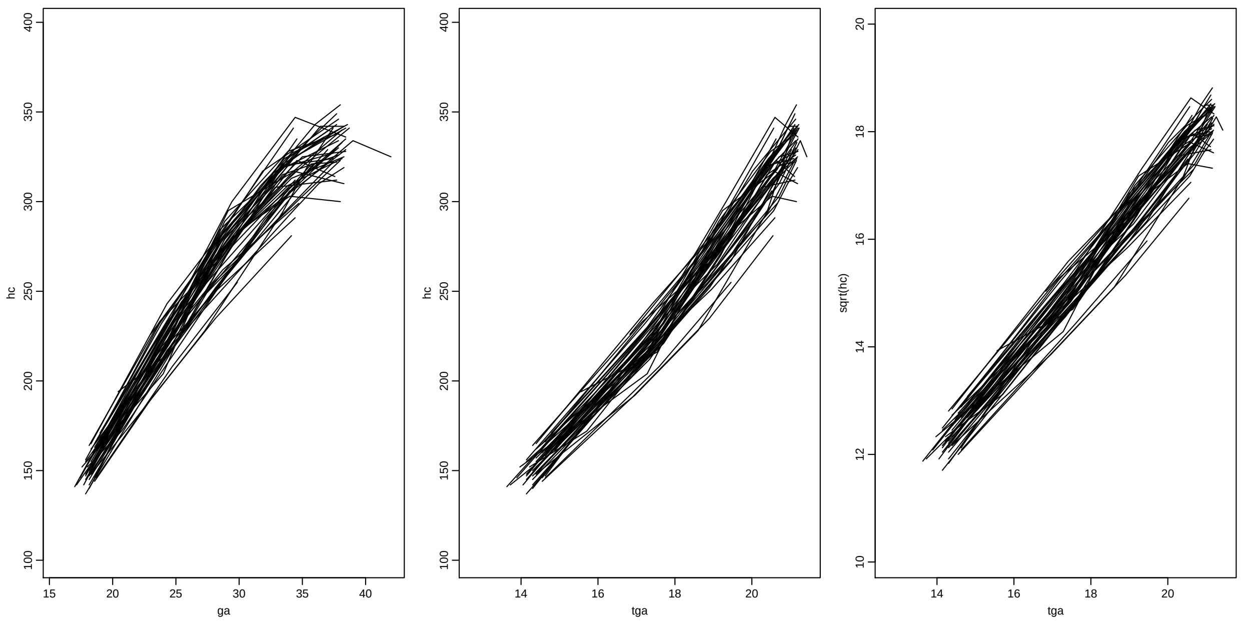


Figure 3.18: *Linearizing transformation of gestational age (quadratic transformation) and head circumference (square root).*

```
> attr( VarCorr(m0)$id, "stddev" )

(Intercept)
  0.2443662

> attr( VarCorr(m0), "sc" )

[1] 0.2503288
```

5. How large is the residual variation relative to the between-persons variation?
6. What is the grovt rate of fetuses' head circumference?
7. This model can be specified in JAGS as follows:

```
> cat("
+ # Fixing data to be used in model definition
+ model
+    {
+    # The model for each observational unit
+      for( j in 1:N )
+      {
+      mu[j] <- beta[1] + beta[2] * ( tga[j]-18 )  + u[id[j]]
+      hc[j] ~ dnorm( mu[j], tau.e )
+      }
+
+    # Random effects for each person
+      for( i in 1:I )
+      {
+      u[i] ~ dnorm(0,tau.u)
+      }
+
+    # Priors:
+
+    # Fixed intercept and slope
+      beta[1] ~ dnorm(0.0,1.0E-5)
+      beta[2] ~ dnorm(0.0,1.0E-5)
+
+    # Residual variance
+      tau.e <- pow(sigma.e,-2)
+    sigma.e  ~ dunif(0,100)
+
+    # Between-person variation
+      tau.u <- pow(sigma.u,-2)
+    sigma.u  ~ dunif(0,100)
+    }",
+      file="fetal0.jag" )
```

Set the model up with suitable initial values (derive them from the `lmer` output. Pay particular attention to the required data supplied to JAGS; note from the code that *two* constants are needed, both the number of units in the dataframe (`N`), but also the number of individuals `I`. The latter can be found using for example:

```
> length( unique(fetal$id) )

[1] 706
```

First we need the data. Note the expression `as.integer(factor(fetal$id))`, which ensures that `id` takes on the values $1, 2, 3, \ldots$, an not just different integer values.

```
> fetal.dat <- list( id = as.integer( factor(fetal$id) ),
+                    hc = fetal$hc,
+                   tga = fetal$tga,
+                     N = nrow(fetal),
+                     I = length( unique(fetal$id) ) )
```

If you inspect the `lmer` object, you can find the estiamtes of the variance componets as follows:

```
> ( sigma.e <- attr(VarCorr(m0),"sc") )

[1] 0.2503288

> ( sigma.u <- attr(VarCorr(m0)$id,"stddev") )

(Intercept)
  0.2443662

> ( beta    <- fixef( m0 ) )

(Intercept)         tga
-0.08066286  0.86833250

> fetal.ini <- list( list( sigma.e = sigma.e/3,
+                          sigma.u = sigma.u/3,
+                             beta = beta   /3 ),
+                    list( sigma.e = sigma.e*3,
+                          sigma.u = sigma.u*3,
+                             beta = beta   *3 ),
+                    list( sigma.e = sigma.e/3,
+                          sigma.u = sigma.u*3,
+                             beta = beta   /3 ),
+                    list( sigma.e = sigma.e*3,
+                          sigma.u = sigma.u/3,
+                             beta = beta   *3 ) )
```

Once we have set up the model-specification, the data and the starting values, we can initialize the model; that is compile the code, and use the inits and the data to run the sampler for a number of iterations

```
> library( rjags )
> system.time(
+ fetal.mod <- jags.model( file = "fetal0.jag",
+                          data = fetal.dat,
+                      n.chains = 4,
+                         inits = fetal.ini,
+                       n.adapt = 100 )
+             )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 13440

Initializing model

   user  system elapsed
    3.4     0.0     3.6
```

With the model in place we now can generate samples from the model using `coda.samples`. In this call we specify which nodes we want to sample. In this case we want to see the posterior distribution of the βs and the variance components:

```
> system.time(
+ fetal.res <- coda.samples( fetal.mod,
+                               var = c("beta","sigma.e","sigma.u"),
+                            n.iter = 500,
+                              thin = 20 ) )
```

```
   user  system elapsed
  13.96    0.00   14.06
```

```
> str( fetal.res )
```

```
List of 4
 $ : mcmc [1:25, 1:4] 159 165 172 178 184 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:4] "beta[1]" "beta[2]" "sigma.e" "sigma.u"
  ..- attr(*, "mcpar")= num [1:3] 120 600 20
 $ : mcmc [1:25, 1:4] 247 247 247 247 246 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:4] "beta[1]" "beta[2]" "sigma.e" "sigma.u"
  ..- attr(*, "mcpar")= num [1:3] 120 600 20
 $ : mcmc [1:25, 1:4] 246 246 246 246 247 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:4] "beta[1]" "beta[2]" "sigma.e" "sigma.u"
  ..- attr(*, "mcpar")= num [1:3] 120 600 20
 $ : mcmc [1:25, 1:4] 247 247 246 246 247 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:4] "beta[1]" "beta[2]" "sigma.e" "sigma.u"
  ..- attr(*, "mcpar")= num [1:3] 120 600 20
 - attr(*, "class")= chr "mcmc.list"
```

```
> summary( fetal.res )
```

```
Iterations = 120:600
Thinning interval = 20
Number of chains = 4
Sample size per chain = 25

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

           Mean       SD Naive SE Time-series SE
beta[1] 240.737 18.71321 1.871321       3.170393
beta[2]  26.516  0.07682 0.007682       0.007997
sigma.e   9.081  0.11900 0.011900       0.009563
sigma.u  12.878 17.03625 1.703625       3.009822

2. Quantiles for each variable:

           2.5%     25%     50%     75%   97.5%
beta[1] 174.643 246.352 246.564 246.882 247.268
beta[2]  26.371  26.466  26.511  26.573  26.680
sigma.e   8.838   9.018   9.085   9.164   9.311
sigma.u   7.104   7.429   7.635   7.859  72.347
```

```
> dim( as.matrix(fetal.res) )

[1] 100   4

> colnames( as.matrix(fetal.res) )

[1] "beta[1]" "beta[2]" "sigma.e" "sigma.u"
```

8. Show the posterior distribution of the between-fetus and the residual standard deviations.

9. How do the estimates for random and fixed effects fit with the `lmer` estimates?

3.7 Linear mixed models for fetal growth

The dataset `fetal.csv` contains measurements of head circumference and gestational age, as well as a transformation of gestational age:

```
> fetal <- read.csv("http://BendixCarstensen.com/Bayes/Cph-2012/data/fetal.csv",header=TRUE)
> str( fetal )
```

```
'data.frame':        3097 obs. of  4 variables:
 $ id : int  1 1 1 1 2 2 2 2 2 2 ...
 $ hc : int  211 274 314 330 141 199 266 297 313 321 ...
 $ ga : num  23 28.4 33.4 38.4 17.7 ...
 $ tga: num  16.8 19 20.4 21.2 14.1 ...
```

```
> head( fetal, 10 )
```

```
   id  hc    ga   tga
1   1 211 23.00 16.83
2   1 274 28.43 19.00
3   1 314 33.43 20.39
4   1 330 38.43 21.20
5   2 141 17.71 14.05
6   2 199 22.86 16.76
7   2 266 27.86 18.81
8   2 297 31.29 19.87
9   2 313 34.57 20.63
10  2 321 36.57 20.97
```

1. This is a so-called repeated measures dataset, we see that there are typically 4 or 5 measurements on each fetus, a few only have one measurement and some have as much as 7 measurements:

   ```
   > with( fetal, addmargins( table( table(id) ) ) )
   ```

   ```
     1   2   3   4   5   6   7 Sum
    11  21  82 206 350  28   8 706
   ```

2. We would like a description of the fetal growth as a linear function of time, but this is not a good description; a non-linear transformation of gestational age to make the relationship linear has been estimated: $\texttt{tga} = \texttt{ga} - 0.0116638 \times \texttt{ga}^2$; the transformed gestational age is for convenience put in the variable `tga`:

   ```
   > par( mfrow=c(1,2), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
   > with( fetal, plot( tga, ga-0.0116638*(ga^2), pch=16, cex=0.5 ) )
   > abline(0,1,col="red")
   > with( fetal, plot( ga, tga, pch=16, cex=0.5,
   +                    xlab="Gestational age (GA)", ylab="Transformed GA" ) )
   > abline(0,1,col="red")
   ```

3. The so called spaghetti-plot of a random sample of 100 of the 706 fetuses shows the linearizing effect of the transformation, but also that the square-root transformation of the head circumference makes the relationship more linear and more homogeneous with respect to the variance:

```
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> id.sub <- sample( unique(fetal$id), 50 )
> with( fetal, plot( ga, hc, type="n" ) )
> for( i in id.sub ) with( subset(fetal,id==i), lines(ga,hc) )
> with( fetal, plot( tga, hc, type="n" ) )
> for( i in id.sub ) with( subset(fetal,id==i), lines(tga,hc) )
> with( fetal, plot( tga, sqrt(hc), type="n" ) )
> for( i in id.sub ) with( subset(fetal,id==i), lines(tga,sqrt(hc)) )
```

Also it appears that the overall variance is stabilized. The particular shape of the transformation is illustrated in figure 3.19

4. As a first attempt at the modelling we set uo a simple random effects model for the measurement y_{ft} on fetus f at time t:

$$y_{ft} = \beta_0 + \beta_1 t + u_{0f} + e_{ft}$$
$$u_{0f} \sim \mathcal{N}(0,\tau), \quad e_{ft} \sim \mathcal{N}(0,\sigma)$$

```
> library( lme4 )
> m0 <- lmer( sqrt(hc) ~ tga + (1|id), data=fetal )
> summary(m0)
```

```
Linear mixed model fit by REML
Formula: sqrt(hc) ~ tga + (1 | id)
   Data: fetal
  AIC  BIC logLik deviance REMLdev
 1381 1405 -686.6     1355    1373
Random effects:
 Groups   Name        Variance Std.Dev.
 id       (Intercept) 0.059715 0.24437
 Residual             0.062665 0.25033
Number of obs: 3097, groups: id, 706

Fixed effects:
             Estimate Std. Error t value
```

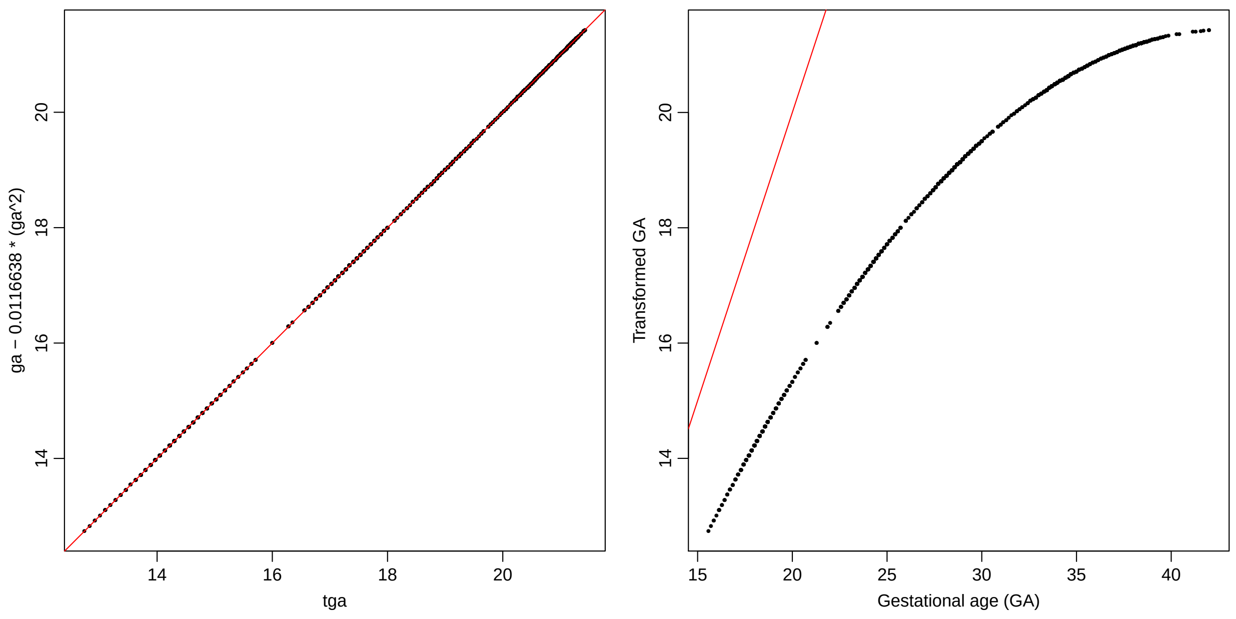


Figure 3.19: *Transformation used for gestational age. The red line is the identity line.*

```
(Intercept) -0.080663   0.036084    -2.2
tga          0.868333   0.001889   459.7

Correlation of Fixed Effects:
    (Intr)
tga -0.958
```

5. We are interested in describing how head circumference varies by the transformed gestational age, but also in describing how growth of the head circumference varies between fetuses. The model of choice is therefore a *linear mixed model* with a random intercept and a random slope term for the measurement y_{ft} on fetus f at time t:

$$y_{ft} = (\beta_0 + u_{0f}) + (\beta_1 + u_{1f})t + e_{ft}$$
$$(u_{0f}, u_{1f}) \sim \mathcal{N}(0, \Sigma), \quad e_{ft} \sim \mathcal{N}(0, \sigma)$$

```
> library( lme4 )
> m0 <- lmer( sqrt(hc) ~ tga + (tga|id), data=fetal )
> summary(m0)
```

```
Linear mixed model fit by REML
Formula: sqrt(hc) ~ tga + (tga | id)
   Data: fetal
  AIC  BIC logLik deviance REMLdev
 1246 1282 -617.1     1217    1234
Random effects:
 Groups   Name        Variance  Std.Dev. Corr
 id       (Intercept) 0.6553008 0.809507
          tga         0.0018567 0.043089 -0.951
 Residual             0.0490636 0.221503
Number of obs: 3097, groups: id, 706

Fixed effects:
             Estimate Std. Error t value
```

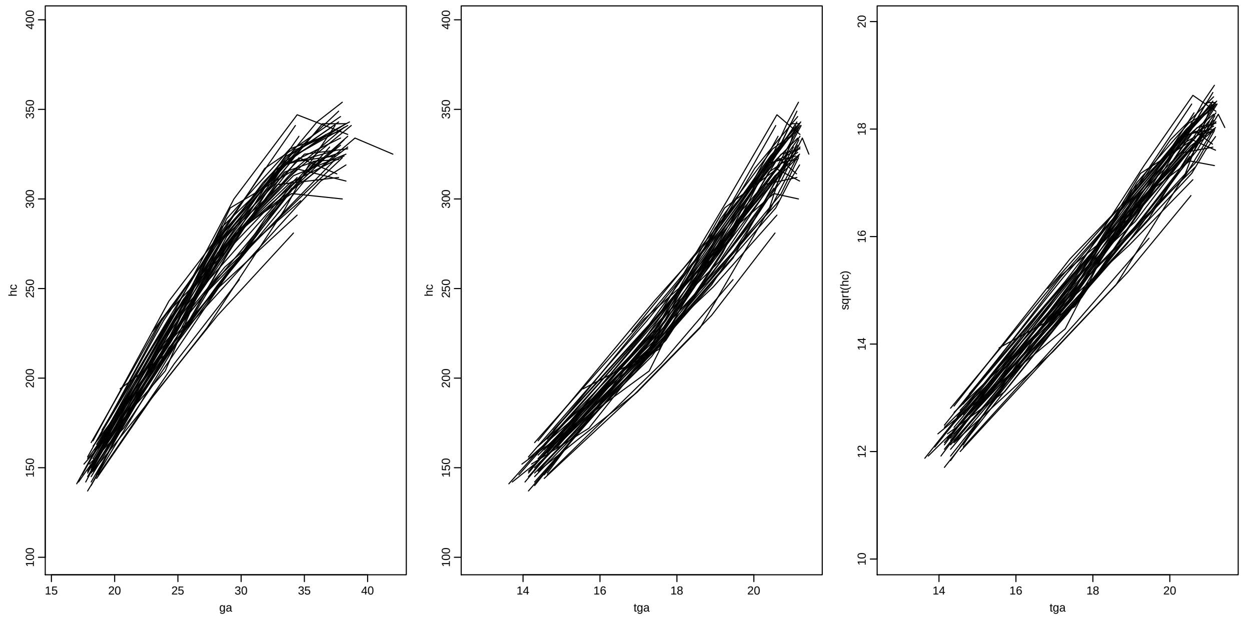


Figure 3.20: *Linearizing transformation of gestational age (quadratic transformation) and head circumference (square root).*

```
(Intercept) -0.084564   0.044378    -1.9
tga          0.868480   0.002386   364.0

Correlation of Fixed Effects:
    (Intr)
tga -0.973
```

6. In the model, Σ is a 2×2 variance-covariance matrix, which we can extract using the `VarCorr` extractor:

```
> VarCorr( m0 )


$id
            (Intercept)         tga
(Intercept)  0.65530084 -0.03318727
tga         -0.03318727  0.00185669
attr(,"stddev")
(Intercept)         tga
 0.80950654  0.04308932
attr(,"correlation")
            (Intercept)        tga
(Intercept)   1.0000000 -0.9514401
tga          -0.9514401  1.0000000

attr(,"sc")
[1] 0.2215031
```

We see that the two random effects u_0 and u_1 are very strongly correlated. This is because u_0 refer to the random level at gestational age 0, which is hardly relevant. Thus it is only the sd. of u_1 which is of relevance; it describes how much the average growth-rates vary between fetuses.

7. It would be more sensible to use for example the median of all measurements, `tga`=18, corresponding to about the 28th week, `ga`=28. This is simply done by centering the variable around this value, corresponding to the model formulation:

$$y_{ft} = (\beta_0 + u_{0f}) + (\beta_1 + u_{1f})\,(t - 18) + e_{ft}$$

To check the adequacy of the square root transformation we fit the model with

```
> m0 <- lmer( hc ~ I(tga-18) + (I(tga-18)|id), data=fetal )
> summary(m0)


Linear mixed model fit by REML
Formula: hc ~ I(tga - 18) + (I(tga - 18) | id)
   Data: fetal
   AIC   BIC logLik deviance REMLdev
 23340 23376 -11664    23324   23328
Random effects:
 Groups   Name        Variance Std.Dev. Corr
 id       (Intercept) 55.9067  7.4771
          I(tga - 18)  1.2153  1.1024   0.641
 Residual             73.7746  8.5892
Number of obs: 3097, groups: id, 706

Fixed effects:
             Estimate Std. Error t value
(Intercept) 246.54773    0.32478   759.1
I(tga - 18)  26.48473    0.07786   340.2
```

```
Correlation of Fixed Effects:
             (Intr)
I(tga - 18) 0.255
```

```
> VarCorr( m0 )
```

```
$id
             (Intercept) I(tga - 18)
(Intercept)   55.906703    5.284188
I(tga - 18)    5.284188    1.215259
attr(,"stddev")
(Intercept) I(tga - 18)
   7.477078    1.102388
attr(,"correlation")
             (Intercept) I(tga - 18)
(Intercept)   1.0000000   0.6410794
I(tga - 18)   0.6410794   1.0000000

attr(,"sc")
[1] 8.589213
```

8. The residuals from this model look substantially more normally distributed for the non-transformed head circumference (figure **??**), so it looks as if

```
> r0 <- residuals(m0)
> par( mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> qqnorm( r0, main="", pch=16,cex=0.6 )
> qqline( r0, col="blue" )
```

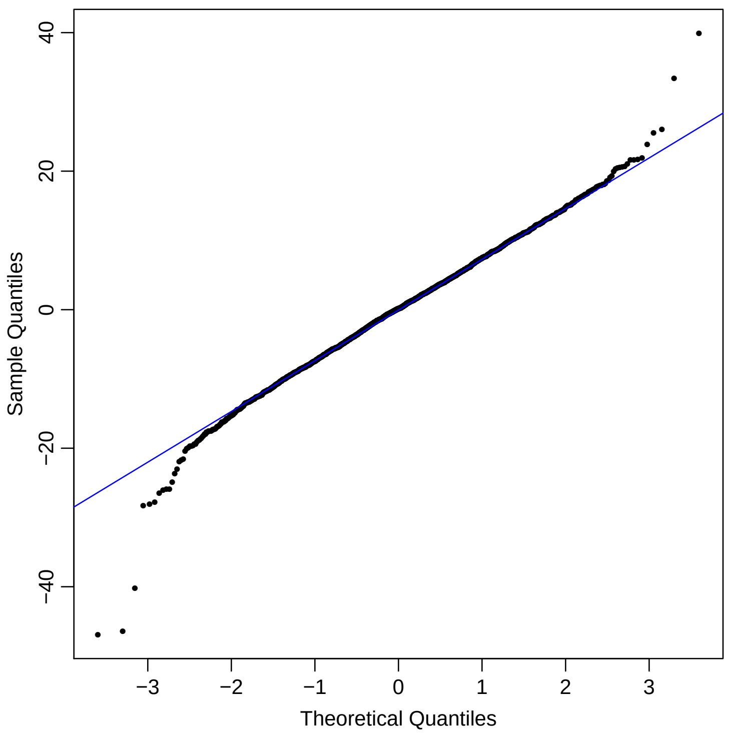


Figure 3.21: *QQ-plot of residuals from the model.*

One missing feature of the output from these models is that there is no handle on the uncertainty of the estimated variance components. This of particular interest when making predictions from the model.

3.7.1 Reporting the model

9. There are two main tings of interest to report from this model:

   (a) The estimated *mean* of head circumference as a function of gestational age, with a confidence interval; that is:

   $$\hat{y}_{ft} = \beta_0 + \beta_1(t - 18)$$

   The confidence interval would be based on the variance-covariance of the βs only.

   (b) A `prediction` interval, that is an interval where you for a *given value* of gestational age would find, say, 95% of the population. The mean would of course be the same, but the interval would be based not only on the variance-covariance of the βs, but also on the estimate of σ and Σ; the variation between individual *in the current study population.*

When we report prediction intervals we are essentially making calculations `as if` the estimated variance components from the model, *sigma* and Σ were known without error and only the βs had an estimation error. In this sense we will presumably be *under*estimating the width of the prediction interval.

We can make these predictions from the output from *lmer*; the *mean* of the head circumference for a given gestational age (for which the transformed value is g_0, say is:

$$\hat{\beta}_0 + \hat{\beta}_1 g_0$$

and the variance of this is:

$$(1 g_0)\Sigma_\beta(1 g_0)'$$

where Σ_β is the estimated variance-covariance of the βs. The latter formula will even work if $(1 g_0)$ is a two-column matrix with a sequence of prediction points. It is automatically computed in the fuction `ci.lin` from the `Epi` package:

```
> library( Epi )
> tga.pt <- 14:22
> ci.lin( m0, ctr.mat=cbind(1,tga.pt) )
```

```
      Estimate   StdErr        z P     2.5%    97.5%
 [1,] 617.3339 1.214076 508.4804 0 614.9544 619.7135
 [2,] 643.8187 1.289445 499.2991 0 641.2914 646.3459
 [3,] 670.3034 1.365094 491.0311 0 667.6278 672.9789
 [4,] 696.7881 1.440978 483.5522 0 693.9638 699.6124
 [5,] 723.2728 1.517063 476.7587 0 720.2994 726.2462
 [6,] 749.7576 1.593319 470.5633 0 746.6347 752.8804
 [7,] 776.2423 1.669724 464.8926 0 772.9697 779.5149
 [8,] 802.7270 1.746257 459.6842 0 799.3044 806.1496
 [9,] 829.2118 1.822903 454.8852 0 825.6389 832.7846
```

Since we are interested in predictions as a function of gestational age, we just define the function that transforms gestational age to the `tga`, so that we can plot the predicted means as a function of gestational age:

```
> tr <- function( ga ) ga-0.0116638*(ga^2)
> ga.pt <- seq(25,41,0.5)
> mn.hc <- ci.lin( m0, ctr.mat=cbind(1,tr(ga.pt)-18) )[,c(1,5,6)]
> matplot( ga.pt, mn.hc, type="l", lty=1, col="black", lwd=c(2,1,1) )
```

10. However we are also interested in making a *population prediction*, that is an interval that for each value of gestational age captures the middle 95% of the fetuses' head circumference.

