Bayesian, MCMC, and Multilevel Modeling
(a foray into the subjective)

Christopher David Desjardins
http://cddesjardins.wordpress.com

27 April 2010
Outline

1. Introduction
2. Prior distributions
3. MCMC
4. MCMCglmm
Bayesian toolbox for multilevel modeling

- **MCMC software**
  - **JAGS.** JAGS is cross-platform, actively developed, and can be called directly from R via the rjags library. It can run any model as long as you can program it!
    - It is CLI only
  - **WinBUGS** and/or **OpenBUGS** for PC. OpenBUGS is supposedly cross-platform. Similar to JAGS but more tested and with a large, active support community.
    - Can use with R via R2WinBUGS, BRugs

- **R software**
  - **MCMCglmm.** This can do LMM and GLMM, meta-analyses, and more. Excellent vignette! Also Jarrod Hadfield is super nice!!!
  - **glmmAK.** Longitudinal ordinal regression.
  - MCMC diagnostics, the coda package.

- **CRAN task view:** Bayesian!
A Presbyterian minister with an itch for gambling, who now rests in Bunhill Fields.
What is Bayesian?

- A statistical paradigm that defines probability as a measure of the state of knowledge, i.e. a 'personal probability'.
- In contrast, traditional frequentist statistics defines probability as the long running frequency of an event.
- Rooted in Bayes theorem
- All parameters in Bayesian are treated as random
- Inferences based on posterior distribution
  - In contrast, frequentist uses maximum likelihood
Reasons for Bayesian inference (Berger 1985)

- You can use prior knowledge.
- Inferences are based only on the data, i.e. doesn’t violate likelihood principle.
- Reason for stopping the experiment doesn’t affect your inferences (Carlin & Louis, 2009).
- Easier to interpret.
- More coherent as all analyses are made based on the posterior.
- Any question can be answered through Bayesian analysis.
- Bayes procedures possess numerous optimality properties.
  - Can calculate actual probability of null hypothesis being true.
  - Ability to test logical hypotheses.
  - Ability to solve more complex models.
Assume we have a normal distribution, with a mean ($\mu$) and variance ($\sigma^2$) and some data ($y$).

The likelihood is defined as the probability of the data given the parameters, $Pr(y|\mu, \sigma^2)$

This is a conditional distribution, we conditioned on the model parameters which are taken as fixed and known.

Isn’t this odd? We’ve already observed the data, and we don’t know what the parameter values are. How can they be fixed and known?

Aren’t we really interested in conditioning the parameters on the data because the parameters aren’t observed?

In Bayesian, we are interested in the posterior distribution, $Pr(\mu, \sigma^2|y)$

Looks simple right?
Bayes theorem

Mathematically

$$Pr(\mu, \sigma^2 | y) = \frac{Pr(y | \mu, \sigma^2)Pr(\mu, \sigma^2)}{Pr(y)}$$

Conceptually

Posterior = \frac{\text{Likelihood} \ast \text{Prior}}{\text{Marginal}}

Fortunately for us it simplifies ... a little

$$Pr(\mu, \sigma^2 | y) \propto Pr(y | \mu, \sigma^2)Pr(\mu, \sigma^2)$$

- So ... what is the elephant in the room?
Prior distribution

\[ Pr(\mu, \sigma^2) \]

Why?
What is this?
How do we handle this?
How do we get rid of this?
Do we want to get rid of this?
Does it have any use?
Does it have any harm?
How can we marginalize this?
Is it useful?
Oh what a prior!

- Bayes thereom, and by extension Bayesian, forces us to incorporate priors
- Priors can be very useful
  - Imagine your study is only one of a 1000 studies on topic X.
  - You can formally synthesize all these results into your models.
  - Gives you a 'stronger model' because it’s based on 1000+1 studies not just 1!?!?
  - Is it cheating?
  - Philosophically, doesn't science build on science? Our very hypotheses on previous studies? Our models and our data are not isolated, objective icebergs!
Oh no ... what a prior!

- If our results are truly unique and we incorporate previous data into our priors we might not see this in our results.
- Priors are subjective!
- With a small sample size, priors have a large effect on our results.
- Choice of your priors can have a profound impact on your inferences
- MLE estimates are 'more objective'
- Aren’t economists Bayesian? Oh no ...
Oh wait ... it is a wonderful prior!

