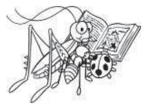
Chapter 19
Bayesian Inference Using Gibbs Sampling – BUGS Project



Beware: MCMC sampling can be dangerous!

– Disclaimer in WinBUGS User Manual

WHAT IS COVERED IN THIS CHAPTER

19.1 Introduction

BUGS is a freely available software for constructing and evaluating Bayesian statistical models using simulation approaches based on the Markov chain Monte Carlo methodology.

BUGS and WinBUGS are distributed freely and are the result of many years of development by a team of statisticians and programmers at the Medical Research Council Biostatistics Unit in Cambridge, UK (BUGS and WinBUGS), and by a team at the University of Helsinki, Finland (OpenBUGS); see the project pages `http://www.mrc-bsu.cam.ac.uk/software/bugs/` and `http://www.openbugs.net`.

Models are represented by a flexible language, and there is also a graphical feature, DOODLEBUGS, that allows users to specify their models as directed graphs. For complex models DOODLEBUGS can be very useful (Lunn et al., 2000). As of May 2017, the latest versions are WinBUGS 1.4.3 and OpenBUGS 3.2.3. A comprehensive overview of WinBUGS programming and applications can be found in Congdon (2005, 2006, 2010, 2014), Lunn et al. (2013), and Ntzoufras (2009).

19.2 Step-by-Step Session

We start this brief tutorial on WinBUGS with a simple regression example. Consider the model

$$\begin{aligned} y_i|\mu_i,\tau &\sim \mathcal{N}(\mu_i,\tau), \;\; i=1,\dots,n, \\ \mu_i &= \alpha+\beta(x_i-\overline{x}), \\ \alpha &\sim \mathcal{N}(0,10^{-4}), \\ \beta &\sim \mathcal{N}(0,10^{-4}), \\ \tau &\sim \mathcal{G}a(0.001,0.001). \end{aligned}$$

The normal distribution is parameterized by a *precision* parameter τ that is the reciprocal of the variance, $\tau = 1/\sigma^2$. Natural priors for precision parameters are gamma, and small values of the precision reflect the flatness (noninformativeness) of the priors. Assume that (x,y) pairs $(1,1)$, $(2,3)$, $(3,3)$, $(4,3)$, and $(5,5)$ are observed.

Estimators in classical, least-squares regression of y on $x-\overline{x}$ are given in the following MATLAB output:

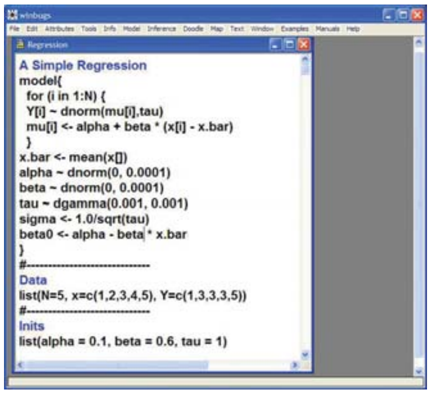
```
y  = [1 3 3 3 5]'; %response
xx = [1 2 3 4 5]';
X  = [ones(size(xx))  xx-mean(xx)];
[b.b,b.int,res.res,res.int,stats] = regress(y,X);

b.b'
%    3.0000   0.8000
stats
%    0.8000  12.0000   0.0405   0.5333
```

Thus, the estimators are $\hat{\alpha}=\overline{y}=3$, $\hat{\beta}=0.8$, and $\hat{\tau}=1/\hat{\sigma}^2=1/0.5333=1.875$.

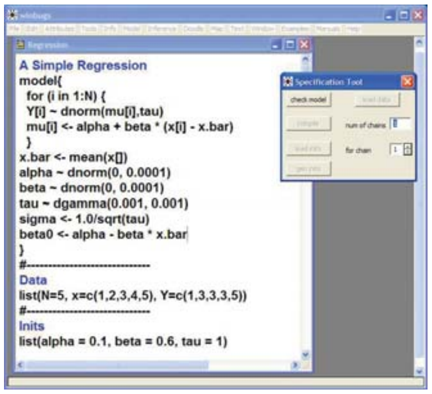
What about Bayesian estimators? We will find the estimators by MCMC simulation, as empirical means of the simulated posterior distributions. Assume that the initial parameter values are $\alpha_0=0.1$, $\beta_0=0.6$, and $\tau=1$. Start WinBUGS and input the following code in **[File > New]**:

```
# A simple regression
model{
```


winbugs
File Edit Attributes Tools Info Model Inference Doodle Map Text Window Examples Manuals Help
Regression

```
A Simple Regression
model{
  for (i in 1:N) {
  Y[i] ~ dnorm(mu[i],tau)
  mu[i] <- alpha + beta * (x[i] - x.bar)
  }
x.bar <- mean(x[])
alpha ~ dnorm(0, 0.0001)
beta ~ dnorm(0, 0.0001)
tau ~ dgamma(0.001, 0.001)
sigma <- 1.0/sqrt(tau)
beta0 <- alpha - beta * x.bar
}
#-----------------------------
Data
list(N=5, x=c(1,2,3,4,5), Y=c(1,3,3,3,5))
#-----------------------------
Inits
list(alpha = 0.1, beta = 0.6, tau = 1)
```