    To this end we must use not only the estimation variance of the βs, but also the population variance and the residual variance. So if the estimated variance of (u_0, u_1) is Σ_u, and the residual variance is σ_e^2, then the total variance for transformed gestational age g_0 is:

$$(1g_0)\Sigma_\beta(1g_0)' + (1g_0)\Sigma_u(1g_0)' + \sigma_e^2 = (1g_0)(\Sigma_\beta + \Sigma_u)(1g_0)' + \sigma_e^2$$

    We can exstract the two matrices from the model object and use them to construct the relevant standard deviations

```
> Sig.u <- as.matrix( VarCorr( m0 )$id )
> Sig.b <- as.matrix( vcov( m0 ) )
> sig.e <- attr( VarCorr(m0), "sc" )
> pr.var <- diag( cbind(1,tr(ga.pt)-18) %*%
+                         (Sig.u+Sig.b) %*%
```

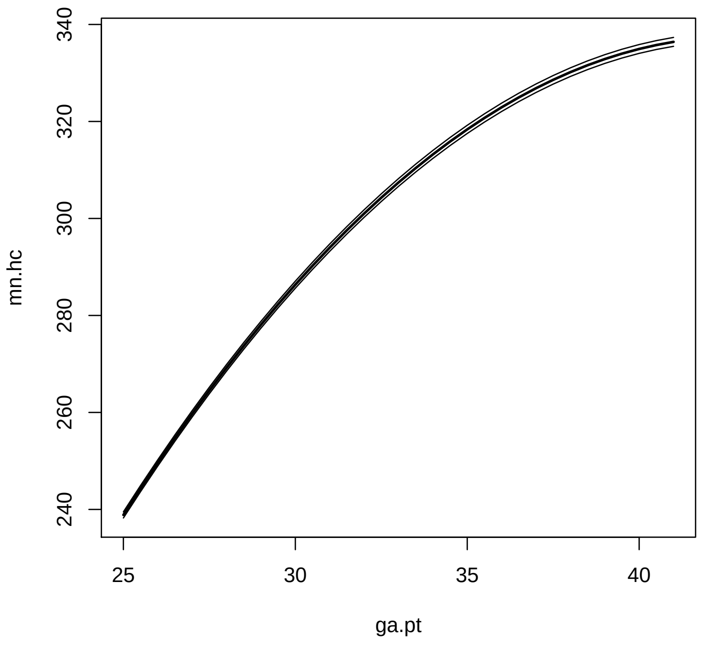


Figure 3.22: *Predicted mean head circumference as function of gestational age. The prediction is a linear function of a quadratic transform of the gestational age.*

```
+                  rbind(1,tr(ga.pt)-18) ) + sig.e
> pr.hc   <- ci.lin( m0, ctr.mat=cbind(1,tr(ga.pt)-18) )[,c(1,5,6,1,1)]
> pr.hc[,4] <- pr.hc[,4] - 1.96*sqrt(pr.var)
> pr.hc[,5] <- pr.hc[,5] + 1.96*sqrt(pr.var)
```

Now we have a 5 column matrix where the first column is the predicted mean head circumference, the two next the confidence interval for the mean and the two last columns the 95% prediction interval.

```
> matplot( ga.pt, pr.hc, type="l", lty=1, col="black", lwd=c(2,1,1,2,2) )
```

Note however that the population terminology can be a bit misleading, because the population that the prediction interval is referring to is the *study population*, so it is only generally interpretable if the study population is representative of some underlying population.

11. The prediction we have just constructed, essentially assumes that the variances are known without error, so we should expect the to be a bit on the small side.

    By using MCMC for estimation we will get a posterior of the joint distribution of β, σ and Σ, meaning that we in the calculation of the prediction interval can use the posterior predictive distribution, which will include the estimation error of the variance components too.

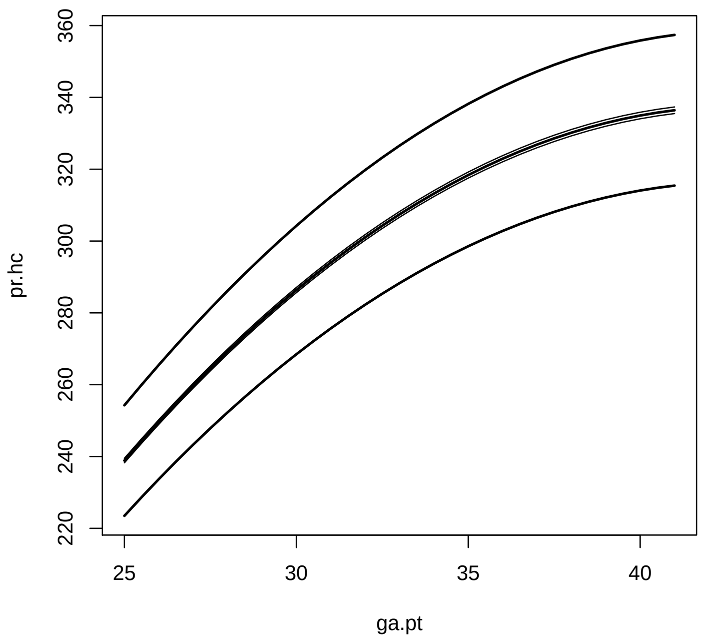


Figure 3.23: *Predicted mean head circumference as function of gestational age. The prediction is a linear function of a quadratic function of gestational age. The outer limits are 95% prediction limits, based on the mixed model.*

3.7.2 Model using JAGS

3.7.2.1 Data

We already have the data needed, in the data frame `fetal`, but we also need the number of rows and the number of items. Note the construction of id: We must use id as a counter in the JAGS code, and hence we must make sure that it takes on the values `1:I`. Also note that we let R compute the number of rows and fetuses

12.
```
> fetal.dat <- list( id = as.integer( factor(fetal$id) ),
+                    hc = fetal$hc,
+                   tga = fetal$tga,
+                     N = nrow(fetal),
+                     F = length( unique(fetal$id) ) )
> str( fetal.dat )
```

```
List of 5
 $ id : int [1:3097] 1 1 1 1 2 2 2 2 2 2 ...
 $ hc : int [1:3097] 211 274 314 330 141 199 266 297 313 321 ...
 $ tga: num [1:3097] 16.8 19 20.4 21.2 14.1 ...
 $ N  : int 3097
 $ F  : int 706
```

3.7.2.2 Model specification

We specify the model that we outlined above, using 18 as the centering point for `tga`,

```
> cat("
+ # Fixing data to be used in model definition
+ data
+    {
+    zero[1] <- 0
+    zero[2] <- 0
+    R[1,1] <- 0.1
+    R[1,2] <- 0
+    R[2,1] <- 0
+    R[2,2] <- 0.5
+    }
+ # Then define model
+ model
+    {
+    # Intercept and slope for each person, including random effects
+      for( f in 1:F )
+      {
+      u[f,1:2] ~ dmnorm(zero,Omega.u)
+      }
+
+    # Define model for each observational unit
+      for( j in 1:N )
+      {
+      mu[j] <- ( beta[1] + u[id[j],1] ) +
+               ( beta[2] + u[id[j],2] ) * ( tga[j]-18 )
+      hc[j] ~ dnorm( mu[j], tau.e )
+      }
+
+    #------------------------------------------------------------
+    # Priors:
+
+    # Fixed intercept and slope
+      beta[1] ~ dnorm(0.0,1.0E-5)
+      beta[2] ~ dnorm(0.0,1.0E-5)
+
+    # Residual variance
+      tau.e <- pow(sigma.e,-2)
+    sigma.e  ~ dunif(0,100)
+
+    # Define prior for the variance-covariance matrix of the random effects
+      Sigma.u <- inverse(Omega.u)
+      Omega.u  ~ dwish( R, 2 )
+    }",
+      file="fetal.jag" )
```

3.7.2.3 Starting values

We can conveniently use as starting values the estimates from the `lmer`; we need starting values for `sigma.e`, `Omega.u` (a 2×2 matrix, the *precision* of the 2-dimensional joint distribution of the random effects), and `beta` (a 2-vector), as these are the quantities (nodes) that are at the top of the graph (DAG), and therefore those for which is relevant to define initial values. Heuristically, the quantities that are on the l.h.s. of a " " in the model specification.

For most well-behaved models (of which this is one), initial values are not needed, as JAGS can generate them on the fly. The purpose of explicitly supplying starting values is to explicitly have the sampling starting at different places in the parameter space, and thus check whether they all lead to the same sable distribution.

In order to find these values we first take a look at what we get when using `VarCorr` to extract variance estimates from the model, the it is clear what we need:

```
> VarCorr( m0 )


$id
            (Intercept) I(tga - 18)
(Intercept)   55.906703    5.284188
I(tga - 18)    5.284188    1.215259
attr(,"stddev")
(Intercept) I(tga - 18)
   7.477078    1.102388
attr(,"correlation")
            (Intercept) I(tga - 18)
(Intercept)   1.0000000   0.6410794
I(tga - 18)   0.6410794   1.0000000

attr(,"sc")
[1] 8.589213
```

We put these into three structures and then use slightly perturbed versions these to define 4 different sets of initial values for 4 chains we run, and then make a list of 4 lists of starting values (since we intend to run 4 chains):

```
> ( sigma.e <- attr(VarCorr(m0),"sc") )


[1] 8.589213


> ( Omega.u <- solve( VarCorr(m0)$id ) )
```

```
            (Intercept) I(tga - 18)
(Intercept)  0.03036744  -0.1320436
I(tga - 18) -0.13204360   1.3970212
```

```
> ( beta    <- fixef( m0 ) )
```

```
(Intercept) I(tga - 18)
  246.54773    26.48473
```

```
> fetal.ini <- list( list( sigma.e = sigma.e/3,
+                          Omega.u = Omega.u/3,
+                             beta = beta   /3 ),
+                    list( sigma.e = sigma.e*3,
+                          Omega.u = Omega.u*3,
+                             beta = beta   *3 ),
+                    list( sigma.e = sigma.e/3,
+                          Omega.u = Omega.u*3,
+                             beta = beta   /3 ),
+                    list( sigma.e = sigma.e*3,
+                          Omega.u = Omega.u/3,
+                             beta = beta   *3 ) )
```

3.7.2.4 Starting the model

Once we have set up the model-specification, the data and the starting values, we can initialize the model; that is compile the code, and use the inits and the data to run the sampler for a number of iterations

```
> library( rjags )
> system.time(
+ fetal.mod <- jags.model( file = "fetal.jag",
+                          data = fetal.dat,
+                      n.chains = 4,
+                         inits = fetal.ini,
+                       n.adapt = 10000 )
+            )
```

```
Compiling data graph
   Resolving undeclared variables
   Allocating nodes
   Initializing
   Reading data back into data table
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 19203

Initializing model

   user  system elapsed
 353.61    0.02  361.90
```

3.7.2.5 Sampling from the model

With the model in place we now can generate samples from the model using `coda.samples`. In this call we specify which nodes we want to sample. In this case we want to see the posterior distribution of the βs and the variance components:

```
> system.time(
+ fetal.res <- coda.samples( fetal.mod,
+                                  var = c("beta","sigma.e","Sigma.u"),
+                               n.iter = 5000,
+                                 thin = 20 ) )
```

```
   user  system elapsed
 139.26    0.00  139.63
```

```
> str( fetal.res )
```

```
List of 4
 $ : mcmc [1:250, 1:7] 56.9 51.8 53.4 56.8 57.3 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:7] "Sigma.u[1,1]" "Sigma.u[2,1]" "Sigma.u[1,2]" "Sigma.u[2,2]" ...
  ..- attr(*, "mcpar")= num [1:3] 10020 15000 20
 $ : mcmc [1:250, 1:7] 60.4 55.9 62.2 49.8 65.4 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:7] "Sigma.u[1,1]" "Sigma.u[2,1]" "Sigma.u[1,2]" "Sigma.u[2,2]" ...
  ..- attr(*, "mcpar")= num [1:3] 10020 15000 20
 $ : mcmc [1:250, 1:7] 52.1 63.8 65.6 56 54.3 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:7] "Sigma.u[1,1]" "Sigma.u[2,1]" "Sigma.u[1,2]" "Sigma.u[2,2]" ...
  ..- attr(*, "mcpar")= num [1:3] 10020 15000 20
 $ : mcmc [1:250, 1:7] 53.9 56.5 59.1 56.1 59.4 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:7] "Sigma.u[1,1]" "Sigma.u[2,1]" "Sigma.u[1,2]" "Sigma.u[2,2]" ...
  ..- attr(*, "mcpar")= num [1:3] 10020 15000 20
 - attr(*, "class")= chr "mcmc.list"
```

```
> summary( fetal.res )
```

```
Iterations = 10020:15000
Thinning interval = 20
Number of chains = 4
Sample size per chain = 250

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                Mean      SD Naive SE Time-series SE
Sigma.u[1,1]  55.457 3.96865 0.125500       0.115709
Sigma.u[2,1]   5.429 0.69794 0.022071       0.026249
Sigma.u[1,2]   5.429 0.69794 0.022071       0.026249
Sigma.u[2,2]   1.082 0.23686 0.007490       0.014858
beta[1]      246.562 0.32137 0.010163       0.009616
beta[2]       26.481 0.07759 0.002454       0.002193
sigma.e        8.656 0.14678 0.004642       0.006075

2. Quantiles for each variable:
```

```
                   2.5%      25%     50%     75%   97.5%
Sigma.u[1,1]  48.2173  52.6885  55.295  58.119  63.691
Sigma.u[2,1]   4.0314   4.9611   5.413   5.834   6.804
Sigma.u[1,2]   4.0314   4.9611   5.413   5.834   6.804
Sigma.u[2,2]   0.6562   0.9086   1.075   1.235   1.555
beta[1]      245.9151 246.3473 246.578 246.765 247.182
beta[2]       26.3308  26.4276  26.480  26.533  26.638
sigma.e        8.3835   8.5569   8.657   8.757   8.953
```

```
> dim( as.matrix(fetal.res) )
```

```
[1] 1000    7
```

```
> colnames( as.matrix(fetal.res) )
```

```
[1] "Sigma.u[1,1]" "Sigma.u[2,1]" "Sigma.u[1,2]" "Sigma.u[2,2]" "beta[1]"
[6] "beta[2]"      "sigma.e"
```

Once we have the posterior samples we can look at the joint distribution of the βs:

```
> plot( fetal.res )
```

Fore better control of the plotting of the posterior samples we can convert the resulting `mcmc.list` object to a data frame. We need to doctor the names in order to be able to refer to them without too much fuss:

```
> fetal.post <- as.data.frame( as.matrix( fetal.res ) )
> names( fetal.post )
```

```
[1] "Sigma.u[1,1]" "Sigma.u[2,1]" "Sigma.u[1,2]" "Sigma.u[2,2]" "beta[1]"
[6] "beta[2]"      "sigma.e"
```

```
> names( fetal.post ) <- gsub( "\\[", ".", names(fetal.post) )
> names( fetal.post ) <- gsub( ",", ".", names(fetal.post) )
> names( fetal.post ) <- gsub( "\\]", "", names(fetal.post) )
> str( fetal.post )
```

```
'data.frame':        1000 obs. of  7 variables:
 $ Sigma.u.1.1: num  56.9 51.8 53.4 56.8 57.3 ...
 $ Sigma.u.2.1: num  5.72 4.92 5.88 5.7 5.75 ...
 $ Sigma.u.1.2: num  5.72 4.92 5.88 5.7 5.75 ...
 $ Sigma.u.2.2: num  1.026 1.021 1.345 0.961 0.965 ...
 $ beta.1     : num  246 246 247 247 247 ...
 $ beta.2     : num  26.4 26.4 26.5 26.4 26.4 ...
 $ sigma.e    : num  8.88 8.69 8.62 8.79 8.73 ...
```

```
> with( fetal.post, plot( beta.1, beta.2, pch=16,cex=0.9,
+                         col=c("black","red","green","blue")[rep(1:4,each=250)] ) )
```

3.7.3 Predictive distributions

13. One of the features of JAGS is the ability to generate predictive distributions for unobserved quantities by specifying these quantities as nodes in the graphical model used by JAGS to generate the simulations.

    We compare the unconditional predictive distribution of head circumference at 38 weeks gestational age with the corresponding *conditional* distribution given the value of the head circumference at 18 weeks gestational age.

    The five observations made on fetus `id = 5` are:

```
> subset( fetal, id==5 )
```

```
   id  hc    ga   tga
18  5 125 18.43 14.47
19  5 232 24.43 17.47
20  5 297 28.43 19.00
21  5 323 34.43 20.60
22  5 338 38.43 21.20
```

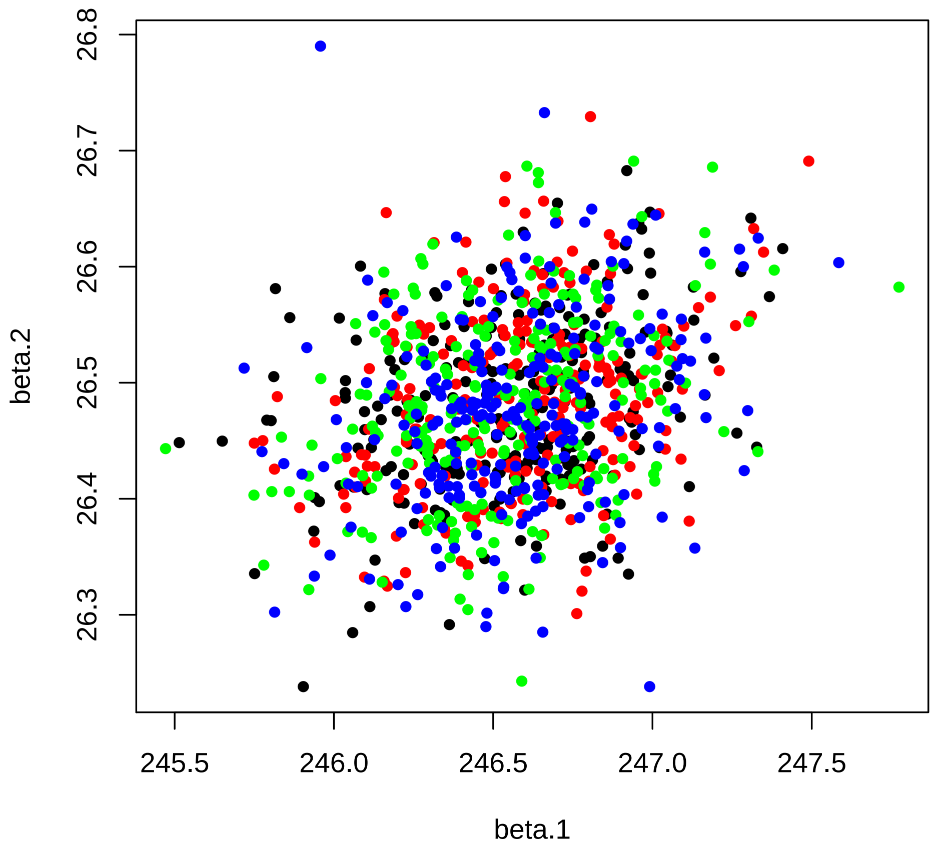


Figure 3.24: *Joint posterior distribution of β_0 and β_1.*

We can get the conditional distribution of head circumference at the final gestational age (38.43 weeks) given the observed measurement at gestational age of 18.43 weeks by creating a new id with identical data for the first gestational age but no observed head circumferences measurements at the final gestational age:

```
> ( xf <- subset( fetal, id==5 )[c(1,5),] )
```

```
   id  hc    ga   tga
18  5 125 18.43 14.47
22  5 338 38.43 21.20
```

```
> xf[2,"hc"] <- NA
> xf[,"id"] <- max(fetal$id)+1
> xf
```

```
    id  hc    ga   tga
18 708 125 18.43 14.47
22 708  NA 38.43 21.20
```

We also add a new observation for a second new fetus to generate the *unconditional* distribution of head circumference at 38.43 weeks gestational age. This is simply replicating the last record but using a new id:

```
> xf <- xf[c(1,2,2),]
> xf[3,"id"] <- max(fetal$id)+2
> xf
```

```
      id  hc    ga   tga
18   708 125 18.43 14.47
22   708  NA 38.43 21.20
22.1 709  NA 38.43 21.20
```

14. Finally we want to make population predictions for gestational weeks as defined in the vector `ga.pt`. This can be done in two ways, one by assuming that we look at the same fetus at all times; the other by making separate predictions for each time:

```
> x.same <- data.frame( id = max(fetal$id)+3,
+                       hc = NA,
+                       ga = ga.pt,
+                      tga = tr(ga.pt) )
> x.diff <- data.frame( id = max(fetal$id)+3+1:length(ga.pt),
+                       hc = NA,
+                       ga = ga.pt,
+                      tga = tr(ga.pt) )
```

In order to get the predicted values we simply monitor the relevant nodes after using JAGS on the dataset expanded with these extra records:

```
> fetal.x <- rbind( fetal, xf, x.same, x.diff )
> fetal.x[nrow(fetal)+0:10,]
```

```
       id  hc    ga      tga
3097  707 348 37.57 21.11000
18100 708 125 18.43 14.47000
22100 708  NA 38.43 21.20000
22.1  709  NA 38.43 21.20000
3101  710  NA 25.00 17.71012
```

```
3102  710   NA 25.50 17.91561
3103  710   NA 26.00 18.11527
3104  710   NA 26.50 18.30910
3105  710   NA 27.00 18.49709
3106  710   NA 27.50 18.67925
3107  710   NA 28.00 18.85558
```

```
> tail( fetal.x )
```

```
      id hc   ga      tga
3161 738 NA 38.5 21.21133
3162 739 NA 39.0 21.25936
3163 740 NA 39.5 21.30156
3164 741 NA 40.0 21.33792
3165 742 NA 40.5 21.36845
3166 743 NA 41.0 21.39315
```

```
> nrow( fetal.x )
```

```
[1] 3166
```

However, there is one more snag to this as we are interested in seeing *prediction* intervals, that is predictions for individual measurements, *including* the measurement errors, in the JAGS code those with precision `tau.e`. And this error term is not included in the nodes `mu`, so we must define a set of new prediction nodes, `pr`, say, to give predictions where the residual error term is included. This is done in this piece of code where we only define the `pr` nodes only for the added units where we want the predictions. In turn that requires an extra constant in data, `n`, the index of the first.

```
> cat("
+ # Fixing data to be used in model definition
+ data
+    {
+    zero[1] <- 0
+    zero[2] <- 0
+    R[1,1] <- 0.1
+    R[1,2] <- 0
+    R[2,1] <- 0
+    R[2,2] <- 0.5
+    }
+ # Then define model
+ model
+    {
+    # Intercept and slope for each person, including random effects
+      for( f in 1:F )
+      {
+      u[f,1:2] ~ dmnorm(zero,Omega.u)
+      }
+
+    # Define model for each observational unit
+      for( j in 1:N )
+      {
+      mu[j] <- ( beta[1] + u[id[j],1] ) +
+               ( beta[2] + u[id[j],2] ) * ( tga[j]-18 )
+      hc[j] ~ dnorm( mu[j], tau.e )
+      }
+      for( j in n:N )
+      {
+      pr[j] ~ dnorm( mu[j], tau.e )
+       }
+
+     #------------------------------------------------------------
+     # Priors:
+
+     # Fixed intercept and slope
+       beta[1] ~ dnorm(0.0,1.0E-5)
+       beta[2] ~ dnorm(0.0,1.0E-5)
+
+     # Residual variance
+       tau.e <- pow(sigma.e,-2)
+     sigma.e  ~ dunif(0,100)
+
+     # Define prior for the variance-covariance matrix of the random effects
+       Sigma.u <- inverse(Omega.u)
+       Omega.u  ~ dwish( R, 2 )
+     }",
+       file="fetalp.jag" )
```

Thus we see that the nodes we are interested in monitoring are (refer to the model definition) `mu[*]` with `*` from 3098 and upwards, so we modify the code and supply the relevant parameters to monitor:

```
> fetal.xdat <- list( id = as.integer( factor(fetal.x$id) ),
+                     hc = fetal.x$hc,
+                    tga = fetal.x$tga,
+                      n = nrow(fetal)+1,
+                      N = nrow(fetal.x),
+                      F = length( unique(fetal.x$id) ) )
> system.time(
+ fetal.xmod <- jags.model( file = "fetalp.jag",
+                           data = fetal.xdat,
+                       n.chains = 4,
+                          inits = fetal.ini,
+                        n.adapt = 5000 )
+            )

Compiling data graph
   Resolving undeclared variables
   Allocating nodes
   Initializing
   Reading data back into data table
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 19831

Initializing model

   user  system elapsed
 137.45    0.00  137.65
```

