

Bayes Seminar: Part II

John Myles White

March 19, 2012

Automating inference using BUGS:

- ▶ BUGS allows us to build automated systems to perform posterior sampling
- ▶ BUGS constructs a sampler based only on a model description
- ▶ BUGS is a descriptive programming language rather than a procedural programming language

- ▶ A BUGS interpreter works because MCMC sampling is automatable
- ▶ BUGS interpreters often use two types of sampling under the hood:
 - ▶ Gibbs sampling is fully automatable for conjugate parts of models
 - ▶ Metropolis-Hasting sampling is automatable even without conjugacy

Why use BUGS?

- ▶ Separating model descriptions from inference procedures encourages exploration of multiple models
- ▶ Novel models can be built from familiar components without thinking about analytic tractability

Some BUGS interpreters:

- ▶ WinBUGS
- ▶ OpenBUGS
- ▶ JAGS

- ▶ We'll start with inference in the beta-binomial model we used before

Binomial Model I

```
model
{
  for (i in 1:N)
  {
    x[i] ~ dbern(p)
  }

  p ~ dbeta(alpha, beta)

  alpha <- 1
  beta <- 1
}
```

- ▶ Every BUGS model starts with `model`
- ▶ Primitive looping is allowed
- ▶ Two types of assignment: deterministic and stochastic

- ▶ We've used a $\text{beta}(1, 1)$ prior
- ▶ This is a flat prior over $[0, 1]$

- ▶ For inference, BUGS needs to be invoked with data
- ▶ We'll use JAGS as our BUGS interpreter
- ▶ We'll use R via the `rjags` package to send data to JAGS

```
library('rjags')
```

```
df <- read.csv(file.path('data',  
                          'binomial',  
                          'binomial.csv'))
```

For the sample data:

- ▶ $p = 0.4$
- ▶ $n = 5$
- ▶ $\hat{p} = 0.2$

"X"

0

0

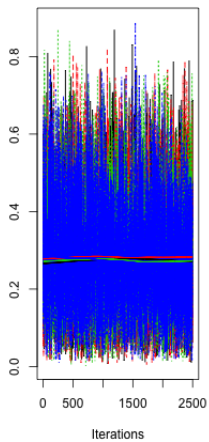
0

1

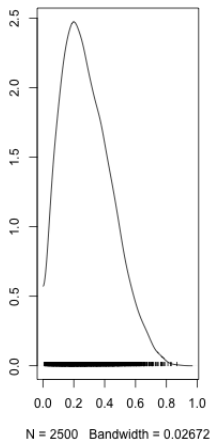
0

```
jags <- jags.model(file.path('bugs',  
                             'binomial',  
                             'binomial.bugs'),  
                  data = list('x' = with(df, X),  
                              'N' = nrow(df)),  
                  n.chains = 4,  
                  n.adapt = 1000)  
  
mcmc.samples <- coda.samples(jags,  
                             c('p'),  
                             2500)  
  
plot(mcmc.samples)  
  
summary(mcmc.samples)
```

Trace of p



Density of p



Iterations = 1:2500

Thinning interval = 1

Number of chains = 4

Sample size per chain = 2500

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

Mean	SD	Naive SE	Time-series SE
0.286161	0.159027	0.001590	0.001576

2. Quantiles for each variable:

2.5%	25%	50%	75%	97.5%
0.04272	0.16242	0.26418	0.39122	0.63907


```
alpha <- with(df, sum(X) + 1)
```

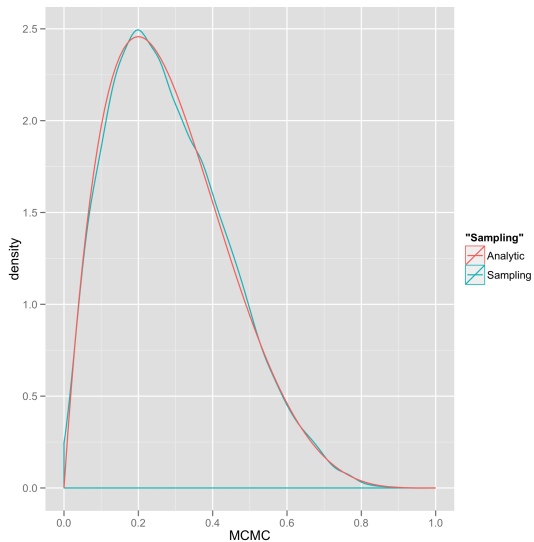
```
beta <- nrow(df) + 2 - alpha
```

```
alpha / (alpha + beta)
```

```
#[1] 0.2857143
```

```
summary(mcmc.samples)$statistics
```

#	Mean	SD	Naive SE	Time-series S
#	0.286160728	0.159026669	0.001590267	0.00157595



```
binom.test(with(df, sum(X)), nrow(df))$conf.int
#[1] 0.005050763 0.716417936
credible.interval <- c(qbeta(0.025, alpha, beta),
                        qbeta(0.975, alpha, beta))
credible.interval
#[1] 0.04327187 0.64123458
binom.test(with(df, sum(X)) + 1, nrow(df) + 2)$conf.int
#[1] 0.03669257 0.70957914
```

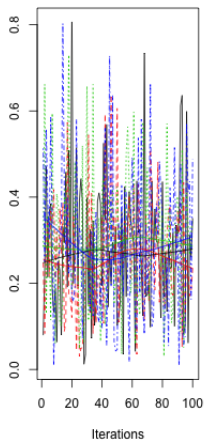
Lessons:

- ▶ BUGS has correctly performed inference
- ▶ Simulation results closely approximate analytic results
- ▶ We get a full posterior, not just a point estimate or CI