(a)



```
A Simple Regression
model{
  for (i in 1:N) {
  Y[i] ~ dnorm(mu[i],tau)
  mu[i] <- alpha + beta * (x[i] - x.bar)
  }
x.bar <- mean(x[])
alpha ~ dnorm(0, 0.0001)
beta ~ dnorm(0, 0.0001)
tau ~ dgamma(0.001, 0.001)
sigma <- 1.0/sqrt(tau)
beta0 <- alpha - beta * x.bar
}
#-----------------------------
Data
list(N=5, x=c(1,2,3,4,5), Y=c(1,3,3,3,5))
#-----------------------------
Inits
list(alpha = 0.1, beta = 0.6, tau = 1)
```

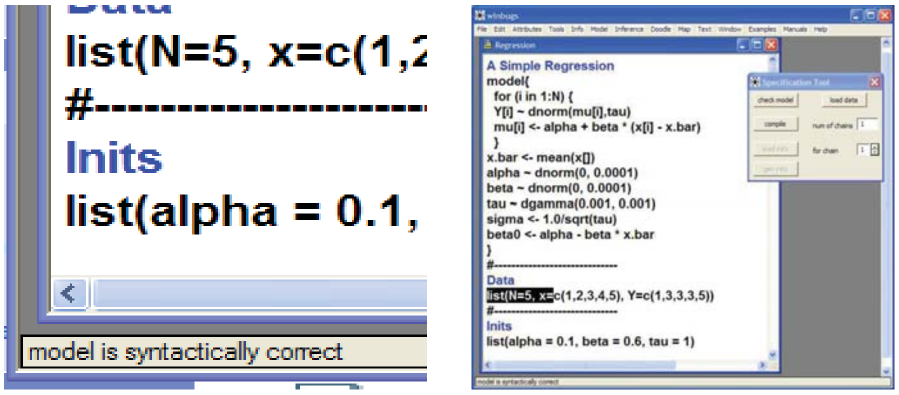
(b)

Fig. 19.1 (a) Opening WinBUGS front end with a simple regression task. The simple regression program is opened or typed in. (b) The front end after selecting **Specification** from the **Model** menu.

```
  for (i in 1:N) {
  Y[i] ~ dnorm(mu[i],tau)
  mu[i] <- alpha + beta * (x[i] - x.bar)
  }
x.bar <- mean(x[])
alpha ~ dnorm(0, 0.0001)
beta ~ dnorm(0, 0.0001)
tau ~ dgamma(0.001, 0.001)
sigma <- 1.0/sqrt(tau)
}
#-----------------------------
DATA
list(N=5, x=c(1,2,3,4,5), Y=c(1,3,3,3,5))
#-----------------------------
INITS
list(alpha = 0.1, beta = 0.6, tau = 1)
```

Next, make sure that the cursor is somewhere within the scope of "model," that is, somewhere between the first open and the last closed curly bracket. Go to the **Model** menu and open **Specification**. The **Specification Tool** window will pop out (Fig. 19.1b). Next, press **check model** in the Specification Tool window. If the model is correct, the response on the lower left border of the window should be: **model is syntactically correct** (Fig. 19.2a). Next, data are read in. Highlight the "list" statement in the data part of your code (Fig. 19.2b). In the Specification Tool window, select **load data**. If the data are in the correct format, you should receive a response in the lower left corner of the WinBUGS window: **data loaded** (Fig. 19.3a). You will need to compile your model in order to activate the **inits** buttons.

Select **compile** in the Specification Tool window. The response should be: **model compiled** (Fig. 19.3b), and the **load inits** and **gen inits** but-



```
list(N=5, x=c(1,2
#--------------------
Inits
list(alpha = 0.1,
```

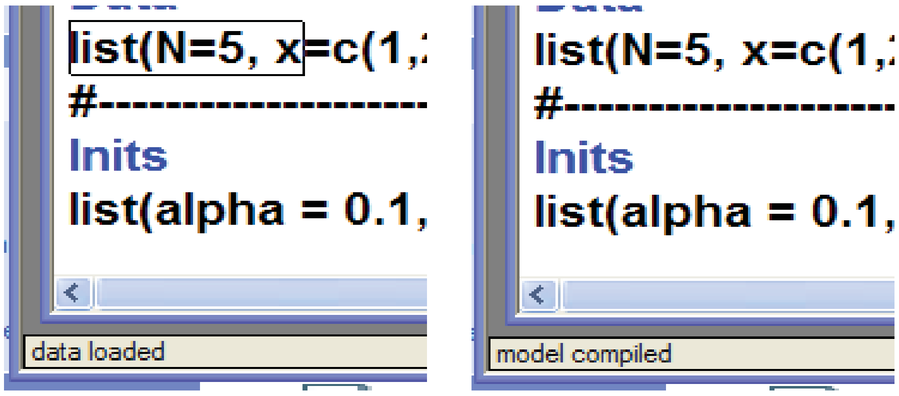
```
model is syntactically correct
```

```
A Simple Regression
model{
  for (i in 1:N) {
  Y[i] ~ dnorm(mu[i],tau)
  mu[i] <- alpha + beta * (x[i] - x.bar)
  }
x.bar <- mean(x[])
alpha ~ dnorm(0, 0.0001)
beta ~ dnorm(0, 0.0001)
tau ~ dgamma(0.001, 0.001)
sigma <- 1.0/sqrt(tau)
beta0 <- alpha - beta * x.bar
}
#----------------------------
Data
list(N=5, x=c(1,2,3,4,5), Y=c(1,3,3,3,5))
#----------------------------
Inits
list(alpha = 0.1, beta = 0.6, tau = 1)
```

```
model is syntactically correct
```