- Confession: I would describe myself as a subjective Bayesian!
  - I believe priors should be informative and guide our models.
  - Bayesian is accepted in ecology.
  - However, I’m not foolish!

- Priors need to be based on sound evidence and must be justifiable.
- When in doubt, use non-informative, uniform priors
  - Estimates are similar to MLE in most instances.
- Guess what ... you’re already using Bayesian!
  - Estimation of random effects in longitudinal analyses and HLM, BIC, IRT!
The legendary R. A. Fischer on probability

“The probability of any event is the ratio between the value at which an expectation depending on the happening of the event ought to be computed, and the value of the thing expected upon its happening.” (In Aldrich, 2008)

But then again, Fischer didn’t believe smoking caused cancer.
How can prior distributions affect our results?

Let's take a look at rjags

```r
> require(rjags)
> dat0 <- na.omit(dat0)
> N <- nrow(dat0)
> Y <- dat0$ACT_ENGL
> x <- dat0$ACT_MATH
> dump(list=c("N","Y","x"),file="lr.data")
> datalr <- read.data(file="lr.data",format="jags")
> datalr

$x
[26]  21  20  20  26  21  21  24  20  20  22  23  17  18  20  19  16  19  15  29  26  17  18  20  17  16
[51]  20  17  16  15  17  16  28  20  25  22  26  18  25  18  22  19  26

$Y

$N
[1] 67
```
Exploring priors

> print(m1 <- jags.model(file="lr8920prior1.bug",data=datalr))

Compiling model graph
  Resolving undeclared variables
  Allocating nodes
  Graph Size: 197

JAGS model:

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, 1.0E-4)
  beta ~ dnorm(0.0, 1.0E-4)
  sigma <- 1.0/sqrt(tau)
  tau ~ dgamma(1.0E-3, 1.0E-3)
}

Fully observed variables:
  N Y x
What do these priors look like?

Priors on intercept and slope
\[ \text{rnorm}(100, 0, 1/\sqrt{1e^{-04}}) \]

Priors on variance
\[ \text{rgamma}(100, \text{shape} = 0.001, \text{scale} = 0.001) \]
> prior1 <- coda.samples(m1,variable.names=c("alpha","beta","sigma","tau"),n.iter=5000,thin=10)

|**************************************************| 100%

> summary(prior1)

Iterations = 10:5000
Thinning interval = 10
Number of chains = 1
Sample size per chain = 500

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

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>SD</th>
<th>Naive SE</th>
<th>Time-series SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>18.73395</td>
<td>0.42902</td>
<td>0.0191865</td>
<td>0.0221243</td>
</tr>
<tr>
<td>beta</td>
<td>0.51495</td>
<td>0.12166</td>
<td>0.0054406</td>
<td>0.0051588</td>
</tr>
<tr>
<td>sigma</td>
<td>3.42628</td>
<td>0.63731</td>
<td>0.0285015</td>
<td>0.0247560</td>
</tr>
<tr>
<td>tau</td>
<td>0.08829</td>
<td>0.01595</td>
<td>0.0007132</td>
<td>0.0006959</td>
</tr>
</tbody>
</table>

2. Quantiles for each variable:

<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>17.82418</td>
<td>18.47420</td>
<td>18.7603</td>
<td>18.98205</td>
<td>19.5555</td>
</tr>
<tr>
<td>beta</td>
<td>0.27909</td>
<td>0.44618</td>
<td>0.5155</td>
<td>0.58573</td>
<td>0.7579</td>
</tr>
<tr>
<td>sigma</td>
<td>2.88946</td>
<td>3.19051</td>
<td>3.3846</td>
<td>3.59386</td>
<td>4.0401</td>
</tr>
<tr>
<td>tau</td>
<td>0.06127</td>
<td>0.07742</td>
<td>0.0873</td>
<td>0.09824</td>
<td>0.1198</td>
</tr>
</tbody>
</table>
Maximum likelihood estimates

> coef(lm(Y ~ 1 + x))

(Intercept)        x
  8.5493151  0.5172069

The posterior distribution estimate of the slope is nearly identical to the maximum likelihood estimate. But the intercept is a little different. What happens if we decrease the variance?
Alternate priors with small variances

```r
> print(m2 <- jags.model(file="lr8920prior2.bug",data=datalr))

Compiling model graph
  Resolving undeclared variables
  Allocating nodes
  Graph Size: 197

JAGS model:

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, 100)
  beta ~ dnorm(0.0, 100)
  sigma <- 1.0/sqrt(tau)
  tau ~ dgamma(1.0E-3, 1.0E-3)
}

Fully observed variables:
  N Y x
```
What do these alternate priors look like?