Once the code has been modified, we need to specify the nodes we shall monitor:

```
> rng <- (nrow(fetal)+1):nrow(fetal.x)
> ( mus <- paste("pr[",paste(range(rng),collapse=":"),"]",sep="") )

[1] "pr[3098:3166]"


> system.time(
+ fetal.xres <- coda.samples( fetal.xmod,
+                               var = c("beta","sigma.e","Sigma.u",mus),
+                            n.iter = 5000,
+                              thin = 10 ) )
```

```
   user  system elapsed
 140.83    0.00  141.14
```

```
> fetal.qnt <- summary( fetal.xres )$quantiles
> pr.rows <- rownames(fetal.qnt)[grep( "pr", rownames(fetal.qnt) )]
> wh <- as.numeric( gsub( "\\]","", gsub("pr\\[","", pr.rows ) ) )
> cbind( fetal.x[wh,c("ga","tga")], fetal.qnt[pr.rows,c(1,3,5)] )
```

```
         ga      tga     2.5%      50%    97.5%
18100 18.43 14.47000 126.2352 144.6005 164.1989
22100 38.43 21.20000 294.2535 321.1074 344.6295
22.1  38.43 21.20000 305.6130 331.1849 359.5924
3101  25.00 17.71012 216.2927 239.3167 260.7162
3102  25.50 17.91561 222.3881 244.3683 266.1499
3103  26.00 18.11527 226.8225 249.8379 272.4841
3104  26.50 18.30910 232.3598 254.4789 277.2190
3105  27.00 18.49709 237.6747 259.9278 282.2659
3106  27.50 18.67925 241.6087 264.9047 287.6344
3107  28.00 18.85558 245.5764 269.8093 292.8567
3108  28.50 19.02608 250.2754 273.9732 296.3040
3109  29.00 19.19074 254.7185 278.1077 301.5751
3110  29.50 19.34958 257.8318 282.7986 306.3199
3111  30.00 19.50258 261.4806 286.2278 310.3895
3112  30.50 19.64975 267.0895 290.3483 313.9614
3113  31.00 19.79109 268.9825 293.5870 319.2532
3114  31.50 19.92659 273.7772 298.0401 322.0668
3115  32.00 20.05627 276.1805 300.8072 327.5909
3116  32.50 20.18011 279.4606 304.1132 329.4959
3117  33.00 20.29812 281.6693 307.1975 332.6382
3118  33.50 20.41030 285.7030 310.3834 334.6220
3119  34.00 20.51665 287.5722 313.3602 339.3332
3120  34.50 20.61716 289.7455 316.0468 340.4908
3121  35.00 20.71185 293.2163 318.7426 343.4868
3122  35.50 20.80070 294.3108 320.6014 345.9549
3123  36.00 20.88372 297.2899 323.3953 346.5127
3124  36.50 20.96090 299.1344 324.7973 350.1196
3125  37.00 21.03226 301.3014 326.6979 351.9439
3126  37.50 21.09778 301.5999 329.0102 354.2413
3127  38.00 21.15747 304.5223 331.0616 356.4948
3128  38.50 21.21133 304.9845 331.5407 356.7227
3129  39.00 21.25936 306.5222 332.8379 359.4839
3130  39.50 21.30156 307.4480 333.6652 360.5283
3131  40.00 21.33792 308.7203 334.4988 360.9192
3132  40.50 21.36845 308.7327 335.6478 361.1650
3133  41.00 21.39315 309.5761 336.7632 361.6987
3134  25.00 17.71012 217.2892 239.1251 259.9514
3135  25.50 17.91561 221.5644 244.8234 266.7026
3136  26.00 18.11527 227.4420 249.5053 272.4663
3137  26.50 18.30910 233.1976 255.1205 278.0360
3138  27.00 18.49709 236.3580 260.3838 283.5945
3139  27.50 18.67925 241.8327 264.1662 287.8413
3140  28.00 18.85558 246.2428 269.2748 292.4904
3141  28.50 19.02608 248.9783 273.5224 298.1352
3142  29.00 19.19074 255.4064 277.6928 302.2835
3143  29.50 19.34958 259.0372 282.0411 306.1916
3144  30.00 19.50258 261.3930 286.2541 310.6169
3145  30.50 19.64975 265.7932 290.1451 313.8741
3146  31.00 19.79109 270.7801 294.1356 318.1423
3147  31.50 19.92659 271.9663 297.2778 321.6288
3148  32.00 20.05627 276.5067 300.8024 326.2354
3149  32.50 20.18011 280.3592 304.3739 329.3103
3150  33.00 20.29812 282.9486 306.7266 332.0393
3151  33.50 20.41030 284.5635 311.3203 335.7465
3152  34.00 20.51665 288.3029 313.2635 339.3042
3153  34.50 20.61716 290.9514 315.5782 340.0704
```

```
3154  35.00 20.71185 292.6941 318.7601 344.7290
3155  35.50 20.80070 295.2649 321.0198 345.6538
3156  36.00 20.88372 297.4315 322.7989 349.1005
3157  36.50 20.96090 298.9932 325.2187 350.5952
3158  37.00 21.03226 301.4454 327.3860 352.2909
3159  37.50 21.09778 301.6702 328.6336 352.9046
3160  38.00 21.15747 303.7928 329.8113 355.5103
3161  38.50 21.21133 305.9601 331.8551 357.6060
3162  39.00 21.25936 307.3541 333.0496 359.5756
3163  39.50 21.30156 308.5406 334.0408 360.2825
3164  40.00 21.33792 309.4474 335.0559 360.6351
3165  40.50 21.36845 309.3527 335.7833 361.7651
3166  41.00 21.39315 309.2070 336.4953 362.3413
```

15. Now we are in a position to compare the prediction obtained by JAGS and by the naive approach by overplotting the two sets of predictions from the JAGS approach on the plot of the simple ones based on the REML-estimates:

```
> matplot( ga.pt, pr.hc, type="l", lty=1, col="black", lwd=c(2,1,1,2,2) )
> matlines( fetal.x[3100+1:33,"ga"],
+           fetal.qnt[paste("pr[",3100+1:33,"]",sep=""),c(1,3,5)],
+           col="red", lty=1, lwd=2 )
> matlines( fetal.x[3100+33+1:33,"ga"],
+           fetal.qnt[paste("pr[",3100+33+1:33,"]",sep=""),c(1,3,5)],
+           col="blue", lty=1, lwd=2 )
```

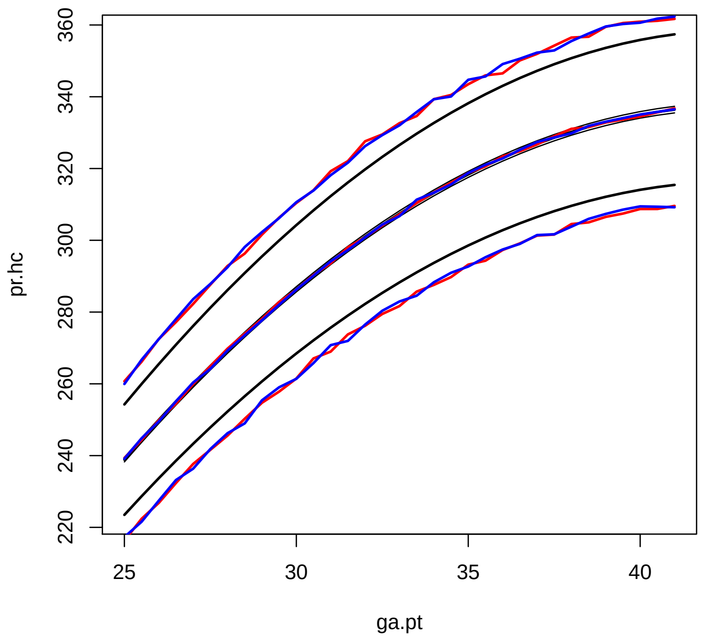


Figure 3.25: *Predicted mean head circumference as function of gestational age. The prediction is a linear function of a quadratic function of gestational age. The outer limits are 95% prediction limits, based on the mixed model. The red curves are the predictions based on a single fetus followed from week 25 through 42, the blue is cross-sectional predictions for the population at each time point.*

16. In order to compare the posteriors of the conditional and marginal prediction, we simply superpose the two densities if the posteriors:

```
> fetal.post <- as.data.frame( as.matrix( fetal.xres ) )
> names( fetal.post ) <- gsub( "\\[", ".", names(fetal.post) )
> names( fetal.post ) <- gsub( ",", ".", names(fetal.post) )
> names( fetal.post ) <- gsub( "\\]", "", names(fetal.post) )
> str( fetal.post )
```

```
'data.frame':        2000 obs. of  76 variables:
 $ Sigma.u.1.1: num  57.4 56.7 49.4 57.2 52.4 ...
 $ Sigma.u.2.1: num  5.61 5.73 5.63 5.02 5.46 ...
 $ Sigma.u.1.2: num  5.61 5.73 5.63 5.02 5.46 ...
 $ Sigma.u.2.2: num  1.32 1.53 1.44 1.02 1.2 ...
 $ beta.1     : num  247 246 247 246 246 ...
 $ beta.2     : num  26.3 26.3 26.4 26.5 26.5 ...
 $ pr.3098    : num  149 138 128 142 154 ...
 $ pr.3099    : num  305 318 314 341 327 ...
 $ pr.3100    : num  320 289 324 335 322 ...
 $ pr.3101    : num  252 226 235 219 233 ...
 $ pr.3102    : num  258 214 235 232 238 ...
 $ pr.3103    : num  284 224 235 252 230 ...
 $ pr.3104    : num  288 251 259 248 259 ...
 $ pr.3105    : num  276 248 246 261 245 ...
 $ pr.3106    : num  287 257 251 260 246 ...
 $ pr.3107    : num  295 262 243 262 272 ...
 $ pr.3108    : num  292 267 268 266 264 ...
 $ pr.3109    : num  300 264 264 253 266 ...
 $ pr.3110    : num  298 277 269 276 256 ...
 $ pr.3111    : num  304 288 264 268 280 ...
 $ pr.3112    : num  319 288 271 296 279 ...
 $ pr.3113    : num  311 285 277 271 263 ...
 $ pr.3114    : num  325 285 294 281 299 ...
 $ pr.3115    : num  320 293 301 295 297 ...
 $ pr.3116    : num  332 289 303 292 273 ...
 $ pr.3117    : num  324 299 284 307 296 ...
 $ pr.3118    : num  339 281 306 293 283 ...
 $ pr.3119    : num  334 304 297 304 294 ...
 $ pr.3120    : num  326 284 303 314 305 ...
 $ pr.3121    : num  333 299 293 312 307 ...
 $ pr.3122    : num  328 289 303 307 308 ...
 $ pr.3123    : num  356 299 310 308 303 ...
 $ pr.3124    : num  350 298 321 313 314 ...
 $ pr.3125    : num  344 317 323 317 307 ...
 $ pr.3126    : num  355 298 310 312 315 ...
 $ pr.3127    : num  356 324 308 305 326 ...
 $ pr.3128    : num  343 301 308 298 320 ...
 $ pr.3129    : num  361 319 306 294 315 ...
 $ pr.3130    : num  368 321 312 306 328 ...
 $ pr.3131    : num  364 311 305 318 319 ...
 $ pr.3132    : num  364 319 320 332 315 ...
 $ pr.3133    : num  352 310 322 316 342 ...
 $ pr.3134    : num  231 247 244 226 233 ...
 $ pr.3135    : num  250 242 244 258 261 ...
 $ pr.3136    : num  243 247 284 245 252 ...
 $ pr.3137    : num  242 273 269 262 263 ...
 $ pr.3138    : num  265 246 248 264 246 ...
 $ pr.3139    : num  287 253 262 237 259 ...
 $ pr.3140    : num  292 286 279 271 278 ...
 $ pr.3141    : num  287 269 263 249 259 ...
 $ pr.3142    : num  272 284 274 273 276 ...
 $ pr.3143    : num  282 295 294 274 264 ...
 $ pr.3144    : num  303 290 283 288 288 ...
 $ pr.3145    : num  302 312 288 270 304 ...
 $ pr.3146    : num  292 285 298 284 263 ...
 $ pr.3147    : num  297 303 299 296 287 ...
 $ pr.3148    : num  319 300 292 309 280 ...
 $ pr.3149    : num  319 283 296 304 286 ...
 $ pr.3150    : num  316 311 288 320 290 ...
 $ pr.3151    : num  309 324 305 307 319 ...
 $ pr.3152    : num  319 307 297 316 323 ...
 $ pr.3153    : num  318 325 291 322 320 ...
 $ pr.3154    : num  300 327 308 308 301 ...
 $ pr.3155    : num  331 311 311 348 330 ...
 $ pr.3156    : num  327 323 323 351 312 ...
 $ pr.3157    : num  321 326 317 322 340 ...
 $ pr.3158    : num  332 335 352 316 342 ...
 $ pr.3159    : num  320 330 348 323 342 ...
 $ pr.3160    : num  360 341 342 334 346 ...
 $ pr.3161    : num  343 322 350 330 343 ...
 $ pr.3162    : num  315 332 322 325 337 ...
 $ pr.3163    : num  335 335 337 320 346 ...
 $ pr.3164    : num  320 344 342 345 327 ...
 $ pr.3165    : num  331 316 344 338 315 ...
 $ pr.3166    : num  353 315 344 354 329 ...
 $ sigma.e    : num  8.57 8.79 8.49 8.6 8.86 ...
```

```
> plot( density( fetal.post$pr.3099 ), lwd=3, col="blue", main="",
+       bty="n", xlab="Head circumference at 38 weeks", xlim=c(250,400) )
> lines( density( fetal.post$pr.3100 ), lwd=3, col="red" )
> rug( quantile( fetal.post$pr.3099, probs=1:3/4 ), lwd=2, col="blue" )
> rug( quantile( fetal.post$pr.3100, probs=1:3/4 ), lwd=2, col="red" )
```

3.7.3.1 Saving it all

For further investigation of the posteriors we save the results:

```
> save( fetal.res, fetal.xres, file="../data/fetal.res" )
```

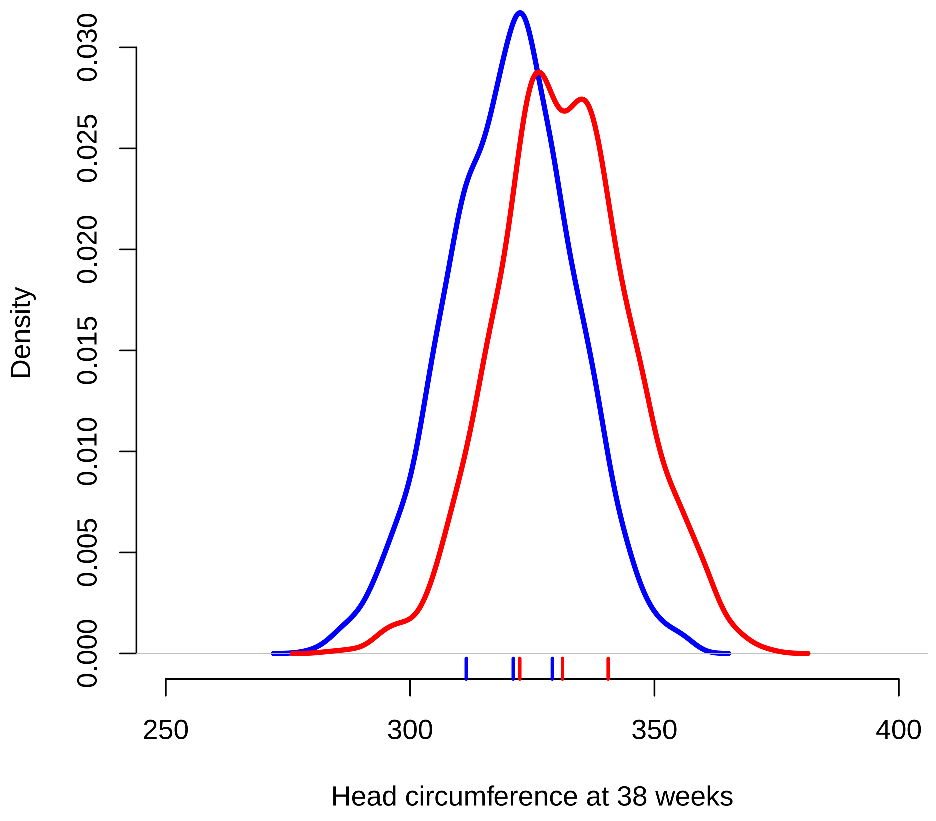


Figure 3.26: *Posterior distribution of the conditional mean (conditional on week 18 measurement being 125) in blue, and the marginal mean of an observation in red.*

3.8 Fetal growth - comparing `lmer`, JAGS and `inla`

We are interested in describing how fetal head circumference varies with (transformed) gestational age, but also in describing how growth of the head circumference varies between fetuses. First we read the data:

```
> # fetal <- read.csv("http://BendixCarstensen.com/Bayes/Cph-2012/data/fetal.csv",header=TRUE)
> fetal <- read.csv("../data/fetal.csv",header=TRUE)
> head( fetal, 11 )
```

```
   id  hc    ga   tga
1   1 211 23.00 16.83
2   1 274 28.43 19.00
3   1 314 33.43 20.39
4   1 330 38.43 21.20
5   2 141 17.71 14.05
6   2 199 22.86 16.76
7   2 266 27.86 18.81
8   2 297 31.29 19.87
9   2 313 34.57 20.63
10  2 321 36.57 20.97
11  3 205 23.43 17.03
```

`ga` is the gestational age and `tga` is a transformation of it which we term t and use as covariate in the following model formulation. The response is head circumference of the fetus, `hc`, which in the model description is termed y.

For ease of model fitting we will center the transformed gestational age at 19.5, corresponding to a gestational age of 30 weeks:

```
> fetal$tga <- fetal$tga - 19.5
```

3.8.1 REML modelling

The relevant model is a *linear mixed model* with a random intercept and a random slope term for the measurement y_{ft} on fetus f at time t:

$$y_{ft} = (\beta_0 + u_{0f}) + (\beta_1 + u_{1f})t + e_{ft}$$
$$(u_{0f}, u_{1f}) \sim \mathcal{N}(0, \Sigma), \quad e_{ft} \sim \mathcal{N}(0, \sigma)$$

We now set up and estimate the parameter of this model using `lmer` from the `lme4` package:

```
> library( lme4 )
> m0 <- lmer( hc ~ tga + (tga|id), data=fetal )
> summary(m0)
```

```
Linear mixed model fit by REML
Formula: hc ~ tga + (tga | id)
   Data: fetal
   AIC   BIC logLik deviance REMLdev
 23340 23376 -11664    23324   23328
Random effects:
 Groups   Name        Variance Std.Dev. Corr
 id       (Intercept) 74.4922  8.6309
          tga          1.2152  1.1024   0.747
 Residual             73.7748  8.5892
Number of obs: 3097, groups: id, 706
```

```
Fixed effects:
             Estimate Std. Error t value
(Intercept) 286.27482    0.37207   769.4
tga          26.48473    0.07786   340.2

Correlation of Fixed Effects:
    (Intr)
tga 0.536
```

Thus we see that the default is to produce a set of correlated random effects, which of course is the only sensible thing to do.

3.8.2 JAGS

We can also set this model up in JAGS the usual way; first we specify data:

```
> fetal.dat <- list( id = as.integer( factor(fetal$id) ),
+                    hc = fetal$hc,
+                   tga = fetal$tga,
+                     N = nrow(fetal),
+                     I = length( unique(fetal$id) ) )
```

In particular we need to specify a variance-covariance matrix for the random effects, which is done by specifying a Wishart prior distribution on the space of variance-covariance matrices, which takes a fixed 2×2-matrix as input for the matrix "mean", which we specify in a `data` section of the JAGS program before defining the `model`:

```
> cat("
+ # Fixing data to be used in model definition
+ data
+    {
+    zero[1] <- 0
+    zero[2] <- 0
+    R[1,1] <- 0.1
+    R[1,2] <- 0
+    R[2,1] <- 0
+    R[2,2] <- 0.5
+    }
+ # Then define model
+ model
+    {
+    # Intercept and slope for each person, including random effects
+      for( i in 1:I )
+      {
+      u[i,1:2] ~ dmnorm(zero,Omega.u)
+      }
+
+    # Define model for each observational unit
+      for( j in 1:N )
+      {
+      mu[j] <- ( beta[1] + u[id[j],1] ) +
+               ( beta[2] + u[id[j],2] ) * ( tga[j] )
+      hc[j] ~ dnorm( mu[j], tau.e )
+      }
+
+    #------------------------------------------------------------
+    # Priors:
+
+    # Fixed intercept and slope
+      beta[1] ~ dnorm(0.0,1.0E-5)
+      beta[2] ~ dnorm(0.0,1.0E-5)
+
+    # Residual variance
+      tau.e <- pow(sigma.e,-2)
+    sigma.e   ~ dunif(0,100)
+
+    # Define prior for the variance-covariance matrix of the random effects
+      Sigma.u <- inverse(Omega.u)
+      Omega.u   ~ dwish( R, 2 )
+    }",
+      file="fetal.jag" )
```

Then we extract the relevant variances/SDs from the `lmer` object and use these as starting values for the MCMC chains:

```
> ( sigma.e <- attr(VarCorr(m0),"sc") )

[1] 8.589226

> ( Omega.u <- solve( VarCorr(m0)$id ) )

            (Intercept)        tga
(Intercept)  0.03036677 -0.1775887
tga         -0.17758872  1.8614539

> ( beta    <- fixef( m0 ) )

(Intercept)         tga
  286.27482    26.48473

> fetal.ini <- list( list( sigma.e = sigma.e/3,
+                          Omega.u = Omega.u/3,
+                             beta = beta   /3 ),
+                    list( sigma.e = sigma.e*3,
+                          Omega.u = Omega.u*3,
+                             beta = beta   *3 ),
+                    list( sigma.e = sigma.e/3,
+                          Omega.u = Omega.u*3,
+                             beta = beta   /3 ),
+                    list( sigma.e = sigma.e*3,
+                          Omega.u = Omega.u/3,
+                             beta = beta   *3 ) )
```

Finally, we can get the whole thing going:

```
> library( rjags )
> system.time(
+ fetal.mod <- jags.model( file = "fetal.jag",
+                          data = fetal.dat,
+                      n.chains = 4,
+                         inits = fetal.ini,
+                       n.adapt = 2500 )
+            )
```

```
Compiling data graph
   Resolving undeclared variables
   Allocating nodes
   Initializing
   Reading data back into data table
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 19036

Initializing model

   user  system elapsed
 105.63    0.02  114.59
```

```
> system.time(
+ fetal.res <- coda.samples( fetal.mod,
+                                  var = c("beta","sigma.e","Sigma.u"),
+                               n.iter = 1000,
+                                 thin = 4 )
+             )
```

```
   user  system elapsed
  45.07    0.00   45.57
```

```
> str( fetal.res )
```

```
List of 4
 $ : mcmc [1:250, 1:7] 68.2 68.7 79.9 72.9 74 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:7] "Sigma.u[1,1]" "Sigma.u[2,1]" "Sigma.u[1,2]" "Sigma.u[2,2]" ...
  ..- attr(*, "mcpar")= num [1:3] 2504 3500 4
 $ : mcmc [1:250, 1:7] 69.5 72.9 71.8 75.4 72.4 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:7] "Sigma.u[1,1]" "Sigma.u[2,1]" "Sigma.u[1,2]" "Sigma.u[2,2]" ...
  ..- attr(*, "mcpar")= num [1:3] 2504 3500 4
 $ : mcmc [1:250, 1:7] 69.4 69.3 70.7 69.2 68.4 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:7] "Sigma.u[1,1]" "Sigma.u[2,1]" "Sigma.u[1,2]" "Sigma.u[2,2]" ...
  ..- attr(*, "mcpar")= num [1:3] 2504 3500 4
 $ : mcmc [1:250, 1:7] 42056 38758 37423 36396 36117 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:7] "Sigma.u[1,1]" "Sigma.u[2,1]" "Sigma.u[1,2]" "Sigma.u[2,2]" ...
  ..- attr(*, "mcpar")= num [1:3] 2504 3500 4
 - attr(*, "class")= chr "mcmc.list"
```

```
> summary( fetal.res )
```

```
Iterations = 2504:3500
Thinning interval = 4
Number of chains = 4
Sample size per chain = 250

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:
```

```
                 Mean        SD  Naive SE Time-series SE
Sigma.u[1,1] 6505.601 1.191e+04 3.767e+02      3.271e+02
Sigma.u[2,1]  781.192 1.418e+03 4.485e+01      3.578e+01
Sigma.u[1,2]  781.192 1.418e+03 4.485e+01      3.578e+01
Sigma.u[2,2]   94.415 1.692e+02 5.352e+00      3.876e+00
beta[1]       325.774 6.976e+01 2.206e+00      1.072e+00
beta[2]        31.263 8.380e+00 2.650e-01      1.044e-01
sigma.e         8.632 1.422e-01 4.497e-03      6.171e-03

2. Quantiles for each variable:

                 2.5%     25%     50%      75%     97.5%
Sigma.u[1,1]  64.9968  71.590  76.231 2673.945 36466.557
Sigma.u[2,1]   5.3074   6.569   7.225  344.883  4264.276
Sigma.u[1,2]   5.3074   6.569   7.225  344.883  4264.276
Sigma.u[2,2]   0.6862   0.972   1.158   45.256   503.087
beta[1]      285.5846 286.145 286.458  313.234   477.941
beta[2]       26.3495  26.451  26.522   30.086    49.068
sigma.e        8.3662   8.531   8.632    8.733     8.914
```

In order to have convenient access to the posterior samples we collect them conveniently in a matrix:

```
> fetal.mat <- as.matrix( fetal.res )
> colnames( fetal.mat )
```

```
[1] "Sigma.u[1,1]" "Sigma.u[2,1]" "Sigma.u[1,2]" "Sigma.u[2,2]" "beta[1]"
[6] "beta[2]"      "sigma.e"
```

