A gentle introduction to Bayesian inference, MCMC, and Stan

Matt Espe

February 10, 2017

Data Science Initiative, University of California - Davis

Outline

Bayesian inference (a crash course)

Bayes Theorem

$$P(\theta \mid X) = \frac{P(X \mid \theta)\, P(\theta)}{P(X)}$$

$P(\theta \mid X)$ is the posterior,

$P(X \mid \theta)$ is the likelihood,

$P(\theta)$ is the prior,

$P(X)$ is the support.

Bayes Theorem (cont.)

$$P(\theta \mid X) \propto P(X \mid \theta)\, P(\theta)$$

Why Bayes?

It presents a coherent method to combine information from multiple sources.

It is flexible.

However, it will not save you from yourself!

The issue with Bayes

For simple problems, we can analytically derive the posterior.

More complicated models might not be analytically tractable.

But, we can sample from $P(\theta \mid X)$ even without knowing the analytical solution!

Using samples from $P(\theta \mid X)$, we can calculate MCMC estimators.

Markov Chain Monte Carlo

MCMC

Markov Chain Monte Carlo (MCMC) is a method to sample from $P(\theta \mid X)$.

Many different algorithms with different strengths and weaknesses.

MCMC estimators will converge to true expectations (eventually).

MCMC (cont.)

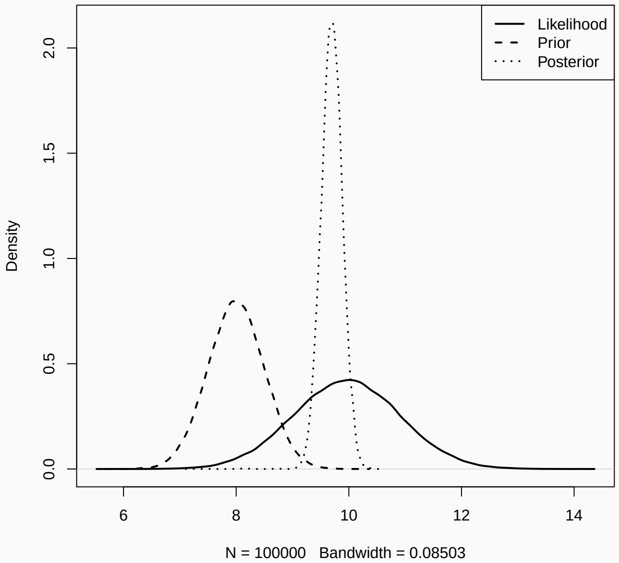
MCMC algorithms are not magic.

- The good: You can write custom samplers, develop highly flexible models, and do generally cool stuff.
- The bad: They cannot overcome issues with model specification, data collection or experimental design, etc.
- The ugly: You never know if they are working properly, only if they are not obviously broken.

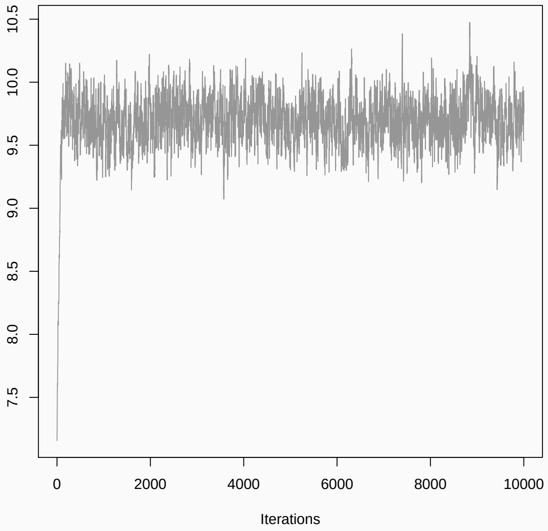
Random Walk Metropolis Hastings

```r
RWmetroNorm <- function(x, prior_mu, prior_sd, known_sd = 1,
                        step_size = 0.1, iter = 100)
{
      samples <- numeric(iter)
      samples[1] <- rnorm(1, prior_mu, prior_sd)

      for(i in 2:iter){
          prop <- rnorm(1, samples[i - 1], step_size)

          cur_ll <- sum(dnorm(x, samples[i - 1], sd = known_sd, log = TRUE)) +
              dnorm(samples[i - 1], prior_mu, prior_sd, log = TRUE)
          prop_ll <- sum(dnorm(x, prop, sd = known_sd, log = TRUE)) +
              dnorm(prop, prior_mu, prior_sd, log = TRUE)

          jump_prob <- min(exp(prop_ll - cur_ll), 1)

          if(runif(1) < jump_prob){
              samples[i] <- prop
          } else {
              samples[i] <- samples[i -1]
          }
      }
      return(samples)
  }
```

Random Walk MH (cont.)