I bet this will have an enormous impact!
Posterior distribution from prior 2

```r
> prior2 <- coda.samples(m2, variable.names=c("alpha", "beta", "sigma", "tau"), n.iter=5000, thin=10)

|**************************************************| 100%

> summary(prior2)

Iterations = 10:5000
Thinning interval = 10
Number of chains = 1
Sample size per chain = 500

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

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>SD</th>
<th>Naive SE</th>
<th>Time-series SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>0.045119</td>
<td>0.341613</td>
<td>1.528e-02</td>
<td>1.453e-02</td>
</tr>
<tr>
<td>beta</td>
<td>0.013496</td>
<td>0.105259</td>
<td>4.707e-03</td>
<td>4.841e-03</td>
</tr>
<tr>
<td>sigma</td>
<td>19.366620</td>
<td>1.742587</td>
<td>7.793e-02</td>
<td>7.326e-02</td>
</tr>
<tr>
<td>tau</td>
<td>0.002731</td>
<td>0.000495</td>
<td>2.214e-05</td>
<td>2.127e-05</td>
</tr>
</tbody>
</table>

2. Quantiles for each variable:

<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>-0.165544</td>
<td>-0.034142</td>
<td>0.031131</td>
<td>0.093523</td>
<td>0.230392</td>
</tr>
<tr>
<td>beta</td>
<td>-0.193316</td>
<td>-0.055528</td>
<td>0.009584</td>
<td>0.082081</td>
<td>0.211800</td>
</tr>
<tr>
<td>sigma</td>
<td>16.482919</td>
<td>18.133185</td>
<td>19.282952</td>
<td>20.382879</td>
<td>22.999065</td>
</tr>
<tr>
<td>tau</td>
<td>0.001891</td>
<td>0.002407</td>
<td>0.002689</td>
<td>0.003041</td>
<td>0.003681</td>
</tr>
</tbody>
</table>

alpha and beta got pulled towards zero and sigma balloned!
This time with uniform priors

This time lets set the priors on alpha and beta to be uniform.

> print(m3 <- jags.model(file="lr8920prior3.bug",data=datalr))

### JAGS model:

```r
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 ~ dunif(-100, 100)
  beta ~ dunif(-100, 100)
  sigma <- 1.0/sqrt(tau)
  tau ~ dgamma(1.0E-3, 1.0E-3)
}
```

**Fully observed variables:**
- N Y x
3rd priors

What do you think the posterior distribution will look like?
Posterior distribution from prior 3

```r
> prior3 <- coda.samples(m3, variable.names=c("alpha", "beta", "sigma", "tau"), n.iter=5000, thin=10)

|**************************************************| 100%

> summary(prior3)

Iterations = 1010:6000
Thinning interval = 10
Number of chains = 1
Sample size per chain = 500

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

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>SD</th>
<th>Naive SE</th>
<th>Time-series SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>18.73255</td>
<td>0.4101</td>
<td>0.018342</td>
<td>0.0171511</td>
</tr>
<tr>
<td>beta</td>
<td>0.51656</td>
<td>0.1133</td>
<td>0.005069</td>
<td>0.0047222</td>
</tr>
<tr>
<td>sigma</td>
<td>3.37070</td>
<td>0.3047</td>
<td>0.013626</td>
<td>0.0131941</td>
</tr>
<tr>
<td>tau</td>
<td>0.09015</td>
<td>0.0161</td>
<td>0.000720</td>
<td>0.0007335</td>
</tr>
</tbody>
</table>