(a) (b)

Fig. 19.2 (a) After selecting **check model**, if the syntax is correct, the response is **model is syntactically correct.** (b) Highlighting the `list` in the data prior to reading data in.



```
list(N=5, x=c(1,2
#--------------------
Inits
list(alpha = 0.1,
```

```
data loaded
```

```
list(N=5, x=c(1,2
#--------------------
Inits
list(alpha = 0.1,
```

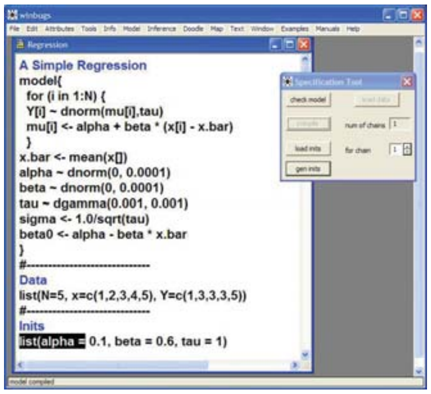
```
model compiled
```

(a) (b)

Fig. 19.3 WinBUGS' responses to (a) **load data** and (b) **compile** in the model specification tool.

tons become active. Finally, highlight the "list" statement in the initials part of your code, and in the Specification Tool window, select **load inits** (Fig. 19.4a). The response should be: **model is initialized** (Fig. 19.4b), and this completes the reading in of the model. If the response is **initial values loaded but this or another chain contains uninitialized variables**, click on the **gen inits** button. The response should be: **initial values generated, model initialized**.

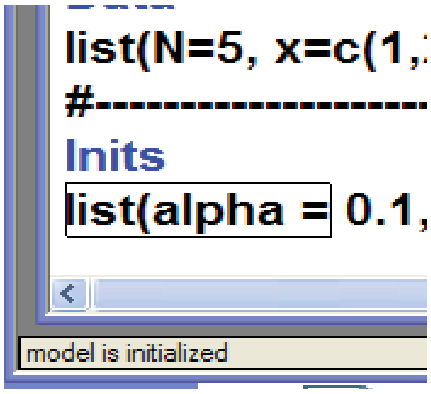
Now you are ready to burn in some simulations and at the same time check if the program works. Recall that burning in the Markov chain model is necessary for the chain to "forget" the initialized parameter values. In



```
A Simple Regression
model{
 for (i in 1:N) {
 Y[i] ~ dnorm(mu[i],tau)
 mu[i] <- alpha + beta * (x[i] - x.bar)
 }
x.bar <- mean(x[])
alpha ~ dnorm(0, 0.0001)
beta ~ dnorm(0, 0.0001)
tau ~ dgamma(0.001, 0.001)
sigma <- 1.0/sqrt(tau)
beta0 <- alpha - beta * x.bar
}
#---------------------------
Data
list(N=5, x=c(1,2,3,4,5), Y=c(1,3,3,3,5))
#---------------------------
Inits
list(alpha = 0.1, beta = 0.6, tau = 1)
```

```
model compiled
```

(a)



```
list(N=5, x=c(1,
#-------------------
Inits
list(alpha = 0.1,
```

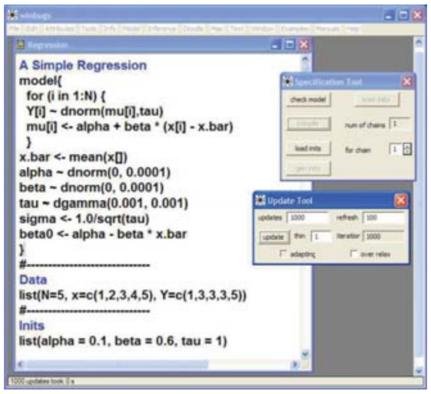
```
model is initialized
```

(b)

Fig. 19.4 (a) Highlighting the `list` to initialize the model. (b) WinBUGS confirms that the model (in fact a Markov chain) is initialized.

the **Model** menu, choose **Update...** and open **Update Tool** to check if your model updates (Fig. 19.5a).

From the **Inference** menu, open **Samples...**. A window titled **Sample Monitor Tool** will pop out (Fig. 19.5b). In the **node** subwindow, input the names of the variables you want to monitor. In this case, the variables are `alpha`, `beta`, and `tau`. If you correctly input the variable name, the **set** button becomes active and you should set the variable. Do this for all three variables of interest. In fact, `sigma` as a transformation of `tau` is available to be set as well.