The ’fuzzy caterpillar’



JAGS: Just Another Gibbs Sampler

A popular MCMC sampler with cross-platform support.

```
mod.text <- '
model{
  for( i in 1:N ){
    x[i] ~ dnorm(mu, (1/known_sd ^2))
    }
  mu ~ dnorm(prior_mu, (1/prior_sd^2))
}'
```

Comparing MCMC algorithms

Comparing MCMC algorithms

Which one is faster?

```
RW_time


##    user  system elapsed
##   1.453   0.067   1.518


JAGS_time


##    user  system elapsed
##   0.053   0.000   0.051
```

Is faster what we want?

Not necessarily - how do we measure speed?

Comparing MCMC (cont.)

Effective size: The equivalent number of independent draws.

```
effectiveSize(ans)
```

```
##    var1
## 4782.68
```

```
effectiveSize(jags.ans)
```

```
##    mu
## 1e+05
```

Comparing MCMC (cont.)

Effective size/time is often what we care about (efficiency).

As the effective number of samples increases, the MCMC estimators will be converge to the true estimator.

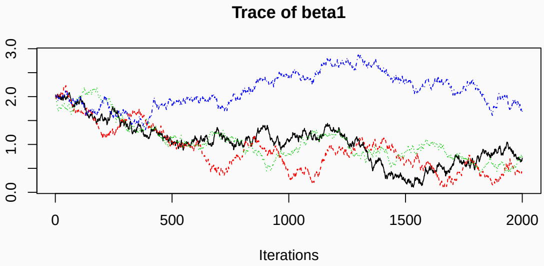
Unless high precision (low MCMC error) is needed, 1000 effective samples is sufficient for most applications.

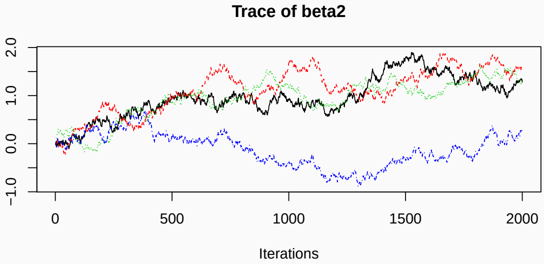
Troublesome posteriors

Let's break things!

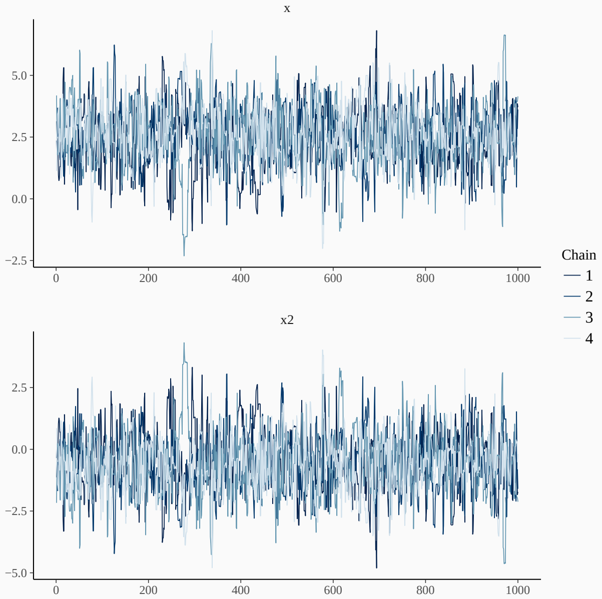
```
x2 <- rnorm(25, x, sd = 0.1)
y <- rnorm(25, x * 2, 0.5)

mod2.text <- '
model{
  for( i in 1:N ){
    y[i] ~ dnorm(beta1 * x[i] + beta2 * x2[i],
                 (1/known_sd ^2))
    }
  beta1 ~ dnorm(0, 1)
  beta2 ~ dnorm(0, 1)
}'
```

Difficult posteriors



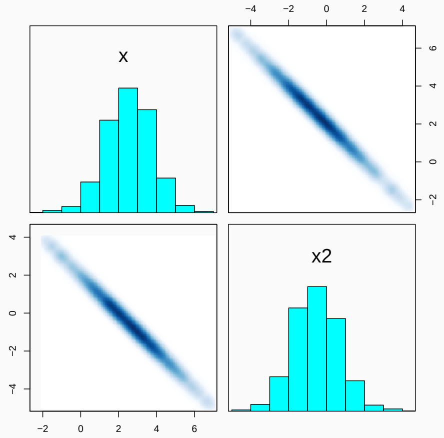


(Enter Stan)



x
5.0
2.5
0.0
−2.5
0
200
400
600
800
1000
x2
2.5
0.0
−2.5
−5.0
0
200
400
600
800
1000
Chain
1
2
3
4

Why does this break?