2. Quantiles for each variable:

<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>17.96480</td>
<td>18.43754</td>
<td>18.74745</td>
<td>19.02622</td>
<td>19.4980</td>
</tr>
<tr>
<td>beta</td>
<td>0.30025</td>
<td>0.44335</td>
<td>0.51497</td>
<td>0.59273</td>
<td>0.7369</td>
</tr>
<tr>
<td>sigma</td>
<td>2.81918</td>
<td>3.18651</td>
<td>3.33087</td>
<td>3.56174</td>
<td>4.0542</td>
</tr>
<tr>
<td>tau</td>
<td>0.06084</td>
<td>0.07883</td>
<td>0.09013</td>
<td>0.09848</td>
<td>0.1258</td>
</tr>
</tbody>
</table>

Very similar to MLE
Your turn to run priors!

- I’d like you guys to break up into groups of 3.
- Using my code and data, tweak the model, and run it!
- Run `jags.model()` then `coda.samples()`.
- Try it with a few different priors and repeat the process for the same priors.
- What were your priors? How did they affect your estimates? Do you get the same estimates when you ran the same priors again? What if you used `set.seed()`?
What are the lessons from priors?

- Priors can greatly affect your estimates, especially the choice of the variance!
- When $n$ increases, the influence of priors goes down
  - How big does $n$ need to be? Bigger than 67 apparently ...
  - For "fixed effects" estimates, $n$ doesn't need to be as big as it does for variance and "random effects".
- Should examine posterior modes, posterior means, and Bayesian confidence intervals. If these are similar to MLE then you are good to go! If not worry?
- Can also do sensitivity analysis of posterior distribution under different prior specifications.
- For GLMM and LMM, when in doubt use non-informative uniform priors on fixed effects and non-informative inverse Wishart on residuals & random effects (discussed later). Results will be similar to MLE.
How do we estimate posterior distributions?

Recall for a simple linear regression of a normally distributed variable...

**Bayes Theorem**

\[ Pr(\mu, \sigma^2 | y) \propto Pr(y | \mu, \sigma^2) Pr(\mu, \sigma^2) \]

- In order to calculate the posterior distribution often requires integrating out many variables, taking integrals of integrals.
  - Imagine the case of a GLMM, where you have to estimate your fixed effects, variances, and random effects.
  - This can not be derived analytically and requires approximation.

- MCMC provides a way to estimate the posterior distribution.
- It can "find" the posterior distribution within MCMC error.
- MCMC works by walking stochastically through space, the Monte Carlo part, and hopefully from areas of low to high probability of where our parameters are living.
Starting values

- The first thing that you need for MCMC are starting values.
- This will help to speed up convergence and reduce wasted wandering.
- All estimated parameters require starting values.
- These may be generated by the program.
- Or supplied by the user.
- Using the MLE, when possible, is a good way to start MCMC.
After we’ve initialized our chain we need to decide where to go next and start sampling from our posterior distribution!

To do this we generate a candidate destination and decide whether we should move there or stay put.

One way to do this is using a Metropolis-Hastings (MH) algorithm.

During the first iteration, a random set of coordinates are picked from a multivariate normal distribution and compared to the initial coordinates.

If the posterior probability is greater for the new coordinates, than we move to the new coordinates.
Metropolis-Hastings continued

- If the posterior probability is lower, than we may stay or we may go.
- Once this iteration is complete, we repeat this again with our coordinates at the end of the last iteration becoming our baseline coordinates.
- We repeat this over and over again depending upon how many iterations we’ve specified.
- Iterations are saved and we examine them when we look at the posterior distribution.
- Pros: Easy to write your own algorithms, less mathematical
- Cons: Very slow to converge!
MH conceptually
Gibbs sampling

- A special case of MH
- In MH, we could move in multiple directions simultaneously, i.e. we moved for both $\mu$ and $\sigma^2$.
- However, we could have moved in just one direction at a time, for example move in the direction of $\mu$ conditioned on some value of $\sigma^2$ say 1.
- Mathematically then, $Pr(\mu|\sigma^2 = 1, y)$
Gibbs sampling continued

- First, let’s move in the direction of $\mu$ conditioned on some value of $\sigma^2$
- We could generate a new candidate randomly and ask if it’s posterior probability is higher but ...
- We often know the equation for this conditional distribution, this is Gibbs Sampling
- We know we’re at $\sigma^2 = 1$, so why don’t we just calculate $\mu$ directly.
- Repeat this process for $\sigma^2$ conditioning on the new value of $\mu$ and so on.
- Pro: Converges much faster!
- Cons: Much more difficult to program .. so let’s let MCMCglmm or JAGS do the work!
MCMC diagnostics - How do we know it’s working right?