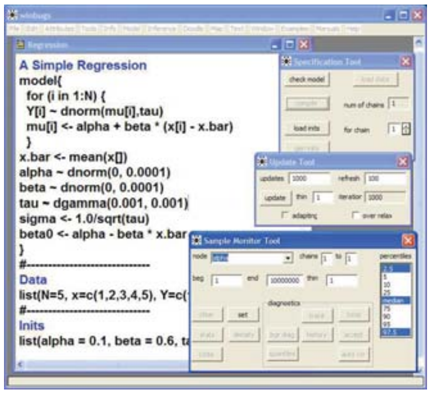


```
A Simple Regression
model{
 for (i in 1:N) {
 Y[i] ~ dnorm(mu[i],tau)
 mu[i] <- alpha + beta * (x[i] - x.bar)
 }
x.bar <- mean(x[])
alpha ~ dnorm(0, 0.0001)
beta ~ dnorm(0, 0.0001)
tau ~ dgamma(0.001, 0.001)
sigma <- 1.0/sqrt(tau)
beta0 <- alpha - beta * x.bar
}
#---------------------------
Data
list(N=5, x=c(1,2,3,4,5), Y=c(1,3,3,3,5))
#---------------------------
Inits
list(alpha = 0.1, beta = 0.6, tau = 1)
```

```
1000 updates took 0 s
```

Update Tool: updates 1000, refresh 100, thin 1, iterator 1000

(a)



```
A Simple Regression
model{
 for (i in 1:N) {
 Y[i] ~ dnorm(mu[i],tau)
 mu[i] <- alpha + beta * (x[i] - x.bar)
 }
x.bar <- mean(x[])
alpha ~ dnorm(0, 0.0001)
beta ~ dnorm(0, 0.0001)
tau ~ dgamma(0.001, 0.001)
sigma <- 1.0/sqrt(tau)
beta0 <- alpha - beta * x.bar
}
#---------------------------
Data
list(N=5, x=c(1,2,3,4,5), Y=c(
#---------------------------
Inits
list(alpha = 0.1, beta = 0.6, t
```

Update Tool: updates 1000, refresh 100, thin 1, iterator 1000

Sample Monitor Tool: node alpha, chains 1 to 1, beg 1, end 10000000, thin 1

(b)

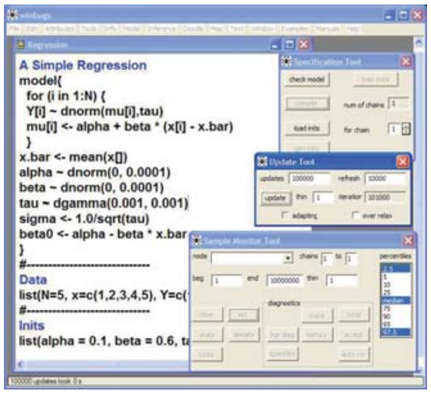
Fig. 19.5 WinBUGS' response to (a) **Update...** tool from the **Model** menu and (b) **Samples...** from the **Inference** menu.

Now choose `alpha` from the subwindow in **Sample Monitor Tool**. All of the buttons (**clear, set, trace, history, density, stats, coda, quantiles, bgr diag, auto cor**) are now active. Return to **Update Tool** and select the desired number of simulations, say 100,000, in the **updates** subwindow. Press the **update** button (Fig. 19.6a).

Return to **Sample Monitor Tool** and check **trace** for the part of the MC trace for α, **history** for the complete trace, **density** for a density estimator of α, etc. For example, pressing the **stats** button will produce something like the following table:

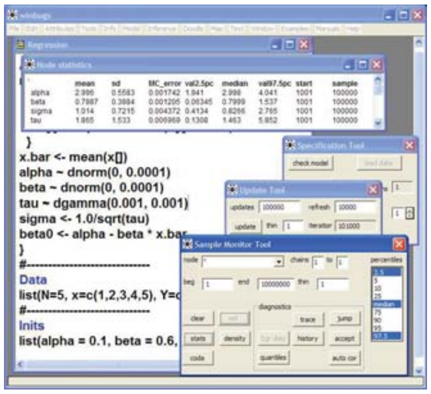
	mean	sd	MC error	val2.5pc	median	val97.5pc	start	sample
alpha	2.996	0.5583	0.001742	1.941	2.998	4.041	1001	100000

The mean 2.996 is the Bayes estimator, as the mean from the sample from the posterior for α. There are two precision outputs, `sd` and `MC error`. The former is an estimator of the standard deviation of the posterior and can be improved by increasing the sample size but not the number of simulations. The latter is the simulation error and can be improved by additional simulations. The 95% credible set (1.941, 4.041) is determined by `val2.5pc` and `val97.5pc`, which are the 0.025 and 0.975 (empirical) quantiles from the posterior. The empirical median of the posterior is given by `median`. The outputs `start` and `sample` show the starting index for the simulations (after burn-in) and the available number of simulations.


```
A Simple Regression
model{
  for (i in 1:N) {
   Y[i] ~ dnorm(mu[i],tau)
   mu[i] <- alpha + beta * (x[i] - x.bar)
  }
x.bar <- mean(x[])
alpha ~ dnorm(0, 0.0001)
beta ~ dnorm(0, 0.0001)
tau ~ dgamma(0.001, 0.001)
sigma <- 1.0/sqrt(tau)
beta0 <- alpha - beta * x.bar
}
#-----------------------------
Data
list(N=5, x=c(1,2,3,4,5), Y=c(
#-----------------------------
Inits
list(alpha = 0.1, beta = 0.6, t
```
Update Tool: updates 100000, refresh 10000, thin 1, iterator 101000