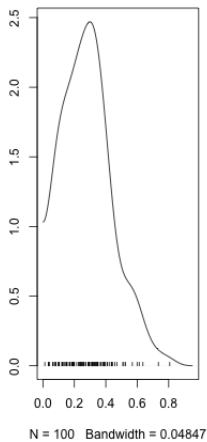
- ▶ What could we have done wrong?
- ▶ Used no adaptive phase
- ▶ Gathered too few samples

```
jags <- jags.model(file.path('bugs',  
                             'binomial',  
                             'binomial.bugs'),  
  data = list('x' = with(df, X),  
              'N' = nrow(df)),  
  n.chains = 4,  
  n.adapt = 0)  
  
mcmc.samples <- coda.samples(jags,  
                             c('p'),  
                             100)  
  
plot(mcmc.samples)
```

Trace of p



Density of p



- ▶ You should play with this example more and try to break it
- ▶ In this case, having no adaptive phase turns out not to matter
- ▶ Using too few samples gives us a wiggly approximation to the posterior

- ▶ Let's move on to inferring the mean of a normal distribution

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

  mu ~ dnorm(0, 0.0001)

  sigma <- 3
  tau <- pow(sigma, -2)
}
```

- ▶ BUGS uses precision rather than variance to specify normal distributions
- ▶ We use a weakly informative Normal prior centered at zero
- ▶ We use arithmetic within BUGS to transform variance into precision

For sample data:

- ▶ $\mu = 25$
- ▶ $\sigma = 3$
- ▶ $n = 100$
- ▶ $\hat{\mu} = 25.32666$

"X"

23.120638567773

25.5509299726662

22.4931141627699

...

23.2802037572893

21.3261621553049

23.5797980906821

```
library('rjags')
```

```
df <- read.csv(file.path('data',  
                          'normal',  
                          'normal_mean.csv'))
```

```
jags <- jags.model(file.path('bugs',  
                             'normal',  
                             'normal_mean.bugs'),  
  data = list('x' = with(df, X),  
              'N' = nrow(df)),  
  n.chains = 4,  
  n.adapt = 500)  
  
mcmc.samples <- coda.samples(jags,  
                             c('mu'),  
                             1000)  
  
plot(mcmc.samples)  
  
summary(mcmc.samples)
```

Iterations = 1:1000

Thinning interval = 1

Number of chains = 4

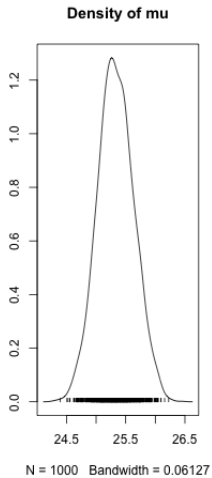
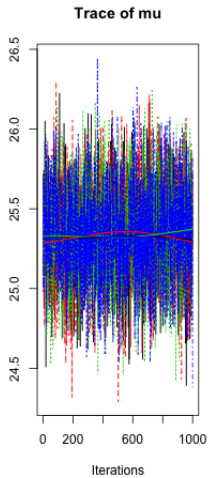
Sample size per chain = 1000

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

Mean	SD	Naive SE	Time-series SE
25.331926	0.303623	0.004801	0.004849

2. Quantiles for each variable:

2.5%	25%	50%	75%	97.5%
24.74	25.13	25.32	25.53	25.93



- ▶ As in the binomial example, we have a conjugate prior here
- ▶ Posterior has closed form solution, which could be used to assess quality of MCMC
- ▶ But we see that MCMC solution is the MLE so we skip that analysis

- ▶ Let's show that inference can be very bad if the wrong prior is used

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

  mu ~ dnorm(0, 10)

  sigma <- 3
  tau <- pow(sigma, -2)
}
```

- ▶ Prior has mean 0
- ▶ Prior has variance $\frac{1}{10}$
- ▶ With so little variance, the prior pulls the posterior towards itself

Iterations = 1:1000

Thinning interval = 1

Number of chains = 4

Sample size per chain = 1000

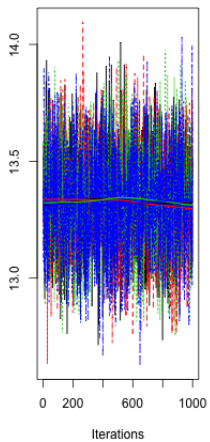
1. Empirical mean and standard deviation for each variable,
plus standard error of the mean:

Mean	SD	Naive SE	Time-series SE
13.327539	0.215767	0.003412	0.003343

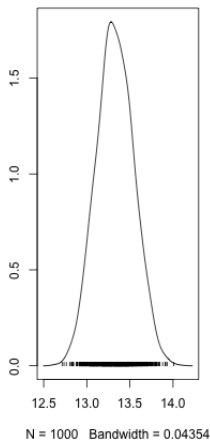
2. Quantiles for each variable:

2.5%	25%	50%	75%	97.5%
12.92	13.18	13.32	13.47	13.75

Trace of mu



Density of mu



- ▶ It can easily get worse


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

  mu ~ dnorm(-1000, 10)

  sigma <- 3
  tau <- pow(sigma, -2)
}
```

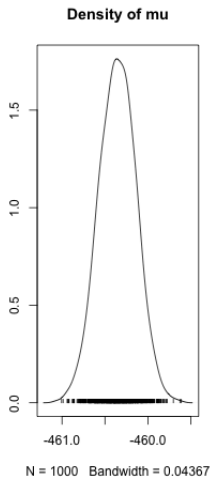
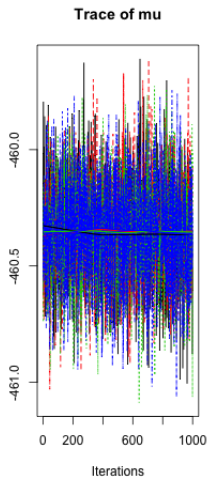
Iterations = 1:1000
Thinning interval = 1
Number of chains = 4
Sample size per chain = 1000

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

Mean	SD	Naive SE	Time-series SE
-4.604e+02	2.164e-01	3.422e-03	3.507e-03

2. Quantiles for each variable:

2.5%	25%	50%	75%	97.5%
-460.8	-460.5	-460.4	-460.2	-459.9



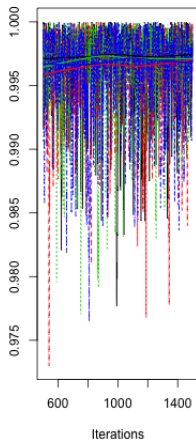
- ▶ Using the wrong prior variance causes trouble
- ▶ Using the wrong prior mean causes more trouble
- ▶ Using the wrong prior functional form can be still worse

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

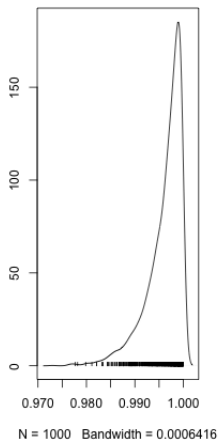
  mu ~ dunif(0, 1)

  sigma <- 3
  tau <- pow(sigma, -2)
}
```