- First thing you should look at our trace plots.
  - They should be free from pattern
  - Each plot should have roughly a horizontal line with no ascending/descending pattern
- Next you should look at autocorrelation between iterations.
  - Iterations have a tendency to be autocorrelated and are not independent.
  - Should examine this empirically and thin out your chain.
- There are lots of helpful diagnostic tools in the coda package
  - raftery.diag() - Tells you how many MCMC iterations you need to run for convergence
  - geweke.diag() - Compares means from first part of Markov chain with the last part.
- Burn, burn, burn, thin, thin, thin
Researchers have reported that MCMC needs some time to 'warm' up to remove autocorrelation, i.e. to reach convergence.

This is specified as the **burn-in** time.

- You run MCMC but do not store the output until after the burn-in period.
- `gelman.diag()` in coda is one way to help you determine the length of the required burn-in until convergence.
- However, the need for burn-in is **debatable**.

Many MCMC programs allow you to specify multiple chains (up to 5).

- `MCMCglmm` uses only one.
- Crossing chains indicate convergence.
MCMCglmm (Hadfield, 2010) is one library in R that can be used to run generalized linear mixed models.

MCMCglmm can run Gaussian, Binomial, Poisson, Zero-Inflated Poisson, Ordinal, Multinomial, and other models.

I came about MCMCglmm when I had longitudinal data that was zero-inflated and could not be estimated with lme4.

Specification of models in MCMCglmm() is very similar to glmer() and lmer().

However, you need to specify priors.
**Pros/Cons MCMCglmm**

**Pros**
- Can estimate very complex models.
- Lots of diagnostic tools available via coda package.
  - Bayesian outshines traditional statistics here.
- Developer extremely helpful and active on r-sig-mixed models mailing list.
- Can calculate actual p-values, 95% Bayesian confidence intervals, Deviance information criterion.

**Cons**
- Very slow with large data sets. My data set can take 24 hours to run!
- Requires specification of priors and starting values
- Doesn’t give frequentist p-values, no t-value
- A little harder to program than lme4
Specifying a `MCMCglmm()` model

### Default arguments to `MCMCglmm()`

```r
MCMCglmm(fixed, random=NULL, rcov= units, family="gaussian", mev=NULL, data,start=NULL, prior=NULL, tune=NULL, pedigree=NULL, nodes="ALL", scale=TRUE, nitt=13000, thin=10, burnin=3000, pr=FALSE, pl=FALSE, verbose=TRUE, DIC=TRUE, singular.ok=FALSE, saveX=FALSE, saveZ=FALSE, slice=FALSE)
```

**fixed** and **random** fixed and random effects formulae; **rcov** residual Cov structure; **family** probability family; **mev** specify measurement error (meta-analysis); **data** data; **start** starting values; **prior** the priors; **tune** Cov matrix for proposed latent variable; **pedigree**, **nodes**, and **scale** useful for genetics; **nitt** # of iterations; **thin** thinning interval; **burnin** # of burn-ins; **pr** and **pl** save posterior of random effects and latent variables; **verbose** get output; **DIC** Deviance information criterion; **singular.ok** when false linear dependencies in fixed effects are removed; **saveX** and **saveZ** save fixed effects and random effects matrix; **slice** use slice sampling (type of MCMC).
Labdosage data set

We’ll use the labdosage.txt file. I believe you all are familiar with it?