Sample Monitor Tool: beg 1, end 10000000, thin 1, chains 1 to 1

100000 updates took 0 s


```
        mean    sd      MC_error val2.5pc median  val97.5pc start  sample
alpha   2.996   0.5583  0.001742 1.941    2.998   4.041     1001   100000
beta    0.7987  0.3884  0.001205 0.06345  0.7999  1.537     1001   100000
sigma   1.014   0.7215  0.004372 0.4134   0.8266  2.765     1001   100000
tau     1.865   1.533   0.006969 0.1308   1.463   5.852     1001   100000
```
```
  }
x.bar <- mean(x[])
alpha ~ dnorm(0, 0.0001)
beta ~ dnorm(0, 0.0001)
tau ~ dgamma(0.001, 0.001)
sigma <- 1.0/sqrt(tau)
beta0 <- alpha - beta * x.bar
}
#-----------------------------
Data
list(N=5, x=c(1,2,3,4,5), Y=c
#-----------------------------
Inits
list(alpha = 0.1, beta = 0.6,
```
Update Tool: updates 100000, refresh 10000, thin 1, iterator 101000

Sample Monitor Tool: node *, beg 1, end 10000000, thin 1, chains 1 to 1

(a) (b)

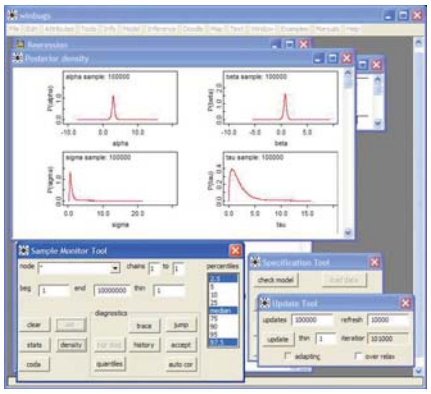
Fig. 19.6 (a) Select the simulation size and update. (b) After the simulation is done, check the `stats` node.

For all parameters a comparative table (Fig. 19.6b) is as follows:

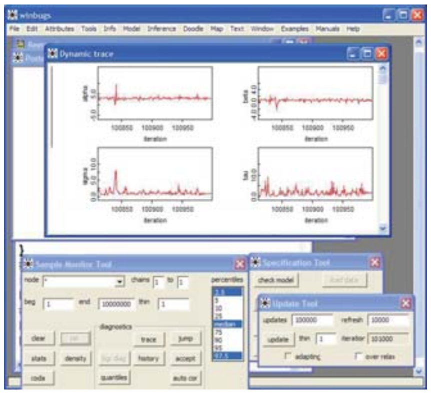
	mean	sd	MC error	val2.5pc	median	val97.5pc	start	sample
alpha	2.996	0.5583	0.001742	1.941	2.998	4.041	1001	100000
beta	0.7987	0.3884	0.001205	0.06345	0.7999	1.537	1001	100000
sigma	1.014	0.7215	0.004372	0.4134	0.8266	2.765	1001	100000
tau	1.865	1.533	0.006969	0.1308	1.463	5.852	1001	100000

We recall the least squares estimators from the beginning of this session: $\hat{\alpha} = 3$, $\hat{\beta} = 0.8$, and $\hat{\tau} = 1.875$, and note that their Bayesian counterparts are very close.

Densities (smoothed histograms) and traces for all parameters are given in Fig. 19.7.



(a)



(b)

Fig. 19.7 Checking (a) `density` and (b) `trace` in the **Sample Monitor Tool**.

If you want to save the trace for α in a file and process it in MATLAB, select **coda**, and the data window will open with an information window as well. Keep the data window active and select **Save As** from the **File** menu. Save the αs in `alphas.txt`, where it will be ready to be imported into MATLAB. Later in this chapter we will discuss the direct interface between WinBUGS and MATLAB called MATBUGS.

19.3 Built-in Functions and Common Distributions in WinBUGS

This section contains two tables: one with the list of built-in functions and another with the list of available distributions.

A first-time WinBUGS user may be disappointed by the selection of built-in functions – the set is minimal but sufficient. The full list of distributions in WinBUGS can be found in **Manuals**>**OpenBUGS User Manual**.