What is the impact of this?

```
effectiveSize(jags.ans2)

##    beta1    beta2
## 14.52412 14.50943

summary(stan.mod)[1:2,9]

##   x  x2
## 940 940
```

MCMC efficiency revisited

JAGS is more efficient due to its speed, but requires many more draws.

```
effectiveSize(jags.ans2)/jags_time['elapsed']
```

```
##    beta1    beta2
## 453.8787 453.4197
```

```
summary(stan.mod)[1:2,9]/stan_time['elapsed']
```

```
##       x      x2
## 57.7431 57.7431
```

Typical JAGS solution

Sample a ton, then throw out all but every *n* draw.

100,000 iterations with a 50 draw thin:

```
effectiveSize(jags.ans2)
```

```
##    beta1    beta2
## 1800.337 1802.250
```

A more difficult model to fit

$$y = \alpha + \beta_{site} + \beta_{cultivar} + \beta_{year} + \beta_{year:site}$$

N = 4087

n cult = 14

n year = 21

n site = 25

n site:year = 168

In JAGS

```
m3 <- 'model{
  for( i in 1:N ){
    mu[i] = alpha + b_site[site[i]] +
        b_cult[cult[i]] + b_year[year[i]] +
        b_siteyr[siteyr[i]]
    yield[i] ~ dnorm(mu[i],
                 (1/sigma ^2))
    }
  alpha ~ dnorm(0, 1)
  for(i in 1:n_site){
    b_site[i] ~ dnorm(0, (1/pow(tau[1],2)))}
  for(i in 1:n_cult){
    b_cult[i] ~ dnorm(0, (1/pow(tau[2],2)))}
  for(i in 1:n_year){
    b_year[i] ~ dnorm(0, (1/pow(tau[3],2)))}
  for(i in 1:n_yr_site){
    b_siteyr[i] ~ dnorm(0, (1/pow(tau[4],2)))}
  for(i in 1:4){
    tau[i] ~ dunif(0, 10)}
  sigma ~ dunif(0, 10)
}'
```

Efficiency

```r
effectiveSize(jags.fit)[1:5]/jags.time2["elapsed"]
```

```
## b_cult[1] b_cult[2] b_cult[3] b_cult[4] b_cult[5]
##  2.179756  2.318169  3.774028  4.925949  1.893628
```

```r
summary(stan.fit, regex_pars = " id")[1:5, 9]/stan.time2["elapsed"]
```

```
## b[(Intercept) id:M103] b[(Intercept) id:M104] b[(Intercept) id:M105]
##               3.421933               3.421933               3.421933
## b[(Intercept) id:M201] b[(Intercept) id:M202]
##               3.421933               3.421933
```

Some caution required

Diagnostics

$\hat{R}$ compares within and between chain variability

$\mathrm{N}_{\mathrm{eff}}$ measures the effective draws a sample contains

$\mathrm{N}_{\mathrm{eff}}/\mathrm{s}$ measures efficiency

$\mathrm{N}_{\mathrm{eff}}/\mathrm{draw}$ is a different measure of efficiency

Traceplots show how well chains mix (i.e., traverse the posterior)

WAIC, PS-loo, Fraction of Bayesian missing information, etc...

Warning!

MCMC is a powerful tool but:

OK diagnostics are a sign that nothing has gone obviously wrong, not a sign that things have gone right.

On that note...

Always run multiple chains, starting from different initial values. A good rule of thumb is minimum of 4 chains, more if possible.

(I typically run 8-16 chains)

More chains increases the chances that one will find a pathological region in the posterior, numeric instability, etc.

Do not just remove a poorly sampling chain! It is a sign something is wrong!

Conclusion

So why Stan?

Stan was built specifically to handle:

1. Difficult posteriors
2. High dimensional problems

Additional benefits

Stan is flexible*

Under active development

Large user base

Cool add-on tools

Conclusions

Bayesian inference for most non-trivial problems requires some way to sample from the posterior distribution, $P(\theta \mid X)$.

Samples can be generated using MCMC algorithms

Algorithms differ in their speed and efficiency

Although flexible and powerful, lots can go wrong

Demonstration