Introduction to (classic) BUGS

- BUGS is a program to carry out Gibbs Sampling in complex models.
- You may download classic BUGS from
  http://www.mrc-bsu.cam.ac.uk/bugs/classic/bugs06/readme.shtml
- Classic BUGS runs from UNIX and DOS OS.
- There is a more modern version known as WINBUGS which you may download from
  http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/contents.shtml
- The examples will be based on classic BUGS but can be easily implemented into WINBUGS.
- The main advantages of WINBUGS over classic BUGS are the user/program interface and the possibility of using additional packages (GeoBUGS or PKBUGS).
- CODA is a companion package to BUGS/WINBUGS available at
  http://www.mrc-bsu.cam.ac.uk/bugs/classic/coda04/readme.shtml
  which is very similar to BOA.

- After installing the software, simply type *bugs* at your UNIX or DOS prompt.

```
Welcome to BUGS on 14 th Apr 2004  at 17:46:6
BUGS : Copyright (c) 1992 .. 1995 MRC Biostatistics Unit.
All rights reserved.
Version 0.600 for unix systems.
For general release: please see documentation for disclaimer.
The support of the Economic and Social Research Council (UK)
is gratefully acknowledged.
Bugs>
```

- **Surgical data example**. This example, considers mortality rates in 12 hospitals performing cardiac surgery in babies. The data are shown below.

	Hospital											
	A	B	C	D	E	F	G	H	I	J	K	L
No. of ops. n	47	148	119	810	211	196	148	215	207	97	256	360
No. of deaths r	0	18	8	46	8	13	9	31	14	8	29	24

- BUGS assumes that we have a Bayesian model, where all the quantities are treated as random variables.
- A model consists of joint probability distribution over all unobserved quantities (parameters and missing observations) given some data.
- BUGS produces MCMC output under a given model.
- For the surgical data example a possible model is:

  The number of deaths r_i for hospital i may be modelled as a binary response variable with 'true' failure probability p_i:

$$r_i \sim Binomial(p_i, n_i); i = 1, 2, \ldots, 12$$

  Rather than assuming complete independence, we suppose that the failure rates across hospitals are similar in some way.

$$logit(p_i) = b_i; \quad b_i \sim Normal(\mu, \tau)$$

- Here $\tau = 1/\sigma^2$ is a precision parameter.
- To complete the model a non-informative prior is used:

$$\mu \sim Normal(0, .000001); \tau \sim Gamma(0.001, 0.001)$$

- The user needs to create a text file with extension *.bug* with the model description
- For the surgical example (stored in surgical.bug) we have,

```
model surgical;
const
 N = 12;    # number of hospitals
var
 r[N],     # number of deaths
 n[N],     # total number of operations
 p[N],     # 'true' probability of death
 mu,     # population mean failure rate (on logit scale)
 tau;     # precision of the random effects

data r, n in "surgical.dat";
inits in "surgical.in";

{
  for (i in 1:N) {
    r[i] ~ dbin(p[i], n[i]);
    logit(p[i]) <- b[i];
    b[i] ~ dnorm(mu, tau);
                }

  # Priors:
     mu ~ dnorm(0.0,1.0E-6);
     tau ~ dgamma(1.0E-3, 1.0E-3);
 }
```

- To compile the file,

```
Bugs>compile("surgical.bug")
```

- Every BUGS run must start with compile()
- Loads model in file surgical.bug
- Prepares model for running MCMC simulations.
- BUGS checks for errors.

```
Parsing model declarations.
Loading data value file(s).
Loading initial value file(s).
Parsing model specification.
Checking model graph for directed cycles.
Generating code.
Generating sampling distributions.
Generating initial values
Checking model specification.
Choosing update methods.
compilation took  00:00:00

Bugs>update(500) time for    500    updates was  00:00:03
```

- Causes BUGS to carry out 500 iterations of the Gibbs Sampling.
- The simulated values are not stored at this stage.

```
Bugs>monitor(p)
Bugs>update(1000)
```