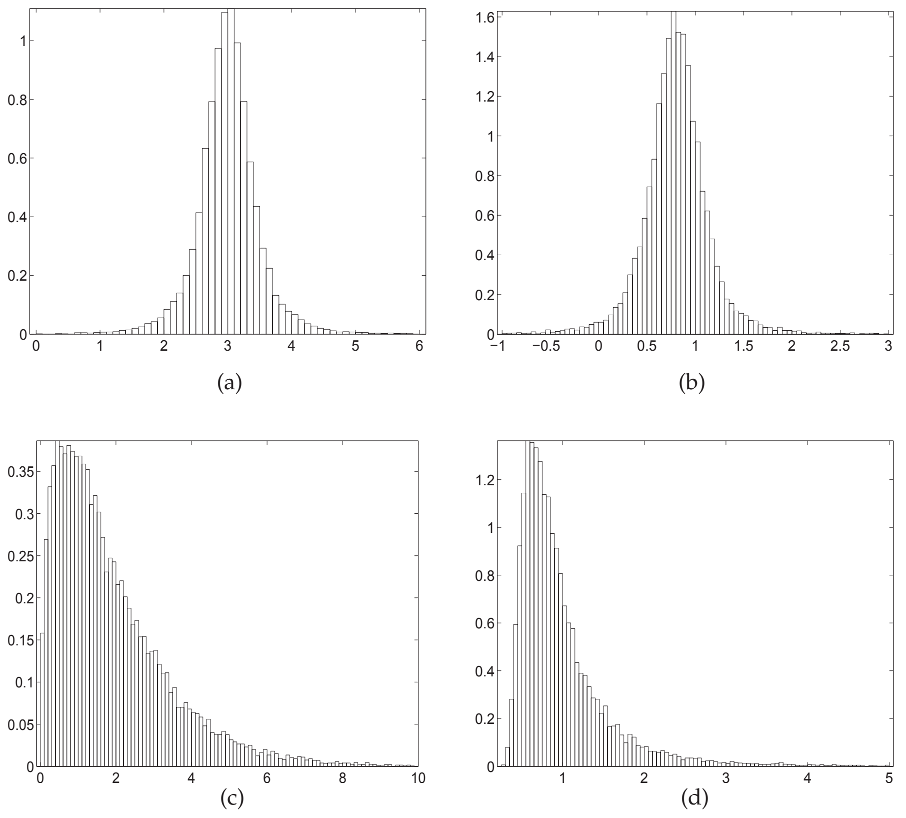


Fig. 19.8 Traces of the four parameters from a simple example: (a) α, (b) β, (c) τ, and (d) σ from WinBUGS. Data are plotted in MATLAB after being exported from WinBUGS.

WinBUGS also allows for the inclusion of distributions for which functions are not built in. Table 19.2 provides a list of important discrete and continuous distributions, with their syntax and parametrizations. WinBUGS has the capability to define custom distributions, both as a likelihood and as a prior, via the so-called zero-tricks (p. 353).

19.4 MATBUGS: A MATLAB Interface to WinBUGS

There is strong motivation to interface WinBUGS with MATLAB. Cutting and pasting results from WinBUGS is cumbersome if the simulation size is in millions or if the number of simulated parameters is large. Also, the data

Table 19.1 Built-in functions in WinBUGS

WinBUGS code	Function
`abs(y)`	$\lvert y\rvert$
`cloglog(y)`	$\ln(-\ln(1-y))$
`cos(y)`	$\cos(y)$
`equals(y, z)`	1 if $y=z$; 0 otherwise
`exp(y)`	$\exp(y)$
`inprod(y, z)`	$\sum_i y_i z_i$
`inverse(y)`	y^{-1} for symmetric positive–definite matrix y
`log(y)`	$\ln(y)$
`logfact(y)`	$\ln(y!)$
`loggam(y)`	$\ln(\Gamma(y))$
`logit(y)`	$\ln(y/(1-y))$
`max(y, z)`	y if $y>z$; y otherwise
`mean(y)`	$n^{-1}\sum_i y_i,\ \ n=dim(y)$
`min(y, z)`	y if $y<z$; z otherwise
`phi(y)`	standard normal CDF $\Phi(y)$
`pow(y, z)`	y^z
`sin(y)`	$\sin(y)$
`sqrt(y)`	$\sqrt{y}$
`rank(v, s)`	number of components of v less than or equal to v_s
`ranked(v, s)`	sth smallest component of v
`round(y)`	nearest integer to y
`sd(v)`	standard deviation of components of y ($n-1$ in denom.)
`step(y)`	1 if $y\geq 0$; 0 otherwise
`sum(y)`	$\sum_i y_i$
`trunc(y)`	greatest integer less than or equal to y

manipulation and graphical capabilities in WinBUGS are quite rudimentary compared to MATLAB.

MATBUGS is a MATLAB program that communicates with WinBUGS. The program `matbugs.m` was written by Kevin Murphy and his team and can be found at: `http://code.google.com/p/matbugs`.

We now demonstrate how to solve Jeremy's IQ problem in MATLAB by calling WinBUGS. First we need to create a simple text file, say, `jeremy.txt`:

```
model{
for(i in 1 : N)
    {
        scores[i] ~ dnorm(theta, tau)
     }
 theta ~ dnorm(mu, xi)
```

and then run the MATLAB file:

```
dataStruct = struct( ...
    'N', 5, ...
    'tau',1/80,...
    'xi',1/120,...
    'mu',110,...
    'scores',[97 110 117 102 98]);
```

Table 19.2 Some important built-in distributions with WinBUGS names and their parameterizations.