> head(dos)

<table>
<thead>
<tr>
<th>id</th>
<th>group</th>
<th>time</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>1</td>
<td>2mg</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>1</td>
<td>2mg</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>1</td>
<td>2mg</td>
<td>3</td>
</tr>
<tr>
<td>1.4</td>
<td>1</td>
<td>2mg</td>
<td>4</td>
</tr>
<tr>
<td>1.5</td>
<td>1</td>
<td>2mg</td>
<td>5</td>
</tr>
<tr>
<td>1.6</td>
<td>1</td>
<td>2mg</td>
<td>6</td>
</tr>
</tbody>
</table>
Specifying priors on a random intercepts model with Poisson data

First, we need to specify the priors

- **B-structure** priors on 'fixed effects'; **G-structure** priors on 'random effects'; **R-structure** priors on residual variance.
- Default $B$ is $\mu = 0$, $V=I*1e+10$, $G$ and $R$ are $\nu=0$, $V=1$, $\alpha.\mu=1$, and $\alpha.V=0$.
- $B$ is multivariate normal; $G$ and $R$ are inverse Wishart with Cauchy parameter expansion (Gelman, 2006)
- Default $B$ is weak and essentially uninformative, $G$ and $R$ are both weakly informative.
Specifying a random intercepts model continued

> prior<-list(R=list(V=1, nu=0.002), G=list(G1=list(V=1, nu=0.002)))

These are essentially the same priors used for the the JAGS model.

> m1 <- MCMCglmm(count ~ 1 + time, random=~id, prior=prior, family="poisson", data=dos)

     MCMC iteration = 0
Acceptance ratio for latent scores = 0.000396
     MCMC iteration = 1000
Acceptance ratio for latent scores = 0.424758
     ..................
Acceptance ratio for latent scores = 0.391338
Summary statistics - Fixed effects

> # Summary of 'Fixed Effects'
> summary(m1$Sol)

Iterations = 3001:12991
Thinning interval = 10
Number of chains = 1
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
(Intercept) 1.93977 0.080995 0.0025613 0.0030749
   time -0.05646 0.006811 0.0002154 0.0002181
```

2. Quantiles for each variable:

```
     2.5%   25%   50%   75%   97.5%
(Intercept) 1.78134 1.88190 1.93970 1.99740 2.09185
   time -0.06871 -0.06132 -0.05666 -0.05219 -0.04249
```

> HPDinterval(m1$Sol)

```
  lower       upper
(Intercept) 1.78774756 2.09483939
   time -0.06889096 -0.04286956
attr("Probability")
[1] 0.95
> # Summary of 'Random Effects
> summary(m1$VCV)

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

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>SD</th>
<th>Naive SE</th>
<th>Time-series SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>0.2815</td>
<td>0.06264</td>
<td>0.0019809</td>
<td>0.001970</td>
</tr>
<tr>
<td>units</td>
<td>0.2424</td>
<td>0.02490</td>
<td>0.0007873</td>
<td>0.001281</td>
</tr>
</tbody>
</table>

2. Quantiles for each variable:

<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>0.1865</td>
<td>0.2372</td>
<td>0.2727</td>
<td>0.3174</td>
<td>0.4166</td>
</tr>
<tr>
<td>units</td>
<td>0.1997</td>
<td>0.2248</td>
<td>0.2410</td>
<td>0.2577</td>
<td>0.2959</td>
</tr>
</tbody>
</table>

> HPDinterval(m1$VCV)

<table>
<thead>
<tr>
<th></th>
<th>lower</th>
<th>upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>0.1728576</td>
<td>0.3990647</td>
</tr>
<tr>
<td>units</td>
<td>0.1967859</td>
<td>0.2921864</td>
</tr>
</tbody>
</table>

attr(,"Probability")
[1] 0.95
Trace plots of 'fixed' and 'random' effects

1. Trace of (Intercept)
   - Iterations: 4000, 8000, 12000
   - N = 1000
   - Bandwidth = 0.02157

2. Density of (Intercept)
   - Iterations: 1.7, 1.9, 2.1
   - N = 1000
   - Bandwidth = 0.001813

3. Trace of id
   - Iterations: 0, 1, 2, 3, 4
   - N = 1000
   - Bandwidth = 0.01592

4. Density of id
   - Iterations: 0.1, 0.3, 0.5, 0.7
   - N = 1000
   - Bandwidth = 0.006523

5. Trace of time
   - Iterations: 4000, 8000, 12000
   - N = 1000
   - Bandwidth = 0.02157

6. Density of time
   - Iterations: −0.08, −0.06, −0.04
   - N = 1000
   - Bandwidth = 0.001813

7. Trace of units
   - Iterations: 0, 5, 10, 15
   - N = 1000
   - Bandwidth = 0.006523

8. Density of units
   - Iterations: 0.15, 0.20, 0.25, 0.30, 0.35
   - N = 1000
   - Bandwidth = 0.006523
Autocorrelations of iterations