- Tells BUGS to store the simulated values for p.
- BUGS carries out an additional 1000 iterations.

```
Bugs>diag(p)
```

```
       mean       sd       mean       sd        Z       sample
[1]   5.61E-2   1.38E-4   5.65E-2   7.29E-5  -1.51E-1     1000
[2]   1.01E-1   1.33E-4   1.01E-1   6.22E-5   6.69E-2     1000
[3]   7.33E-2   4.40E-5   7.22E-2   1.88E-5   7.05E-1     1000
[4]   6.04E-2   1.16E-5   5.99E-2   1.03E-5   4.66E-1     1000
[5]   5.35E-2   5.19E-5   5.27E-2   3.20E-5   4.46E-1     1000
[6]   7.03E-2   3.92E-5   6.98E-2   1.30E-5   3.33E-1     1000
[7]   6.94E-2   2.79E-5   6.88E-2   1.97E-5   4.92E-1     1000
[8]   1.19E-1   1.65E-4   1.19E-1   8.19E-5  -1.06E-1     1000
[9]   7.06E-2   2.44E-5   7.04E-2   1.45E-5   1.14E-1     1000
[10]  7.85E-2   4.18E-5   7.76E-2   2.39E-5   5.95E-1     1000
[11]  1.00E-1   5.46E-5   1.00E-1   2.63E-5  -2.66E-1     1000
[12]  6.95E-2   2.98E-5   6.90E-2   1.29E-5   4.43E-1     1000
Bugs>stats(p)

       mean        sd       2.5% :  97.5%  CI    median       sample
[1]   5.554E-2   1.918E-2   2.073E-2   9.381E-2   5.529E-2     1000
[2]   1.018E-1   2.271E-2   6.593E-2   1.511E-1   9.890E-2     1000
[3]   7.133E-2   1.745E-2   4.124E-2   1.106E-1   7.004E-2     1000
[4]   5.965E-2   8.337E-3   4.354E-2   7.644E-2   5.941E-2     1000
[5]   5.304E-2   1.339E-2   2.798E-2   8.011E-2   5.227E-2     1000
[6]   6.982E-2   1.440E-2   4.149E-2   9.822E-2   6.968E-2     1000
[7]   6.794E-2   1.619E-2   3.820E-2   1.035E-1   6.668E-2     1000
[8]   1.208E-1   2.342E-2   7.908E-2   1.689E-1   1.196E-1     1000
[9]   7.041E-2   1.401E-2   4.617E-2   1.011E-1   6.938E-2     1000
[10]  7.839E-2   1.905E-2   4.638E-2   1.227E-1   7.664E-2     1000
[11]  1.010E-1   1.780E-2   7.051E-2   1.405E-1   1.001E-1     1000
[12]  6.929E-2   1.216E-2   4.793E-2   9.338E-2   6.877E-2     1000
Bugs>q()
```

- After quitting BUGS, all the monitored samples will be written to a file called *bugs.out*. An index file *bugs.ind* is also created.

This is one part of bugs.out

```
501          4.21927E-2
502          7.98577E-2
503          5.12284E-2
504          5.13588E-2
505          1.86455E-2
506          5.72273E-2
507          6.07437E-2
508          6.93815E-2
509          6.75286E-2
510          4.51312E-2
511          4.87130E-2
512          3.91395E-2
513          4.55509E-2
```

Here is bugs.ind

```
p[1]       1              1000
p[2]       1001           2000
p[3]       2001           3000
p[4]       3001           4000
p[5]       4001           5000
p[6]       5001           6000
p[7]       6001           7000
p[8]       7001           8000
p[9]       8001           9000
p[10]       9001           10000
p[11]       10001          11000
p[12]       11001          12000
```

- These files may be read into CODA or BOA to carry out convergence diagnostics.
- Also a file *bugs1.out* is created, and contains the output of the stats command.
- The file that contains the data *surgical.dat* looks like this:

```
 0  47
18 148
 8 119
46 810
 8 211
13 196
 9 148
31 215
14 207
 8  97
29 256
24 360
```

- The initial value file *surgical.in* an contains:

list(tau = 1, mu = 0)

Example 2 *Rats:* Normal hierarchical model with missing data.

- This example concerns 30 rats whose weights were measured weekly for five weeks.
- Y_{ij} is the weight of the ith rat measured at age x_j.
- The proposed model is:

$$Y_{ij} \sim N(\alpha_i + \beta_i(x_j - \bar{x}), \tau_c)$$

$$\alpha_i \sim N(\alpha_c, \tau_\alpha)$$

$$\beta_i \sim N(\beta_c, \tau_\beta); i = 1, \ldots, 30, j = 1, \ldots, 5$$

- The parameters α_c, τ_α, β_c, τ_β and τ_c are given independent "non-informative" priors.
- Particular interest focusses on the intercept at time 0 $\alpha_0 = \alpha_c - \beta_c\bar{x}$.