Distribution	WinBUGS code	Density
Bernoulli	`x ~ dbern(p)`	$p^x(1-p)^{1-x}, \;\; x=0,1;\, 0 \le p \le 1$
Binomial	`x ~ dbin(p, n)`	$\binom{n}{x} p^x(1-p)^{n-x}, \;\; x=0,\ldots,n;\, 0 \le p \le 1$
Categorical	`x ~ dcat(p[])`	$p[x], \;\; x=1,2,\ldots,\dim(p)$
Negative Binomial	`x ~ dnegbin(p, r)`	$\frac{(x+r-1)!}{x!(r-1)!} p^r(1-p)^x, \;\; x=0,1,\ldots;\, 0 \le p \le 1$
Poisson	`x ~ dpois(lambda)`	$\frac{\lambda^x}{x!}\exp\{-\lambda\}, \;\; x=0,1,2,\ldots;\, \lambda > 0$
Beta	`x ~ dbeta(a,b)`	$\frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1}, \;\; 0 \le x \le 1,\, a,b > -1$
Chi-square	`x ~ dchisqr(k)`	$\frac{x^{k/1-1}\exp\{-x/2\}}{2^{k/2}\Gamma(k/2)}, \;\; x \ge 0,\, k > 0$
Double exponential	`x ~ ddexp(mu, tau)`	$\frac{\tau}{2}\exp\{-\tau\lvert x-\mu\rvert\}, \;\; x \in R,\, \tau > 0,\, \mu \in R$
Exponential	`x ~ dexp(lambda)`	$\lambda\exp\{-\lambda x\}, \;\; x \ge 0,\, \lambda \ge 0$
Flat	`x ~ dflat()`	constant; not a proper density
Gamma	`x ~ dgamma(a, b)`	$\frac{b^a x^{a-1}}{\Gamma(a)}\exp(-bx), \;\; x,a,b > 0$
Normal	`x ~ dnorm(mu, tau)`	$\sqrt{\tau/(2\pi)}\exp\{-\frac{\tau}{2}(x-\mu)^2\}, \;\; x,\mu \in R,\, \tau > 0$
Pareto	`x ~ dpar)alpha,c)`	$\alpha c^\alpha x^{-(\alpha+1)}, \;\; x > c$
t	`x ~ dt(mu, tau, k)`	$\frac{\Gamma((k+1)/2)}{\Gamma(k/2)}\sqrt{\frac{\tau}{k\pi}}[1+\frac{\tau}{k}(x-\mu)^2]^{-(k+1)/2}, \;\; x \in \mathbb{R},\, k \ge 2$
Uniform	`x ~ dunif(a, b)`	$\frac{1}{b-a}, \;\; a \le x \le b$
Weibull	`x ~ dweib(v, lambda)`	$v\lambda x^{v-1}\exp\{-\lambda x^v\}, \;\; x,v,\lambda > 0,$
Multinomial	`x[] ~ dmulti(p[], N)`	$\frac{(\sum_i x_i)!}{\prod_i x_i!}\prod_i p_i^{x_i}, \;\; \sum_i x_i = N,\, 0 < p_i < 1,\, \sum_i p_i = 1$
Dirichlet	`p[] ~ ddirch(alpha[])`	$\frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)}\prod_i p_i^{\alpha_i-1}, \;\; 0 < p_i < 1,\, \sum_i p_i = 1$
Multivariate normal	`x[] ~ dmnorm(mu[], T[,])`	$(2\pi)^{-d/2}\lvert T\rvert^{1/2}\exp\{-1/2(x-\mu)'T(x-\mu)\}, \;\; x \in R^d$
Multivariate t	`x[] ~ dmt(mu[], T[,], k)`	$\frac{\Gamma((k+d)/2)}{\Gamma(k/2)}\frac{\lvert T\rvert^{1/2}}{k^{d/2}\pi^{d/2}}\left[1+\frac{1}{k}(x-\mu)'T(x-\mu)\right]^{-(k+d)/2}, \;\; x \in \mathbb{R}^d, k \ge 2$
Wishart	`x[,] ~ dwish(R[,], k)`	$\lvert R\rvert^{k/2}\lvert x\rvert^{(k-p-1)/2}\exp\{-1/2Tr(Rx)\}, \;\; x$ p.d.; $k > p-1$

```
 initStruct = struct( ...
    'theta', 100 );

cd('C:\MyBugs\matbugs\')
[samples, stats] = matbugs(dataStruct, ...
fullfile(pwd, 'jeremy.txt'), ...
'init', initStruct, ...
        'nChains', 1, ...
'view', 0, ...
        'nburnin', 2000, ...
        'nsamples', 50000, ...
'thin', 1, ...
'monitorParams', {'theta'}, ...
        'Bugdir', 'C:/Program Files/BUGS');

baymean = mean(samples.theta)
frmean=mean(dataStruct.scores)

  figure(1)
  [p, x] = ksdensity(samples.theta);
  plot(x, p);
```