Trace of mu



Density of mu



Iterations = 501:1500

Thinning interval = 1

Number of chains = 4

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
9.962e-01	3.691e-03	5.836e-05	1.361e-04

2. Quantiles for each variable:

2.5%	25%	50%	75%	97.5%
0.9861	0.9946	0.9973	0.9989	0.9999

- ▶ Because our prior assigns 0 probability to most values of μ , it can't reach them even with infinite data
- ▶ Lindley refers to the requirement that priors not assign 0 probability as Cromwell's Rule:

I beseech you, in the bowels of Christ, think it possible that you may be mistaken.

- ▶ When you violate Cromwell's Rule, you can get extremely tight posteriors around very wrong values
- ▶ In this case, you can discover your mistake because of the truncated posterior

- ▶ But take away this lesson:
 - ▶ If you have valid prior information, put it in the prior
 - ▶ Otherwise, give the prior very high variance

- ▶ Let's leave single parameter inference behind
- ▶ We'll implement a Bayesian linear regression

```
model
{
  for (i in 1:N)
  {
    y[i] ~ dnorm(mu[i], tau)
    mu[i] <- a * x[i] + b
  }

  a ~ dnorm(0, 0.0001)
  b ~ dnorm(0, 0.0001)

  tau <- pow(sigma, -2)
  sigma ~ dunif(0, 10000)
}
```

"X", "Y"

0,84.3386547314417

0.1,105.591083105552

0.2,81.1092846897488

0.3,142.882020053445

...

9.7,182.668364644078

9.8,167.384684627541

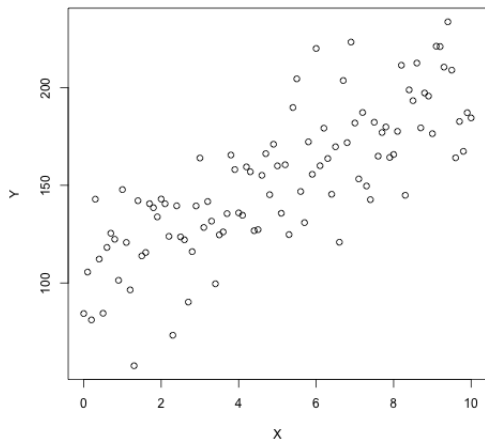
9.9,187.164984089017

10,184.490833069397

For the sample data:

- ▶ $a = 10$
- ▶ $b = 100$
- ▶ $\sigma = 25$

```
library('rjags')  
  
df <- read.csv(file.path('data',  
                          'ols',  
                          'ols_regression.csv'))  
  
with(df, plot(X, Y))
```



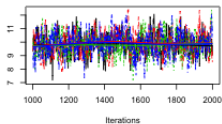
```
jags <- jags.model(file.path('bugs',  
                             'ols',  
                             'ols_regression.bugs'),  
  data = list('x' = with(df, X),  
              'y' = with(df, Y),  
              'N' = nrow(df)),  
  n.chains = 4,  
  n.adapt = 1000)
```

```
mcmc.samples <- coda.samples(jags,  
                             c('a', 'b', 'sigma'),  
                             1000)
```

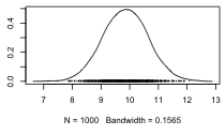
```
plot(mcmc.samples)
```

```
summary(mcmc.samples)
```

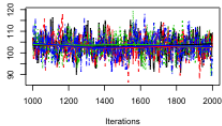

Trace of a



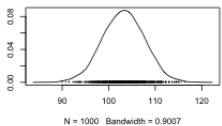
Density of a



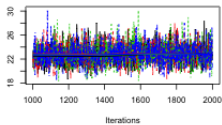
Trace of b



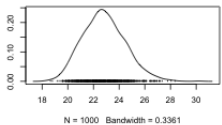
Density of b



Trace of sigma



Density of sigma



Iterations = 1001:2000

Thinning interval = 1

Number of chains = 4

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
a	9.875	0.8035	0.01270	0.03743
b	103.023	4.6591	0.07367	0.22097
sigma	22.805	1.6429	0.02598	0.03003

2. Quantiles for each variable:

	2.5%	25%	50%	75%	97.5%
a	8.239	9.349	9.887	10.40	11.45
b	93.962	99.996	103.088	105.97	112.60
sigma	19.895	21.640	22.687	23.83	26.32

Call:

```
lm(formula = Y ~ X, data = df)
```

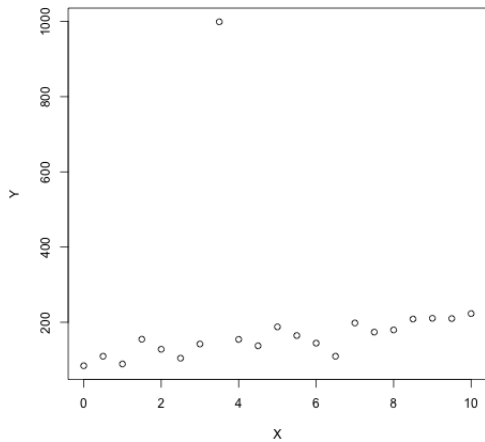
Coefficients:

(Intercept)	X
103.620	9.784

- ▶ Implementing simple linear regression is easy
- ▶ BUGS's virtue is that alternatives are just as easy to implement

- ▶ Consider a case where the Gaussian noise assumption is wrong
- ▶ One solution is to replace the OLS objective with the LAD objective
- ▶ For a Bayesian this amounts to assuming Laplace noise

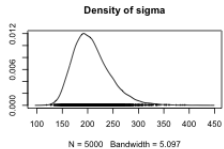
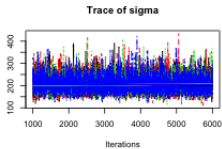
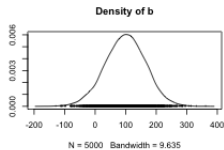
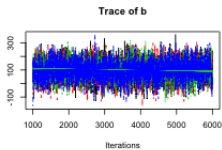
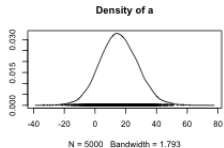
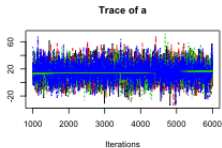
```
library('rjags')  
  
df <- read.csv(file.path('data',  
                          'ols',  
                          'ols_regression_outlier.csv'))  
  
with(df, plot(X, Y))
```



For the sample data:

- ▶ $a = 10$
- ▶ $b = 100$
- ▶ $\sigma = 25$
- ▶ One outlier was added post hoc


```
jags <- jags.model(file.path('bugs',  
                             'ols',  
                             'ols_regression_outlier.bugs'),  
  data = list('x' = with(df, X),  
              'y' = with(df, Y),  
              'N' = nrow(df)),  
  n.chains = 4,  
  n.adapt = 1000)  
  
mcmc.samples <- coda.samples(jags,  
                             c('a', 'b', 'sigma'),  
                             5000)  
  
plot(mcmc.samples)  
  
summary(mcmc.samples)
```



Iterations = 1001:6000
Thinning interval = 1
Number of chains = 4
Sample size per chain = 5000

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

	Mean	SD	Naive SE	Time-series SE
a	15.41	12.27	0.08675	0.1764
b	98.72	65.88	0.46585	0.9934
sigma	208.10	36.85	0.26056	0.4147

2. Quantiles for each variable:

	2.5%	25%	50%	75%	97.5%
a	-8.068	7.127	15.16	23.55	39.64
b	-33.903	54.731	99.57	143.41	225.82
sigma	150.876	181.921	203.17	228.62	293.77

- ▶ Using a heavier-tailed error distribution gives the outlier less influence
- ▶ We use Laplace errors, but we could use other distributions

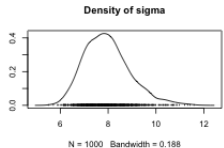
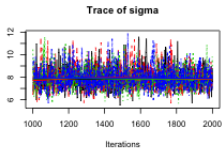
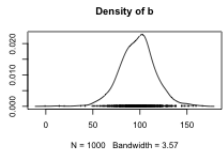
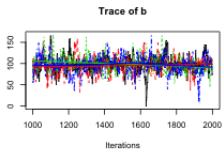
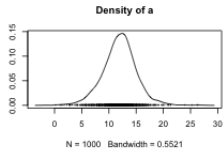
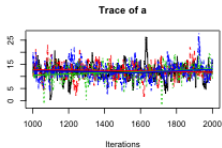
```
model
{
  for (i in 1:N)
  {
    y[i] ~ ddexp(mu[i], tau)
    mu[i] <- a * x[i] + b
  }

  a ~ dnorm(0, 0.0001)
  b ~ dnorm(0, 0.0001)

  tau <- pow(sigma, -2)
  sigma ~ dunif(0, 100)
}
```

```
library('rjags')  
  
df <- read.csv(file.path('data',  
                          'lad',  
                          'lad_regression.csv'))  
  
with(df, plot(X, Y))
```

```
jags <- jags.model(file.path('bugs',  
                             'lad',  
                             'lad_regression.bugs'),  
  data = list('x' = with(df, X),  
              'y' = with(df, Y),  
              'N' = nrow(df)),  
  n.chains = 4,  
  n.adapt = 1000)  
  
mcmc.samples <- coda.samples(jags,  
                             c('a', 'b', 'sigma'),  
                             1000)  
  
plot(mcmc.samples)  
  
summary(mcmc.samples)
```



Iterations = 1001:2000
Thinning interval = 1
Number of chains = 4
Sample size per chain = 1000

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

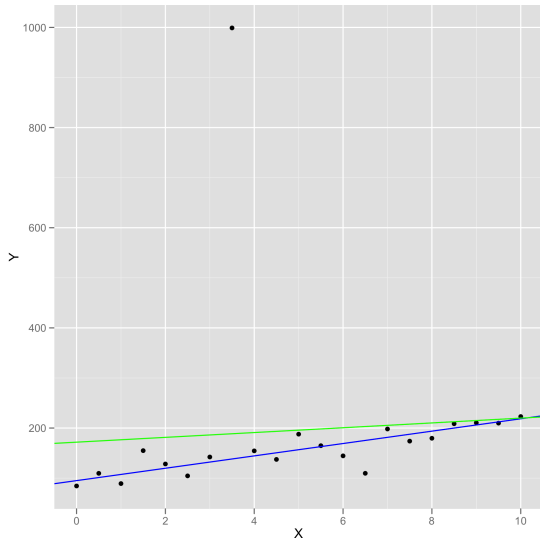
	Mean	SD	Naive SE	Time-series SE
a	11.956	3.1146	0.04925	0.18855
b	97.419	19.0487	0.30119	1.10237
sigma	7.959	0.9604	0.01519	0.02498

2. Quantiles for each variable:

	2.5%	25%	50%	75%	97.5%
a	5.654	10.144	12.038	13.811	18.28
b	59.353	85.228	98.091	108.933	135.82
sigma	6.361	7.264	7.869	8.513	10.22

- ▶ Our inferences for a and b are much better
- ▶ σ is no longer meaningful since we have a different error distribution