3.8.3 INLA

In order for INLA to work properly with factors these must (as for JAGS) assume consecutive numbers from 1 and upwards. This is in R accomplished with the construction `as.integer(factor())`:

```
> fetal <- transform(fetal, tgac=tga,
+                              id=as.integer(factor(id)) )
```

The INLA version of the model with uncorrelated random intercept and slope (which is, incidentally, a daft model) looks like this:

```
> library( INLA )
> system.time(
+ im1 <- inla( hc ~ tga + f(id) + f(tgac), data=fetal )
+            )
```

```
   user  system elapsed
   0.85    0.39   14.63
```

```
> summary( im1 )
```

```
Call:
"inla(formula = hc ~ tga + f(id) + f(tgac), data = fetal)"

Time used:
 Pre-processing    Running inla Post-processing           Total
      0.8744171      12.8195870       0.7182710      14.4122751

Fixed effects:
                 mean        sd 0.025quant  0.5quant 0.975quant kld
(Intercept) 286.86237 0.5502102  285.78513 286.86151  287.94467   0
tga          26.53119 0.1685165   26.19978  26.53141   26.86142   0

Random effects:
Name          Model                  Max KLD
id   IID model
tgac   IID model

Model hyperparameters:
                                        mean      sd        0.025quant
Precision for the Gaussian observations 0.0167998 0.0003828 0.0159817
Precision for id                        0.0156304 0.0009128 0.0140332
Precision for tgac                      0.0525634 0.0094877 0.0368886
                                        0.5quant  0.975quant
Precision for the Gaussian observations 0.0168312 0.0174707
Precision for id                        0.0155545 0.0176060
Precision for tgac                      0.0515342 0.0740095

Expected number of effective parameters(std dev): 677.14(5.774)
Number of equivalent replicates : 4.574

Marginal Likelihood:  -11564.41
Warning: Interpret the marginal likelihood with care if the prior model is improper.
```

However, we want the model with correlated random effects, so we need the "replication" trick, where we assigning an additional vector of random effects for the `Nb` fetuses. It is actually this trick that requires the `id`s to be consecutive integers from 1:

```
> Nb <- max(fetal$id)
> fetal$xid <- fetal$id + Nb
> system.time(
+ im2 <- inla( hc ~ 1 + tga +
+                   f(  id, model = "iid2d", n=2*Nb ) +
+                   f( xid, tga, copy="id" ),
+              data = fetal )
+            )


   user  system elapsed
   1.10    0.47   26.47



> summary( im2 )


Call:
c("inla(formula = hc ~ 1 + tga + f(id, model = \"iid2d\", n = 2 * ",  "    Nb) + f(xid, tga, copy =

Time used:
 Pre-processing    Running inla Post-processing           Total
       0.624584       22.079045        2.966774       25.670403

Fixed effects:
                 mean         sd 0.025quant  0.5quant 0.975quant kld
(Intercept) 286.29050 0.37156200  285.56141 286.29062  287.01907   0
```

```
tga          26.48216 0.07735731   26.33047  26.48215   26.63396   0

Random effects:
Name          Model                  Max KLD
id   IID2D model
xid   Copy

Model hyperparameters:
                                        mean      sd        0.025quant
Precision for the Gaussian observations 0.0133703 0.0007231 0.0123121
Precision for id (component 1)          0.0147195 0.0009279 0.0128233
Precision for id (component 2)          0.9978198 0.2229477 0.6008056
Rho1:2 for id                           0.7818731 0.0650576 0.6307188
                                        0.5quant  0.975quant
Precision for the Gaussian observations 0.0132479 0.0150640
Precision for id (component 1)          0.0147608 0.0164454
Precision for id (component 2)          0.9872997 1.4681026
Rho1:2 for id                           0.7904603 0.8841909

Expected number of effective parameters(std dev): 625.73(26.73)
Number of equivalent replicates : 4.949

Marginal Likelihood:  -25030.47
Warning: Interpret the marginal likelihood with care if the prior model is improper.
```

```
> names( im2 )
```

```
 [1] "names.fixed"                "summary.fixed"
 [3] "marginals.fixed"            "summary.lincomb"
 [5] "marginals.lincomb"          "size.lincomb"
 [7] "summary.lincomb.derived"    "marginals.lincomb.derived"
 [9] "size.lincomb.derived"       "mlik"
[11] "cpo"                        "model.random"
[13] "summary.random"             "marginals.random"
[15] "size.random"                "summary.linear.predictor"
[17] "marginals.linear.predictor" "summary.fitted.values"
[19] "marginals.fitted.values"    "size.linear.predictor"
[21] "summary.hyperpar"           "marginals.hyperpar"
[23] "internal.summary.hyperpar"  "internal.marginals.hyperpar"
[25] "si"                         "total.offset"
[27] "model.spde2.blc"            "summary.spde2.blc"
[29] "marginals.spde2.blc"        "size.spde2.blc"
[31] "misc"                       "dic"
[33] "mode"                       "neffp"
[35] "joint.hyper"                "nhyper"
[37] "version"                    "Q"
[39] "graph"                      "cpu.used"
[41] "control.compute"            "control.predictor"
[43] "control.lincomb"            "control.data"
[45] "control.inla"               "control.results"
[47] "control.fixed"              "control.mode"
[49] "control.expert"             "call"
[51] "family"                     "data"
[53] "formula"                    "inla.call"
[55] "silent"                     "model.matrix"
[57] ".control.defaults"          ".internal"
```

The quantities with which to compare the output from INLA and JAGS are the precisions from the REML fit from the `lmer` object above. However it is better to make comparisons on a relevant scale, namely the standard deviation scale.

3.8.4 Comparing `lmer`, JAGS and INLA

This means that we would like to see the posterior density of the *inverse* of the precision.

We can now summarize the results from the three analyses; first for the fixed effects:

```
> ### LMER ###
> fixef(m0)


(Intercept)         tga
  286.27482    26.48473


> ### JAGS ###
> summary(fetal.res)$quantiles[c("beta[1]","beta[2]"),]


             2.5%       25%       50%       75%     97.5%
beta[1] 285.58464 286.14532 286.45850 313.23351 477.94141
beta[2]  26.34947  26.45093  26.52231  30.08648  49.06786


> ### INLA ###
> im2$summary.fixed


                 mean         sd 0.025quant  0.5quant 0.975quant kld
(Intercept) 286.29050 0.37156200  285.56141 286.29062  287.01907   0
tga          26.48216 0.07735731   26.33047  26.48215   26.63396   0
```

Then for the random effcts variances

```
> ### LMER ###
> diag(VarCorr(m0)$id)


(Intercept)         tga
  74.492153    1.215225


> ### JAGS ###
> summary(fetal.res)$quantiles[c("Sigma.u[1,1]","Sigma.u[2,2]"),"50%",drop=FALSE]


                   50%
Sigma.u[1,1] 76.231233
Sigma.u[2,2]  1.158171


> ### INLA ###
> 1/im2$summary.hyperpar[2:3,c("mean","0.5quant")]


                                    mean  0.5quant
Precision for id (component 1) 67.937253 67.747191
Precision for id (component 2)  1.002185  1.012864
```

and finally for the residual standard deviations:

```
> ### LMER ###
> attr( VarCorr(m0), "sc" )
```

```
[1] 8.589226

> ### JAGS ###
> summary(fetal.res)$quantiles["sigma.e","50%",drop=FALSE]

              50%
sigma.e 8.631956

> ### INLA ###
> 1/sqrt(im2$summary.hyperpar[1,c("mean","0.5quant"),drop=FALSE])

                                            mean 0.5quant
Precision for the Gaussian observations 8.648273 8.688147
```

and finally for the correlation between the slope and the intercept (for whatever that is worth):

```
> ### LMER ###
> attr(VarCorr(m0)$id,"correlation")[1,2]

[1] 0.7469472

> ### JAGS ###
>       summary(fetal.res)$quantiles["Sigma.u[1,2]",3]/
+  sqrt(summary(fetal.res)$quantiles["Sigma.u[1,1]",3]*
+       summary(fetal.res)$quantiles["Sigma.u[2,2]",3])

[1] 0.7688772

> ### INLA ###
> im2$summary.hyperpar[4,1,drop=FALSE]

                     mean
Rho1:2 for id 0.7818731
```

3.8.5 Posterior samples from INLA

Even though only the *marginal* densities are avilable as output form `INLA`, it is occasionally useful to have access to a sample from the posterior distribution. This is achieved by the `inla.rmarginal` function, which takes the numer of random draws and a marginal object from an `INLA` fit as arguments.

So as an example we generate a sample of 10,000 from the posterior of the precison of the random slopes and make a histogram of this, and also of the corresponding sd, as shown in figure **??**

```
> sd.mx <- max( fetal.mat[,"Sigma.u[2,2]"]^0.5 ) + 0.2
> par( mfrow=c(3,1), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> hist( prc.beta <- inla.rmarginal(10000,im2$marginals.hyperpar[[3]]),
+       col="gray", freq=F, breaks=100, main="",
+       xlab="Precison of random slope (INLA)" )
> lines(im2$marginals.hyperpar[[3]],lwd=2,col="red")
> hist( prc.beta^-2,
+       col="gray", freq=F, breaks=seq(0,sd.mx,0.2), main="",
+       xlab="SD of random slope (INLA)" )
> hist( fetal.mat[,"Sigma.u[2,2]"]^0.5,
+       col="gray", freq=F, breaks=seq(0,sd.mx,0.2), main="",
+       xlab="SD of random slope (JAGS)" )
```

.

From figure 3.27 it is seen that the posterior of the random slope is (to put it mildly) not well-determined, and that the assumption about a posterior unimodal istribution (which is underlying both models may not be correct).

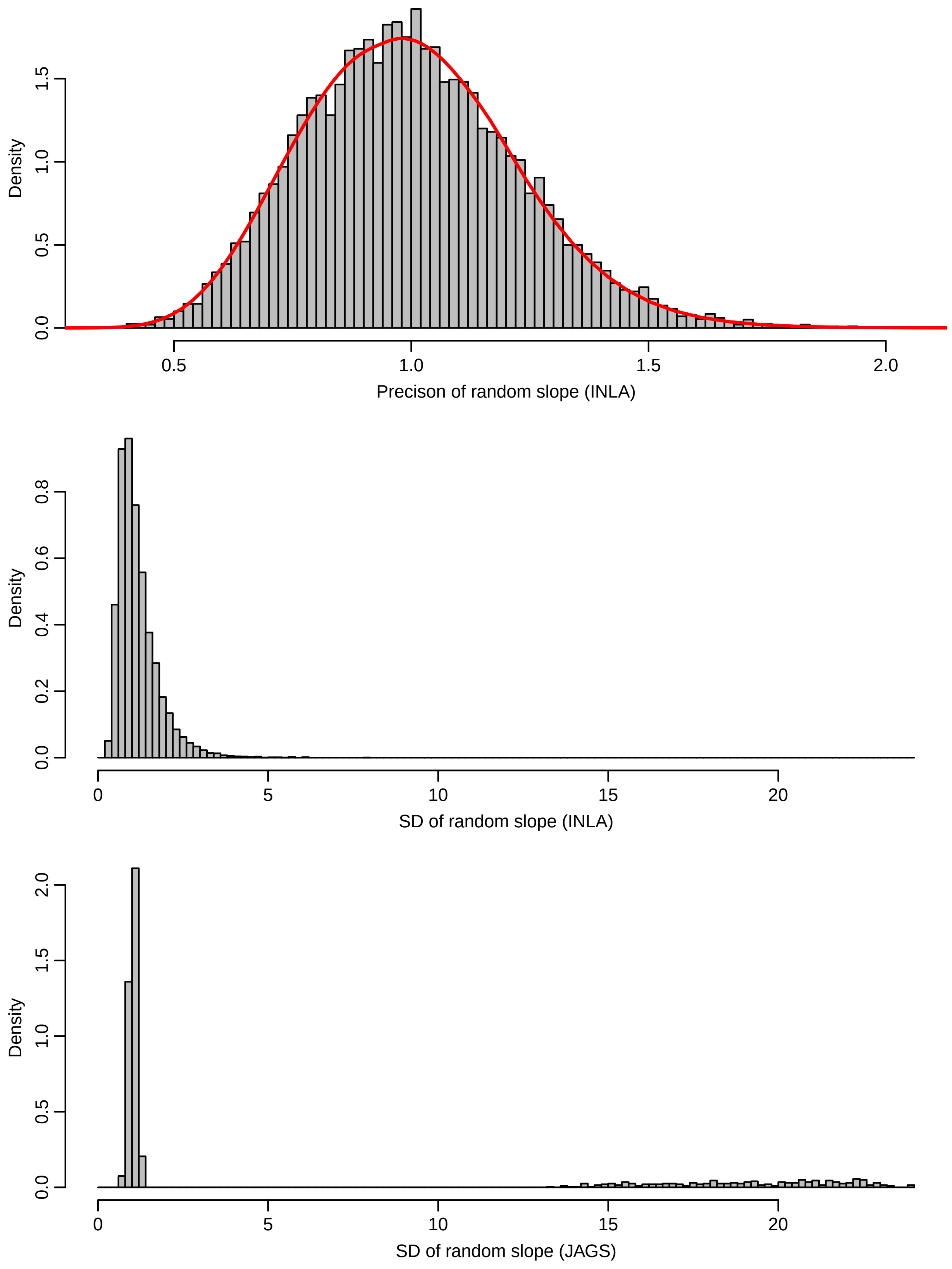


Figure 3.27: *Marginal posterior distributions of the precision and sd of the random slope*

3.9 Generalized linear mixed model in JAGS

```
> library( rjags )
> ###########################################################################
>
> ################## Question 5 ##############################################
>
> # JAGS code for GLMM warts and PID data exercise 10
>
> kitdata  <- read.csv( "../data/wartpid.csv" , header = T)
> wartpid <- kitdata[,c("consults","warts")]
> # wartpid <- wartpid[wartpid$PID/wartpid$consults < 0.025,]
>
> ############# Model 1 ##################################################
>
> # warts
> # doctor-specific diagnosis frequency fixed effects
>
> # The JAGS-code for the model
>
> cat( "model
+     {
+     a[1] ~ dnorm(0,10000)
+     for(i in 2:23)
+     {
+     a[i] ~ dnorm(0,0.0001)
+     }
+     for(i in 1:23)
+     {
+     mean[i] <- mu + a[i]
+     logit(pr[i]) <-  mean[i]
+     warts[i] ~ dbin(pr[i],consults[i])
+     }
+
+     mu ~ dnorm(0,0.001);
+
+     }",file="m5.jag" )
> # Data as a list
>
> e.dat <- as.list( wartpid )
> # Inits as a list
>
> e.ini <- list(mu = -3, a = c(0,rep(0,22)))
> # Names of the parameters to monitor
>
> e.par <-c("mean")
> # Model compilation and burn-in
>
> e.mod <- jags.model( file = "m5.jag",
+                      data = e.dat,
+                      n.chains = 3,
+                      inits = e.ini,
+                      n.adapt = 1500 )
> # Sampling from the posterior
>
> e.res <- coda.samples( model = e.mod,
+                        var = e.par,
+                        n.iter = 1500,
+                        thin = 1 )
> # Take a look at the output from JAGS
>
> summary(e.res)
> fixeffs <- cbind(as.vector(summary(e.res)$statistics[,1]),as.vector(summary(e.res)$quantiles[,1]),
+             as.vector(summary(e.res)$quantiles[,5]))
> expit <- function(x) {exp(x)/(1 + exp(x))}
> plot(x = expit(fixeffs[,1]),y = 5*(1:23)-2, xlim = c(0,0.125), pch = 16, ylab = "Doctor", xlab = '
> segments(x0 = expit(fixeffs[,2]), y0 = 5*(1:23)-2, x1 = expit(fixeffs[,3]), y1 = 5*(1:23)-2, lty =
> ############# Model 2 ###################################################
>
> # warts
> # single common random effect (sigma2a) per doctor
>
> cat( "model
+     {
+     for(i in 1:23)
+     {
+     a[i] ~ dnorm(0,taua);
+     mean[i] <- mu + a[i];
+     logit(pr[i]) <-  mean[i];
+     warts[i] ~ dbin(pr[i],consults[i]);
+     }
+
+     taua ~ dpar(1,0.01);
+
+     sigma2a <- 1/taua;
+     sigmaa <- sqrt(sigma2a);
+
+     mu ~ dnorm(0,0.001);
+
+     }",file="m6.jag" )
> # Data as a list
>
> f.dat <- as.list( wartpid )
> # Inits as a list
>
> f.ini <- list(mu = -3, taua = 1)
> # Names of the parameters to monitor
>
> f.par <-c("mean","mu","sigma2a","sigmaa","a")
> # Model compilation and burn-in
>
> f.mod <- jags.model( file = "m6.jag",
+                      data = f.dat,
+                      n.chains = 3,
+                      inits = f.ini,
+                      n.adapt = 1500 )
> # Sampling from the posterior
>
> f.res <- coda.samples( model = f.mod,
+                        var = f.par,
+                        n.iter = 1500,
+                        thin = 1 )
> # Take a look at the output from JAGS
>
> summary(f.res)
> raneffs <- cbind(as.vector(summary(f.res)$statistics[(1:23)+23,1]),as.vector(summary(f.res)$quanti
+                           as.vector(summary(f.res)$quantiles[(1:23)+23,5]))
> plot(x = expit(fixeffs[,1]),y = 5*(1:23)-2, xlim = c(0,0.125), pch = 16, ylab = "Doctor", xlab = '
> segments(x0 = expit(fixeffs[,2]), y0 = 5*(1:23)-2, x1 = expit(fixeffs[,3]), y1 = 5*(1:23)-2, lty =
> points(x = expit(raneffs[,1]),y = 5*(1:23)-4, pch = 15)
> segments(x0 = expit(raneffs[,2]), y0 = 5*(1:23)-4, x1 = expit(raneffs[,3]), y1 = 5*(1:23)-4, lty =
> abline(v = expit(summary(f.res)$statistics["mu",1]), col = "red", lty = 2, lwd = 2)
> legend(x = 0.09, y = 100,legend = c("Fixed","Random","Mean"), lty = c(1,2,2), pch = c(16,15,NA), l
> ############# Model 3 ###################################################
>
> # PID = pelvic inflammatory disease
> # doctor-specific diagnosis rate fixed effects
>
> wartpid <- kitdata[,c("consults","PID")]
> # The JAGS-code for the model
>
> cat( "model
+     {
+     a[1] ~ dnorm(0,10000)
+     for(i in 2:23)
+     {
+     a[i] ~ dnorm(0,0.0001)
+     }
+     for(i in 1:23)
+     {
+     mean[i] <- mu + a[i]
+     logit(pr[i]) <-  mean[i]
+     PID[i] ~ dbin(pr[i],consults[i])
+     }
+
+     mu ~ dnorm(0,0.001);
+
+     }",file="m7.jag" )
> # Data as a list
>
> g.dat <- as.list( wartpid )
> # Inits as a list
>
> g.ini <- list(mu = -4, a = c(0,rep(0,22)))
> # Names of the parameters to monitor
>
> g.par <-c("mean")
> # Model compilation and burn-in
>
> g.mod <- jags.model( file = "m7.jag",
+                      data = g.dat,
+                      n.chains = 3,
+                      inits = g.ini,
+                      n.adapt = 1500 )
> # Sampling from the posterior
>
> g.res <- coda.samples( model = g.mod,
+                        var = g.par,
+                        n.iter = 1500,
+                        thin = 1 )
> # Take a look at the output from JAGS
>
> summary(g.res)
> fixeffs2 <- cbind(as.vector(summary(g.res)$statistics[,1]),as.vector(summary(g.res)$quantiles[,1])
+                  as.vector(summary(g.res)$quantiles[,5]))
> plot(x = expit(fixeffs2[,1]),y = 5*(1:23)-2, xlim = c(0,0.125), pch = 16, ylab = "Doctor", xlab =
> segments(x0 = expit(fixeffs2[,2]), y0 = 5*(1:23)-2, x1 = expit(fixeffs2[,3]), y1 = 5*(1:23)-2, lty
> ############# Model 4 ###################################################
>
> # PID
> # single common random effect (sigma2a) per doctor
>
> cat( "model
+     {
+     for(i in 1:23)
+     {
+     a[i] ~ dnorm(0,taua);
+     mean[i] <- mu + a[i];
+     logit(pr[i]) <-  mean[i];
+     PID[i] ~ dbin(pr[i],consults[i]);
+     }
+
+     taua ~ dpar(1,0.01);
+
+     sigma2a <- 1/taua;
+     sigmaa <- sqrt(sigma2a);
+
+     mu ~ dnorm(0,0.001);
```

```
+
+     }",file="m8.jag" )
> # Data as a list
>
> h.dat <- as.list( wartpid )
> # Inits as a list
>
> h.ini <- list(mu = -4, taua = 1)
> # Names of the parameters to monitor
>
> h.par <-c("mean","mu","sigma2a","sigmaa","a")
> # Model compilation and burn-in
>
> h.mod <- jags.model( file = "m8.jag",
+                      data = h.dat,
+                      n.chains = 3,
+                      inits = h.ini,
+                      n.adapt = 1500 )
> # Sampling from the posterior
>
> h.res <- coda.samples( model = h.mod,
+                        var = h.par,
+                        n.iter = 1500,
+                        thin = 1 )
> # Take a look at the output from JAGS
>
> summary(h.res)
> raneffs2 <- cbind(as.vector(summary(h.res)$statistics[(1:23)+23,1]),as.vector(summary(h.res)$quant
+                  as.vector(summary(h.res)$quantiles[(1:23)+23,5]))
> plot(x = expit(fixeffs2[,1]),y = 5*(1:23)-2, xlim = c(0,0.125), pch = 16, ylab = "Doctor", xlab =
> segments(x0 = expit(fixeffs2[,2]), y0 = 5*(1:23)-2, x1 = expit(fixeffs2[,3]), y1 = 5*(1:23)-2, lty
> points(x = expit(raneffs2[,1]),y = 5*(1:23)-4, pch = 15)
> segments(x0 = expit(raneffs2[,2]), y0 = 5*(1:23)-4, x1 = expit(raneffs2[,3]), y1 = 5*(1:23)-4, lty
> abline(v = expit(summary(h.res)$statistics["mu",1]), col = "red", lty = 2, lwd = 2)
> legend(x = 0.09, y = 100,legend = c("Fixed","Random","Mean"), lty = c(1,2,2), pch = c(16,15,NA), l
> # Density plots
>
> plot(density(as.matrix(f.res)[,"sigmaa"]), col = "blue", lwd = 2, xlim = c(0,3), main = "Density p
> segments(x0 = quantile(as.matrix(f.res)[,"sigmaa"], prob = 0.50), y0 = 0,
+          x1 = quantile(as.matrix(f.res)[,"sigmaa"], prob = 0.50), y1 = 5, col = "blue", lty = 2, l
> segments(x0 = quantile(as.matrix(f.res)[,"sigmaa"], prob = c(0.05,0.95)), y0 = c(0,0),
+          x1 = quantile(as.matrix(f.res)[,"sigmaa"], prob = c(0.05,0.95)), y1 = c(1.0,0.8), col = '
> lines(density(as.matrix(h.res)[,"sigmaa"]), col = "red", lwd = 2, lty = 1)
> segments(x0 = quantile(as.matrix(h.res)[,"sigmaa"], prob = 0.50), y0 = 0,
+          x1 = quantile(as.matrix(h.res)[,"sigmaa"], prob = 0.50), y1 = 1.5, col = "red", lty = 2,
> segments(x0 = quantile(as.matrix(h.res)[,"sigmaa"], prob = c(0.05,0.95)), y0 = c(0,0),
+          x1 = quantile(as.matrix(fh.res)[,"sigmaa"], prob = c(0.05,0.95)), y1 = c(0.65,0.2), col =
> legend(x = 2, y = 5, legend = c("warts","PID"), lty = c(1,1), lwd = c(2,2), col = c("blue","red"))
>
>
```