> autocorr(m1$Sol)

, , (Intercept)

(Intercept) time
Lag 0 1.00000000 -0.49419125
Lag 10 0.08802531 -0.07252791
Lag 50 0.06445487 -0.04806522
Lag 100 -0.02629911 0.06327561
Lag 500 0.03672346 -0.04482030

, , time

(Intercept) time
Lag 0 -0.49419125 1.00000000
Lag 10 0.02048099 0.04815938
Lag 50 -0.04384447 0.03041308
Lag 100 -0.01538620 -0.03313733
Lag 500 -0.00999942 0.02657281
How does this compare with `glmer()`

```r
> m1.glmer <- glmer(count ~ 1 + time + (1 | id), family="poisson", data=dos)
```

<table>
<thead>
<tr>
<th>Effect</th>
<th>MCMCglmm</th>
<th>lme4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1.9398</td>
<td>2.0389</td>
</tr>
<tr>
<td>Time</td>
<td>-0.0565</td>
<td>-0.0546</td>
</tr>
<tr>
<td>id</td>
<td>0.2815</td>
<td>0.2652</td>
</tr>
</tbody>
</table>

Very similar! This is great! Additionally, the standard errors were similar too with `MCMCglmm` being slightly smaller.
Predicted growth curve from MCMCglmm syntax

```r
> beta <- colMeans(m1$Sol)
> plot(dos$count ~ as.factor(dos$time), xlab = "Time",
     ylab = "Predicted # of Seizures")
> points(tapply(dos$count, as.factor(dos$time), mean) ~ I(1:12), pch = 16)
> lines(exp(pred + 0.5 * mean(rowSums(m1$VCV))) ~ I(1:12))
```
Predicted growth curve
Random slopes MCMCglmm model

> m2 <- MCMCglmm(count ~ 1 + time, random=~id + time, prior=prior,
     family="poisson",data=dos, verbose=FALSE)

Which model is better: Random intercepts or random slope?

> m1$DIC;m2$DIC

[1] 3871.923

[1] 3861.976
> summary(m2$VCV)

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

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>SD</th>
<th>Naive SE</th>
<th>Time-series SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>0.28073</td>
<td>0.05920</td>
<td>0.0018722</td>
<td>0.0022649</td>
</tr>
<tr>
<td>time</td>
<td>0.02867</td>
<td>0.01890</td>
<td>0.0005977</td>
<td>0.0007020</td>
</tr>
<tr>
<td>units</td>
<td>0.22134</td>
<td>0.02244</td>
<td>0.0007097</td>
<td>0.0009372</td>
</tr>
</tbody>
</table>

2. Quantiles for each variable:

<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>0.181775</td>
<td>0.24027</td>
<td>0.27404</td>
<td>0.31639</td>
<td>0.41390</td>
</tr>
<tr>
<td>time</td>
<td>0.007685</td>
<td>0.01663</td>
<td>0.02452</td>
<td>0.03464</td>
<td>0.07818</td>
</tr>
<tr>
<td>units</td>
<td>0.180095</td>
<td>0.20586</td>
<td>0.22075</td>
<td>0.23545</td>
<td>0.26900</td>
</tr>
</tbody>
</table>
Trace plots of random slopes model

- **Trace of id**: 
  - Iterations range from 4000 to 12000.
  - The trace plot shows the variation of the id parameter over iterations.
  - Density plot with N = 1000 and Bandwidth = 0.01513.

- **Trace of time**: 
  - Iterations range from 4000 to 12000.
  - The trace plot shows the variation of the time parameter over iterations.
  - Density plot with N = 1000 and Bandwidth = 0.00358.