- ▶ Visualizing the OLS and LAD lines is even more compelling



Instead of modeling continuous data, we can model 0/1 data:

- ▶ Logit regression
- ▶ Probit regression
- ▶ Robit regression

```
model
{
  for (i in 1:N)
  {
    y[i] ~ dbern(p[i])
    logit(p[i]) <- a * x[i] + b
  }

  a ~ dnorm(0, 0.0001)
  b ~ dnorm(0, 0.0001)
}
```

```
library('rjags')  
  
df <- read.csv(file.path('data',  
                          'logit',  
                          'logit_regression.csv'))  
  
with(df, plot(X, Y))
```

"X", "Y"

3.1,0

3.2,0

3.3,0

3.4,1

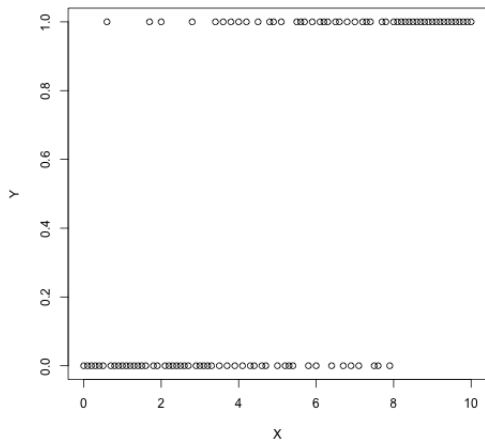
...

7.1,0

7.2,1

7.3,1

7.4,1



For the sample data:

- ▶ $y \sim f(ax + b)$
- ▶ $a = 0.55$
- ▶ $b = -3$

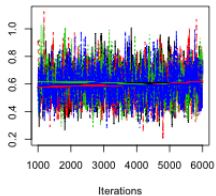
```
jags <- jags.model(file.path('bugs',  
                             'logit',  
                             'logit.bugs'),  
                  data = list('x' = with(df, X),  
                              'y' = with(df, Y),  
                              'N' = nrow(df)),  
                  n.chains = 4,  
                  n.adapt = 1000)
```

```
mcmc.samples <- coda.samples(jags,  
                             c('a', 'b'),  
                             5000)
```

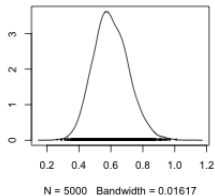
```
plot(mcmc.samples)
```

```
summary(mcmc.samples)
```

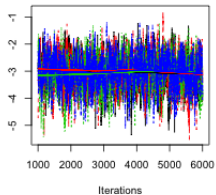
Trace of a



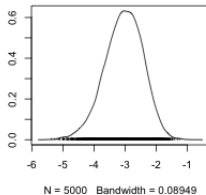
Density of a



Trace of b



Density of b



Iterations = 1001:6000
Thinning interval = 1
Number of chains = 4
Sample size per chain = 5000

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

	Mean	SD	Naive SE	Time-series SE
a	0.6012	0.1105	0.0007816	0.003321
b	-3.0437	0.6119	0.0043267	0.018217

2. Quantiles for each variable:

	2.5%	25%	50%	75%	97.5%
a	0.4002	0.5246	0.5956	0.6738	0.8313
b	-4.3329	-3.4444	-3.0180	-2.6117	-1.9305

- ▶ The probit model, popular in econometrics, is largely equivalent to the logit model
- ▶ The probit link function is used instead of the logit link function

```
model
{
  for (i in 1:N)
  {
    y[i] ~ dbern(p[i])
    probit(p[i]) <- a * x[i] + b
  }

  a ~ dnorm(0, 0.0001)
  b ~ dnorm(0, 0.0001)
}
```

```
library('rjags')  
  
df <- read.csv(file.path('data',  
                          'probit',  
                          'probit_regression.csv'))  
  
with(df, plot(X, Y))
```


"X", "Y"

3.1,0

3.2,0

3.3,0

3.4,0

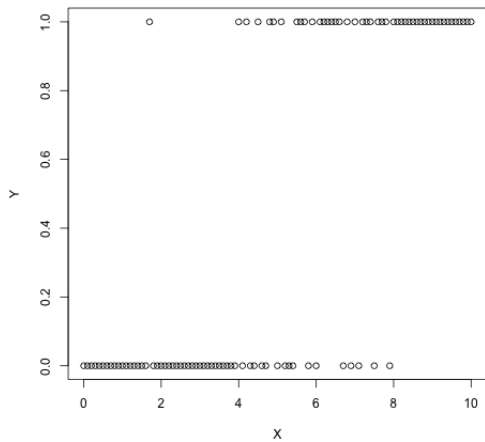
...

7.1,0

7.2,1

7.3,1

7.4,1



For the sample data:

- ▶ $y \sim f(ax + b)$
- ▶ $a = 0.55$
- ▶ $b = -3$

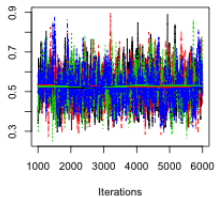
```
jags <- jags.model(file.path('bugs',  
                             'probit',  
                             'probit.bugs'),  
                  data = list('x' = with(df, X),  
                              'y' = with(df, Y),  
                              'N' = nrow(df)),  
                  n.chains = 4,  
                  n.adapt = 1000)
```

```
mcmc.samples <- coda.samples(jags,  
                             c('a', 'b'),  
                             5000)
```

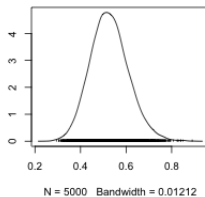
```
plot(mcmc.samples)
```

```
summary(mcmc.samples)
```

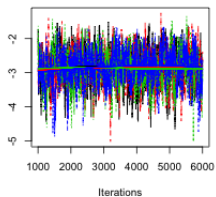
Trace of a



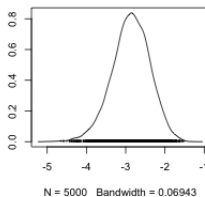
Density of a



Trace of b



Density of b



Iterations = 1001:6000
Thinning interval = 1
Number of chains = 4
Sample size per chain = 5000