3.10 Classical twin model in JAGS

This program in covariance for MD

```
> library( rjags )
> # JAGS code for paired and twin data exercise 10
> mgram <- read.csv("../data/mgram.csv", header = TRUE)
> ######################## Question 1 #################################
>
> # The JAGS-code for the model
>
> cat( "model
+       {
+       for (i in 1:951)
+       {
+       pdens1[i] ~ dnorm(a[i],tau.e)
+       pdens2[i] ~ dnorm(a[i],tau.e)
+       a[i] ~ dnorm(mu,tau.a)
+       }
+
+       tau.a <- pow(sigma.a,-2)
+       sigma.a ~ dunif(0,1000)
+
+       tau.e <- pow(sigma.e,-2)
+       sigma.e ~ dunif(0,1000)
+
+       mu ~ dnorm(0,1.0E-6)
+
+       sigma2.a <- pow(sigma.a,2)
+       sigma2.e <- pow(sigma.e,2)
+
+       }",
+ file = "m1.jag" )
> # Inits as a list of parameter names
>
> a.ini <- list(mu = 37, sigma.a = 16, sigma.e = 13.5)
> # Data as a list
>
> a.dat <- as.list( mgram )
> # Names of the parameters to monitor
>
> a.par <- c("mu","sigma2.a","sigma2.e","sigma.a","sigma.e")
> # Model compilation and burn-in
> a.mod <- jags.model( file = "m1.jag",
+                      data = a.dat,
+                      n.chains = 3,
+                      inits = a.ini,
+                      n.adapt = 1500 )
> # Sampling from the posterior
> a.res <- coda.samples( model = a.mod,
+                        var = a.par,
+                        n.iter = 1500,
+                        thin = 1 )
> # Take a look at the output from JAGS
>
> summary(a.res)
> ######################## Question 2 #################################
>
> # The JAGS-code for the model
>
> cat(" model
+       {
+       for (i in 1:951)
+       {
+       pdens1[i] ~ dnorm(mean.pdens1[i],tau.e)
+       pdens2[i] ~ dnorm(mean.pdens2[i],tau.e)
+       mean.pdens1[i] <- b.int + sqrt(rho)*a1[i] + sqrt(1-rho)*a2[i]
+       mean.pdens2[i] <- b.int + sqrt(rho)*a1[i] + mz[i]*sqrt(1-rho)*a2[i] + dz[i]*sqrt(1-rho)*a3[i
+       a1[i] ~ dnorm(0,tau.a)
+       a2[i] ~ dnorm(0,tau.a)
+       a3[i] ~ dnorm(0,tau.a)
+       }
+
+       rho ~ dunif(0,1)
+
+       b.int ~ dnorm(0,0.0001)
+
+       tau.a <- pow(sigma.a,-2)
+       sigma.a ~ dunif(0,1000)
+
+       tau.e <- pow(sigma.e,-2)
+       sigma.e ~ dunif(0,1000)
+
+       sigma2.a <- pow(sigma.a,2)
+       sigma2.e <- pow(sigma.e,2)
+
+       }",
+ file = "m2.jag")
> # Inits as a list of parameter names
>
> b.ini <- list(rho = 0.5, b.int = 37, sigma.a = 16, sigma.e = 13.5)
> # Names of the parameters to monitor
>
> b.par <- c("b.int","rho","sigma2.a","sigma2.e","sigma.a","sigma.e")
> # Model compilation and burn-in
>
> a.mod <- jags.model( file = "m2.jag",
+                      data = a.dat,
+                      n.chains = 3,
+                      inits = b.ini,
+                      n.adapt = 1500 )
> # Sampling from the posterior
>
> b.res <- coda.samples( model = a.mod,
+                        var = b.par,
+                        n.iter = 1500,
+                        thin = 1 )
> # Take a look at the output from JAGS
>
> summary(b.res)
> ######################## Question 3 #################################
>
> # The JAGS-code for the model
>
> cat(" model
+       {
+       for (i in 1:951)
+       {
+       pdens1[i] ~ dnorm(mean.pdens1[i],tau.e)
+       pdens2[i] ~ dnorm(mean.pdens2[i],tau.e)
+       mean.pdens1[i] <- b.int + b.age*agemgram1[i] + sqrt(rho)*a1[i]
+                                                    + sqrt(1-rho)*a2[i]
+       mean.pdens2[i] <- b.int + b.age*agemgram2[i] + sqrt(rho)*a1[i]
+                                                    + mz[i]*sqrt(1-rho)*a2[i]
+                                                    + dz[i]*sqrt(1-rho)*a3[i]
+       a1[i] ~ dnorm(0,tau.a)
+       a2[i] ~ dnorm(0,tau.a)
+       a3[i] ~ dnorm(0,tau.a)
+       }
+
+       rho ~ dunif(0,1)
+
+       b.int ~ dnorm(0,0.0001)
+       b.age ~ dnorm(0,0.0001)
+       tau.a <- pow(sigma.a,-2)
+       sigma.a ~ dunif(0,1000)
+
+       tau.e <- pow(sigma.e,-2)
+       sigma.e ~ dunif(0,1000)
+
+       sigma2.a <- pow(sigma.a,2)
+       sigma2.e <- pow(sigma.e,2)
+       }",
+ file = "m3.jag")
> # Inits as a list
>
> c.ini <- list(rho = 0.5, b.int = 37, b.age = -0.75, sigma.a = 16, sigma.e = 13.5)
> # Names of the parameters to monitor
>
> c.par <- c("b.age","b.int","rho","sigma2.a","sigma2.e","sigma.a","sigma.e")
> # Model compilation and burn-in
>
> c.mod <- jags.model( file = "m3.jag",
+                      data = a.dat,
+                  n.chains = 3,
+                     inits = c.ini,
+                   n.adapt = 1500 )
> # Sampling from the posterior
>
> c.res <- coda.samples( model = c.mod,
+                          var = c.par,
+                       n.iter = 1500,
+                         thin = 1 )
> # Take a look at the output from JAGS
>
> summary(c.res)
> ######################## Question 4 #################################
>
> # The JAGS-code for the model
>
> cat(" model
+       {
+       for (i in 1:951)
+       {
+       pdens1[i] ~ dnorm(mean.pdens1[i],tau.e)
+       pdens2[i] ~ dnorm(mean.pdens2[i],tau.e)
+       mean.pdens1[i] <- b.int + b.age*agemgram1[i] + b.wgt*weight1[i] + sqrt(rho)*a1[i] +
+       mean.pdens2[i] <- b.int + b.age*agemgram2[i] + b.wgt*weight2[i] + sqrt(rho)*a1[i] +
+       a1[i] ~ dnorm(0,tau.a)
+       a2[i] ~ dnorm(0,tau.a)
+       a3[i] ~ dnorm(0,tau.a)
+       }
+
+       dumnode <- weight1[1] + weight2[1] + mz[1] + dz[1] + agemgram1[1] + agemgram2[1] + study[1]
+
+       rho ~ dunif(0,1)
+
+       b.int ~ dnorm(0,0.0001)
+       b.age ~ dnorm(0,0.0001)
+       b.wgt ~ dnorm(0,0.0001)
+       tau.a <- pow(sigma.a,-2)
+       sigma.a ~ dunif(0,1000)
+
+       tau.e <- pow(sigma.e,-2)
+       sigma.e ~ dunif(0,1000)
+
+       sigma2.a <- pow(sigma.a,2)
+       sigma2.e <- pow(sigma.e,2)
```

```
+
+       }",
+ file = "m4.jag")
> # Inits as a list
>
> d.ini <- list(rho = 0.5, b.int = 76, b.age = -0.75, b.wgt = -0.64, sigma.a = 16, sigma.e = 13.5)
> # Names of the parameters to monitor
>
> d.par <- c("b.age","b.int","b.wgt","rho","sigma2.a","sigma2.e","sigma.a","sigma.e")
> # Model compilation and burn-in
>
> d.mod <- jags.model( file = "m4.jag",
+                      data = a.dat,
+                  n.chains = 3,
+                     inits = d.ini,
+                   n.adapt = 1500 )
> # Sampling from the posterior
>
> d.res <- coda.samples( model = d.mod,
+                          var = d.par,
+                       n.iter = 1500,
+                         thin = 1 )
> # Take a look at the output from JAGS
>
> summary(d.res)
```

3.11 DIC and other model diagnostics

This exercise aims at showing how the diagnostics DIC (deviance information criterion) behave in a situation where we know the model fits and where we know the model does not fit

The idea is snatched from Bob O'Hara's website deepthoughtsandsilliness.blogspot.com/2007/12/focus-on-dic.html with a few minor modifications. The idea is to simulate two datasets using two models where one is a sub-model of the other, and use JAGS to fit the two data-generation models to both datasets.

First load the `rjags` library:

```
> library( rjags )
```

1. The idea is to generate data in 10 groups, each group having a mean drawn from a normal distribution. The difference between the two data-generating models is whether the distributions from which we draw the group means are identical or have means that vary with the group number.

   First we set up the number of groups and the group sizes ad generate the group indicator for all units we will simulate for:

```
> ng    <- 10
> gs    <- 50
> sd.w  <- 1
> sd.b  <- 1
> gr.no <- 1:ng
> group <- rep(gr.no, each=gs)
> beta  <- 1
```

   Then we simulate the first dataset where we first draw group means from normal distributions with means proportional to the group number.

```
> gr.mean1 <- rnorm( ng, beta*gr.no, sd.b)
>       y1 <- rnorm(length(group), gr.mean1[group], sd.w)
>    data1 <- list( N = length(y1),
+                   G = ng,
+               group = group,
+                   y = y1 )
```

   Then we repeat the exercise drawing group means from identical distributions:

```
> gr.mean2 <- rnorm( ng, mean(beta*gr.no), sd.b)
>       y2 <- rnorm(length(group), gr.mean2[group], sd.w)
>    data2 <- list( N = length(y2),
+                   G = ng,
+               group = group,
+                   y = y2 )
```

2. In order to show how the data looks we plot the data in two panels showing the differences in how we generated data:

```
> par(mfrow=c(1,2), mar=c(2.1,2.1,1.1,1.1),
+     oma=c(2,2,0,0), las=1, bty="n" )
> plot(jitter(group), y1, pch=3, col="grey40",ylim=range(c(y1,y2)), xaxt="n" )
> points(gr.no, gr.mean1, pch=3, cex=1.5, lwd=3, col="blue")
> axis( side=1, at=1:ng, labels=1:ng, col="transparent" )
> abline(0, beta, col="blue" )
> plot(jitter(group), y2, pch=3, col="grey40", ylim=range(c(y1,y2)), xaxt="n" )
> points(gr.no, gr.mean2, pch=3, cex=1.5, lwd=3, col="blue")
> axis( side=1, at=1:ng, labels=1:ng, col="transparent" )
> abline( h=mean(beta*gr.no), col="blue")
> mtext("Group", 1, outer=T)
> mtext("y", 2, outer=T)
```

3. We then set up the two models we used for generating data, in JAGS, as well as the parameters we want to monitor from the two models:

```
> # Model 1: Slope between groups
> cat("model{
+ for( i in 1:N )
+    {
+    y[i] ~ dnorm(muGrp[group[i]], tau.wti)
+    }
+ for( j in 1:G )
+    {
+    muGrp[j]   ~ dnorm(muG[j], tau.btw)
+      muG[j] <- mu0 + betaGrp*(j-5.5)
+    }
+       mu0   ~ dnorm (0.0, 1.0E-6)
+   betaGrp   ~ dnorm (0.0, 1.0E-6)
+   tau.wti <- pow(sigma.wti, -2)
+ sigma.wti   ~ dunif (0, 1000)
+   tau.btw <- pow(sigma.btw, -2)
+ sigma.btw   ~ dunif (0, 1000)
+ }",
```

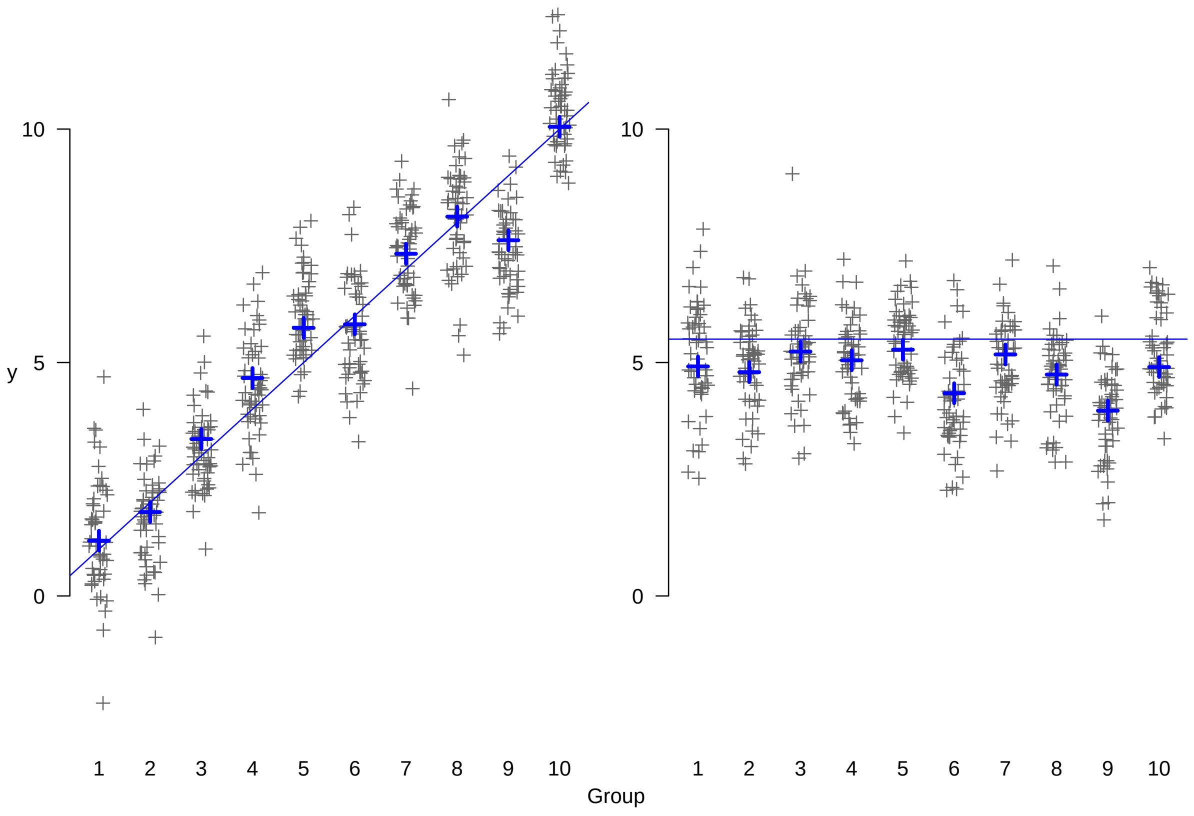


Figure 3.28: *The two datasets generated for illustration of model fitting diagnostics.*

```
+ file="model1.jag" )
> m1.par <- c("mu0","muGrp","betaGrp","sigma.wti","sigma.btw")
> # Model 2: Group means from identical distributions
> cat("model{
+ for( i in 1:N )
+     {
+     y[i] ~ dnorm(muGrp[group[i]], tau.wti)
+     }
+ for( j in 1:G )
+     {
+     muGrp[j]  ~ dnorm(mu0, tau.btw)
+     }
+       mu0  ~ dnorm (0.0, 1.0E-6)
+   tau.wti <- pow(sigma.wti, -2)
+ sigma.wti  ~ dunif (0, 1000)
+   tau.btw <- pow(sigma.btw, -2)
+ sigma.btw  ~ dunif (0, 1000)
+ }",
+ file="model2.jag")
> m2.par <- c("mu0","muGrp","sigma.wti","sigma.btw")
```

In order to run the models in `JAGS` we prudently set up initial values

```
> # Initial values
> inits1=list(list(mu0=0, betaGrp=1, sigma.btw=5, sigma.wti=1),
+             list(mu0=2, betaGrp=0, sigma.btw=5, sigma.wti=1) )
> inits2=list(list(mu0=0, sigma.btw=5, sigma.wti=1),
+             list(mu0=2, sigma.btw=5, sigma.wti=1) )
```

4. Once all this has been set up, we can fit the two models to the two datasets and inspect the results:

```
> # Model1 for data1
> m1d1.mod <- jags.model( file = "model1.jag",
+                         data = data1,
+                     n.chains = 2,
+                        inits = inits1,
+                      n.adapt = 1000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 1063

Initializing model

> m1d1.res <- coda.samples( m1d1.mod,
+                          var = m1.par,
+                       n.iter = 10000,
+                         thin = 10 )
> # Model1 for data2
> m1d2.mod <- jags.model( file = "model1.jag",
+                         data = data2,
+                     n.chains = 2,
+                        inits = inits1,
+                      n.adapt = 1000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 1063

Initializing model
```

```
> m1d2.res <- coda.samples( m1d2.mod,
+                            var = m1.par,
+                         n.iter = 10000,
+                           thin = 10 )
> # Model2 for data1
> m2d1.mod <- jags.model( file = "model2.jag",
+                           data = data1,
+                       n.chains = 2,
+                          inits = inits2,
+                        n.adapt = 1000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 1022

Initializing model

> m2d1.res <- coda.samples( m2d1.mod,
+                            var = m2.par,
+                         n.iter = 10000,
+                           thin = 10 )
> # Model2 for data2
> m2d2.mod <- jags.model( file = "model2.jag",
+                           data = data2,
+                       n.chains = 2,
+                          inits = inits2,
+                        n.adapt = 1000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 1022

Initializing model

> m2d2.res <- coda.samples( m2d2.mod,
+                            var = m2.par,
+                         n.iter = 10000,
+                           thin = 10 )
> summary( m1d1.res )$q

                 2.5%        25%        50%        75%     97.5%
betaGrp     0.7291544  0.8825424  0.9420689  1.0038989  1.146878
mu0         4.9616061  5.3927564  5.5750248  5.7551384  6.140843
muGrp[1]    1.0336575  1.2157459  1.3131269  1.4013919  1.577079
muGrp[2]    1.4411046  1.6011550  1.6986447  1.7928989  1.989970
muGrp[3]    2.8683389  3.0559593  3.1532644  3.2444498  3.415647
muGrp[4]    4.2294643  4.4120201  4.5047182  4.5987829  4.769644
muGrp[5]    5.7425186  5.9254667  6.0183691  6.1155522  6.303988
muGrp[6]    5.4162408  5.5951732  5.6835812  5.7761045  5.944644
muGrp[7]    7.0994518  7.2728650  7.3690948  7.4591183  7.650577
muGrp[8]    7.8248757  8.0012714  8.0927859  8.1850786  8.355299
muGrp[9]    7.1773096  7.3543971  7.4529952  7.5508691  7.718288
muGrp[10]  10.1617102 10.3295559 10.4266943 10.5191723 10.700563
sigma.btw   0.4963866  0.6741923  0.8004497  0.9797378  1.559873
sigma.wti   0.9396652  0.9777178  1.0001049  1.0228228  1.060608

> summary( m2d2.res )$q
```

```
                2.5%       25%       50%       75%     97.5%
mu0        4.5037009 4.7584810 4.8747735 4.9864860 5.2418517
muGrp[1]   4.8432304 5.0296512 5.1200653 5.2041167 5.3693166
muGrp[2]   4.6546448 4.8215746 4.9125821 5.0062570 5.1720393
muGrp[3]   4.9891466 5.1558449 5.2410242 5.3310003 5.4971653
muGrp[4]   4.7307851 4.9032815 4.9897211 5.0776309 5.2509070
muGrp[5]   5.1392445 5.3015489 5.3989199 5.4839998 5.6700611
muGrp[6]   3.9939789 4.1575997 4.2468308 4.3430589 4.5199623
muGrp[7]   4.7371859 4.8928386 4.9821561 5.0752727 5.2448484
muGrp[8]   4.4781883 4.6324549 4.7209118 4.8140982 4.9753242
muGrp[9]   3.6817211 3.8462492 3.9356203 4.0233085 4.2077565
muGrp[10]  4.9658082 5.1527503 5.2386908 5.3279502 5.5031844
sigma.btw  0.3242853 0.4378185 0.5274825 0.6394112 0.9962192
sigma.wti  0.9134098 0.9483695 0.9687341 0.9899994 1.0328566
```

We see that we in most cases get the parameters back that was used to generate the data, when we use the model that was used to generate the data — hardly surprising.

But when we use model 2 (the simpler one) to fit the data generated by model 1 (those with a slope), we get more or less the generated means in each of the groups back as posterior medians of `muGrp`:

```
> summary( m2d1.res )$q

                 2.5%        25%        50%       75%     97.5%
mu0         3.2679096  4.9258255  5.6015530  6.265118  7.894955
muGrp[1]    1.0449221  1.2200544  1.3174197  1.411092  1.602107
muGrp[2]    1.4109130  1.5885819  1.6852026  1.777890  1.951274
muGrp[3]    2.8649950  3.0479586  3.1458357  3.236773  3.422819
muGrp[4]    4.2317574  4.4109076  4.5098046  4.609598  4.782332
muGrp[5]    5.7744804  5.9498119  6.0457708  6.139968  6.298350
muGrp[6]    5.3983026  5.5796625  5.6787662  5.771591  5.950838
muGrp[7]    7.1179342  7.2816786  7.3814707  7.472044  7.657607
muGrp[8]    7.8373509  8.0027653  8.0994938  8.198310  8.378878
muGrp[9]    7.1233279  7.3035861  7.4031827  7.500409  7.680710
muGrp[10]  10.1465261 10.3351880 10.4325907 10.528867 10.719024
sigma.btw   2.0788726  2.7435160  3.2394344  3.889556  5.964470
sigma.wti   0.9394812  0.9784468  0.9989127  1.020748  1.063559

> gr.mean1

 [1]  1.182657  1.797204  3.361135  4.668211  5.740934  5.818173  7.328617
 [8]  8.124100  7.618937 10.047804
```

But we can also see that the price paid for nor modeling the mean in the groups correctly is that the variation between groups (that is around the stipulated model for the means) is that the between-group variation `sigma.btw` is grossly overestimated.

However, this is not the case when the more elaborate model is used; which should neither be a surprise, the model used to generate data is a proper sub-model of the one used to fit them, namely the model where $\beta = 0$.

```
> summary( m1d2.res )$q

                2.5%        25%         50%         75%      97.5%
betaGrp   -0.1994824 -0.1012177 -0.06136163 -0.02011444 0.07224254
mu0        4.4778879  4.7671907  4.87871650  4.98757742 5.24542659
muGrp[1]   4.8770241  5.0472218  5.13928995  5.22432422 5.39408978
muGrp[2]   4.6790185  4.8390653  4.93092531  5.01992583 5.19033750
```

```
muGrp[3]   4.9960562  5.1662560  5.25454408  5.34205960 5.51233249
muGrp[4]   4.7305637  4.9064697  5.00487643  5.09195445 5.25235893
muGrp[5]   5.1276759  5.3091500  5.39734419  5.48782536 5.66413609
muGrp[6]   3.9829154  4.1583229  4.24666354  4.33924050 4.51641582
muGrp[7]   4.7105297  4.8879684  4.97786354  5.07140498 5.23441844
muGrp[8]   4.4654096  4.6266208  4.71355330  4.80368660 4.96337075
muGrp[9]   3.6456729  3.8271673  3.91959005  4.01445079 4.20133736
muGrp[10]  4.9546955  5.1242265  5.21534358  5.30739520 5.48558048
sigma.btw  0.3044523  0.4299169  0.52579913  0.64767505 1.02203519
sigma.wti  0.9093876  0.9486424  0.96973612  0.99060099 1.03292208
```

5. If we want to assess the model fit by DIC (`pD`) or popt (`popt`), we must run the chain again in order to collect these statistics:

```
> pD11 <- dic.samples(m1d1.mod, n.iter=10000, thin = 10, type="pD" )
> pD12 <- dic.samples(m1d2.mod, n.iter=10000, thin = 10, type="pD" )
> pD21 <- dic.samples(m2d1.mod, n.iter=10000, thin = 10, type="pD" )
> pD22 <- dic.samples(m2d2.mod, n.iter=10000, thin = 10, type="pD" )
> pop11 <- dic.samples(m1d1.mod, n.iter=10000, thin = 10, type="popt" )
> pop12 <- dic.samples(m1d2.mod, n.iter=10000, thin = 10, type="popt" )
> pop21 <- dic.samples(m2d1.mod, n.iter=10000, thin = 10, type="popt" )
> pop22 <- dic.samples(m2d2.mod, n.iter=10000, thin = 10, type="popt" )
> pD11


Mean deviance:  1418
penalty 10.98
Penalized deviance: 1429


> pD12


Mean deviance:  1387
penalty 10.71
Penalized deviance: 1398


> pD21


Mean deviance:  1418
penalty 10.94
Penalized deviance: 1429


> pD22


Mean deviance:  1387
penalty 10.76
Penalized deviance: 1398


> pop11


Mean deviance:  1417
penalty 21.56
Penalized deviance: 1439


> pop12
```

```
Mean deviance:  1387
penalty 21.67
Penalized deviance: 1409
```

```
> pop21
```

```
Mean deviance:  1418
penalty 22.44
Penalized deviance: 1440
```

```
> pop22
```

```
Mean deviance:  1387
penalty 22.27
Penalized deviance: 1410
```

What we see is that the measures of fit are not very different between models. This is because the measure are measures of how well the model *predicts* and not measures of whether the models can be improved by adding further covariates.

The latter is of course also quite a futile expectation; it would be difficult for any criterion to guess what covariates were missing in data. Definition of covariates is always a subject matter definition.