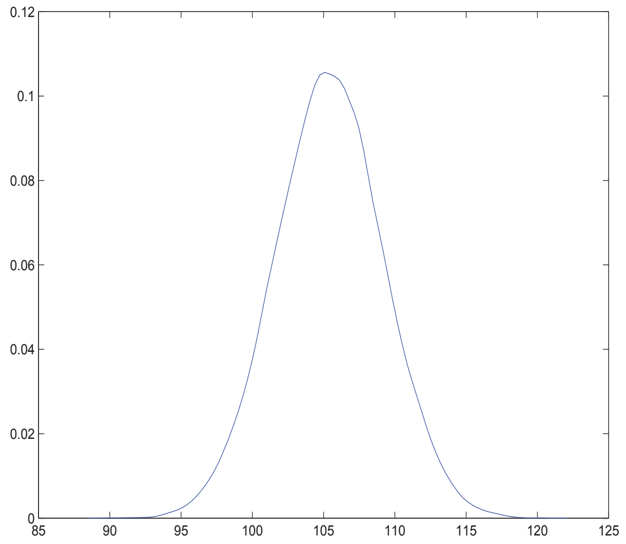


Fig. 19.9 Posterior for Jeremy's data set. Data are plotted in MATLAB after being exported from WinBUGS by MATBUGS.

19.5 Exercises

19.1. **A Coin and a Die.** The following WinBUGS code simulates flips of a coin. The outcome **H** is coded by 1 and **T** by 0. Mimic this code to simulate rolls of a fair die.

```
#coin
model{
flip ~ dcat(p.coin[])
coin <- flip - 1
}
DATA
list(p.coin=c(0.5, 0.5))
#just generate initials
```

19.2. **De Mere Paradox in WinBUGS.** In Exercise 3.6 (b) we examined de Mere's paradox: In playing a game with three fair dice, the sum 11 was advantageous to the sum 12.
(a) Using WinBUGS/OpenBUGS demonstrate that, in playing a game with 300 fair dice, the sum 1111 is advantageous to the sum 1112.
(b) Which of the two sums from (a) is more advantageous if the 300 dice are loaded, with probabilities 0.15, 0.15, 0.16, 0.2, 0.17, and 0.17, for sides 1, . . . , 6, respectively.
Hint:

```
 part of the code...
for (i in 1:300) {
dice[i] ~ dcat(p.dice[]);
}
is1111 <- equals(sum(dice[]),1111)
is1112 <- equals(sum(dice[]),1112)
```

19.3. **Simulating the Probability of an Interval.** Consider an exponentially distributed random variable X, $X \sim \mathcal{E}\left(\frac{1}{10}\right)$, with density $f(x) = \frac{1}{10}\exp\{-x/10\}$, $x > 0$. Compute $\mathbb{P}(10 < X < 16)$ using (a) exact integration, (b) MATLAB's `expcdf`, and (c) WinBUGS.

19.4. **WinBUGS as a Calculator.** WinBUGS can approximate definite integrals, solve nonlinear equations, and even find values of definite integrals over random intervals. The following WinBUGS program finds an approximation to $\int_0^\pi \sin(x)dx$, solves the equation $y^5 - 2y = 0$, and finds the integral $\int_0^R z^3(1-z^4)dz$, where R is a beta $\mathcal{B}e(2,2)$ random variable. Verify the following code and find the solution:

```
model{
F(x) <- sin(x)
int <- integral(F(x), 0, pi, 1.0E-6)
pi<- 3.141592659
y0 <- solution(F(y), 1,2, 1.0E-6)
F(y)  <- pow(y,5) - 2*y
zero <-  pow(y0, 5)-2*y0

randint <- integral(F(z), 0, randbound, 1.0E-6)
F(z) <- pow(z,3)*(1-pow(z,4))
randbound ~ dbeta(2,2)
}

NO DATA

INITS
    list(x=1, y=0, z=NA, randbound=0.5)
```

After model checking, one should go directly to compiling (no data to load in) and initializing the model. There is NO need to update the model, to go to the Inference tool, to set the variables for monitoring or to sample. One simply goes to the **Info** menu and checks **Node Info**. In the Node Info tool one specifies `int` for the approximation of an integral, `y0` for the solution of an equation, `zero` for checking that `y0` satisfies the equation (approximately), and `randint` for the value of a random interval.

MATLAB AND WINBUGS FILES AND DATA SETS USED IN THIS CHAPTER

http://statbook.gatech.edu/Ch19.WinBUGS/

simple.m

DeMere.odc, jeremy.odc, picktrick.odc, Regression1.odc, Regression2.odc, simulationd.odc

alpha.txt, beta.txt, sigma.txt, tau.txt

CHAPTER REFERENCES

Congdon, P. (2005). *Bayesian Models for Categorical Data*. Wiley, Hoboken, NJ.

Congdon, P. (2006). *Bayesian Statistical Modelling*, Second Edition. Wiley, Hoboken, NJ.

Congdon, P. (2010). *Hierarchical Bayesian Modelling*. Chapman & Hall/CRC, Boca Raton, FL.

Congdon, P. (2014). *Applied Bayesian Modelling*, Second Edition. Wiley, Hoboken, NJ.

Lunn, D., Jackson, C., Best, N., Thomas, A., and Spiegelhalter, D. (2013). *The BUGS Book*, CRC Press, Boca Raton, FL.

Lunn, D. J., Thomas, A., Best, N., and Spiegelhalter, D. (2000). WinBUGS – a Bayesian modelling framework: concepts, structure, and extensibility. *Stat. Comput.*, **10**, 325–337.

Ntzoufras, I. (2009). *Bayesian Modeling Using WinBUGS*. Wiley, Hoboken.