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

	Mean	SD	Naive SE	Time-series SE
a	0.5267	0.08412	0.0005948	0.003186
b	-2.8686	0.48380	0.0034210	0.018409

2. Quantiles for each variable:

	2.5%	25%	50%	75%	97.5%
a	0.3721	0.4688	0.5231	0.5799	0.703
b	-3.8845	-3.1716	-2.8515	-2.5355	-1.972

- ▶ As with the OLS model, outliers can cause trouble for the logit/probit models
- ▶ Like the LAD model, the robit model attempts to be more robust to outliers

```
model
{
  for (i in 1:N)
  {
    y[i] ~ dbern(p[i])
    p[i] <- pt(z[i], 0, 1, 1)
    z[i] <- a * x[i] + b
  }

  a ~ dnorm(0, 0.0001)
  b ~ dnorm(0, 0.0001)
}
```



```
library('rjags')  
  
df <- read.csv(file.path('data',  
                          'robit',  
                          'robit_regression.csv'))  
  
with(df, plot(X, Y))
```

"X", "Y"

0,0

0.5,0

1,0

1.5,1

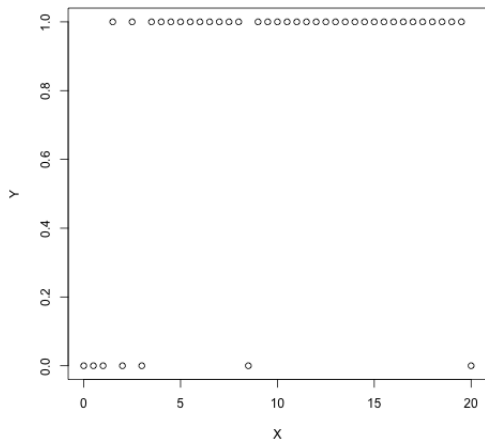
...

18.5,1

19,1

19.5,1

20,0



For the sample data:

- ▶ $y \sim f(ax + b)$

- ▶ $a = 2$

- ▶ $b = -5$

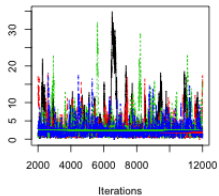
```
jags <- jags.model(file.path('bugs',  
                             'probit',  
                             'probit.bugs'),  
                  data = list('x' = with(df, X),  
                              'y' = with(df, Y),  
                              'N' = nrow(df)),  
                  n.chains = 4,  
                  n.adapt = 1000)
```

```
mcmc.samples <- coda.samples(jags,  
                             c('a', 'b'),  
                             5000)
```

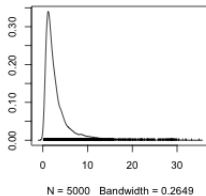
```
plot(mcmc.samples)
```

```
summary(mcmc.samples)
```

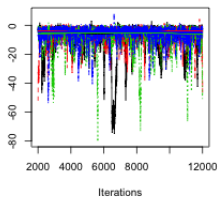
Trace of a



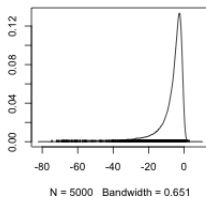
Density of a



Trace of b



Density of b



- ▶ Because our posteriors are very skewed, we take medians rather than means

```
apply(as.matrix(mcmc.samples), 2, median)
#           a           b
# 2.031132 -4.533720
```


- ▶ So far these models aren't outperforming classical procedures
- ▶ The only virtue has been the simplicity of implementing new models in BUGS

- ▶ Bayesian methods start to shine for more complex models
 - ▶ Models with many parameters
 - ▶ Models using missing data
 - ▶ Models that require label imputation

Let's go through three such problems:

- ▶ Hierarchical OLS regression
- ▶ Hierarchical logistic regression
- ▶ The ideal points model

- ▶ In a hierarchical model, we have groups of parameters
- ▶ We believe parameters within a group should be similar
- ▶ We induce this by adding a layer of adjustable priors to our model

- ▶ We start with hierarchical linear regression
- ▶ We assume that we have K groups

```
model
{
  for (i in 1:N)
  {
    y[i] ~ dnorm(mu[i], tau)
    mu[i] <- a[g[i]] * x[i] + b[g[i]]
  }

  for (j in 1:K)
  {
    a[j] ~ dnorm(mu.a, tau.a)
    b[j] ~ dnorm(mu.b, tau.b)
  }
}
```

```
mu.a ~ dnorm(0, 0.0001)
```

```
mu.b ~ dnorm(0, 0.0001)
```

```
tau <- pow(sigma, -2)
```

```
sigma ~ dunif(0, 1000)
```

```
tau.a <- pow(sigma.a, -2)
```

```
tau.b <- pow(sigma.b, -2)
```

```
sigma.a ~ dunif(0, 1000)
```

```
sigma.b ~ dunif(0, 1000)
```

```
}
```

```
library('rjags')
```

```
df <- read.csv(file.path('data',  
                          'hierarchical',  
                          'hierarchical_linear.csv'))
```


"g", "t", "x", "y"

1,1,2.01681931037456,8.90568024969301

1,2,6.60797792486846,49.1123529360177

1,3,2.05974574899301,2.70506413639457

...