3.12 Measurement comparison in oximetry

1. The model we consider is one where there is fixed difference between the two methods:

$$y_{(co),ir} - y_{(pulse),ir} = d_{ir} \sim \mathcal{N}(\delta, \sigma^2)$$

   (a) This is just a standard normal model with mean and standard deviation as parameters, and so easily fitted in R:

```
> library( Epi )
> oxw <- read.table( "../data/ox.dat", header=TRUE )
> str(oxw)

'data.frame':        177 obs. of  4 variables:
 $ item : int  1 1 1 2 2 2 3 3 3 4 ...
 $ repl : int  1 2 3 1 2 3 1 2 3 1 ...
 $ co   : num  78 76.4 77.2 68.7 67.6 68.3 82.9 80.1 80.7 62.3 ...
 $ pulse: int  71 72 73 68 67 68 82 77 77 43 ...

> m1 <- lm( I(pulse-co) ~ 1, data=oxw )
> summary( m1 )

Call:
lm(formula = I(pulse - co) ~ 1, data = oxw)

Residuals:
     Min       1Q   Median       3Q      Max
-19.0226  -3.5226  -0.4226   3.1774  29.8774

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -2.4774     0.4642  -5.337 2.88e-07

Residual standard error: 6.176 on 176 degrees of freedom
```

   A 95% confidence interval for the mean differnce can be found using `ci.lin` from the `Epi` package:

```
> ci.lin( m1 )

             Estimate    StdErr         z            P      2.5%     97.5%
(Intercept) -2.477401 0.4641864 -5.337083 9.445382e-08 -3.38719 -1.567613
```

   (b) The prior distribution $p(\sigma^2) \propto (\sigma^2)^{-1}$ corresponds to $\nu_0 = \sigma_0^2 = 0$ so we have

$$p(\sigma^2|d) = \text{Inv-}\chi^2(n-1, s^2)$$

   where $n = 177$ and s^2 is the standard deviation from the model. To obtain an observation Y from the scaled Inv-$\chi^2(n-1, s^2)$ distribution, first draw X from the χ^2_{n-1} distribution and then let $Y = (n-1)s^2/X$. The 2.5 and 97.5 percentiles of the χ^2_{n-1} distribution with $n = 177$ are found by:

```
> qchisq(c(0.025,0.975),177-1)

[1] 141.1571 214.6284
```

   so a 95% posterior region for σ^2 will be the inverse of these two values multiplied by $(n-1)s^2$, so a confidence interval for σ is the square root of this:

```
> sqrt( (177-1) * summary(m1)$sigma^2 / qchisq(c(0.975,0.025),177-1) )

[1] 5.592317 6.895788
```

(c) The posterior distribution of $(\delta - \bar{d})/(s_d/\sqrt{n})$ is a t-distribution with $n-1$ degrees of freedom. So a 95% posterior interval for δ is:

$$\bar{d} \pm t_{0.975}(n-1) \times (s_d/\sqrt{n})$$

which is easily accomplished as:

```
> n <- nrow( oxw )
> coef(m1) + c(-1,1) * qt(0.975,n-1) * ( summary(m1)$sigma / sqrt(n) )

[1] -3.393489 -1.561313
```

(d) To run this in JAGS we must provide a model specification, data, initial values and the parameters to monitor:

```
> library( rjags )
> cat( "model
+       {
+       for( i in 1:I )
+          {
+          d[i] ~ dnorm( delta, tausq )
+          }
+       tausq <- pow( sigma, -2 )
+       sigma ~ dunif( 0, 1000 )
+       delta ~ dnorm( 0, 0.000001 )
+       }",
+       file="m1.jag" )
> m1.dat <- list( d=oxw$co-oxw$pulse, I=nrow(oxw) )
> m1.ini <- list( list( sigma=5, delta=0 ),
+                 list( sigma=6, delta=1 ),
+                 list( sigma=4, delta=-1 ) )
> m1.par <- c("sigma","delta")
> m1.mod <- jags.model( file = "m1.jag",
+                       data = m1.dat,
+                   n.chains = length(m1.ini),
+                      inits = m1.ini,
+                    n.adapt = 20000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 186

Initializing model

> m1.res <- coda.samples( m1.mod,
+                     var = m1.par,
+                  n.iter = 20000,
+                    thin = 10 )
```

We can the n inspect the resulting object and plot the joint posterior distribution of `delta` and `sigma`.

```
> str( m1.res )

List of 3
 $ : mcmc [1:2000, 1:2] 2.69 2.56 3.18 3.3 1.7 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "delta" "sigma"
  ..- attr(*, "mcpar")= num [1:3] 20010 40000 10
 $ : mcmc [1:2000, 1:2] 2.33 2.63 2.53 1.22 2.15 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "delta" "sigma"
  ..- attr(*, "mcpar")= num [1:3] 20010 40000 10
 $ : mcmc [1:2000, 1:2] 2.52 2.63 2.39 2.56 1.88 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "delta" "sigma"
  ..- attr(*, "mcpar")= num [1:3] 20010 40000 10
 - attr(*, "class")= chr "mcmc.list"

> m1.mat <- as.matrix(m1.res)
> par( mar=c(6,6,1,1)/2, mgp=c(3,1,0)/1.6 )
> plot( m1.mat[,"delta"], m1.mat[,"sigma"],
+       xlab="delta", ylab="sigma",
+       pch=16, cex=0.4, las=1, bty="n" )
```

(e) We can just use `summary` function to get a 95% posterior interval for the parameters:

```
> summary( m1.res )

Iterations = 20010:40000
Thinning interval = 10
Number of chains = 3
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
delta 2.479 0.4681 0.006043       0.006021
sigma 6.218 0.3335 0.004305       0.004283

2. Quantiles for each variable:

       2.5%   25%   50%   75% 97.5%
delta 1.569 2.164 2.484 2.795 3.404
sigma 5.602 5.990 6.199 6.431 6.899
```

We can also show the posterior marginal densities:

```
> par( mfrow=c(1,2), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6, las=1 )
> plot( density( m1.mat[,"delta"] ), lwd=3,
+       xlab="delta", ylab="", bty="n", main="" )
> abline( v=quantile(m1.mat[,"delta"],probs=c(2.5,25,50,75,97.5)/100),
+         col="gray" )
> plot( density( m1.mat[,"sigma"] ), lwd=3,
+       xlab="delta", ylab="", bty="n", main="" )
> abline( v=quantile(m1.mat[,"sigma"],probs=c(2.5,25,50,75,97.5)/100),
+         col="gray" )
```

(f) We introduce limits $\delta \pm 2\sigma$ as nodes `agree.lo` and `agree.hi` in the `BUGS` code:

```
> cat( "model
+       {
+       for( i in 1:I )
+          {
+          d[i] ~ dnorm( delta, tausq )
+          }
+       tausq <- pow( sigma, -2 )
+       sigma ~ dunif( 0, 1000 )
+       delta ~ dnorm( 0, 0.000001 )
+     agree.lo <- delta - 2*sigma
+     agree.hi <- delta + 2*sigma
+       }",
+       file="m2.jag" )
> m2.dat <- list( d=oxw$co-oxw$pulse, I=nrow(oxw) )
> m2.ini <- list( list( sigma=5, delta=0 ),
+                 list( sigma=6, delta=1 ),
+                 list( sigma=4, delta=-1 ) )
> m2.par <- c("sigma","delta","agree.lo","agree.hi")
> m2.mod <- jags.model( file = "m2.jag",
+                       data = m2.dat,
+                   n.chains = length(m2.ini),
+                      inits = m2.ini,
+                    n.adapt = 20000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 189

Initializing model

> m2.res <- coda.samples( m2.mod,
+                        var = m2.par,
+                     n.iter = 20000,
+                       thin = 10 )
> summary( m2.res )

Iterations = 20010:40000
Thinning interval = 10
Number of chains = 3
Sample size per chain = 2000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:
```

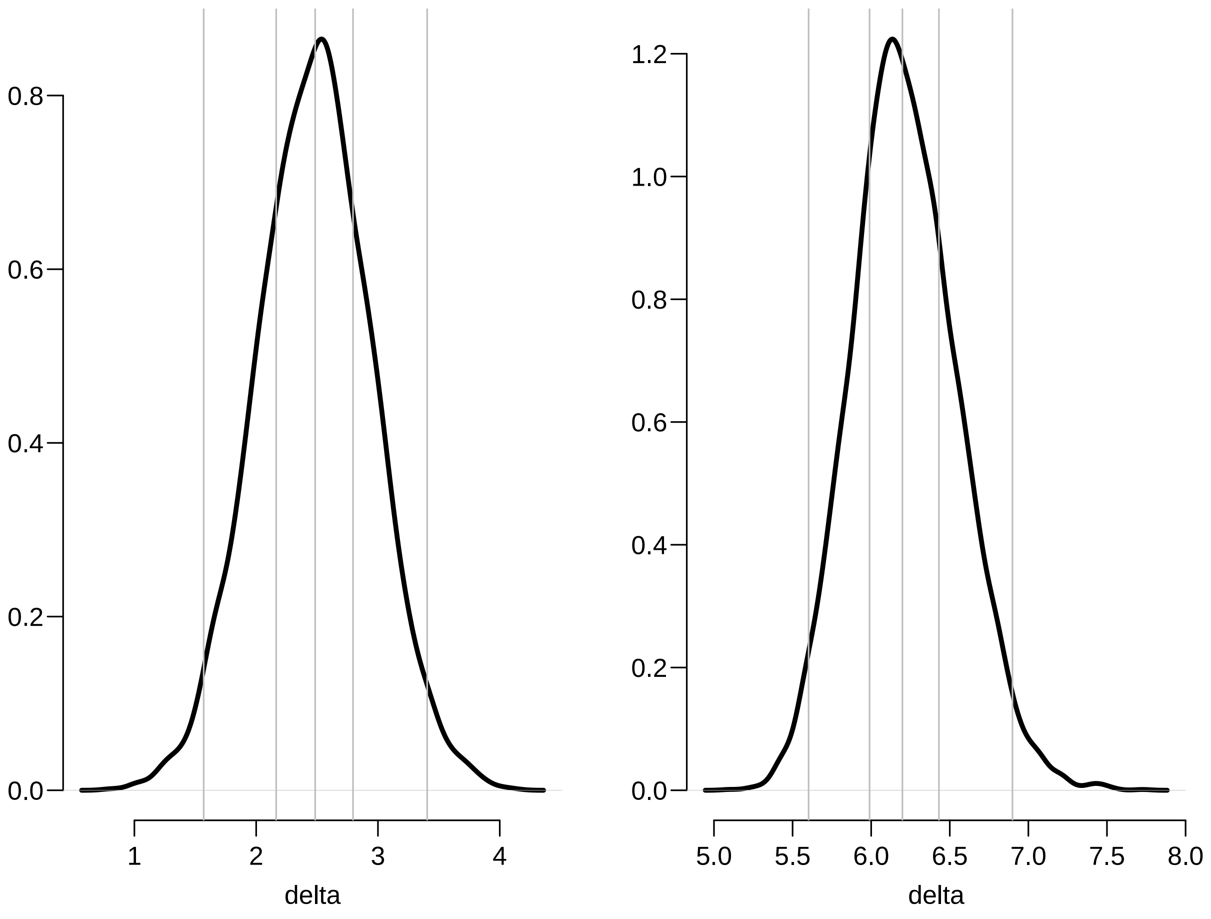


Figure 3.29: *Marginal posterior densities for the two parameters. The vertical lines are the 2.5, 25, 50, 75 and 97.5 percentiles.*

```
            Mean     SD Naive SE Time-series SE
agree.hi  14.929 0.8206 0.010594       0.011124
agree.lo  -9.968 0.8202 0.010588       0.010720
delta      2.481 0.4692 0.006057       0.006464
sigma      6.224 0.3365 0.004344       0.004374

2. Quantiles for each variable:

            2.5%     25%    50%    75%  97.5%
agree.hi  13.430  14.360 14.892 15.469 16.605
agree.lo -11.655 -10.503 -9.938 -9.412 -8.445
delta      1.579   2.165  2.477  2.800  3.396
sigma      5.600   5.995  6.209  6.440  6.911
```

One of the advantages of the BUGS machinery is that it is not necessary to re-run the code if you want the posterior of a simple function of the parameters; we can just use the posterior sample and calculate a posterior of these parameter functions:

```
> M1 <- as.matrix( m1.res )
> a1.lo <- M1[,"delta"] - 2*M1[,"sigma"]
> a1.hi <- M1[,"delta"] + 2*M1[,"sigma"]
> M2 <- as.matrix( m2.res )
> layout( cbind(1:2), heights=1:2 )
> par( mar=c(3,3,1,1), mgp=c(3,1,0)/1.6, las=1, bty="n" )
> plot( density( a1.hi ), type="l", xlim=c(-20,20), lwd=3, main="" )
> lines( density( a1.lo ), lwd=3 )
> lines( density( M2[,"agree.hi"] ), lwd=2, col="red" )
> lines( density( M2[,"agree.lo"] ), lwd=2, col="red" )
> plot( M2[,"agree.hi"], M2[,"agree.lo"],
+       xlab="Upper limit", ylab="Lower limit",
+       pch=16, cex=0.3 )
```

This point can be demonstrated using the posterior sample from model m2 directly:

```
> summary( M2[,"agree.lo"] - (M2[,"delta"]-2*M2[,"sigma"]) )

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      0       0       0       0       0       0
```

(g) If we instead use an informative prior corresponding to 95% in an interval 3% on either side of 0, *i.e.* $\mathcal{N}(0, 1.5^2)$, we change the JAGS code accordingly. Recall that JAGS parametrizes by the precision, i.e. the inverse variance so we use $1/1.5^2 = 0.44444$:

```
> cat( "model
+       {
+       for( i in 1:I )
+          {
+          d[i] ~ dnorm( delta, tausq )
+          }
+       tausq <- pow( sigma, -2 )
+       sigma ~ dunif( 0, 1000 )
+       delta ~ dnorm( 0, 0.4444444 )
+       }",
+       file="m3.jag" )
> m3.dat <- list( d=oxw$co-oxw$pulse, I=nrow(oxw) )
> m3.ini <- list( list( sigma=5, delta=0 ),
+                 list( sigma=6, delta=1 ),
+                 list( sigma=4, delta=-1 ) )
> m3.par <- c("sigma","delta")
> m3.mod <- jags.model( file = "m3.jag",
+                       data = m3.dat,
+                   n.chains = length(m3.ini),
+                          inits = m3.ini,
+                        n.adapt = 20000 )
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 186

Initializing model

> m3.res <- coda.samples( m3.mod,
+                       var = m3.par,
```

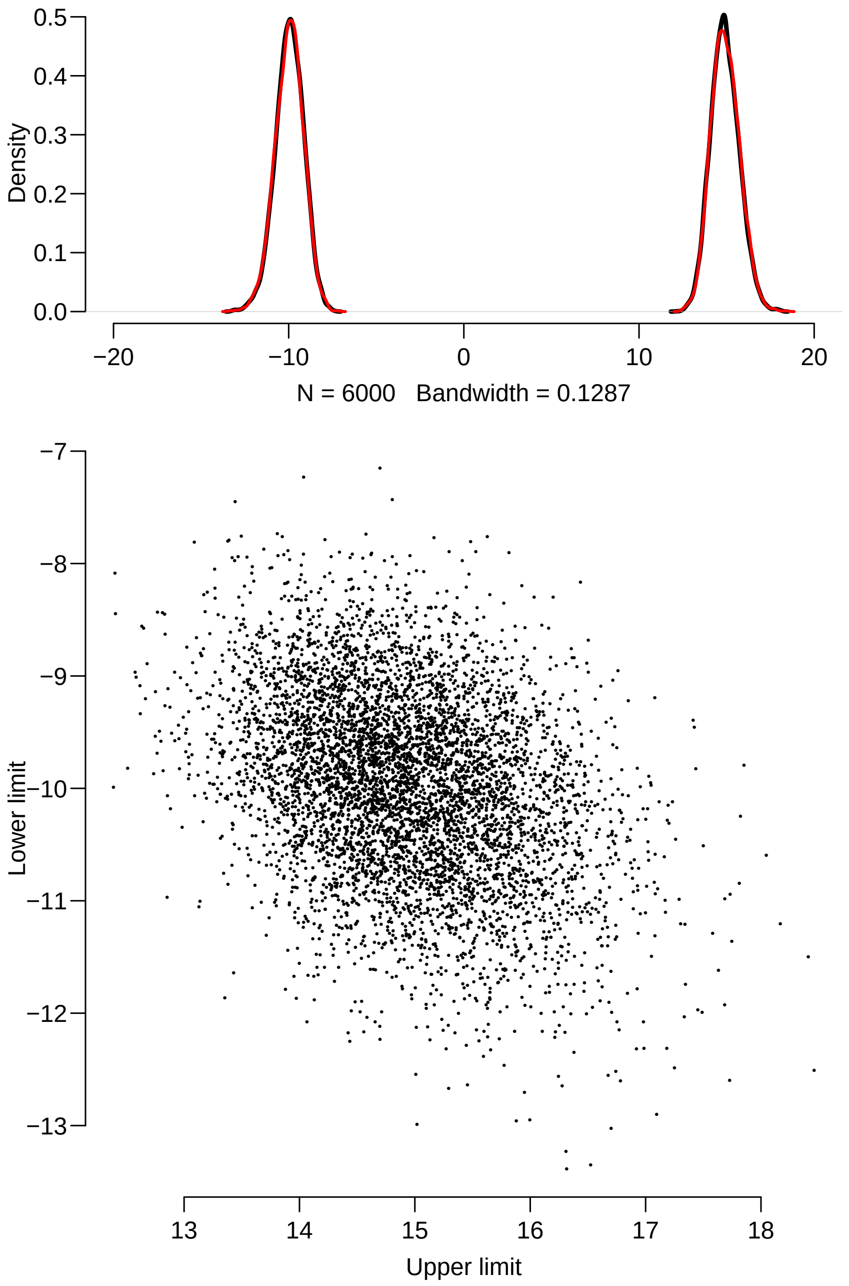


Figure 3.30: *Comparison of posterior densities for the upper and lower LoA from calculation inside* ***JAGS*** *(red) and from calculations on the posterior sample of* δ *and* σ*. The bottom plot is the joint posterior of the two limits.*

```
+                 n.iter = 20000,
+                   thin = 10 )
> summary( m3.res )

Iterations = 20010:40000
Thinning interval = 10
Number of chains = 3
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
delta 2.262 0.4526 0.005843       0.005693
sigma 6.220 0.3325 0.004292       0.004424

2. Quantiles for each variable:

       2.5%   25%   50%   75% 97.5%
delta 1.381 1.958 2.269 2.561 3.161
sigma 5.603 5.996 6.208 6.428 6.921
```

We compare the posterior in this case with the previously obtained, by plotting the posterior densities on top of each other. Also we include the prior density.

```
> M3 <- as.matrix( m3.res )
> par( mar=c(3,3,1,1), mgp=c(3,1,0)/1.6, las=1 )
> plot( density( M3[,"delta"]), type="l", col=gray(0.2), lwd=3,
+                               main="", bty="n", xlab="" )
> lines( density( M2[,"delta"] ), lwd=3, col="red" )
> xx <- seq(0,5,,200)
> lines( xx, dnorm(xx,mean=0,sd=1.5), lwd=3, col=gray(0.6) )
```

We see that the posterior is drawn toward 0 (the mean of the informative prior) and slightly narrower (corresponding to the larger amount of information)

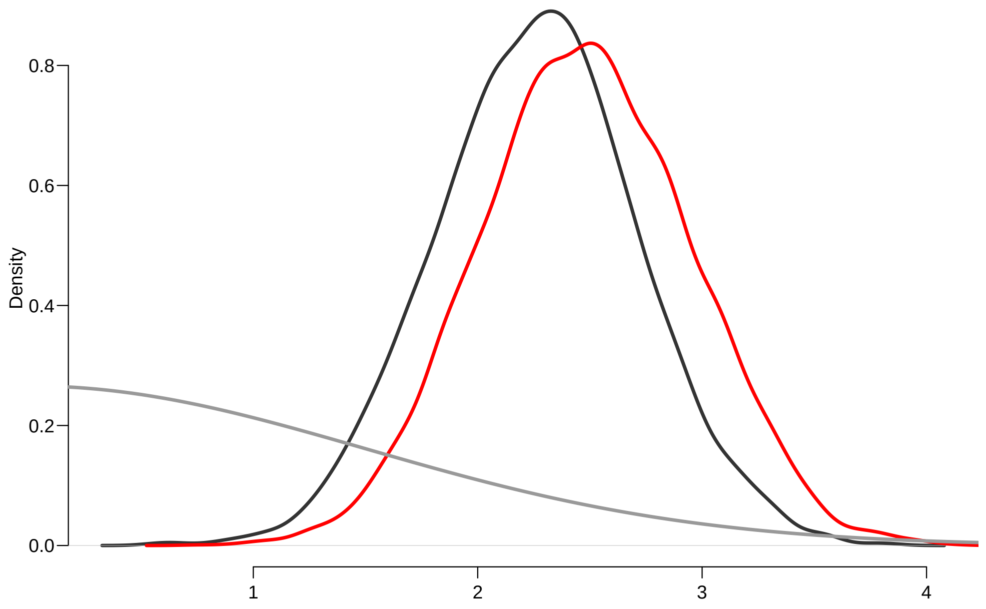


Figure 3.31: *Comparison of posterior densities using different priors for δ; informative is gray, uninformative is red. (Part of) the informative prior used is shown in light gray.*

2. In order to account for the individual effect of child, we introduce a *subject-specific* effect μ_i shared by all measurements on the i^{th} infant:

$$\begin{aligned} y_{\text{co},ir} &= \mu_i + e_{\text{co},ir} \\ y_{\text{pulse},ir} &= \mu_i + \delta + e_{\text{pulse},ir} \end{aligned}$$

where $e_{mij} \sim \text{N}(0, \sigma_m^2)$, $m = \text{co}, \text{pulse}$. Note that the error terms for the two methods are different as it would rather daft to assume that the measurement error were the same for two different methods.

(a) The distribution of $d_{ir} = y_{\text{co},ir} - y_{\text{pulse},ir}$ under this model is normal with mean δ and standard deviation $\sqrt{\sigma_{\text{co}}^2 + \sigma_{\text{pulse}}^2}$. So as far as the differences are concerned, the model is the same as above, but with this extended model we can actually identify the separate variances using the replicate measurements in the data.

(b) The expansion of the model to model the two types of measurement requires a bit or rearrangement in the code. Note that the nodes `mu.co[i]` are defined as stochastic nodes, whereas `mu.pl[i]` are deterministic as a sum of two stochastic nodes.

```
> cat( "model
+       {
+       for( i in 1:I )
+          {
+         mu.co[i] ~ dnorm( 0, 0.000001 )
+         mu.pl[i] <- mu.co[i] + delta
+          y.co[i] ~ dnorm( mu.co[i], tausq.co )
+          y.pl[i] ~ dnorm( mu.pl[i], tausq.pl )
+          }
+       tausq.co <- pow( sigma.co, -2 )
+       tausq.pl <- pow( sigma.pl, -2 )
+       sigma.co ~ dunif( 0, 1000 )
+       sigma.pl ~ dunif( 0, 1000 )
+       delta ~ dnorm( 0, 0.000001 )
+       }",
+       file="m4.jag" )
> nr <- nrow(oxw)
> m4.dat <- list( y.co=oxw$co, y.pl=oxw$pulse, I=nr )
> m4.ini <- list( list( sigma.co=5, sigma.pl=5, mu.co=rep(80,nr), delta=0 ),
+                 list( sigma.co=6, sigma.pl=6, mu.co=rep(70,nr), delta=1 ),
+                 list( sigma.co=4, sigma.pl=4, mu.co=rep(90,nr), delta=-1 ) )
> m4.par <- c("sigma.pl","sigma.co","delta")
> m4.mod <- jags.model( file = "m4.jag",
+                       data = m4.dat,
+                   n.chains = length(m4.ini),
+                      inits = m4.ini,
+                    n.adapt = 20000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 719

Initializing model

> m4.res <- coda.samples( m4.mod,
+                        var = m4.par,
+                    n.iter = 20000,
+                      thin = 10 )
> summary( m4.res )
```

```
Iterations = 20010:40000
Thinning interval = 10
Number of chains = 3
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
delta    -2.462 0.4659 0.006015        0.01035
sigma.co  4.372 1.6341 0.021097        0.11933
sigma.pl  3.729 1.8436 0.023801        0.13718

2. Quantiles for each variable:

            2.5%    25%    50%    75%  97.5%
delta    -3.4029 -2.769 -2.463 -2.150 -1.550
sigma.co  0.9628  3.126  4.726  5.768  6.570
sigma.pl  0.3665  2.229  4.036  5.337  6.383
```

(c) When we get to these slightly more complicated models it is prudent to make a traceplot to ensure that the convergence i acceptable. In this case it does not really seem to be the case; it appears that the two variance components are very closely negatively correlated. Specifically the joint distribution is concentrated on a circle with radius 6, i.e. the sum of the two variances is 36, and this is pretty well determined, but the relative size of them is not.

```
> print( xyplot( m4.res[,c("delta","sigma.co","sigma.pl")],
+                aspect="fill", layout=c(3,1) ) )

> M4 <- as.matrix( m4.res, chains=TRUE )
> plot( M4[,"sigma.co"], M4[,"sigma.pl"], pch=16, cex=0.5, col=rainbow(3)[M4[,"CHAIN"]] )
```

The simplest overview of the data can be made by the `densityplot` method which gives an overview of the monitored parameters:

```
> print( densityplot( m4.res[,c("delta","sigma.co","sigma.pl")],
+                     aspect="fill", layout=c(3,1), lwd=3 ) )
```

3. In order to account for the linking of the replicates we incorporate a random effect a_{ir} with variance ω^2, modelling the individual variation between timepoints of measurement:

$$\begin{aligned} y_{\text{co},ir} &= \mu_i + a_{ir} + e_{\text{co},ir} \\ y_{\text{pulse},ir} &= \mu_i + \delta + a_{ir} + e_{\text{pulse},ir} \end{aligned}$$

(a) We modify the JAGS code by including specification of this new variance component. In order to do this we must supply the replicate number from the data. Note the nested indexing needed in order to get the right random effect added in the right place.

```
> cat( "model
+       {
+       for( i in 1:I )
+          {
+            mu[i] ~ dunif( 0, 100 )
+         mu.co[i] <- mu[i] + a[i,repl[i]]
+         mu.pl[i] <- mu[i] + a[i,repl[i]] + delta
+          y.co[i] ~ dnorm( mu.co[i], tausq.co )
+          y.pl[i] ~ dnorm( mu.pl[i], tausq.pl )
+          for( r in 1:3 )
+            {
+            a[i,r] ~ dnorm( 0, iomegasq )
+            }
+         }
+      tausq.co <- pow( sigma.co, -2 )
+      tausq.pl <- pow( sigma.pl, -2 )
+      iomegasq <- pow( omega, -2 )
+      sigma.co ~ dunif( 0, 1000 )
+      sigma.pl ~ dunif( 0, 1000 )
+      omega    ~ dunif( 0, 1000 )
+      delta ~ dnorm( 0, 0.000001 )
+      }",
+      file="m5.jag" )
> m5.dat <- list( y.co=oxw$co, y.pl=oxw$pulse, repl=oxw$repl, I=nr )
> m5.ini <- list( list( sigma.co=5, sigma.pl=5, omega=4, mu=rep(80,nr), delta= 0 ),
+                 list( sigma.co=6, sigma.pl=6, omega=4, mu=rep(70,nr), delta= 1 ),
+                 list( sigma.co=4, sigma.pl=4, omega=4, mu=rep(90,nr), delta=-1 ) )
> m5.par <- c("sigma.pl","sigma.co","omega","delta")
> m5.mod <- jags.model( file = "m5.jag",
+                       data = m5.dat,
+                   n.chains = length(m5.ini),
+                      inits = m5.ini,
+                    n.adapt = 20000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 1607

Initializing model

> m5.res <- coda.samples( m5.mod,
+                     var = m5.par,
+                  n.iter = 20000,
+                    thin = 10 )
> summary( m5.res )

Iterations = 20010:40000
Thinning interval = 10
Number of chains = 3
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
delta    -2.503 0.4891 0.006314        0.01596
omega     2.191 1.6189 0.020900        0.12246
sigma.co  4.177 1.8490 0.023870        0.14215
sigma.pl  3.819 1.8763 0.024223        0.14160

2. Quantiles for each variable:

             2.5%     25%    50%    75%  97.5%
delta    -3.49265 -2.8273 -2.496 -2.168 -1.563
omega     0.08341  0.8338  1.872  3.271  5.865
sigma.co  0.32641  2.7818  4.679  5.754  6.576
sigma.pl  0.40115  2.1964  4.062  5.502  6.509
```

(b) We can then make a traceplot of the chains sampled to see if they have converged. Based on the result shown in figure 3.34 it is a bit difficult to say, particularly for the variance parameters σ_{co}, σ_{pulse} and ω^2.

```
> print( xyplot( m5.res[,c("delta","omega","sigma.co","sigma.pl")],
+                aspect="fill", scales="same", layout=c(4,1) ) )
```

(c) We can also explore the relationship between the varaince estimates by maing a plot of the marginal 2-dimenstinal posterior distributions:

```
> M5 <- as.matrix( m5.res )
> pairs( M5[,-1], gap=0, pch=16, cex=0.3 )
```

It is seen in figure 3.35 that neither σ_{co} nor σ_{pulse} are particularly well determined, but their sum is — or rather as seem from the shape — the sum of theirs squares is. This is pretty much in line with common sense: The amount of data needed to determine the LoA is considerably smaller than the amount of data need to sort out the relative precision of the two methods.