- **Trace of units**: 
  - Iterations range from 4000 to 12000.
  - The trace plot shows the variation of the units parameter over iterations.
  - Density plot with N = 1000 and Bandwidth = 0.005879.
Random slopes model with a static predictor

```r
> m3 <- MCMCglmm(count ~ 1 + time + group, random=~id + time,
>                prior=prior, family="poisson", data=dos,
>                verbose=FALSE)

> m2$DIC; m3$DIC

[1] 3861.976
[1] 3860.961
```
### Summary statistics: Fixed effects

> `summary(m3$Sol)`

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

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>SD</th>
<th>Naive SE</th>
<th>Time-series SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>2.05451</td>
<td>0.14353</td>
<td>0.0045389</td>
<td>0.0050984</td>
</tr>
<tr>
<td>time</td>
<td>-0.05584</td>
<td>0.01520</td>
<td>0.0004808</td>
<td>0.0005261</td>
</tr>
<tr>
<td>group2mg</td>
<td>-0.27097</td>
<td>0.14036</td>
<td>0.0044387</td>
<td>0.0043285</td>
</tr>
</tbody>
</table>

2. Quantiles for each variable:

<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>1.75796</td>
<td>1.9676</td>
<td>2.05187</td>
<td>2.14983</td>
<td>2.33057</td>
</tr>
<tr>
<td>time</td>
<td>-0.08692</td>
<td>-0.0651</td>
<td>-0.05606</td>
<td>-0.04607</td>
<td>-0.02599</td>
</tr>
<tr>
<td>group2mg</td>
<td>-0.53615</td>
<td>-0.3666</td>
<td>-0.27407</td>
<td>-0.17733</td>
<td>0.00905</td>
</tr>
</tbody>
</table>
Predicted growth curves syntax

```r
> beta <- colMeans(m3$Sol)
> plot(dos$count ~ as.factor(dos$time), xlab = "Time",
+       ylab = "Predicted # of Seizures")
> points(tapply(dos$count, as.factor(dos$time),
+               mean) ~ I(1:12), pch = 16)
> lines(exp(pred.t + 0.5 * mean(rowSums(m3$VCV))) ~ I(1:12),
+       col = "blue")
> lines(exp(pred.b + 0.5 * mean(rowSums(m3$VCV))) ~ I(1:12),
+       col = "red")
```
Growth curve
Let's set some pretty illogical priors

```r
> priors2 <- list(B=list(mu=10,V=1),
                 R=list(V=1, nu=0.002),
                 G=list(G1=list(V=0, nu=10),
                       G2=list(V=1,nu=0.002)))
> m4 <- MCMCglmm(count ~ 1 + time + group, random=~id + time,
                prior=prior2, family="poisson",data=dos,
                verbose=FALSE)
```
And the results …

<table>
<thead>
<tr>
<th>Effects</th>
<th>Uninformative priors</th>
<th>Crazy priors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>2.055</td>
<td>2.061</td>
</tr>
<tr>
<td>Time</td>
<td>-0.056</td>
<td>-0.056</td>
</tr>
<tr>
<td>Group</td>
<td>-0.271</td>
<td>-0.270</td>
</tr>
<tr>
<td>Random Intercept</td>
<td>0.264</td>
<td>0.266</td>
</tr>
<tr>
<td>Random Slope</td>
<td>0.029</td>
<td>0.0274</td>
</tr>
</tbody>
</table>

With 780 data points, the likelihood already swamps the prior!
Running `MCMCglmm()` in your group

- Spend some time now in a group running different priors and different models on the data set.
- Maybe try specifying starting values?
  - `start <- list(B=2.05,-0.06,-0.27)`
  - `m3a <- MCMCglmm(count ~ 1 + time + group, random=~id + time,family="poisson",data=dos, prior=prior,start=start,verbose=FALSE)`
  - I don't recommend starting values for $G$ and $R$
- If you need help with syntax please ask.
- Did you find anything interesting?
To Bayesian or not to Bayesian ...

- In my opinion, it’s useful to know Bayesian methods as ML approaches & least squares approaches don’t always work.
- I believe that Bayesian methods will become more and more important as computers become more and more powerful.
- Bayesian frees you from the awkwardness of frequentist statistics and creates the opportunity to include prior information.
- But ... like everything there are costs to Bayesian.
- Hopefully I’ve give you just enough information to be dangerous
Thanks!