10,98,2.81181986443698,12.7356528582657

10,99,8.29269654583186,66.2010735394361

10,100,0.0725971511565149,-8.11623818910855

For the sample data:

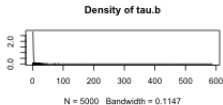
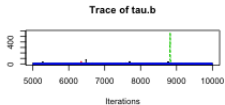
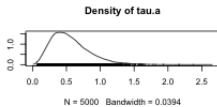
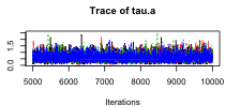
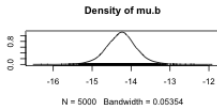
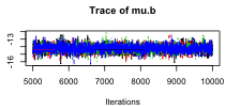
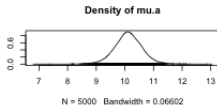
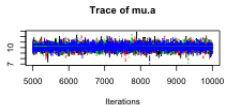
- ▶ $y \sim a_{g(i)}x + b_{g(i)}$
- ▶ $\mu_a = 10$
- ▶ $\mu_b = -15$

```
jags <- jags.model(file.path('bugs',  
                             'hierarchical',  
                             'hierarchical_linear.bugs'),  
  data = list('x' = with(df, X),  
              'y' = with(df, Y),  
              'N' = nrow(df)),  
  n.chains = 4,  
  n.adapt = 1000)
```

```
mcmc.samples <- coda.samples(jags,  
                             c('mu.a',  
                               'mu.b',  
                               'tau.a',  
                               'tau.b'),  
                             5000)
```

```
plot(mcmc.samples)
```

```
summary(mcmc.samples)
```



Iterations = 5001:10000
Thinning interval = 1
Number of chains = 4
Sample size per chain = 5000

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

	Mean	SD	Naive SE	Time-series SE
mu.a	10.1173	0.4890	0.003458	0.003566
mu.b	-14.2479	0.3997	0.002826	0.005236
tau.a	0.5579	0.2833	0.002004	0.002953
tau.b	1.8164	10.0432	0.071016	0.350492

- ▶ Let's also implement a hierarchical logit model

```
model
{
  for (i in 1:N)
  {
    y[i] ~ dbern(p[i])
    logit(p[i]) <- a[g[i]] * x[i] + b[g[i]]
  }

  for (j in 1:K)
  {
    a[j] ~ dnorm(mu.a, tau.a)
    b[j] ~ dnorm(mu.b, tau.b)
  }
}
```

```
mu.a ~ dnorm(0, 0.0001)
```

```
mu.b ~ dnorm(0, 0.0001)
```

```
tau.a <- pow(sigma.a, -2)
```

```
tau.b <- pow(sigma.b, -2)
```

```
sigma.a ~ dunif(0, 1000)
```

```
sigma.b ~ dunif(0, 1000)
```

```
}
```



```
library('rjags')
```

```
df <- read.csv(file.path('data',  
                          'hierarchical',  
                          'hierarchical_logit.csv'))
```

"g", "t", "x", "y"
1,1,2.01681931037456,1
1,2,9.44675268605351,0
1,3,6.2911404389888,0
...
25,48,6.72132012201473,0
25,49,3.42435503611341,0
25,50,7.37378113903105,0

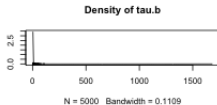
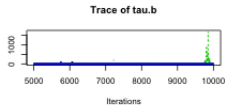
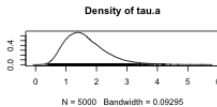
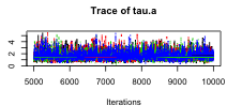
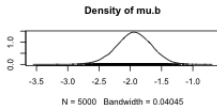
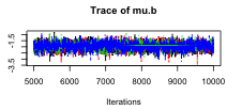
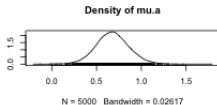
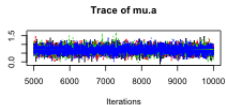
For the sample data:

- ▶ $y \sim f(a_{g(i)}x + b_{g(i)})$
- ▶ $\mu_a = 0.5$
- ▶ $\mu_b = -2$

```
jags <- jags.model(file.path('bugs',  
                             'hierarchical',  
                             'hierarchical_logit.bugs'),  
  data = list('x' = with(df, x),  
              'y' = with(df, y),  
              'g' = with(df, g),  
              'N' = nrow(df),  
              'K' = with(df, max(g))),  
  n.chains = 4,  
  n.adapt = 5000)
```

```
mcmc.samples <- coda.samples(jags,  
                             c('mu.a',  
                               'mu.b',  
                               'tau.a',  
                               'tau.b'),  
                             5000)
```

```
plot(mcmc.samples)
```



Iterations = 5001:10000
Thinning interval = 1
Number of chains = 4
Sample size per chain = 5000

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

	Mean	SD	Naive SE	Time-series SE
mu.a	0.6795	0.1854	0.001311	0.002471
mu.b	-1.9432	0.2827	0.001999	0.006487
tau.a	1.6534	0.6791	0.004802	0.013086
tau.b	3.8376	31.0577	0.219611	1.581173

- ▶ Both of these hierarchical models are very powerful in practice
- ▶ Try them out on your data and see if they make better predictions on held out data

- ▶ As one of two closing models, let's implement the ideal points model
- ▶ This model is quite complex to implement without Bayesian methods
- ▶ It also produces beautiful results

Senator,Bill,Vote

BYRD (D WV),2,0

ENZI (R WY),2,1

```
model
{
  for (i in 1:M)
  {
    for (j in 1:N)
    {
      votes[i, j] ~ dbern(p[i, j])
      logit(p[i, j]) <- g[j] * (a[i] - b[j])
    }
  }
}
```

```
for (i in 1:M)
{
  a[i] ~ dnorm(0, tau.a)
}

tau.a <- pow(sigma.a, -2)
sigma.a ~ dunif(0, 100)
```

```
for (j in 1:N)
{
  b[j] ~ dnorm(mu.b, tau.b)
  g[j] ~ dnorm(0, tau.g)
}

mu.b ~ dnorm(0, .0001)

tau.b <- pow(sigma.b, -2)
sigma.b ~ dunif(0, 100)

tau.g <- pow(sigma.g, -2)
sigma.g ~ dunif(0, 100)
}
```

```
library('rjags')  
  
binary.votes <- read.table(file.path('data',  
                                     'ideal_points',  
                                     'ideal_points.txt'))  
binary.votes <- as.matrix(binary.votes)
```

```
a <- rep(NA, nrow(binary.votes))  
a[which(row.names(binary.votes) == "COBURN (R OK)")] <- 2  
  
jags <- jags.model(file.path('bugs',  
                             'ideal_points',  
                             'ideal_points.bugs'),  
                  data = list('votes' = binary.votes,  
                              'M' = nrow(binary.votes),  
                              'N' = ncol(binary.votes),  
                              'a' = a),  
                  n.chains = 4,  
                  n.adapt = 500)
```

```
j.samples <- jags.samples(jags,  
                           c('a', 'b', 'g'),  
                           250)  
  
estimated.a <- apply(j.samples$a, 1, mean)  
  
df <- data.frame(Senator = row.names(binary.votes),  
                 IdealPoint = estimated.a)  
  
head(df)  
tail(df)
```