(d) The model can also be fitted by conventional methods, in this case we resort to `lme`. For this we first stack the data and then run the model.

```
> oxl <- data.frame( y = c(oxw$co,oxw$pulse),
+                 repl = factor( rep(oxw$repl,2) ) ,
+                   id = factor( rep(oxw$item,2) ),
+                 meth = factor( rep(c("co","pulse"),each=177) ) )
> library( nlme )
> m1 <- lme( y ~ meth + id,
+                random = list( id = pdIdent( ~ repl-1 ) ),
+                weights = varIdent( form = ~1 | meth ),
+                data = oxl,
+                control = lmeControl(returnObject=TRUE) )
> m1

Linear mixed-effects model fit by REML
  Data: oxl
  Log-restricted-likelihood: -928.2544
  Fixed: y ~ meth + id
 (Intercept)    methpulse          id2          id3          id4          id5
 76.55534468  -2.47740113  -7.89502948   4.65685241 -11.28966181  -1.47555983
         id6          id7          id8          id9         id10         id11
  2.13562002   9.39463233   3.73777992  -4.99939663 -18.78304003  12.66927107
        id12         id13         id14         id15         id16         id17
-48.82331286   4.40123881  -3.66225215   6.23157059   0.48016527  13.40114334
        id18         id19         id20         id21         id22         id23
  1.48858186  -2.87219320  -1.26322060   5.64182935  -0.58513579   3.47155776
        id24         id25         id26         id27         id28         id29
  7.93409556   1.77884704   2.27263771  -9.33914552 -12.38561237   0.49639508
        id30         id31         id32         id33         id34         id35
  3.28705740 -29.97656035   5.86498335   5.75400972   8.86758775   1.12199462
        id36         id37         id38         id39         id40         id41
  3.49839611   3.56750833   6.61899307   1.73377785  -8.49118627   0.29487062
        id42         id43         id44         id45         id46         id47
 -5.97335257 -22.83052270 -17.79787217   1.82712400   4.46314117   2.91386369
        id48         id49         id50         id51         id52         id53
 -4.66545993  10.83433385 -25.14483090 -19.82772738  -0.35877402  -4.90744813
        id54         id55         id56         id57         id58         id59
 -0.05488344  11.70312835   9.29807839  12.48918523  13.11478478  14.47416217
        id60         id61
  7.63341276  -1.66927107

Random effects:
 Formula: ~repl - 1 | id
 Structure: Multiple of an Identity
          repl1   repl2   repl3 Residual
StdDev: 2.92452 2.92452 2.92452 3.005045

Variance function:
 Structure: Different standard deviations per stratum
 Formula: ~1 | meth
 Parameter estimates:
      co    pulse
1.000000 1.795366
```

```
Number of Observations: 354
Number of Groups: 61
```

The estimates from the REML-model are $\hat{\sigma}_{\text{co}} = 3.005$ $\hat{\sigma}_{\text{pulse}} = 3.005 \times 1.795 = 5.40$ and $\omega = 2.92$, where the posterior medians are for these are 4.25, 4.47 and 2.37.

4. The simplest way to allow for a difference that varies by the true measurement levels is to introduce a linear relationship between the means:

$$\begin{aligned} y_{\text{co},ir} &= \mu_i + a_{ir} + e_{\text{co},ir} \\ y_{\text{pulse},ir} &= \alpha + \beta(\mu_i + a_{ir}) + e_{\text{pulse},ir} \end{aligned}$$

   (a) We extend the JAGS code by an extra mean value parameter, β, and rename the other to α, as this no longer represents a general difference between methods:

```
> cat( "model
+       {
+       for( i in 1:I )
+          {
+            mu[i] ~ dunif( 0, 100 )
+         mu.co[i] <-                  mu[i] + a[i,repl[i]]
+         mu.pl[i] <- alpha + beta * ( mu[i] + a[i,repl[i]] )
+          y.co[i] ~ dnorm( mu.co[i], tausq.co )
+          y.pl[i] ~ dnorm( mu.pl[i], tausq.pl )
+          for( r in 1:3 )
+             {
+             a[i,r] ~ dnorm( 0, iomegasq )
+             }
+          }
+       tausq.co <- pow( sigma.co, -2 )
+       tausq.pl <- pow( sigma.pl, -2 )
+       iomegasq <- pow( omega, -2 )
+       sigma.co ~ dunif( 0, 1000 )
+       sigma.pl ~ dunif( 0, 1000 )
+       omega    ~ dunif( 0, 1000 )
+       alpha ~ dnorm( 0, 0.000001 )
+       beta  ~ dunif( 0, 2 )
+       }",
+       file="m6.jag" )
> m6.dat <- list( y.co=oxw$co, y.pl=oxw$pulse, repl=oxw$repl, I=nr )
> m6.ini <- list( list( sigma.co=5, sigma.pl=5, omega=4 ),
+                 list( sigma.co=6, sigma.pl=6, omega=4 ),
+                 list( sigma.co=4, sigma.pl=4, omega=4 ) )
> m6.par <- c("sigma.pl","sigma.co","omega","alpha","beta")
> m6.mod <- jags.model( file = "m6.jag",
+                       data = m6.dat,
+                   n.chains = length(m6.ini),
+                      inits = m6.ini,
+                    n.adapt = 20000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 1785

Initializing model

> m6.res <- coda.samples( m6.mod,
+                         var = m6.par,
+                      n.iter = 20000,
+                        thin = 10 )
> summary( m6.res )
```

```
Iterations = 20010:40000
Thinning interval = 10
Number of chains = 3
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
alpha    10.8703 2.70845 0.0349660       0.158250
beta      0.8242 0.03531 0.0004559       0.002079
omega     2.3479 1.68452 0.0217471       0.128204
sigma.co  2.8861 2.14045 0.0276331       0.160357
sigma.pl  4.7948 1.52543 0.0196933             NA

2. Quantiles for each variable:

           2.5%    25%     50%     75%   97.5%
alpha    5.6742 9.1164 10.8433 12.6909 16.2287
beta     0.7540 0.8000  0.8253  0.8474  0.8922
omega    0.1322 0.8423  2.1089  3.5403  6.1775
sigma.co 0.1237 0.9756  2.4730  4.4464  7.1080
sigma.pl 0.4695 4.4891  5.3726  5.7609  6.2925
```

(b) We might as well have chosen pulse-oximetry as the reference method and re-expressed the model as

$$\begin{aligned} y_{\text{co},ir} &= \alpha^\star + \beta^\star(\mu_i + a_{ir}) + e_{\text{co},ir} \\ y_{\text{pulse},ir} &= \mu_i + a_{ir} + e_{\text{pulse},ir} \end{aligned}$$

Swapping the reference method is a pretty straightforward change to the JAGS program:

```
> cat( "model
+        {
+        for( i in 1:I )
+           {
+             mu[i] ~ dunif( 0, 100 )
+          mu.co[i] <- alpha + beta * ( mu[i] + a[i,repl[i]] )
+          mu.pl[i] <-                  mu[i] + a[i,repl[i]]
+           y.co[i] ~ dnorm( mu.co[i], tausq.co )
+           y.pl[i] ~ dnorm( mu.pl[i], tausq.pl )
+           for( r in 1:3 )
+              {
+              a[i,r] ~ dnorm( 0, iomegasq )
+              }
+           }
+        tausq.co <- pow( sigma.co, -2 )
+        tausq.pl <- pow( sigma.pl, -2 )
+        iomegasq <- pow( omega, -2 )
+        sigma.co ~ dunif( 0, 1000 )
+        sigma.pl ~ dunif( 0, 1000 )
+        omega    ~ dunif( 0, 1000 )
+        alpha ~ dnorm( 0, 0.000001 )
+        beta  ~ dunif( 0, 2 )
+        }",
+        file="m7.jag" )
> m7.dat <- list( y.co=oxw$co, y.pl=oxw$pulse, repl=oxw$repl, I=nr )
> m7.ini <- list( list( sigma.co=5, sigma.pl=5, omega=4 ),
+                 list( sigma.co=6, sigma.pl=6, omega=4 ),
+                 list( sigma.co=4, sigma.pl=4, omega=4 ) )
> m7.par <- c("sigma.pl","sigma.co","omega","alpha","beta")
> m7.mod <- jags.model( file = "m7.jag",
+                       data = m7.dat,
+                   n.chains = length(m7.ini),
+                      inits = m7.ini,
+                    n.adapt = 20000 )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 1785

Initializing model

> m7.res <- coda.samples( m7.mod,
+                         var = m7.par,
+                      n.iter = 20000,
+                        thin = 10 )
> summary( m7.res )

Iterations = 20010:40000
Thinning interval = 10
Number of chains = 3
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
alpha    8.6678 2.88026 0.0371840        0.18685
beta     0.9159 0.03876 0.0005005        0.00252
omega    2.8777 1.78138 0.0229975        0.14397
sigma.co 3.9881 2.01352 0.0259945        0.15107
sigma.pl 4.1521 2.17432 0.0280704        0.16413

2. Quantiles for each variable:

           2.5%   25%    50%     75%   97.5%
alpha    2.7470 6.757 8.7049 10.8068 13.8057
beta     0.8456 0.887 0.9156  0.9415  0.9967
omega    0.2720 1.562 2.4835  3.9035  6.9768
sigma.co 0.1055 2.227 4.6423  5.7426  6.5103
sigma.pl 0.2789 2.287 4.3254  6.1709  7.2585
```

(c) If $\alpha + \beta\mu = \xi$ then we have $\mu = -\alpha/\beta + \xi/\beta$, hence the relationship between the parameters of the means in the two formulations are:

$$\beta^\star = 1/\beta \quad \text{and} \quad \alpha^\star = \alpha/\beta$$

(d) The `summary` function for `mcmc.list` objects allows you to extract all the relevant quantities and check whether the relationship is fulfilled for the either the mean or the median:

```
> # Mean
> ( ab6 <- summary( m6.res )$statistics[c("alpha","beta"),"Mean"] )

     alpha       beta
10.8702543  0.8242373

> ( ab7 <- summary( m7.res )$statistics[c("alpha","beta"),"Mean"] )

    alpha      beta
8.6677989 0.9158546

> abt <- c( -ab6[1]/ab6[2], 1/ab6[2] )
> round( cbind( ab6, ab7, abt ), 3 )

         ab6   ab7     abt
alpha 10.870 8.668 -13.188
beta   0.824 0.916   1.213
```

```
> # Median
> ( ab6 <- summary( m6.res )$quantiles[c("alpha","beta"),"50%"] )

     alpha       beta
10.8433312  0.8253322

> ( ab7 <- summary( m7.res )$quantiles[c("alpha","beta"),"50%"] )

    alpha      beta
8.7049174 0.9155644

> abt <- c( -ab6[1]/ab6[2], 1/ab6[2] )
> round( cbind( ab6, ab7, abt ), 3 )

         ab6   ab7     abt
alpha 10.843 8.705 -13.138
beta   0.825 0.916   1.212
```

Apparently the two pieces of BUGS code do not refer to the same model. Despite the fact that the model specifications look deceptively identical they do not give the same relationship between the models. In fact the two models are (bar the variance components) pretty close to the standard regressions of one method on the other:

```
> round(ci.lin(lm(pulse~co,data=oxw))[,c(1,5,6)],3)

            Estimate  2.5%  97.5%
(Intercept)   11.010 5.681 16.339
co             0.822 0.752  0.891

> round(summary(m6.res)$quantiles[4:5,c(3,1,5)],3)

           50%  2.5% 97.5%
sigma.co 2.473 0.124 7.108
sigma.pl 5.373 0.470 6.292

> round(ci.lin(lm(co~pulse,data=oxw))[,c(1,5,6)],3)

            Estimate 2.5%  97.5%
(Intercept)    8.503 2.75 14.256
pulse          0.918 0.84  0.995

> round(summary(m7.res)$quantiles[4:5,c(3,1,5)],3)

           50%  2.5% 97.5%
sigma.co 4.642 0.106 6.510
sigma.pl 4.325 0.279 7.258
```

In conclusion, an a-symmetric formulation of the model in JAGS may lead to wrong results. The specification of a model with certain symmetries should reflect these.

5. In order to get the model right we reformulate it so that it is symmetric in the two methods:

$$\begin{aligned} y_{\text{co},ir} &= \alpha_{\text{co}} + \beta_{\text{co}}(\mu_i + a_{ir}) + e_{\text{co},ir} \\ y_{\text{pulse},ir} &= \alpha_{\text{pulse}} + \beta_{\text{pulse}}(\mu_i + a_{ir}) + e_{\text{pulse},ir} \end{aligned}$$

(a) The relationship between the means of the two methods is found by setting all the variance components to 0 and then isolating μ_i from the first equation and inserting in the second:

$$\begin{aligned}
\mu_i &= (y_{\text{co}} - \alpha_{\text{co}})/\beta_{\text{co}} \\
&\Downarrow \\
y_{\text{pulse}} &= \alpha_{\text{pulse}} + \beta_{\text{pulse}}(y_{\text{co}} - \alpha_{\text{co}})/\beta_{\text{co}} \\
&= \left(\alpha_{\text{pulse}} - \alpha_{\text{co}}\frac{\beta_{\text{pulse}}}{\beta_{\text{co}}}\right) + \frac{\beta_{\text{pulse}}}{\beta_{\text{co}}} y_{\text{co}}
\end{aligned}$$

So the relevant parameters in terms of those in the model are

$$\alpha_{\text{pulse|co}} = \alpha_{\text{pulse}} - \alpha_{\text{co}}\frac{\beta_{\text{pulse}}}{\beta_{\text{co}}} \qquad \beta_{\text{pulse|co}} = \frac{\beta_{\text{pulse}}}{\beta_{\text{co}}}$$

(b)

(c) The modification is quite straightforward, however it should be noted that even if the model is over-parametrized, you can still get `JAGS` to run the chains, but there is no guarantee for convergence. You might for example see the μ_is wander off to infinity and the βs going toward 0. So precisely in this case it is essential to have a finite support for the prior of the μs as this ensures a finite support for the *posterior* of the μs too.

```
> cat( "model
+       {
+       for( i in 1:I )
+          {
+            mu[i] ~ dunif( 0, 100 )
+         mu.co[i] <- alpha.co + beta.co * ( mu[i] + a[i,repl[i]] )
+         mu.pl[i] <- alpha.pl + beta.pl * ( mu[i] + a[i,repl[i]] )
+          y.co[i] ~ dnorm( mu.co[i], tausq.co )
+          y.pl[i] ~ dnorm( mu.pl[i], tausq.pl )
+          for( r in 1:3 )
+             {
+             a[i,r] ~ dnorm( 0, iomegasq )
+             }
+          }
+       tausq.co <- pow( sigma.co, -2 )
+       tausq.pl <- pow( sigma.pl, -2 )
+       iomegasq <- pow( omega, -2 )
+       sigma.co ~ dunif( 0, 1000 )
+       sigma.pl ~ dunif( 0, 1000 )
+       omega    ~ dunif( 0, 1000 )
+       alpha.co ~ dnorm( 0, 0.000001 )
+       alpha.pl ~ dnorm( 0, 0.000001 )
+       beta.co  ~ dunif( 0, 2 )
+       beta.pl  ~ dunif( 0, 2 )
+       }",
+       file="m8.jag" )
> m8.dat <- list( y.co=oxw$co, y.pl=oxw$pulse, repl=oxw$repl, I=nrow(oxw) )
> m8.ini <- list( list( sigma.co=5, sigma.pl=5, omega=4 ),
+                 list( sigma.co=6, sigma.pl=6, omega=4 ),
+                 list( sigma.co=4, sigma.pl=4, omega=4 ) )
> m8.par <- c("sigma.pl","sigma.co","omega",
+            "alpha.pl","alpha.co",
+             "beta.pl", "beta.co")
> m8.mod <- jags.model( file = "m8.jag",
+                       data = m8.dat,
+                   n.chains = length(m8.ini),
+                      inits = m8.ini,
+                    n.adapt = 20000 )
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 2141

Initializing model

> m8.res <- coda.samples( m8.mod,
+                    var = m8.par,
+                 n.iter = 20000,
+                   thin = 10 )
> summary( m8.res )

Iterations = 20010:40000
Thinning interval = 10
Number of chains = 3
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
alpha.co  69.8536  2.54291 0.0328288       0.176750
alpha.pl  67.7044  2.38949 0.0308482       0.165661
beta.co    0.1181  0.04715 0.0006087       0.003374
beta.pl    0.1112  0.04380 0.0005654       0.003133
omega    113.9745 59.18509 0.7640763       4.261394
sigma.co   3.9677  1.93954 0.0250394       0.138669
sigma.pl   3.9227  1.65836 0.0214093       0.120637

2. Quantiles for each variable:

             2.5%      25%     50%      75%    97.5%
alpha.co 64.63038 68.07376 70.0058  71.8003  74.1636
alpha.pl 62.86919 65.95268 68.0185  69.4978  71.7307
beta.co   0.04542  0.07647  0.1166   0.1524   0.2105
beta.pl   0.04188  0.07389  0.1100   0.1415   0.2007
omega    49.06948 71.79005 94.7646 146.2587 258.9672
sigma.co  0.05500  2.39208  4.5474   5.6247   6.4437
sigma.pl  0.64964  2.59258  4.2322   5.4084   6.1727
```

(d) Once we have run the chains we can inspect the traces using `xyplot`; the subsetting is to get the displays in the right order — panels are filled from bottom left going left then up.

```
> print(xyplot( m8.res[,c(7,3,6,2,5,1,4)], layout=c(2,4), aspect="fill" ))
```

(e) The relevant parameters are the intercepts and the slopes in the linear relation between the methods. Therefore we compute these 4. Currently this is a bit of a hazzle; first convert the `mcmc` components to a dataframe, do the computations and turn it back into a `mcmc` object:

```
> m8b.res <- m8.res
> m8.res <- m8b.res
> str( m8.res )

List of 3
 $ : mcmc [1:2000, 1:7] 74.8 74 75 75.1 75.7 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:7] "alpha.co" "alpha.pl" "beta.co" "beta.pl" ...
  ..- attr(*, "mcpar")= num [1:3] 20010 40000 10
 $ : mcmc [1:2000, 1:7] 64.7 64.8 65.1 65.9 66.4 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:7] "alpha.co" "alpha.pl" "beta.co" "beta.pl" ...
  ..- attr(*, "mcpar")= num [1:3] 20010 40000 10
 $ : mcmc [1:2000, 1:7] 69.3 69.4 69.5 69.5 69 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:7] "alpha.co" "alpha.pl" "beta.co" "beta.pl" ...
  ..- attr(*, "mcpar")= num [1:3] 20010 40000 10
 - attr(*, "class")= chr "mcmc.list"

> for( i in 1:length(m8.res) )
+    {
+ att <- attributes( m8.res[[i]] )
+ dfr <- as.data.frame( m8.res[[i]] )
+ dfr$beta.co.pl  <- dfr$beta.co /  dfr$beta.pl
+ dfr$alpha.co.pl <- dfr$alpha.co - dfr$alpha.pl * dfr$beta.co.pl
+ dfr$beta.pl.co  <- dfr$beta.pl /  dfr$beta.co
+ dfr$alpha.pl.co <- dfr$alpha.pl - dfr$alpha.co * dfr$beta.pl.co
+ dfr <- as.matrix( dfr )
+ att$dim <- dim( dfr )
+ att$dimnames <- dimnames( dfr )
+ attributes( dfr ) <- att
+ m8.res[[i]] <- dfr
+    }
> str( m8.res )

List of 3
 $ : mcmc [1:2000, 1:11] 74.8 74 75 75.1 75.7 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:11] "alpha.co" "alpha.pl" "beta.co" "beta.pl" ...
  ..- attr(*, "mcpar")= num [1:3] 20010 40000 10
 $ : mcmc [1:2000, 1:11] 64.7 64.8 65.1 65.9 66.4 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:11] "alpha.co" "alpha.pl" "beta.co" "beta.pl" ...
  ..- attr(*, "mcpar")= num [1:3] 20010 40000 10
 $ : mcmc [1:2000, 1:11] 69.3 69.4 69.5 69.5 69 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:11] "alpha.co" "alpha.pl" "beta.co" "beta.pl" ...
  ..- attr(*, "mcpar")= num [1:3] 20010 40000 10
 - attr(*, "class")= chr "mcmc.list"

> summary( m8.res )

Iterations = 20010:40000
Thinning interval = 10
Number of chains = 3
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
alpha.co     69.8536  2.54291 0.0328288       0.176750
alpha.pl     67.7044  2.38949 0.0308482       0.165661
beta.co       0.1181  0.04715 0.0006087       0.003374
beta.pl       0.1112  0.04380 0.0005654       0.003133
omega       113.9745 59.18509 0.7640763       4.261394
sigma.co      3.9677  1.93954 0.0250394       0.138669
sigma.pl      3.9227  1.65836 0.0214093       0.120637
beta.co.pl    1.0651  0.11494 0.0014838       0.007644
alpha.co.pl  -2.2812  8.42431 0.1087574       0.558642
beta.pl.co    0.9499  0.10206 0.0013175       0.006786
alpha.pl.co   1.3135  7.73354 0.0998395       0.512659

2. Quantiles for each variable:
```

```
                 2.5%      25%     50%      75%    97.5%
alpha.co     64.63038 68.07376 70.0058  71.8003  74.1636
alpha.pl     62.86919 65.95268 68.0185  69.4978  71.7307
beta.co       0.04542  0.07647  0.1166   0.1524   0.2105
beta.pl       0.04188  0.07389  0.1100   0.1415   0.2007
omega        49.06948 71.79005 94.7646 146.2587 258.9672
sigma.co      0.05500  2.39208  4.5474   5.6247   6.4437
sigma.pl      0.64964  2.59258  4.2322   5.4084   6.1727
beta.co.pl    0.88047  0.96665  1.0531   1.1626   1.2778
alpha.co.pl -17.74983 -9.40183 -1.4494   4.8883  11.3003
beta.pl.co    0.78261  0.86013  0.9496   1.0345   1.1358
alpha.pl.co -12.82860 -5.05094  1.3750   8.0717  13.8788
```

```
> round( ci.lin( lm( co ~ pulse, data=oxw ) ), 3 )

            Estimate StdErr      z     P 2.5%  97.5%
(Intercept)    8.503  2.935  2.897 0.004 2.75 14.256
pulse          0.918  0.040 23.165 0.000 0.84  0.995
```

```
> round( ci.lin( lm( pulse ~ co, data=oxw ) ), 3 )

            Estimate StdErr      z P  2.5%  97.5%
(Intercept)   11.010  2.719  4.049 0 5.681 16.339
co             0.822  0.035 23.165 0 0.752  0.891
```

We see that the slope for converting from one method to another lies between the two regression slopes we get from ordinary linear regressions.

(f) We can check whether we have reasonable mixing of the chains for the parameters of interest by `xyplot` and `density` plot — we are not impressed!