Model specification in BUGS

```
model rats;
 const
   N = 30,  # number of rats
   T = 5;   # number of time points
   var
   tau.c, alpha0, alpha.c, beta.c, x[T],
   mu[N,T], Y[N,T], alpha[N], beta[N],
   tau.alpha, tau.beta, sigma, x.bar;
    data Y in "ratsy.dat", x in "ratsx.dat";
    inits in "rats.in";

    {
     for (i in 1:N) {
      for (j in 1:T) {
        mu[i,j] <- alpha[i] + beta[i]*(x[j] - x.bar);
        Y[i,j]   ~ dnorm(mu[i,j],tau.c)
          }
       alpha[i] ~ dnorm(alpha.c,tau.alpha);
       beta[i]  ~ dnorm(beta.c,tau.beta);
         }
   alpha.c   ~ dnorm(0,1.0E-4);
   beta.c    ~ dnorm(0,1.0E-4);
   tau.c     ~ dgamma(1.0E-3,1.0E-3);
   tau.alpha ~ dgamma(1.0E-3,1.0E-3);
   tau.beta  ~ dgamma(1.0E-3,1.0E-3);
   sigma    <- 1.0/sqrt(tau.c);
   x.bar    <- mean(x[]);
   alpha0   <- alpha.c - beta.c*x.bar;
       }
```

- The response data are in the file *ratsy.dat*

```
151 199 246 283 320
145 199 249 293 354
147 214 263 312 328
155 200 237 272 297
 .....
 .....
157 205 248 289 316
137 180 219 258 291
153 200 244 286 324
```

- The measurement times x are in ratsx.dat:

```
8.0
15.0
22.0
29.0
36.0
```

- Alternatively, the data may be input in S format, as in file ratsS.dat. In this case, both y and x may be included in the same file:

```
list(Y =  c(151.0,199.0,246.0,283.0,320.0,
            145.0,199.0,249.0,293.0,354.0,
            .....
            153.0,200.0,244.0,286.0,324.0),
     x = c(8.0,15.0,22.0,29.0,36.0))
```

- The data statement on line 10 of the rats.bug file must be changed to

```
data in "ratsS.dat";
```

- A naive run, using no diagnostics for convergence, gave the following results:

```
Bugs>update(500)   500    updates took  00:00:02
Bugs>monitor(alpha0)
Bugs>monitor(beta.c)
Bugs>update(1000)   1000   updates took  00:00:04
Bugs>stats(alpha0)
     mean        sd       2.5% :  97.5%  CI    median      sample
   1.063E+2   3.590E+0   9.968E+1   1.132E+2   1.061E+2     1000
Bugs>stats(beta.c)
     mean        sd       2.5% :  97.5%  CI    median      sample
   6.183E+0   1.095E-1   5.968E+0   6.393E+0   6.179E+0     1000
```

- The example also considers the problem of missing data, and delete the last observation of cases 6-10, the last two from 11-20, the last 3 from 21-25 and the last 4 from 26-30.
- The data file is now *ratsmiss.dat*, and is obtained by simply replacing data values by NA (see below).
- The rats.bug file only has to change the data declaration to data Y in *ratsmiss.dat*.

```
Data file "ratsmiss.dat"
151 199 246 283 320
145 199 249 293 354
 .....
153  NA  NA  NA  NA
```

- The predictions for the final 4 observations on rat 26 are obtained automatically in BUGS by monitoring the relevant $Y[]$ nodes. The following is a sample run.

```
Bugs>update(500)   500    updates took  00:00:02
Bugs>monitor(beta.c)
Bugs>monitor(Y[26,])
Bugs>update(1000)   1000   updates took  00:00:04
Bugs>stats(beta.c)
      mean         sd        2.5% :  97.5%  CI      median    sample
     6.537E+0   1.411E-1   6.260E+0   6.811E+0   6.533E+0    1000
Bugs>stats(Y[26,])
         mean        sd        2.5% :  97.5%  CI    median     sample
[26,2] 2.044E+2  8.937E+0   1.865E+2   2.212E+2  2.046E+2     1000
[26,3] 2.497E+2  1.076E+1   2.294E+2   2.706E+2  2.493E+2     1000
[26,4] 2.952E+2  1.280E+1   2.700E+2   3.216E+2  2.949E+2     1000
[26,5] 3.413E+2  1.603E+1   3.115E+2   3.742E+2  3.401E+2     1000
```

- The observed weights for rat 26 were 207, 257, 303 and 345 and the predictions are 204, 250, 295 and 341.