Senator IdealPoint

1	BUSH (R USA)	1.321483
2	SESSIONS (R AL)	2.042710
3	SHELBY (R AL)	1.346424
4	MURKOWSKI (R AK)	1.000675
5	STEVENS (R AK)	1.069571
6	KYL (R AZ)	1.757788

Senator IdealPoint

97	BYRD (D WV)	-0.7941595
98	ROCKEFELLER (D WV)	-1.1672899
99	FEINGOLD (D WI)	-1.5509000
100	KOHL (D WI)	-1.0407957
101	ENZI (R WY)	1.8374551
102	THOMAS (R WY)	1.8247487

- ▶ As a last model, we'll implement a mixture of two Gaussians

```

model
{
  for (i in 1:N)
  {
    z[i] ~ dbern(p)
    mu[i] <- z[i] * mu1 + (1 - z[i]) * mu2
    x[i] ~ dnorm(mu[i], tau)
  }

  p ~ dbeta(1, 1)

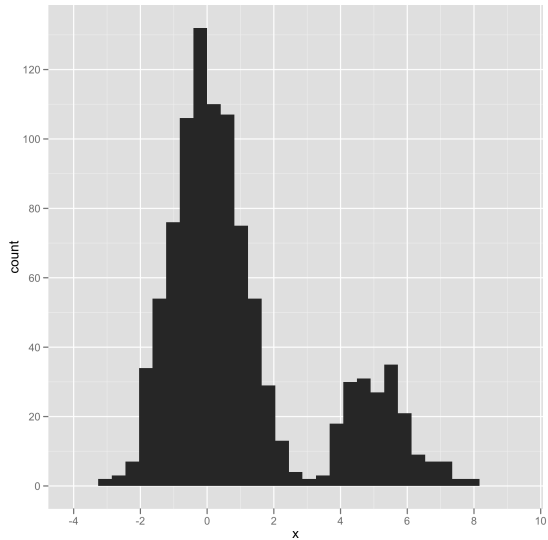
  mu1 ~ dnorm(0, 0.0001)
  mu2 ~ dnorm(1, 0.0001)

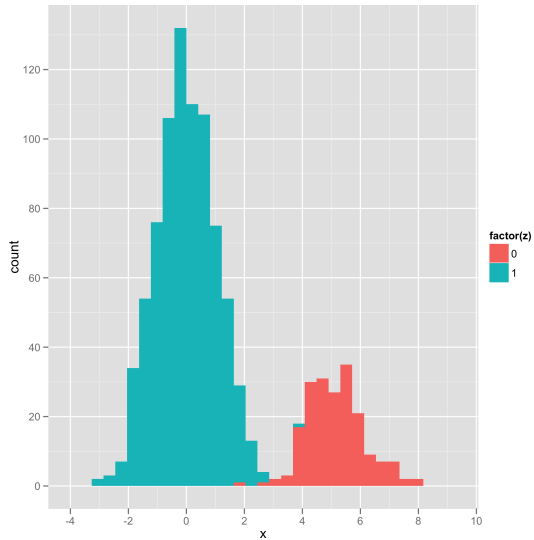
  tau <- pow(sigma, -2)
  sigma ~ dunif(0, 100)
}

```

```
library('rjags')
```

```
df <- read.csv(file.path('data',  
                          'mixture_models',  
                          'two_normals.csv'))
```





For the sample data:

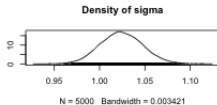
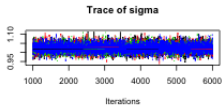
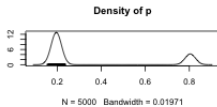
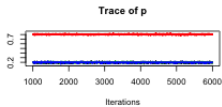
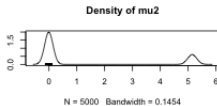
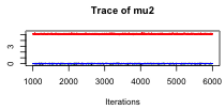
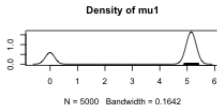
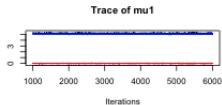
- ▶ $p = 0.8$

- ▶ $\mu_1 = 0$

- ▶ $\mu_2 = 5$

- ▶ $\sigma = 1$

```
jags <- jags.model(file.path('bugs',  
                             'mixture_models',  
                             'two_normals.bugs'),  
  data = list('x' = with(df, x),  
              'N' = nrow(df)),  
  n.chains = 4,  
  n.adapt = 1000)  
  
mcmc.samples <- coda.samples(jags,  
                             c('p', 'mu1', 'mu2', 'sigma'),  
                             5000)  
  
plot(mcmc.samples)  
  
summary(mcmc.samples)
```



Iterations = 1001:6000
Thinning interval = 1
Number of chains = 4
Sample size per chain = 5000

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

	Mean	SD	Naive SE	Time-series SE
mu1	3.858	2.23225	0.0157844	5.426e-04
mu2	1.281	2.23140	0.0157784	3.988e-04
p	0.348	0.26360	0.0018640	8.817e-05
sigma	1.024	0.02339	0.0001654	2.155e-04

- ▶ We'll stop there, but BUGS can easily be used for other models:
 - ▶ LDA with a fixed number of topics
 - ▶ Social Network Analysis via the stochastic blockmodel
 - ▶ ...