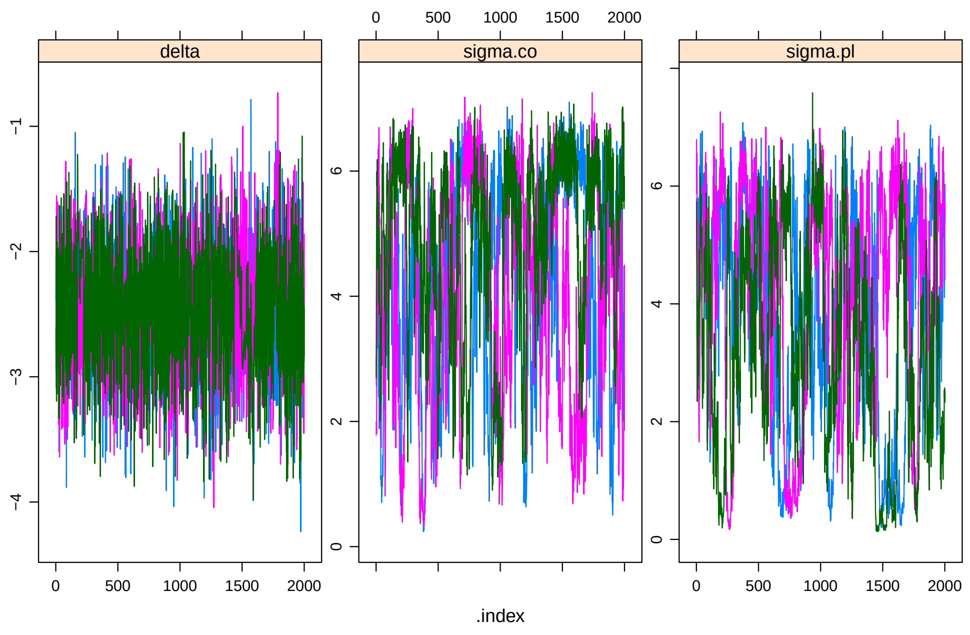
```
> wh <- c( grep( "sigma", varnames( m8.res ) ),
+          grep( "omega", varnames( m8.res ) ),
+          grep( "pl.co", varnames( m8.res ) ),
+          grep( "co.pl", varnames( m8.res ) ) )
> print(xyplot( m8.res[,wh], layout=c(4,2), aspect="fill", lwd=2 ))

> print( densityplot(m8.res[,wh],layout=c(4,2),lwd=2,aspect="fill") )
```

(g) Based on the posterior medians we would say that the relations ship between the methods were something like:

$$y_{\text{co}} = -0.50 + 1.04 y_{\text{pulse}}$$

which is shown in figure **??**

```
> with( oxw, plot( co ~ pulse, pch=16, xlim=c(20,100), ylim=c(20,100) ) )
> abline(0,1)
> abline( lm( co~pulse, data=oxw), col="red", lwd=2 )
> cf <- coef( lm( pulse ~ co, data=oxw) )
> abline( -cf[1]/cf[2], 1/cf[2], col="red", lwd=2 )
> qnt <- summary( m8.res )$quantiles
> qnt <- qnt[grep("co.pl",rownames(qnt)),"50%"]
> abline( qnt[2], qnt[1], col="blue", lwd=2 )
```



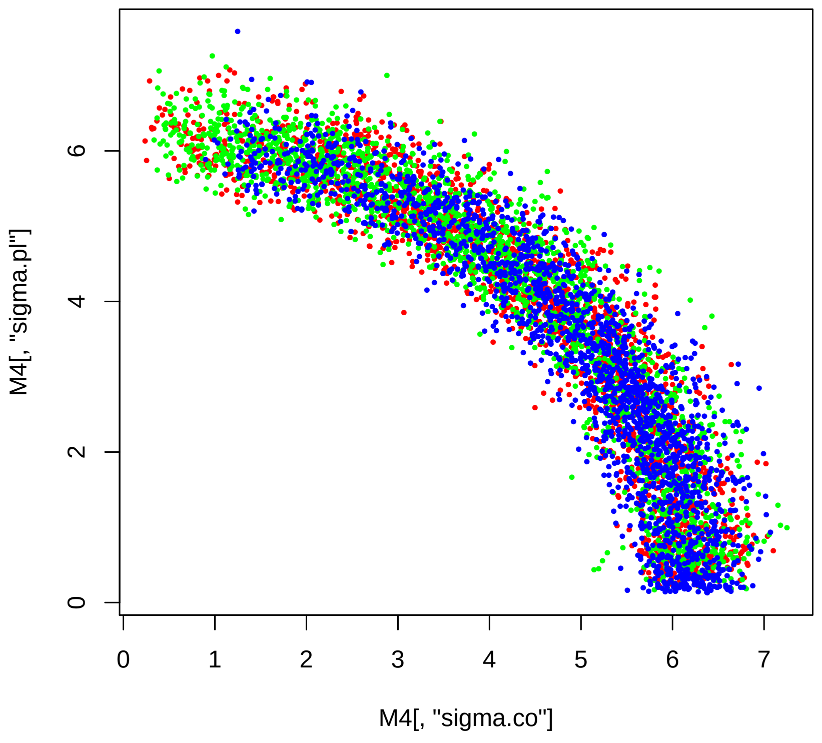


Figure 3.32: *Traces of the three chains for the three parameters of interest (top) and joint posterior distribution of the two variance components (bottom).*

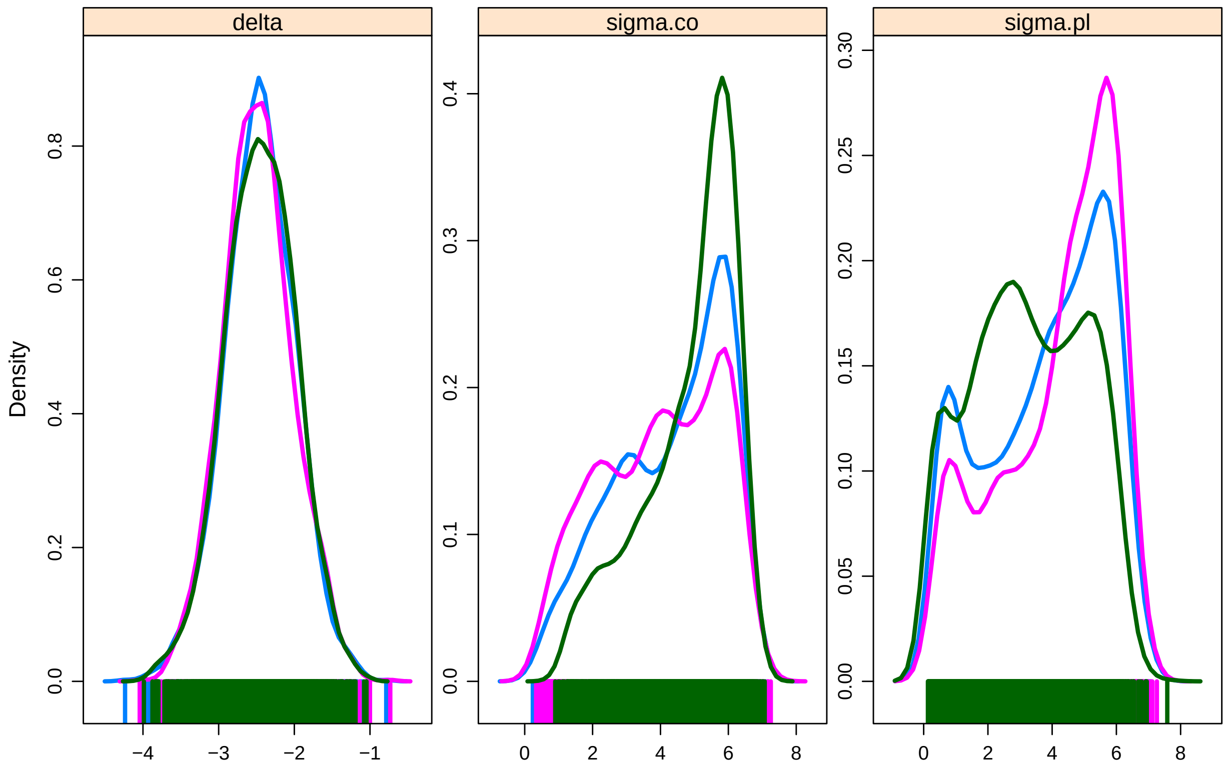


Figure 3.33: *Posterior densities for the overall difference between methods and the two residual standard deviations. Densities from each of the 3 chains.*

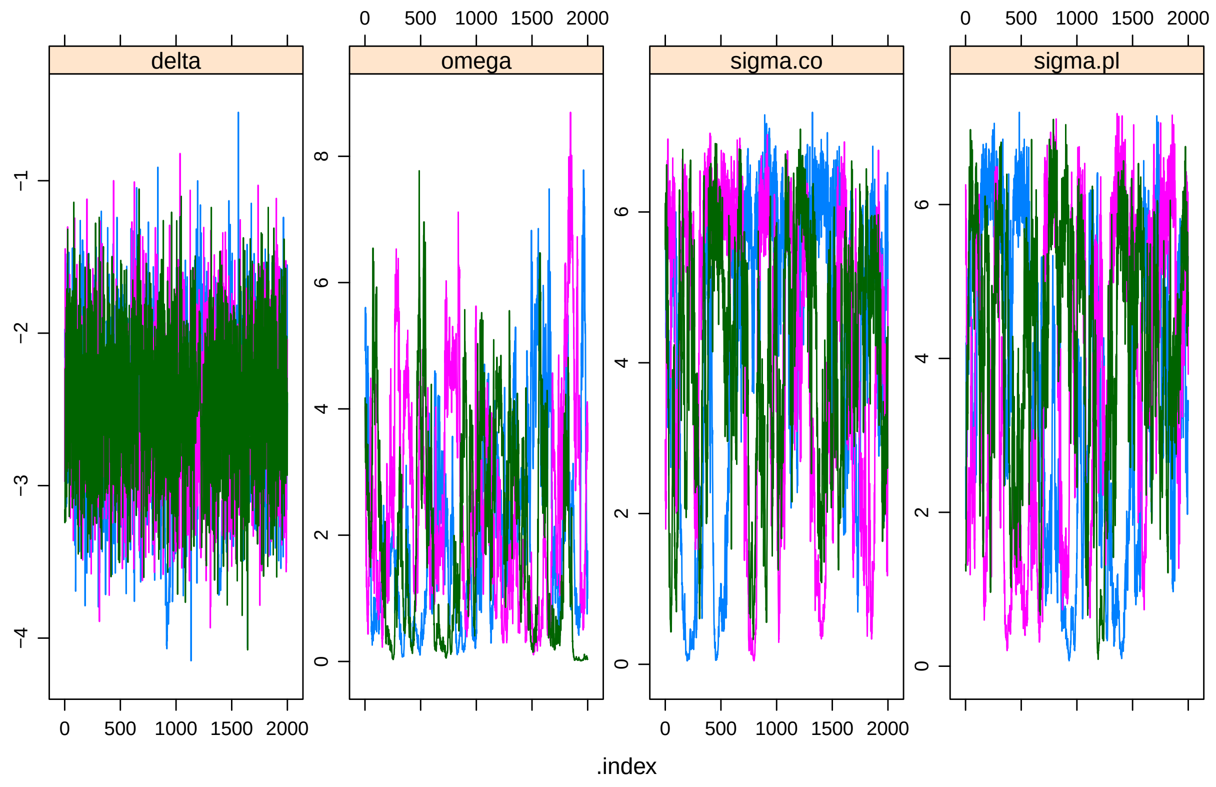


Figure 3.34: *Trace plot for the model with allowance for linked replicates.*

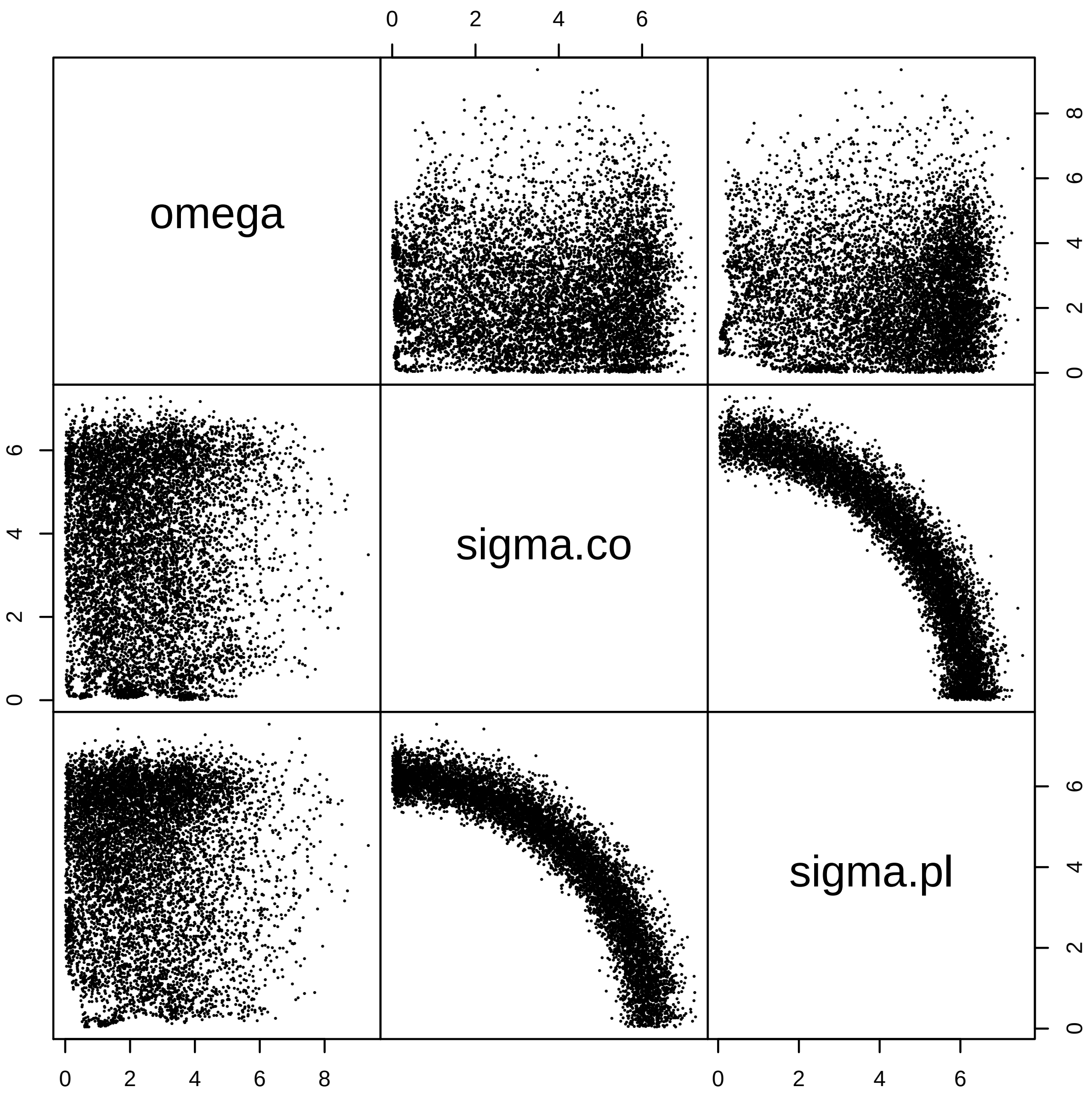


Figure 3.35: *Marginal 2-dimensional posteriors from the model with linked replicates.*

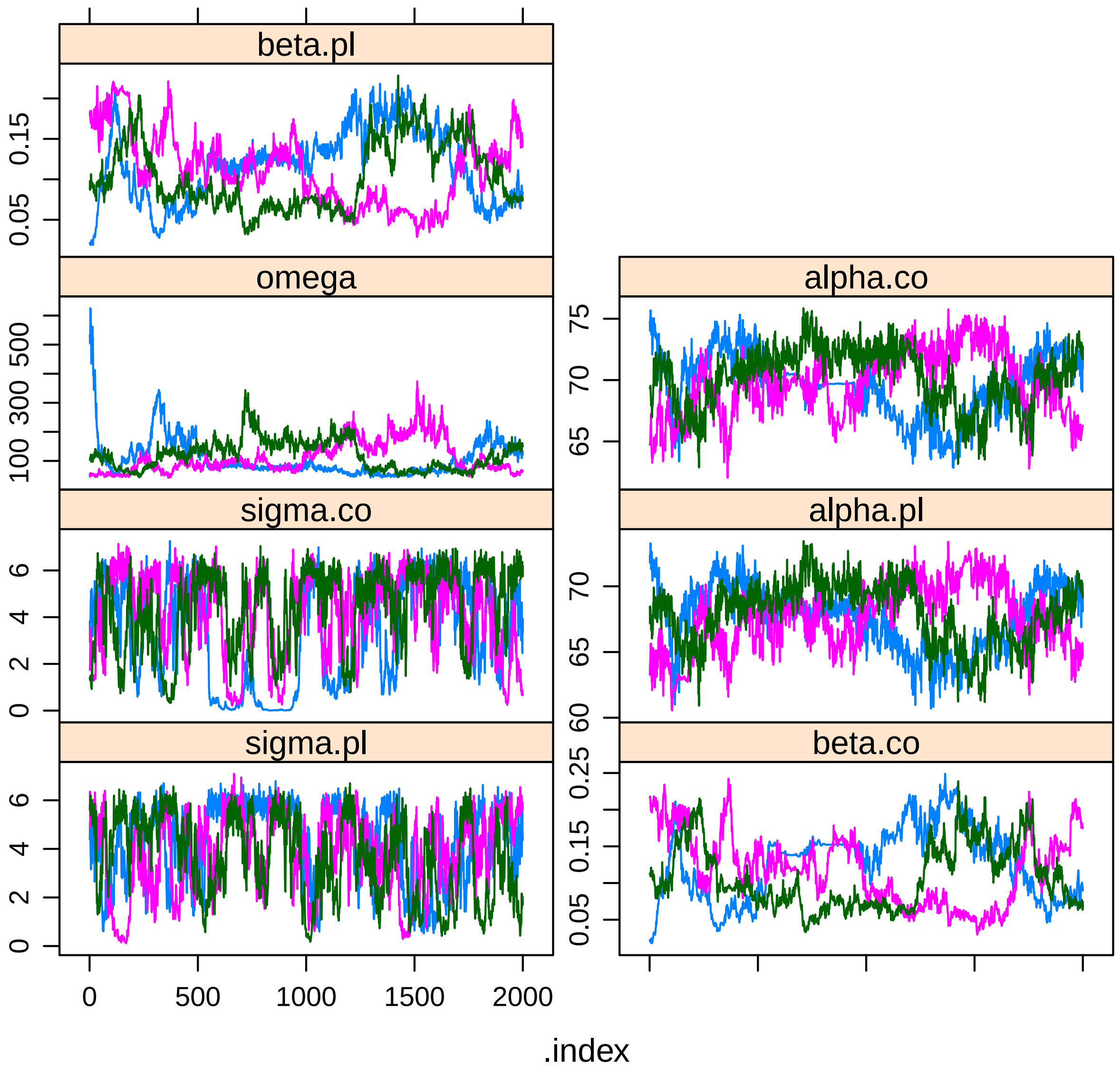
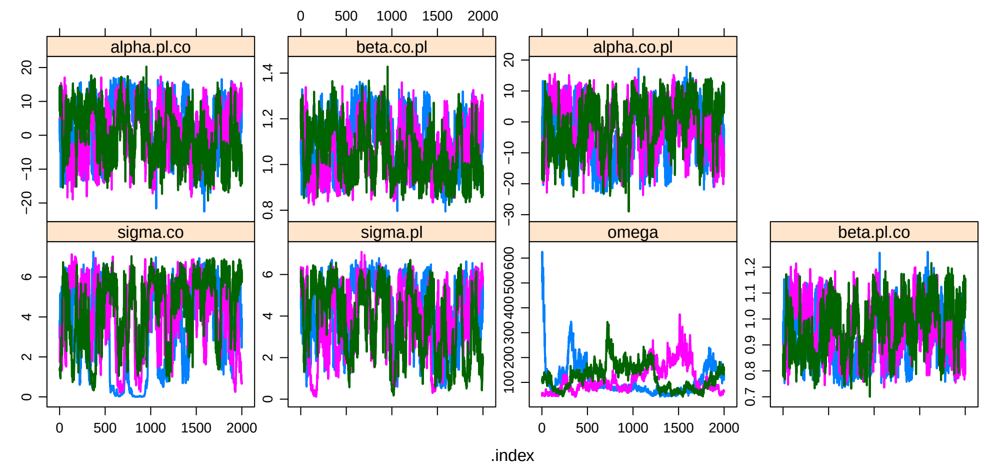


Figure 3.36: *Traces of parameters in the over-parametrized model.*



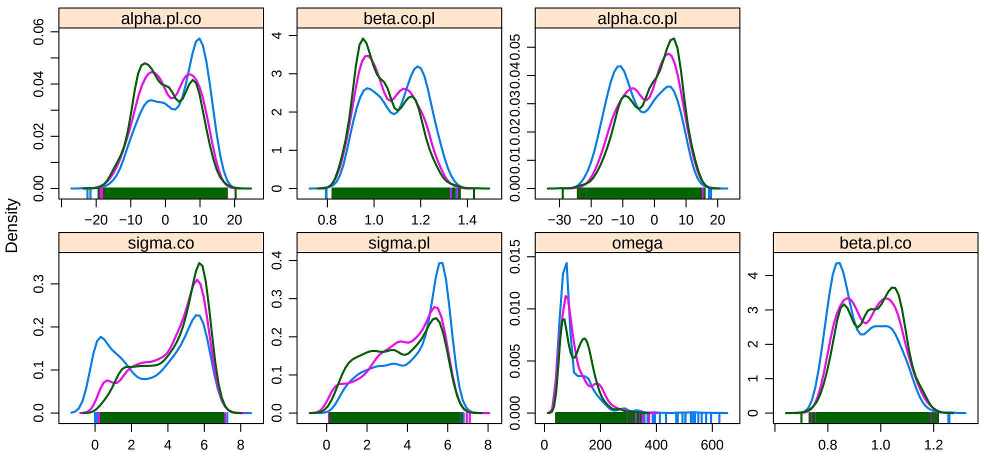


Figure 3.37: *Traces and densities of transformed parameters .*

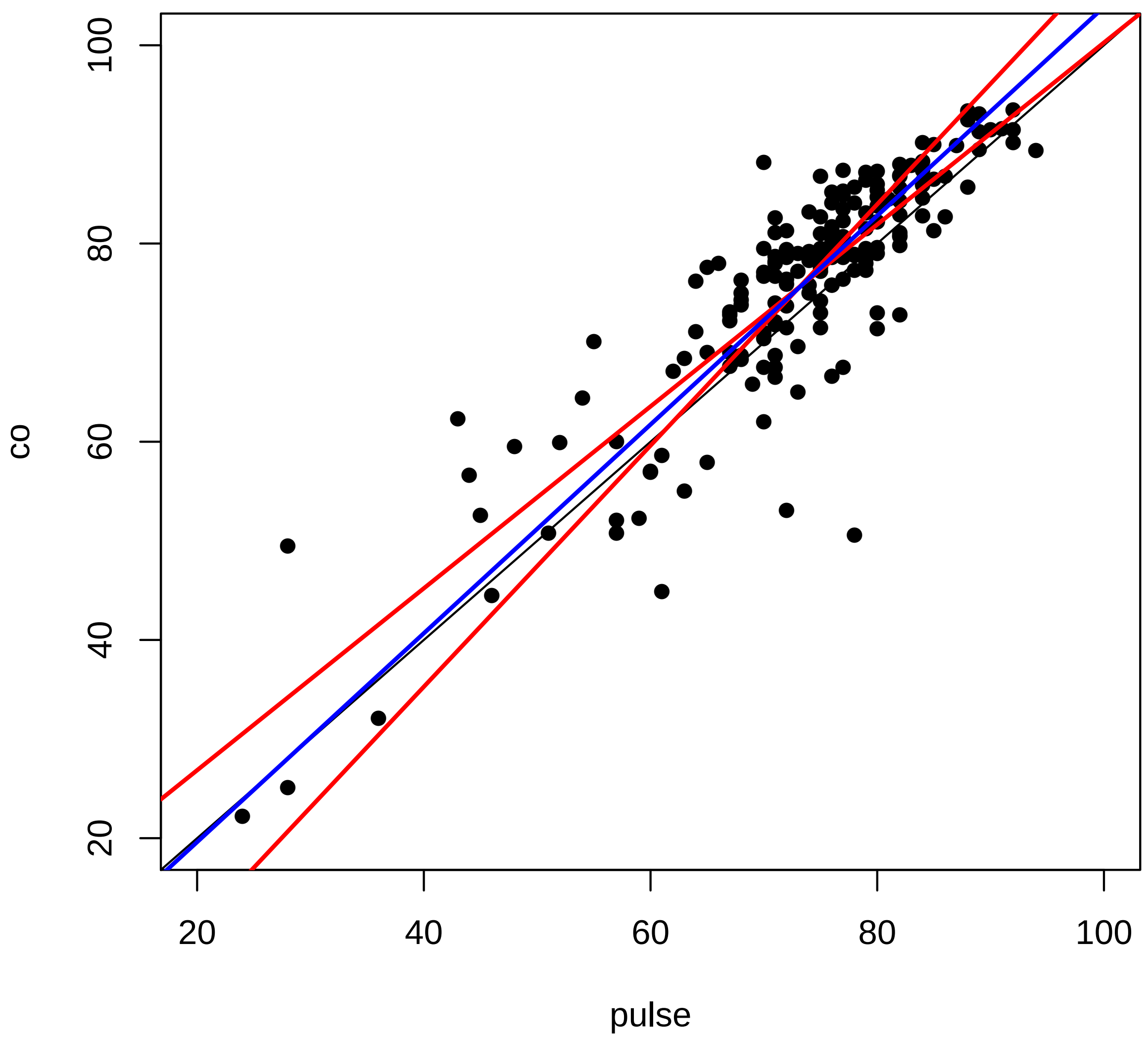


Figure 3.38: *Individual datapoints and traditional regression lines together with the line based on the posterior medians.*